

第 5 章



Python 绘图

数学中,数之间的联系经常需要借助图形的生动性和直观性来阐明。在本章中,将介绍一种有效的数据表示方法:使用 Matplotlib 为数据绘制图形。Matplotlib 是一个用于在 Python 中创建静态、动画和交互式视图的综合库。它能让使用者很轻松地将数据图形化,并且提供多样化的输出格式。

5.1 安装 Matplotlib

Anaconda 集成了 NumPy,而 Matplotlib 是 NumPy 的一个库。如果安装了 Anaconda 则无须单独安装 Matplotlib。

如果需要安装 Matplotlib 及其依赖项,在 macOS、Windows 和 Linux 发行版上,可以使用以下命令安装 Wheel 包:

(1) `python -m pip install -U pip`。

(2) `python -m pip install -U matplotlib`。

对于 Linux 发布版,可以使用如下命令安装第三方发布版:

(1) Debian / Ubuntu: `sudo apt-get install python3-matplotlib`。

(2) Fedora: `sudo dnf install python3-matplotlib`。

(3) Red Hat: `sudo yum install python3-matplotlib`。

(4) Arch: `sudo pacman-S python-matplotlib`。

5.2 绘制简单图形

我们使用 Python 来绘制的第一张图形是连接(2, 3)、(6, 9)两点的线段。参看以下代码:

```
In [1]: import matplotlib.pyplot as plt
        x_axis = [2, 3]
        y_axis = [6, 9]
        plt.plot(x_axis, y_axis)
        plt.show()
```

在这段代码中,首先导入 pyplot 绘图接口,并为其起了一个别名 plt,这样后续使用时

可以减少代码的输入量。有的程序中使用 `from pylab import *`, 这样在后续的代码中可以直接使用 `plot` 和 `show`。这种写法是为了照顾 MATLAB 用户的使用习惯, 本书不推荐使用这种写法, 特别是在大型工程中, 因为这样做可能会污染工程的命名空间。

注意: 不要直接导入 `pylab`。

接下来的两行定义了两个列表, 分别表示 X 坐标轴和 Y 坐标轴上两点的坐标值。函数 `plot()` 在当前图形的当前平面直角坐标系内, 使用线条、标记或两者的结合标识出 x 和 y 的映射关系。函数 `plot()` 接收可变长度的入参, X 轴上的坐标列表是可选输入, Y 轴上的坐标列表是必须的。如果当前图形和当前平面直角坐标系不存在, 则函数 `plot()` 会首先创建它们, 然后创建并返回一个 `Line2D` 对象。更准确地说, 返回一个包含多个 `Line2D` 对象的列表。最后一行的函数 `show()` 使用 Matplotlib 的默认配置将图形显示出来。

如果使用 Notebook, 则可以在绘图之前包括命令 `%matplotlib inline` 或 `%matplotlib Notebook`, 以使绘图自动显示在 Notebook 内部。如果代码通过 IPython 控制台运行, 则图形可能会自动出现在控制台中。在 Spyder 中, 绘图结果默认在右上角的 Plots 窗口显示。无论哪种方式, 始终使用函数 `show()` 以显示绘图的做法都是一种好的习惯。

以上代码的运行结果如图 5-1 所示。

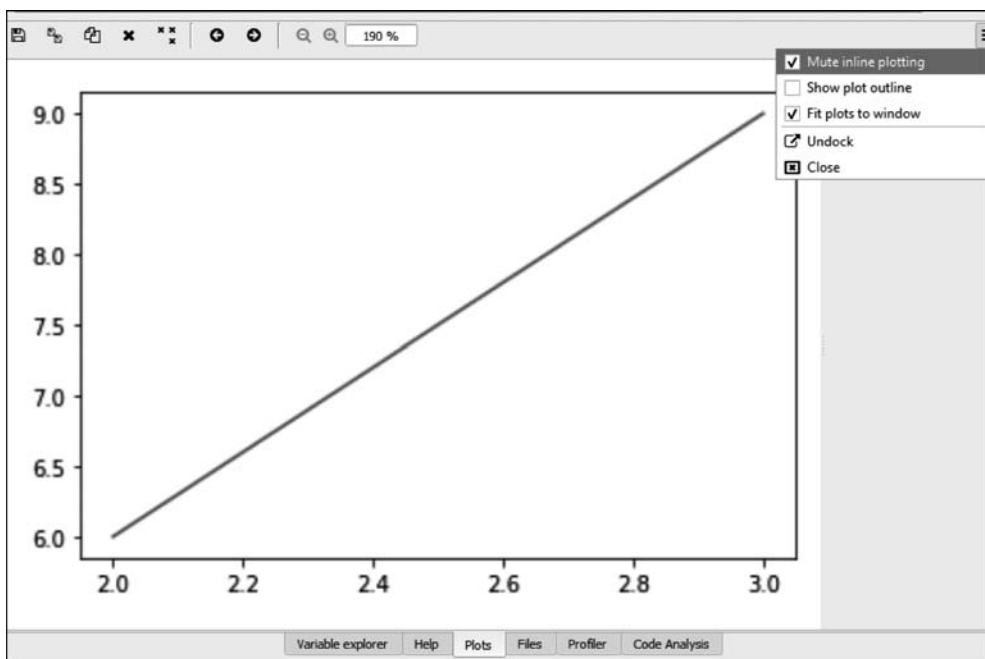


图 5-1 (2, 6)、(3, 9)两点的连线

默认情况下, 绘制的结果仅在 Plots 窗口显示, 如果希望它同时能够在 IPython Console 内显示, 需在 option 栏内取消对 `Mute inline plotting` 的选择。

函数 `plot()` 有多个关键字参数, 其中关键字参数 `marker` 用于设置各点在图中的显示方式。增加该参数后, 函数 `plot()` 绘图时会将列表中的点在连线上标注出来, 代码如下:

```
In [2]: x_axis = [2, 4, 6]
        y_axis = [6, 12, 18]
        plt.plot(x_axis, y_axis, marker = 'o')
        plt.show()
```

由于调用 `plot()` 时增加了关键字参数 `marker` (关于关键字参数, 详情可参见 3.5.2 节), 参数列表中的每个点均被标识了出来, 如图 5-2 所示。

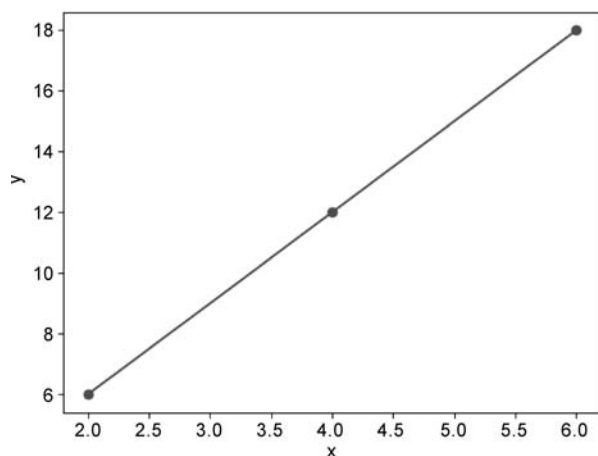


图 5-2 三点被标识的连线

以下代码中缺失了关键字 `marker`:

```
In [3]: x_axis = [2, 4, 6]
        y_axis = [6, 12, 18]
        plt.plot(x_axis, y_axis, 'o')
        plt.show()
```

此时输出的结果如图 5-3 所示, 绘图结果是散点图。

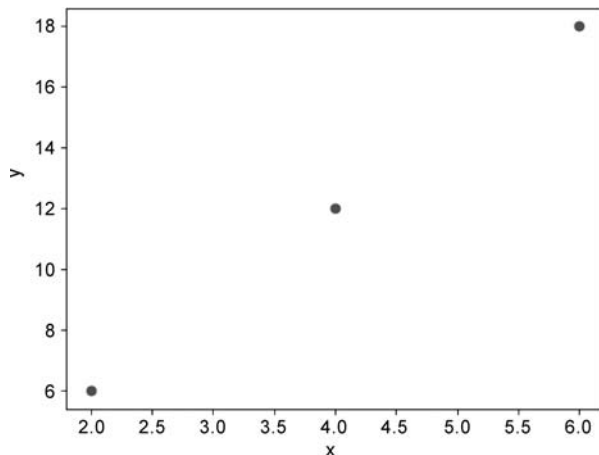


图 5-3 散点图

Matplotlib 中可供选择的简单的参数 marker 选项如表 5-1 所示,可尝试修改示例代码中参数 marker 的值。

表 5-1 Matplotlib 中的参数 marker

marker	显示符号	说 明
'.'	•	点
','	.	像素
'o'	●	实心圆
'v'	▼	倒三角
'^'	▲	正三角
'<'	◀	左三角
'>'	▶	右三角
'1'	⋈	
'2'	⋈	
'3'	⋈	
'4'	⋈	
'8'	●	八边形
's'	■	正方形 1
'p'	⬠	五边形
'P'	⊕	十字
'*'	★	星形
'h'	⬡	六边形 1
'H'	⬡	六边形 2
'+'	+	加号
'x'	×	乘号
'X'	⊗	叉号
'D'	◆	正方形 2
'd'	◆	菱形
' '		竖线
'_'	—	横线

针对同样的一组数据,Matplotlib 还支持柱状图和散点图的绘制,代码如下:

```
In [4]: import matplotlib.pyplot as plt
        x_axis = [2, 4, 6]
        y_axis = [6, 12, 18]

        # 绘制散点图
        plt.subplot(121)
        plt.scatter(x_axis, y_axis)

        # 绘制柱状图
        plt.subplot(122)
        plt.bar(x_axis, y_axis)
```

函数 `scatter()` 用于绘制散点图, 函数 `bar()` 用于绘制柱状图。函数 `subplot()` 用于创建子图, 关于子图详见 5.5.3 节。以上代码的运行结果如图 5-4 所示。

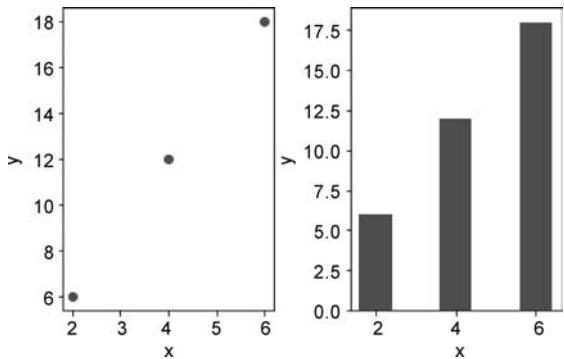


图 5-4 散点图和柱状图

关键字参数 `color` (代码中为 `c`) 可以指定线条的颜色。颜色最简单的一种表示方式是使用字符, 可选字符是集合 `{'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'}` 中元素之一, 它们分别代表蓝色、绿色、红色、青色、洋红色、黄色、黑色和白色。

以下代码将线条的颜色指定为黑色:

```
In [5]: x_axis = [2, 4, 6]
        y_axis = [6, 12, 18]
        plt.plot(x_axis, y_axis, c='k')
        plt.show()
```

关键字参数 `linestyle` (代码中为 `ls`) 用来指定线条的类型。有 2 种输入方式, 其中最简便的是使用预定义的格式字符, 可选类型字符如表 5-2 所示。

表 5-2 线条类型参数

参 数	描 述
'-' or 'solid'	实线
'--' or 'dashed'	虚线
'-. ' or 'dashdot'	点画线
':' or 'dotted'	点虚线
'None' or 'or'	无线条

以下代码以黑色点画线的形式输出:

```
In [6]: x_axis = [2, 4, 6]
        y_axis = [6, 12, 18]
        plt.plot(x_axis, y_axis, c='k', ls='-. ')
        plt.show()
```

代码执行结果如图 5-5 所示。

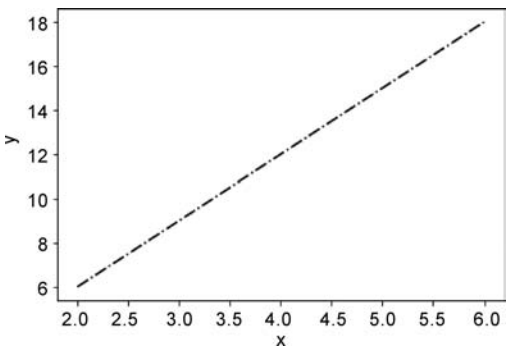


图 5-5 黑色点画线图

5.3 北京、上海和广州三地的平均温度

表 5-3 是北京、上海、广州三地的平均温度(数据来源: www.weatherbase.com)。

表 5-3 北京、上海、广州三地的平均温度

平均温度													单位: °C
	全年	1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
北京	12	-3	0	6	13	20	24	26	25	20	13	5	-5
上海	16	4	5	8	15	20	23	28	27	23	18	12	6
广州	22	14	15	18	22	26	27	28	28	27	24	20	15

平均最高温度													单位: °C
	全年	1 月	2 月	3 月	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月
北京	17	1	3	11	19	25	29	30	29	25	18	9	2
上海	19	7	8	11	18	23	27	31	30	26	22	16	10
广州	25	17	17	20	25	28	30	32	32	31	27	23	20

相比数据,图形能够更加直观地对比三地之间的气温。同时,我们也可以学习如何在同一个坐标系中绘制多条曲线。本例需同时绘制三条折线,它们之间的区别为各点的 Y 坐标,因此,这个程序需要有 4 个列表: X 坐标、北京平均温度、上海平均温度、广州平均温度,代码如下:

```
In [7]: month = list(range(1, 13))
        beijing = [-3, 0, 6, 13, 20, 24, 26, 25, 20, 13, 5, -5]
        shangh = [4, 5, 8, 15, 20, 23, 28, 27, 23, 18, 12, 6]
        guangz = [14, 15, 18, 22, 26, 27, 28, 28, 27, 24, 20, 15]
        plt.plot(month, beijing, month, shangh, month, guangz)
        plt.legend(['Beijing', 'Shanghai', 'Guangzhou'])
        plt.show()
```

代码运行后的结果如图 5-6 所示。

在这段代码中,函数 plot() 的入参是多个坐标值的列表。此时,它们会两两一组,前一个被视为 X 坐标,后一个被视为 Y 坐标。如果列表是奇数,则最后一组只有 Y 坐标,其 X

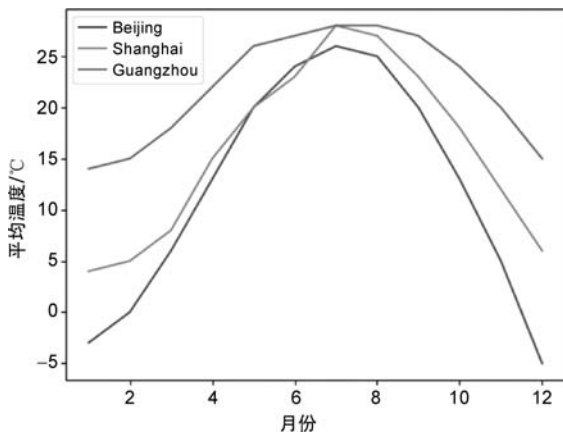


图 5-6 北京、上海、广州三地的月平均温度曲线

坐标使用默认值(从 0 开始,步长为 1)。可以删除示例程序中第 3 个 month,对比一下输出。函数 `legend()` 增加示例图框以标识图上的每幅图形,输入的标签列表需要和函数 `plot()` 中的坐标列表相匹配。还可以为函数 `legend()` 指定第 2 个参数 `loc`,以指定示例框的位置,例如 `lower center`、`center left` 和 `upper left`。参数为 `best` 时,Matplotlib 会自行选择最佳位置以确保图例的位置不会干扰图形。

5.4 绘制函数图形

函数图形是函数的一种重要表达方式,它可以帮助我们理解函数的意义。本节将介绍如何绘制函数图形。

以下代码绘制正弦函数的曲线:

```
In [8]: import matplotlib.pyplot as plt
        from math import sin, pi

        x = []
        y = []
        for i in range(201):
            x_point = 0.01 * i
            x.append(x_point)
            y.append(sin(pi * x_point) * 2)
        plt.plot(x, y, linewidth=1, marker='.',
                 markerfacecolor='k', markersize=8)
        plt.show()
```

与 5.3 节例子不同的地方是,绘制函数图形时所需的数据是在代码运行的过程中生成的。另外,这段代码在调用函数 `plot()` 时,增加了 4 个参数: `linewidth` 用于设置连线的宽度; `marker` 用于设置各点的标识符样式; `markerfacecolor` 用于设置标识符的颜色; `markersize` 用于设置标识符的大小。代码运行后的效果如图 5-7 所示。

通过第 8 段代码可以看到,绘制函数图形的通用步骤如下:

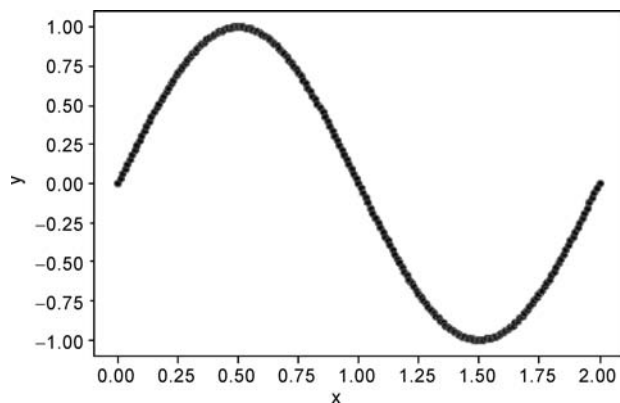


图 5-7 正弦函数曲线

(1) 首先,创建两个空列表。

(2) 然后,利用循环迭代将 x 和 y 的数值添加到列表中。注意两个表中的数据一定要满足函数的映射关系,否则不能正确绘制函数的图形。

(3) 最后,使用绘图函数创建绘图对象。

接下来我们实现一段比较通用的代码,并为式(5-1)绘制函数图形。

$$f(x) = e^{-x} \sin(2\pi x) \quad (5-1)$$

代码如下:

```
In [9]: import matplotlib.pyplot as plt
        from math import pi, sin, exp

        # 修改函数定义,可以绘制不同函数的曲线
        func = lambda x: exp(-x) * sin(2 * pi * x)

        def plot_func(f, start = 0, stop = 100, step = 1, lab = None):
            """
            绘制函数图形

            参数
            ----
            f: 函数
              待绘制的函数
            start: float, 可选项
              自变量的起始值,默认为 0。
            stop: float, 可选项
              自变量的终止值,默认为 100。
            step: float, 可选项
              自变量递增值,默认为 1。
            lab: str, 可选项
              函数说明字符串。

            返回值
            -----
            无。

            """
            l = 'Plot for func ' + lab
```

```

# 生成 x 和 y 的数值列表
x = []
y = []
i = start
while (i < stop):
    x.append(i)
    y.append(f(i))
    i += step

# 绘图
plt.plot(x, y)
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
if (lab == None):
    plt.title('Plot for simple function')
else:
    plt.title(l)
plt.show()

plot_func(func, 0, 4, 0.01, r'$\exp\{-x\}\sin(2\pi\{x\})$')

```

这里自定义的函数 `plot_func()` 可以绘制任意一元函数的图形。函数的 5 个人参分别是：待绘制函数、自变量起始值、自变量终止值、自变量递增值和图形标题。在绘图时，函数 `title()` 为绘制的函数图形添加标题。函数 `xlabel()` 和函数 `ylabel()` 分别为 X 轴和 Y 轴添加说明标签。代码中与标签相关的几个字符串的形式看起来比较特别，X 轴的标签是 `x`，Y 轴标签是 `y`，调用函数 `plot_func()` 传入的函数说明字符串是 `r'$\exp\{-x\}\sin(2\pi\{x\})$'`。这些是 LaTeX 格式的字符串，LaTeX 格式是专业级数学的标准排版方法。字符 `r` 不是 LaTeX 格式的一部分，它是一个 Python 符号，用来标明其后的字符串是一个原始字符串，这样反斜杠 `\` 之后的字符将不被 Python 转义。Matplotlib 有一个内置的 LaTeX 表达式解析器和布局引擎，并提供自己的数学字体。以上这些原始字符串将由 Matplotlib 对其进行转义和设置输出字体。

示例代码中，数学函数由 Lambda 表达式定义。也可以修改这个表达式，尝试着绘制别的函数图形。式(5-1)的部分图形如图 5-8 所示。

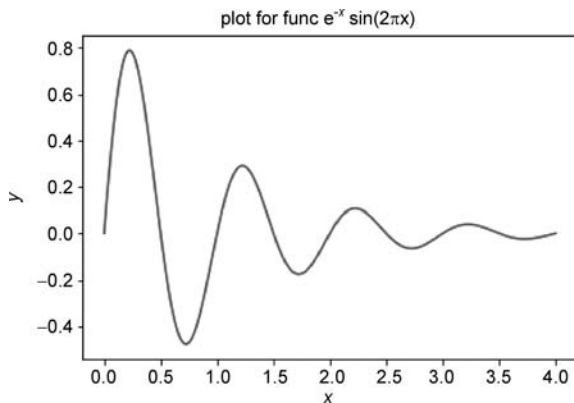


图 5-8 示例函数的图形

5.5 Matplotlib 对象层次结构

Matplotlib 的一个重要概念是它的对象层次结构,“层次结构”意味着在每个绘图之下都有一个类似树的 Matplotlib 对象结构。认识 Matplotlib 的对象层次结构,并使用面向对象的方法,可以更好地控制和自定义绘图。

图 5-9 很好地说明了 Matplotlib 的对象层次结构。该图由 Matplotlib 生成,源代码可以在 Matplotlib 官网搜索关键字 Anatomy of a figure 获得。图 5-9 使用的代码出自 <https://matplotlib.org/3.3.2/gallery/showcase/anatomy.html?highlight=figure%20anatomy>。

图形(Figure)对象是最外层容器,该对象可以包含多个子图(Axes)对象。不同的子图占据图形中不同的区域,它们可能会有交集也可能相互分离。每个子图对象内则包含多个较小的对象,如:刻度标记、线条、图例和文本框。为了表述方便,对只含有一个子图的图形,本书会忽略子图对象和图形对象之间的区别,需读者注意。

来自 Pyplot 的绝大多数函数,例如 `plot()`,要么隐式地引用当前图形和当前子图,要么在不存在的情况下创建它们。当一张图形或者子图被创建后,我们可以使用相应的配置函数修改对象树上每个对象的属性。我们已经使用了函数 `title()`、`xlabel()`、`ylabel()` 和 `legend()` 来修改子图内特定对象的属性。如图 5-9 所示,子图内的每条线条也是一个独立的 Line2D 对象。

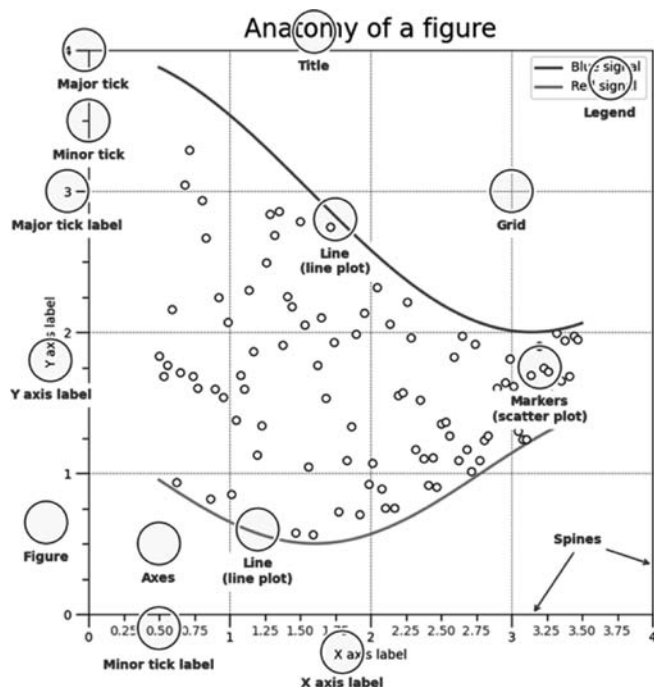


图 5-9 Matplotlib 的对象层次

5.5.1 Line2D 对象

函数 `plot()` 返回的是一个列表,列表的成员为 `Line2D` 对象。我们可以显式地将列表中的成员赋值给独立的变量,然后使用函数 `setp()` 修改每个变量的默认属性。函数 `setp()` 的第一个入参是被修改的对象,随后是所修改的属性列表。修改后的代码如下:

```
In [10]: import matplotlib.pyplot as plt

month = list(range(1, 13))
beijing = [-3, 0, 6, 13, 20, 24, 26, 25, 20, 13, 5, -5]
shangh = [4, 5, 8, 15, 20, 23, 28, 27, 23, 18, 12, 6]
guangz = [14, 15, 18, 22, 26, 27, 28, 28, 27, 24, 20, 15]
bj, sh, gz = plt.plot(month, beijing, month, shangh, month, guangz)
plt.setp(bj, c='k', ls='-', label='Beijing')
plt.setp(sh, c='r', ls='-.', label='Shanghai')
plt.setp(gz, c='b', ls='--', label='Guangzhou')
plt.legend(loc='best')
plt.title('Average temperature in Beijing, Shanghai & Guangzhou')
plt.xlabel('Month')
plt.ylabel('Degrees Celsius')
plt.show()
```

当然,也可以使用函数 `plot()` 依次创建 `Line2D` 对象,这样做的好处是可以在创建时指明对象属性,代码如下:

```
In [11]: import matplotlib.pyplot as plt

month = list(range(1, 13))
beijing = [-3, 0, 6, 13, 20, 24, 26, 25, 20, 13, 5, -5]
shangh = [4, 5, 8, 15, 20, 23, 28, 27, 23, 18, 12, 6]
guangz = [14, 15, 18, 22, 26, 27, 28, 28, 27, 24, 20, 15]
plt.plot(month, beijing, c='k', ls='-', label='Beijing')
plt.plot(month, shangh, c='r', ls='-.', label='Shanghai')
plt.plot(month, c='b', ls='--', label='Guangzhou')
plt.legend(loc='best')
plt.title('Average temperature in Beijing, Shanghai & Guangzhou')
plt.xlabel('Month')
plt.ylabel('Degrees Celsius')
plt.show()
```

函数 `legend()` 输入参数 `loc='best'`, Python 会自动选择图示框的位置。以上两段代码执行后的效果相同,均如图 5-10 所示。

5.5.2 添加文本

细心的读者可能会发现前文的例子中所使用的文字都是英文,这是因为考虑到可能部分读者的默认字体库不支持中文,所以代码中一直未使用中文字符。Matplotlib 支持非西文文本输出,前提是当前使用的字体库内有希望输出的文字。如果当前字体库中没有需要

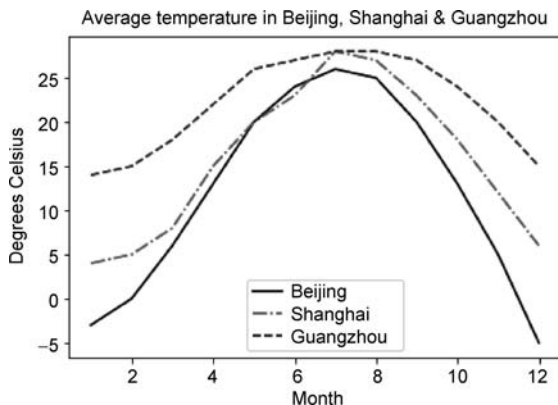


图 5-10 北京、上海和广州月平均温度曲线

输出的字符,则 Matplotlib 会输出一个方块,如图 5-11 所示。

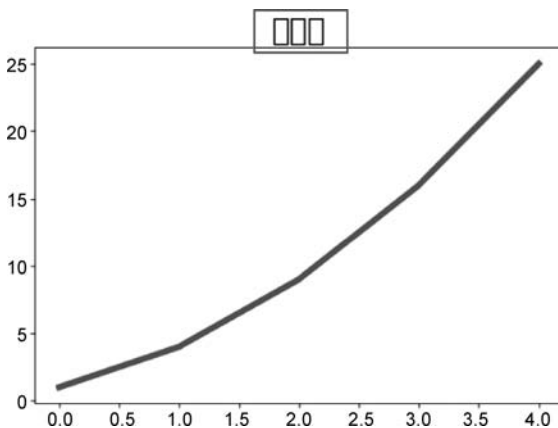


图 5-11 输出不支持的字符

支持中文的常用字体库有“宋体”“黑体”和“楷体”,因此在输出中文之前,只需指定需要的字体库。方法之一是修改全局字体库,参见以下代码:

```
In [12]: import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei'] # 将字体替换为黑体
plt.rcParams['axes.unicode_minus'] = False # 解决坐标轴负数的负号显示问题
plt.plot([0, 1], [1, 2])
plt.xlabel("x 轴")
plt.ylabel("y 轴")
plt.title("标题")
```

此时,绘制的图形便可支持中文输出,如图 5-12 所示。

也可以为不同的标签设置不同的字体。除去各种标签文本之外,Matplotlib 还可以使用函数 `text()` 在子图中的任意位置添加文本。使用函数 `annotate()` 添加特殊形式的标签,代码如下:

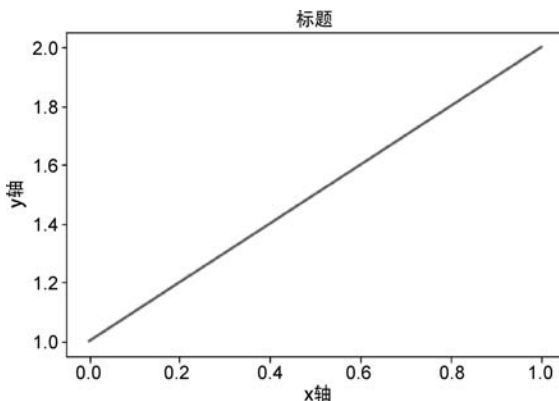


图 5-12 中文标签

```
In [13]: import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
ax.axis([0, 10, 0, 10]) # 设置 x 轴和 y 轴的长度
ax.set_xlabel("黑体 x 轴", fontproperties="SimHei", size=18) # 黑体
ax.set_ylabel("宋体 y 轴", fontproperties="SimSun", size=18) # 宋体
ax.set_title("楷体标题", fontproperties="KaiTi", size=28) # 楷体

# 在(2,6)点添加质能方程
ax.text(2, 6, r'$E = mc^2$', fontsize = 15)
ax.text(4, 0.05, 'X 轴上的彩色文本', verticalalignment = 'bottom',
        color = 'green', fontsize = 15)

# 在(2,1)处绘制一个点, 并为其增加注释
ax.plot([2], [1], 'o')
ax.annotate('注释', xy = (2, 1), xytext = (3, 4),
            arrowprops = dict(facecolor = 'black', shrink = 0.05))
plt.show()
```

函数 `text()` 的前 2 个参数指明文本在子图坐标系中的位置, 第 3 个参数是文本字符串。除此以外还可以使用一些关键字参数设定文本的其他属性, 例如: 字体、大小、颜色等。有时我们需要对图上的一点做特别说明, 此时可以使用函数 `annotate()`。该函数在子图上输出一个带指示线的说明文字。参数 `xy` 通常是子图上需要说明的点, 参数 `xytext` 是文本所在位置, 第 1 个参数是说明文字。参数 `arrowprops` 是一个 `dict` 型对象, 它用于描述指示线的各种属性。

上述代码遵循了面向对象的编码原则, 首先创建图形, 然后创建子图, 接下来创建其他的子对象。建议大家在编程实践中一定要遵循这一原则。

以上代码运行效果如图 5-13 所示。

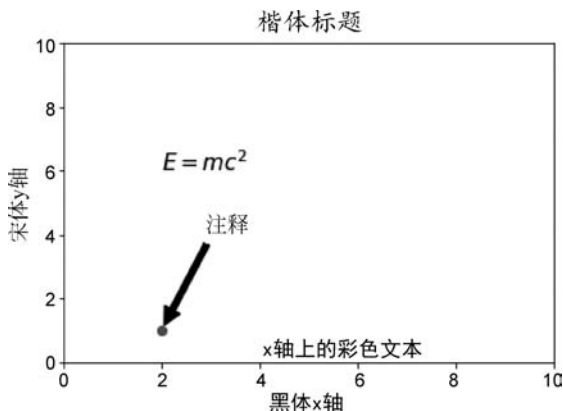


图 5-13 不同字体的标签

5.5.3 多个子图(Axes)

在 Matplotlib 的树状层次结构中,每幅图形对象内可能包含多个子图对象。所有的绘图均是在当前图形的当前子图中完成。如同可以在当前子图内创建多个线条一样,我们也可以在当前图形内创建多个子图。创建多个子图的函数有 `subplot()`、`subplots()` 和 `subplot2grid()`,函数 `subplot()` 使用起来比较简单,函数 `subplots()` 和 `subplot2grid()` 则更加灵活。

1. 函数 `subplot()`

函数 `subplot()` 用于在当前图形中添加一个子图。它的入参的形式是(行数,列数,位置)。行数、列数、位置均是一个正整数。函数 `subplot()` 将图形视为被均匀分割的坐标网格。第一个网格的序号是 1,从左上方开始向右编号,依次递增直至右下角的网格。每个子图占据其中的一些连续网格。参数的具体意义可以参看图 5-14。其中图 5-14(a)是水平分布,图 5-14(b)是垂直分布,图 5-14(c)是 $n \times n$ 的等分阵列(图中 $n=2$)。

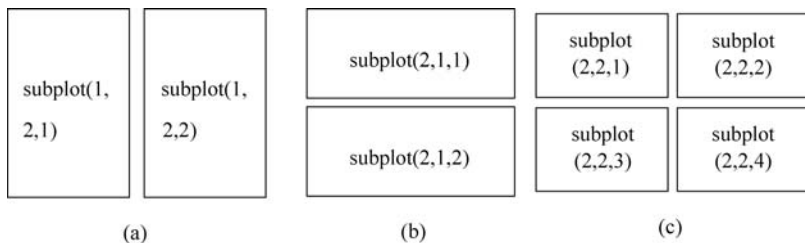


图 5-14 子图分布

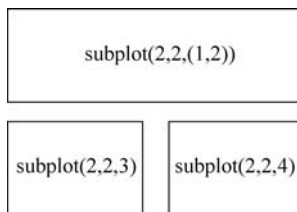


图 5-15 非对称的子图分布

函数 `subplot()` 的第 3 个入参也可以是一个包含两个正整数的元组。它表示这个子图在图形上包含连续的几个网格,这里连续的网格必须在同一行上。这样将会得到非对称的子图分布。函数参数和效果示意如图 5-15 所示。

我们回到北京、上海和广州三地平均温度的例子。现

在向图形中增加月平均最高温度的柱状图。此时,图形中需要增加一个新的子图对象以供柱状图使用。该对象位于月平均温度曲线的下方,代码如下:

```
In [14]: import matplotlib.pyplot as plt

month = list(range(1, 13))
beijing = [-3, 0, 6, 13, 20, 24, 26, 25, 20, 13, 5, -5]
shangh = [4, 5, 8, 15, 20, 23, 28, 27, 23, 18, 12, 6]
guangz = [14, 15, 18, 22, 26, 27, 28, 28, 27, 24, 20, 15]

# 创建 12×8 的图形对象
plt.figure(figsize=(12, 8))

# 创建第一个子图
plt.subplot(211)
bj, sh, gz = plt.plot(month, beijing, month, shangh, month, guangz)
plt.setp(bj, color='k', label='北京')
plt.setp(sh, color='r', ls='-.', label='上海')
plt.setp(gz, color='b', ls=(0,(1,2,4,8)), label='广州')
plt.legend(loc='best')
plt.title('北京、上海和广州的月平均温度')
plt.xlabel('月')
plt.ylabel('摄氏度')

# 创建第二个子图
plt.subplot(212)
bj_h = [1, 3, 11, 19, 25, 29, 30, 29, 25, 18, 9, 2]
sh_h = [7, 8, 11, 18, 23, 27, 31, 30, 26, 22, 16, 10]
gz_h = [17, 17, 20, 25, 28, 30, 32, 32, 31, 27, 23, 20]
w = 0.2
bj_x = list(i - w for i in range(1, 13))
sh_x = month
gz_x = list(i + w for i in range(1, 13))

# 绘制柱状图
plt.bar(bj_x, bj_h, w, label='北京')
plt.bar(sh_x, sh_h, w, hatch='/', label='上海')
plt.bar(gz_x, gz_h, w, hatch='O', label='广州')
plt.legend(loc='best')
plt.title('北京、上海和广州的月平均最高温度')
plt.xlabel('月')
plt.ylabel('摄氏度')
plt.legend(loc='best')

# 重置 x 轴主刻度
x_major_locator = plt.MultipleLocator(1)
ax = plt.gca() # 当前子图为柱状图
print(ax.xaxis.get_major_locator())
ax.xaxis.set_major_locator(x_major_locator)
plt.tight_layout()
plt.show()
```

以上代码执行过程有如下几部分: ①创建图形; ②创建子图; ③在子图内创建线条对

象；④修改子图内对象参数。

函数 `figure()` 用于创建图形对象，入参 `figsize` 指明图形大小。函数 `subplot()` 的 3 个入参之间的逗号也可以被省略。函数 `bar()` 用于柱状图的绘制，它的第 1 个参数是刻度列表，指明每条柱状图中心在 X 轴上的位置，它的第 2 个参数是柱高列表，它的第 3 个参数是柱宽，宽度值不宜过大，否则柱状图会重叠。由于本例中绘制的是分组柱状图，我们使用了关键字参数 `label` 以区别每组中的数据。需特别注意分组柱状图各成员之间的位置关系，否则会出现重叠或间隔。本例中每条柱状图的宽度为 0.2，这样每组 3 个柱的总宽度为 0.6，这个值小于 1，组与组之间在不同刻度之间便不会发生重叠。另外组内成员位置的计算也需注意，否则会产生组内重叠或者间隙过大的情况。本例中绘制的是垂直柱状图，函数 `barh()` 用于绘制水平柱状图。函数 `barh()` 的各输入参数与函数 `bar()` 的意义类似，差别是前 2 个入参对应的坐标轴交换了位置。

子图被创建时，其 X、Y 坐标轴上主刻度的间隔采用系统默认值。当有子对象（如前文示例代码中的 `Line2D` 对象或者柱状图对象）在其内被创建后，Matplotlib 将会依据被创建对象在 X 轴和 Y 轴上的取值范围，设定 X、Y 坐标轴上主刻度的间隔，但是这种自动产生的刻度间隔可能无法满足我们的需求，因此需要在代码中调整。每个二维的子图内会有 X 和 Y 两个轴（Axis）对象，轴类型的成员函数 `set_major_locator()` 可以设置该坐标轴的主刻度（Major Tick）。函数 `set_major_locator()` 的入参是一个 Locator 类型的对象，在代码中由函数 `MultipleLocator()` 产生，函数 `MultipleLocator()` 的入参为期望的 X 轴主刻度间隔。本例中希望 X 轴上的主刻度以 1 为间隔，因此代码中的输入值为 1。函数 `set_minor_locator()` 用于设置从刻度（Minor Tick）间隔。从刻度不同于主刻度，从刻度上没有数值标签。

函数 `tight_layout()` 用于调整子图参数，使子图很好地适合于图形。该函数当前仅考虑了锚定到轴的轴标签、刻度标签、轴标题和示例框。在多数情况下，它能够使包含多个子图的绘图以比较满意的效果呈现出来，但在某些情况下，也可能因绘图过于复杂而无法得到满意的输出效果，此时需要更精细地去调整不同对象的位置参数，这里就不展开了。

2. 函数 `subplots()`

也可以使用函数 `subplots()` 创建图形，这个函数的作用是创建一幅图形和一组子图。函数 `subplots()` 经常使用的入参是 `nrows` 和 `ncols`，用于指明所创建图形中网格的数量。它们的默认值都是 1，也就是说，如果函数 `subplots()` 在调用时没有传入 `nrow` 和 `ncols`，则它创建的图形内仅能有一个子图。

现在我们需要在创建的图形中创建竖排的两个子图，因此输入参数中至少需要有 `nrows=2`。函数 `subplots()` 返回一个图形对象和一个子图数组（NumPy 数组）。我们可以将子图数组赋值给一个变量，此时该变量的类型是子图数组，也可以将数组中的子图对象分别赋值给一组变量。参看以下示例代码：

```
In [15]: import matplotlib.pyplot as plt

month = list(range(1, 13))
beijing = [-3, 0, 6, 13, 20, 24, 26, 25, 20, 13, 5, -5]
shangh = [4, 5, 8, 15, 20, 23, 28, 27, 23, 18, 12, 6]
```

```

guangz = [14, 15, 18, 22, 26, 27, 28, 28, 27, 24, 20, 15]
bj_h = [1, 3, 11, 19, 25, 29, 30, 29, 25, 18, 9, 2]
sh_h = [7, 8, 11, 18, 23, 27, 31, 30, 26, 22, 16, 10]
gz_h = [17, 17, 20, 25, 28, 30, 32, 32, 31, 27, 23, 20]

# 创建图形和子图列表
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(12, 8))

# 获取各子图默认主刻度
print(ax2.xaxis.get_major_locator())
print(ax2.yaxis.get_major_locator())

# 绘制月平均温度曲线
bj, sh, gz = ax1.plot(month, beijing, month, shangh, month, guangz)
plt.setp(bj, color='k', label='北京')
plt.setp(sh, color='r', ls='-.', label='上海')
plt.setp(gz, color='b', ls=(0,(1,2,4,8)), label='广州')
ax1.legend(loc='best')
ax1.set_title('北京、上海和广州的月平均温度')
ax1.set_xlabel('月')
ax1.set_ylabel('摄氏度')
ax1.set_ylabel('Degrees Celsius')

# 绘制月平均最高温度柱状图
w = 0.3
bj_x = list(i - w for i in range(1, 13))
sh_x = month
gz_x = list(i + w for i in range(1, 13))
ax2.bar(bj_x, bj_h, w, label='北京')
ax2.bar(sh_x, sh_h, w, hatch='/', label='上海')
ax2.bar(gz_x, gz_h, w, hatch='O', label='广州')
ax2.set_title('北京、上海和广州的月平均最高温度')
ax2.set_xlabel('月')
ax2.set_ylabel('摄氏度')
ax2.legend(loc='best')

# 显示当前 X 轴和 Y 轴主刻度
print(ax2.xaxis.get_major_locator())
print(ax2.yaxis.get_major_locator())

# 重置 X 轴主刻度
x_major_locator = plt.MultipleLocator(1)
ax2.xaxis.set_major_locator(x_major_locator)
x_major_locator = plt.MultipleLocator(0.1)
ax2.xaxis.set_minor_locator(x_major_locator)
plt.tight_layout()
plt.show()

```

SubPlots()的参数 figsize 用于指明绘图的大小。函数 get_major_locator()是 Axis 类

的成员函数,它可以获得当前轴的主刻度对象。通过调试代码 `print(ax2.xaxis.get_major_locator())` 和 `print(ax2.yaxis.get_major_locator())`,我们可以查看子图被创建后,各坐标轴上主刻度的默认值。`(ax1, ax2)` 将函数 `subplots()` 返回的子图数组中的每个子图对象分别赋值给两个变量,后续的代码分别对它们进行操作。这种面向对象的方法使得程序结构更加清晰,维护起来也会更加方便。以上两种方法创建并实现的代码运行结果如图 5-16 所示。

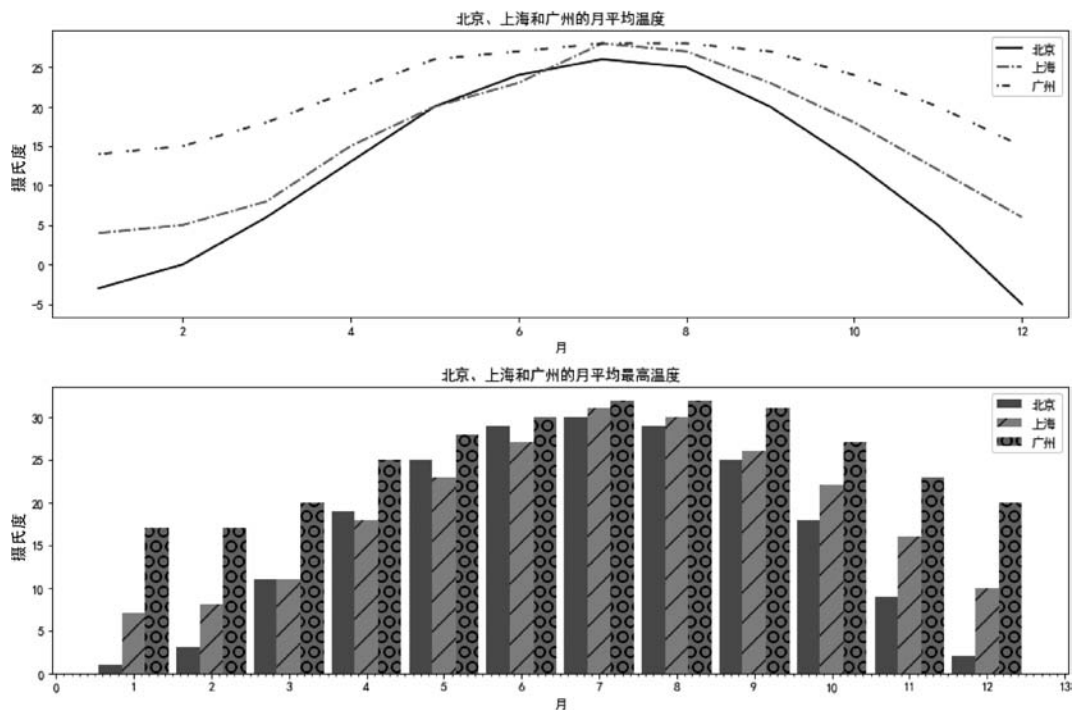


图 5-16 北京、上海和广州月平均温度和月平均最高温度对比图

3. 函数 `subplot2grid()`

函数 `subplot2grid()` 可以跨越多行和多列,因此可以更加灵活地创建子图,它的函数原型如下:

```
subplot2grid(shape, location, rowspan, colspan)
```

参数 `shape` 是一个含有两个整数的元组,指明子图占据的栅格阵列大小。参数 `location` 也是一个含有两个整数的元组,指明子图在栅格阵列中的起始栅格。参数 `rowspan` 表示子图向下跨越的行数。参数 `colspan` 表示子图向右跨越的列数。参数 `rowspan` 和参数 `colspan` 的默认值都是 1。以下代码对如何使用该函数进行了说明:

```
In [16]: import matplotlib.pyplot as plt
```

```
def annotate_axes(fig):
```

```
    """
```

```
    本函数修改子图参数并在子图中心位置添加说明文本
```

```

"""
for i, ax in enumerate(fig.axes):
    ax.set_xticks([])
    ax.set_yticks([])
    ax.text(0.5, 0.5, "子图 %d" % (i + 1),
           va = "center", ha = "center")

fig = plt.figure()
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))

annotate_axes(fig)

plt.show()

```

自定义函数 `annotate_axes()` 依次修改各子图参数并在其中增添文本。函数 `subplot2grid()` 所创建的子图会被添加到图形对象的子图数组 `axes` 中。for 循环中 Python 内嵌函数 `enumerate()` 被用来迭代这个数组, 变量 `i` 被用作子图索引, `ax` 在每次循环中是每个子图的副本。子图(Axes)类的成员函数 `set_xticks()` 和 `set_yticks()` 的入参是一个空列表, 因此 X 轴和 Y 轴上的刻度均被清除, 这样输出的将是一个矩形框。每个子图在创建时, X 轴和 Y 轴默认的最大刻度为 1, 因此函数 `text()` 前 2 个入参 0.5 表示文本将出现在子图的正中心。"子图%d" % (i+1) 是格式字符串, 详见 2.2.7 节。参数 `va` 和 `ha` 用来指定文本的对齐方式, 代码中设定文本在水平方向和垂直方向均中心对齐。代码中将创建的图形划分成 3×3 的栅格阵列。第 1 幅子图从第 1 行第 1 个栅格开始, 向右横跨 3 列。第 2 幅子图从第 2 行第 1 个栅格开始, 向右横跨 2 列。第 3 幅子图从第 2 行第 3 个栅格开始, 向下横跨 2 行。第 4 幅子图从第 3 行第 1 个栅格开始, 向右横跨 1 列。第 5 幅子图从第 3 行第 2 个栅格开始, 向右横跨 1 列。这里需要注意的是, 函数 `subplot2grid()` 的第 1 个入参表示子图所在图形的栅格阵列的大小, 因此所创建的子图应该相同。

以上代码的执行效果如图 5-17 所示。



图 5-17 函数 `subplot2grid()` 创建子图效果

5.6 字典(Dictionary)类型

5.5.2 节的示例代码中函数 `annotate()` 的参数 `arrowprops` 是一个字典型的对象。通过 `type()` 可以查看 `dict(facecolor='black', shrink=0.05)` 的类型,代码如下:

```
In [17]: type(dict(facecolor='black', shrink=0.05))
Out[17]: dict
```

字典是另一种可变容器类型,且可存储任意类型对象。它可以方便地实现关键字查找。假设我们要根据同学的名字来查询其在某次测验中的成绩,如果用 4.2 节介绍过的列表实现,则需要两个列表,示例代码如下:

```
Students = ['Richard', 'Sam', 'Tom']
Score = [99, 90, 76]
```

在这种实现方式中,两个列表中的成员的顺序必须严格匹配。根据姓名查找成绩,首先必须在姓名列中找到该姓名的位置,然后依此序号索引成绩列表。因此,这种方式效率非常低,而且容易出错。

Python 中字典类型因为可以进行关键字查找,所以可以方便利用姓名查找成绩。字典的每个键 `key` 和值 `value` 组成一对,它们之间用冒号“:”分隔,每个键值对之间用逗号“,”分隔,整个字典包括在花括号“{ }”中,格式如下:

```
d = {key1: value1, key2: value2, ... }
```

字典索引和列表类似,其区别是索引时使用的是关键字而非索引号,格式如下:

```
d[keyn]
```

现在,我们将学生成绩列表修改如下:

```
In [18]: scores = {'Richard':99, 'Sam':90, 'Tom':76}
          print(scores['Richard'])
99
```

键应该是唯一的,否则,最后出现的键值对将会替换前面的键值对,代码如下:

```
In [19]: scores = {'Richard':99, 'Sam':90, 'Tom':76, 'Sam':60}
          print(scores['Sam'])
60
```

由于关键字'Sam'被多次使用,因此只有最后一组的值被保留下来。

5.7 本章小结

本章介绍了 Matplotlib 的基本使用方法。内容包括：

- (1) 如何使用 Matplotlib 绘制曲线图、散点图和柱状图。
- (2) 如何增添文本标签。
- (3) 如何指定字体库。
- (4) 如何编写通用函数绘制数学函数曲线。
- (5) Matplotlib 的对象层次, 如何使用面向对象的方法绘制图形。

本章是后续章节的基础, 关于 Matplotlib 绘图的更多的内容会在后续章节中继续深入介绍。

5.8 练习

练习 1:

在一个大小为 8×4 的图上绘制函数 $f(x) = x^2$ 的曲线, x 在区间 $[-3, 3]$ 取值, 颜色是红色, 线的粗细为 2 (默认为 1)。

练习 2:

在一个大小为 8×4 的图上绘制函数 $f(x) = \sin^2(x-2)e^{-x^2}$ 的曲线, x 在区间 $[0, 2]$ 取值, 颜色是黑色, 线的粗细为 2 (默认为 1)。

练习 3:

假设我们扔出去一个球, 它的飞行距离由初速度和抛射的角度决定。假设球从水平地面抛出, 绘制该球在不同仰角和初速度下的运动轨迹。

编程提示:

假设球的发射初速度是 u , 仰角 θ 。仰角采用角度值, 速度单位为 km/s, 重力加速度 $g = 9.81 \text{ m/s}^2$ 。

球的上升时间可由式(5-2)算出:

$$t = \frac{u \sin \theta}{g} \quad (5-2)$$

球飞行轨迹上的每个点可由式(5-3)算出:

$$x = t' u \cos \theta, \quad y = u \sin(\theta t') \frac{gt'^2}{2} \quad (5-3)$$

其中 t' 在区间 $[0, 2t]$ 取值。

练习 4:

单峰映射(Logistic Map)可以构建一个数字序列, 它的数学形式如式(5-4)所示。

$$x_{n+1} = rx_n(1 - x_n) \quad (5-4)$$

其中 x_n 是介于 0 和 1 之间的数, r 是正数。

现在完成以下编程任务:

- (1) 选定 x_0 和 r , 计算数列中前 N 个数值。
- (2) 取 $x_0=0.5$, 计算在 $r=1.5$ 和 $r=3.5$ 时数列的前 2000 个数。使用最后 100 个数据绘图, 对比分析两种情况下图形数据的趋势。
- (3) 取 $x_0=0.5$, r 从 1 开始递增至 4, 每次增值为 0.01。计算 r 取不同值时, 数列的前 2000 个数值, 并绘制它们。X 轴对应 r , Y 轴对应数列中的值。