1. Matplotlib库简介

Matplotlib是一个基于Python的绘图库，它提供了一套和MATLAB相似的命令API，广泛应用于科学计算、数据分析、机器学习等领域，便于用户快速上手。

其特点为：

（1）易于上手：Matplotlib的API设计简洁，绘图流程清晰，初学者可以快速掌握。

（2）高度可定制：Matplotlib提供了丰富的绘图参数，用户可以根据需求调整图形样式、颜色、字体等。

（3）支持多种数据源：Matplotlib可以处理多种数据格式，如列表、数组、Pandas数据框等。

（4）跨平台：Matplotlib支持Windows、Linux和macOS等多个操作系统。

我们使用Matplotlib库时，更多的使用的是Matplotlib库下的`pyplot`模块，`pyplot`模块包含了大量的绘图函数，这些函数能够快速地创建各种类型的图表，包括线图、散点图、条形图、饼图等，并且Matplotlib库在绘制图像时，可以很好的与Numpy库进行结合使用。

> 注意：在实际使用的过程中，通常将matplotlib.pyplot缩写为plt。

2. Matplotlib库的安装

Matplotlib库是一个第三方库，因此在使用时需要我们提前安装，安装的命令是：

```
pip install matplotlib
```

3. 线图

在Matplotlib库中，绘制线图所用的函数为`plot`函数，函数原型为：

```
plt.plot(x, y, fmt, **kwargs)
```

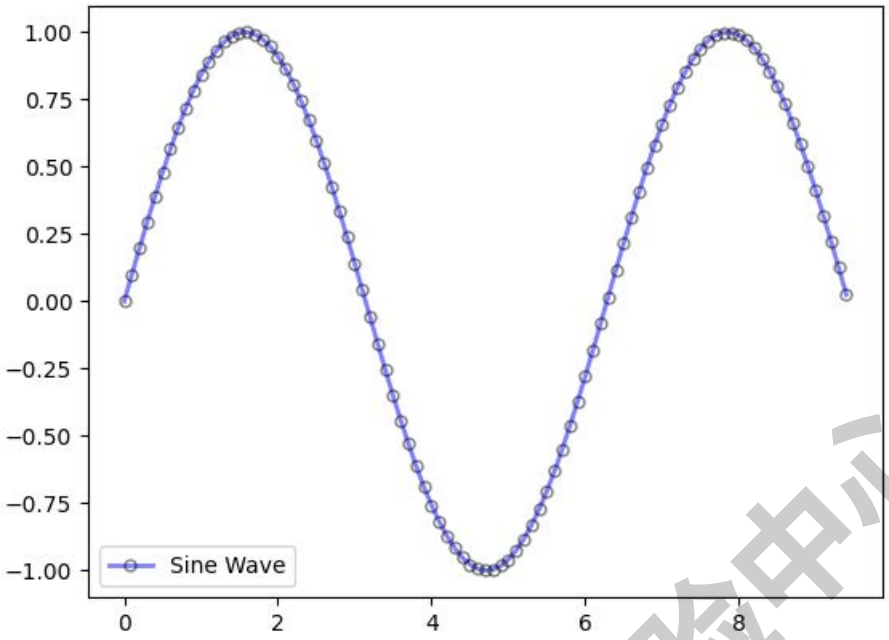
- `x`: 这个参数是数据点的 x 轴坐标，可以是一个列表或者数组。如果 `x` 没有被指定，那么它默认为 `range(len(y))`。
- `y`: 这个参数是数据点的 y 轴坐标，同样可以是一个列表或者数组。

- `fmt`：这是一个可选的字符串参数，用于定义图形的颜色和样式。它由以下部分组成：
  - `颜色`： 例如 'r' 代表红色，'g' 代表绿色等。
  - `标记`： 例如 'o' 代表圆圈，'s' 代表正方形等。
  - `线型`： 例如 '-' 代表实线，'--' 代表虚线，':'代表点等。
- `**kwargs`：这是可选的关键字参数，用于进一步自定义图形的属性。常用的关键字参数包括：
  - `label`：图例的标签。
  - `linewidth` 或 `lw`：线宽。
  - `color` 或 `c`：线的颜色。
  - `marker`：标记样式，如 'o'，'s' 等。
  - `markersize` 或 `ms`：标记的大小。
  - `markeredgecolor`：标记边缘的颜色。
  - `markeredgewidth`：标记边缘的宽度。
  - `markerfacecolor`：标记内部的颜色。
  - `alpha`：透明度。

字符	类型	字符	类型
-	实线型	v	下三角型
--	虚线型	^	上三角型
-.	虚点线型	<	左三角型
:	点型	>	右三角型
.	句号型	1	下三叉型
_	横线	2	上三叉型
o	圆点型	3	左三叉型
s	正方型	4	右三叉型
p	五角型	*	星型
h	六边形1	H	六边形2
+	加号型	x	乘号型
D	菱形	d	瘦菱形

3.1 正弦图像

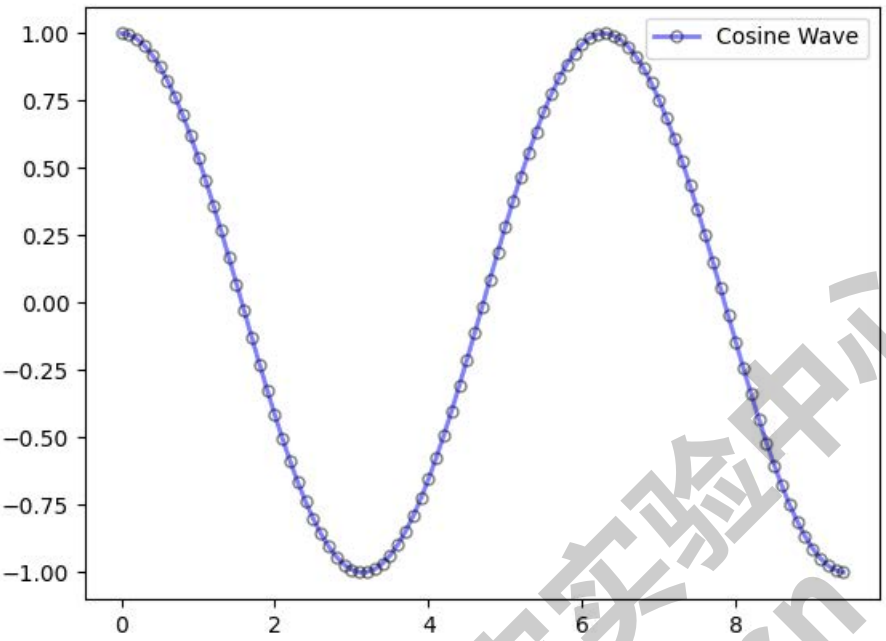
```
import numpy as np
import matplotlib.pyplot as plt

# 计算曲线上的点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# 使用Matplotlib绘制点，并添加fmt和kwargs属性
plt.plot(x, y, '-',
         label='Sine Wave',  # 图例标签
         linewidth=2,  # 线宽
         color='blue',  # 线的颜色
         marker='o',  # 标记样式
         markersize=5,  # 标记的大小
         markeredgecolor='black',  # 标记边缘的颜色
         markeredgewidth=1,  # 标记边缘的宽度
         markerfacecolor='none',  # 标记内部的颜色
         alpha=0.5 # 透明度
         )

# 显示图例
plt.legend()

# 显示图形
plt.show()
```



3.2 余弦图像

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

# 计算曲线上的点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y = np.cos(x)

# 使用Matplotlib绘制点，并添加fmt和kwargs属性
plt.plot(x, y, '-',
         label='Cosine Wave',  # 图例标签
         linewidth=2,  # 线宽
         color='blue',  # 线的颜色
         marker='o',  # 标记样式
         markersize=5,  # 标记的大小
         markeredgecolor='black',  # 标记边缘的颜色
         markeredgewidth=1,  # 标记边缘的宽度
         markerfacecolor='none',  # 标记内部的颜色
         alpha=0.5 # 透明度
         )

# 显示图例
plt.legend()
```

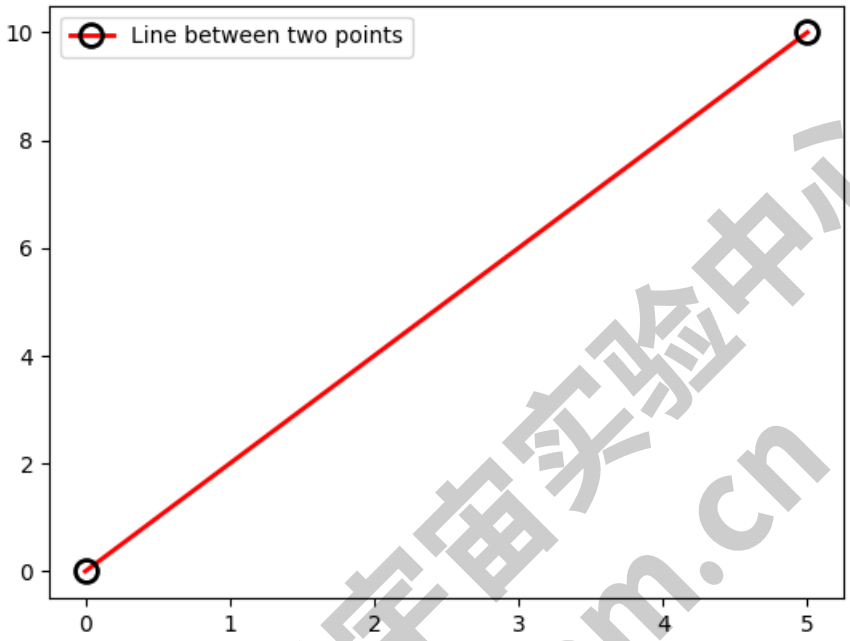
```
# 显示图形
plt.show()
```



3.3 直线图

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

# 只有两个点的x和y坐标
x = np.array([0, 5])  # 两个点的x坐标
y = np.array([0, 10])       # 对应的y坐标

# 使用Matplotlib绘制两个点的直线，并添加kwargs属性
plt.plot(x, y, '-',
         label='Line between two points',  # 图例标签
         linewidth=2,  # 线宽
         color='red',  # 线的颜色
         marker='o',  # 标记样式
         markersize=10,  # 标记的大小
         markeredgecolor='black',  # 标记边缘的颜色
         markeredgewidth=2,  # 标记边缘的宽度
         markerfacecolor='none',  # 标记内部的颜色
         alpha=1.0  # 透明度
         )
```

```
# 显示图例
plt.legend()

# 显示图形
plt.show()
```



4. 散点图

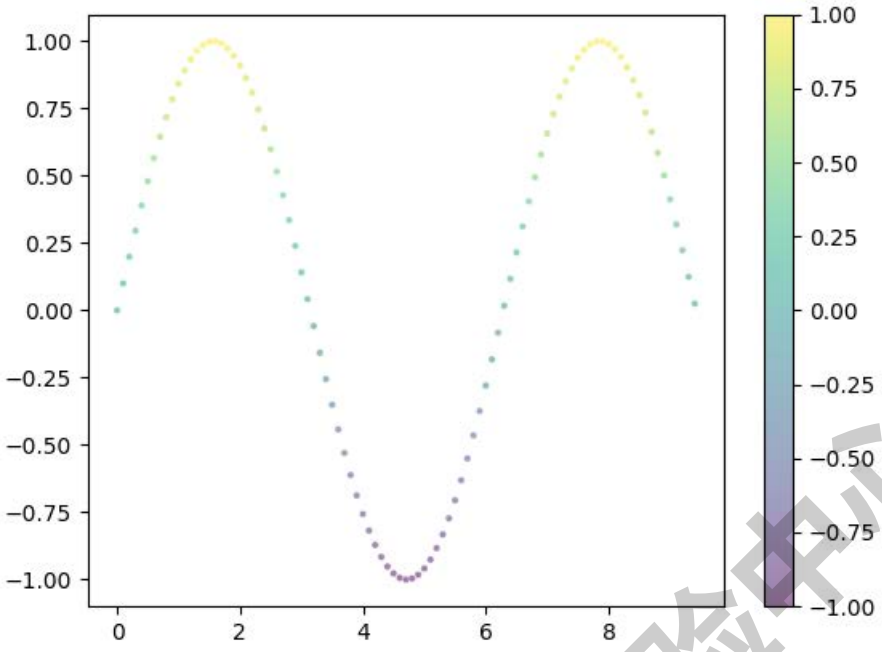
在Matplotlib库中，使用`scatter`来绘制散点图，函数原型为：

```
matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None,
cmap=None, norm=None, vmin=None, vmax=None, alpha=None,
linewidths=None, edgecolors=None, **kwargs)
```

- `x`, `y`: 数组或标量，代表散点图中每个点的 x 和 y 坐标。
- `s`: 散点的大小。可以是一个标量或数组，用于指定每个点的大小。
- `c`: 散点的颜色。可以是单个颜色，也可以是数组，为每个点指定颜色。如果提供了 `cmap` 参数，则 `c` 可以是颜色映射的值。
- `marker`: 散点的标记样式，例如 `'o'` 表示圆圈，`'s'` 表示正方形。
- `cmap`: Colormap，用于将 `c` 中的数值映射到颜色。
- `norm`: 用于标准化 `c` 的颜色数据的 Normalize 对象，确保数据在指定的范围内映射到颜色映射。
- `vmin`, `vmax`: 分别是 `c` 中数据的最小值和最大值，用于色彩映射。
- `alpha`: 透明度，范围从 0（完全透明）到 1（完全不透明）。

- `linewidths`：散点边缘的线宽。
- `edgecolors`：散点边缘的颜色。

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

# 计算正弦曲线上的点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# 创建一个颜色数组，基于y值映射到颜色
colors = y

# 绘制散点图，添加所有属性
plt.scatter(x, y,
            s=10,                # 散点的大小
            c=colors,            # 散点的颜色，这里使用y值映射颜色
            marker='o',          # 散点的标记样式
            cmap='viridis',      # 颜色映射
            norm=None,           # 默认的标准化
            vmin=-1,             # 颜色映射的最小值
            vmax=1,              # 颜色映射的最大值
            alpha=0.5,           # 透明度
            linewidths=0.5,      # 散点边缘的线宽
            edgecolors='w'       # 散点边缘的颜色
            )

# 添加颜色条
plt.colorbar()

# 显示图形
plt.show()
```



5. 条形图

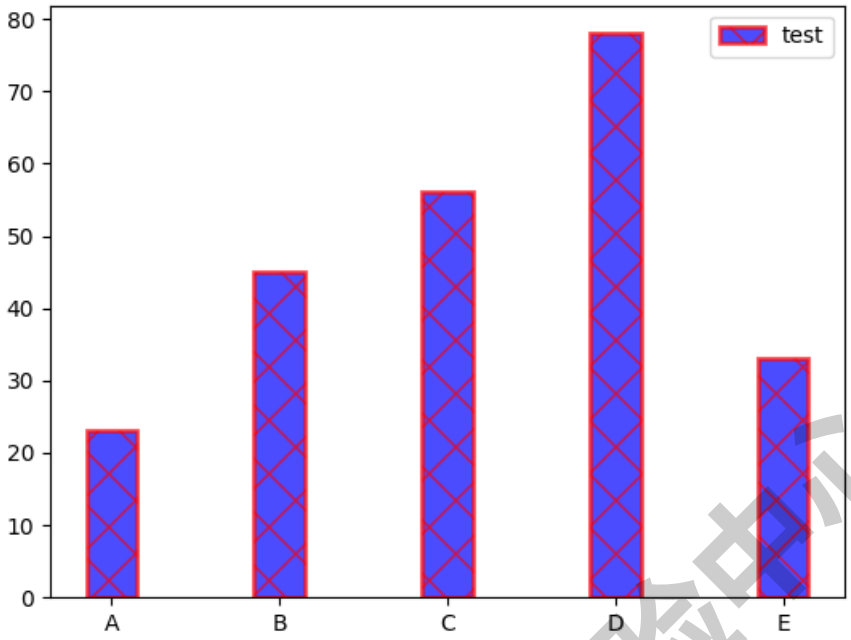
生成条形图（也叫柱状图），使用的函数为`bar`，其函数原型为：

```
matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *,
align='center', data=None, **kwargs)
```

- `x`：条形中心的x坐标。这个参数可以是单个值或者一个列表，表示每个条形的中心位置。
- `height`：条形的高度。这通常是一个数值或者一个列表，表示每个条形的高度。
- `width`：条形的宽度，默认为0.8。这可以是单个值，用于所有条形，或者一个列表，用于指定每个条形的宽度。
- `bottom`：条形的起始位置，默认为0。这可以是单个值或者一个列表，用于堆叠条形图。
- `align`：条形的对齐方式，可以是'center'（中心对齐，默认），'edge'（左对齐）。
- `data`：如果提供了`data`参数，则x和height参数可以指定为字符串，这些字符串将被解释为pandas DataFrame列的名称。
- `**kwargs`：其他关键字参数，包括：
  - `color` 或 `facecolor`：条形的填充颜色。
  - `edgecolor` 或 `linewidth`：条形边缘的颜色和线宽。

- `linestyle`: 条形边缘的线型，例如`'-'`, `'--'`, `'-.'`, `':'`。
- `alpha`: 条形的透明度，范围从0（完全透明）到1（完全不透明）。
- `hatch`: 条形的填充图案，例如`'/'`, `'\\'`, `'|'`, `'-'`, `'+'`, `'x'`, `'o'`, `'O'`, `'*'`。
- `picker`: 控制条形是否可以被交互式选择。
- `log`: 如果设置为True，则条形的高度将以对数尺度表示。
- `label`: 为条形创建图例时使用的标签。

```
import matplotlib.pyplot as plt

# 数据
labels = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 78, 33]

# 创建条形图，并添加关键字参数
plt.bar(labels, values,
        width=0.3,                  # 条形的宽度
        color='b',                  # 条形的填充颜色
        edgecolor='r',              # 条形边缘的颜色
        linewidth=2,                # 条形边缘的线宽
        linestyle='-',              # 条形边缘的线型
        alpha=0.7,                  # 条形的透明度
        hatch='x',                  # 条形的填充图案
        align='center',             # 条形与x位置的对齐方式
        label='test'                # 为条形创建图例时使用的标签
       )

# 显示图例
plt.legend()

# 显示图表
plt.show()
```



如果想要显示中文图例的话，可以设置一下中文字体：

```
import matplotlib.pyplot as plt

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

# 数据
labels = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 78, 33]

# 创建条形图，并添加关键字参数
plt.bar(labels, values,
        width=0.3,                  # 条形的宽度
        color='b',                  # 条形的填充颜色
        edgecolor='r',              # 条形边缘的颜色
        linewidth=2,                # 条形边缘的线宽
        linestyle='-',              # 条形边缘的线型
        alpha=0.7,                  # 条形的透明度
        hatch='x',                  # 条形的填充图案
        align='center',             # 条形与x位置的对齐方式
        label='条形图'              # 为条形创建图例时使用的标签
       )

# 显示图例
```

```
plt.legend()

# 显示图表
plt.show()
```

6. 饼图

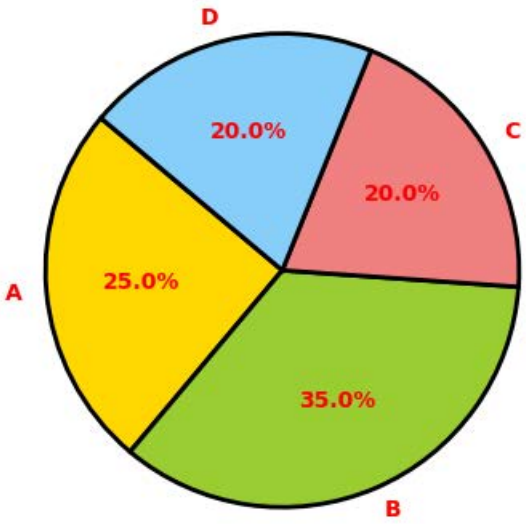
在Matplotlib库中使用`pie`来创建饼图，其函数原型为：

```
plt.pie(x, explode=None, labels=None, colors=None, autopct=None,
startangle=0, shadow=False, radius=1, wedgeprops=None,
textprops=None, center=(0, 0), frame=False, rotatelabels=False,
**kwargs)
```

- `x`：饼图中每个扇形的尺寸。它应该是一个非负的数组或序列。
- `explode`：一个与 `x` 相同长度的数组，用于指定每个扇形是否突出显示。如果 `explode[i]` 不为 0，则第 `i` 个扇形将与其他扇形分离。
- `labels`：一个字符串列表，用于指定每个扇形的标签。
- `colors`：一个颜色列表，用于指定每个扇形的颜色。
- `autopct`：一个格式化字符串，用于在饼图上显示每个扇形的百分比。可以使用 `%1.1f%%` 来显示一个保留一位小数的百分比。
- `startangle`：饼图开始的角度，默认为 0（即从 x 轴正方向开始）。
- `shadow`：布尔值，用于指定是否为饼图添加阴影。
- `radius`：饼图的半径，默认为 1。
- `wedgeprops`：字典，用于指定饼图中每个扇形的属性，如边缘颜色和宽度。
- `textprops`：字典，用于指定饼图中标签的文本属性。
- `center`：一个元组，用于指定饼图的中心位置。
- `frame`：布尔值，用于指定是否为饼图添加一个框。
- `rotatelabels`：布尔值，用于指定是否旋转标签以适应扇形。
- `**kwargs`：其他关键字参数，包括：
  - `normalize`：布尔值，默认为 `True`。如果 `normalize` 为 `True`，则 `x` 中的值将被归一化以使它们的总和等于 1。如果 `normalize` 为 `False`，则 `x` 中的值将直接用于绘制饼图，而不进行归一化处理。

- `hatch`：字符串或列表，用于设置饼图各个部分的阴影或图案填充。如果提供一个字符串，则所有扇形都将使用相同的图案。如果提供一个列表，则每个扇形可以有不同的图案。例如，`hatch='/'` 将为所有扇形添加斜线图案。

```
import matplotlib.pyplot as plt

# 数据
sizes = [25, 35, 20, 20]
labels = ['A', 'B', 'C', 'D']
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']


# 绘制饼图，使用 **kwargs 定制扇形
plt.pie(sizes,                                  # 饼图中每个扇形的尺寸
        explode=[0, 0, 0, 0],                   # 用于指定每个扇形是否突出显示
        labels=labels,                          # 用于指定每个扇形的标签
        colors=colors,                          # 用于指定每个扇形的颜色
        autopct='%1.1f%%',                      # 用于在饼图上显示每个扇形的百
分比
        startangle=140,                         # 饼图开始的角度，默认为 0（即
从 x 轴正方向开始）
        shadow=False,                           # 用于指定是否为饼图添加阴影
        radius=1,                               # 饼图的半径
        wedgeprops=dict(edgecolor='black', linewidth=2,
linestyle='-'),  # 指定饼图中每个扇形的属性
        textprops=dict(color='red', weight='bold'),  # 用于指定饼图
中标签的文本属性，这里指定文本的颜色和字体的粗细
        center=(0, 0),                          # 用于指定饼图的中心位置
        frame=False,                             # 用于指定是否为饼图添加一个框
)
# 显示图表
plt.show()
```



7. 其他常用函数

7.1 xlabel

在matplotlib中，`plt.xlabel()`函数用于为当前活动的坐标轴（Axes）设置x轴的标签。当你想要标识x轴代表的数据或单位时，这个函数非常有用。

```
plt.xlabel('xlabel text')
```

其中`'xlabel text'`是你想要显示在x轴旁边的文本。

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

# 计算曲线上的点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# 使用Matplotlib绘制点，并添加fmt和kwargs属性
plt.plot(x, y,
         label='Sine Wave',  # 图例标签
         linewidth=2,  # 线宽
         linestyle='-',  # 线型
         color='blue',  # 线的颜色
         marker='o',  # 标记样式
```

```
            markersize=5,  # 标记的大小
            markeredgecolor='black',  # 标记边缘的颜色
            markeredgewidth=1,  # 标记边缘的宽度
            markerfacecolor='none',  # 标记内部的颜色
            alpha=0.5 # 透明度
            )

# 显示x轴标签
plt.xlabel('x')

# 显示图例
plt.legend()

# 显示图形
plt.show()
```

7.2 ylabel

在matplotlib中，`plt.ylabel()` 函数用于为当前活动的坐标轴（Axes）设置y轴的标签。当你想要标识y轴代表的数据或单位时，这个函数非常有用。

下面是 `plt.ylabel()` 函数的基本用法：

python

复制

```
plt.ylabel('ylabel text')
```

其中 `'ylabel text'` 是你想要显示在y轴旁边的文本。

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

# 计算曲线上的点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# 使用Matplotlib绘制点，并添加fmt和kwargs属性
plt.plot(x, y,
         label='Sine Wave',  # 图例标签
         linewidth=2,  # 线宽
```

```
          linestyle='-',  # 线型
          color='blue',  # 线的颜色
          marker='o',  # 标记样式
          markersize=5,  # 标记的大小
          markeredgecolor='black',  # 标记边缘的颜色
          markeredgewidth=1,  # 标记边缘的宽度
          markerfacecolor='none',  # 标记内部的颜色
          alpha=0.5 # 透明度
          )

# 显示x轴标签
plt.xlabel('x')

# 显示y轴标签
plt.ylabel('y')

# 显示图例
plt.legend()

# 显示图形
plt.show()
```

7.3 title

在matplotlib中，`title`函数用于为当前活动的坐标轴（Axes）设置图表的标题。标题通常位于图表的顶部中央，用于简要描述图表的内容。

```
plt.title('Chart Title')
```

其中`'Chart Title'`是你想要显示在图表顶部的文本。

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

# 计算曲线上的点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# 使用Matplotlib绘制点，并添加fmt和kwargs属性
plt.plot(x, y,
          label='Sine Wave',  # 图例标签
```

```
            linewidth=2,  # 线宽
            linestyle='-',  # 线型
            color='blue',  # 线的颜色
            marker='o',  # 标记样式
            markersize=5,  # 标记的大小
            markeredgecolor='black',  # 标记边缘的颜色
            markeredgewidth=1,  # 标记边缘的宽度
            markerfacecolor='none',  # 标记内部的颜色
            alpha=0.5 # 透明度
            )

# 显示x轴标签
plt.xlabel('x')

# 显示y轴标签
plt.ylabel('y')

# 显示标题
plt.title('Sine Wave')

# 显示图例
plt.legend()

# 显示图形
plt.show()
```

7.4 subplot

在 Matplotlib 中，`subplot`函数用于在一个图形窗口（`Figure`）中创建多个子图布局，每个子图布局区域对应一个`Axes`对象，通过该`Axes`对象可以进行数据绘制操作，使用 `subplot` 可以在网格中安排这些子图。

```
plt.subplot(nrows, ncols, plot_number)
```

这里：

- `nrows`：子图的行数。
- `ncols`：子图的列数。
- `plot_number`：当前子图的编号，从1开始，从左上角到右下角按行优先顺序递增。

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

# 创建数据
x = np.arange(0, 3 * np.pi, 0.01)
y_sin = np.sin(x)
y_cos = np.cos(x)

# 在第一个位置创建子图
plt.subplot(2, 1, 1)  # 2行1列，第一个子图
plt.plot(x, y_sin)
plt.title('Sine Wave')
plt.xlabel('x')
plt.ylabel('y_sin')

# 在第二个位置创建子图
plt.subplot(2, 1, 2)  # 2行1列，第二个子图
plt.plot(x, y_cos)
plt.title('Cosine Wave')
plt.xlabel('x')
plt.ylabel('y_cos')

# 调用 tight_layout 来自动调整子图参数
plt.tight_layout()

# 显示图形
plt.show()
```


8. Figure

在 Matplotlib 中，`Figure`对象是整个绘图的顶级容器，它是所有绘图元素的基础，提供了一个用于绘制图形的画布空间。

在 Matplotlib 中，`Axes`对象是进行数据绘制和设置坐标轴等操作的核心区域，它与`Figure`对象紧密相关，共同构建了完整的绘图体系。

也就是说，在使用Matplotlib绘图时的第一步就是创建`Figure`对象和`Axes`，这两个对象的创建分两种方式：

1. **隐式创建**：当直接调用绘图函数（如`plt.plot()`）而没有预先创建`Figure`对象时，Matplotlib 会自动隐式创建一个`Figure`对象和一个`Axes`对象。

2. **显式创建**：使用函数来手动创建一个`Figure`对象。

8.1 创建Figure对象

8.1.1 figure

在Matplotlib 中，`figure` 函数用于创建一个新的图形窗口，或者获取当前活跃的图形窗口。每个图形窗口可以包含一个或多个坐标轴（Axes），可以在其上绘制数据。函数原型为：

```
plt.figure(num=None, figsize=None, dpi=None, facecolor=None,
edgecolor=None, frameon=True, clear=False, **kwargs)
```

- `num`: 可选参数，指定图形窗口的编号或名称。如果 `num` 为 `None`，则创建一个新的图形窗口。如果 `num` 已经存在，则获取该图形窗口。
- `figsize`: 可选参数，指定图形窗口的大小，以英寸为单位。默认为 `None`，使用 Matplotlib 的默认值。
- `dpi`: 可选参数，指定图形的分辨率，以点每英寸为单位。默认为 `None`，使用 Matplotlib 的默认值。
- `facecolor`: 可选参数，指定图形窗口的背景颜色。默认为 `None`，使用 Matplotlib 的默认值。
- `edgecolor`: 可选参数，指定图形窗口的边缘颜色。默认为 `None`，使用 Matplotlib 的默认值。
- `frameon`: 可选参数，布尔值，指定是否绘制图形窗口的边框。默认为 `True`。
- `clear`: 可选参数，布尔值，指定是否清除图形窗口中的内容。默认为 `False`。
- `**kwargs`: 其他关键字参数，用于进一步自定义图形窗口的属性。

```
from matplotlib import pyplot as plt

# 创建一个新的图形窗口
fig = plt.figure(figsize=(8, 6), dpi=120)

# 绘制折线
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])

# 显示图形
plt.show()
```

8.1.2 subplots

在Matplotlib中，`subplots`用于创建包含多个子图（subplot）的图形。这允许你在同一个 `Figure` 对象内组织多个图表，从而方便地比较数据或者展示多方面的信息。

```
fig, axs = matplotlib.pyplot.subplots(nrows=1, ncols=1,
sharex=False, sharey=False, squeeze=True, width_ratios=None,
height_ratios=None, subplot_kw=None, gridspec_kw=None)
```

- `nrows=1`: 子图的行数，设置为1，表示创建一个子图的行。
- `ncols=1`: 子图的列数，设置为1，表示创建一个子图的列。
- `sharex=False`: 布尔值，设置为False，表示子图之间不共享x轴刻度。每个子图都有自己的x轴。
- `sharey=False`: 布尔值，设置为False，表示子图之间不共享y轴刻度。每个子图都有自己的y轴。
- `squeeze=True`: 布尔值，设置为True，表示如果只创建了一个子图，则返回一个子图对象而不是一个只包含一个子图对象的数组。
- `width_ratios=None`: 列宽比例的序列，设置为None表示默认比例。如果提供序列，它将用于调整各列的宽度比例。
- `height_ratios=None`: 行高比例的序列，设置为None表示默认比例。如果提供序列，它将用于调整各行的高度比例。
- `subplot_kw=None`: 字典，设置为None，表示没有额外的关键字参数传递给`add_subplot`调用。如果需要，可以传递一个字典来设置子图的属性。
- `gridspec_kw=None`: 字典，设置为None，表示没有额外的关键字参数传递给GridSpec对象。如果需要，可以传递一个字典来控制子图的布局。

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

# 创建数据
x = np.arange(0, 10, 0.01)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.tan(x)
y4 = 1 / (1 + np.exp(-x))  # Sigmoid function

# 创建一个 2x2 的子图网格
fig, axs = plt.subplots(2, 2, figsize=(10, 8))
```

```
# 第一个子图
axs[0, 0].plot(x, y1)
axs[0, 0].set_title('Sine Wave')
axs[0, 0].set_xlabel('X-axis')
axs[0, 0].set_ylabel('Y-axis')

# 第二个子图
axs[0, 1].plot(x, y2)
axs[0, 1].set_title('Cosine Wave')
axs[0, 1].set_xlabel('X-axis')
axs[0, 1].set_ylabel('Y-axis')

# 第三个子图
axs[1, 0].plot(x, y3)
axs[1, 0].set_title('Tangent Wave')
axs[1, 0].set_xlabel('X-axis')
axs[1, 0].set_ylabel('Y-axis')

# 第四个子图
axs[1, 1].plot(x, y4)
axs[1, 1].set_title('Sigmoid Function')
axs[1, 1].set_xlabel('X-axis')
axs[1, 1].set_ylabel('Y-axis')

# 调整子图间距
plt.tight_layout()

# 显示图形
plt.show()
```


8.2 常用的Figure对象的方法

8.2.1 add_subplot

```
fig.add_subplot(nrows, ncols, plot_number)
```

- `nrows`：子图的总行数。
- `ncols`：子图的总列数。
- `plot_number`：当前子图在其所在布局中的位置，从1开始计数，从左上角到右下角。

该函数会返回一个 `Axes` 对象。

```
import matplotlib.pyplot as plt

# 创建一个图形窗口
fig = plt.figure()

# 添加第一个子图，位于2行1列布局中的第1个位置
ax1 = fig.add_subplot(2, 1, 1)
ax1.plot([0, 1], [0, 1])

# 添加第二个子图，位于2行1列布局中的第2个位置
ax2 = fig.add_subplot(2, 1, 2)
ax2.plot([0, 1], [1, 0])

# 显示图形
plt.show()
```

8.2.2 add_axes

该函数允许你在图形（`Figure`）中的任意位置添加一个子图（`Axes`）。与 `subplots` 或 `add_subplot` 不同，`add_axes` 允许你精确地指定子图的位置和大小。

```
matplotlib.figure.Figure.add_axes(rect, **kwargs)
```

- `rect`：一个形如 `[左边界，底边界，宽度，高度]` 的列表，指定子图的位置和大小。这些值是在归一化坐标中的，其中 0 表示图形的左边界，1 表示右边界，0 表示底部，1 表示顶部。
- `**kwargs`：其他可选的关键字参数，用于控制子图的属性。

```
import matplotlib.pyplot as plt

# 创建一个图形
fig = plt.figure()

# 添加一个子图，位置是图形左下角的1/4处，大小为图形宽度的1/2和高度的1/2
ax = fig.add_axes([0.1, 0.1, 0.5, 0.5])

# 使用这个子图绘制一些数据
ax.plot([0, 1], [0, 1])

# 显示图形
plt.show()
```

8.2.3 suptitle

该函数用于在图形（`Figure`）上添加一个中心对齐的标题，这个标题位于所有子图（`Axes`）的顶部。使用 `suptitle` 可以为整个图形设置一个总标题，而不仅仅是单个子图。

```
matplotlib.figure.Figure.suptitle(t, **kwargs)
```

- `t`：字符串，表示要添加的标题文本。
- `**kwargs`：其他可选的关键字参数，用于控制标题的样式，如字体大小（`fontsize`）、颜色（`color`）等。

```
import matplotlib.pyplot as plt

# 创建一个图形
fig = plt.figure()

# 在图形上添加两个子图
ax1 = fig.add_subplot(211)  # 第一个子图，两行一列的第一个位置
ax2 = fig.add_subplot(212)  # 第二个子图，两行一列的第二个位置

# 使用第一个子图绘制一些数据
ax1.plot([0, 1], [0, 1])

# 使用第二个子图绘制一些数据
ax2.plot([0, 1], [1, 0])
```

```
# 为整个图形添加一个总标题
fig.suptitle('Test Suptitle', fontsize=16, color='red')

# 显示图形
plt.show()
```

8.2.4 text

该函数用于在图表的任意位置添加文本，通常在 `Axes` 对象上调用，用于在绘图区域的指定坐标位置上放置文本。

```
matplotlib.axes.Axes.text(x, y, s, **kwargs)
```

- `x, y`：浮点数或数组，指定文本的位置坐标。
- `s`：字符串，要添加的文本内容。
- `**kwargs`：其他可选的关键字参数，用于控制文本的样式，如 `fontsize`（字体大小）、`color`（颜色）、`horizontalalignment`（水平对齐方式，如 'left', 'center', 'right'）、`verticalalignment`（垂直对齐方式，如 'top', 'center', 'bottom'）等。

```
import matplotlib.pyplot as plt

# 创建一个图形和一个子图
fig, ax = plt.subplots()

# 绘制一些数据
ax.plot([0, 1, 2], [0, 1, 0])

# 在坐标 (1, 0.5) 处添加文本
ax.text(1, 0.5, 'Sample Text', fontsize=12, color='red',
        horizontalalignment='center', verticalalignment='center')

# 显示图形
plt.show()
```

8.2.5 axes

在 `matplotlib` 中，如果你想要获取一个图形（`Figure`）中的所有轴（`Axes`），你可以查看 `Figure` 对象的 `axes` 属性。

```
import matplotlib.pyplot as plt

# 创建一个图形和一个子图
fig, ax = plt.subplots(2, 2)

# 获取图形中的所有轴
axes = fig.axes

# axes 是一个包含所有轴的列表
for ax in axes:
    # 包含了每一个轴的x和y的相对位置(从左下角开始计算)、轴的宽度和高度(相对于整个图形)
    print(ax)

# 显示图像
plt.show()
```

8.2.6 get_facecolor

该函数是一个用于获取轴（Axes）或填充区域（Patch）背景色的方法。具体来说，它返回一个表示颜色的 RGBA 元组，其中 RGBA 分别代表红色、绿色、蓝色和 alpha（透明度）值。这些值通常在 0 到 1 的范围内。

```
import matplotlib.pyplot as plt

# 创建一个图形
fig = plt.figure(figsize=(8, 6), facecolor='green')

# 获取背景色
facecolor = fig.get_facecolor()
print(facecolor)

# 显示图像
plt.show()
```

8.2.7 get_dpi

该方法用于获取图形（Figure）对象的分辨率。

```
import matplotlib.pyplot as plt

# 创建一个图形对象
fig = plt.figure()

# 获取图形的DPI
dpi = fig.get_dpi()
print(f"The DPI of the figure is: {dpi}")

# 显示图像
plt.show()
```

8.2.8 get_gca

该函数是 “get current axis” 的缩写，它用于获取当前图形（figure）中的当前坐标轴（axis）对象。这个函数非常有用，因为它允许你访问并修改当前活动的坐标轴的属性。

```
import matplotlib.pyplot as plt

# 创建一个图形和坐标轴
plt.figure()
plt.plot([1, 2, 3], [1, 2, 3])

# 获取当前坐标轴
ax = plt.gca()

# 使用坐标轴对象进行自定义
ax.set_title('Sample Plot')
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')

# 显示图形
plt.show()
```

8.2.9 get_label

该方法用于获取坐标轴（例如，线和柱状图）的标签。这个方法通常用于获取已经设置好的标签，以便于后续的检查或修改。

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

```
# 创建一些数据
x = [1, 2, 3]
y = [1, 2, 3]

# 绘制线图并设置标签
line, = plt.plot(x, y, label='Line 1')

# 获取线的标签
label = line.get_label()
print('Label of the line:', label)

# 显示图形和图例
plt.legend()
plt.show()
```

8.2.10 get_size_inches

该方法用于获取图形的大小，以英寸为单位。

```
import matplotlib.pyplot as plt

# 创建一个图形对象，可以指定大小（宽度，高度）单位为英寸
fig = plt.figure(figsize=(8, 6))

# 使用 get_size_inches() 方法获取图形的大小
size_in_inches = fig.get_size_inches()
print('Size of the figure in inches:', size_in_inches)

# 绘制一些数据
plt.plot([1, 2, 3], [1, 2, 3])

# 显示图形
plt.show()
```

8.2.11 set_size_inches

该方法用于设置图形对象的大小，单位为英寸。这个方法允许用户在创建图形后改变其尺寸，从而影响输出图像的分辨率和显示效果。

```
matplotlib.figure.Figure.set_size_inches(width, height, forward=True)
```

- `width, height`：浮点数，指定图形的宽度和高度，单位为英寸。
- `forward`：布尔值，默认为 `True`。如果为 `True`，则在调整图形大小时，同时更新所有子图的大小和位置。如果为 `False`，则仅改变图形的大小，而不更新子图。

```
import matplotlib.pyplot as plt

# 创建一个图形对象
fig = plt.figure(figsize=(2, 2))

# 使用 set_size_inches 方法设置图形的大小
fig.set_size_inches(10, 6)  # 设置宽度为10英寸，高度为6英寸

# 绘制一些数据
plt.plot([1, 2, 3], [1, 2, 3])

# 显示图形
plt.show()
```

8.2.12 set_dpi

该方法用于设置图形对象的分辨率，单位为每英寸点数（Dots Per Inch, DPI）。这个方法影响输出图像的质量和文件大小，通常与 `set_size_inches` 方法配合使用来精确控制图形的尺寸和输出质量。

```
matplotlib.figure.Figure.set_dpi(dpi)
```

- `dpi`：一个浮点数或整数，指定图形的分辨率，即每英寸的点数。

```
import matplotlib.pyplot as plt

# 创建一个图形对象
fig, ax = plt.subplots()

# 设置图形的尺寸为 6x4 英寸
fig.set_size_inches(6, 4)

# 使用 set_dpi 方法设置图形的分辨率
```

```
fig.set_dpi(100)  # 设置分辨率为 100 DPI

# 绘制一些数据
ax.plot([1, 2, 3], [1, 2, 3])

# 显示图形
plt.show()
```

8.2.13 tight_layout

该函数用于自动调整子图参数，使之填充整个图像区域，同时确保子图之间的标签和标题不会重叠。这个方法对于创建布局整齐的图形非常有用，尤其是在子图较多或标签较长时。

```
import matplotlib.pyplot as plt

# 创建一个包含多个子图的图形
fig, axs = plt.subplots(2, 2)

# 在每个子图中绘制一些数据
axs[0, 0].plot([1, 2, 3], [1, 2, 3])
axs[0, 1].plot([1, 2, 3], [3, 2, 1])
axs[1, 0].plot([1, 2, 3], [1, 3, 2])
axs[1, 1].plot([1, 2, 3], [2, 1, 3])

# 使用 tight_layout 自动调整子图布局
fig.tight_layout()

# 显示图形
plt.show()
```


8.2.14 subplots_adjust

该函数允许用户手动调整子图之间的空间，以及子图与图形边缘之间的空间。这个方法提供了比 `tight_layout` 更细粒度的控制，特别适用于需要精确控制布局的情况。

```
fig.subplots_adjust(left=None, bottom=None, right=None, top=None,
wspace=None, hspace=None)
```

- `left`: 子图区域左边缘与图形左边缘之间的距离，范围从 0 到 1。

- `bottom`: 子图区域下边缘与图形下边缘之间的距离，范围从 0 到 1。
- `right`: 子图区域右边缘与图形右边缘之间的距离，范围从 0 到 1。
- `top`: 子图区域上边缘与图形上边缘之间的距离，范围从 0 到 1。
- `wspace`: 子图之间的水平间距，范围从 0 到 1。
- `hspace`: 子图之间的垂直间距，范围从 0 到 1。

```
import matplotlib.pyplot as plt

# 创建一个包含多个子图的图形
fig, axs = plt.subplots(2, 2)

# 在每个子图中绘制一些数据
axs[0, 0].plot([1, 2, 3], [1, 2, 3])
axs[0, 1].plot([1, 2, 3], [3, 2, 1])
axs[1, 0].plot([1, 2, 3], [1, 3, 2])
axs[1, 1].plot([1, 2, 3], [2, 1, 3])

# 使用 subplots_adjust 手动调整子图布局
fig.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1,
hspace=0.5, wspace=0.5)

# 显示图形
plt.show()
```

8.2.15 clear

在`matplotlib`库中，`Axes`对象（代表绘图区域）和`Figure`对象（代表整个图形）都有`clear`函数。`Axes`对象的`clear`函数主要用于清除该轴（子图）上的所有图形元素，包括线条、标记、文本、图像等；`Figure`对象的`clear`函数用于清除整个图形中的所有子图和其他相关元素

```
import matplotlib.pyplot as plt

# 创建一个包含多个子图的图形
fig, axs = plt.subplots(2, 2)

# 在每个子图中绘制一些数据
axs[0, 0].plot([1, 2, 3], [1, 2, 3])
axs[0, 1].plot([1, 2, 3], [3, 2, 1])
axs[1, 0].plot([1, 2, 3], [1, 3, 2])
```

```
axs[1, 1].plot([1, 2, 3], [2, 1, 3])

# 使用 tight_layout 自动调整子图布局
fig.tight_layout()

# Axes对象的clear
axs[0, 0].clear()

# Figure对象的clear
fig.clear()

# 显示图形
plt.show()
```

8.3 其他函数

8.3.1 clf

与Figure对象的clear方法的作用类似，但clf归属于pyplot库，而不是Figure对象的方法。

```
import matplotlib.pyplot as plt

# 创建一个包含多个子图的图形
fig, axs = plt.subplots(2, 2)

# 在每个子图中绘制一些数据
axs[0, 0].plot([1, 2, 3], [1, 2, 3])
axs[0, 1].plot([1, 2, 3], [3, 2, 1])
axs[1, 0].plot([1, 2, 3], [1, 3, 2])
axs[1, 1].plot([1, 2, 3], [2, 1, 3])

# 使用 tight_layout 自动调整子图布局
fig.tight_layout()

# Axes对象的clear
axs[0, 0].clear()

# pyplot库下的clf()
plt.clf()
```

```
# 显示图形
plt.show()
```

8.3.2 gcf

该函数用于返回当前活动的图形（figure）对象。

```
matplotlib.pyplot.gcf()
```

- 返回当前活动的 `Figure` 对象。如果没有活动的图形，这个函数将创建一个新的图形，并将其作为当前活动的图形返回。

使用场景：

- 当你需要在脚本中获取或操作当前图形时，`plt.gcf()` 非常有用。
- 它允许你访问和修改当前图形的属性，比如大小、标题、背景颜色等。
- 你可以使用返回的 `Figure` 对象来添加新的子图（axes）、调整布局、保存图形到文件等。

```
import matplotlib.pyplot as plt

# 创建一个新的图形和子图
fig, ax = plt.subplots()
ax.plot([1, 2, 3], [1, 2, 3])

# 获取当前活动的图形对象
current_fig = plt.gcf()

# 设置图形的大小
current_fig.set_size_inches(8, 6)

# 设置图形的标题
current_fig.suptitle('Example Plot')

# 显示图形
plt.show()
```

8.3.3 savefig

该函数用于将当前图形保存到文件中。该函数通常在完成图形绘制后调用，以便将图形输出为图像文件，便于保存或分享。

```
matplotlib.pyplot.savefig(fname, dpi='figure', format=None,
metadata=None, bbox_inches=None, pad_inches=0.1, facecolor='auto',
edgecolor='auto', backend=None, **kwargs)
```

- **fname** (`str` 或 `pathlib.Path` 或 `file-like` 对象)：文件的路径或文件对象，用于保存图像。如果未指定 `format` 参数，Matplotlib 会根据文件扩展名推断图像格式。
- **dpi** (`float` 或 `'figure'`)：图像的分辨率，以每英寸点数为单位。如果设置为 `'figure'`，则使用当前图形的分辨率设置。
- **format** (`str`, 可选)：图像的文件格式，例如 'png', 'pdf', 'svg', 'ps', 'eps' 等。如果未指定，则从 `fname` 的扩展名推断。
- **metadata** (`dict`, 可选)：要存储在图像文件中的元数据。
- **bbox_inches** (`str` 或 `Bbox`, 可选)：图像的边界框。如果设置为 'tight'，则 Matplotlib 会尝试紧凑地剪裁图像周围的空白。
- **pad_inches** (`float`, 可选)：在边界框周围添加额外的空白（英寸）。
- **facecolor** (`color`, 可选)：图形背景的颜色。
- **edgecolor** (`color`, 可选)：图形边缘的颜色。
- **backend** (`str`, 可选)：用于保存图像的特定后端。这通常不需要设置，除非你想要覆盖默认的后端。
- **kwargs** (`dict`, 可选)：其他关键字参数，将传递给底层后端的保存函数。

```
import matplotlib.pyplot as plt

# 创建一些数据
x = [0, 1, 2, 3, 4]
y = [0, 1, 4, 9, 16]

# 绘制图形
plt.plot(x, y)

# 保存图形到文件
plt.savefig('plot.png', dpi=300, bbox_inches='tight',
facecolor='g')
```

```
# 显示图形（可选）
plt.show()
```

8.3.4 imshow

该函数可以用来显示二维数据数组，其中每个数组元素对应图像的一个像素。

```
matplotlib.pyplot.imshow(X, cmap=None, norm=None, aspect=None,
interpolation=None, alpha=None, vmin=None, vmax=None, origin=None,
**kwargs)
```

- **X**：图像数据，通常是二维数组，但也可以是一维数组（在这种情况下，数据将被解释为线图）。数据类型通常是浮点数或整数。
- **cmap**：颜色映射（colormap），用于指定图像中数值到颜色的映射。默认是 `None`，此时将使用默认的 colormap，通常是 `viridis`。
- **norm**：用于标准化数据值的归一化对象，可以确保颜色映射是线性的或者使用其他映射方式。
- **aspect**：控制图像的纵横比。`'auto'` 保持宽高比，`'equal'` 使每个像素都是正方形，或者可以是一个数字。
- **interpolation**：指定图像缩放时的插值方法。常见的选项有 `'nearest'`（最近邻插值）、`'bilinear'`（双线性插值）、`'bicubic'`（双三次插值）等。
- **alpha**：图像的透明度，取值范围是 [0, 1]。
- **vmin, vmax**：用于颜色映射的数据值范围。默认情况下，这些值是从数据中自动推断出来的。
- **origin**：指定图像的原点，`'upper'` 表示原点在左上角，`'lower'` 表示原点在左下角。
- ****kwargs**：其他关键字参数。

与`plt.show`的区别是：

- `imshow` 是用于创建图像的函数，而 `show` 是用于显示图像的函数。
- 你可以在一个脚本中多次调用 `imshow` 来绘制多个图像，但通常只需要在脚本的末尾调用一次 `show` 来显示所有图像。

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

# 创建两个简单的数据集
data1 = np.random.rand(10, 10)
data2 = np.random.rand(10, 10)
```

```
# 创建一个包含两个子图的图形
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

# 在第一个子图上显示第一个图像，并应用 viridis 颜色映射
ax1.imshow(data1, cmap='viridis')
ax1.set_title('Image 1')

# 在第二个子图上显示第二个图像，并应用 plasma 颜色映射
ax2.imshow(data2, cmap='plasma')
ax2.set_title('Image 2')

# 调整子图之间的间距
plt.tight_layout()

# 显示图像
plt.show()
```

8.3.5 close

该函数用于关闭一个或多个打开的图形窗口。在 Matplotlib 中，每次调用绘图命令（如 `plt.plot()` 或 `plt.imshow()`）时，都会创建一个新的图形（Figure）和一个或多个子图（Axes）。`close` 函数可以用来关闭这些图形窗口，释放资源，并允许用户清理不再需要的图形。

```
matplotlib.pyplot.close(fig=None)
```

- **fig**（`Figure` 对象或整数，可选）：指定要关闭的图形。如果是一个 `Figure` 对象，则关闭该图形。如果是一个整数，则关闭与该编号对应的图形。如果省略或为 `None`，则关闭当前图形。'all'表示关闭所有的图像。

```
import matplotlib.pyplot as plt

# 创建并显示第一个图形
plt.figure()
plt.plot([1, 2, 3], [1, 2, 3])
plt.show()

# 创建并显示第二个图形
plt.figure()
plt.plot([3, 2, 1], [1, 2, 3])
```

```
plt.show()

# 关闭当前图形
plt.close()

# 创建第三个图形，但不显示
plt.figure()
plt.plot([1, 2, 3], [3, 2, 1])

# 关闭所有图形
plt.close(fig='all')
```

8.3.6 pause

用于在动画或交互式绘图过程中暂停一段时间。这个函数在动画制作或者需要在绘图之间插入延迟时非常有用。

```
matplotlib.pyplot.pause(interval)
```

- `interval`：这个参数是一个浮点数，表示暂停的秒数。默认值通常是 0.1 秒。

`plt.pause` 函数的主要作用如下：

1. **暂停执行**：当调用 `plt.pause(interval)` 时，Python 脚本会暂停执行指定的时间（以秒为单位）。
2. **更新显示**：如果在绘图之前调用了 `plt.pause`，它将允许动态更新的图形（例如动画）在屏幕上显示出来。这对于交互式绘图非常有用，因为它可以让用户看到图形的更新。

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

# 创建一个图形和一个子图
fig, ax = plt.subplots()

# 生成一些数据
t = np.arange(0, 10, 0.01)
s = np.sin(t)

# 绘制第一条线
ax.plot(t, s)
```

```
# 显示图形
plt.show(block=False)

# 更新线的数据并暂停
for phase in np.arange(0, 2 * np.pi, 0.05):
    ax.plot(t + phase, np.sin(t + phase))
    plt.pause(0.01)

# 关闭图形
plt.close()
```

9. GridSpec

GridSpec提供了更灵活的方式来安排子图（subplot）的布局。在 Matplotlib 中，默认的子图布局是通过 `plt.subplot()` 或 `plt.subplots()` 等函数实现的，但这些函数都是按照行列均匀划分子图的，而GridSpec可以根据具体需求定制每个子图的位置和大小。

该类处于`matplotlib.gridspec`库下。

```
matplotlib.gridspec.GridSpec(nrows, ncols, figure=None, left=None,
bottom=None, right=None, top=None, wspace=None, hspace=None,
width_ratios=None, height_ratios=None)
```

- `nrows`: 网格的行数。
- `ncols`: 网格的列数。
- `figure`: 可选，如果提供，`GridSpec` 将与这个指定的 `Figure` 实例关联。如果没有提供，`GridSpec` 将与当前活动的 `Figure` 关联。
- `left`, `bottom`, `right`, `top`: 可选，这些参数用于指定网格在整个画布中的位置，它们的值应该在 0 到 1 之间，表示画布宽度和高度的百分比。例如，`left=0.1` 表示网格左边界距离画布左边界 10% 的位置。
- `wspace`: 可选，子图之间的水平间距，这个值是以画布宽度的百分比来表示的。
- `hspace`: 可选，子图之间的垂直间距，这个值是以画布高度的百分比来表示的。
- `width_ratios`: 可选，指定各列的相对宽度。默认情况下，所有列的宽度都是相等的。如果提供这个参数，那么每一列的宽度将是其对应值的相对比例。
- `height_ratios`: 可选，指定各行的相对高度。默认情况下，所有行的高度都是相等的。如果提供这个参数，那么每一行的高度将是其对应值的相对比例。

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

```
from matplotlib.gridspec import GridSpec

fig = plt.figure(figsize=(10, 6))

# 创建一个 2 行 3 列的 GridSpec，第一列宽度是其他列的两倍，第二行高度是其他行的一倍半
gs = GridSpec(2, 3)

# 添加子图
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])
ax3 = fig.add_subplot(gs[0, 2])
ax4 = fig.add_subplot(gs[1, 0])
ax5 = fig.add_subplot(gs[1, 1:])
# 注意：gs[1, 1:] 表示第二行的第二列和第三列

plt.show()
```