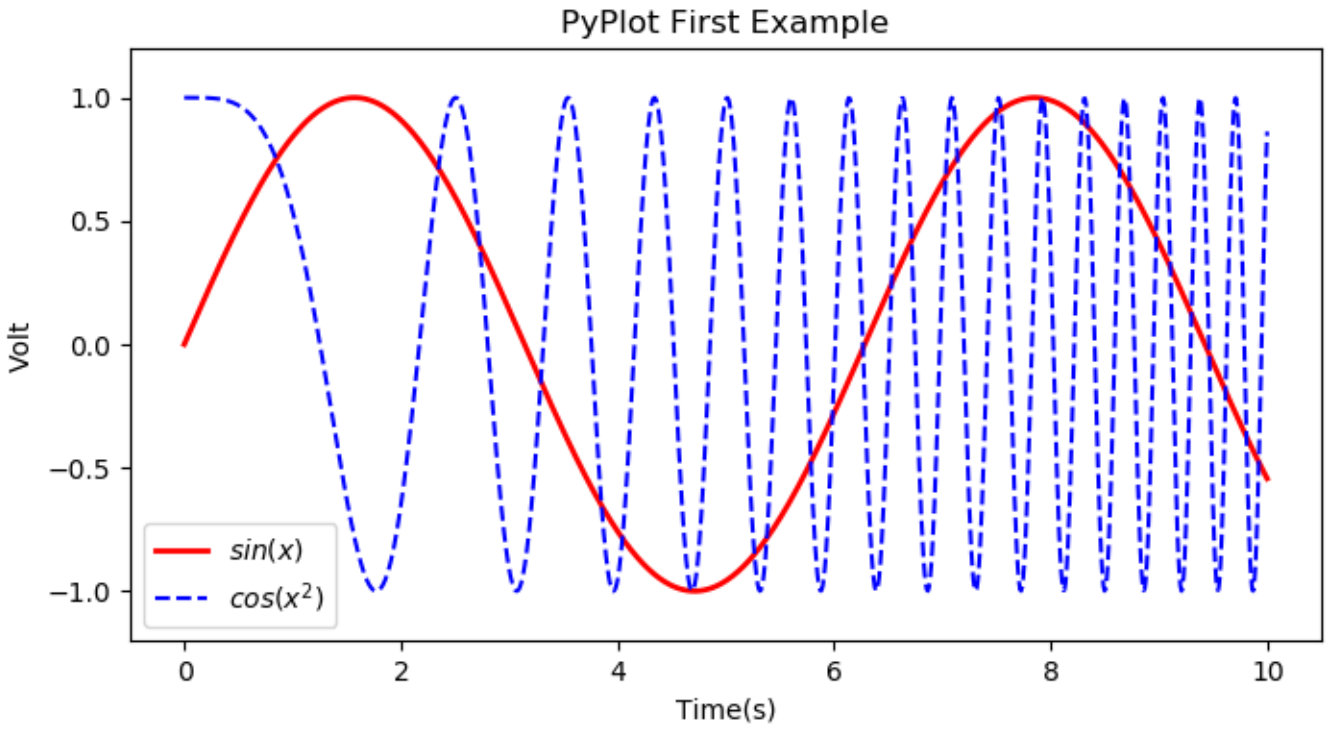
Matplotlib

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


PyPlot First Example
1.0
0.5
0.0
−0.5
−1.0
Volt
0
2
4
6
8
10
Time(s)
sin(x)
cos(x²)

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*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。

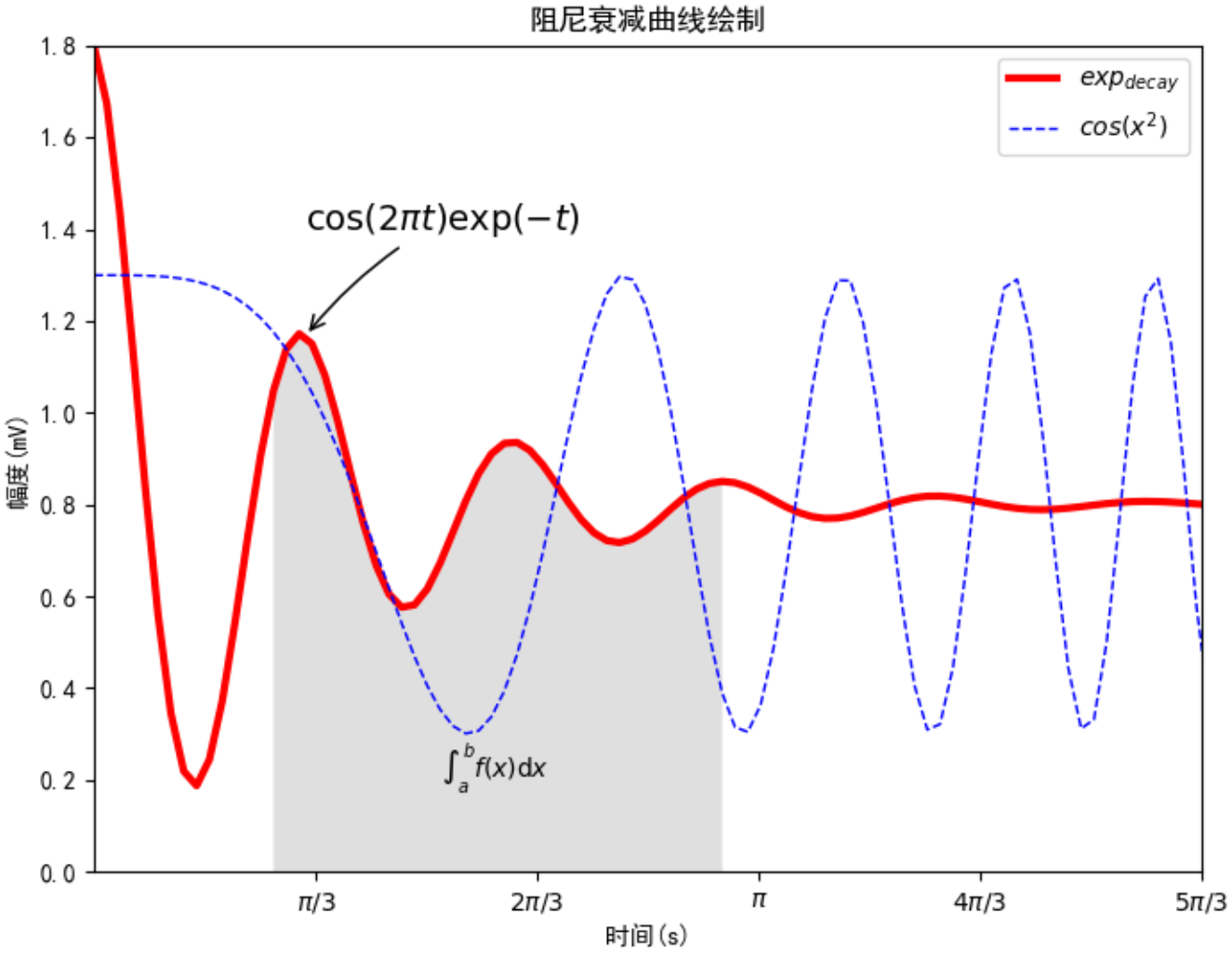
快速绘图

- 调用plt.savefig()将当前的Figure对象保存成图像文件
- 下面的程序将当前的图表保存为“test.png”

```
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

阻尼衰减曲线坐标图绘制



阻尼衰减曲线坐标图绘制

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
# 字体设置，SimHei为黑体
matplotlib.rcParams['font.family']='SimHei'
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
```

阻尼衰减曲线坐标图绘制

```
def Draw(pcolor, nt_point, nt_text, nt_size):
    plt.plot(x, y, 'k', label="$exp_{decay}$", color=pcolor,
linewidth=3, linestyle="-")
    plt.plot(x, z, "b--", label="$cos(x^2)$", linewidth=1)
    plt.xlabel('时间(s)')
    plt.ylabel('幅度(mV)')
    plt.title("阻尼衰减曲线绘制")
    plt.annotate('$\cos(2 \pi t) \exp(-t)$', xy=nt_point,
xytext=nt_text, fontsize=nt_size,
arrowprops=dict(arrowstyle='->',
connectionstyle="arc3,rad=.1"))
```

阻尼衰减曲线坐标图绘制

```
def Shadow(a, b):
    ix = (x>a) & (x<b)
    plt.fill_between(x,y,0,where=ix,facecolor='grey',
alpha=0.25)
    plt.text(0.5 * (a + b), 0.2, "$\int_a^b f(x)\mathrm{d}x$",
horizontalalignment='center')

def XY_Axis(x_start, x_end, y_start, y_end):
    plt.xlim(x_start, x_end)
    plt.ylim(y_start, y_end)
    plt.xticks([np.pi/3, 2 * np.pi/3, 1 * np.pi, 4 * np.pi/3, 5
* np.pi/3], ['$\pi/3$', '$2\pi/3$', '$\pi$', '$4\pi/3$',
'$5\pi/3$'])
```

阻尼衰减曲线坐标图绘制

```
x = np.linspace(0.0, 6.0, 100)
y = np.cos(2 * np.pi * x) * np.exp(-x)+0.8
z = 0.5 * np.cos(x ** 2)+0.8
note_point,note_text,note_size = (1, np.cos(2 * np.pi) *
np.exp(-1)+0.8),(1, 1.4), 14
fig = plt.figure(figsize=(8, 6), facecolor="white")
plt.subplot(111)
Draw("red", note_point, note_text, note_size)
XY_Axis(0, 5, 0, 1.8)
Shadow(0.8, 3)
plt.legend()
plt.savefig('sample.JPG')
plt.show()
```

快速绘图

- 绘制多轴图（子图）
  - 一个绘图对象(figure)可以包含多个轴，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：

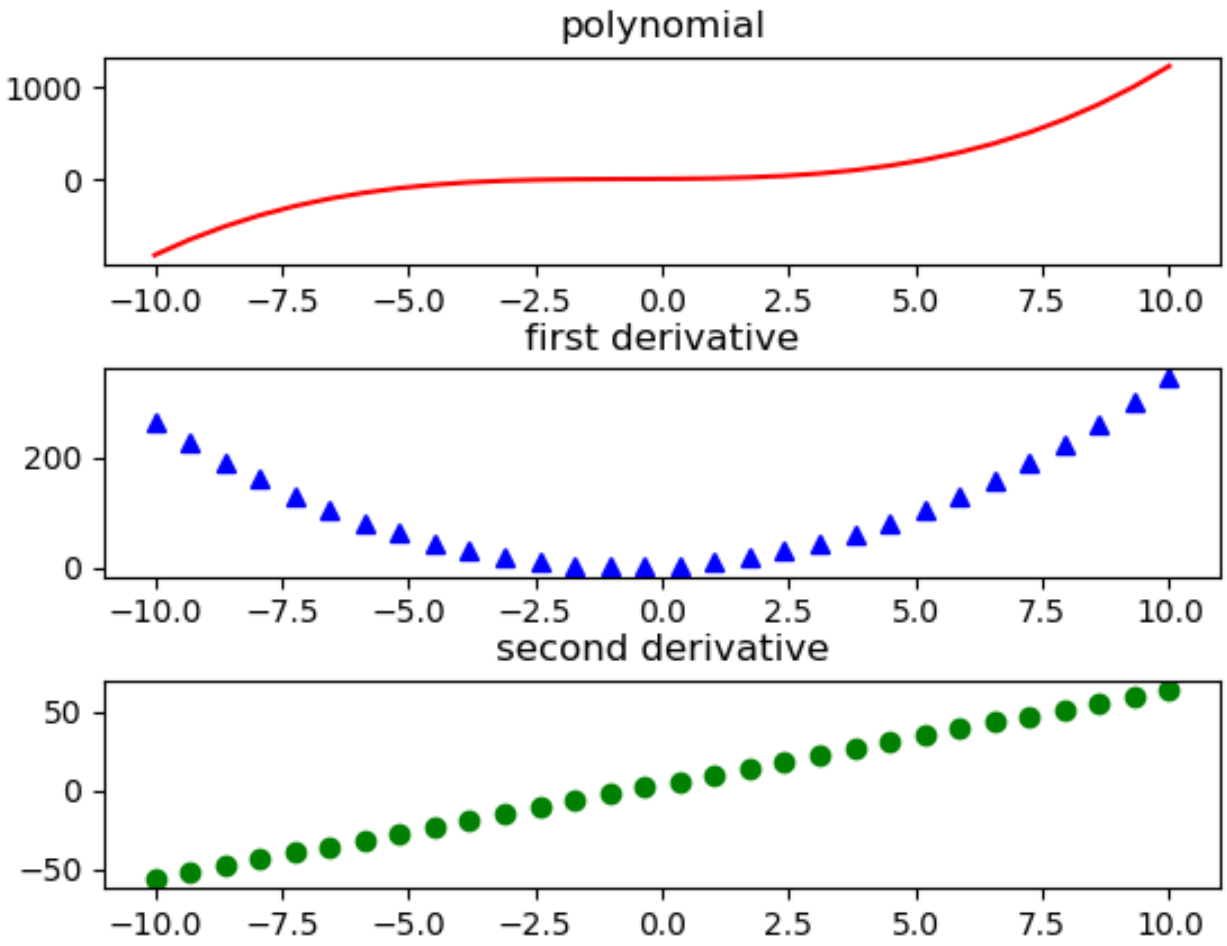
```
subplot(numRows, numCols, plotNum)
```

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，左上的子区域的编号为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。

子图例子：

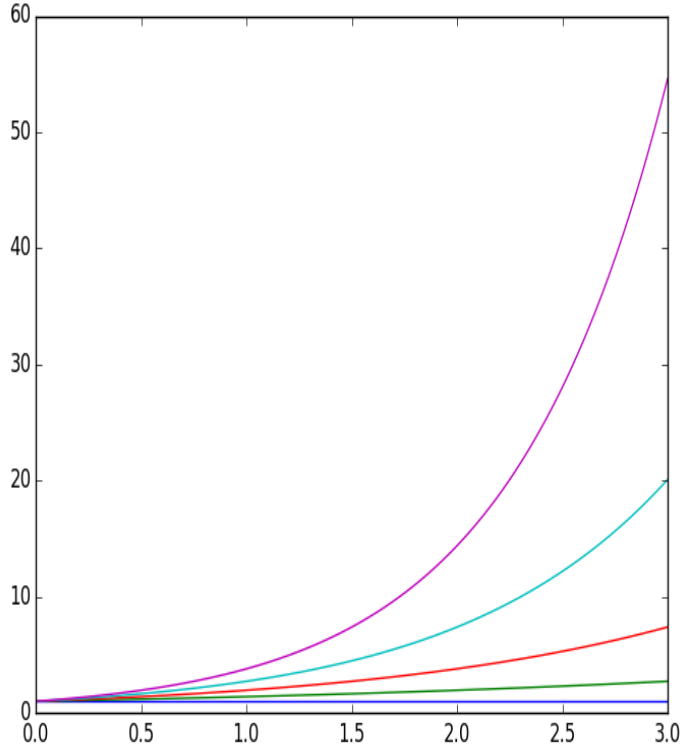
```
import numpy as np
import matplotlib.pyplot as plt

func = np.poly1d(np.array([1,2,3,4]).astype(float)) #原函数
func1 = func.deriv(1)                               #一阶导函数
func2 = func.deriv(2)                               #二阶导函数
x = np.linspace(-10,10,30)
y, y1, y2 = func(x), func1(x), func2(x)
plt.subplot(311)                          #3行1列排布子图，该为第1个子图
plt.plot(x,y,'r-')
plt.title('polynomial')
plt.subplot(312)                          #3行1列排布子图，该为第2个子图
plt.plot(x,y1,'b^')
plt.title('first derivative')
plt.subplot(313)                          #3行1列排布子图，该为第3个子图
plt.plot(x,y2,'go')
plt.title('second derivative')
plt.subplots_adjust(hspace=.5)
plt.show()
```

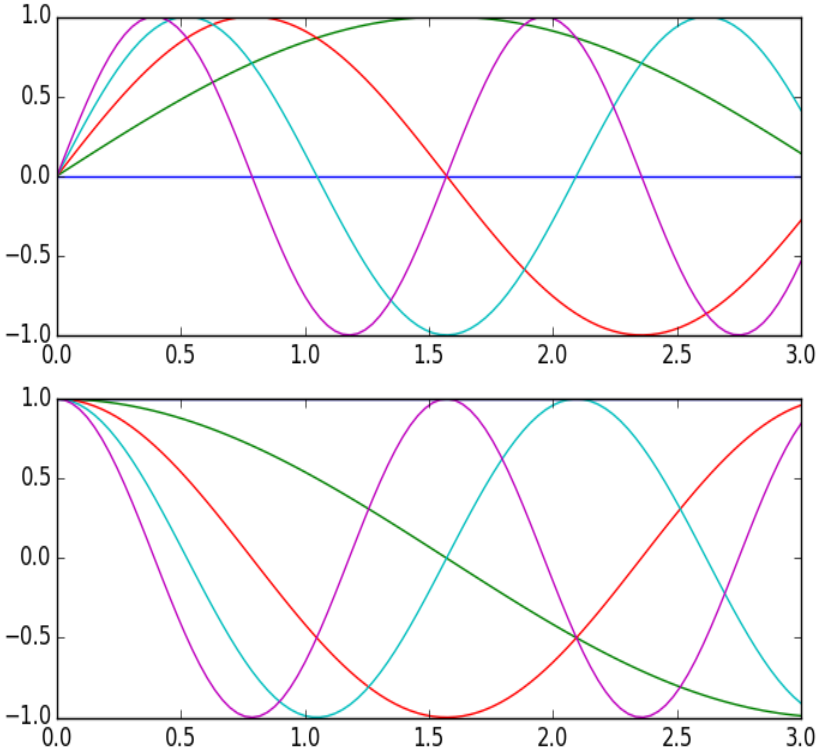

polynomial
1000
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
first derivative
200
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
second derivative
50
0
−50
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

选中绘图对象

- 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。

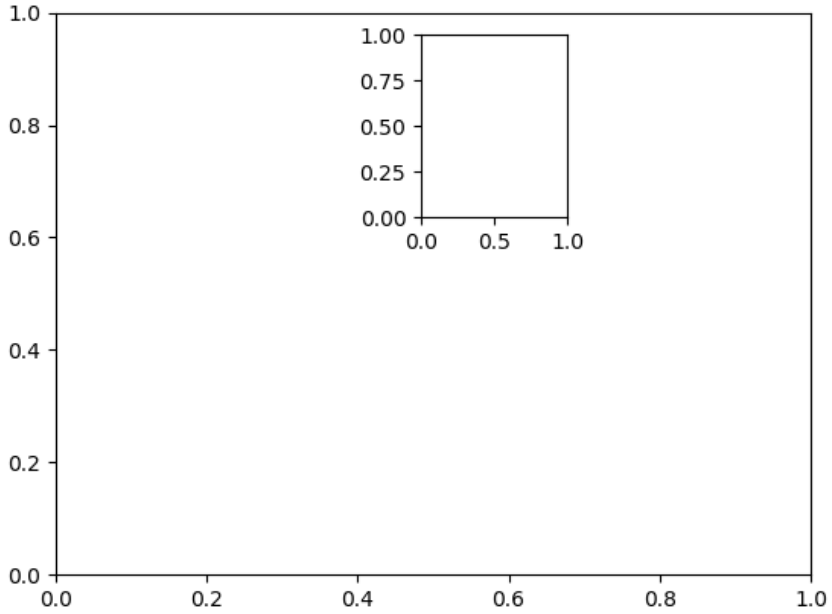
```
import numpy as np
import matplotlib.pyplot as plt

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)
for i in range(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()
```


60
50
40
30
20
10
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


1.0
0.5
0.0
−0.5
−1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

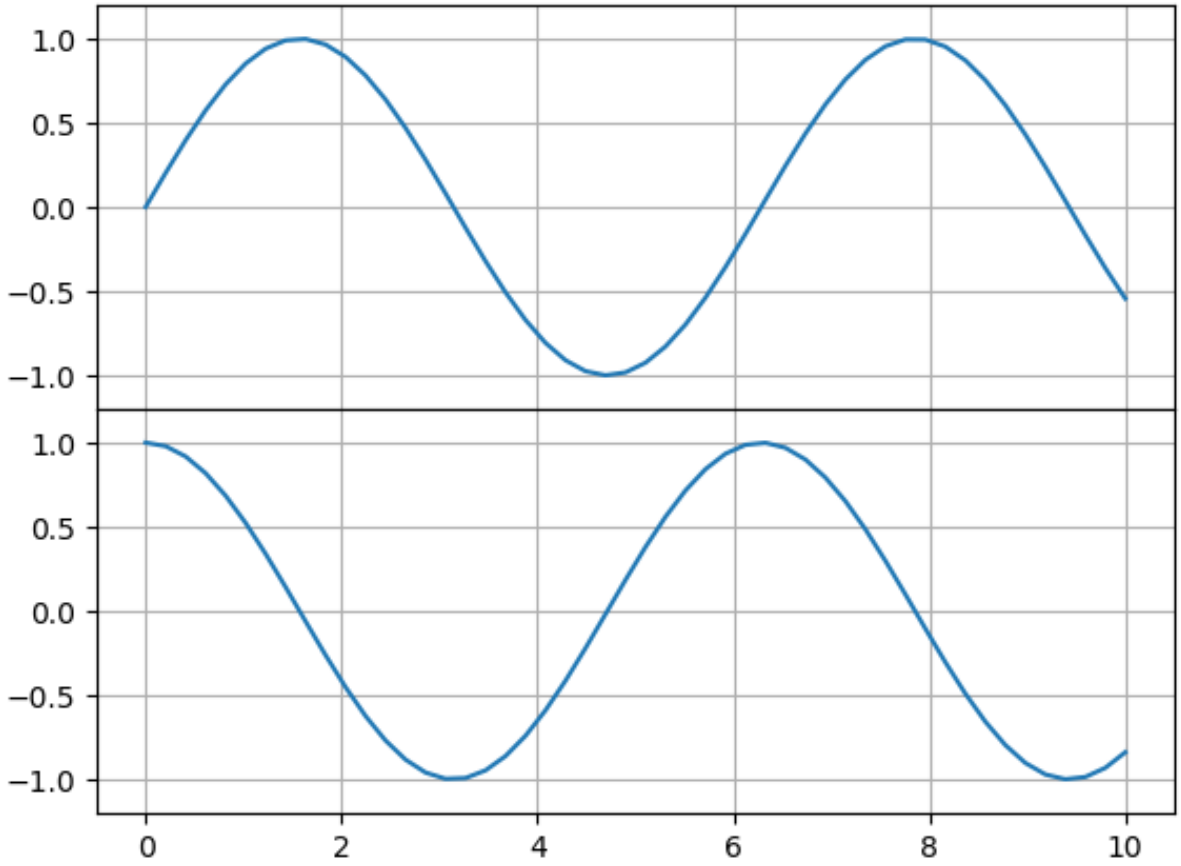
子图布局设置

```
ax1 = plt.axes()
ax2 = plt.axes([0.5, 0.6, 0.15, 0.25])
plt.show()
```



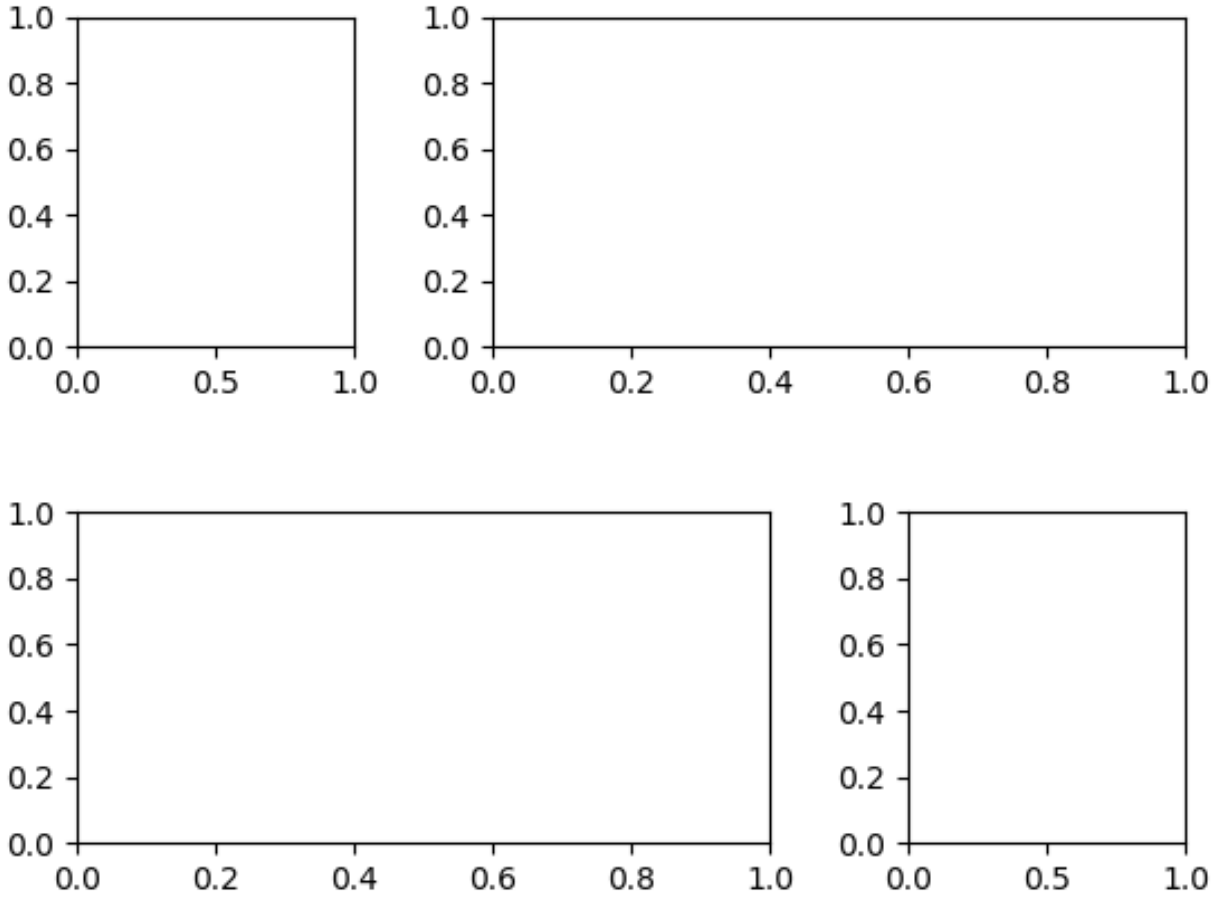
子图布局设置

```
x = np.linspace(0, 10)
plt.axes([0.1, 0.5, 0.8, 0.4], ylim=(-1.2, 1.2))
plt.grid(True)
plt.plot(x, np.sin(x))

plt.axes([0.1, 0.1, 0.8, 0.4], ylim=(-1.2, 1.2))
plt.grid(True)
plt.plot(x, np.cos(x))
plt.show()
```


1.0
0.5
0.0
−0.5
−1.0
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10

不规则多行多列子图

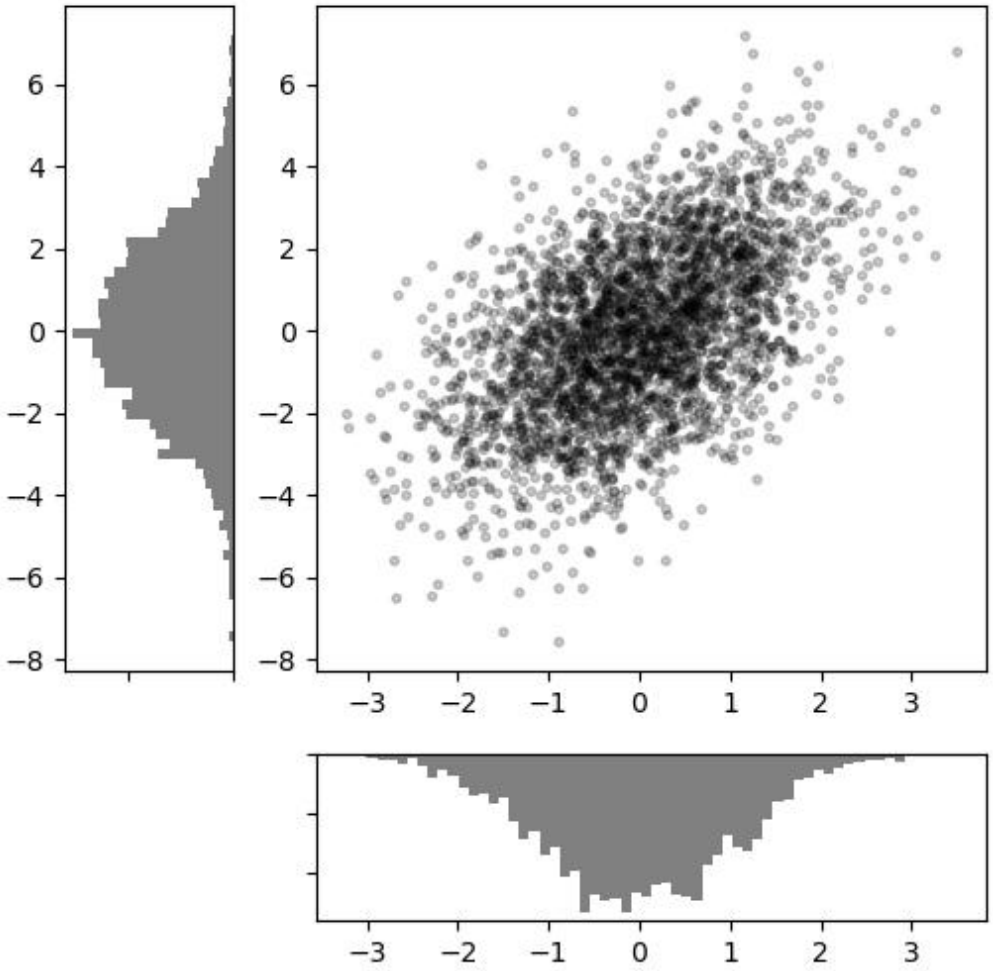
```
grid = plt.GridSpec(2, 3, wspace=0.5, hspace=0.5)
plt.subplot(grid[0,0])
plt.subplot(grid[0,1:3])
plt.subplot(grid[1,0:2])
plt.subplot(grid[1,2])

plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0

不规则多行多列子图

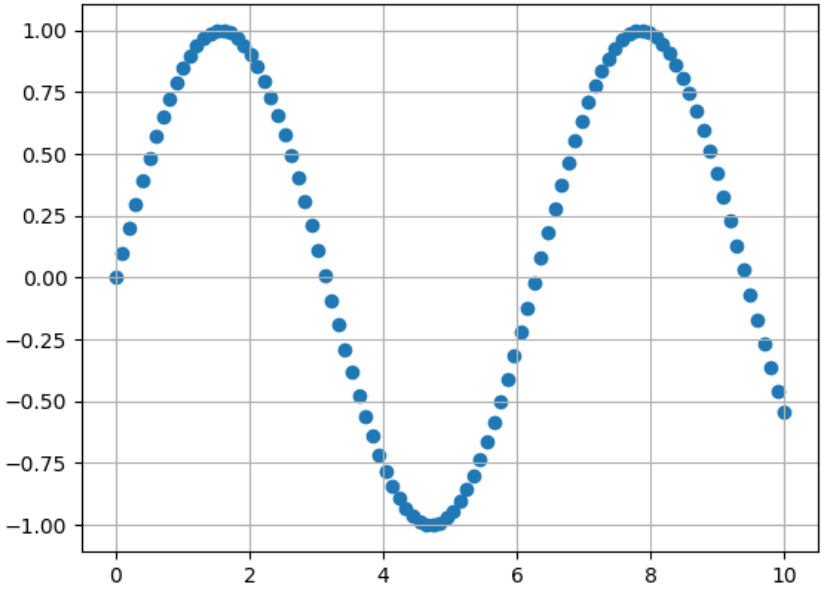
- 例子：在一个子图中画出二元正态分布的联合分布图，而在另两个子图中分别画出x轴和y轴方向上的边缘分布图

```
mean = [0, 0]
cov = [[1, 1], [1, 4]]
x, y = np.random.multivariate_normal(mean, cov, 3000).T
plt.figure(figsize=(6,6))
grid = plt.GridSpec(4, 4, wspace=0.5, hspace=0.5)

main_ax = plt.subplot(grid[0:3,1:4])
plt.plot(x,y,'ok',markersize=3,alpha=0.2)

y_hist = plt.subplot(grid[0:3,0],xticklabels=[],sharey=main_ax)#和大子图共y轴
plt.hist(y,60,orientation='horizontal',color='gray')#图形水平绘制
y_hist.invert_xaxis()#x轴调换方向

x_hist = plt.subplot(grid[3,1:4],yticklabels=[],sharex=main_ax)#和大子图共x轴
plt.hist(x,60,orientation='vertical', color='gray')#图形垂直绘制
x_hist.invert_yaxis()#y轴调换方向

plt.show()
```


6
4
2
0
−2
−4
−6
−8
−3
−2
−1
0
1
2
3

散点图

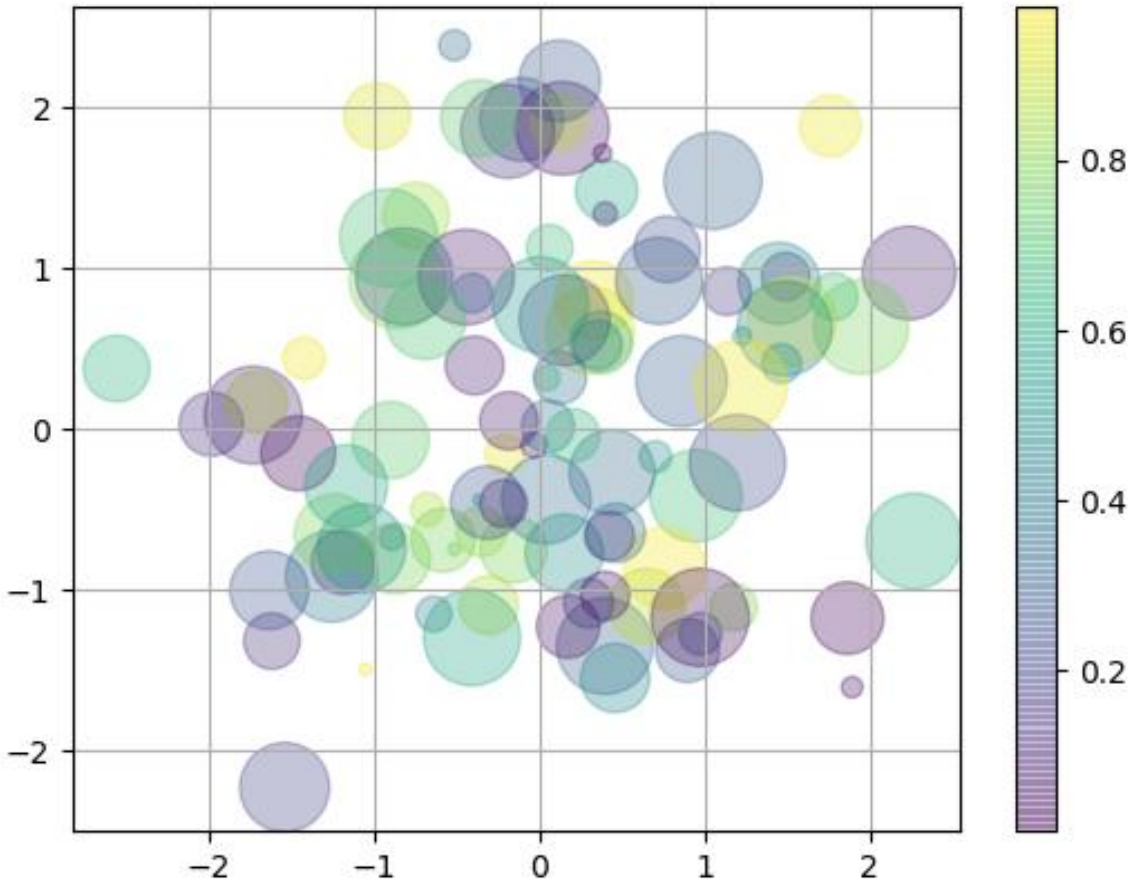
```
x = np.linspace(0,10,100)
plt.scatter(x, np.sin(x), marker='o')
plt.grid()
plt.show()
```



例子

画一组散点图，点的位置坐标（x, y）是服从标准正态分布的随机值，点的颜色灰度值（0,1）空间中的随机样本，点的大小是随机值的1000倍，单位是像素

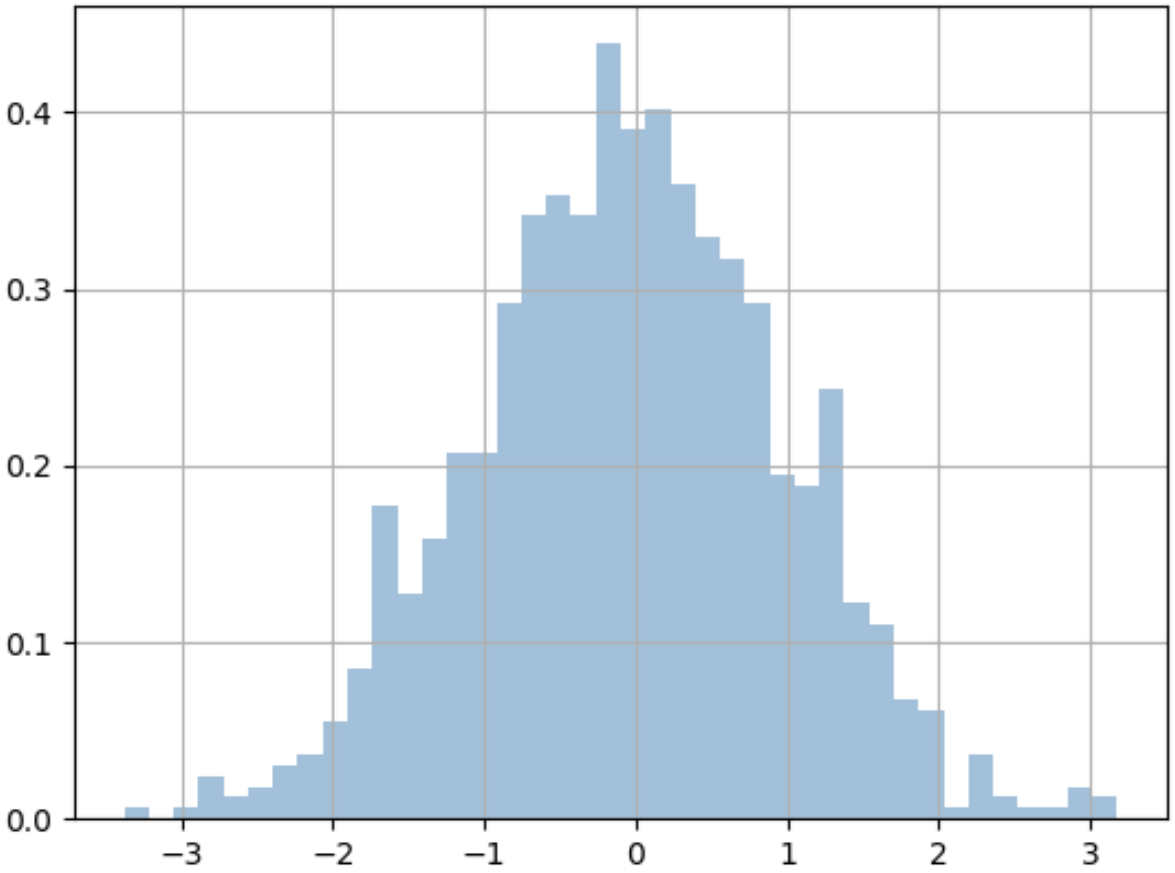
```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)
plt.scatter(x,y,c=colors,s=sizes,alpha=0.3)
plt.colorbar()
plt.grid(True)
plt.show()
```


2
1
0
−1
−2
−2
−1
0
1
2
0.8
0.6
0.4
0.2

频次直方图

例子：生成1000个服从正态分布的随机变量，观察每个点的出现次数

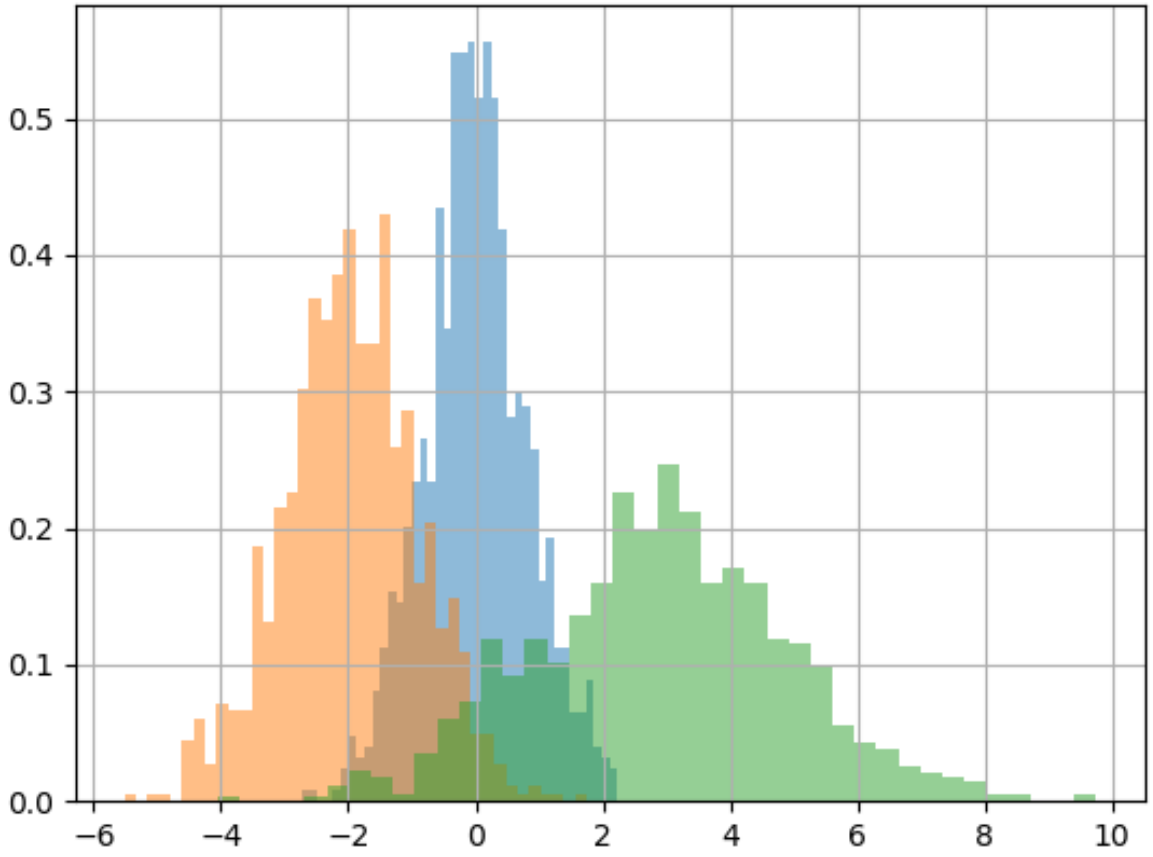
```
data = np.random.randn(1000)
plt.hist(data, bins=40, normed=True, alpha=0.5,
         histtype='stepfilled',color='steelblue')
plt.grid(True)
plt.show()
```


0.4
0.3
0.2
0.1
0.0
−3
−2
−1
0
1
2
3

频次直方图

例子：将不同分布进行对比

```
data1 = np.random.normal(0, 0.8, 1000)
data2 = np.random.normal(-2, 1, 1000)
data3 = np.random.normal(3, 2, 1000)
plt.grid(True)
kwargs = dict(histtype='stepfilled', alpha=0.5, normed=True,
bins=40)
plt.hist(data1, **kwargs)
plt.hist(data2, **kwargs)
plt.hist(data3, **kwargs)
plt.show()
```

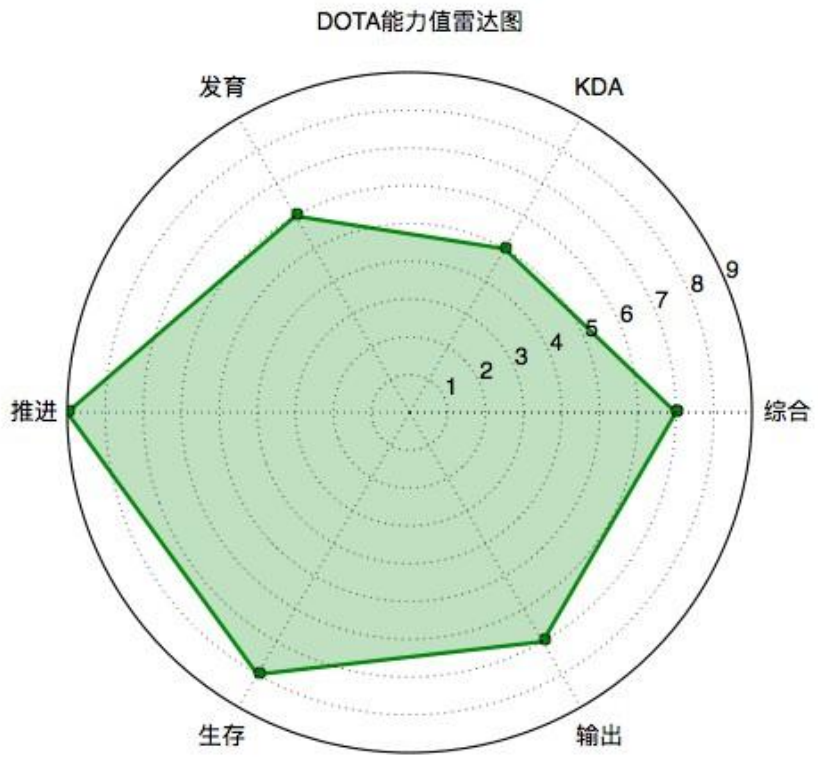

0.5
0.4
0.3
0.2
0.1
0.0
−6
−4
−2
0
2
4
6
8
10

多级雷达图

- 雷达图是通过多个离散属性比较对象的最直观工具，掌握绘制雷达图将会为生活和工作带来乐趣。

- 游戏角色中经常出现表示人物能力值的雷达图。DOTAMAX测试版曾经推出过显示玩家能力值分布的雷达图，只要点击自己或是好友头像，就可以看到能力值在综合、KDA、发育、推进、生存、输出等方面的能力分布

DOTA人物能力值雷达图



DOTA人物能力值雷达图

- 使用Python 来绘制多级雷达图，即在一组同心圆上填充不规则六边形，其每个顶点到圆心的距离代表人物某个属性的数据。

DOTA人物能力值雷达图

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family']='SimHei'
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
labels = np.array(['综合', 'KDA', '发育', '推进', '生存','输出'])
nAttr = 6
data = np.array([7, 5, 6, 9, 8, 7]) #数据值
angles = np.linspace(0, 2*np.pi, nAttr, endpoint=False)
data = np.concatenate((data, [data[0]]))
angles = np.concatenate((angles, [angles[0]]))
fig = plt.figure(facecolor="white")
```

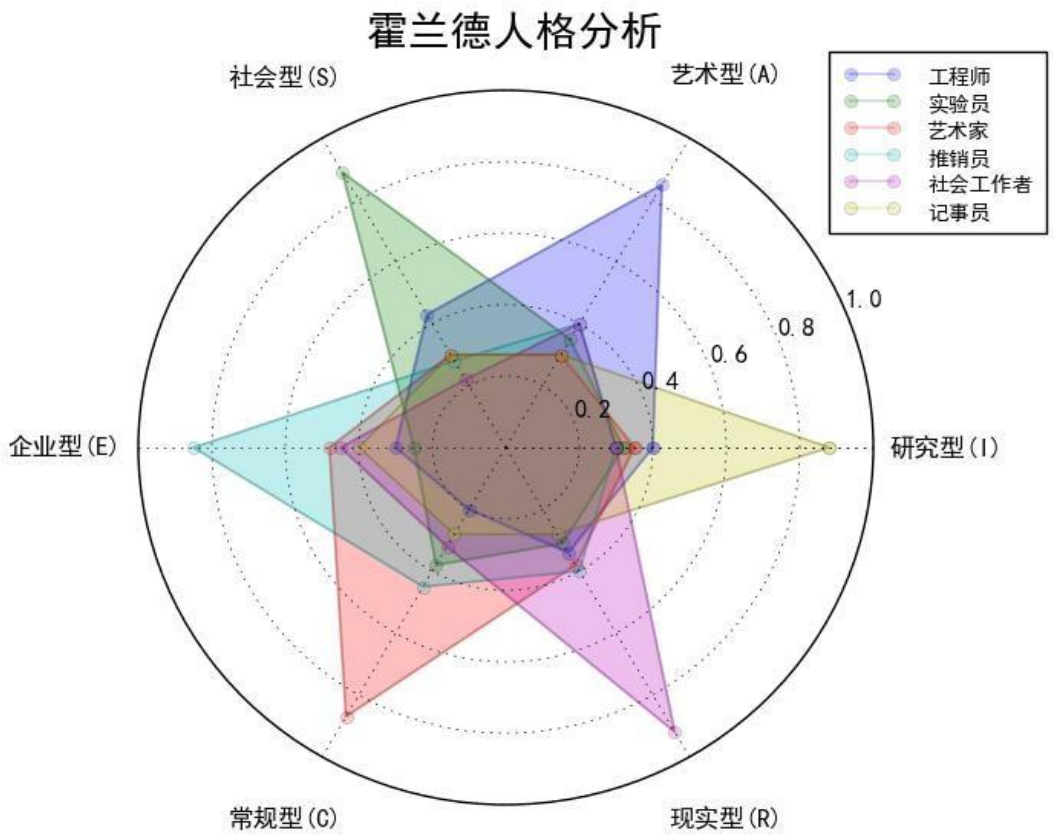
DOTA人物能力值雷达图

```
plt.subplot(111, polar=True)
plt.plot(angles,data,'bo-',color ='g',linewidth=2)
plt.fill(angles,data,facecolor='g',alpha=0.25)
plt.thetagrids(angles*180/np.pi, labels)
plt.figtext(0.52, 0.95, 'DOTA能力值雷达图', ha='center')
plt.grid(True)
plt.show()
```

多级雷达图绘制

- 除了DOTA游戏，雷达图应用广泛。美国约翰霍普金斯大学霍兰德教授认为兴趣是人们活动的巨大动力，凡是具有职业兴趣的职业，都可以提高人们的积极性，促使人们积极地、愉快地从事该职业。因此，他研究了人格类型、兴趣与职业间的关系，提出了“霍兰德职业兴趣理论”，认为人格可分为现实型、研究型、艺术型、社会型、企业型和常规型等六种类型。

霍兰德人格分析雷达图


霍兰德人格分析
社会型(S)
艺术型(A)
企业型(E)
研究型(I)
常规型(C)
现实型(R)
0.2
0.4
0.6
0.8
1.0
工程师
实验员
艺术家
推销员
社会工作者
记事员

霍兰德人格分析雷达图

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family']='SimHei'
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
radar_labels = np.array(['研究型(I)','艺术型(A)','社会型(S)','
企业型(E)','常规型(C)','现实型(R)'])
```

霍兰德人格分析雷达图

```
nAttr = 6
data = np.array([[0.40, 0.32, 0.35, 0.30, 0.30, 0.88],
                 [0.85, 0.35, 0.30, 0.40, 0.40, 0.30],
                 [0.43, 0.89, 0.30, 0.28, 0.22, 0.30],
                 [0.30, 0.25, 0.48, 0.85, 0.45, 0.40],
                 [0.20, 0.38, 0.87, 0.45, 0.32, 0.28],
                 [0.34, 0.31, 0.38, 0.40, 0.92, 0.28]]) #数据值
data_labels = ('工程师', '实验员', '艺术家', '推销员', '社会工作者
','记事员')
angles = np.linspace(0, 2*np.pi, nAttr, endpoint=False)
data = np.concatenate((data, [data[0]]))
angles = np.concatenate((angles, [angles[0]]))
```

霍兰德人格分析雷达图

```
fig = plt.figure(facecolor="white")
plt.subplot(111, polar=True)
plt.plot(angles,data,'o-', linewidth=1.5, alpha=0.2)
plt.fill(angles,data, alpha=0.25)
plt.thetagrids(angles*180/np.pi, radar_labels)
plt.figtext(0.52, 0.95, '霍兰德人格分析', ha='center', size=20)
legend = plt.legend(data_labels, loc=(0.94, 0.80), labelspacing=0.1)
plt.setp(legend.get_texts(), fontsize='small')
plt.grid(True)
plt.show()
```

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
```

坐标轴设定

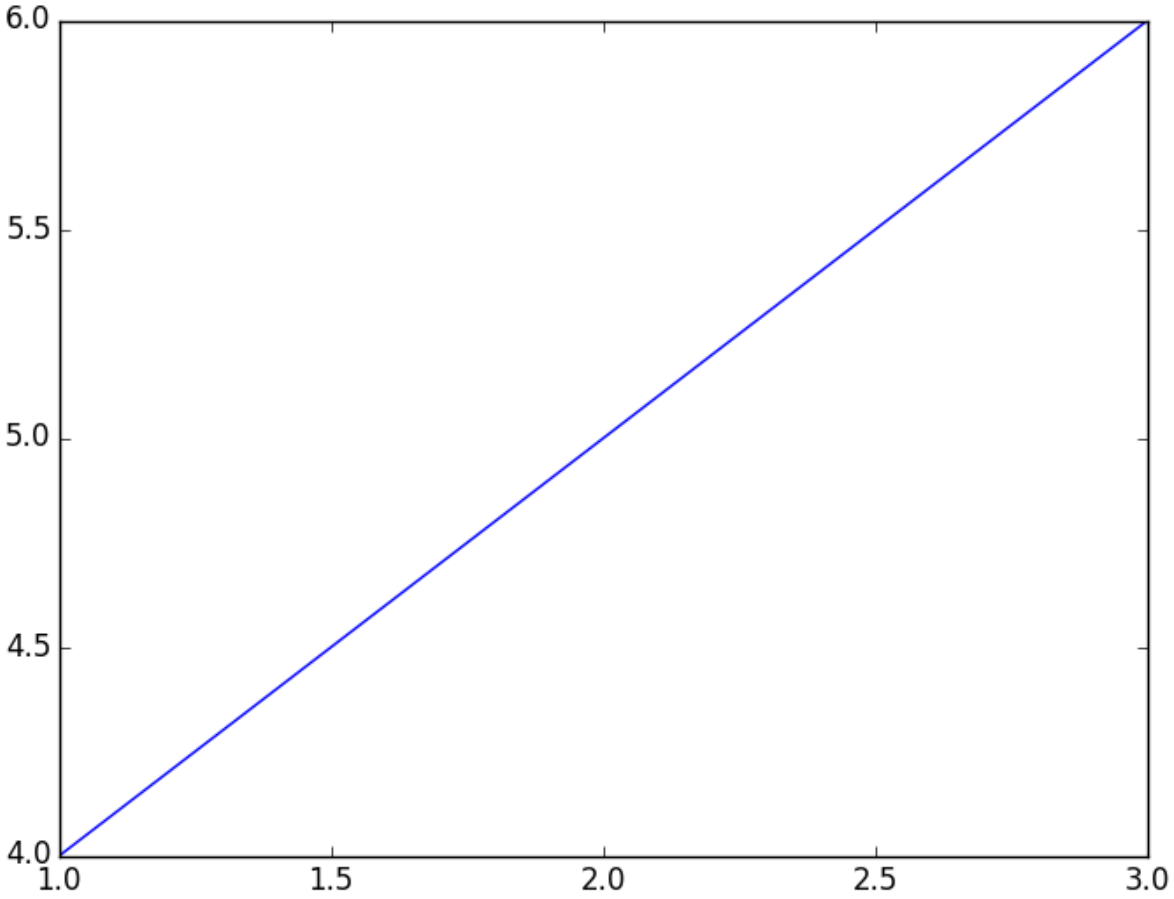
- 获得axis对象的刻度位置列表：

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

- 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
['1.0', '1.5', '2.0', '2.5', '3.0']
```

坐标轴设定



坐标轴设定

- 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)
```

坐标轴设定

- 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。

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FuncFormatter
x = np.linspace(0, 3*np.pi, 100)
plt.plot(x,np.sin(x))
ax = plt.axes()
```

坐标轴设定

- 程序中通过format_func()计算出刻度值对应的刻度文本

```
def format_func(value, tick_number):
    N = int(np.round(2 * value / np.pi))
    if N == 0:
        return '0'
    elif N == 1:
        return r"$\pi/2$"
    elif N == 2:
        return r"$\pi$"
    elif N % 2 > 0:
        return r"${}\pi/2$".format(N)
    else:
        return r"${}\pi$".format(N//2)
```

坐标轴设定

- 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/2))
ax.xaxis.set_minor_locator(MultipleLocator(np.pi/4))
```

- 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(format_func))
```

坐标轴设定