# 介绍

## Pylot教程

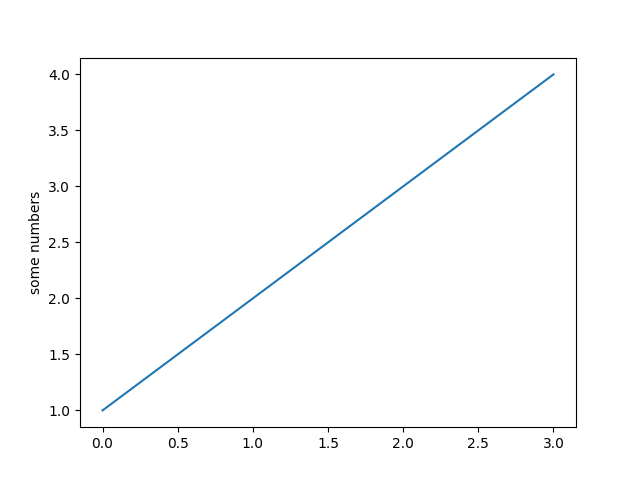
Matplotlib.pylot（http://matplotlib.org/api/pyplot\_api.html#module-matplotlib.pyplot）是一个command样式函数的集合，它提供一个类似于MATLAB的绘图框架。每个pyplot函数对一个图形进行一些更改，例如，创建一个图形，在一个图形中创建一个绘图区域，在一个绘图区域中绘制一些线条，用标签装饰图，等等。 通过matplotlib.pyplot多种状态可以通过函数调用来保持，所以它跟踪当前图和绘图区域,和绘图函数是针对当前轴(请注意“轴”,在大多数地方文档是指轴图的一部分,而不是严格的数学术语多个轴)。

import matplotlib.pyplot as plt

plt.plot([1,2,3,4])

plt.ylabel(*'some numbers'*)

plt.show()



如果您为plot()命令提供一个单列表或数组，那么matplotlib假设它是一个y值序列，并自动为您生成x值。由于python的范围从0开始，默认的x向量与y的长度相同，但从0开始。因此，x数据是[0,1,2,3]。

plot()是一个通用的命令，它将会使用任意数量的参数。例如，绘制x和y的图，可以发出命令:

plt.plot([1,2,3,4], [1,4,9,16])

对于每一个x，y参数对，有一个可选的第三个参数，它是表示图颜色和线类型的格式字符串。格式字符串的字母和符号来源于MATLAB，它将一个颜色字符串与一个行样式的字符串连接起来。默认的格式字符串是“b-”，这是一条蓝色的实线。例如，用红色圆圈来绘制上面的图，你就会使用字符串’ro’:

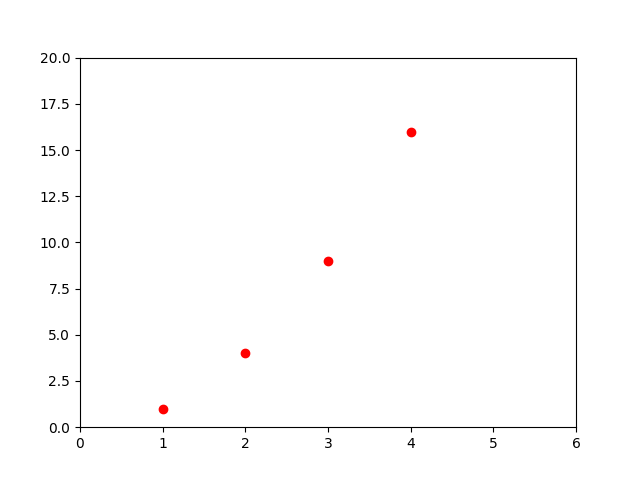
import numpy as np

import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [1,4,9,16], *'ro'*)

plt.axis([0, 6, 0, 20])

plt.show()



请参阅plot()文档以获得完整的行样式和格式字符串列表。在上面的示例中，axis()命令获取了[xmin、xmax、ymin、ymax]的列表，并指定了轴的视窗。

Help on function plot in module matplotlib.pyplot:

plot(\*args, \*\*kwargs)

Plot lines and/or markers to the

:class:`~matplotlib.axes.Axes`. \*args\* is a variable length

argument, allowing for multiple \*x\*, \*y\* pairs with an

optional format string. For example, each of the following is

legal::

plot(x, y) # plot x and y using default line style and color

plot(x, y, 'bo') # plot x and y using blue circle markers

plot(y) # plot y using x as index array 0..N-1

plot(y, 'r+') # ditto, but with red plusses

If \*x\* and/or \*y\* is 2-dimensional, then the corresponding columns

will be plotted.

If used with labeled data, make sure that the color spec is not

included as an element in data, as otherwise the last case

``plot("v","r", data={"v":..., "r":...)``

can be interpreted as the first case which would do ``plot(v, r)``

using the default line style and color.

If not used with labeled data (i.e., without a data argument),

an arbitrary number of \*x\*, \*y\*, \*fmt\* groups can be specified, as in::

a.plot(x1, y1, 'g^', x2, y2, 'g-')

Return value is a list of lines that were added.

By default, each line is assigned a different style specified by a

'style cycle'. To change this behavior, you can edit the

axes.prop\_cycle rcParam.

The following format string characters are accepted to control

the line style or marker:

================ ===============================

character description

================ ===============================

``'-'`` solid line style

``'--'`` dashed line style

``'-.'`` dash-dot line style

``':'`` dotted line style

``'.'`` point marker

``','`` pixel marker

``'o'`` circle marker

``'v'`` triangle\_down marker

``'^'`` triangle\_up marker

``'<'`` triangle\_left marker

``'>'`` triangle\_right marker

``'1'`` tri\_down marker

``'2'`` tri\_up marker

``'3'`` tri\_left marker

``'4'`` tri\_right marker

``'s'`` square marker

``'p'`` pentagon marker

``'\*'`` star marker

``'h'`` hexagon1 marker

``'H'`` hexagon2 marker

``'+'`` plus marker

``'x'`` x marker

``'D'`` diamond marker

``'d'`` thin\_diamond marker

``'|'`` vline marker

``'\_'`` hline marker

================ ===============================

The following color abbreviations are supported:

========== ========

character color

========== ========

'b' blue

'g' green

'r' red

'c' cyan

'm' magenta

'y' yellow

'k' black

'w' white

========== ========

In addition, you can specify colors in many weird and

wonderful ways, including full names (``'green'``), hex

strings (``'#008000'``), RGB or RGBA tuples (``(0,1,0,1)``) or

grayscale intensities as a string (``'0.8'``). Of these, the

string specifications can be used in place of a ``fmt`` group,

but the tuple forms can be used only as ``kwargs``.

Line styles and colors are combined in a single format string, as in

``'bo'`` for blue circles.

The \*kwargs\* can be used to set line properties (any property that has

a ``set\_\*`` method). You can use this to set a line label (for auto

legends), linewidth, anitialising, marker face color, etc. Here is an

example::

plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)

plot([1,2,3], [1,4,9], 'rs', label='line 2')

axis([0, 4, 0, 10])

legend()

If you make multiple lines with one plot command, the kwargs

apply to all those lines, e.g.::

plot(x1, y1, x2, y2, antialiased=False)

Neither line will be antialiased.

You do not need to use format strings, which are just

abbreviations. All of the line properties can be controlled

by keyword arguments. For example, you can set the color,

marker, linestyle, and markercolor with::

plot(x, y, color='green', linestyle='dashed', marker='o',

markerfacecolor='blue', markersize=12).

See :class:`~matplotlib.lines.Line2D` for details.

The kwargs are :class:`~matplotlib.lines.Line2D` properties:

agg\_filter: unknown

alpha: float (0.0 transparent through 1.0 opaque)

animated: [True | False]

antialiased or aa: [True | False]

axes: an :class:`~matplotlib.axes.Axes` instance

clip\_box: a :class:`matplotlib.transforms.Bbox` instance

clip\_on: [True | False]

clip\_path: [ (:class:`~matplotlib.path.Path`, :class:`~matplotlib.transforms.Transform`) | :class:`~matplotlib.patches.Patch` | None ]

color or c: any matplotlib color

contains: a callable function

dash\_capstyle: ['butt' | 'round' | 'projecting']

dash\_joinstyle: ['miter' | 'round' | 'bevel']

dashes: sequence of on/off ink in points

drawstyle: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']

figure: a :class:`matplotlib.figure.Figure` instance

fillstyle: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']

gid: an id string

label: string or anything printable with '%s' conversion.

linestyle or ls: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | ``'-'`` | ``'--'`` | ``'-.'`` | ``':'`` | ``'None'`` | ``' '`` | ``''``]

linewidth or lw: float value in points

marker: :mod:`A valid marker style <matplotlib.markers>`

markeredgecolor or mec: any matplotlib color

markeredgewidth or mew: float value in points

markerfacecolor or mfc: any matplotlib color

markerfacecoloralt or mfcalt: any matplotlib color

markersize or ms: float

markevery: [None | int | length-2 tuple of int | slice | list/array of int | float | length-2 tuple of float]

path\_effects: unknown

picker: float distance in points or callable pick function ``fn(artist, event)``

pickradius: float distance in points

rasterized: [True | False | None]

sketch\_params: unknown

snap: unknown

solid\_capstyle: ['butt' | 'round' | 'projecting']

solid\_joinstyle: ['miter' | 'round' | 'bevel']

transform: a :class:`matplotlib.transforms.Transform` instance

url: a url string

visible: [True | False]

xdata: 1D array

ydata: 1D array

zorder: any number

kwargs \*scalex\* and \*scaley\*, if defined, are passed on to

:meth:`~matplotlib.axes.Axes.autoscale\_view` to determine

whether the \*x\* and \*y\* axes are autoscaled; the default is

\*True\*.

.. note::

In addition to the above described arguments, this function can take a

\*\*data\*\* keyword argument. If such a \*\*data\*\* argument is given, the

following arguments are replaced by \*\*data[<arg>]\*\*:

\* All arguments with the following names: 'x', 'y'.

None

如果matplotlib只能用于处理列表，那么它对于数字处理将是相当无用的。通常，您将使用numpy数组。实际上，所有的序列都在内部转换为numpy数组。下面的示例演示了使用数组在一个命令中使用不同格式样式的几行代码。

import numpy as np

import matplotlib.pyplot as plt

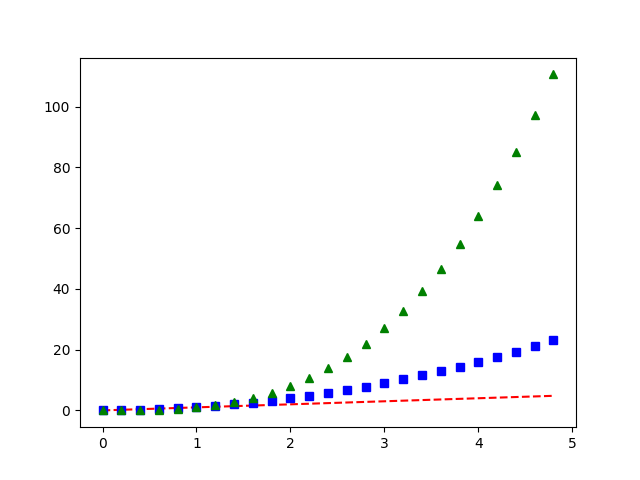
# evenly sampled time at 200ms intervals

t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles

plt.plot(t, t, *'r--'*, t, t\*\*2, *'bs'*, t, t\*\*3, *'g^'*)

plt.show()



## Controlling line properties

line有许多你可以设置的属性:线宽、dash风格、抗锯齿等等;参见matplotlib.lines.Line2D。如下有几种设置行属性的方法：

* 使用关键字参数

plt.plot(x, y, linewidth=2.0)

* 使用Line2D实例的setter方法。plot返回一个Line2D对象的列表;线1，线2=绘图(x1，y1，x2，y2)。在下面的代码中，我们假设我们只有一行，所以返回的列表长度为1。我们使用元组解压缩，来获取列表的第一个元素:

line, = plt.plot(x, y, *'-'*)

line.set\_antialiased(False) # turn off antialising

* 使用setp()命令。下面的例子使用了一个matlab\_style命令来在一个行列表中设置多个属性。setp可以透明地使用对象列表或单个对象。您可以使用python关键字参数或matlab样式的字符串/值对.

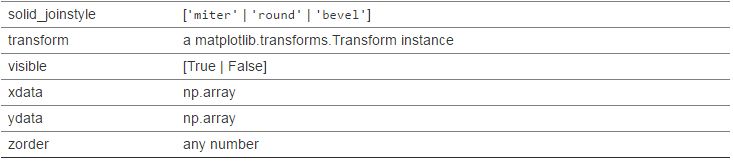
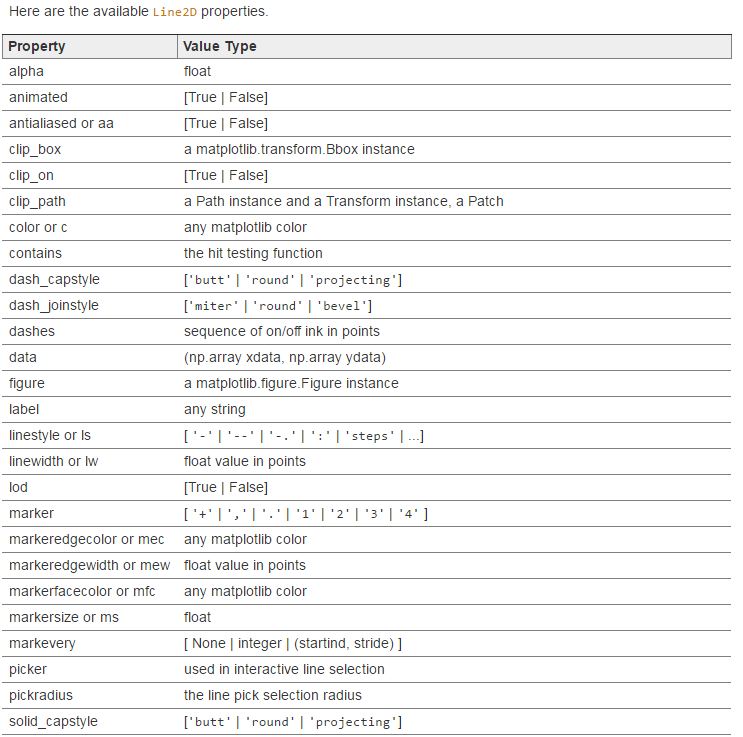
lines = plt.plot(x1, y1, x2, y2)

# use keyword args

plt.setp(lines, color=*'r'*, linewidth=2.0)

# or MATLAB style string value pairs

plt.setp(lines, *'color'*, *'r'*, *'linewidth'*, 2.0)



lines = plt.plot([1,2,3])

plt.setp(lines)

agg\_filter: unknown

alpha: float (0.0 transparent through 1.0 opaque)

animated: [True | False]

antialiased or aa: [True | False]

axes: an :class:`~matplotlib.axes.Axes` instance

clip\_box: a :class:`matplotlib.transforms.Bbox` instance

clip\_on: [True | False]

clip\_path: [ (:class:`~matplotlib.path.Path`, :class:`~matplotlib.transforms.Transform`) | :class:`~matplotlib.patches.Patch` | None ]

color or c: any matplotlib color

contains: a callable function

dash\_capstyle: ['butt' | 'round' | 'projecting']

dash\_joinstyle: ['miter' | 'round' | 'bevel']

dashes: sequence of on/off ink in points

drawstyle: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']

figure: a :class:`matplotlib.figure.Figure` instance

fillstyle: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']

gid: an id string

label: string or anything printable with '%s' conversion.

linestyle or ls: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | ``'-'`` | ``'--'`` | ``'-.'`` | ``':'`` | ``'None'`` | ``' '`` | ``''``]

linewidth or lw: float value in points

marker: :mod:`A valid marker style <matplotlib.markers>`

markeredgecolor or mec: any matplotlib color

markeredgewidth or mew: float value in points

markerfacecolor or mfc: any matplotlib color

markerfacecoloralt or mfcalt: any matplotlib color

markersize or ms: float

markevery: [None | int | length-2 tuple of int | slice | list/array of int | float | length-2 tuple of float]

path\_effects: unknown

picker: float distance in points or callable pick function ``fn(artist, event)``

pickradius: float distance in points

rasterized: [True | False | None]

sketch\_params: unknown

snap: unknown

solid\_capstyle: ['butt' | 'round' | 'projecting']

solid\_joinstyle: ['miter' | 'round' | 'bevel']

transform: a :class:`matplotlib.transforms.Transform` instance

url: a url string

visible: [True | False]

xdata: 1D array

ydata: 1D array

zorder: any number

## Working with multiple figures and axes

MATLAB和pyplot都有当前图形和当前轴的概念。所有的绘图指令都适用于当前的轴。函数gca()返回当前的轴(一个matplotlib.axes.Axes实例)和gcf()返回当前图形(matplotlib.figure.Figure实例)。通常情况下，你不必担心这些，因为这些都是在后台处理的。下面是创建两个子图的脚本：

import numpy as np

import matplotlib.pyplot as plt

def **f**(t):

return np.exp(-t) \* np.cos(2\*np.pi\*t)

t1 = np.arange(0.0, 5.0, 0.1)

t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)

plt.subplot(211)

plt.plot(t1, f(t1), *'bo'*, t2, f(t2), *'k'*)

plt.subplot(212)

plt.plot(t2, np.cos(2\*np.pi\*t2), *'r--'*)

plt.show()

在这里figure()命令是可选的，因为figure(1)在默认情况下会被创建。如果您不手动指定任何轴，那么将默认创建子图subplot(111)。subplot()命令指定了numrows、numcols、fignum，其中fignum范围从1到numrows \*numcols。如果numrows \*numcols小于10，subplot命令中的逗号是可选的。因此，subplot(211)与subplot(2,1,1)相同。你可以创建任意数量的子图和轴。如果你想手动放置坐标轴，也就是。而不是在一个矩形网格中，使用axes()命令，它允许你指定位置做为轴([left, bottom, width, height])，所有的值都在小数部分(0到1)坐标中。参见pylab示例代码:axes\_demo.py是一个手动放置轴和pylab示例示例代码subplots\_demo.py是一个有很多子图的例子。

你可以通过调用多个figure()来创建多个图形，并使用一个递增的图形数字。当然，每个图形都可以包含你所需要的许多轴和子图:

import matplotlib.pyplot as plt

plt.figure(1) # the first figure

plt.subplot(211) # the first subplot in the first figure

plt.plot([1, 2, 3])

plt.subplot(212) # the second subplot in the first figure

plt.plot([4, 5, 6])

plt.figure(2) # a second figure

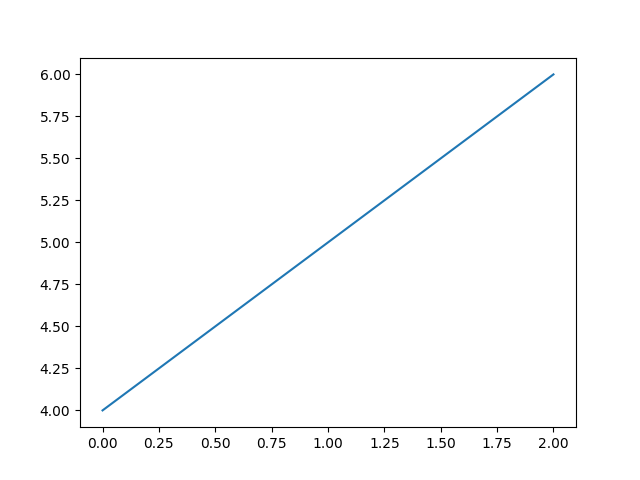
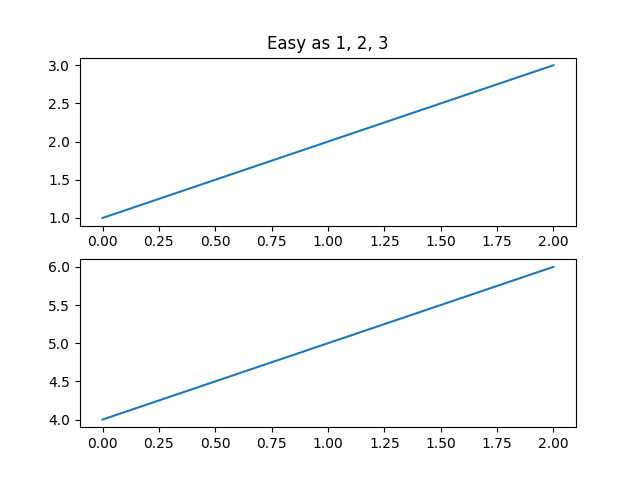
plt.plot([4, 5, 6]) # creates a subplot(111) by default

plt.figure(1) # figure 1 current; subplot(212) still current

plt.subplot(211) # make subplot(211) in figure1 current

plt.title(*'Easy as 1, 2, 3'*) # subplot 211 title

plt.show()



你可以清除当前的图形使用clf()函数，清除当前的刻度使用cla()函数。

如果您正在做大量的数据，那么您需要知道另外一件事:一个图形所需要的内存直到该图形用close()显式关闭时才会被完全释放。删除所有对图的引用，或者使用窗口管理器来杀死图中显示在屏幕上的窗口，这是不够的，因为pyplot在调用close()之前维护内部引用。

## Working with text

text()命令可用于在任意位置添加文本，而xlabel()、ylabel()和title()用于在指定的位置添加文本(参见文本介绍，以获得更详细的示例)。

import numpy as np

import matplotlib.pyplot as plt

# Fixing random state for reproducibility

np.random.seed(19680801)

mu, sigma = 100, 15

x = mu + sigma \* np.random.randn(10000)

# the histogram of the data

n, bins, patches = plt.hist(x, 50, normed=1, facecolor=*'g'*, alpha=0.75)

plt.xlabel(*'Smarts'*)

plt.ylabel(*'Probability'*)

plt.title(*'Histogram of IQ'*)

plt.text(60, .025, *r'$\mu=100,\ \sigma=15$'*)

plt.axis([40, 160, 0, 0.03])

plt.grid(True)

plt.show()

