Introdução aos Gráficos usando Python

Prof. Donald Mark Santee

Indice

Primeiros Passos

Para a geração de gráficos é necessário importar o módulo `matplotlib.pyplot`. Esse módulo contém uma coleção de comandos para criar, alterar e configurar um gráfico dentro de uma, ou mais, novas janelas do sistema operacional.

Para testar a instalação do `matplotlib` digite, dentro do terminal do *Python*:

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

Se não der uma mensagem de erro, a biblioteca `matplotlib` está instalada e funcionando. Assim pode-se prosseguir para as demais linhas:

```
>>> plt.plot((1,2,3,4))
[<matplotlib.lines.Line2D object at 0x8fd48ac>]
>>> plt.ylabel(u'Alguns Números')
>>> plt.show()
```

Ao longo deste texto a biblioteca será importada e renomeada para `plt`. Assim, todos os comandos importados dessa biblioteca terão o prefixo `plt`.

Após o comando `plt.plot(...)`, o gráfico não aparece, entretanto ele é gerado na memória do computador. A mensagem `[<matplotlib.lines.Line2D object at 0x8fd48ac>]` indica que o gráfico foi criado.

O comando `plt.ylabel(...)` coloca o nome no eixo *y*. A letra `u` antes da *string* indica que a *string* está em *unicode*. Essa indicação é necessária para que as letras acentuadas sejam escritas corretamente.

O gráfico só será mostrado na tela quando o comando `plt.show()` for dado. Normalmente esse comando é colocado como o último comando do gráfico. Mesmo depois que o gráfico é mostrado na tela ele pode ser alterado, isto é, podem existir comandos que alteram o gráfico depois que o gráfico é mostrado.

Observa-se, no gráfico, que o eixo *x* varia de zero a três, enquanto que o eixo *y* varia de um a quatro. Isso ocorre porque o eixo *x* refere-se aos índices dos dados, que começam por zero. O eixo *y* mostra os valores dados. Nesse exemplo os pontos estão ligados por uma reta.

Quando se fornecem duas tuplas, ou listas, ao comando `plt.plot()`, a primeira refere-se às coordenadas *x* e a segunda às coordenadas *y*. Por exemplo:

```
>>> plt.plot((1,2,3,4), (1,4,9,16))
```

Além do par de tuplas (ou listas) indicando as coordenadas dos pontos *x* e *y* pode-se colocar um terceiro argumento chamado de "*string* de formatação". Como o nome indica, é uma *string* que define a cor e o tipo da marcação dos pontos. A *string* de formatação padrão é `'b-'` que indica que a cor é *blue* (azul em inglês) e os pontos são unidos por uma linha. O exemplo a seguir marca os pontos com círculos vermelhos (*red*).

```
>>> import matplotlib.pyplot as plt
>>> plt.plot((1,2,3,4), (1,4,9,16), 'ro')
[<matplotlib.lines.Line2D object at 0xa302c4c>]
>>> plt.axis((0, 6, 0, 20))
(0, 6, 0, 20)
>>> plt.show()
```

Se a biblioteca `matplotlib.pyplot` já tiver sido importada, a primeira linha é desnecessária (mas inofensiva).

O comando `plt.plot(...)` é o que efetivamente faz o desenho. Com a *string* de formatação `'ro'` os pontos serão marcados com círculos (indicados pela letra `o`) vermelhos (indicados pela letra `r`).

As tabelas abaixo apresentam a relação dos códigos reconhecidos para a *string* de formatação na especificação da cor, tipo da linha que une os pontos, e forma de marcar os pontos.

Cor	**Caractere**
Azul	‘b’
Verde	‘g’
Vermelho	‘r’
Ciano	‘c’
Magenta	‘m’
Amarelo	‘y’
Preto	‘k’
Branco	‘w’

Tipo da linha	**Caractere**
Linha cheia	'-'
Linha tracejada	'--'
Linha traço-ponto	'-.'
Linha pontilhada	':'

Tipo do Marcador	**Caractere**
Ponto	'.'
Pixel	','
Círculo	'o'
Triângulo	'v'
Triângulo	'^'
Triângulo	'<'
Triângulo	'>'
Y para baixo	'1'
Y para cima	'2'

Tipo do Marcador	Caractere
Y para a esquerda	`'3'`
Y para a direita	`'4'`
Quadrado	`'s'`
Pentágono	`'p'`
Estrela	`'*'`
Hexágono	`'h'`
Hexágono	`'H'`
Sinal de mais	`'+'`
Sinal cruzado	`'x'`
Losango largo	`'D'`
Losango estreito	`'d'`
Linha vertical	`'|'`
Linha horizontal	`'_'`

O comando `plt.axis(...)` especifica os limites superior e inferior de cada eixo. Os primeiros dois números referem-se aos limites do eixo *x* e os outros dois aos limites do eixo *y*.

Para desenhar mais de uma linha em um mesmo gráfico usa-se mais de um comando `plt.plot(...)`. Cada um contendo a lista de coordenadas *x*, coordenadas *y*, e sua *string* de formatação. O exemplo a seguir ilustra um gráfico com três curvas.

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.linspace(0,1,num=100)
>>> y = x
>>> plt.plot(x,y,'r--')
[<matplotlib.lines.Line2D object at 0xa6f1c8c>]
>>> z = [t**2 for t in x]
>>> plt.plot(x,z,'bo')
[<matplotlib.lines.Line2D object at 0xa6fac4c>]
>>> w = [t**3 for t in x]
>>> plt.plot(x,w,'g^')
[<matplotlib.lines.Line2D object at 0x9eac40c>]
>>> plt.show()
```

Alternativamente pode-se usar apenas um comando `plt.plot(...)` onde se colocam os trios :coordenadas *x*, coordenadas *y* e *string* de formatação, para cada gráfico consecutivamente. O exemplo a seguir ilustra o mesmo gráfico anterior mas com um único comando `plt.plot(...)`.

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.linspace(0,1,num=100)
```

```
>>> y = x
>>> z = [t**2 for t in x]
>>> w = [t**3 for t in x]
>>> plt.plot(x,y,'r--',x,z,'bs',x,w,'g^')
[<matplotlib.lines.Line2D object at 0x90f8c8c>,
<matplotlib.lines.Line2D object at 0x9107fec>,
<matplotlib.lines.Line2D object at 0x910e2cc>]
>>> plt.show()
```

Configurando os Eixos e a Grade

Quando um gráfico é desenhado, o comando `plt.plot(...)` automaticamente determina que a escala de cada coordenada varie do menor valor até o maior valor. Entretanto, muitas vezes deseja-se alterar esses valores, por exemplo deseja-se que a escala de *x* varie de zero até o valor máximo, e não do valor mínimo ao máximo. Para especificar novos valores usa-se o comando `plt.axis(...)`. Esse comando possui um argumento, que é uma tupla. Essa tupla deve conter quatro valores referentes ao *x* mínimo, *x* máximo, *y* mínimo e *y* máximo que se deseja usar.

Por exemplo, para especificar que o eixo *x* varia de 0 a 1 enquanto o eixo *y* varia de 0 a 10, a instrução é:

```
plt.axis((0,1,0,10))
```

Os parêntesis duplos são necessários porque o comando `plt.axis(...)` exige uma tupla (ou lista) como argumento.

Alternativamente, para configurar os limites dos eixos *x* e *y* separadamente usam-se os comandos `plt.xlim(...)` e `plt.ylim(...)` respectivamente, com os limites superior e inferior como argumentos. Assim o exemplo anterior é equivalente a:

```
plt.xlim(0,1)
plt.ylim(0,1)
```

Para colocar o desenho de uma grade ao fundo de um gráfico, tem-se a comando

```
plt.grid(True)
```

Esse comando requer um único argumento. Se for `True` a grade é desenhada, se for `False` a grade não é desenhada.

Configurando as propriedades das curvas

Alem das propriedades como estilo do traço e cor da linha, existem outras propriedades que podem ser configuradas com parâmetros opcionais. A tabela mostra algumas das propriedades importantes que podem ser configuradas.

Propriedade	Nome	Valores
Cor da linha	`color` ou `c`	Qualquer cor da *string* de formatação. Alternativamente pode-se expressar a cor como RGB na forma '#FFDD33'
Estilo da linha	`linestyle` ou `ls`	Qualquer valor de estilo de linha da string de formatação. Por exemplo: '-.'para linhas traço-ponto.
Largura da linha	`linewidth` ou `lw`	Valor inteiro indicando o número de pontos de largura.
Estilo do desenho da linha	`drawstyle`	Os valores possíveis são: `'default'` para pontos unidos por uma reta. `'steps-pre'` para linha horizontal iniciando no ponto inicial. `'steps-mid'` linha horizontal com um salto entre os pontos, e `'steps-post'` para linha horizontal partindo do ponto final.
Formato da marcação de um ponto.	`marker`	Qualquer valor da *string* de formatação.
Cor da linha de contorno da marcação do ponto.	`markeredgecolor` ou `mec`	Qualquer cor da *string* de formatação. Alternativamente pode-se expressar a cor como RGB na forma `'#FFDD33'`
Largura da linha de contorno da marcação do ponto.	`markeredgewidth` ou `mew`	O número de pixels de largura.
Cor da marcação	`markerfacecolor`	Qualquer cor da *string* de formatação. Alternativamente pode-se expressar a cor como RGB na forma `'#FFDD33'`
Tamanho da marcação dos pontos.	`markersize` ou `ms`	Tamanho da marcação em *pixels*.
Preenchimento da marcação de pontos.	`fillstyle`	Os valores possíveis são `'full'` para um marcador totalmente preenchido, ou `'left'`, `'right'`, `'bottom'` e `'top'` para marcadores preenchidos pela metade.

O exemplo a seguir ilustra a criação de uma curva em que várias propriedades são configuradas.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot((3,4,5), c='#FFCC00', lw=3, marker='o', ms=12,\
mfc='r', mec='b', mew=3, drawstyle='steps-mid')
>>> plt.axis((-0.1, 2.1, 2.9, 5.1))
>>> plt.show()
```

Trabalhando com vários gráficos e janelas

O módulo `matplotlib.pyplot` usa o conceito de “janela corrente” e “gráfico corrente”. Todos os comandos de desenho são aplicados à janela e ao gráfico corrente.

A função `plt.figure(...)` é o comando que cria, e seleciona, uma janela no terminal onde o gráfico será mostrado. Nesse exemplo, e nos anteriores, esse comando é opcional pois se omitido será considerado como `plt.figure(1)` quando o gráfico for criado pela primeira vez (pelo comando `plt.plot(...)`).. Dentro de uma mesma janela, pode haver mais de um gráfico. Para criar os diferentes gráficos dentro da janela usa-se o comando `plt.subplot(...)`. O argumento do comando `plt.subplot(...)` é formado por três valores que representam: Número de Linhas, Número de Colunas, e Número do Gráfico.

Segue um exemplo que, em uma única janela, cria dois gráficos

```
  # programa graficos.py
1.      # coding: utf-8
2.
3.      import matplotlib.pyplot as plt
4.      import numpy as np
5.
6.      f = lambda t: np.exp(-t) * np.cos(2*np.pi*t)
7.
8.      t1 = np.linspace(5,10,num=100)
9.      yt1 = f(t1)
10.     t2 = np.linspace(5,25,num=100)
11.     yt2 = f(t2)
12.
13.     plt.figure(1)       # Cria a janela
14.     plt.subplot(2,1,1) # Cria a área do primeiro gráfico
15.     plt.plot(t1, yt1, 'bo', t2, yt2, 'k') # Desenha o
  gráfico
16.
17.     plt.subplot(2,1,2)   # Cria a área do segundo gráfico
18.     ycos = np.cos(2*np.pi*t2)
```

```
19.    plt.plot(t2, ycos, 'r--')  # Desenha o segundo gráfico
20.    plt.show()
```

O exemplo abaixo cria duas janelas. Uma das janelas possui dois gráficos e a outra apenas um. O exemplo mostra como se pode navegar entre janelas e figuras usando `plt.figure(...)` e `plt.subplot(...)`.

```
1.     # program graficos2.py
2.     # coding: utf-8
3.     import matplotlib.pyplot as plt
4.     plt.figure(1)          # a primeira janela
5.     plt.subplot(2,1,1)     # o primeiro gráfico na primeira
  janela
6.     plt.plot((1,2,3))
7.     plt.subplot(2,1,2)     # o segundo gráfico na primeira
  janela
8.     plt.plot((4,5,6))
9.
10.
11.    plt.figure(2)          # uma segunda janela
12.    plt.plot((4,5,6))      # cria o gráfico em subplot(1,1,1
  ) por padrão
13.
14.    plt.figure(1)          # torna a janela 1 a janela
  corrente; subplot(2,1,2) ainda é o gráfico corrente
15.    plt.subplot(2,1,1)     # faz subplot(2,1,1) na janela 1
  o gráfico corrente
16.    plt.title(u'fácil como 1,2,3') # Título do gráfico 2,1,1
17.    plt.show()
```

Trabalhando com textos

Para adicionar um texto em uma posição qualquer do gráfico corrente usa-se o comando `plt.text(...)`. Esse comando requer três argumentos: coordenada *x*, coordenada *y*, e a *string* a ser mostrada.

Para adicionar o nome do eixo *x* e *y* usam-se os comandos `plt.xlabel(...)` e `plt.ylabel(...)` respectivamente, que requerem um argumento que é a *string* a ser mostrada.

Para adicionar um título, usa-se o comando `plt.title(...)`.

Assim como as curvas podem-se configurar as propriedades de textos usando argumentos opcionais nos comandos acima. As tabelas a seguir presentam as principais propriedades de texto que podem ser configuradas:

Propriedade	Nome	Valores
Cor da letra	`color`	Qualquer cor da string de formatação. Alternativamente pode-se expressar a cor como RGB na forma '#FFDD33'
Cor do fundo	`backgroundcolor`	Qualquer cor da string de formatação. Alternativamente pode-se expressar a cor como RGB na forma '#FFDD33'
Tamanho da letra	`size`	O tamanho da letra em pixels, ou qualquer opção abaixo: `'xx-small'` (super extra pequena) `'x-small'` (extra pequena) `'small'` (pequena) `'medium'` (média) `'large'` (grande) `'x-large'` (extra grande) `'xx-large'` (super extra grande)
Fonte	`family`	Nome da fonte ou qualquer dos valores abaixo: `'serif'` (fonte com abas) `'sans-serif'` (fonte sem abas) `'cursive'` (letras arredondadas) `'fantasy'` `'monospace'` (mono espaçadas)
Estilo da fonte (itálico ou não)	`style`	`'normal'` (normal) `'italic'` (itálico) `'oblique'` (ligeiramente inclinada)
Força (negrito ou não)	`weight`	Um número de 0 a 1000 indicando o grau de força, ou qualquer uma das opções abaixo: `'ultralight'` (muito fraco) `'light'` (fraco) `'normal'` (normal) `'regular'` (negrito) `'book'` (negrito) `'medium'` (médio) `'roman'` (médio) `'semibold'` (médio) `'demibold'` (médio)

Propriedade	Nome	Valores
		`'demi'` (forte) `'bold'` (forte) `'heavy'` (forte)
Alinhamento horizontal	`horizontalignment`	`'center'` (centralizado) `'right'` (à direita) `'left'` (à esquerda)
Alinhamento vertical	`verticalalignment` ou `va`	`'center'` (centralizado) `'top'` (em cima) `'bottom'` (em baixo) `'baseline'` (em baixo)
Espaçamento entre linhas	`linespacing`	Um número real onde 1.0 representa a largura da linha. O valor padrão é 1.2.

O exemplo a seguir ilustra a criação do rótulo do eixo x no tamanho grande e de cor vermelha.

```
plt.xlabel('eixo x', fontsize='large', color='r')
```

Usando símbolos matemáticos em um texto.

Para a escrita de fórmulas, aceitam-se as marcações de fórmulas do TeX. Por exemplo para escrever a expressão $\sigma_i=23$ em um título pode-se escrever a expressão em TeX envolto em cifrões (como no TeX). Alem disso é necessário usar o modificador r antes da string para que a barra invertida não seja confundida com um caractere de escape:

```
plt.title(r'$\sigma_i=23$')
```

Não é necessário ter o TeX instalado no computador para que as expressões funcionem. Entretanto se usar marcações específicas do LaTeX é necessário que o mesmo esteja instalado.

O exemplo abaixo ilustra o uso das funções de posicionamento de texto, incluindo o uso de símbolos matemáticos.

```
1.      # program texto.py
2.      import matplotlib.pyplot as plt
3.      from random import random
4.      from math import sqrt
5.
6.      mu, sigma = 100, 15
7.
8.      x = list()
```

```
9.      for i in range(10000):
10.         u = random()-0.5
11.         if u >=0 :
12.             x.append(mu + sigma*sqrt(u) )
13.         else:
14.             x.append(mu - sigma*sqrt(-u))
15.
16.     # Desenha o histograma dos dados
17.     plt.hist(x, 50, normed=1, facecolor='g')
18.
19.     plt.xlabel('Resultado')
20.     plt.ylabel('Probabilidade')
21.     plt.title('Histograma do Teste')
22.     plt.text(95, .08, r'$\mu=100,\ \sigma=15$')
23.     plt.axis([85, 115, 0, 0.1])
24.     plt.grid(True)
25.     plt.show()
```

Destacando pontos

Para destacar elementos importantes em um gráfico usa-se o comando `plt.annotate(...)`. Esse comando marca um ponto no gráfico, e coloca um texto (que pode ser diferente do ponto marcado). Opcionalmente desenha uma seta partindo do texto até o ponto marcado. Os argumentos da função são: a *string* do texto a ser mostrado, uma tupla `xy` indicando o ponto marcado, uma tupla `xytext` com as coordenadas da posição do texto.

Para que a seta apareça é necessário criar um dicionário para o argumento `arrowprops` que contenha a chave `facecolor`, que dá a cor da seta. Esteticamente fica melhor se a seta ficar ligeiramente mais curta. Consegue-se esse efeito colocando a chave `shrink` no dicionário com valor de 0.05.

O exemplo abaixo ilustra o uso de `plt.annotate(...)`

```
1.      # programa annotate.py
2.      # coding: utf-8
3.      import matplotlib.pyplot as plt
4.      import numpy as np
5.
6.      plt.subplot(111)
7.      t = np.linspace(0,5,num=500 )
8.      s = np.cos(2*np.pi*t)
9.      plt.plot(t, s, lw=2)
10.
11.     plt.annotate(u'máximo local', xy=(2, 1), xytext=(3, 1.5
  ), arrowprops=dict(facecolor='black',shrink=0.05))
12.
13.     plt.ylim(-2,2)
14.     plt.show()
```