# 3教程

## 3.1 介绍

### 3.1.1 Pyplot教程

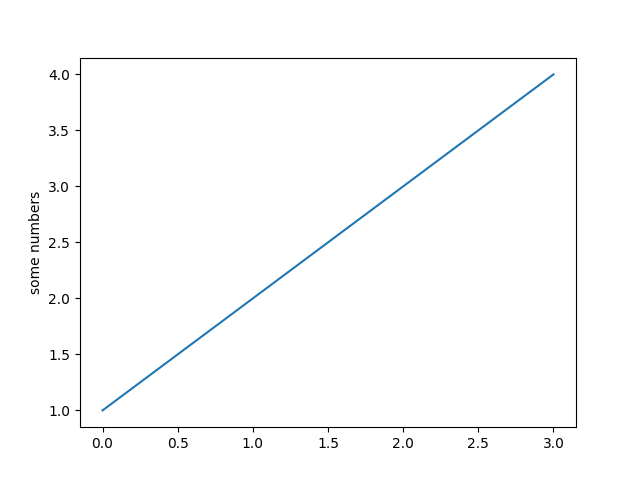
Matplotlib.pyplot是一个命令样式函数集合，它使matplotlib像MATLAB一样工作。每一个pyplot函数都会对图像做出修改，例如，创建一个图像，图形中创建一块绘图区域，在绘图区域中绘制一些线条，用标签装饰图像等。

import matplotlib.pyplot as plt

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

plt.ylabel(*'some numbers'*)

plt.show()



如果只提供给plot()函数一个序列或列表作为参数，matplotlib会假设这个序列是y轴的值，并且会自动为你生成x轴的值。由于python的范围是从0开始的，所以x轴向量默认从0开始，步长为1，与Y序列个数相同，即[0,1,2,3]。

Plot()是一个通用的命令，它可以接受任意数量的参数，例如：

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

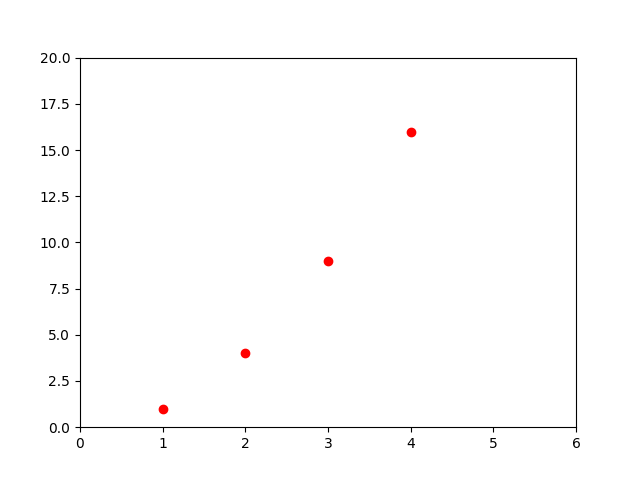
对于每一对x和y的参数，有一个可选的第三参数，一个格式化的字符串用来标识绘图线条的颜色和形状，默认的参数是’b-’，表示蓝色的实线。例如，用红色圆圈来绘图:

import matplotlib.pyplot as plt

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

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

plt.show()



axis()命令指定了刻度轴 [xmin, xmax, ymin, ymax]。

如果matplotlib只能用于处理列表，那么它是没有用的。一般来说，都是使用numpy数组。事实上，所有的序列都转化为了numpy数组。

import numpy as np

import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals

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

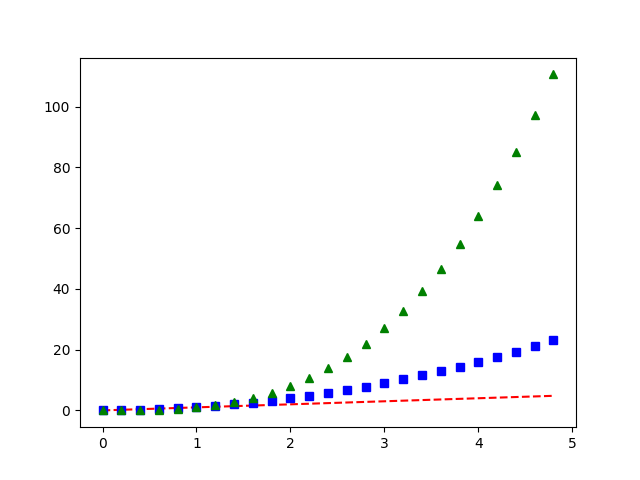
# red dashes, blue squares and green triangles

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

plt.show()

序列t:

[ 0. 0.2 0.4 0.6 0.8 1. 1.2 1.4 1.6 1.8 2. 2.2 2.4 2.6 2.8 3. 3.2 3.4 3.6 3.8 4. 4.2 4.4 4.6 4.8]



#### **设置line的属性**

可参照matplotlib.lines.Line2D

* 使用关键字参数

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

* 使用Line2D实例的set方法，plot返回了一个Line2D列表；比如line1, line2 = plot(x1, y1, x2, y2)。例如：

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

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

* 使用setp（）命令。使用setp命令可以为一个列表中的line统一设置属性，第一个参数是line列表。

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

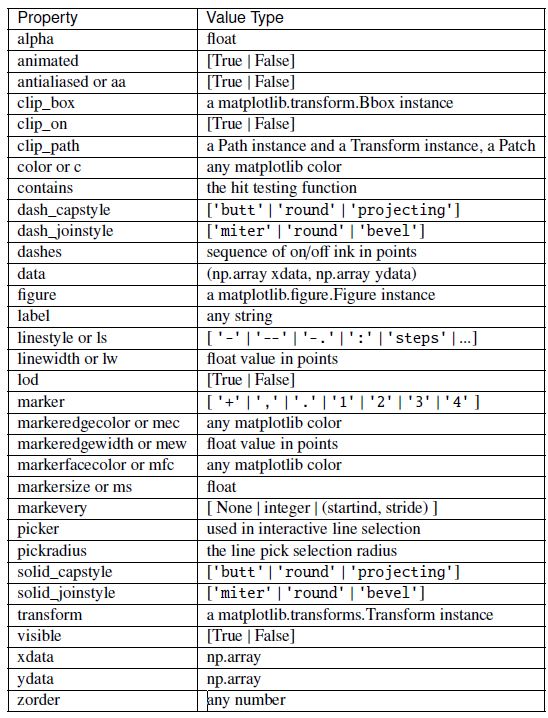
# use keyword args

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

# or MATLAB style string value pairs

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

Line2D的属性：



要获取可设置line属性的列表，可以调用setp函数使用line或lines作为参数。

In [69]: lines = plt.plot([1, 2, 3])

In [70]: plt.setp(lines)

alpha: float

animated: [True | False]

antialiased or aa: [True | False]

**使用多个图标和轴**

Matlib和pylot都有当前图形和当前轴的概念，函数gca()返回当前轴（一个matpplotlib.axes.Axes实例），函数gcf()返回当前图（matplotlib.figure.Figure实例）。

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)

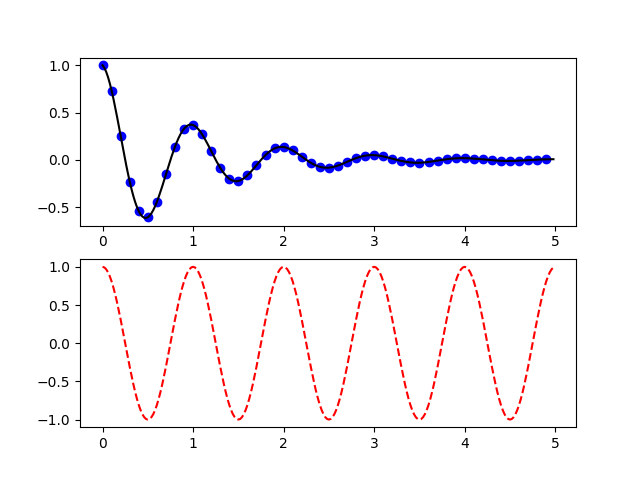
plt.subplot(211)

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()



Figure()命令是可选的，figure（1）会被默认创建。就像你不手动指定任何轴，subplot(111)会被默认创建一样。Subplot()命令指定numrows、numcols、fignum,fignum的范围从1到numrows\*numcols。如果numrows\*numcols<10,那么subplot命令中的逗号是可选的，因此subplot（211）等同于subplot(2,1,1)。

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

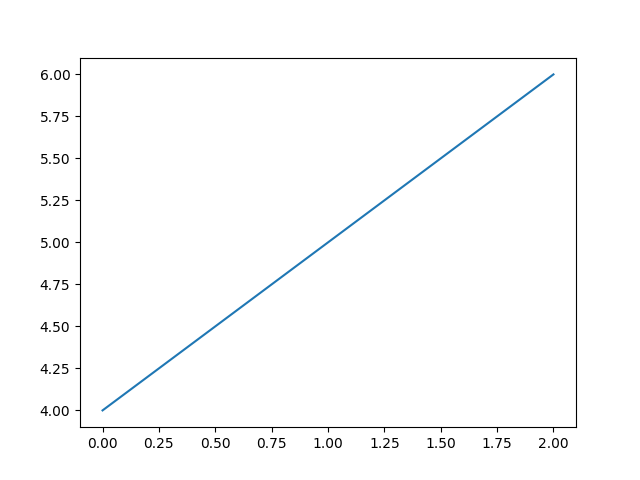
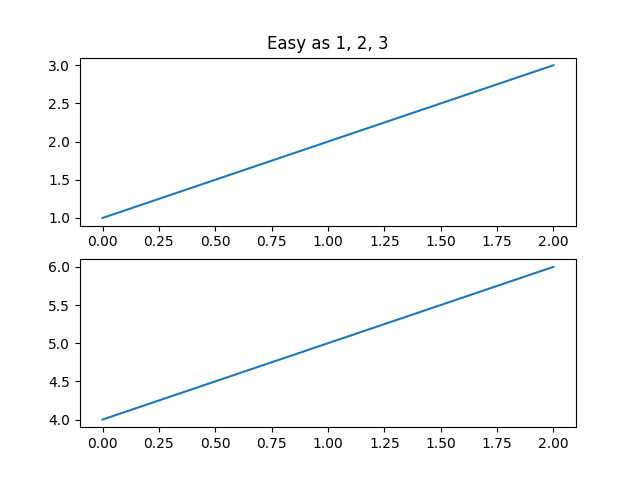
plt.plot([4, 5, 6]) # creates a subplot(111) by default

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()



创建多个图表使用调用多个figure()函数，参数随着递增。Clf()函数用来清除当前图表，cla()函数用来清除当前轴。

如果需要创建大量的图，你需要直到额外的一件事：一个图形所需要的内存直到这个图形显示的用close()函数关闭才会释放。删除这个图形的引用，或者杀死图形显示的窗口是并不足够的，因为pyplot维护内部的引用，直到close()被调用。

#### 处理文本

text()命令可用于在任意位置添加文本，xlabel()、ylabel()、title用于在指定位置添加文本。

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])

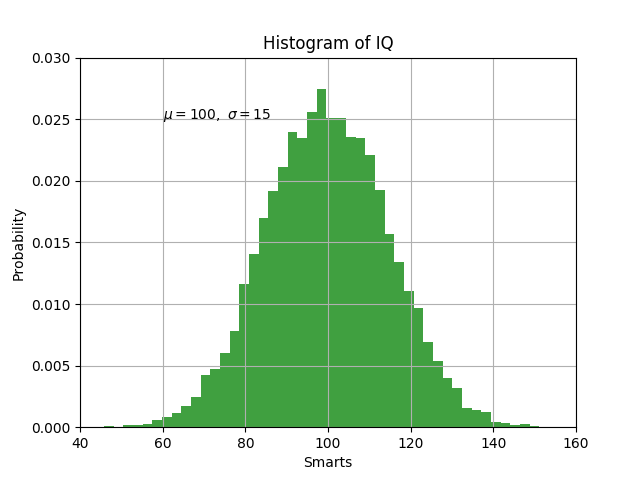
plt.grid(True)

plt.show()

注：seed（）用于指定随机数生成时所用算法开始的整数值，如果使用相同的seed( )值，则每次生成的随即数都相同，如果不设置这个值，则系统根据时间来自己选择这个值，此时每次生成的随机数因时间差异而不同。

randn返回一个样本，具有标准正态分布。

hist函数，给定一堆数据，统计数据在某一值的个数。plot是给定横/纵坐标向量，描绘点列。



所有的text()命令返回一个matplotlib.text.Text实例。与lines类似，也可以通过setp()传递关键字参数设置text属性。

t = plt.xlabel(*'my data'*, fontsize=14, color=*'red'*)

#### 在文本中使用数学表达式

Matplotlib在text中接受TeX表达式，例如在title中写一个表达式，你可以写一个被$包围的特克斯表达式：

plt.title(*r'$\sigma\_i=15$'*)

字符串前的r表示这个字符串是个raw字符串，而不是转义字符串。Matplotlib有其内置的TeX表达式解析器和布局引擎，也有自己的数学表达式字体。

#### 注释文本

Text的一个常见的用法是用来注释图形的相关特性，annotate()方法用来实现这个功能。在注释中，有两点需要考虑:被注释的位置由参数xy和文本xytext的位置表示。这两个参数都是(x，y)元组。

import numpy as np

import matplotlib.pyplot as plt

ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)

s = np.cos(2\*np.pi\*t)

line, = plt.plot(t, s, lw=2)

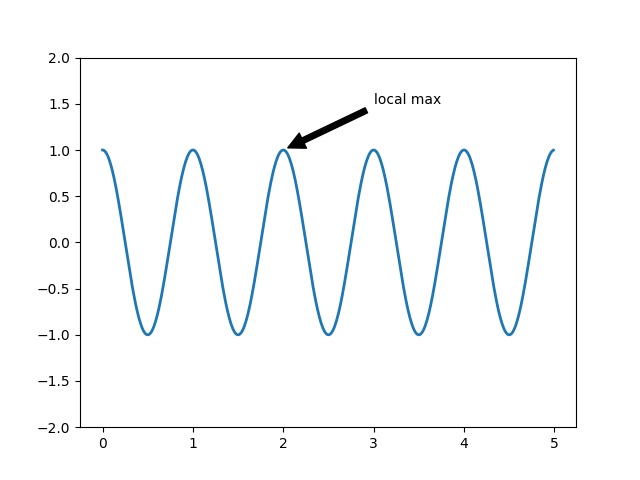
plt.annotate(*'local max'*, xy=(2, 1), xytext=(3, 1.5),

arrowprops=dict(facecolor=*'black'*, shrink=0.05),

)

plt.ylim(-2,2)

plt.show()



#### 对数和其他非线性轴

Matplotlib.pyplot不仅支持线性轴表，也支持对数以及loggit表，这通常被用于数据跨越多个数量级的情况。改变轴的刻度是很简单的：plt.xscale(‘log’)

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.ticker import NullFormatter # useful for `logit` scale

# Fixing random state for reproducibility

np.random.seed(19680801)

# make up some data in the interval ]0, 1[

y = np.random.normal(loc=0.5, scale=0.4, size=1000)

y = y[(y > 0) & (y < 1)]

y.sort()

x = np.arange(len(y))

# plot with various axes scales

plt.figure(1)

# linear

plt.subplot(221)

plt.plot(x, y)

plt.yscale(*'linear'*)

plt.title(*'linear'*)

plt.grid(True)

# log

plt.subplot(222)

plt.plot(x, y)

plt.yscale(*'log'*)

plt.title(*'log'*)

plt.grid(True)

# symmetric log

plt.subplot(223)

plt.plot(x, y - y.mean())

plt.yscale(*'symlog'*, linthreshy=0.01)

plt.title(*'symlog'*)

plt.grid(True)

# logit

plt.subplot(224)

plt.plot(x, y)

plt.yscale(*'logit'*)

plt.title(*'logit'*)

plt.grid(True)

# Format the minor tick labels of the y-axis into empty strings with

# `NullFormatter`, to avoid cumbering the axis with too many labels.

plt.gca().yaxis.set\_minor\_formatter(NullFormatter())

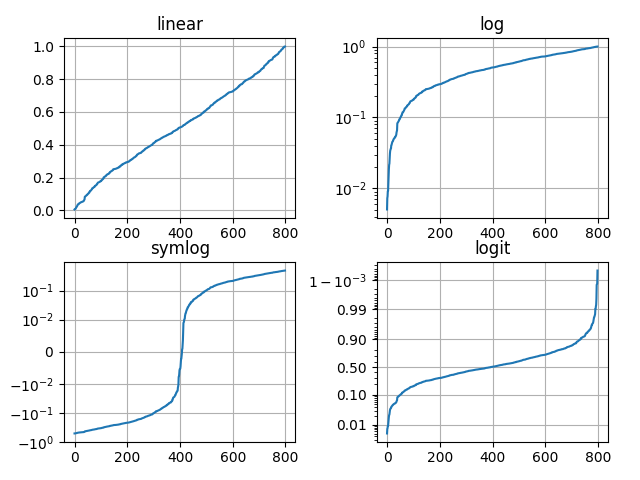
# Adjust the subplot layout, because the logit one may take more space

# than usual, due to y-tick labels like "1 - 10^{-3}"

plt.subplots\_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,

wspace=0.35)

plt.show()



### 3.1.2 图片教程

In [2]: import matplotlib.pyplot as plt

In [3]: import matplotlib.image as mpimg

In [4]: import numpy as np

#### 将图片数据导入numpy数组

加载图像可以通过Pillow库。Matplotlib只支持PNG格式的图像。

img=mpimg.imread(*'dog.png'*)

>>>

[[[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

...,

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]]

[[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

...,

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]]

[[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

...,

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]]

...,

[[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

...,

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]]

[[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

...,

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]]

[[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]

...,

[ 0. 0. 0. 0.]

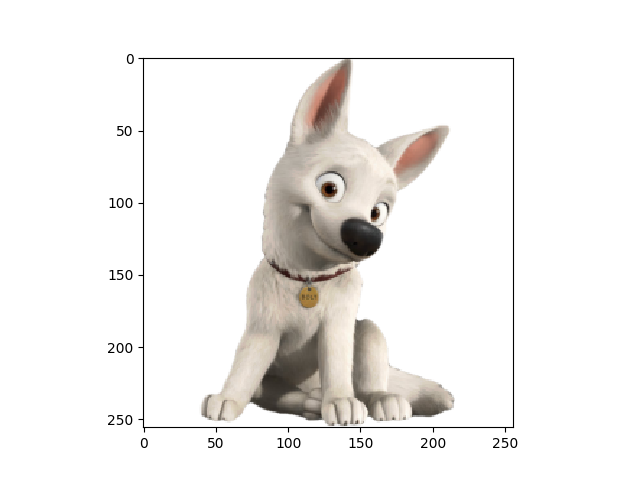
[ 0. 0. 0. 0.]

[ 0. 0. 0. 0.]]]

#### 绘制numpy数组成为图像

在matplotlib中使用imshow操作图像的numpy数组。

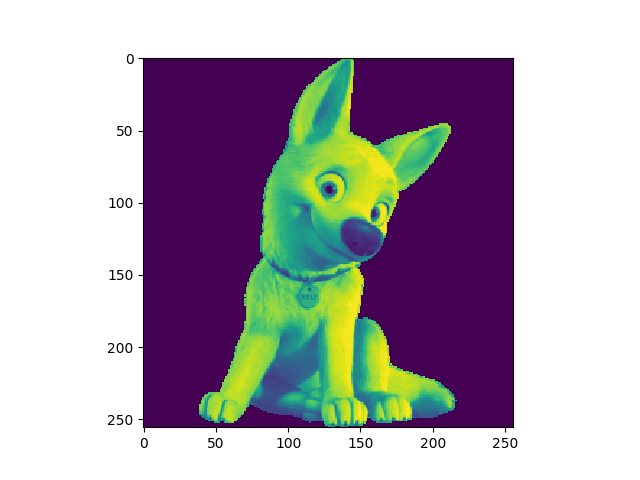
imgplot = plt.imshow(img)



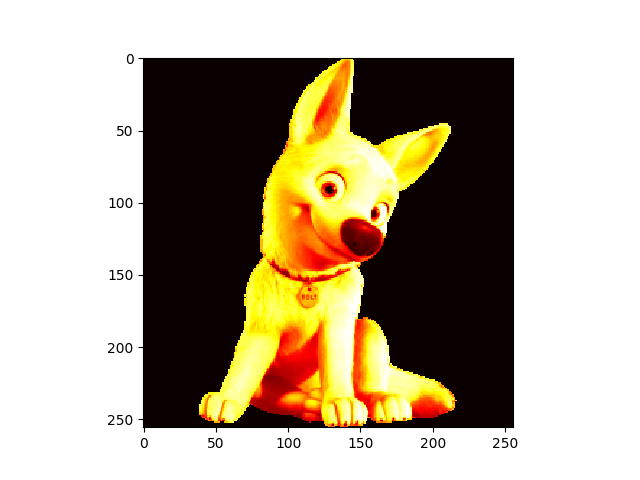
#### 将伪色应用于图像上

伪色可以是一个很有用的工具，可以更容易地增强对比度和可视化数据。这在使用投影仪进行数据演示时尤其有用——他们的对比通常很差。伪色只与单通道、灰度、亮度图像有关。由于R、G和B都是相似的(请参阅上面或在您的数据中)，我们可以选择一个数据通道:

lum\_img = img[:,:,0]



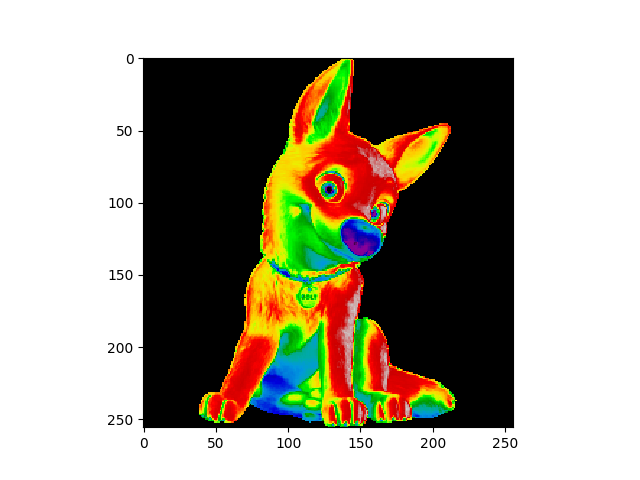
使用亮度图像：plt.imshow(lum\_img, cmap=*"hot"*)



注意，您还可以使用setcmap()方法修改现有的绘图对象的colormaps。

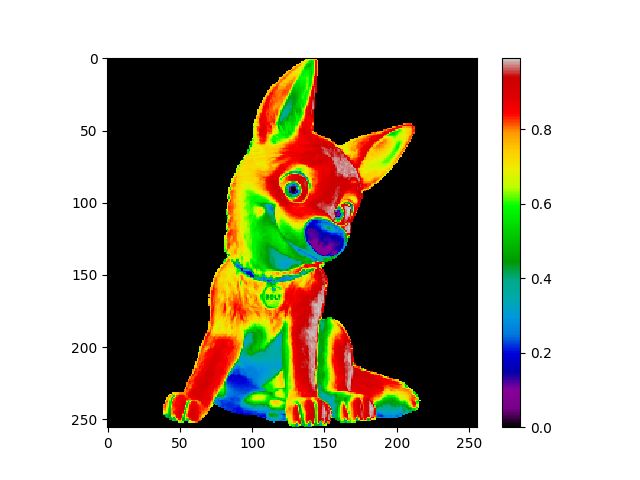
imgplot = plt.imshow(lum\_img)

imgplot.set\_cmap(*'nipy\_spectral'*)



#### 颜色刻度

plt.colorbar()



#### 检查指定的数据范围

有时你想要增强图像的对比度，或者在特定区域扩大对比度，同时牺牲不太大的细节，或者无关紧要。找到有趣的区域的一个好工具是直方图。为了创建图像数据的直方图，我们使用hist()函数。

plt.hist(lum\_img.ravel(), bins=256, range=(0.0, 1.0), fc=*'k'*, ec=*'k'*)

### 3.1.3 使用GridSpec定制子图表的位置

GridSpec 指定一个子图将被放置的网格的几何形状。需要设置网格的行数和列数。可选地，子图布局参数(如:左、右等)可以调整。

SubplotSpec 指定给定GridSpec中的子图的位置

Subplot2grid() 一个类似于subplot()的帮助函数，但是使用基于0索引然后让子图占据多个单元格。

#### 使用subplot2grid的基本例子

ax = plt.subplot2grid((2, 2), (0, 0)) 等价于

ax = plt.subplot(2, 2, 1)

import numpy as np

import matplotlib.pyplot as plt

ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)

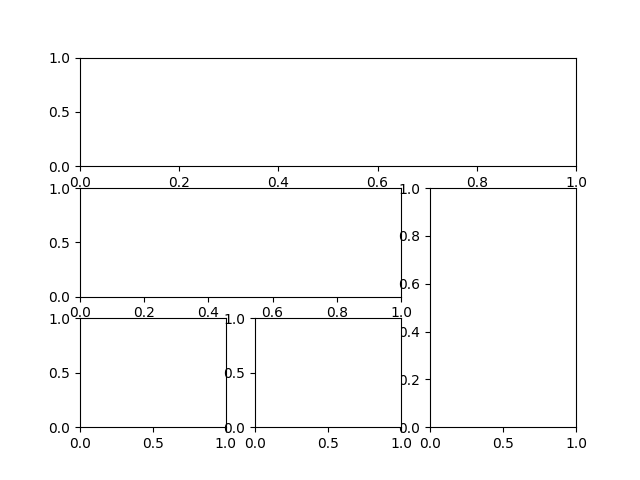
ax2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)

ax3 = plt.subplot2grid((3, 3), (1, 2), rowspan=2)

ax4 = plt.subplot2grid((3, 3), (2, 0))

ax5 = plt.subplot2grid((3, 3), (2, 1))

plt.show()



#### GridSpec and SubplotSpec

你可以显式地创建GridSpec，并使用它们创建子图。

ax = plt.subplot2grid((2, 2), (0, 0)) 等价于：

import matplotlib.gridspec as gridspec

import matplotlib.pyplot as plt

gs = gridspec.GridSpec(2, 2)

ax = plt.subplot(gs[0, 0])

plt.show()

GridSpec实例提供了类似数组（2D或1D）的索引，并返回subplotspec实例。对于一个SubplotSec跨越多个网格，可以使用slice：

import matplotlib.gridspec as gridspec

import matplotlib.pyplot as plt

gs = gridspec.GridSpec(3, 3)

ax1 = plt.subplot(gs[0, :])

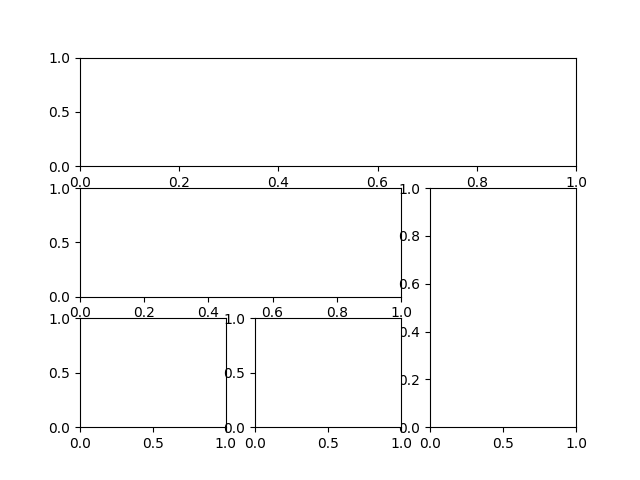
ax2 = plt.subplot(gs[1, :-1])

ax3 = plt.subplot(gs[1:, -1])

ax4 = plt.subplot(gs[-1, 0])

ax5 = plt.subplot(gs[-1, -2])

plt.show()



#### 调整GridSpec布局

import matplotlib.gridspec as gridspec

import matplotlib.pyplot as plt

gs1 = gridspec.GridSpec(3, 3)

gs1.update(left=0.05, right=0.48, wspace=0.05)

ax1 = plt.subplot(gs1[:-1, :])

ax2 = plt.subplot(gs1[-1, :-1])

ax3 = plt.subplot(gs1[-1, -1])

gs2 = gridspec.GridSpec(3, 3)

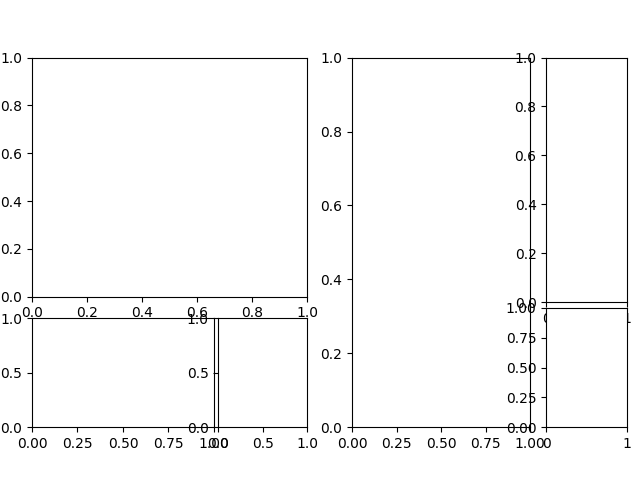
gs2.update(left=0.55, right=0.98, hspace=0.05)

ax4 = plt.subplot(gs2[:, :-1])

ax5 = plt.subplot(gs2[:-1, -1])

ax6 = plt.subplot(gs2[-1, -1])

plt.show()



#### GridSpec with Varying Cell Sizes

默认情况下，GridSpec创建大小相同的单元格。你可以调整行和列的相对高度和宽度。请注意，绝对值是无意义的，只有它们的相对比率是重要的。

import matplotlib.gridspec as gridspec

import matplotlib.pyplot as plt

gs = gridspec.GridSpec(2, 2,

width\_ratios=[1, 3],

height\_ratios=[9, 1]

)

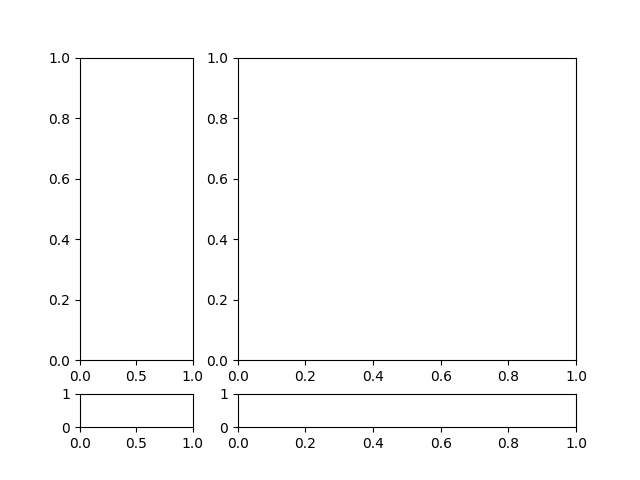
ax1 = plt.subplot(gs[0])

ax2 = plt.subplot(gs[1])

ax3 = plt.subplot(gs[2])

ax4 = plt.subplot(gs[3])

plt.show()



#### 3.1.4 紧凑的布局指南

### 

### matplotlib是Python最著名的绘图库，它提供了一整套和MATLAB类似的绘图函数集，十分适合编写短小的脚本程序进行快速绘图。此外，matplotlib采用面向对象的技术实现，因此组成图表的各个元素都是对象，在编写较大的应用程序时通过面向对象的方式使用matplotlib将更加有效。

matplotlib的文档十分完备，并且其展示页面中有上百幅图表的缩略图及其源程序。因此如果读者需要绘制某种类型的图表，只需要在这个页面中“浏览/复制/粘贴”一下，基本上都能快速解决。

<http://matplotlib.sourceforge.net/gallery.html>

matplotlib的展示页面地址

## 快速绘图

matplotlib的pyplot模块提供了和MATLAB类似的绘图API，方便用户快速绘制二维图表。我们先看一个简单的例子：

# -\*- coding: utf-8 -\*-

*"""*

*绘制简单的曲线。*

*"""*

import numpy as np

import matplotlib.pyplot as plt

print help(np)

x = np.linspace(0, 10, 1000)

y = np.sin(x)

z = np.cos(x\*\*2)

plt.figure(figsize=(8,4))

plt.plot(x,y,label=*"$sin(x)$"*,color=*"red"*,linewidth=2)

plt.plot(x,z,*"b--"*,label=*"$cos(x^2)$"*)

plt.xlabel(*"Time(s)"*)

plt.ylabel(*"Volt"*)

plt.title(*"PyPlot First Example"*)

plt.ylim(-1.2,1.2)

plt.legend()

plt.show()

* 首先载入matplotlib的绘图模块pyplot，并且重命名为plt。