

Matplotlib

—绘制精美的图表

目录

□ 快速绘图

- 快速绘图
- 绘制多轴图
- 图中图
- 次坐标轴
- 坐标轴设定

□ 绘图函数简介

- 对数坐标图
- 极坐标图

目录

- 柱状图
- 散列图
- 图像
- 等值线图
- 三维绘图
- 3D作图（例）
- Animation 动画

matplotlib 是python最著名的绘图库，它提供了一整套和**matlab**相似的命令**API**，十分适合交互式地进行制图。而且也可以方便地将它作为绘图控件，嵌入**GUI**应用程序中。

它的文档相当完备，并且**Gallery**页面中有上百幅缩略图，打开之后都有源程序。因此如果你需要绘制某种类型的图，只需要在这个页面中浏览/复制/粘贴一下，基本上都能搞定。

展示页面的地址：

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

~~<https://matplotlib.org/2.0.2/gallery.html>~~

快速绘图

□ 快速绘图

`matplotlib`的`pyplot`子库提供了和`matlab`类似的绘图API，方便用户快速绘制2D图表。
(`matplotlib_simple_plot.py`)

pylab 模块

`matplotlib`还提供了名为`pylab`的模块，其中包括了许多`numpy`和`pyplot`中常用的函数，方便用户快速进行计算和绘图，可以用于`IPython`中的快速交互式使用。

快速绘图

matplotlib中的快速绘图的函数库可以通过如下语句载入：

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

接下来调用**figure**创建一个绘图对象，并且使它成为当前的绘图对象。

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

通过**figsize**参数可以指定绘图对象的宽度和高度，单位为英寸；**dpi**参数指定绘图对象的分辨率，即每英寸多少个像素，缺省值为**80**。因此本例中所创建的图表窗口的宽度为 $8 \times 80 = 640$ 像素。

快速绘图

也可以不创建绘图对象直接调用接下来的 **plot** 函数直接绘图，**matplotlib** 会自动创建一个绘图对象。

如果需要同时绘制多幅图表的话，可以是给 **figure** 传递一个整数参数指定图标的序号，如果所指定序号的绘图对象已经存在的话，将不创建新的对象，而只是让它成为当前绘图对象。

下面的两行程序通过调用 **plot** 函数在当前的绘图对象中进行绘图：

```
plt.plot(x,y,label="$sin(x)$",color="red",linewidth=2)  
plt.plot(x,z,"b--",label="$cos(x^2)$")
```

快速绘图

```
plt.plot(x,y,label="$sin(x)$",color="red",linewidth=2)  
plt.plot(x,z,"b--",label="$cos(x^2)$")
```

plot函数的调用方式很灵活，第一句将**x,y**数组传递给**plot**之后，用关键字参数指定各种属性：

- • **label** : 给所绘制的曲线一个名字，此名字在图示(legend)中显示。只要在字符串前后添加"\$"符号，**matplotlib**就会使用其内嵌的**latex**引擎绘制的数学公式。
- • **color** : 指定曲线的颜色
- • **linewidth** : 指定曲线的宽度
- 第三个参数**'b--'**指定曲线的颜色和线型

快速绘图

接下来通过一系列函数设置绘图对象的各个属性：

```
plt.xlabel("Time(s)")  
plt.ylabel("Volt")  
plt.title("PyPlot First Example")  
plt.ylim(-1.2,1.2)  
plt.legend()
```

- • **xlabel / ylabel** : 设置X轴/Y轴的文字
- • **title** : 设置图表的标题
- • **ylim** : 设置Y轴的范围
- • **legend** : 显示图示

最后调用**plt.show()**显示出创建的所有绘图对象。

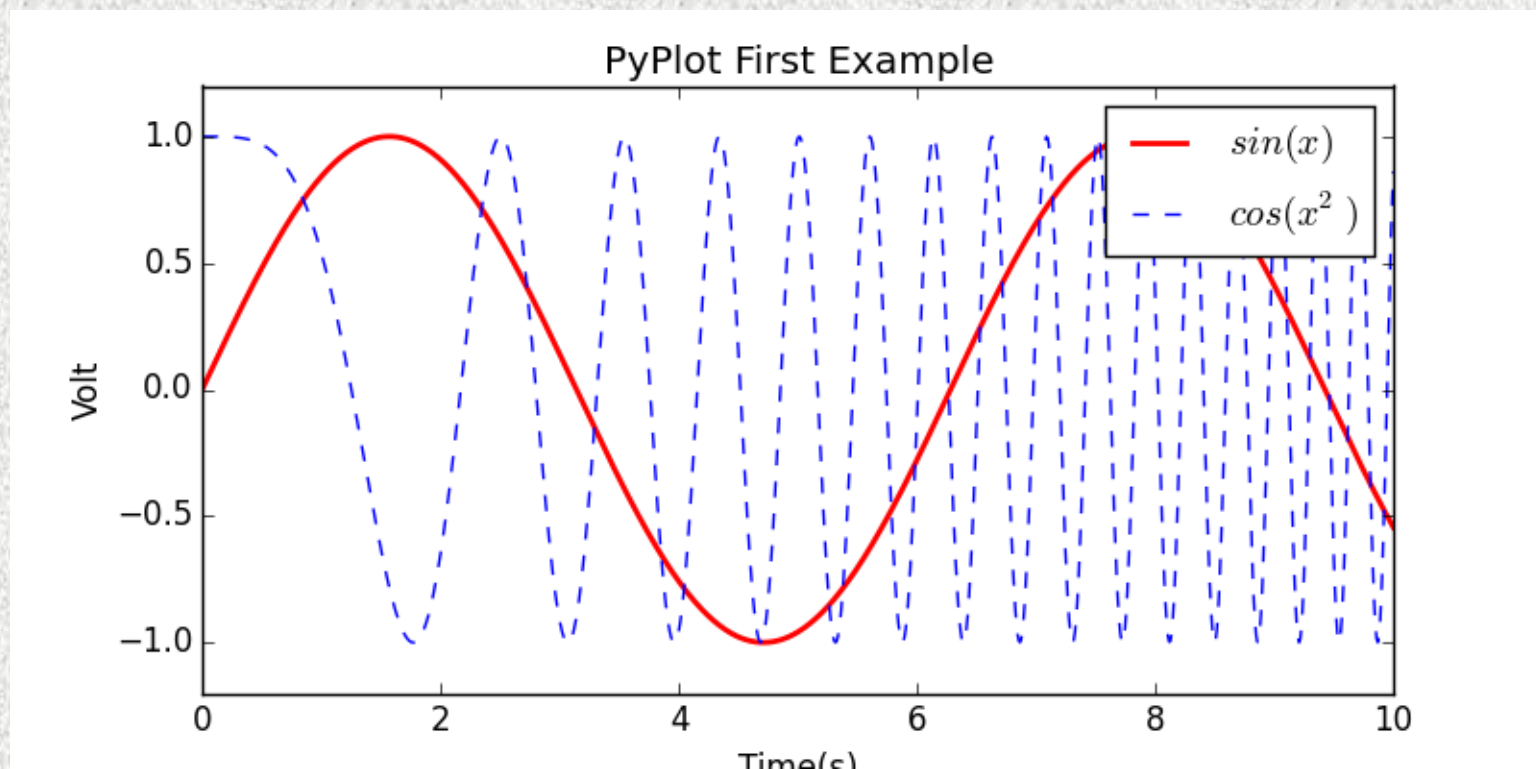
快速绘图

```
import numpy as np
import matplotlib.pyplot as plt

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()
```

快速绘图



快速绘图

还可以调用`plt.savefig()`将当前的Figure对象保存成图像文件，图像格式由图像文件的扩展名决定。下面的程序将当前的图表保存为“test.png”，并且通过`dpi`参数指定图像的分辨率为120，因此输出图像的宽度为“8X120 = 960”个像素。

```
run matplotlib_simple_plot.py  
plt.savefig("test.png",dpi=120)
```

实际上不需要调用`show()`显示图表，可以直接用`savefig()`将图表保存成图像文件。使用这种方法可以很容易编写出批量输出图表的程序。

快速绘图

□ 绘制多轴图

一个绘图对象(**figure**)可以包含多个轴(**axis**), 在**Matplotlib**中用轴表示一个绘图区域, 可以将其理解为子图。上面的第一个例子中, 绘图对象只包括一个轴, 因此只显示了一个轴(子图(**Axes**))。可以使用**subplot**函数快速绘制有多个轴的图表。**subplot**函数的调用形式如下:

```
subplot(numRows, numCols, plotNum)
```

快速绘图

`subplot`将整个绘图区域等分为`numRows`行和 `numCols`列个子区域，然后按照从左到右，从上到下的顺序对每个子区域进行编号，左上子区域的编号为1。如果`numRows`，`numCols`和`plotNum`这三个数都小于10的话，可以把它们缩写为一个整数，例如 `subplot(323)`和`subplot(3,2,3)`是相同的。`subplot`在`plotNum`指定的区域中创建一个轴对象。如果新创建的轴和之前创建的轴重叠的话，之前的轴将被删除。

快速绘图

下面的程序创建**3行2列**共**6个**轴，通过**axisbg**参数给每个轴设置不同的背景颜色。

```
for idx, color in enumerate("rgbbyck"):
    plt.subplot(320+idx+1, axisbg=color)
plt.show()
```

如果希望某个轴占据整个行或者列的话，可以如下调用**subplot**:

```
plt.subplot(221) # 第一行的左图
plt.subplot(222) # 第一行的右图
plt.subplot(212) # 第二整行
plt.show()
```

快速绘图

还可以调用`subplot2grid()`进行复杂的表格布局。

```
subplot2grid(shape, loc, rowspan=1, colspan=1, **kwargs)
```

其中:**shape**为表格形状的元组: (行数,列数)。loc为子图左上角所在的坐标: (行,列)。 **rowspan**和**colspan**分别为子图所占据的行数和列数。

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6, 6))
ax1 = plt.subplot2grid((3,3),(0,0),colspan=2)
ax2 = plt.subplot2grid((3,3),(0,2),rowspan=2)
ax3 = plt.subplot2grid((3,3),(1,0),rowspan=2)
ax4 = plt.subplot2grid((3,3),(2,1),colspan=2)
ax5 = plt.subplot2grid((3,3),(1,1))
plt.suptitle("subplot2grid")
plt.show()
```


快速绘图

gridspec 同样能帮助画出前文的图，
gridspec 的使用很方便的，因为它允许使用索引的方式指定小图的大小和位置。

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

plt.figure(figsize=(8, 8))
gs = gridspec.GridSpec(3, 3)

ax6 = plt.subplot(gs[0, :])
ax7 = plt.subplot(gs[1, :2])
ax8 = plt.subplot(gs[1:, 2])
ax9 = plt.subplot(gs[-1, 0])
ax10 = plt.subplot(gs[-1, -2])
```

快速绘图

当绘图对象中有多个轴的时候，可以通过工具栏中的**Configure Subplots**按钮，交互式地调节轴之间的间距和轴与边框之间的距离。如果希望在程序中调节的话，可以调用 `subplots_adjust` 函数，它有 `left`, `right`, `bottom`, `top`, `wspace`, `hspace` 等几个关键字参数，这些参数的值都是0到1之间的小数，它们是以绘图区域的宽高为1进行正规化之后的坐标或者长度。

```
plt.subplots_adjust?
```

快速绘图

subplot()返回它所创建的**Axes**对象，可以将它用变量保存起来，然后用**sca()**交替让它们成为当前**Axes**对象，并调用**plot()**在其中绘图。如果需要同时绘制多幅图表，可以给**figure()**传递一个整数参数指定**Figure**对象的序号，如果序号所指定的**figure**对象已经存在，将不创建新的对象，而只是让它成为当前的**figure**对象。下面的程序演示了如何依次在不同图表的不同子图中绘制曲线。

(matplotlib_multi_figure.py)

快速绘图

首先通过`figure()`创建了两个图表，它们的序号分别为**1**和**2**。然后在图表**2**中创建了上下并排的两个子图，并用变量**ax1**和**ax2**保存。

```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(1) # 创建图表1
plt.figure(2) # 创建图表2
ax1 = plt.subplot(211) # 在图表2中创建子图1
ax2 = plt.subplot(212) # 在图表2中创建子图2

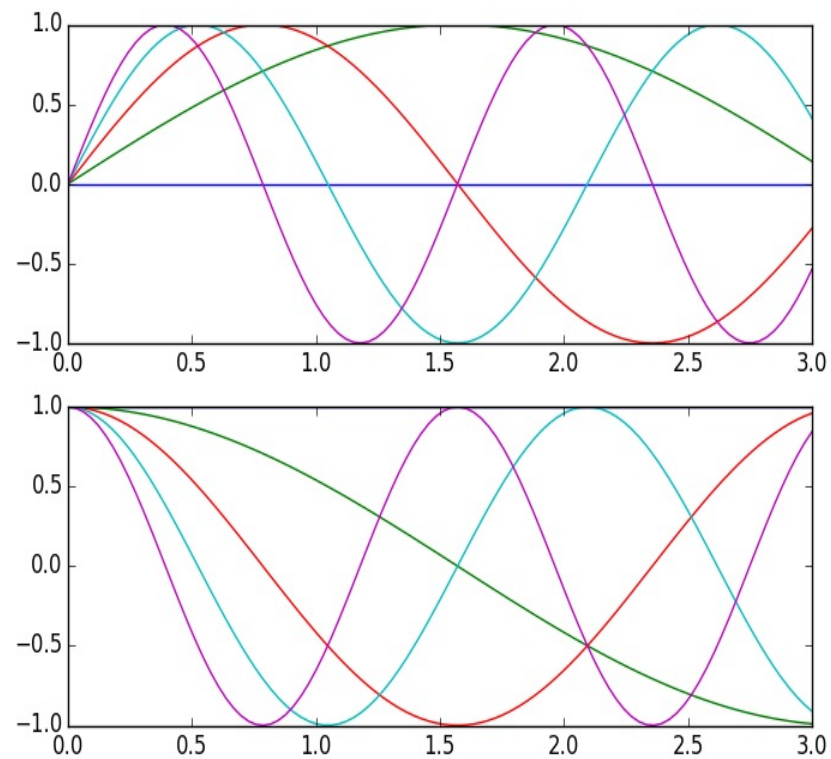
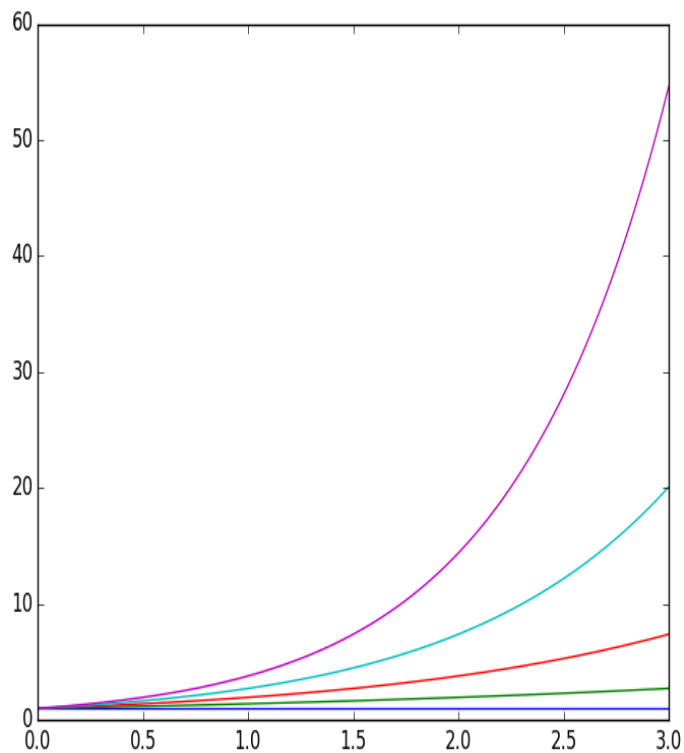
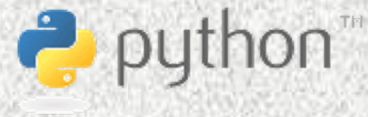
x = np.linspace(0, 3, 100)
```


快速绘图

在循环中，先调用`figure(1)`让图表1成为当前图表，并在其中绘图。然后调用`sca(ax1)`和`sca(ax2)`分别让子图`ax1`和`ax2`成为当前子图，并在其中绘图。当它们成为当前子图时，包含它们的图表2也自动成为当前图表，因此不需要调用`figure(2)`依次在图表1和图表2的两个子图之间切换，逐步在其中添加新的曲线

```
for i in xrange(5):  
    plt.figure(1) # 选择图表1  
    plt.plot(x, np.exp(i*x/3))  
    plt.sca(ax1) # 选择图表2的子图1  
    plt.plot(x, np.sin(i*x))  
    plt.sca(ax2) # 选择图表2的子图2  
    plt.plot(x, np.cos(i*x))  
plt.show()
```

快速绘图



快速绘图

□ 图中图

`matplotlib` 里一个很有意思的功能，叫做图中图(plot in plot)。通过它，可以在大图中嵌入小图。

```
import matplotlib.pyplot as plt
# 创建数据
x = [1, 2, 3, 4, 5, 6, 7]
y = [1, 3, 4, 2, 5, 8, 6]
#绘制大图：首先确定大图左下角的位置以及宽高：
left, bottom, width, height = 0.1, 0.1, 0.8, 0.8
```

注意，4个值都是占整个figure窗口的百分比。假设figure的大小是10x10，那么大图就被包含在由(1, 1)开始，宽8，高8的坐标系内。

快速绘图

```
fig = plt.figure()
```

```
ax1 = fig.add_axes([left, bottom, width, height])  
ax1.plot(x, y, 'r')  
ax1.set_xlabel('x')  
ax1.set_ylabel('y')  
ax1.set_title('title')
```

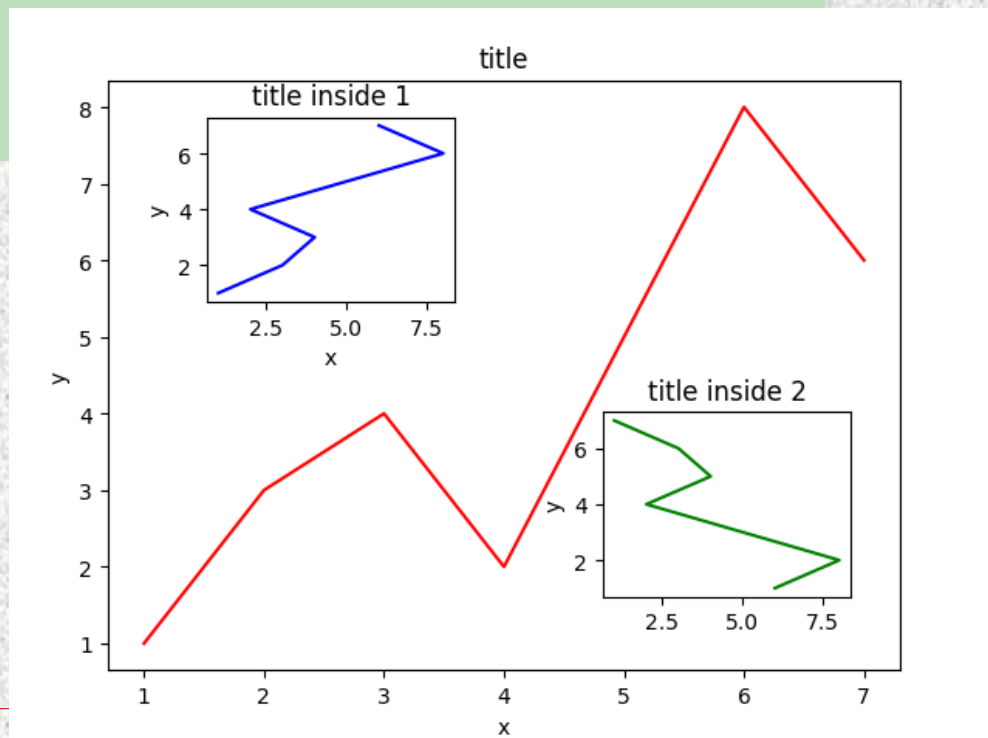
#绘制左上角的小图，步骤和绘制大图一样，注意坐标系位置和
大小的改变：

```
left, bottom, width, height = 0.2, 0.6, 0.25, 0.25  
ax2 = fig.add_axes([left, bottom, width, height])  
ax2.plot(y, x, 'b')  
ax2.set_xlabel('x')  
ax2.set_ylabel('y')  
ax2.set_title('title inside 1')
```


快速绘图

#最后，再来绘制右下角的小图。这里我们采用一种更简单方法，即直接往plt里添加新的坐标系：

```
plt.axes([0.6, 0.2, 0.25, 0.25])  
plt.plot(y[::-1], x, 'g') # 注意对y进行了逆序处理  
plt.xlabel('x')  
plt.ylabel('y')  
plt.title('title inside 2')  
plt.show()
```



快速绘图

□ 次坐标轴

有时候我们会用到次坐标轴，即在同个图上有第2个y轴存在。

```
# 数据生成:  
x = np.arange(0, 10, 0.1)  
y1 = 0.05 * x**2  
y2 = -1 * y1
```

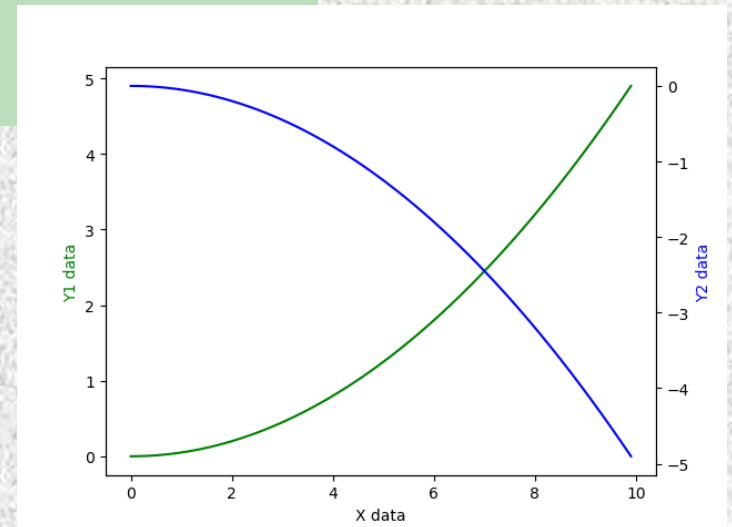
可以看到，**y2**和**y1**是互相倒置的。所以，先获取**figure**默认的坐标系 **ax1**；然后对**ax1**调用**twinx()**方法，生成如同镜面效果后的**ax2**；最后接着进行绘图，将 **y1**, **y2** 分别画在 **ax1**, **ax2** 上：

快速绘图

```
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
```

```
ax1.plot(x, y1, 'g-') # green, solid line
ax1.set_xlabel('X data')
ax1.set_ylabel('Y1 data', color='g')
```

```
ax2.plot(x, y2, 'b-') # blue
ax2.set_ylabel('Y2 data', color='b')
```



快速绘图

□ 坐标轴设定

Axis容器包括坐标轴的刻度线、刻度标签、坐标网格以及坐标轴标题等内容。刻度包括主刻度和副刻度，分别通过`get_major_ticks()`和`get_minor_ticks()`方法获得。每个刻度线都是一个`XTick`或`YTick`对象，它包括实际的刻度线和刻度标签。为了方便访问刻度线和文本，**Axis**对象提供了 `get_ticklabels()`和`get_ticklines()`方法,可以直接获得刻度标签和刻度线。下面例子进行绘图并得到当前子图的X轴对象**axis**:

```
>>> plt.plot([1,2,3],[4,5,6])
>>> plt.show()
>>> axis = plt.gca().xaxis
```


快速绘图

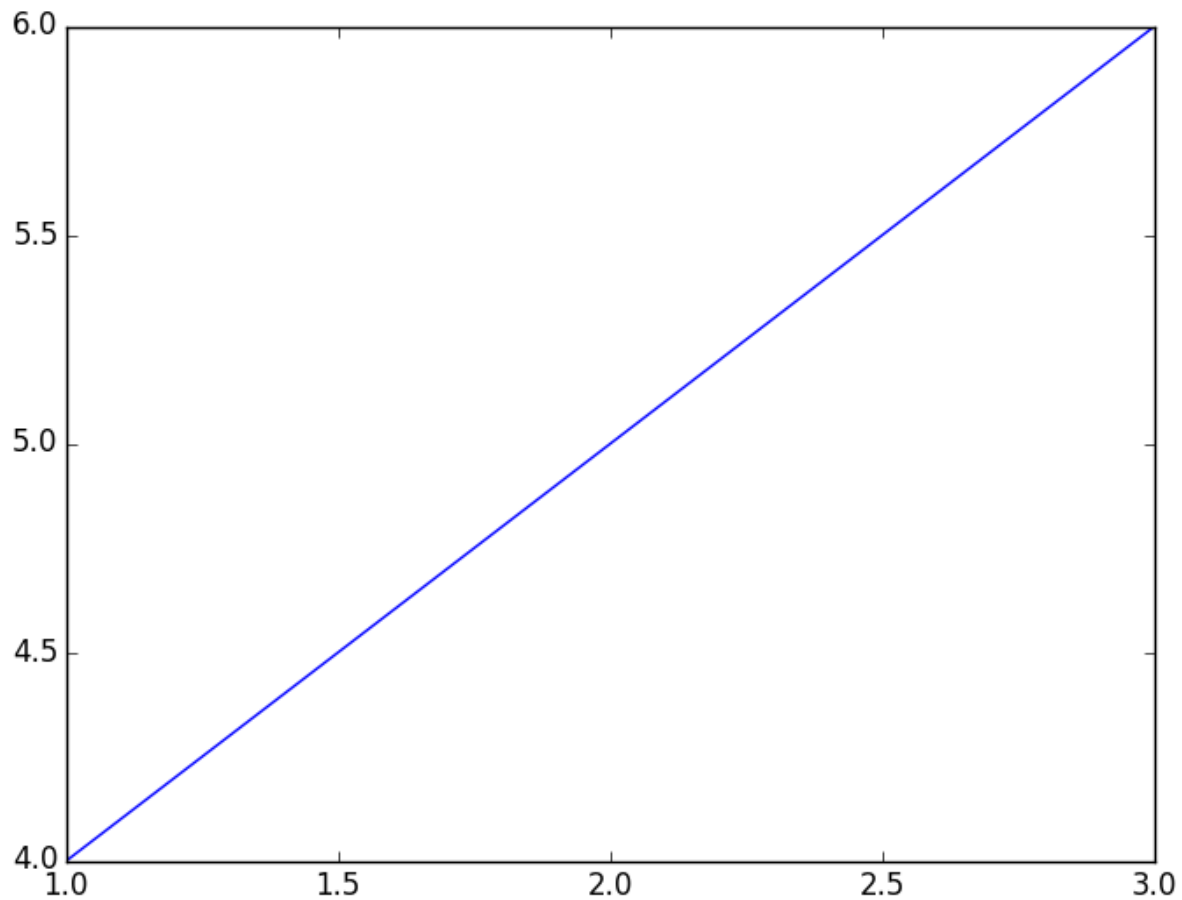
获得**axis**对象的刻度位置列表:

```
>>> axis.get_ticklocs()  
array([ 1. , 1.5, 2. , 2.5, 3. ])
```

下面获得**axis**对象的刻度标签以及标签中的文字:

```
>>> axis.get_ticklabels() # 获得刻度标签列表  
<a list of 5 Text major ticklabel objects>  
>>> [x.get_text() for x in axis.get_ticklabels()]  
# 获得刻度的文本字符串  
[u'1.0', u'1.5', u'2.0', u'2.5', u'3.0']
```

快速绘图



下面获得X轴上表示主刻度线的列表，可看到X轴上共有**10**条刻度线

```
>>> axis.get_ticklines()  
<a list of 10 Line2D ticklines objects>
```

由于没有副刻度线，因此副刻度线列表的长度为**0**：

```
>>> axis.get_ticklines(minor=True) # 获得副刻度线列表  
<a list of 0 Line2D ticklines objects>
```

使用pyplot模块中的xticks()能够完成X轴上刻度标签的配置：

```
>>> plt.xticks(fontsize=16, color="red", rotation=45)
```

快速绘图

上面的例子中副刻度线列表为空，这是因为用于计算副刻度位置的对象默认为 **NullLocator**，它不产生任何刻度线。而计算主刻度位置的对象为 **AutoLocator**，它会根据当前的缩放等配置自动计算刻度的位置。

matplotlib提供了多种配置刻度线位置的 **Locator**类，以及控制刻度标签显示的 **Formatter**类。下面的程序设置X轴的主刻度为 $\pi/4$ ，副刻度为 $\pi/20$ ，并且主刻度上的标签用数学符号显示 π 。（`matplotlib_axis_text.py`自定义坐标轴的刻度和文字）

快速绘图

与刻度定位和文本格式化相关的类都在 `matplotlib.ticker` 模块中定义，程序从中载入了两个类： `MultipleLocator`, `FuncFormatter`.

```
from matplotlib.ticker import MultipleLocator, FuncFormatter
```

```
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FuncFormatter
import numpy as np
x = np.arange(0, 4*np.pi, 0.01)
y = np.sin(x)
plt.figure(figsize=(8,4))
plt.plot(x, y)
ax = plt.gca()
```

快速绘图

程序中通过`pi_formatter()`计算出刻度值对应的刻度文本。（很繁琐）

```
def pi_formatter(x, pos):  
    m = np.round(x / (np.pi/4))  
    n = 4  
    while m!=0 and m%2==0: m, n = m//2, n//2  
    if m == 0:  
        return "0"  
    if m == 1 and n == 1:  
        return "$\pi$"  
    if n == 1:  
        return r"$%d \pi$" % m  
    if m == 1:  
        return r"$\frac{\pi}{%d}$" % n  
    return r"$\frac{%d \pi}{%d}$" % (m,n)
```

快速绘图

```
>>>X = np.linspace(0, 4*np.pi, 17, endpoint=True)
>>>X
array([ 0.          ,  0.78539816,  1.57079633,  2.35619449,
        3.14159265,  3.92699082,  4.71238898,  5.49778714,
        6.28318531,  7.06858347,  7.85398163,  8.6393798 ,
        9.42477796, 10.21017612, 10.99557429, 11.78097245,
       12.56637061])

>>>plt.xticks([ 0.          ,  0.78539816,  1.57079633,
                2.35619449,
                3.14159265,  3.92699082,  4.71238898,  5.49778714,
                6.28318531,  7.06858347,  7.85398163,  8.6393798 ,
                9.42477796, 10.21017612, 10.99557429, 11.78097245,
                12.56637061],
               [r'$0$', r'$\pi/4$', r'$\pi/2$', r'$3\pi/4$',
                r'$\pi$', r'$5\pi/4$', r'$3\pi/2$', r'$7\pi/4$',
                r'$2\pi$', r'$9\pi/4$', r'$5\pi/2$', r'$11\pi/4$',
                r'$3\pi$', r'$13\pi/4$', r'$7\pi/2$', r'$15\pi/4$', r'$4\pi$'])
# r'$ \frac{2\pi}{3} $',
```

快速绘图

以指定值的整数倍为刻度放置主、副刻度线。

```
ax.xaxis.set_major_locator( MultipleLocator(np.pi/4) )  
ax.xaxis.set_minor_locator( MultipleLocator(np.pi/20) )
```

使用指定的函数计算刻度文本，它会将刻度值和刻度的序号作为参数传递给计算刻度文本的函数。

```
ax.xaxis.set_major_formatter( FuncFormatter( pi_formatter ) )
```

```
# 设置两个坐标轴的范围  
pl.ylim(-1.5,1.5)  
pl.xlim(0, np.max(x))
```


快速绘图

```
plt.subplots_adjust(bottom = 0.15) # 设置图的底边距

plt.grid() # 开启网格

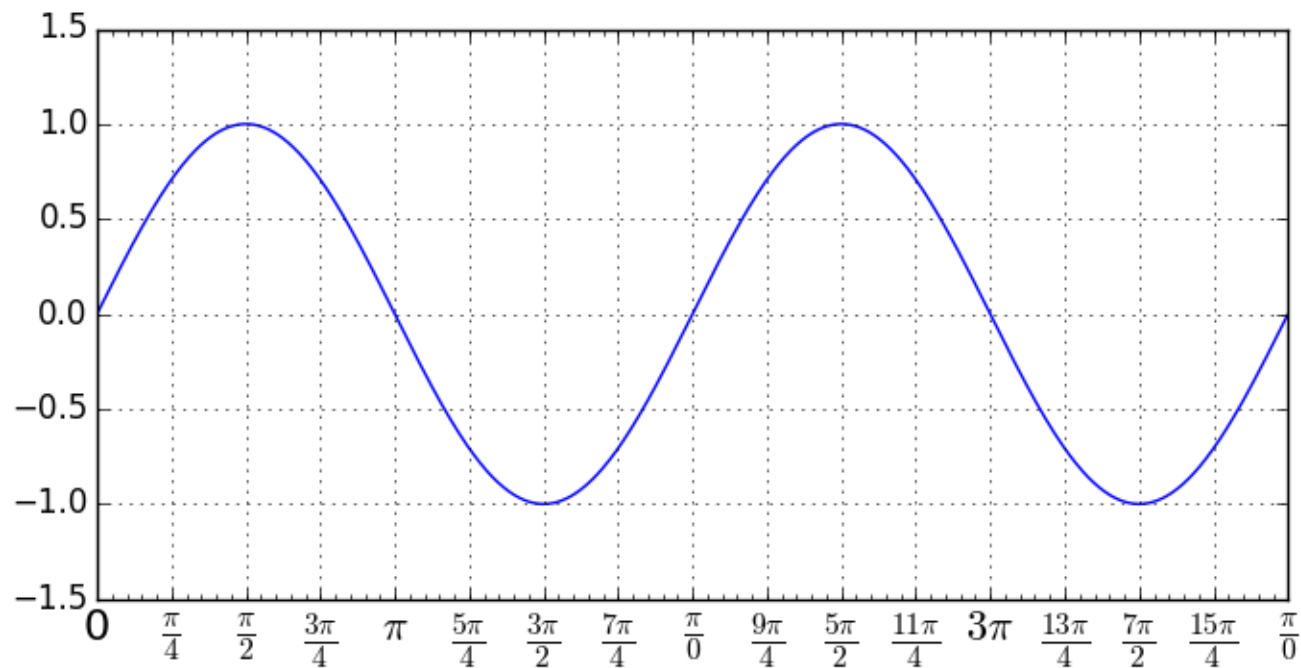
# 主刻度为 $\pi/4$ 
ax.xaxis.set_major_locator( MultipleLocator(np.pi/4) )

# 主刻度文本用pi_formatter函数计算
ax.xaxis.set_major_formatter( FuncFormatter( pi_formatter ) )

# 副刻度为 $\pi/20$ 
ax.xaxis.set_minor_locator( MultipleLocator(np.pi/20) )

# 设置刻度文本的大小
for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_fontsize(16)
plt.show()
```

快速绘图



绘图函数简介

□ 对数坐标图

前面介绍过如何使用`plot()`绘制曲线图，所绘制图表的X-Y轴坐标都是算术坐标。下面看看如何在对数坐标系中绘图。

绘制对数坐标图的函数有三个：
`semilogx()`、`semilogy()`和`loglog()`，它们分别绘制X轴为对数坐标、Y轴为对数坐标以及两个轴都为对数坐标时的图表。

绘图函数简介

下面的程序使用4种不同的坐标系绘制低通滤波器的频率响应曲线。其中，左上图为`plot()`绘制的算术坐标系，右上图为`semilogx()`绘制的X轴对数坐标系，左下图为`semilogy()`绘制的Y轴对数坐标系，右下图为`loglog()`绘制的双对数坐标系。使用双对数坐标系表示的频率响应曲线通常被称为波特图。(matplotlib_log.py)

```
import numpy as np
import matplotlib.pyplot as plt

w = np.linspace(0.1, 1000, 1000)
p = np.abs(1/(1+0.1j*w)) # 计算低通滤波器的频率响应
```


绘图函数简介



```
plt.subplot(221)
plt.plot(w, p, linewidth=2)
plt.ylim(0,1.5)
```

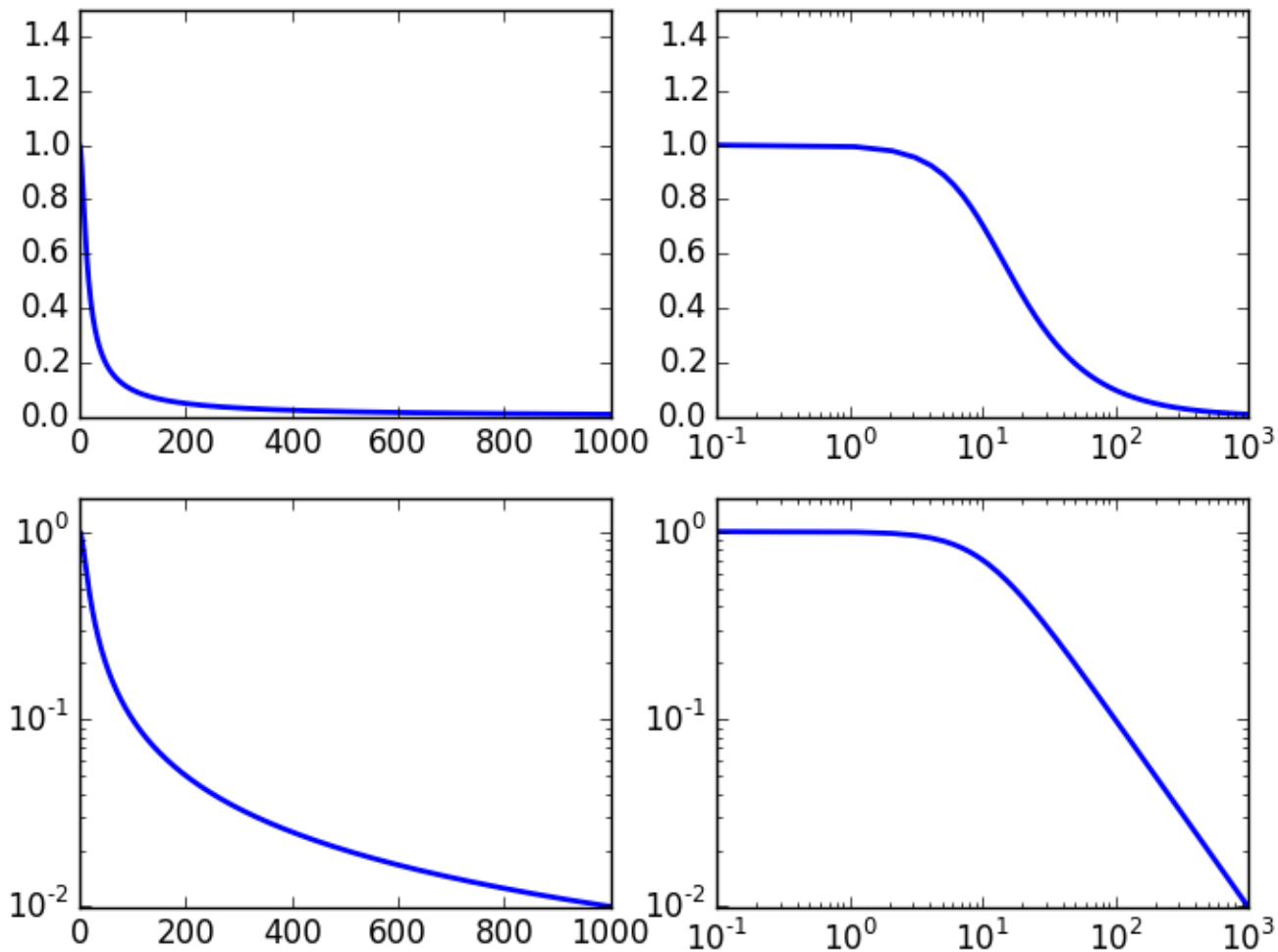
```
plt.subplot(222)
plt.semilogx(w, p, linewidth=2)
plt.ylim(0,1.5)
```

```
plt.subplot(223)
plt.semilogy(w, p, linewidth=2)
plt.ylim(0,1.5)
```

```
plt.subplot(224)
plt.loglog(w, p, linewidth=2)
plt.ylim(0,1.5)
```

```
plt.show()
```

绘图函数简介



绘图函数简介

□ 极坐标图

极坐标系是和笛卡尔(X-Y)坐标系完全不同的坐标系，极坐标系中的点由一个夹角和一段相对中心点的距离来表示。下面的程序绘制极坐标图，(matplotlib_polar.py)。

```
import numpy as np
import matplotlib.pyplot as plt

theta = np.arange(0, 2*np.pi, 0.02)
```

绘图函数简介

```
plt.subplot(121, polar=True)
plt.plot(theta, 1.6*np.ones_like(theta), linewidth=2)
plt.plot(3*theta, theta/3, "--", linewidth=2)
```

程序中调用**subplot()**创建子图时通过设**polar**参数为**True**,创建一个极坐标子图。然后调用**plot()**在极坐标子图中绘图。也可以使用**polar()**直接创建极坐标子图并在其中绘制曲线。

```
plt.polar(theta, 1.6*np.ones_like(theta), linewidth=2)
```

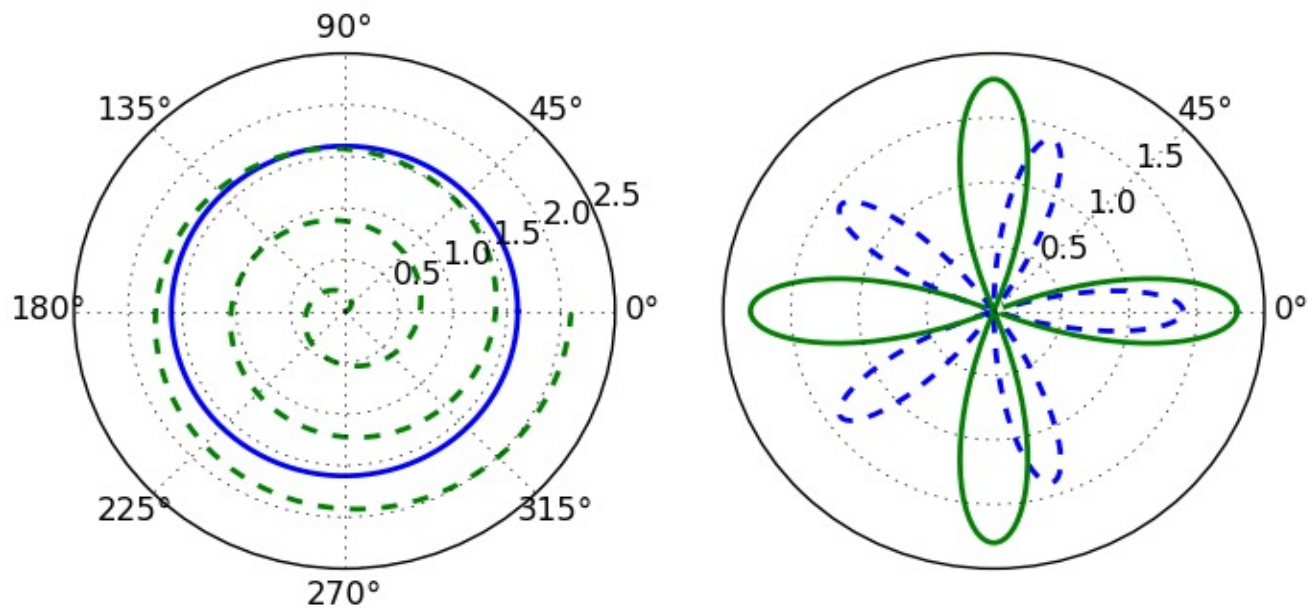

绘图函数简介

```
plt.subplot(122, polar=True)
plt.plot(theta, 1.4*np.cos(5*theta), "--", linewidth=2)
plt.plot(theta, 1.8*np.cos(4*theta), linewidth=2)
plt.rgrids(np.arange(0.5, 2, 0.5), angle=45)
plt.thetagrids([0, 45])

plt.show()
```

rgrids()设置同心圆栅格的半径大小和文字标注的角度。因此右图中的虚线圆圈有三个，半径分别为**0.5**、**1.0**和**1.5**，这些文字沿着**45°**线排列。**Thetagrids()**设置放射线栅格的角度，因此右图中只有两条放射线，角度分别为**0°**和**45°**。

绘图函数简介



绘图函数简介

□ 柱状图

柱状图用其每根柱子的长度表示值的大小，它们通常用来比较两组或多组值。下面的程序从文件中读入中国人口的年龄分布数据,并使用柱状图比较男性和女性的年龄分布。（**matplotlib_bar.py** 绘制比较男女人口的年龄分布图）

```
import numpy as np
import matplotlib.pyplot as plt
```

绘图函数简介

```
data = np.loadtxt("china_population.txt")  
width = (data[1,0] - data[0,0])*0.4
```

读入的数据中，第**0**列为年龄，它将作为柱状图的横坐标。首先计算柱状图中每根柱子的宽度，因为要在每个年龄段上绘制两根柱子，因此柱子的宽度应该小于年龄段的二分之一。这里以年龄段的**0.4**倍作为柱子的宽度。

绘图函数简介

```
plt.figure(figsize=(8,5))
```

```
plt.bar(data[:,0]-width/2, data[:,1]/1e7, width,  
color="b", label=u"男")
```

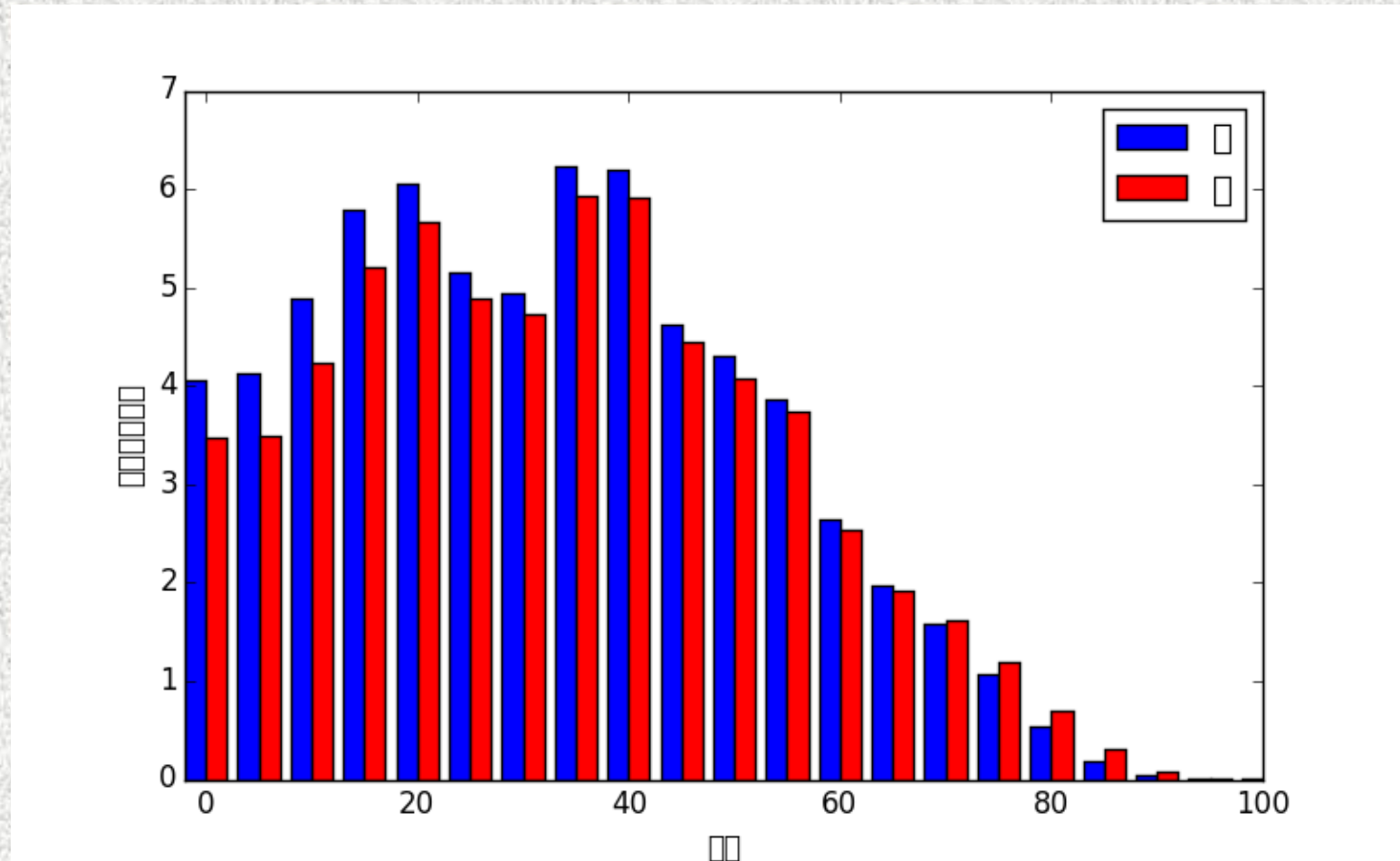
调用**bar()**绘制男性人口分布的柱状图。它的第一个参数为每根柱子中心的横坐标，为了让男性和女性的柱子以年龄刻度为中心，这里让每根柱子左侧的横坐标为“年龄减去柱子一半的宽度”。**Bar()**的第二个参数为每根柱子的高度，第三个参数指定所有柱子的宽度。当第三个参数为序列时，可以为每根柱子指定宽度。

绘图函数简介

```
plt.bar(data[:,0]+width/2, data[:,2]/1e7, width,  
color="r", label=u"女")  
plt.xlim(-width, 100)  
plt.xlabel(u"年龄")  
plt.ylabel(u"人口（千万）")  
plt.legend()  
  
plt.show()
```

绘制女性人口分布的柱状图，这里以年龄为柱子的中心横坐标，因此女性和男性的人口分布图以年龄刻度为中心。由于**bar()**不自动修改颜色，因此程序中通过**color**参数设置两个柱状图的颜色。

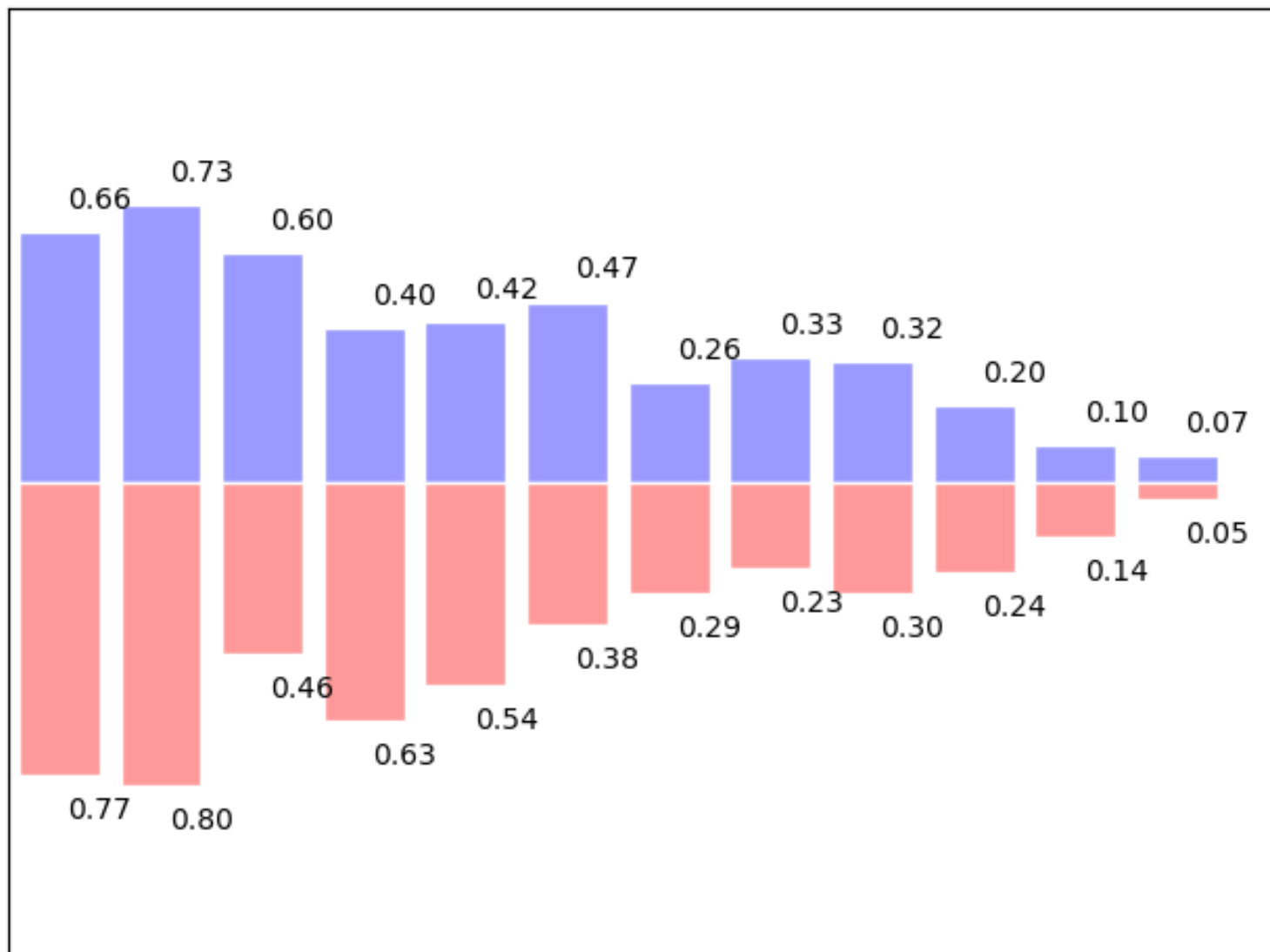
绘图函数简介



绘图函数简介

```
import numpy as np
import matplotlib.pyplot as plt
n = 12
X = np.arange(n)
Y1 = (1-X/float(n)) * np.random.uniform(0.5,1.0,n)
Y2 = (1-X/float(n)) * np.random.uniform(0.5,1.0,n)
plt.axes([0.025,0.025,0.95,0.95])
plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')
for x,y in zip(X,Y1):
    plt.text(x+0.04, y+0.05, '%.2f' % y, ha='center', va=
'bottom')
for x,y in zip(X,Y2):
    plt.text(x+0.04, -y-0.05, '%.2f' % y, ha='center', va= 'top')
plt.xlim(-.5,n), plt.xticks([])
plt.ylim(-1.25,+1.25), plt.yticks([])
# savefig('bar_ex.png', dpi=48)
plt.show()
```


绘图函数简介



绘图函数简介

□ 散列图

使用`plot()`绘图时，如果指定样式参数为仅绘制数据点，那么所绘制的就是一幅散列图。
例如：

```
>>> plt.plot(np.random.random(100),  
             np.random.random(100), "o")
```

但是这种方法所绘制的点无法单独指定颜色和大小。而`scatter()`所绘制的散列图却可以指定每个点的颜色和大小。下面的程序演示`scatter()`的用法 (`matplotlib_scatter.py`)。

绘图函数简介

```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))
x = np.random.random(100)
y = np.random.random(100)
plt.scatter(x, y, s=x*1000, c=y, marker=(5, 1),
            alpha=0.8, lw=2, facecolors="none")
plt.xlim(0,1)
plt.ylim(0,1)

plt.show()
```

scatter()的前两个参数是数组，分别指定每个点的X轴和Y轴的坐标。**s**参数指定点的大小，值和点的面积成正比。它可以是一个数，

绘图函数简介

指定所有点的大小；也可以是数组，分别对每个点指定大小。

c参数指定每个点的颜色，可以是数值或数组。这里使用一维数组为每个点指定了一个数值。通过颜色映射表，每个数值都会与一个颜色相对应。默认的颜色映射表中蓝色与最小值对应，红色与最大值对应。当**c**参数是形状为(N,3)或(N,4)的二维数组时，则直接表示每个点的RGB颜色。

marker参数设置点的形状，可以是个表示形状的字符串，也可以是表示多边形的两个元素的元组，第一个元素表示多边形的边数，

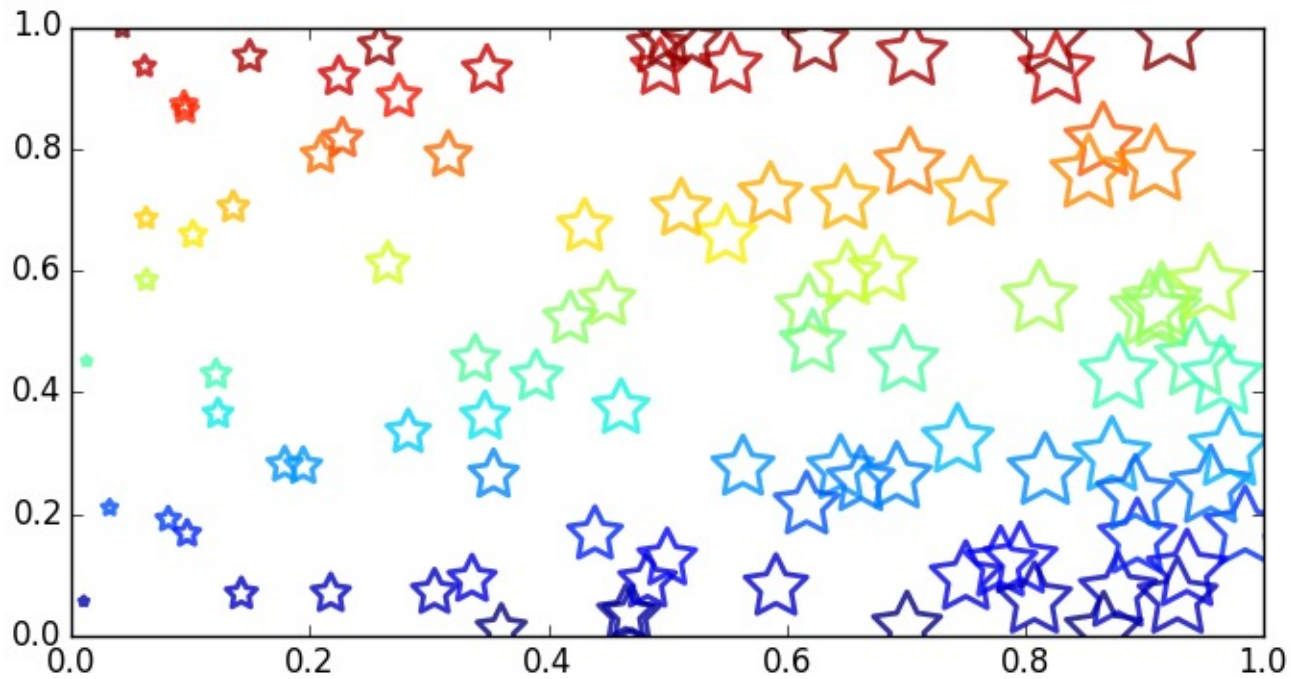
绘图函数简介

第二个元素表示多边形的样式，取值范围为0、1、2、3。0表示多边形，1表示星形，2表示放射形，3表示忽略边数而显示为圆形。

alpha参数设置点的透明度，通过**lw**参数设置线宽。**facecolors**参数为“none”时，表示散列点没有填充色。

plt.plot 与 **plt.scatter** 除了特征上的差异之外，当数据变大到几千个散点时，**plt.plot** 的效率将大大高于**plt.scatter**。这是由于 **plt.scatter** 会对每个散点进行单独的大小与颜色的渲染，因此渲染器会消耗更多的资源。而在 **plt.plot** 中，散点基本都彼此复制。

绘图函数简介



绘图函数简介


□ 图像

`imread()`和`imshow()`提供了简单的图像载入和显示功能.

```
>>>img = plt.imread("lena.jpg")
```

`imread()`可以从图像文件读入数据,得到一个表示图像的NumPy数组。它的第一个参数是文件名或文件对象, **format**参数指定图像类型, 如果省略, 就由文件的扩展名决定图像类型。对于灰度图像, 它返回一个形状为(M, N)的数组; 对于彩色图像, 返回形状为(M, N, C)的数组。其中, M为图像的高度, N为图像的宽度, C为3或4, 表示图像的通道数。

绘图函数简介

下面的程序从“lena.jpg”中读入图像数据，得到的数组是一个形状为(393,512,3)的单字节无符号整数数组。这是因为通常使用的图像都是采用单字节分别保存每个像素的红、绿、蓝三个通道的分量：

```
>>> img = plt.imread("lena.jpg")
```

```
>>> img.shape  
(393L, 512L, 3L)
```

```
>>> img.dtype  
dtype('uint8')
```


绘图函数简介


`imshow()`可以用来显示`imread()`返回的数组。如果数组是表示多通道图像的三维数组，那么每个像素的颜色由各个通道的值决定：

```
>>> plt.imshow(img)
```

请注意，从JPG图像中读入的数据是上下颠倒的，为了正常显示图像，可以将数组的第0轴反转，或者设置`imshow()`的`origin`参数为“lower”，从而让所显示图表的原点在左下角：

```
>>> plt.imshow(img[::-1]) # 反转图像数组的第0轴  
#or  
>>> plt.imshow(img, origin="lower") # 让图表的  
原点在下半角                                upper
```

绘图函数简介

如果三维数组的元素类型为浮点数，那么元素的取值范围为0.0到1.0,与颜色值0到255对应。超出这个范围可能会出现颜色异常的像素。下面的例子将数组转换为浮点数组并用 `imshow()`进行显示：

```
>>> img = img[: :-1]
>>> plt.imshow(img*1.0) #取值范围为0.0到255.0的浮点数组，不能正确显示颜色
>>> plt.imshow(img/255.0) #取值范围为0.0到1.0的浮点数组，能正确显示颜色
>>> plt.imshow(np.clip(img/200.0, 0, 1)) # 使用clip()限制取值范围，整个图像变亮
```

绘图函数简介

如果`imshow()`的参数是二维数组，就使用颜色映射表决定每个像素的颜色。下面显示图像中的红色通道：

```
>>> plt.imshow(img[:, :, 0])
```

显示效果比较吓人，因为默认的图像映射将最小值映射为蓝色、将最大值映射为红色。可以使用`colorbar()`将颜色映射表在图表中显示出来：

```
>>> plt.colorbar()
```

绘图函数简介

通过`imshow()`的`cmap`参数可以修改显示图像时所采用的颜色映射表。颜色映射表是一个 **ColorMap**对象，`matplotlib`中已经预先定义好了很多颜色映射表。

(`matplotlib_imshow.py`)

下面使用名为**copper**的颜色映射表显示图像的红色通道，很有老照片的味道：

```
>>>plt.imshow(img[:, :, 0], cmap=cm.copper)
```


绘图函数简介

```
[ import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm

plt.subplots_adjust(0,0,1,1,0.05,0.05)
plt.subplot(331)
img = plt.imread("lena.jpg")
plt.imshow(img)

plt.subplot(332)
plt.imshow(img[::-1])

plt.subplot(333)
plt.imshow(img, origin="lower")

img = img[::-1]
plt.subplot(334)
plt.imshow(img*1.0)
```

绘图函数简介

```
[ plt.subplot(335)  
  plt.imshow(img/255.0)
```

```
plt.subplot(336)  
plt.imshow(np.clip(img/200.0, 0, 1))
```

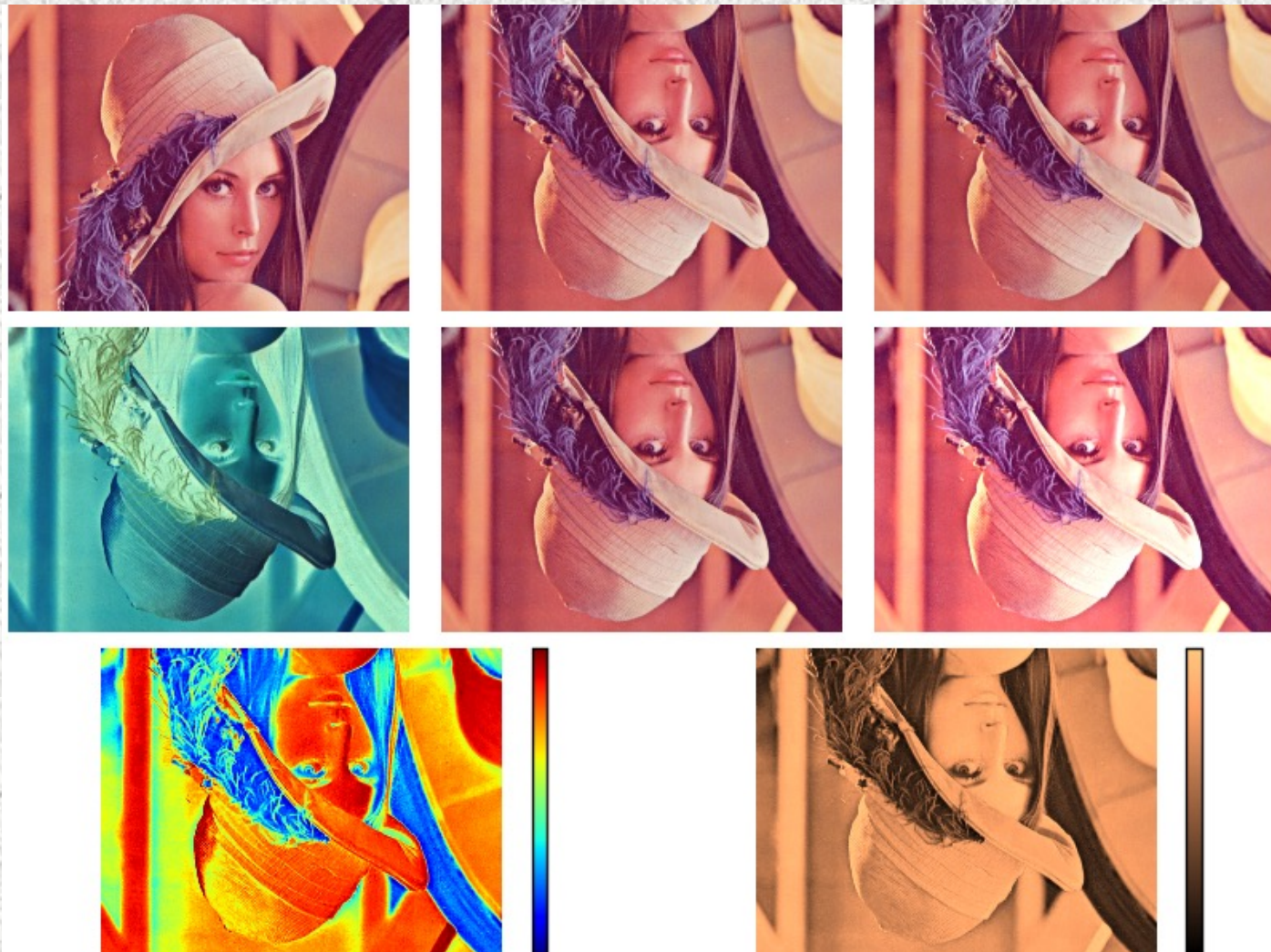
```
plt.subplot(325)  
plt.imshow(img[:, :, 0])  
plt.colorbar()
```

```
plt.subplot(326)  
plt.imshow(img[:, :, 0], cmap=cm.copper)  
plt.colorbar()
```

```
for ax in plt.gcf().axes:  
    ax.set_axis_off()  
    #ax.set_axis_off()
```

```
plt.show()
```

绘图函数简介



绘图函数简介

还可以使用`imshow()`显示任意的二维数据，例如下面的程序使用图像直观地显示了二元函数 $f(x, y) = xe^{-x^2 - y^2}$ 。

(`matplotlib_2dfunc.py` 使用`imshow()`可视化二元函数)

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

```
y, x = np.ogrid[-2:2:200j, -2:2:200j]
z = x * np.exp( - x**2 - y**2)
```

```
extent = [np.min(x), np.max(x), np.min(y), np.max(y)]
```


绘图函数简介

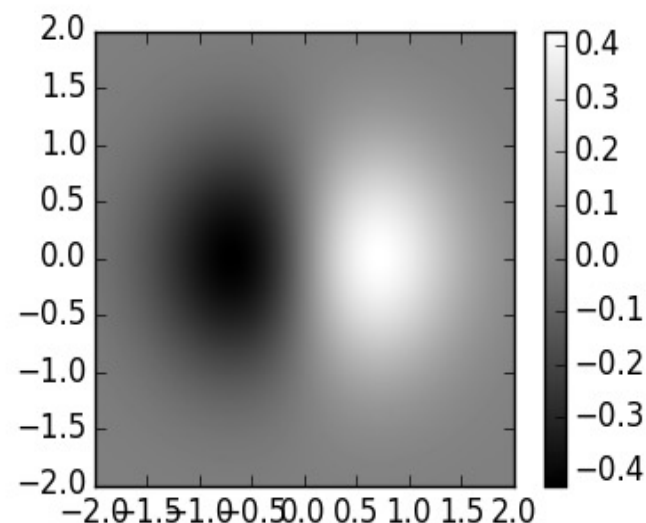
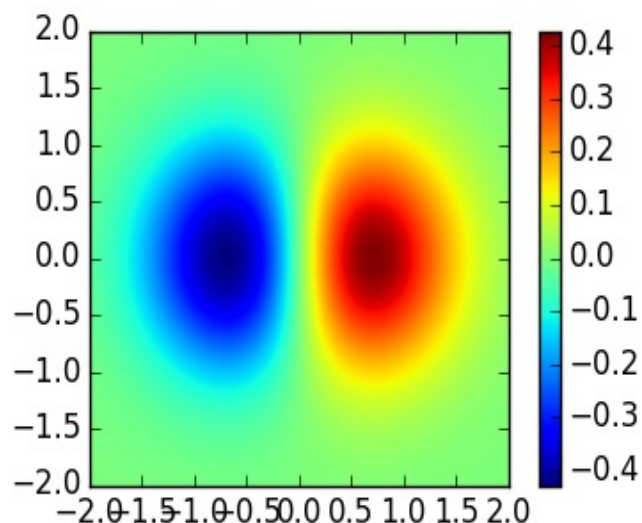
```
plt.figure(figsize=(10,3))
plt.subplot(121)
plt.imshow(z, extent=extent, origin="lower")
plt.colorbar()
plt.subplot(122)
plt.imshow(z, extent=extent, cmap=cm.gray,
origin="lower")
plt.colorbar()

plt.show()
```

首先通过数组的广播功能计算出表示函数值的二维数组 \mathbf{Z} ,注意它的第0轴表示Y轴、第1轴表示X轴。然后将X、Y轴的取值范围保存到 \mathbf{extent} 列表中。

绘图函数简介

将`extent`列表传递给 `imshow()` 的 `extent` 参数，这样一来，图表的X、Y轴的刻度标签将使用 `extent` 列表所指定的范围。



绘图函数简介

□ 等值线图

还可以使用等值线图表示二元函数。所谓等值线，是指由函数值相等的各点连成的平滑曲线。等值线可以直观地表示二元函数值的变化趋势，例如等值线密集的地方表示函数值在此处的变化较大。**matplotlib**中可以使用 **contour()** 和 **contourf()** 描绘等值线，它们的区别是：**contourf()** 所得到的是带填充效果的等值线。（**matplotlib_contour.py** 用 **contour** 和 **contourf** 描绘等值线图）

绘图函数简介

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

y, x = np.ogrid[-2:2:200j, -3:3:300j]
z = x * np.exp(- x**2 - y**2)

extent = [np.min(x), np.max(x), np.min(y), np.max(y)]

plt.figure(figsize=(10,4))
plt.subplot(121)
cs = plt.contour(z, 10, extent=extent)
plt.clabel(cs)
plt.subplot(122)
plt.contourf(x.reshape(-1), y.reshape(-1), z, 20) #格式要求: x,y一维或x, y为网格 (m, n)
plt.show()
```

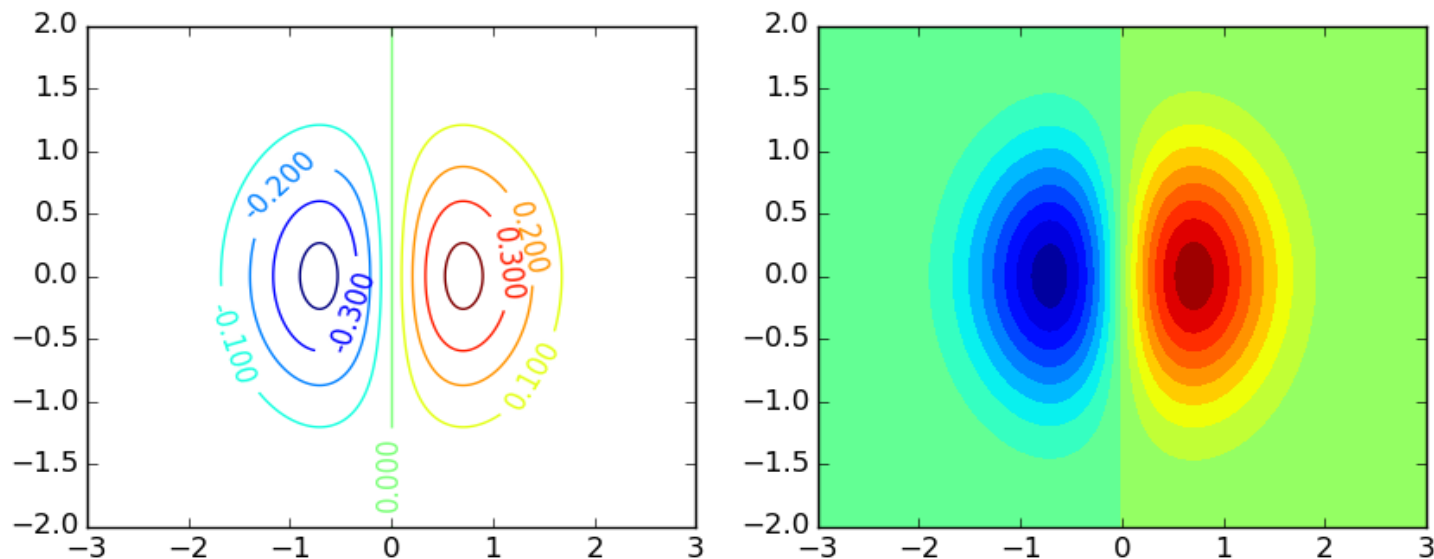

绘图函数简介

为了更清楚地区分X轴和Y轴，这里让它们的取值范围和等分次数均不相同。这样得到的数组`z`的形状为`(200, 300)`，它的第0轴对应Y轴、第1轴对应X轴。

调用`contour()`绘制数组`z`的等值线图，第二个参数为10，表示将整个函数的取值范围等分为10个区间，即显示的等值线图中将有9条等值线。和`imshow()`一样，可以使用`extent`参数指定等值线图的X轴和Y轴的数据范围。`contour()`所返回的是一个`QuadContourSet`对象，将它传递给`clabel()`，为其中的等值线标上对应的值。

绘图函数简介

调用`contourf()`, 绘制将取值范围等分为20份、带填充效果的等值线图。这里演示了另外一种设置X、Y轴取值范围的方法。它的前两个参数分别是计算数组`z`时所使用的X轴和Y轴上的取样点, 这两个数组必须是一维的。

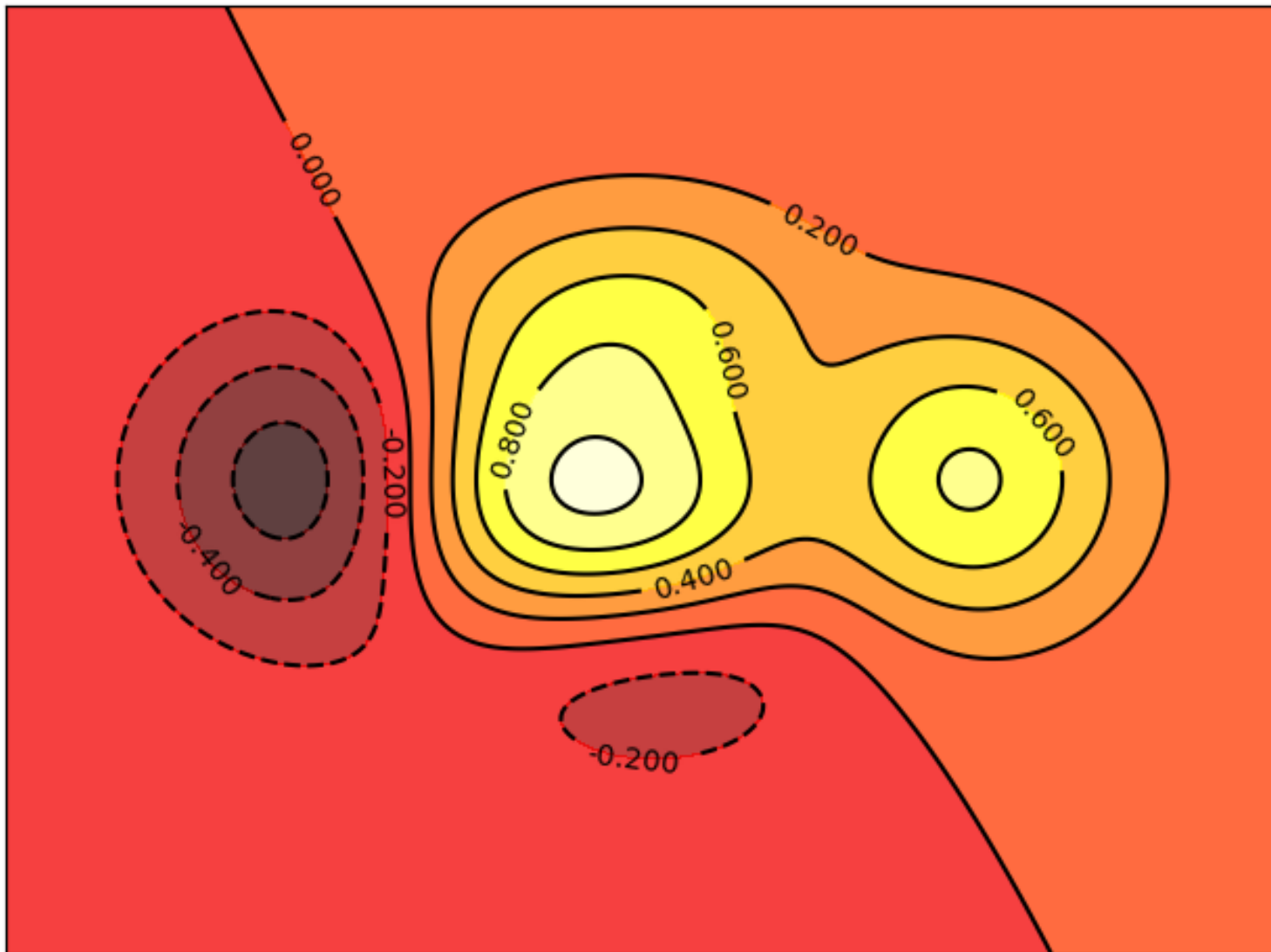


绘图函数简介

```
import numpy as np
import matplotlib.pyplot as plt
def f(x,y):
    return (1-x/2+x**5+y**3)*np.exp(-x**2-y**2)

n = 256
x = np.linspace(-3,3,n)
y = np.linspace(-3,3,n)
X,Y = np.meshgrid(x,y)
plt.axes([0.025,0.025,0.95,0.95])
plt.contourf(X, Y, f(X,Y), 8, alpha=.75, cmap=plt.cm.hot)
# plt.contourf(x, y, f(X,Y), 8, alpha=.75, cmap=plt.cm.hot)
C = plt.contour(X, Y, f(X,Y), 8, colors='black', linewidth=.5)
plt.clabel(C, inline=1, fontsize=10)
plt.xticks([]), plt.yticks([])
# savefig('../figures/contour_ex.png',dpi=48)
plt.show()
```

绘图函数简介



绘图函数简介

还可以使用等值线绘制隐函数曲线。显然，无法像绘制一般函数那样，先创建一个等差数组表示变量的取值点，然后计算出数组中每个 x 所对应的 y 值。可以使用等值线解决这个问题，显然隐函数的曲线就是值等于0的那条等值线。下面的程序绘制函数 $f(x, y) = (x^2 + y^2)^4 - (x^2 - y^2)^2$ 在 $f(x, y) = 0$ 和 $f(x, y) - 0.1 = 0$ 时的曲线。
(matplotlib_implicit_func.py)

```
import numpy as np
import matplotlib.pyplot as plt

y, x = np.ogrid[-1.5:1.5:200j, -1.5:1.5:200j]
f = (x**2 + y**2)**4 - (x**2 - y**2)**2
```

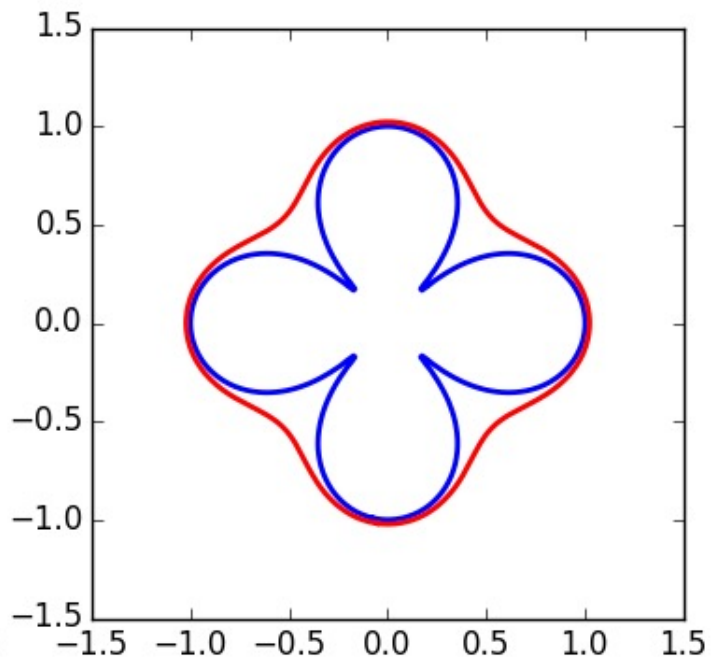
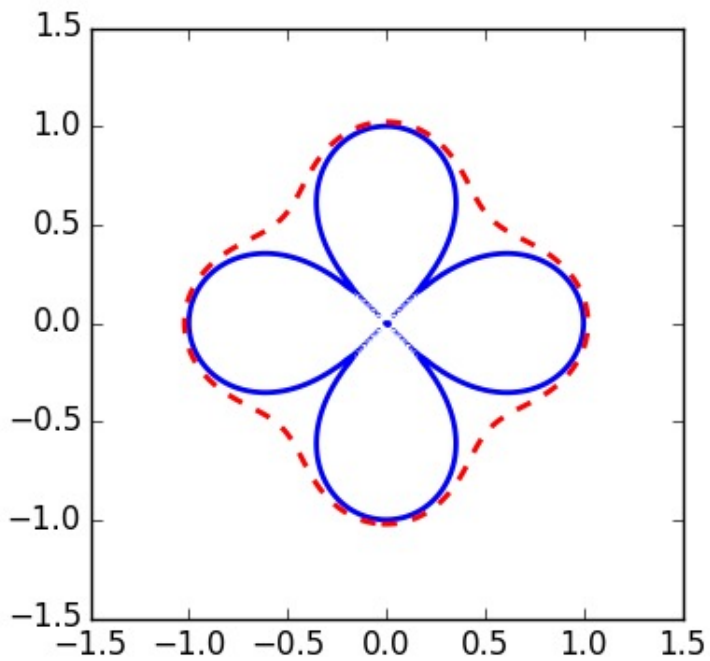
绘图函数简介

```
plt.figure(figsize=(9,4))
plt.subplot(121)
extent = [np.min(x), np.max(x), np.min(y), np.max(y)]
cs = plt.contour(f, extent=extent, levels=[0, 0.1],
colors=["b", "r"], linestyles=["solid", "dashed"],
linewidths=[2, 2])

plt.subplot(122)
for c in cs.collections:
    data = c.get_paths()[0].vertices
    plt.plot(data[:,0], data[:,1],
    color=c.get_color()[0], linewidth=c.get_linewidth()[0])

plt.show()
```

绘图函数简介



绘图函数简介

在调用`contour()`绘制等值线时，可以通过`levels`参数指定所绘制等值线对应的函数值，这里设置`levels`参数为`[0, 0.1]`，因此最终将绘制两条等值线。

观察图会发现，表示隐函数 $f(x)=0$ 蓝色实线并不是完全连续的，在图的中间部分它由许多孤立的小段构成。因为等值线在 origin 附近无限靠近，因此无论对函数 f 的取值空间如何进行细分，总是会有无法分开的地方，最终造成了图中的那些孤立的细小区域。而表示隐函数 $f(x,y)-0.1=0$ 的红色虚线则是闭合且连续的。

绘图函数简介

可以通过`contour()`返回的对象获得等值线上每点的数据，下面在IPython中观察变量`cs`，它是一个 `QuadContourSet` 对象：

```
>>> run matplotlib_implicit_func.py
>>> cs
<matplotlib.contour.QuadContourSet instance at 0x0A348E90>
```

`cs`对象的`collections`属性是一个等值线列表，每条等值线用一个`LineCollection`对象表示：

```
>>> cs.collections
<a list of 2 mcoll.LineCollection objects>
```

绘图函数简介

每个**LineCollection**对象都有它自己的颜色、线型、线宽等属性，注意这些属性所获得的结果外面还有一层封装，要获得其第0个元素才是真正的配置：

```
>>> c.get_color()[0]
array([ 1.,  0.,  0.,  1.])
>>> c.get_linewidth()[0]
2
```

由类名可知，**LineCollection**对象是一组曲线的集合，因此它可以表示像蓝色实线那样由多条线构成的等值线。它的**get_paths()**方法获得构成等值线的所有路径，本例中蓝色实线

绘图函数简介

所表示的等值线由**42**条路径构成：

```
>>> len(cs.collections[0].get_paths())  
42
```

路径是一个**Path**对象，通过它的**vertices**属性可以获得路径上所有点的坐标：

```
>>> path = cs.collections[0].get_paths()[0]  
>>> type(path)  
<class 'matplotlib.path.Path'  
>>> path.vertices  
array([[ -0.08291457, -0.98938936],  
       [ -0.09039269, -0.98743719],  
       ...,  
       [ -0.08291457, -0.98938936]])
```

绘图函数简介

下面的程序从等值线集合**cs**中找到表示等值线的路径，并使用**plot()**将其绘制出来。

```
plt.subplot(122)
for c in cs.collections:
    data = c.get_paths()[0].vertices
    plt.plot(data[:,0], data[:,1],
             color=c.get_color()[0], linewidth=c.get_linewidth()[0])
```


绘图函数简介

□ 饼图

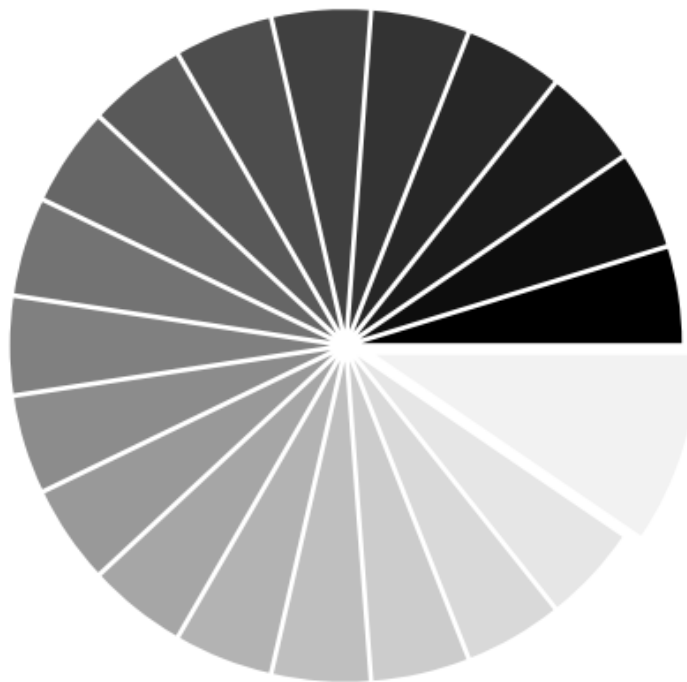
绘制简单饼图：需要改变 Z 。

```
from pylab import *  
n = 20  
Z = np.random.uniform(0,1,n)  
pie(Z), show()
```

```
import numpy as np  
import matplotlib.pyplot as plt  
n = 20  
Z = np.ones(n)  
Z[-1] *= 2  
plt.axes([0.025,0.025,0.95,0.95])  
plt.pie(Z, explode=Z*.05, colors = ['%f' % (i/float(n)) for i  
in range(n)])
```

绘图函数简介

```
plt.gca().set_aspect('equal') #让他生成一个正圆  
# 不是椭圆  
plt.xticks([]), plt.yticks([])  
# savefig('../figures/pie_ex.png',dpi=48)  
plt.show()
```



绘图函数简介

❑ 误差线图

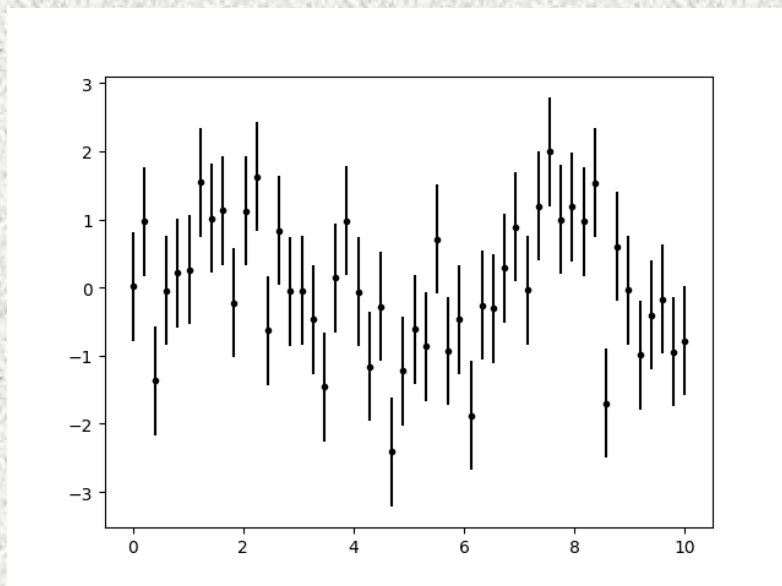
基本误差线（**errorbar**）可以通过一个 **Matplotlib** 函数来创建。

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 50)
dy = 0.8
y = np.sin(x) + dy * np.random.randn(50)

plt.errorbar(x, y, yerr=dy, fmt='.k')
plt.show()
```

其中，**fmt** 是一种控制线条和点的外观的代码格式。

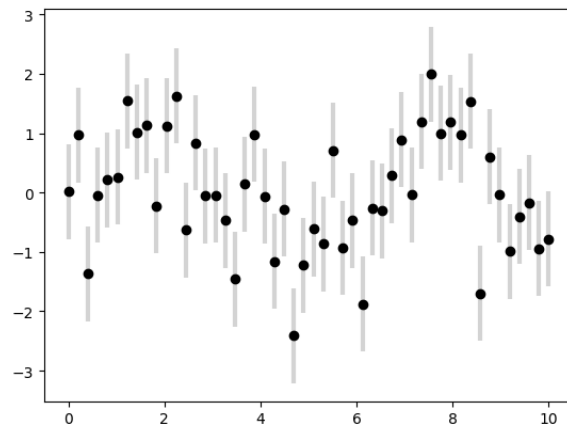


除了基本选项之外，**errorbar** 还有许多改善结果的选项。通过这些额外的选项，可以轻松自定义误差线图形的绘画风格。经验是，让误差线的颜色比数据点的颜色浅一点效果会非常好，尤其是在那些比较密集的图形中：

绘图函数简介

```
plt.errorbar(x, y, yerr=dy, fmt='o', color='black',  
ecolor='lightgray', elinewidth=3, capsize=0);
```

除了这些选项之外，还可以设置水平方向的误差线（**xerr**）、单侧误差线（**one-sided errorbar**），以及其他形式的误差线。参考 **plt.errorbar** 的程序文档。



绘图函数简介

□ 三维绘图

`mpl_toolkits.mplot3d`模块在`matplotlib`基础上提供了三维绘图的功能。由于它使用`matplotlib`的二维绘图功能来实现三维图形的绘制工作，因此绘图速度有限，不适合用于大规模数据的三维绘图。如果需要更复杂的三维数据可视化功能，可使用`Seaborn`、`Mayavi`。

绘图函数简介

三维数据点与线

最基本的三维图是由 (x, y, z) 三维坐标点构成的线图与散点图。与前面介绍的普通二维图类似，可以用 `plot3D` 与 `scatter3D` 函数来创建它们。下面来画一个三角螺旋线，在线上随机分布一些散点：

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
```

导入 `Matplotlib` 自带的 `mplot3d` 工具箱来画三维图。

绘图函数简介

导入这个子模块之后，就可以在创建任意一个普通坐标轴的过程中加入**projection='3d'**关键字，从而创建一个三维坐标轴：

```
fig = plt.figure()
ax = plt.axes(projection='3d')

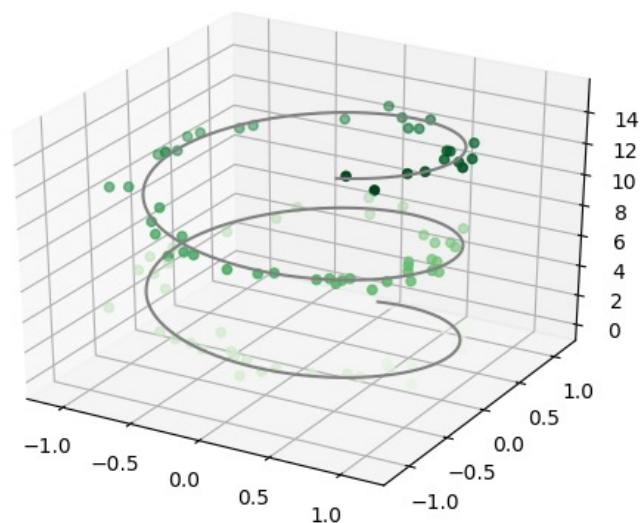
# 三维线的数据
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'gray')

# 三维散点的数据
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.randn(100)
```


绘图函数简介

```
ydata = np.cos(zdata) + 0.1 * np.random.randn(100)
ax.scatter3D(xdata, ydata, zdata, c=zdata,
             cmap='Greens')
#ax.scatter3D(xdata, ydata, zdata, c=zdata,
#             cmap='summer')
plt.show()
```

默认情况下，散点会自动改变透明度，以在平面上呈现出立体感。



绘图函数简介

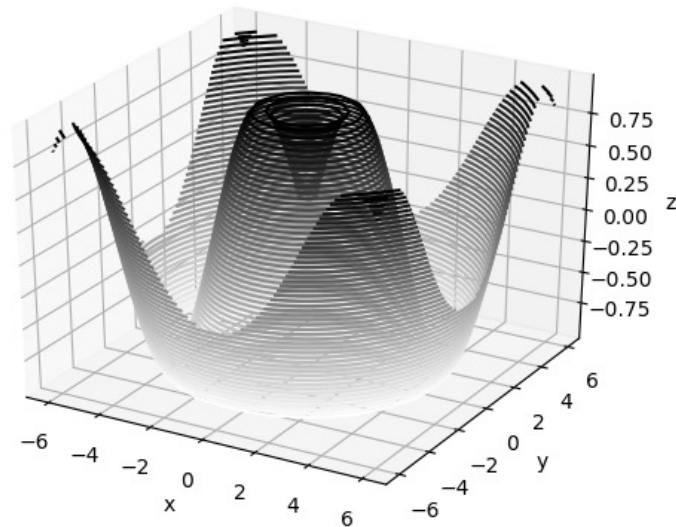
三维等高线图

与二维 `contour` 图形一样，`contour3D` 要求所有数据都是二维网格数据的形式，并且由函数计算 z 轴数值。下面演示一个用三维正弦函数画的三维等高线图：

```
def f(x, y):  
    return np.sin(np.sqrt(x ** 2 + y ** 2))  
  
x = np.linspace(-6, 6, 30)  
y = np.linspace(-6, 6, 30)  
  
X, Y = np.meshgrid(x, y)  
Z = f(X, Y)
```

绘图函数简介

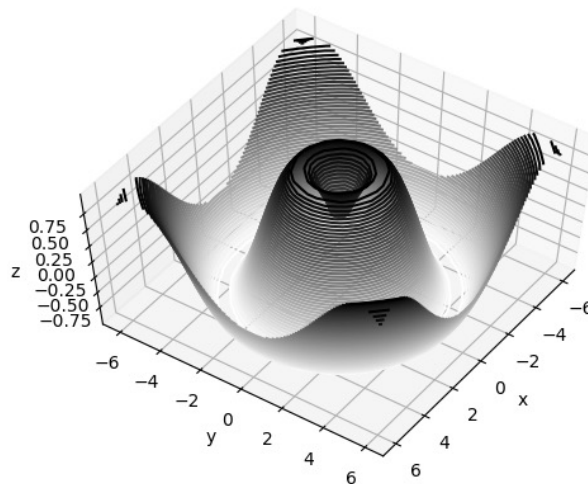
```
fig = plt.figure()  
ax = plt.axes(projection='3d')  
ax.contour3D(X, Y, Z, 50, cmap='binary')  
ax.set_xlabel('x')  
ax.set_ylabel('y')  
ax.set_zlabel('z')  
plt.show()
```



绘图函数简介

默认的初始观察角度有时不是最优的，**view_init** 可以调整观察角度与方位角。把俯仰角调整为 **60** 度（这里的 **60** 度是 **x-y** 平面的旋转角度），方位角调整为 **35** 度（就是绕 **z** 轴顺时针旋转 **35** 度）：

```
ax.view_init(60, 35)  
fig
```



绘图函数简介

线框图和曲面图

线框图:

```
import numpy as np
import mpl_toolkits.mplot3d
import matplotlib.pyplot as plt

x, y = np.mgrid[-2:2:20j, -2:2:20j]
z = x * np.exp(- x**2 - y**2)
ax = plt.subplot(111, projection='3d')
ax.plot_wireframe(x, y, z, color='black')
ax.set_title('wireframe')

plt.show()
```

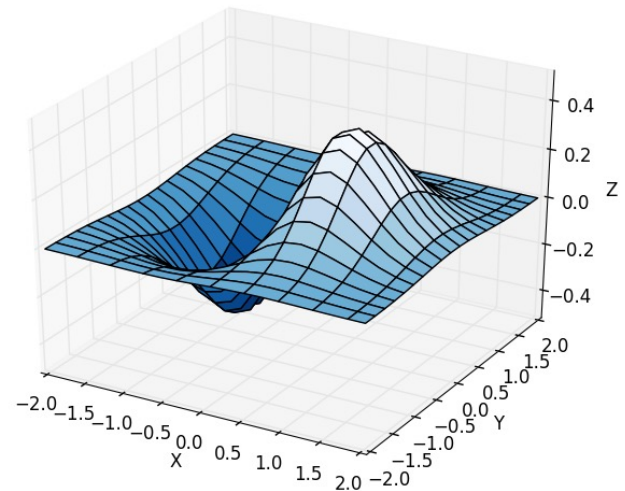
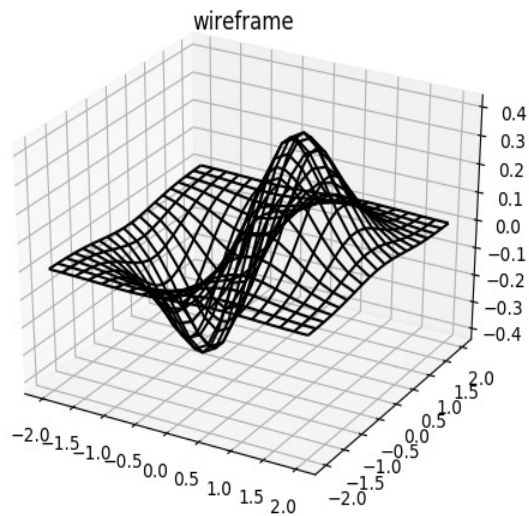
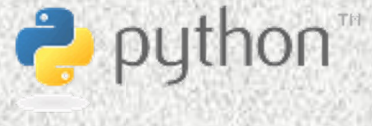
(matplotlib_surface1.py 绘制三维曲面)

```
import numpy as np
import mpl_toolkits.mplot3d
import matplotlib.pyplot as plt

x, y = np.mgrid[-2:2:20j, -2:2:20j]
z = x * np.exp( - x**2 - y**2)

ax = plt.subplot(111, projection='3d')
ax.plot_surface(x, y, z, rstride=2, cstride=1, cmap =
plt.cm.Blues_r)
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.show()
```

绘图函数简介



绘图函数简介

首先载入`mplot3d`模块，`matplotlib`中与三维绘图相关的功能均在此模块中定义。使用`mgrid`创建X-Y平面的网格并计算网格上每点的高度`z`。由于绘制三维曲面的函数要求X、Y和Z轴的数据都用相同形状的二维数组表示，因此这里不能使用`ogrid`创建。和之前的`imshow()`不同，数组的第0轴可以表示X和Y轴中的任意一个，在本例中第0轴表示X轴、第1轴表示Y轴。

在当前图表中创建一个子图，通过`projection`参数指定子图的投影模式为“3d”，这样`subplot()`将返回一个用于三维绘图的`Axes3D`子图对象。

绘图函数简介

投影模式：投影模式决定了点从数据坐标转换为屏幕坐标的方式.可以通过下面的语句获得当前有效的投影模式的名称：

```
>>> from matplotlib import projections
>>> projections.get_projection_names()
['3d', 'aitoff', 'hammer', 'lambert', 'mollweide', 'polar',
'rectilinear']
```

只有在载入mplot3d模块之后，此列表中才会出现'3d'投影模式. 'aitoff'、'hammer', 'lambert', 'mollweide'等均为地图投影, 'polar' 为极坐标投影, 'rectilinear'则是默认的直线投影模式.

绘图函数简介

调用**Axes3D**对象的**plot_surface()**绘制三维曲面。其中：参数**x**、**y**、**z**都是形状为**(20,20)**的二维数组，数组**x**和**y**构成了**X-Y**平面上的网格，而数组**z**则是网格上各点在曲面上的取值。通过**cmap**参数来指定值和颜色之间的映射，即曲面上各点的高度值与其颜色的对应关系。**rstride** 和**cstride**参数分别是数组的第**0**轴和第**1**轴的下标间隔.对于很大的数组，使用较大的间隔可以提高曲面的绘制速度。程序中，**plot_surface()**调用和下面的语句是等价的：

```
ax.plot_surface(x[::2,:], y[::2,:], z[::2:],  
               rstride=1, cstride=1)
```

绘图函数简介

除了绘制三维曲面之外，**Axes3D**对象还提供了许多其他的三维绘图方法。可以通过下面的链接地址找到各种三维绘图的演示程序：

<http://matplotlib.sourceforge.net/examples/mplot3d/index.html>

<https://matplotlib.org/2.0.2/examples/mplot3d/index.html>

绘图函数简介

□ 3D作图（投影例）

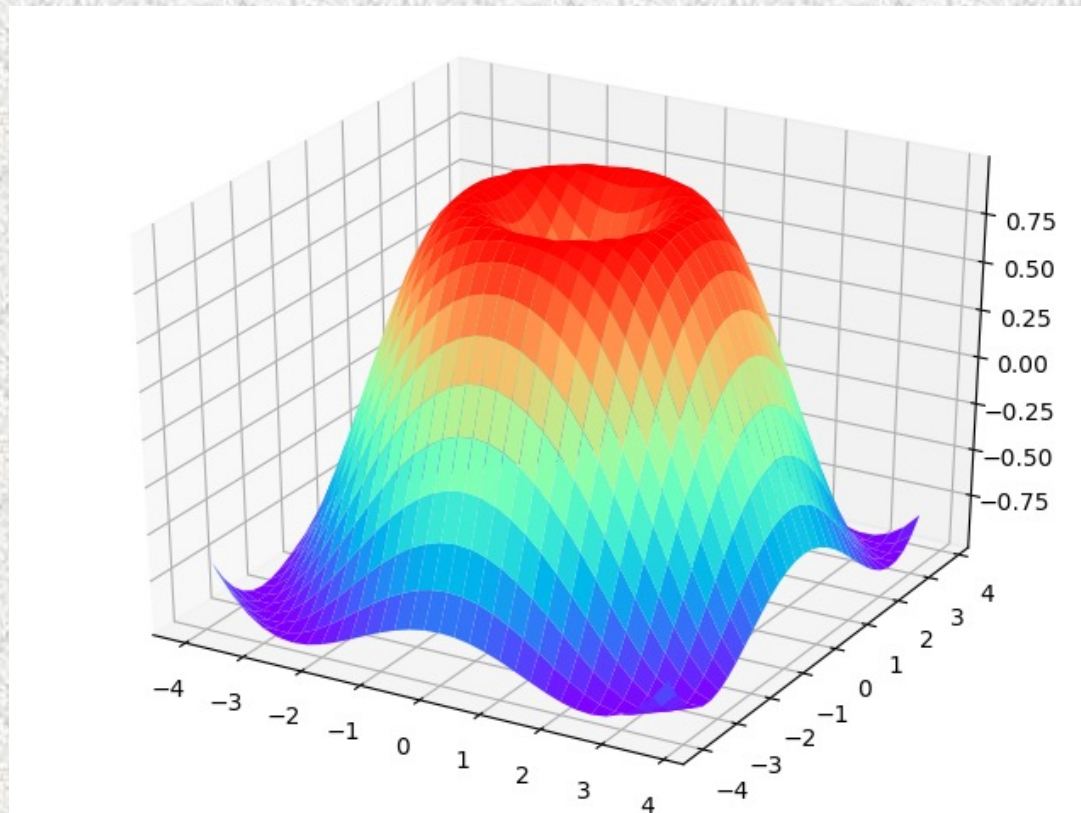
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)

# X, Y value
X = np.arange(-4, 4, 0.25)
Y = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(X, Y) # x-y 平面的网格
R = np.sqrt(X ** 2 + Y ** 2)
# height value
Z = np.sin(R)
```


绘图函数简介

```
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,  
cmap=plt.get_cmap('rainbow'))  
plt.show()
```



绘图函数简介

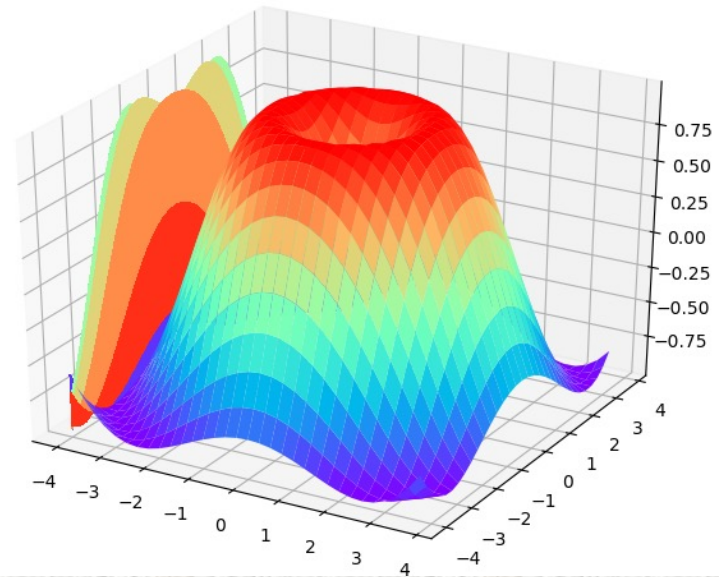
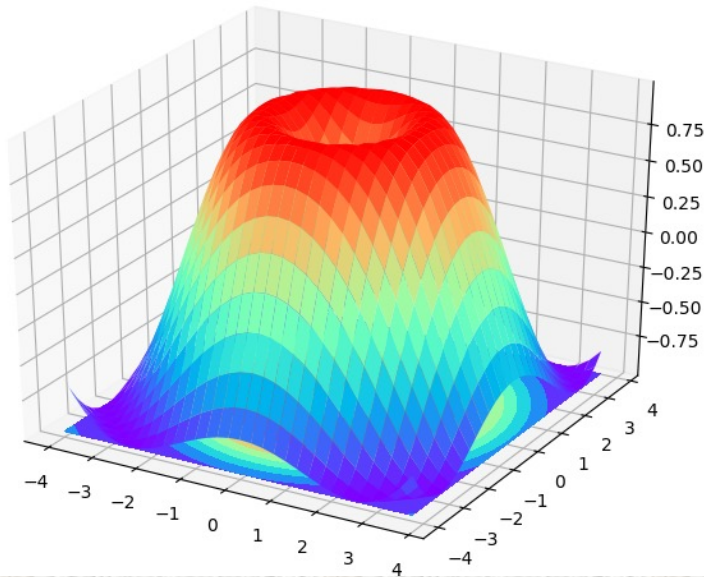
投影

有时候在观察**3D**图形时，可能需要图形映射到平面中来观察。等高线图可以帮助我们对**3D**图像进行投影。下面代码为添加 **XY** 平面的等高线,如果 **zdir** 选择了**x**，那么效果将会是对于 **XZ** 平面的投影，而调整**offset**可以调整投影出现的位置。

```
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
cmap=plt.get_cmap('rainbow'))
ax.contourf(X, Y, Z, zdir='z', offset=-1,
cmap=plt.get_cmap('rainbow'))
```

绘图函数简介

```
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
cmap=plt.get_cmap('rainbow'))
ax.contourf(X, Y, Z, zdir='x', offset=-4,
cmap=plt.get_cmap('rainbow'))
```



绘图函数简介

□ Animation 动画

matplotlib还提供了动画的接口。这里使用其中一种方式 **function animation** 。

```
from matplotlib import pyplot as plt
from matplotlib import animation
import numpy as np
```

```
fig, ax = plt.subplots()
```

```
x = np.arange(0, 2*np.pi, 0.01)
line, = ax.plot(x, np.sin(x))
```


绘图函数简介

接着，构造自定义动画函数**animate**，用来更新每一帧上各个**x**对应的**y**坐标值，参数表示第**i**帧；然后，构造开始帧函数**init**：

```
def animate(i):  
    line.set_ydata(np.sin(x + i/10.0))  
    return line,  
  
def init():  
    line.set_ydata(np.sin(x))  
    return line,  
  
ani = animation.FuncAnimation(fig=fig, func=animate,  
                              frames=100, init_func=init, interval=20, blit=True)  
  
plt.show()
```

绘图函数简介

用FuncAnimation函数生成动画。参数说明：

1.**fig** 进行动画绘制的figure

2.**func** 自定义动画函数，即传入刚定义的函数
animate

3.**frames** 动画长度，一次循环包含的帧数

4.**init_func** 自定义开始帧，即传入刚定义的函数
init

5.**interval** 更新频率，以ms计

6.**blit** 选择更新所有点，还是仅更新产生变化的点。
应选择**True**，但**mac**用户请选择**False**，否则无法显示动画

