

# matplotlib

## 笔记本

吴嘉军

2026 年 1 月 19 日

### 摘要

这里是摘要  
关键词：这里是关键词 1、关键词 2

### 目录

§1、 简单入门	1
1、 安装	1
1.1、 导入库	1
1.2、 创建数据	1
2、 绘制图形	1
3、 插入中文	1
4、 坐标布局	2
5、 坐标轴	2
6、 箭头	3
7、 文字说明	4
§2、 patches	5
1、 矩形	5
2、 三角形	6
2.1、 边角边画三角形	6
2.2、 向量	7
2.3、 三边画三角形	7
2.4、 角边角画三角形	8
3、 角	8
4、 圆形和椭圆	9
5、 扇形	9
§3、 隐函数	11
1、 圆锥曲线标准方程	11
2、 圆锥曲线一般方程	11

## §1、 简单入门

一般来说,matplotlib 需要和 numpy, 搭配着使用, 因为 matplotlib 主要是用来绘图的, 而 numpy 主要是用来进行数值计算的, 两者结合起来可以更好地进行数据处理和可视化.

### 1、 安装

在使用 matplotlib 之前, 需要先安装它. 可以使用 pip 命令进行安装

```
1 pip install matplotlib
```

安装完成后, 可以使用以下命令来验证是否安装成功.

#### 1.1、 导入库

在使用 matplotlib 之前, 需要先导入它. 通常情况下, 我们会导入 matplotlib.pyplot 模块, 并将其重命名为 plt. 这样可以方便我们后续的使用.

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

#### 1.2、 创建数据

在绘图之前, 我们需要先创建一些数据. 通常情况下, 我们会使用 numpy 来创建数据. 例如, 我们可以使用 numpy 的 linspace 函数来创建一个等间隔的数组.

```
1 x = np.linspace(0, 10, 100) # 创建一个从0到10的等间隔数组,包含100个点
2 y = np.sin(x) # 计算x对应的正弦值
```

## 2、 绘制图形

使用 matplotlib 绘制图形非常简单. 我们只需要调用 plt.plot 函数, 并传入 x 和 y 数据即可. 然后, 使用 plt.show 函数来显示图形.

```
1 plt.plot(x, y) # 绘制图形
2 plt.xlabel('X轴标签') # 设置X轴标签
3 plt.ylabel('Y轴标签') # 设置Y轴标签
4 plt.title('图形标题') # 设置图形标题
5 plt.show() # 显示图形
```

下面是一个简单的例子, 展示如何使用 matplotlib 绘制一个正弦波.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 # 创建数据
4 x = np.linspace(0, 4 * np.pi, 100)
5 y = np.sin(x)
6
7 # 绘制图形
8 plt.plot(x, y)
9 plt.xlabel('X')
10 plt.ylabel('Y')
11 plt.title('sin')
12
13 plt.savefig('figures/section1/1.1.png') # 保存图形为文件
14 # Show the plot
15 plt.show()
```

运行上述代码, 将会显示一个包含正弦波的图形窗口.

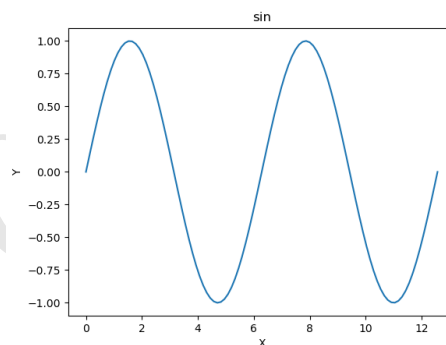


图 1.1: 正弦波图形

## 3、 插入中文

在使用 matplotlib 绘图时, 如果需要插入中文, 可以使用以下方法:

```
1 plt.xlabel('X轴标签', fontproperties='SimHei') # 设置X轴标签为中文
2 plt.ylabel('Y轴标签', fontproperties='SimHei') # 设置Y轴标签为中文
3 plt.title('图形标题', fontproperties='SimHei') # 设置图形标题为中文
```

这里的'SimHei' 是黑体字体的名称, 你也可以根据需要选择其他字体. 确保你的系统中已经安装了所需的中文字体.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 # 创建数据
4 x = np.linspace(0, 4 * np.pi, 100)
5 y = np.sin(x)
6
7 # 绘制图形
8 plt.plot(x, y)
9 plt.xlabel('X轴标签', fontproperties='SimHei') # 设置X轴标签为中文
10 plt.ylabel('Y轴标签', fontproperties='SimHei') # 设置Y轴标签为中文
11 plt.title('图形标题', fontproperties='SimHei') # 设置图形标题为中文
12
13 plt.savefig('figures/section1/1.2.png') # 保存图形为文件
14 # Show the plot
```

```
15 plt.show()
```

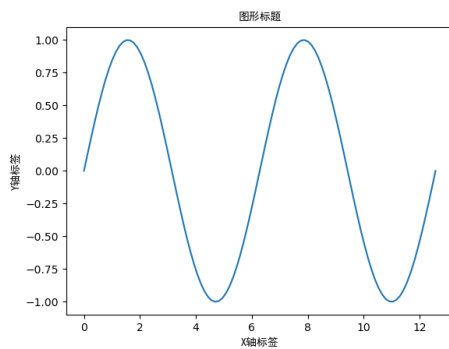


图 1.2

如果觉得太麻烦了, 可以在代码开头添加如下代码, 这样就可以全局使用中文字体了.

```
1 import matplotlib.pyplot as plt
2 from matplotlib import rcParams
3 rcParams['font.sans-serif'] = ['SimHei'] # 全局设置中文字体为黑体
4 rcParams['axes.unicode_minus'] = False # 解决负号显示问题
```

运行上述代码, 将可以在 fig 中添加中文, 以下代码为例:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 # 创建数据
4 plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置中文字体为SimHei
5 plt.rcParams['axes.unicode_minus'] = False # 解决负号显示问题
6 x = np.linspace(0, 4 * np.pi, 100)
7 y = np.sin(x)
8
9 # 绘制图形
10 plt.plot(x, y)
11 plt.xlabel('X轴标签') # 设置X轴标签为中文
12 plt.ylabel('Y轴标签') # 设置Y轴标签为中文
13 plt.title('图形标题') # 设置图形标题为中文
14
15 plt.savefig('figures/section1/1.3.png') # 保存图形为文件
16 # Show the plot
17 plt.show()
```

## 4、坐标布局

在 matplotlib 中, 可以通过多种方式来调整图形的坐标布局. 以下是一些常用的方法:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
5 plt.rcParams['axes.unicode_minus'] = False # 用来正常显示负号
6 fig = plt.figure("title") # an empty figure with no Axes
7 fig, ax = plt.subplots() # a figure with a single Axes
8 fig.suptitle("Figure_title") # set the Figure title
9 ax.set_title("Axes_title") # set the Axes title
10 fig, axs = plt.subplots(2, 2) # a figure with a 2x2 grid of Axes
```

```
11 axs[0, 0].set_title("Axes_[0,0]_title") # set the title for a specific Axes
12 axs[1, 1].set_title("Axes_[1,1]_title")
13 fig.tight_layout() # adjust spacing between subplots
14
15 # a figure with one Axes on the left, and two on the right:
16 fig, axs = plt.subplot_mosaic([['left', 'right_top'],
17                                ['left', 'right_bottom']])
18 axs['left'].set_title("左侧子图标题")
19 axs['right_top'].set_title("右上子图标题")
20 axs['right_bottom'].set_title("右下子图标题")
21 fig.tight_layout() # adjust spacing between subplots
22 plt.savefig('figures/section1/1.4.png') # 保存图形为文件
23 plt.show()
```

运行上述代码, 将会显示一个包含三个子图的图形窗口, 一个在左, 两个在右. 如下所示:

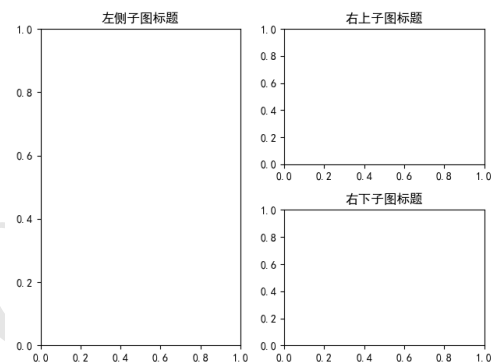


图 1.3: 多个子图布局

## 5、坐标轴

在 matplotlib 中, 可以通过多种方式来调整图形的坐标轴. 以下是一些常用的方法:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
5 plt.rcParams['axes.unicode_minus'] = False # 用来正常显示负号
6
7 fig, ax = plt.subplots()
8 x = np.arange(0, 4*np.pi, 0.01)
9 y = np.sin(x)
10 ax.plot(x, y)
11 # 设置坐标轴标签
12 ax.set_xlabel("X轴标签")
13 ax.set_ylabel("Y轴标签")
14 # 设置坐标轴范围
15 ax.set_xlim(0, 9)
16 ax.set_ylim(-1, 1)
17 # 设置刻度
18 ax.set_xticks([0, 0.5*np.pi, np.pi, 1.5*np.pi, 2*np.pi, 2.5*np.pi, 3*np.pi, 3.5*np.pi, 4*np.pi])
19 ax.set_yticks([-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1])
20
21 # 设置刻度标签
22 ax.set_xticklabels(['0', '零点五', '1', '一点五', '2', '二点五', '3', '三点五', '4'])
23 # ax.set_yticklabels(['-1', '-0.8', '-0.6', '-0.4', '-0.2', '0', '0.2', '0.4', '0.6', '0.8', '1'])
24 # 设置标题
25 ax.set_title("坐标轴示例")
```

```
26 # 显示网格
27 ax.grid(True)
28 # 上下左右边框可见性
29 ax.spines['top'].set_visible(False)
30 ax.spines['right'].set_visible(False)
31 # 设置刻度方向、长度、宽度和颜色
32 ax.tick_params(direction='inout', length=10, width=10, colors='red')
33 # 设置刻度线位置
34 ax.xaxis.set_ticks_position('bottom')
35 ax.yaxis.set_ticks_position('left')
36 # 设置刻度线样式
37 # 打开次刻度线
38 # 关闭次刻度线
39 ax.minorticks_off()
40 # 显示主次网格线
41 ax.grid(which='minor', linestyle='--', linewidth=0.5)
42 ax.minorticks_on()
43 ax.xaxis.set_tick_params(which='major', size=10, width=1, direction='out',
44 , colors='blue')
44 ax.xaxis.set_tick_params(which='minor', size=5, width=1, direction='out',
45 , colors='red')
45 ax.yaxis.set_tick_params(which='major', size=10, width=0.5, direction='out')
46 ax.yaxis.set_tick_params(which='minor', size=5, width=1, direction='in',
47 , colors='green')
48 # 设置坐标轴0位置
49 ax.spines['bottom'].set_position(('data', 0)) # x轴位置
50 ax.spines['left'].set_position(('data', 0)) # y轴位置
51 # 添加箭头 ,clip_on=False的作用是防止箭头被裁掉
52 # 添加箭头
53 ax.plot(1, 0, '>r', transform=ax.get_yaxis_transform(), clip_on=False) #
54 transform=ax.get_yaxis_transform()表示箭头在y轴上
54 ax.plot(0, 1, '>b', transform=ax.get_xaxis_transform(), clip_on=False) #
55 transform=ax.get_xaxis_transform()表示箭头在x轴上
56 plt.savefig('figures/section1/1.5.png') # 保存图形为文件
57 plt.show()
```

运行上述代码, 将会显示一个包含自定义坐标轴的图形窗口。如下所示:

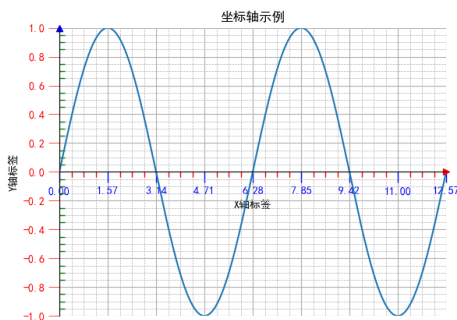


图 1.4: 自定义坐标轴

## 6、 箭头

在 matplotlib 中, 可以通过多种方式来添加箭头。以下是一些常用的方法:

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
```

```
3
4 fig, ax = plt.subplots()
5
6 # 弯箭头示例
7 arrow = mpatches.FancyArrowPatch(
8     # 起点和终点坐标
9     (0.2, 0.2), (.8, .8),
10    # 箭头样式和属性
11    arrowstyle='->', head_length=0.4, head_width=0.2',
12    connectionstyle='arc3,rad=0.3', # 弯曲半径
13    # 其他属性,箭头颜色,线宽,大小等
14    color='blue', linewidth=2, mutation_scale=20,
15    # 坐标系是data坐标系,不裁剪
16    transform=ax.transData, clip_on=False
17 )
18 ax.add_patch(arrow)
19
20 plt.savefig('figures/section1/1.6.png') # 保存图形为文件
21 plt.show()
```

运行上述代码, 将会显示一个包含箭头的图形窗口。如下所示:

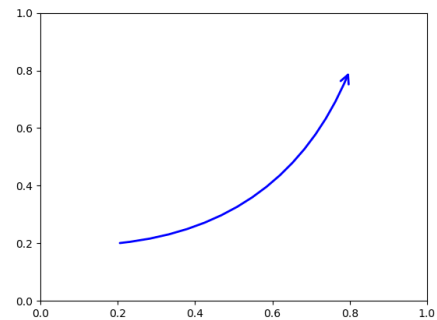


图 1.5: 箭头示例

再举例说明一下如何添加不同样式的箭头。

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4
5 plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签
6 plt.rcParams['axes.unicode_minus'] = False # 用来正常显示负号
7
8 fig, ax = plt.subplots()
9 x = np.linspace(0, 4*np.pi, 100)
10 y = np.sin(x)
11 ax.set_title("箭头示例2")
12 ax.plot(x, y)
13 # 直箭头示例
14 arrow1 = mpatches.FancyArrowPatch(
15     (0.7*np.pi, 1.2), (0.5 * np.pi, 1),
16     arrowstyle='->', mutation_scale=20,
17     connectionstyle='arc3,rad=0.3',
18     color='red',
19     transform=ax.transData, clip_on=False
20 )
21 arrow2 = mpatches.FancyArrowPatch(
22     (3.7*np.pi, -1.2), (3.5 * np.pi, -1),
23     arrowstyle='->', mutation_scale=20,
24     connectionstyle='arc3,rad=0.3',
25     color='red',
26     transform=ax.transData, clip_on=False
```

```
27 )
28
29 ax.add_patch(arrow1)
30 ax.add_patch(arrow2)
31
32 ax.spines['top'].set_visible(False)
33 ax.spines['right'].set_visible(False)
34 ax.spines['bottom'].set_position(('data', 0)) # x轴位置
35 ax.spines['left'].set_position(('data', 0)) # y轴位置
36
37 # 添加箭头, clip_on=False的作用是防止箭头被裁剪掉
38 ax.plot(1, 0, '>r', transform=ax.get_yaxis_transform(), clip_on=False) #
    transform=ax.get_yaxis_transform()表示箭头在y轴上
39 ax.plot(0, 1, '<b', transform=ax.get_xaxis_transform(), clip_on=False) #
    transform=ax.get_xaxis_transform()表示箭头在x轴上
40
41 # 点一个(2,1)位置
42 ax.plot(0.5*np.pi, 1, 'og', markersize=8)
43 ax.plot(3.5*np.pi, -1, 'ob', markersize=8)
44
45 # 标注文字说明
46 ax.text(0.7*np.pi - 0.1, 1.2 - 0.1, "最高点( $\pi/2, 1$ )", color='red', fontsize=10)
47 ax.text(3.7*np.pi - 0.1, -1.2 - 0.1, "最低点( $7\pi/2, -1$ )", color='red',
    fontsize=10)
48
49 plt.savefig('figures/section1/1.7.png') # 保存图形为文件
50 plt.show()
```

运行上述代码, 将会显示一个包含不同样式箭头的图形窗口. 如下所示:

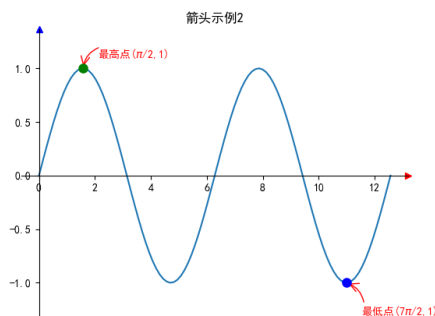


图 1.6: 不同样式箭头示例

## 7、 文字说明

在 matplotlib 中, 可以通过多种方式来添加文字说明. 以下是一些常用的方法:

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7
8
9 fig, ax = plt.subplots()
10 fig.suptitle("文字说明")
11 ax.set_title("latex文字说明")
12
13 ax.plot([0.5, 1, 1.5, 2, 2.5], [-0.5, -0.25, 0, 0.25, 0.5], 'og', markersize=2)
14 # 添加文字说明
15 ax.text(0.5, -0.5, "matplotlib可以插入latex公式", color='green', fontsize=12)
16 ax.text(1, -0.25, r"$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$", color='green', fontsize=15)
17 ax.text(1.5, 0, r"$\int_a^b f(x) dx = F(b) - F(a)$", color='green', fontsize=15)
18 ax.text(2, 0.25, r"$e^{i\pi} = -1$", color='green', fontsize=15)
19 ax.text(2.5, 0.5, r"$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}$", color='green', fontsize=15)
20
21
22
23 ax.set_xlim(0, 3.5)
24 ax.set_ylim(-1, 1)
25
26 ax.grid(True)
27 plt.savefig('figures/section1/1.8.png') # 保存图形为文件
28 plt.show()
```

运行上述代码, 将会显示一个包含文字说明的图形窗口. 如下所示:

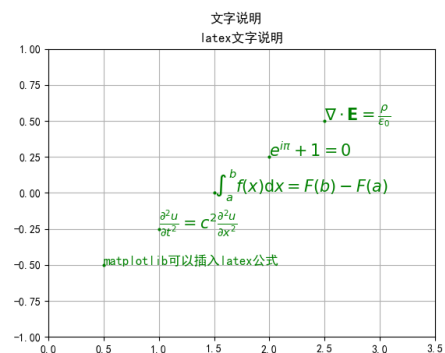


图 1.7: 文字说明示例

## §2、 patches

在 matplotlib 中,patches 模块提供了许多用于绘制图形元素的类. 这些类可以用来创建各种形状,如矩形、圆形、椭圆等. 以下是一些常用的 patches 类:

- Rectangle: 用于绘制矩形.
- Circle: 用于绘制圆形.
- Ellipse: 用于绘制椭圆.
- Polygon: 用于绘制多边形.

例如,我们可以使用 patches.Rectangle 来创建一个正方形:

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7
8 fig, ax = plt.subplots()
9 # 创建一个正方形
10 rec = mpatches.Rectangle((0.2,0.2), 0.6, 0.6, facecolor='blue', edgecolor='black')
11 ax.add_patch(rec)
12
13 ax.plot(0.2,0.2,color = 'red', marker = 'o', markersize=10)
14
15 ax.set_xlim(0,1)
16 ax.set_ylim(0,1)
17 ax.set_title('正方形示例')
18 # x轴和y轴一样长
19 ax.set_aspect('equal')
20 # 保持图形
21 plt.savefig('figures/section2/2.1.png')
22 plt.show()
```

运行上述代码, 将会显示一个包含正方形的图形窗口. 如下所示:

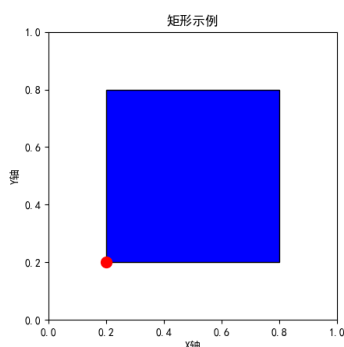


图 2.1: 正方形示例

## 1、 矩形

上面的例子已经展示了如何使用 `patches.Rectangle` 来创建一个矩形. 我们还可以通过设置不同的参数来调整矩形的属性, 如颜色、边框宽度等.

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7 # 创建子图
8 fig, ax = plt.subplots(3,2,figsize=(10,15))
9
10 # angle=0表示不旋转,facecolor表示填充颜色,edgecolor表示边框颜色,linewidth表示
   # 边框宽度,alpha表示透明度
11 rect = mpatches.Rectangle((0.2,0.2),0.6,0.6,angle=0,facecolor="lightblue",
   edgecolor="blue",linewidth=2,alpha=0.7)
12 ax[0,0].add_patch(rect)
13 ax[0,0].set_xlim(0,1)
14 ax[0,0].set_ylim(0,1)
15 # adjustable参数设置为box时, 长宽比固定
16 ax[0,0].set_aspect('equal', adjustable='box')
17
18 # angle=45表示旋转45度,默认是绕着左下角旋转
19 rect = mpatches.Rectangle((0.2,0.2),0.6,0.6,angle=45,facecolor="lightgreen",
   edgecolor="green",linewidth=2,alpha=0.7)
20 ax[0,1].add_patch(rect)
21 ax[0,1].set_xlim(0,1)
22 ax[0,1].set_ylim(0,1)
23 ax[0,1].set_aspect('equal', adjustable='box')
24
25 # 填充颜色和边框颜色,填充无颜色,边框填充红色
26 rect = mpatches.Rectangle((0.2,0.2),0.6,0.6,angle=0,facecolor='none',
   edgecolor="red",linewidth= 2,alpha=1)
27 ax[1,0].add_patch(rect)
28 ax[1,0].set_xlim(0,1)
29 ax[1,0].set_ylim(0,1)
30 ax[1,0].set_aspect('equal', adjustable='box')
31 # 修改长方形的位置在原点,默认是以左下角为原点
32 rect = mpatches.Rectangle((0,0),0.6,0.6,angle=0,facecolor="orange",
   edgecolor="brown",linewidth=2,alpha=0.7)
33 ax[1,1].add_patch(rect)
34 ax[1,1].set_xlim(0,1)
35 ax[1,1].set_ylim(0,1)
36 ax[1,1].set_aspect('equal', adjustable='box')
37 ax[1,1].plot(0,0,'ro',markersize = 10) # 标记原点位置
38
39 # 修改长方形的长和宽,宽为1,长为0.6
40 rect = mpatches.Rectangle((0,0,0.1),1,0.6,angle=0,facecolor="purple",
   edgecolor="none",linewidth=2,alpha=0.7)
41 ax[2,0].add_patch(rect)
42 ax[2,0].set_xlim(0,1)
43 ax[2,0].set_ylim(0,1)
44 ax[2,0].set_aspect('equal', adjustable='box')
45
46 # 修改长方形的透明度,alpha=0.3表示透明度为30%
47 rect1 = mpatches.Rectangle((0.2,0.2),0.6,0.6,angle=0,facecolor="red",
   edgecolor="none",linewidth=2,alpha=0.3)
48 rect2 = mpatches.Rectangle((0.3,0.3),0.6,0.6,angle=0,facecolor="blue",
   edgecolor="none",linewidth=2,alpha=0.4)
49 ax[2,1].add_patch(rect1)
50 ax[2,1].add_patch(rect2)
51 ax[2,1].set_xlim(0,1)
52 ax[2,1].set_ylim(0,1)
53 ax[2,1].set_aspect('equal', adjustable='box')
54
```

```
55 plt.savefig("figures/section2/2.2.png",dpi=300)
56 plt.show()
```

运行上述代码, 将会显示一个包含自定义矩形的图形窗口. 如下所示:

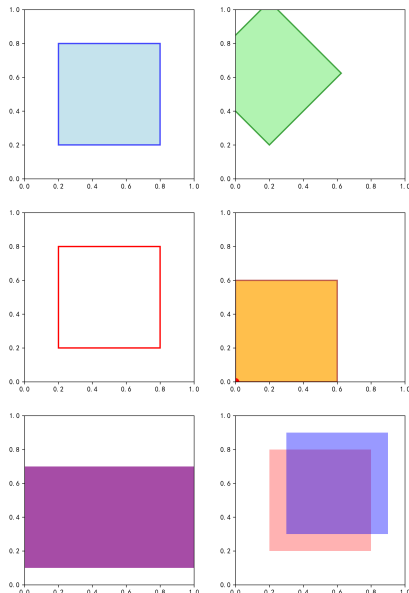


图 2.2: 自定义矩形示例

## 2、 三角形

我们可以使用 `patches.Polygon` 来创建一个三角形. 以下是一个示例:

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7
8 # 创建子图
9 fig, ax = plt.subplots(1, 2, figsize=(10,5))
10 # 创建一个等边三角形,中心在(0.5,0.5),边长为0.8,RegularPolygon表示正多边形,
    numVertices=3表示三边形,radius表示外接圆半径,orientation表示旋转角度 pi
    /2表示旋转90度
11 triangle = mpatches.RegularPolygon((0.5, 0.5), numVertices=3, radius=0.4,
    orientation=np.pi / 2,
12                                     facecolor='lightcoral', edgecolor='darkred',
    linewidth=2, alpha=0.7)
13 ax[0].add_patch(triangle)
14 ax[0].set_xlim(0, 1)
15 ax[0].set_ylim(0, 1)
16 ax[0].set_aspect('equal')
17 ax[0].set_title('等边三角形')
18
```

```
19 # 创建一个直角三角形,顶点坐标为(0.2,0.2),(0.8,0.2),(0.8,0.8)
20 right_triangle = mpatches.Polygon([(0.2, 0.2), (0.8, 0.2), (0.8, 0.8)], facecolor
    = 'lightblue', edgecolor='darkblue', linewidth=2, alpha=0.7)
21 ax[1].add_patch(right_triangle)
22 ax[1].set_xlim(0, 1)
23 ax[1].set_ylim(0, 1)
24 ax[1].set_aspect('equal')
25 ax[1].set_title('直角三角形')
26
27 plt.savefig("figures/section2/2.3.png", dpi=300)
28 plt.show()
```

运行上述代码, 将会显示一个包含三角形的图形窗口. 如下所示:

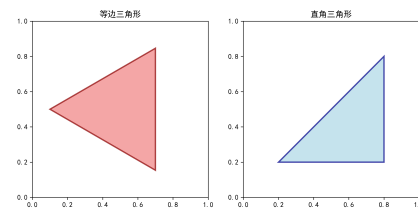


图 2.3: 三角形示例

这样绘制的三角形是已知顶点坐标的. 如果是只知道两条边长和其中的夹角, 可以通过计算得到其他两个顶点的坐标.

### 2.1、 边角边画三角形

举个例子, 假设已知三角形的一条边长为  $a$ , 另一条边长为  $b$ , 夹角为  $\theta$ , 则可以通过以下代码计算出其他两个顶点的坐标并绘制三角形:

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7
8 # 创建子图
9 fig, ax = plt.subplots(1, 1, figsize=(10,5))
10 # 知道三角形的两条边长和夹角,计算顶点坐标
11 a = 0.6 # 边长a
12 b = 0.5 # 边长b
13 theta = np.radians(60) # 夹角60度转换为弧度
14
15 # 计算顶点坐标
16 A = [0.2, 0.2]
17 B = [A[0] + a, A[1]]
18 C = [A[0] + b * np.cos(theta), A[1] + b * np.sin(theta)]
19 # 创建三角形
20 triangle = mpatches.Polygon([A, B, C], facecolor='lightcoral', edgecolor='
    darkred', linewidth=2, alpha=0.7)
21 ax.add_patch(triangle)
22 ax.set_xlim(0, 1)
23 ax.set_ylim(0, 1)
24 ax.set_aspect('equal')
```

```
25 ax.set_title('已知两边和夹角的三角形')
26 # *A表示解包坐标,*解包操作符,表示将元组拆开,写字在旁边,
27 ax.plot(*A, 'ro', label='顶点A') # 标记顶点A
28 ax.plot(*B, 'go', label='顶点B') # 标记顶点B
29 ax.plot(*C, 'bo', label='顶点C') # 标记顶点C
30
31 ax.text(*A, 'A', fontsize=12, ha='right', va='bottom')
32 ax.text(*B, 'B', fontsize=12, ha='left', va='bottom')
33 ax.text(*C, 'C', fontsize=12, ha='center', va='bottom')
34 ax.legend()
35
36 plt.savefig("figures/section2/2.4.png", dpi=300)
37 plt.show()
```

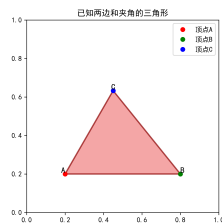


图 2.4: 三角形示例 2

## 2.2、 向量

还可以使用向量在图像中需找特殊点, 测量边长, 测量角, 绘制角平分线.

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7
8 a = 1
9 b = 1
10 theta = np.radians(60) # 夹角60度转换为弧
11 A = [0.2, 0.2]
12 B = [A[0] + a, A[1]]
13 C = [A[0] + b * np.cos(theta), A[1] + b * np.sin(theta)]
14 A = np.array(A)
15 B = np.array(B)
16 C = np.array(C)
17 print(f'{A},{B},{C}')
18 fig, ax = plt.subplots()
19 ax.text(A[0], A[1], "A")
20 ax.text(B[0], B[1], "B")
21 ax.text(C[0], C[1], "C")
22 triangle = mpatches.Polygon([A, B, C], facecolor='lightblue', edgecolor='darkred', linewidth=2, alpha=0.7)
23
24 vec_ab = B - A
25 vec_ac = C - A
26 vec_bc = B - C
27 print(vec_ab)
28 ab = np.linalg.norm(vec_ab)
29 print(f'边长ab:{ab}')
30
31 mid_ab = (A + B) / 2
32 print(f'中点坐标:{mid_ab}')
33
34 angle_A = np.dot(vec_ab, vec_ac) / (np.linalg.norm(vec_ab) * np.linalg.norm(vec_ac))
```

```
35 # 计算弧度角
36 angle_radians = np.arccos(angle_A)
37 # 转换为度数
38 angle_degrees = np.degrees(angle_radians)
39 print(f"A夹角{angle_degrees}")
40
41 # 计算角平分线
42 # 归一化向量
43 unit_vec_ab = vec_ab / np.linalg.norm(vec_ab)
44 unit_vec_ac = vec_ac / np.linalg.norm(vec_ac)
45 print(unit_vec_ab)
46 print(unit_vec_ac)
47
48
49 # # 计算角平分线向量
50 bisector = unit_vec_ab + unit_vec_ac + A
51 ax.text(bisector[0], bisector[1], '角平分线')
52 x, y = zip(A, bisector) # 使用zip解包坐标
53 print(x)
54 print(y)
55 plt.plot(x, y, 'bo', linestyle='--', label='角平分线')
56
57 angle_A = np.dot(vec_ab, bisector) / (np.linalg.norm(vec_ab) * np.linalg.norm(bisector))
58 # 计算弧度角
59 angle_radians = np.arccos(angle_A)
60 # 转换为度数
61 angle_degrees = np.degrees(angle_radians)
62 print(f"A夹角{angle_degrees}")
63
64 # # 计算中线
65 mid_bc = (B + C) / 2
66 ax.text(mid_bc[0], mid_bc[1], '中点')
67 print(mid_bc)
68 x, y = zip(A, mid_bc) # 使用zip解包坐标
69 # print(x)
70 # print(y)
71 plt.plot(x, y, 'ro', linestyle='--', label='中线')
72
73
74 ax.add_patch(triangle)
75 ax.set_aspect('equal')
76 plt.savefig('figures/section2/2.5.png', dpi=300)
77 plt.show()
```

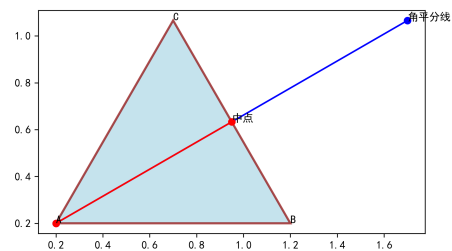


图 2.5: 三角形示例 3

## 2.3、 三边画三角形

如果知道三条边的长度 a,b,c, 也可以用于绘制三角形.



```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4
5 a,b,c = 1,1,1
6 A = [0.,0.]
7 # 利用余弦定理
8 # cosA=(b^2+c^2-a^2)/2bc
9 thetaA = (b**2 + c**2 - a**2) / (2*b*c)
10 angle_radians = np.arccos(thetaA)
11 # 转换为度数
12 angle_degrees = np.degrees(angle_radians)
13
14 print(f"angle_degrees:{2f}")
15 B = [A[0] + c, A[1]]
16 C = [A[0] + b * np.cos(angle_radians), A[1] + b * np.sin(angle_radians)]
17
18 fig,ax = plt.subplots()
19 triangle = mpatches.Polygon([A, B, C], facecolor='lightblue', edgecolor='
20     darkred', linewidth=2, alpha=0.7)
21 ax.add_patch(triangle)
22 ax.text(0.5,0.5,r"$\cos A=\frac{b^2+c^2-a^2}{2bc}$",fontsize =
23     18)
24 ax.text(*A,'$A$',color = "red")
25 ax.text(*B,'$B$')
26 ax.text(*C,'$C$',color = 'blue')
27
28 ax.set_xlim(0,c + 0.1)
29 ax.set_ylim(0,b)
30 ax.set_aspect('equal')
31 plt.savefig("figures/section2/2.6.png")
32 plt.show()
```

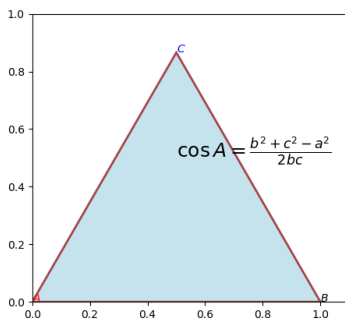


图 2.6: 三角形示例 4

```
8 # 初始条件
9 thetaA,thetaB = 30,90
10 c = 1
11 A = [0.,0.]
12
13 thetaC = 180 - thetaA - thetaB
14 if thetaC < 0:
15     print("构造不了三角形")
16     pass
17
18 angle_radiansA = np.radians(thetaA)
19 angle_radiansB = np.radians(thetaB)
20 angle_radiansC = np.radians(thetaC)
21
22 a = np.sin(angle_radiansA) / (np.sin(angle_radiansC) / c)
23 b = np.sin(angle_radiansB) / (np.sin(angle_radiansC) / c)
24
25 print(f"{a},{b}")
26
27 B = [A[0] + c, A[1]]
28 C = [A[0] + b * np.cos(angle_radiansA), A[1] + b * np.sin(angle_radiansA
29     )]
30 fig,ax = plt.subplots()
31 triangle = mpatches.Polygon([A, B, C], facecolor='lightblue', edgecolor='
32     darkred', linewidth=2, alpha=0.7)
33 ax.add_patch(triangle)
34 ax.set_xlim(0,c + 0.1)
35 ax.set_ylim(0,b)
36 ax.text(0.5,0.3,r"$\frac{a}{\sin A}=\frac{b}{\sin B}=\frac{c}{\sin C}$",fontsize = 18)
37 ax.text(*A,'$A$',color = "red")
38 ax.text(*B,'$B$')
39 ax.text(*C,'$C$',color = 'blue')
40 ax.set_aspect('equal')
41 plt.savefig("figures/section2/2.7.png")
42 plt.show()
```

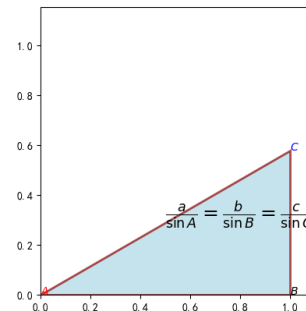


图 2.7: 三角形示例 5

## 2.4、 角边角画三角形

知道三角形的两个角, 和两角中间的边长, 也可  
以绘制三角形. 可以运用正弦定理:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

用下面的例子绘图:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import matplotlib.patches as mpatches
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7
```

## 3、 角

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import matplotlib.patches as mpatches
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7
8 A = [0.,0.]
9 B = [1.,0.]
10 theta = 120
```

```
11 radians = np.radians(theta)
12
13 C = [A[0] + 1. * np.cos(radians), A[1] + 1. * np.sin(radians)]
14
15 fig, ax = plt.subplots()
16 x, y = zip(A, B)
17 plt.plot(x, y, 'r-')
18 x, y = zip(A, C)
19 plt.plot(x, y, 'r-')
20 # 绘制弧线, width和height是椭圆形的长边和短边, 相等时相当于圆形弧线, theta1和
    theta2分别是起点和终点角度
21 arc = mpatches.Arc((0,0), width=0.3, height=0.3, angle=0, theta1=0, theta2=
    theta, linewidth=2, color="blue")
22
23 ax.add_patch(arc)
24 ax.text(0.1, 0.1, fr'$\alpha=\theta$', fontsize=12)
25 ax.set_aspect('equal')
26 plt.savefig('figures/section2/2.8.png')
27 plt.show()
```

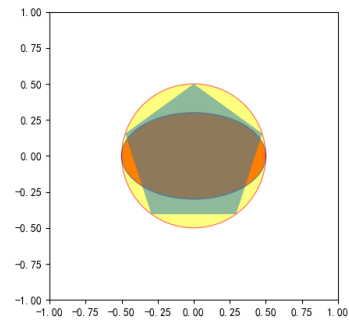


图 2.9: 圆形和椭圆

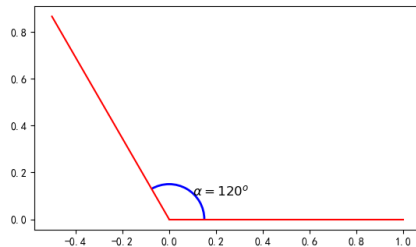


图 2.8: 角

## 4、 圆形和椭圆

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import numpy as np
4 import os
5
6 plt.rcParams['font.sans-serif'] = ['SimHei']
7 plt.rcParams['axes.unicode_minus'] = False
8
9 fig, ax = plt.subplots()
10
11 ellipse = mpatches.Ellipse((0,0), 1., 0.6, fc='red', ec='blue')
12 circle = mpatches.Circle((0,0), 0.5, fc='yellow', alpha=0.5, ec='red')
13 regularPolygon = mpatches.RegularPolygon((0,0), numVertices=5, radius
    =0.5, alpha=0.5)
14 ax.add_patch(ellipse)
15 ax.add_patch(circle)
16 ax.add_patch(regularPolygon)
17 ax.set_xlim(-1,1)
18 ax.set_ylim(-1,1)
19 ax.set_aspect('equal')
20 # 文件保存
21 script_name = os.path.splitext(os.path.basename(__file__))[0]
22 plt.savefig("figures/section2/" + script_name + '.png')
23 plt.show()
```

## 5、 扇形

```
1 import matplotlib.pyplot as plt
2 import matplotlib.patches as mpatches
3 import os
4
5 plt.rcParams['font.sans-serif'] = ['SimHei']
6 plt.rcParams['axes.unicode_minus'] = False
7 fig, ax = plt.subplots()
8
9 # 使用 patches 自定义样式
10 wedge0 = mpatches.Wedge((0.,0.), 0.5, theta1=0, theta2=30, linewidth=2,
    edgcolor='purple', facecolor='plum', alpha=0.7)
11 wedge1 = mpatches.Wedge((0.,0.), 0.5, theta1=30, theta2=60, linewidth=2,
    edgcolor='green', facecolor='purple', alpha=0.7)
12 wedge2 = mpatches.Wedge((0.,0.), 0.5, theta1=60, theta2=150, linewidth=2,
    edgcolor='black', facecolor='green', alpha=0.7)
13 wedge3 = mpatches.Wedge((0.,0.), 0.5, theta1=150, theta2=290, linewidth=2,
    edgcolor='blue', facecolor='yellow', alpha=0.7)
14 wedge4 = mpatches.Wedge((0.,0.), 0.5, theta1=290, theta2=360, linewidth=2,
    edgcolor='plum', facecolor='red', alpha=0.7)
15
16 ax.add_patch(wedge0)
17 ax.add_patch(wedge1)
18 ax.add_patch(wedge2)
19 ax.add_patch(wedge3)
20 ax.add_patch(wedge4)
21 ax.set_xlim(-1,1)
22 ax.set_ylim(-1,1)
23 ax.set_aspect('equal')
24 # 文件保存
25 script_name = os.path.splitext(os.path.basename(__file__))[0]
26 plt.savefig("figures/section2/" + script_name + '.png')
27 plt.show()
```

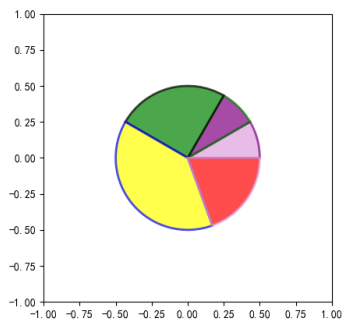


图 2.10: 扇形

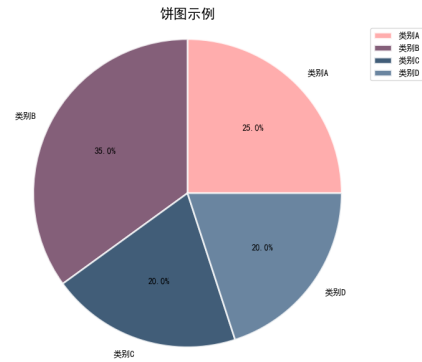


图 2.11: 扇形饼图

## plt 也可以绘制扇形饼图

```
1 # 利用matplotlib.pyplot也可以直接绘制扇形饼图
2 import matplotlib.pyplot as plt
3 # import matplotlib.patches as mpatches
4 import os
5
6 plt.rcParams['font.sans-serif'] = ['SimHei']
7 plt.rcParams['axes.unicode_minus'] = False
8 # 准备数据
9 labels = ['类别A', '类别B', '类别C', '类别D']
10 sizes = [25, 35, 20, 20]
11 colors = ["#ff9999", "#663758", "#123456", "#456789"]
12
13 # 创建图形
14 fig, ax = plt.subplots(figsize=(8, 6))
15
16 # 绘制饼图并获取扇形patch
17 wedges, texts, autotexts = ax.pie(sizes, labels=labels, colors=colors, autopct
    = '%1.1f%%')
18
19 # 自定义每个扇形的样式
20 for i, wedge in enumerate(wedges):
21     # 设置边框颜色和宽度
22     wedge.set_edgecolor('white')
23     wedge.set_linewidth(2)
24     # 设置透明度
25     wedge.set_alpha(0.8)
26
27 # 添加标题
28 ax.set_title('饼图示例', fontsize=16, fontweight='bold')
29
30 # 确保饼图是圆形
31 ax.axis('equal')
32
33 # 添加图例
34 ax.legend()
35
36 # 调整布局
37 plt.tight_layout()
38
39 # 文件保存
40 script_name = os.path.splitext(os.path.basename(__file__))[0]
41 plt.savefig("figures/section2/" + script_name + '.png')
42 plt.show()
```

## §3、 隐函数

matplotlib 也可以按照隐函数进行绘图. 这里主要提供两种方法进行画图

### 1、 圆锥曲线标准方程

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import os
4
5 # 设置中文字体, 例如使用 "SimHei"
6 plt.rcParams['font.sans-serif'] = ['SimHei'] # 添加'SimHei'到字体列表
7 plt.rcParams['axes.unicode_minus'] = False # 解决负号显示为 的问题
8 fig, ax = plt.subplots()
9 fig.suptitle(r'隐函数$\frac{x^2}{4} + \frac{y^2}{1} = 1$')
10 # 定义x和y的范围
11 x = np.linspace(-3, 3, 100)
12 y = np.linspace(-1.5, 1.5, 100)
13 X, Y = np.meshgrid(x, y)
14
15 # 计算Z值, Z = x^2/4 + y^2/1 - 1
16 Z = X**2/4 + Y**2/1 - 1
17
18 # 绘制等高线图
19 ax.contour(X, Y, Z, levels=[0], colors='black')
20 ax.plot(1, 0, 'r', transform=ax.get_yaxis_transform(), clip_on=False) #
    transform=ax.get_yaxis_transform() 表示箭头在x轴上
21 ax.plot(0, 1, 'b', transform=ax.get_xaxis_transform(), clip_on=False) #
    transform=ax.get_xaxis_transform() 表示箭头在y轴上
22 ax.set_aspect(1)
23 ax.set_title(r'隐函数$\frac{x^2}{4} + \frac{y^2}{1} = 1$')
24
25
26 ax.spines['right'].set_visible(False)
27 ax.spines['top'].set_visible(False)
28 # 改变坐标轴的位置
29 ax.spines['bottom'].set_position(('data', 0))
30 ax.spines['left'].set_position(('data', 0))
31 # 文件保存
32 script_name = os.path.splitext(os.path.basename(__file__))[0]
33 plt.savefig("figures/section3/" + script_name + '.png')
34 plt.show()
```

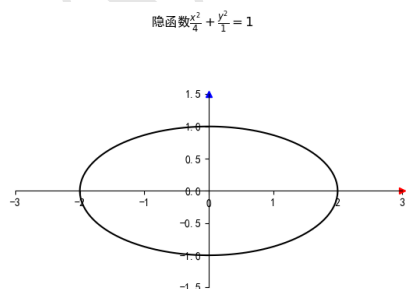


图 3.1: 椭圆

$$\text{隐函数}\frac{x^2}{4} - \frac{y^2}{1} = 1$$

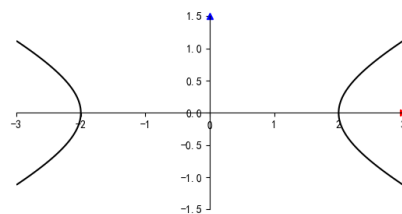


图 3.2: 双曲线

$$\text{隐函数}y^2 = 2x$$

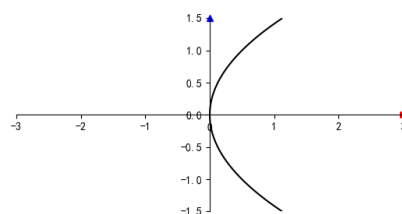


图 3.3: 抛物线

以上可以看出画图的方式是用到了 contour 绘制等高线的方式. 下面我们还可以用到 sympy 库, 利用 sympy 可以直接解出 y 与 x 的关系表达式:  $y = f(x)$

### 2、 圆锥曲线一般方程

圆锥曲线一般方程:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

用圆锥曲线的一般方程来绘制一些图片

```
1 import sympy as sp
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import os
5
6 plt.rcParams['font.sans-serif'] = ['SimHei']
7 plt.rcParams['axes.unicode_minus'] = False
8
9 fig, ax = plt.subplots()
10 ax.set_title('圆锥曲线的一般方程')
11 A, B, C, D, E, F, x, y = sp.symbols('A,B,C,D,E,F,x,y')
12 expr = A*x**2 + B*x*y + C*y**2 + D*x + E*y + F
13 eq = sp.Eq(expr, 0)
14 # 如果圆锥曲线在原点, 则没有x和y的一次项, 也就是D和E为0
```

```
15 subs_dict = {A:-3,B:4,C:-4,D:0,E:0,F:4}
16 result = eq.subs(subs_dict)
17
18 # 解出y关于x的表达式
19 solutions = sp.solve(result, y)
20
21 print(sp.latex(solutions))
22
23 # 绘制每个解
24 x_vals = np.linspace(-10, 10, 2000)
25 for i, sol in enumerate(solutions):
26     # 将符号表达式转换为数值函数
27     y_func = sp.lambdify(x, sol, 'numpy')
28
29     try:
30         y_vals = y_func(x_vals)
31         # 只绘制实数部分
32         mask = np.isreal(y_vals)
33         if np.any(mask):
34             ax.plot(x_vals[mask], np.real(y_vals[mask]), color = 'red',
35                     label=f'解{i+1}', linewidth=2)
36     except Exception as e:
37         print(f"绘制解{i+1}时出错: {e}")
38         continue
39
40 ax.text(0, 0., f"${sp.latex(result)}$",
41         bbox={ 'boxstyle': 'round',
42               'facecolor': 'lightblue',
43               'alpha': 0.7,
44               'pad': 0.5 }
45         , alpha = 0.8)
46
47 offset_x = 0.5 # x方向的偏移量 (向左为负)
48 offset_y = 0.5 # y方向的偏移量 (向下为负)
49 text_x = ax.get_xlim()[0] + offset_x # 获取x轴最小值并加上偏移量
50 text_y = ax.get_ylim()[0] + offset_y # 获取y轴最小值并加上偏移量
51
52 ax.text(text_x, text_y, f"$y={sp.latex(solutions[0])}$", fontsize = 12)
53 ax.text(text_x, text_y + offset_y, f"$y={sp.latex(solutions[1])}$", fontsize =
54         12)
55 # 文件保存
56 script_name = os.path.splitext(os.path.basename(__file__))[0]
57 plt.savefig("figures/section3/" + script_name + '.png')
58 plt.show()
```

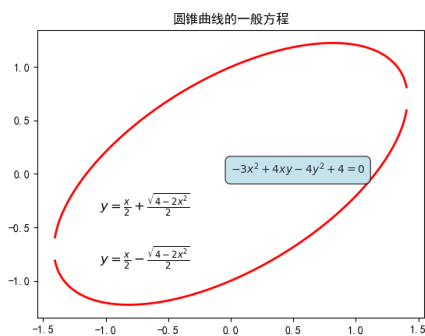


图 3.4: 圆锥曲线一般方程

## 参考文献

- [1] Matplotlib 官方文档, <https://matplotlib.org/stable/contents.html>
- [2] NumPy 官方文档, <https://numpy.org/doc/>

Wu Jia Jun