

## 11

## 2D and 3D Visualizations

## 二维和三维可视化

平面上的散点图、等高线、热图、网格面...



文明的传播像是星星之火可以燎原；首先是星星之火，然后是闪烁的炬火，最后是燎原烈焰，排山倒海、势不可挡。

*The spread of civilization may be likened to a fire; first, a feeble spark, next a flickering flame, then a mighty blaze, ever increasing in speed and power.*

—— 尼古拉·特斯拉 (Nikola Tesla) | 发明家、物理学家 | 1856 ~ 1943



- ◀ Axes3D.plot\_surface() 绘制三维曲面
- ◀ matplotlib.pyplot.contour() 绘制等高线图，轴对象可以为三维
- ◀ matplotlib.pyplot.contourf() 绘制平面填充等高线，轴对象可以为三维
- ◀ numpy.cumsum() 计算给定数组中元素的累积和，返回一个具有相同形状的数组
- ◀ numpy.exp() 计算给定数组中每个元素的 e 的指数值
- ◀ numpy.linspace() 在指定的范围内创建等间隔的一维数组
- ◀ numpy.meshgrid() 生成多维网格化数组
- ◀ plotly.express.data.iris() 导入鸢尾花数据集
- ◀ plotly.express.imshow() 绘制可交互的热图
- ◀ plotly.express.line() 创建可交互的折线图的图形
- ◀ plotly.express.scatter() 创建可交互的散点图
- ◀ plotly.express.scatter\_3d() 创建可交互的三维散点图
- ◀ plotly.graph\_objects.Contour() 绘制可交互的等高线图
- ◀ plotly.graph\_objects.Scatter3d() 绘制可交互的散点、线图
- ◀ plotly.graph\_objects.Surface() 绘制可交互的三维曲面
- ◀ seaborn.heatmap() 绘制热图
- ◀ seaborn.load\_dataset() Seaborn 库中用于加载示例数据集
- ◀ seaborn.scatterplot() 创建散点图



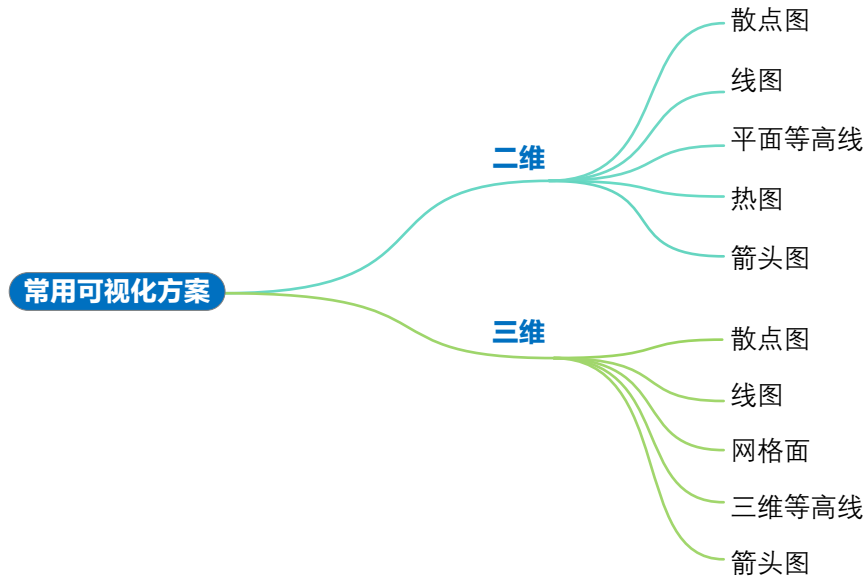
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



## 11.1 二维可视化方案

上一章，我们介绍了如何用 Matplotlib 和 Plotly 绘制线图，本章将分别介绍《编程不难》中常用的二维和三维可视化方案。

散点、线图、等高线、热图是鸢尾花书最常见的四类平面可视化方案。

- ▶ **散点图 (scatter plot)**: 散点图用于展示两个变量之间的关系，其中每个点的位置表示两个变量的取值。可以通过设置点的颜色、大小、形状等属性来表示其他信息。
- ▶ **线图 (line plot)**: 线图用于展示数据随时间或其他变量而变化的趋势。线图由多个数据点连接而成，通常用于展示连续数据。
- ▶ **等高线图 (contour plot)**: 等高线图用于展示二维数据随着两个变量的变化而变化的趋势。每个数据点的值表示为等高线的高度，从而形成连续的轮廓线。
- ▶ **热图 (heatmap)**: 热图用于展示二维数据的值，其中每个值用颜色表示。热图常用于数据分析中，用于显示数据的热度、趋势等信息。建议使用 Seaborn 库绘制热图。

下面，我们先从二维可视化方案说起。

## 11.2 平面散点

二维散点图是平面直角坐标系（也叫笛卡儿坐标系）中一种用于可视化二维数据分布的图形表示方法。它由一系列离散的数据点组成，其中每个数据点都由两个坐标值。

Matplotlib 中的 `scatter()` 函数可以用于创建散点图。可以使用 `matplotlib.pyplot.scatter()` 函数来指定数据点的坐标和其他绘图参数，例如颜色、大小等。



鸢尾花书《数学要素》第 5 章专门讲解笛卡儿坐标系。

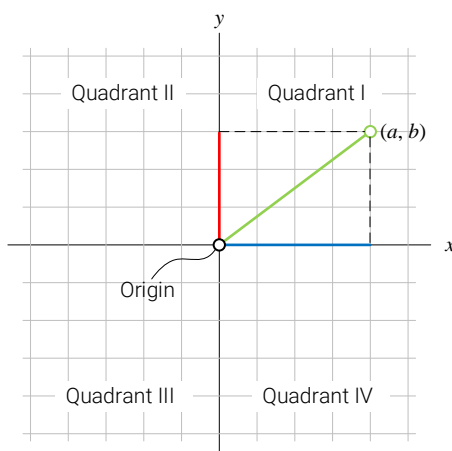


图 1. 笛卡儿坐标系，平面直角坐标系

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



### 什么是平面直角坐标系？

平面直角坐标系，也称笛卡儿坐标系，是一种二维空间中的坐标系统，由两条相互垂直的直线组成。其中一条直线称为  $x$  轴，另一条直线称为  $y$  轴。它们的交点称为原点，通常用  $O$  表示。平面直角坐标系可以用来描述二维空间中点的位置，其中每个点都可以由一对有序实数  $(a, b)$  表示，分别表示点在  $a$  轴和  $b$  轴上的距离。 $x$  轴和  $y$  轴的正方向可以是任意方向，通常  $x$  轴向右， $y$  轴向上。平面直角坐标系是解析几何中重要的工具，用于研究点、直线、曲线以及它们之间的关系和性质。

特别推荐大家使用 `seaborn.scatterplot()` 函数来创建二维散点图，并传递数据点的坐标和其他可选参数。

大家还可以使用 `plotly.express.scatter()` 和 `plotly.graph_objects.Scatter()` 函数创建可交互的散点图，并指定数据点的坐标、样式等参数。本节下面利用 Seaborn 和 Plotly 这两个库中函数绘制散点图。

## Seaborn

图 2 所示为利用 `seaborn.scatterplot()` 绘制鸢尾花数据集的散点图。这两幅散点图的横轴都是花萼长度，纵轴为花萼宽度。图 2 (b) 用颜色标识鸢尾花类别。

使用 `seaborn.scatterplot()` 函数的基本语法如下：

```
import seaborn as sns
sns.scatterplot(data=data_frame, x="x_variable", y="y_variable")
```

其中，`x_variable` 是数据集中表示  $x$  轴的变量列名，`y_variable` 是表示  $y$  轴的变量列名，`data_frame` 是包含要绘制的数据的 Pandas DataFrame 对象。

我们还可以指定 `hue` 参数，用于对数据点进行分组并在图中用不同颜色表示的列名，`size` 参数指定了数据点的大小根据 `value` 列的值进行缩放。除了 `hue` 和 `size`，还可以使用其他参数如 `style`、`palette`、`alpha` 等来进一步定制散点图的外观和风格。

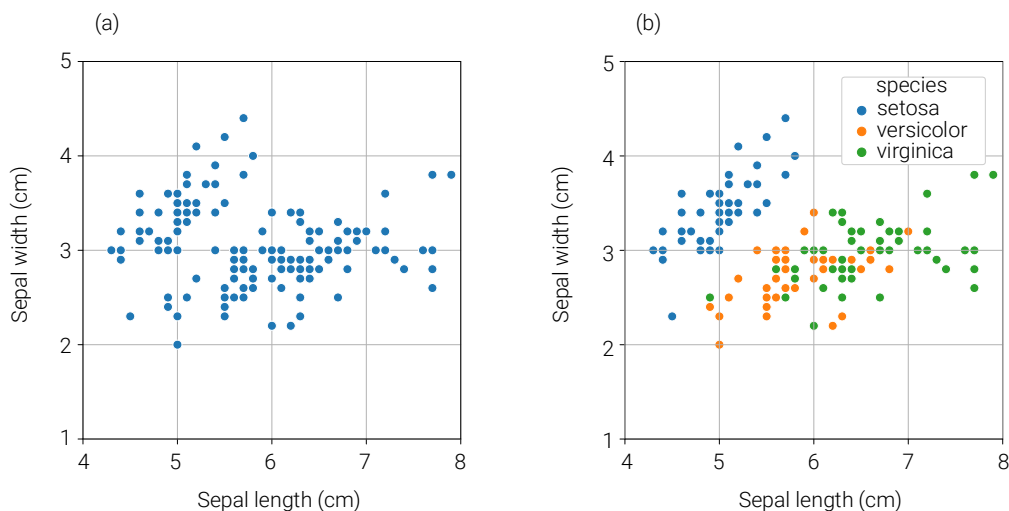


图 2. 使用 `seaborn.scatterplot()` 绘制鸢尾花数据集散点图

代码 1 绘制图 2，下面讲解其中关键语句。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

**a** 从 `sklearn.datasets` 模块中导入了一个叫做 `load_iris` 的函数。这个函数的作用是加载经典的鸢尾花数据集。大家对鸢尾花数据集已经不陌生。简单来说，鸢尾花数据集是一个常用于机器学习和统计学习的示例数据集，其中包含了三种不同品种的鸢尾花（`setosa`、`versicolor` 和 `virginica`）各 50 个样本，总计 150 个样本。每个样本有四个特征，**花萼长度**（`sepal length`）、**花萼宽度**（`sepal width`）、**花瓣长度**（`petal length`）、**花瓣宽度**（`petal width`）。

**b** 用 `load_iris()` 加载鸢尾花数据。大家通过 `type(iris)` 可以发现 `iris` 的数据类型为 `sklearn.utils._bunch.Bunch`，它是 `scikit-learn` 中一个简单的数据容器类。这种数据类型类似字典 `dict`，但提供了一些额外的便捷方法和属性。

比如，`iris.data` 包含特征数据的数组，具体类型为 `NumPy Array`。每一行代表数据集中的一个样本，每一列代表一个特征（花萼长度、花萼宽度、花瓣长度和花瓣宽度）。

`iris.target` 包含目标标签的数组。对于每个样本，`target` 中的相应元素是该样本所属的类别（`setosa` 对应 0、`versicolor` 对应 1、`virginica` 对应 2）。

`iris.target_names` 包含目标标签的数组，即 `['setosa', 'versicolor', 'virginica']`。

`feature_names` 包含特征名称的数组，即 `['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']`。

**c** 用 `iris.data[:, 0]` 提取 `NumPy` 数组的索引为 0 列，也就是第 1 例。`[:, 0]` 中 `:` 代表选择所有行，0 代表选择第一列。`iris.data[:, 0]` 为花萼长度样本数据。



本书第 14 章将专门介绍 `NumPy` 数组的索引和切片。

**d** 用 `iris.data[:, 1]` 提取 `NumPy` 数组的索引为 1 列，也就是第 2 例。`iris.data[:, 1]` 为花萼宽度样本数据。

**e** 提取鸢尾花数据集标签数组。大家可以试着用 `np.unique(iris.target)` 获取数组独特值，结果为 `array([0, 1, 2])`。使用 `numpy.unique()` 时，大家可以用 `np.unique(iris.target, return_counts=True)` 获取独特值的计数（频率），结果为元组 `tuple (array([0, 1, 2]), array([50, 50, 50], dtype=int64))`。

**f** 用 `matplotlib.pyplot.subplots()`，简做 `plt.subplots()`，创建图形对象 `fig`、轴对象 `ax`。数组 `sepal_length` 是横轴上的数据点，代表每个样本的花萼长度。

数组 `sepal_width` 是纵轴上的数据点，代表每个样本的花萼宽度。`c=target` 指定了散点的颜色，颜色由 `target` 数组决定。

`cmap='rainbow'` 指定了颜色映射。`target` 中的三个值（0、1、2）将通过 `'rainbow'` 映射到三个颜色，表达鸢尾花三个类别。

**g** 用 `matplotlib.pyplot.scatter()`，简做 `plt.scatter()`，“提笔就画”散点图。

**h** 装饰散点图，请大家逐行注释。并且试着用轴对象 `ax` 方法替换这三句的 `plt` 方法。

**i** 设置横纵轴刻度。请大家注释 `np.arange(4, 8 + 1, step=1)` 的用法。

**j** 将横纵轴比例尺设置为 1:1。

**k** 增加网格线，请大家用 `ls` 代替 `linestyle`，用 `lw` 代替 `linewidth`，用 `c` 代替 `color`，重写这一句。

**l** 设置横纵轴取值范围。请大家利用 `ax.set_xlim()` 和 `ax.set_ylim()` 替换这两句。

```

# 导入包
import matplotlib.pyplot as plt
a from sklearn.datasets import load_iris
import numpy as np

# 加载鸢尾花数据集
b iris = load_iris()

# 提取花萼长度和花萼宽度作为变量
c sepal_length = iris.data[:, 0]
d sepal_width = iris.data[:, 1]
e target = iris.target

f fig, ax = plt.subplots()

# 创建散点图
g plt.scatter(sepal_length, sepal_width, c=target, cmap='rainbow')

# 添加标题和轴标签
h plt.title('Iris sepal length vs width')
plt.xlabel('Sepal length (cm)')
plt.ylabel('Sepal width (cm)')

# 设置横纵轴刻度
i ax.set_xticks(np.arange(4, 8 + 1, step=1))
ax.set_yticks(np.arange(1, 5 + 1, step=1))

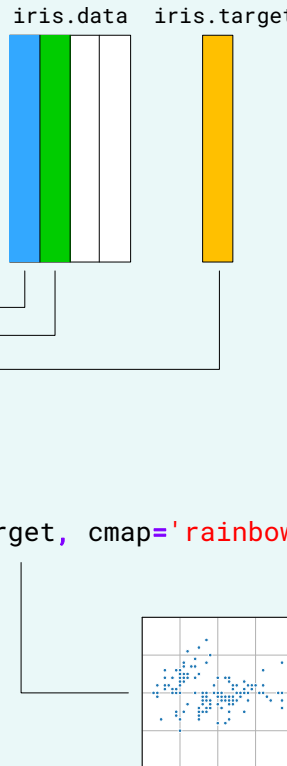

# 设定横纵轴尺度1:1
j ax.axis('scaled')

# 增加刻度网格，颜色为浅灰
k ax.grid(linestyle='--', linewidth=0.25, color=[0.7,0.7,0.7])

# 设置横纵轴范围
l ax.set_xbound(lower = 4, upper = 8)
ax.set_ybound(lower = 1, upper = 5)

# 显示图形
plt.show()

```


代码 1. 用 Matplotlib 绘制散点图;  Bk1\_Ch11\_01.ipynb

## Plotly

图 3 所示为使用 `plotly.express.scatter()` 绘制鸢尾花数据集散点图。在本章配套的 Jupyter Notebook 中大家可以看到这两幅子图为可交互图像。

`plotly.express.scatter()` 用来可视化两个数值变量之间的关系，或者展示数据集中的模式和趋势。这个函数的基本语法如下：

```

import plotly.express as px
fig = px.scatter(data_frame, x="x_variable", y="y_variable")

```

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

```
fig.show()
```

其中, `data_frame` 是包含要绘制的数据的 Pandas DataFrame 对象, `x_variable` 是数据集中表示  $x$  轴的变量列名, `y_variable` 是表示  $y$  轴的变量列名。可以根据需要添加其他参数, 例如 `color`、`size`、`symbol` 等, 以进一步定制散点图的外观。

最后, 通过 `fig.show()` 方法显示绘制好的散点图。



《可视之美》将专门讲解散点图。

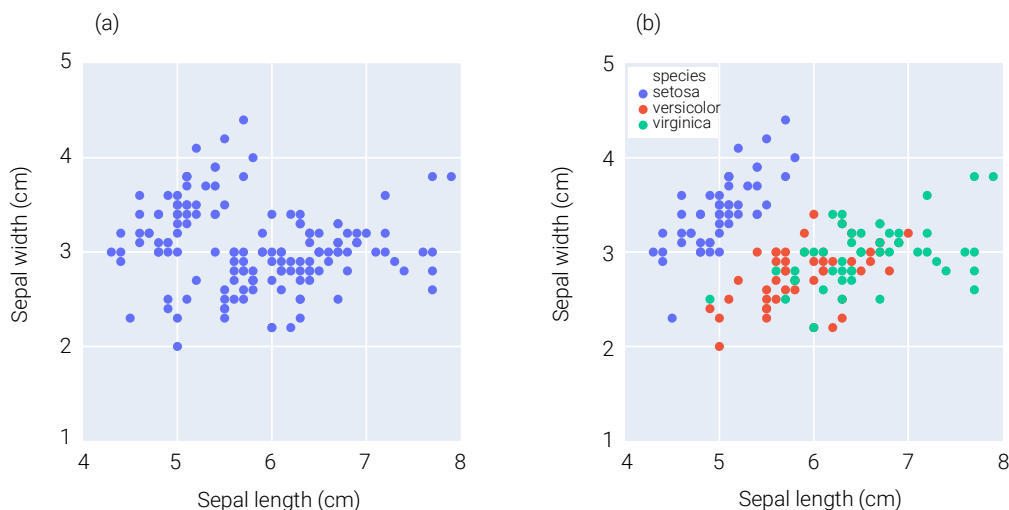


图 3. 使用 `plotly.express.scatter()` 绘制鸢尾花数据集散点图

代码 2 绘制图 3, 下面讲解其中关键语句。

- a** 将 `plotly.express` 模块导入, 简作 `px`。
- b** 用 `plotly.express.data.iris()`, 简作 `px.data.iris()`, 导入鸢尾花数据。数据类型为 Pandas DataFrame。
- c** 用 `plotly.express.scatter()`, 简作 `px.scatter()`, 绘制散点图。  
`iris_df` 为绘制散点图所需要的数据。  
`x="sepal_length"` 和 `y="sepal_width"` 指定了在散点图横纵轴分别使用数据帧 `iris_df` 哪两个特征。  
`width=600` 和 `height=600` 指定了图形的宽度和高度, 分别设置为 600 像素。  
`labels={"sepal_length": "Sepal length (cm)", "sepal_width": "Sepal width (cm)"}` 将横轴标签设置为 "Sepal length (cm)", 纵轴标签设置为 "Sepal width (cm)". 默认标签为数据帧列标签。
- d** 用 Plotly 的 `update_layout` 方法来调整横纵轴的取值范围。`xaxis_range=[4, 8]` 设置横轴的范围为从 4 到 8, `yaxis_range=[1, 5]` 设置纵轴的范围为从 1 到 5。
- e** 和 **f** 也用 `update_layout` 方法调整横纵轴刻度。实际上, **d**、**e**、**f** 这三句可以合并, 但是为了让大家看清图片修饰的具体细节, 我们把它们分开来写。
- g** 类似 **c**, 也是用 `plotly.express.scatter()`, 简作 `px.scatter()`, 绘制散点图; 不同的是, 我们指定 `color="species"` 渲染散点颜色, 可视化鸢尾花分类。

本 PDF 文件为作者草稿, 发布目的为方便读者在移动终端学习, 终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有, 请勿商用, 引用请注明出处。

代码及 PDF 文件下载: <https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教, 本书专属邮箱: [jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



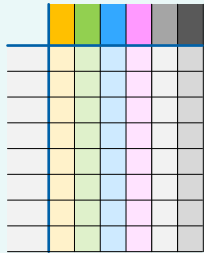
**h** 也是用 `update_layout` 方法将图例位置调整为左上角，并微调具体位置。

`yanchor="top"` 设置图例的垂直锚点为顶部，即图例的上边缘与指定的纵轴值对齐。

`y=0.99` 指定图例上边缘相对于图形区域顶部的位置。在这里，`0.99` 是个相对值，表示图例的上边缘距离图形区域顶部的距离占整个图形区域高度的 99%。

`xanchor="left"` 设置图例的水平锚点为左侧，即图例的左边缘将与指定的 `x` 值对齐。

`x=0.01` 指定图例左边缘相对于图形区域左侧的位置。在这里，`0.01` 也是相对值，表示图例的左边缘距离图形区域左侧的距离占整个图形区域宽度的 1%。



```

# 导入包
import numpy as np
a import plotly.express as px

# 从Plotly中导入鸢尾花样本数据
b iris_df = px.data.iris()

# 绘制散点图，不渲染marker
c fig = px.scatter(iris_df, x="sepal_length", y="sepal_width",
                  width = 600, height = 600,
                  labels={"sepal_length": "Sepal length (cm)",
                        "sepal_width": "Sepal width (cm)"})

# 修饰图像
d fig.update_layout(xaxis_range=[4, 8], yaxis_range=[1, 5])
    xticks = np.arange(4,8+1)
    yticks = np.arange(1,5+1)
e fig.update_layout(xaxis = dict(tickmode = 'array',
                                tickvals = xticks))
f fig.update_layout(yaxis = dict(tickmode = 'array',
                                tickvals = yticks))

fig.show()


# 绘制散点图，渲染marker展示鸢尾花分类
g fig = px.scatter(iris_df, x="sepal_length", y="sepal_width",
                  color="species",
                  width = 600, height = 600,
                  labels={"sepal_length": "Sepal length (cm)",
                        "sepal_width": "Sepal width (cm)"})

# 修饰图像
fig.update_layout(xaxis_range=[4, 8], yaxis_range=[1, 5])
fig.update_layout(xaxis = dict(tickmode = 'array',
                                tickvals = xticks))
fig.update_layout(yaxis = dict(tickmode = 'array',
                                tickvals = yticks))

h fig.update_layout(legend=dict(yanchor="top", y=0.99,
                                xanchor="left", x=0.01))

fig.show()

```

代码 2. 用 Plotly 绘制散点图;  Bk1\_Ch11\_02.ipynb

## 导入鸢尾花数据三个不同途径

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



大家可能发现，我们经常从不同的 Python 第三方库导入鸢尾花数据。本例用 `sklearn.datasets.load_iris()`，这是因为 `sklearn` 中的鸢尾花数据将特征数据和标签数据分别保存，而且数据类型都是 NumPy Array，方便使用 `matplotlib` 绘制散点图。此外，NumPy Array 数据类型还方便调用 NumPy 中的线性代数函数。

此外，我们也用 `seaborn.load_dataset("iris")` 导入鸢尾花数据集，数据类型为 Pandas DataFrame。数据帧的列标签为 'sepal\_length'、'sepal\_width'、'petal\_length'、'petal\_width'、'species'。其中，标签中的独特值为三个字符串 'setosa'、'versicolor'、'virginica'。

Pandas DataFrame 获取某列独特值的函数为 `pandas.unique()`。这种数据类型方便利用 Seaborn 进行统计可视化。此外，Pandas DataFrame 也特别方便利用 Pandas 的各种数据帧工具。

下一章会专门介绍利用 Seaborn 绘制散点图和其他常用统计可视化方案。

在利用 Plotly 可视化鸢尾花数据时，我们会直接从 Plotly 中用 `plotly.express.data.iris()` 导入鸢尾花数据，数据类型也是 Pandas DataFrame。这个数据帧的列标签为 'sepal\_length'、'sepal\_width'、'petal\_length'、'petal\_width'、'species'、'species\_id'。前五列和 Seaborn 中鸢尾花数据帧相同，不同的是 'species\_id' 这一列的标签为整数 0、1、2。

## 11.3 平面等高线

### 等高线原理

等高线图是一种展示三维数据的方式，其中相同数值的数据点被连接成曲线，形成轮廓线。

形象地说，如图 4 所示，二元函数相当于一座山峰。在平行于  $x_1x_2$  平面在特定高度切一刀，得到的轮廓线就是一条等高线。这是一条三维空间等高线。然后，将等高线投影到  $x_1x_2$  平面，我们便得到一条平面等高线。

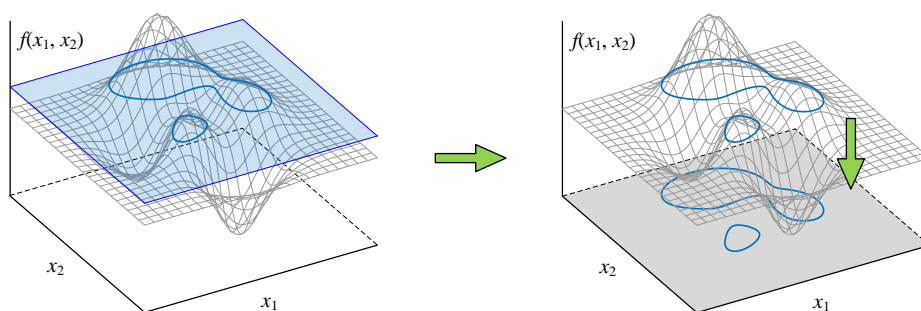


图 4. 平行  $x_1x_2$  平面切  $f(x_1, x_2)$  获得等高线，然后等高线投影到  $x_1x_2$  平面



#### 什么是二元函数？

二元函数是指具有两个自变量和一个因变量的函数。它接受两个输入，并返回一个输出。一般表示为  $y=f(x_1, x_2)$ ，其中  $x_1$  和  $x_2$  是自变量， $y$  是因变量。二元函数常用于描述和分析具有两个相关变量之间关系的数学模型。它可以用于表示二维空间中的曲面、表达物理或经济关系、进行数据建模和预测等。在可视化二元函数时，常使用三维图形或等高线图。三维图形以  $x_1$  和  $x_2$  作为坐标轴，将因变量  $y$  的值映射为曲面的高度。等高线图则使用等高线来表示  $y$  值的等值线，轮廓线的密集程度反映了函数值的变化。

一系列轮廓线的高度一般用不同的颜色或线型表示，使得我们可以通过视觉化方式看到数据的分布情况。如图 5 所示，将一组不同高度的等高线投影到平面便得到右侧平面等高线。右侧子图还增加了色

谱条，用来展示不同等高线对应的具体高度。这一系列高度可以是一组用户输入的数值。大家可能已经发现，等高线图和海拔高度图原理完全相同。类似的图还有，等温线、等降水线、等距线等等。

Matplotlib 的填充等高线是在普通等高线的基础上添加填充颜色来表示不同区域的数据密度。可以使用 `contourf()` 函数来绘制填充等高线。

图 5 左图则是三维等高线，这是下一章要介绍的内容。

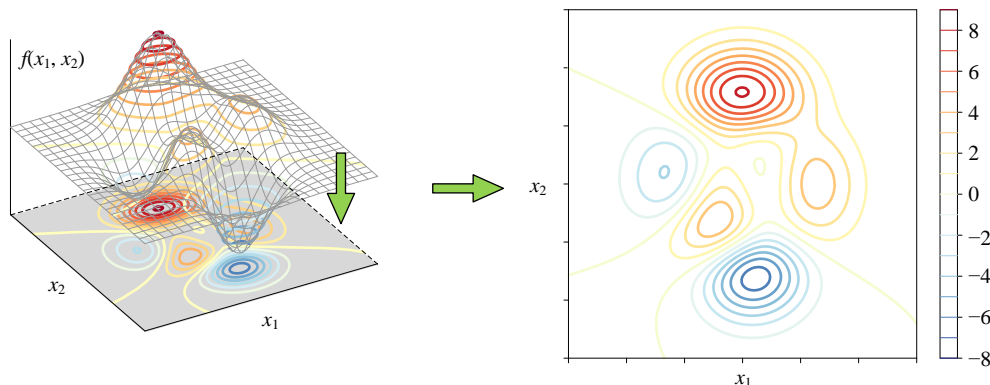


图 5. 将不同高度值对应的一组等高线投影到  $x_1x_2$  平面

## 网格数据

为了绘制平面等高线，我们需要利用 `numpy.meshgrid()` 产生网格数据。相信大家对网格数据这个概念已经很熟悉，本书前文自定义函数生成网格化数据的二维数组。

简答来说，`numpy.meshgrid()` 接受若干一维数组作为输入，并生成二维、三维乃至多维数组来表示网格坐标。

原理上，如图 6 所示，`numpy.meshgrid()` 函数会将输入的一维数 (`x1_array` 和 `x2_array`) 组扩展为二维数组 (`xx1` 和 `xx2`)，其中一个数组的每一行都是输入数组的复制，而另一个数组的每一列都是输入数组的复制。这样，通过组合这两个二维数组的元素，就形成了一个二维网格。

fx

```
xx1, xx2 = numpy.meshgrid(x1, x2)
```

提供两个一维数组 `x1` 和 `x2` 作为输入。函数将生成两个二维数组 `xx1` 和 `xx2`，用于表示一个二维网格。

请大家在 JupyterLab 中自行学习下例。

```
import numpy as np
```

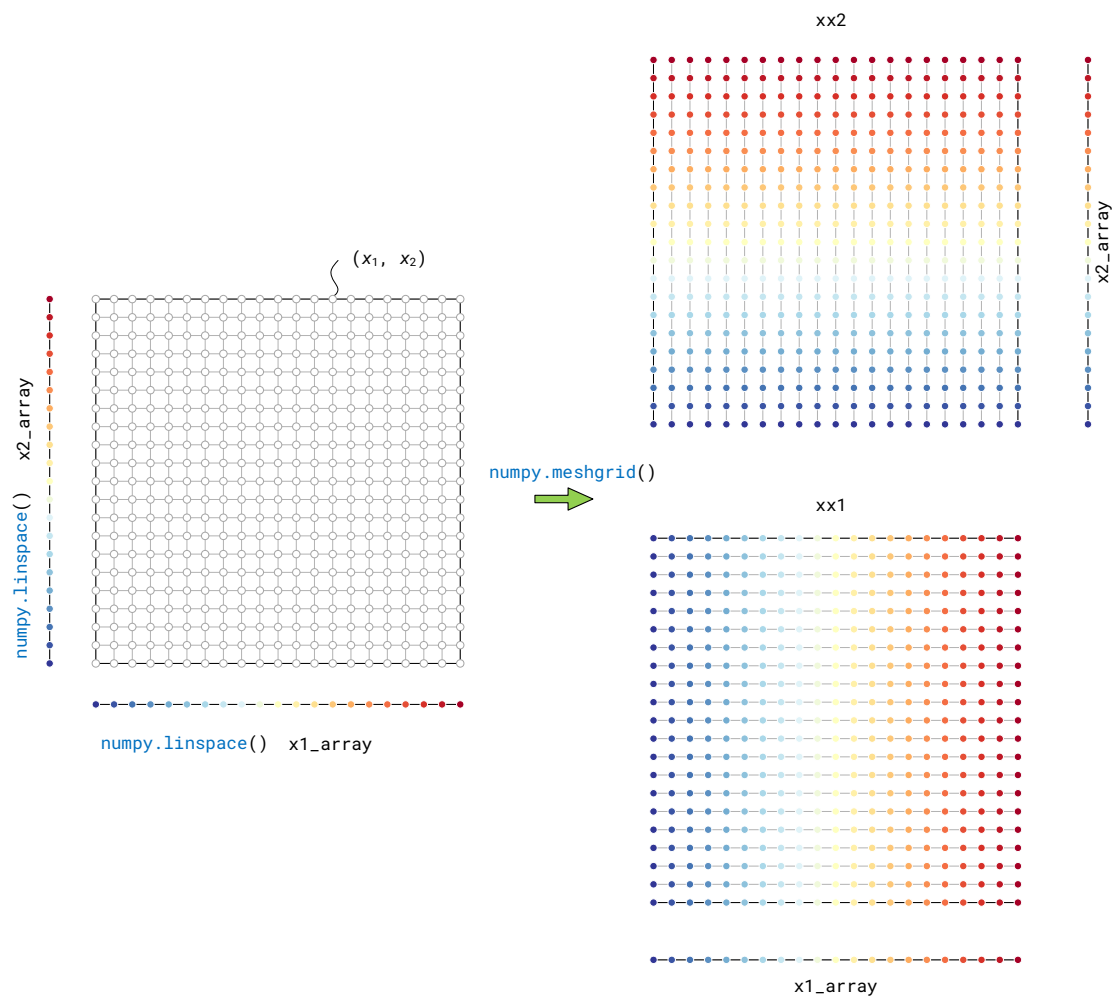
```
x1 = np.arange(10)
```

```
# 第一个一维数组
```

```
x2 = np.arange(5)
```

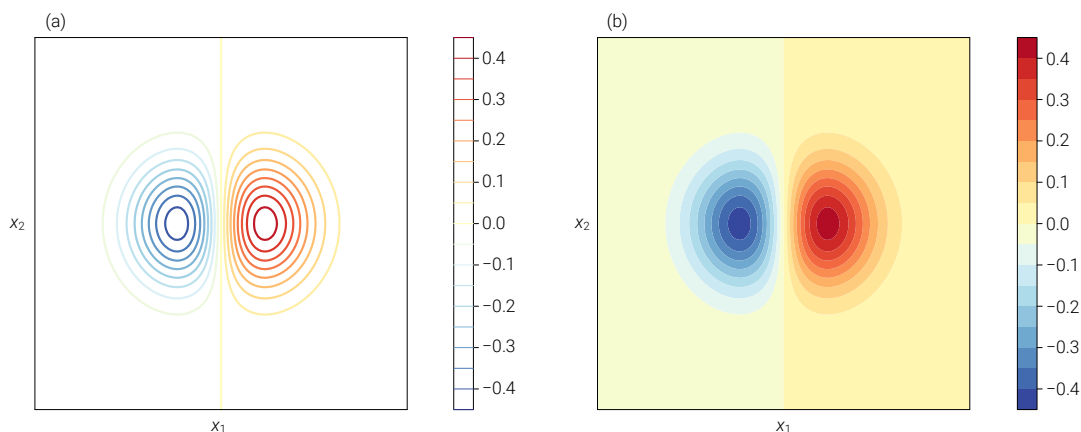
```
# 第二个一维数组
```

```
xx1, xx2 = np.meshgrid(x1, x2)
```

图 6. 用 `numpy.meshgrid()` 生成二维网络数据

## Matplotlib

图 7 所示为利用 `Matplotlib` 中等高线可视化二元函数  $f(x_1, x_2) = x_1 \exp(-x_1^2 - x_2^2)$ 。

图 7. 用 `Matplotlib` 生成的平面等高线

填充等高线的原理是通过在等高线之间创建颜色渐变来表示不同区域的数值范围。这样可以增强等高线图的可视化效果，更直观地展示数据的分布和变化。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

在 Matplotlib 中，填充等高线可以通过使用 `contourf()` 函数实现。该函数与 `contour()` 函数类似，但会填充等高线之间的区域。



`matplotlib.pyplot.contour(X,Y,Z,levels,cmap)`

下面是 `contour()` 函数的常用输入参数：

- **X**: 二维数组，表示数据点的横坐标。
- **Y**: 二维数组，表示数据点的纵坐标。
- **Z**: 二维数组，表示数据点对应的函数值或高度。
- **levels**: 用于指定绘制的等高线层级或数值列表。
- **colors**: 用于指定等高线的颜色，可以是单个颜色字符串、颜色序列或 `colormap` 对象。
- **cmap**: 颜色映射，用于将数值映射为颜色。可以是预定义的 `colormap` 名称或 `colormap` 对象。
- **linestyles**: 用于指定等高线的线型，可以是单个线型字符串或线型序列。
- **linewidths**: 用于指定等高线的线宽，可以是单个线宽值或线宽序列。
- **alpha**: 用于指定等高线的透明度。

请大家在 JupyterLab 中自行学习下列。

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

# 创建二维数据
x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)
Z = X**2 + Y**2 # 示例函数，可以根据需要自定义

# 绘制等高线图
plt.contour(X, Y, Z, levels = np.linspace(0,8,16 + 1), cmap = 'RdYlBu_r')

# 添加颜色图例
plt.colorbar()

# 显示图形
plt.show()
```

代码 3 生成图 7 两幅子图，下面聊聊其中关键词句。

**a** 和 **b** 用 `numpy.linspace()`，简作 `np.linspace()`，生成等差数列。数据类型都是 NumPy Array。

**c** 利用 `numpy.meshgrid()`，简作 `np.meshgrid()`，构造网格坐标点。

**d** 计算这些坐标点的二元函数值。

**e** 利用 `matplotlib.pyplot.subplots()`，简作 `plt.subplots()`，生成图像对象 `fig` 和轴对象 `ax`。

**f** 在轴对象 `ax` 上用 `contour()` 绘制平面等高线。

`xx1`，`xx2`，`ff` 这三个参数是数据，用于表示在二维平面上的函数 `ff` 的等高线。`xx1` 和 `xx2` 是坐标网格，`ff` 是这个网格上的函数值。

`levels=20` 是指定等高线的数量。

`cmap='RdYlBu_r'` 是指定等高线颜色映射的参数。例子中使用了 `'RdYlBu_r'`，表示红、黄、蓝渐变的颜色映射，`_r` 表示翻转。

`linewidths=1` 指定等高线的线宽为 1 pt。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

这句返回值 CS 是一个等高线图的对象。这个对象包含了等高线图的信息，比如线条的位置、颜色等。

**g** 用 `colorbar()` 方法在图形对象 `fig` 上添加颜色条的语句。CS 是之前创建的等高线图对象，颜色条将基于这个对象的颜色映射进行创建。

**h** 类似之前代码，只不过用的是 `contourf()` 方法在 `ax` 上绘制平面填充等高线。

```
# 导入包
import numpy as np
import matplotlib.pyplot as plt

# 生成数据
a x1_array = np.linspace(-3,3,121)
b x2_array = np.linspace(-3,3,121)

c xx1, xx2 = np.meshgrid(x1_array, x2_array)
d ff = xx1 * np.exp(- xx1**2 - xx2 **2)

# 等高线
e fig, ax = plt.subplots()


f CS = ax.contour(xx1, xx2, ff, levels = 20,
                  cmap = 'RdYlBu_r', linewidths = 1)

g fig.colorbar(CS)
ax.set_xlabel('$\\it{x_1}$'); ax.set_ylabel('$\\it{x_2}$')
ax.set_xticks([]); ax.set_yticks([])
ax.set_xlim(xx1.min(), xx1.max())
ax.set_ylim(xx2.min(), xx2.max())
ax.grid(False)
ax.set_aspect('equal', adjustable='box')

# 填充等高线
fig, ax = plt.subplots()

h CS = ax.contourf(xx1, xx2, ff, levels = 20,
                  cmap = 'RdYlBu_r')

fig.colorbar(CS)
ax.set_xlabel('$\\it{x_1}$'); ax.set_ylabel('$\\it{x_2}$')
ax.set_xticks([]); ax.set_yticks([])
ax.set_xlim(xx1.min(), xx1.max())
ax.set_ylim(xx2.min(), xx2.max())
ax.grid(False)
ax.set_aspect('equal', adjustable='box')
```

代码 3. 用 Matplotlib 生成平面等高线;  Bk1\_Ch11\_03.ipynb

## Plotly

图 8 所示为利用 `plotly.graph_objects.Contour()` 绘制的（填充）等高线。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

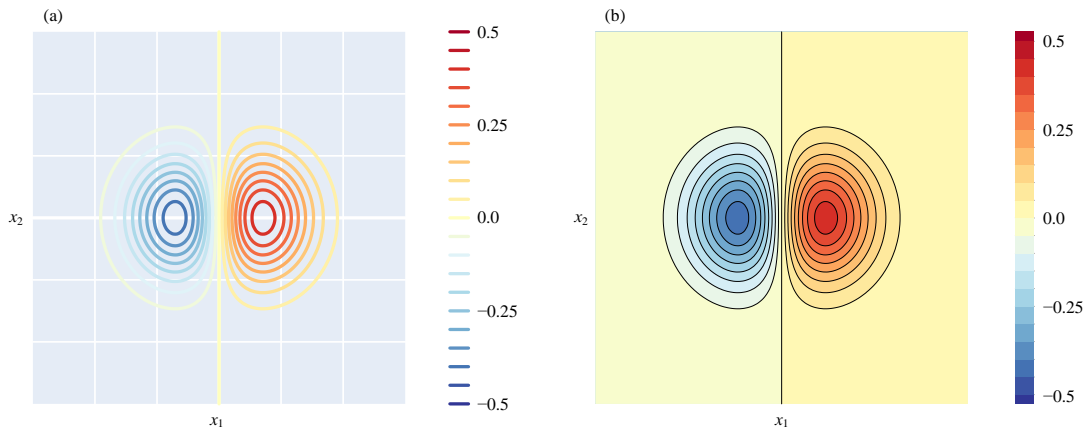


图 8. 用 Plotly 生成的平面 (填充) 等高线

代码 4 绘制图 8，下面聊聊其中关键语句。

**a** 导入 Plotly 库中的 `graph_objects` 模块，简作 `go`。模块 `plotly.graph_objects` 有丰富的可视化方案。

**b** 构造一个字典 `dict(start=-0.5, end=0.5, size=0.05)`，包含了等高线的设置，包括开始值 (`start`)、结束值 (`end`) 和间隔大小 (`size`)。

**c** 用 `plotly.graph_objects.Contour()`，简作 `go.Contour()`，创建 Plotly 的等高线对象。

`x=x1_array, y=x2_array, z=ff` 这三个参数分别是 `x`、`y`、`z` 轴的数据，用于表示二维平面上的二元函数坐标点。

参数 `contours_coloring='lines'` 指定了等高线的着色方式，这里是使用线条颜色。

参数 `line_width=2` 指定等高线的线宽为 2 pt。

参数 `colorscale='RdYlBu_r'` 指定等高线颜色映射。

参数 `contours=levels` 这是指定等高线参数，使用了之前定义的 `levels` 字典。

**d** 用 `plotly.graph_objects.Layout()`，简作 `go.Layout()`，创建 Plotly 的图形布局对象。

`width=600, height=600` 这两个参数分别设置了图形的宽度和高度，单位是像素。

`xaxis=dict(title=r'$x_1$')` 设置 `x` 轴标题。`r'$x_1$'` 中的 `r` 表示将字符串按照原始字符串处理，`$x_1$` 是用于显示数学符号的 LaTeX 语法。

同理，`yaxis=dict(title=r'$x_2$')` 设置 `y` 轴标题。同样，`r'$x_2$'` 是用于显示数学符号的 LaTeX 语法。

**e** 用 `plotly.graph_objects.Figure()`，简作 `go.Figure()`，创建 Plotly 的图形对象 `fig`。

`data=data` 是之前定义的包含图形数据信息的对象，即等高线对象。

`layout=layout` 是之前定义的包含图形布局信息的对象，用于设置图形的外观和布局。

**f** 通过 `fig.show()` 显示交互图形。



```

# 导入包
import numpy as np
import matplotlib.pyplot as plt
a import plotly.graph_objects as go
# 生成数据
x1_array = np.linspace(-3,3,121)
x2_array = np.linspace(-3,3,121)

xx1, xx2 = np.meshgrid(x1_array, x2_array)
ff = xx1 * np.exp(- xx1**2 - xx2 **2)


# 等高线设置
b levels = dict(start=-0.5,end=0.5,size=0.05)
c data = go.Contour(x=x1_array,y=x2_array,z=ff,
                    contours_coloring='lines',
                    line_width=2,
                    colorscale = 'RdYlBu_r',
                    contours=levels)

# 创建布局
d layout = go.Layout(
    width=600, # 设置图形宽度
    height=600, # 设置图形高度
    xaxis=dict(title=r'$x_1$'),
    yaxis=dict(title=r'$x_2$'))

# 创建图形对象
e fig = go.Figure(data=data, layout=layout)

f fig.show()

```

代码 4. 用 Plotly 生成平面等高线;  Bk1\_Ch11\_04.ipynb

## 11.4 热图

在 Matplotlib 中, 可以使用 `matplotlib.pyplot.imshow()` 函数来绘制热图 (heatmap), 也叫**热力图**。`imshow()` 函数可以将二维数据矩阵的值映射为不同的颜色, 从而可视化数据的密度、分布或模式。

鸢尾花书中一般会用 Seaborn 绘制静态热图, 特别是在可视化矩阵运算。

*fx*

`seaborn.heatmap(data, vmin, vmax, cmap, annot)`

下面是函数的常用输入参数:

- **data**: 二维数据数组, 要绘制的热图数据。
- **vmin**: 可选参数, 指定热图颜色映射的最小值。
- **vmax**: 可选参数, 指定热图颜色映射的最大值。

本 PDF 文件为作者草稿, 发布目的为方便读者在移动终端学习, 终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有, 请勿商用, 引用请注明出处。

代码及 PDF 文件下载: <https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教, 本书专属邮箱: [jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



- **cmap**: 可选参数, 指定热图的颜色映射。可以是预定义的颜色映射名称或 **colormap** 对象。
- **annot**: 可选参数, 控制是否在热图上显示数据值。默认为 **False**, 不显示数据值; 设为 **True** 则显示数据值。
- **xticklabels**: 可选参数, 控制是否显示 X 轴的刻度标签。可以是布尔值或标签列表。
- **yticklabels**: 可选参数, 控制是否显示 Y 轴的刻度标签。可以是布尔值或标签列表。

请大家在 JupyterLab 中自行学习下例。

```
import seaborn as sns
import numpy as np

# 创建二维数据
data = np.random.rand(10,10)

# 绘制热图
sns.heatmap(data, vmin=0, vmax=1,
             cmap='viridis',
             annot=True,
             xticklabels=True,
             yticklabels=True)
```

图 9 所示为分别用 Seaborn 和 Plotly 热图可视化鸢尾花数据集四个量化特征数据。

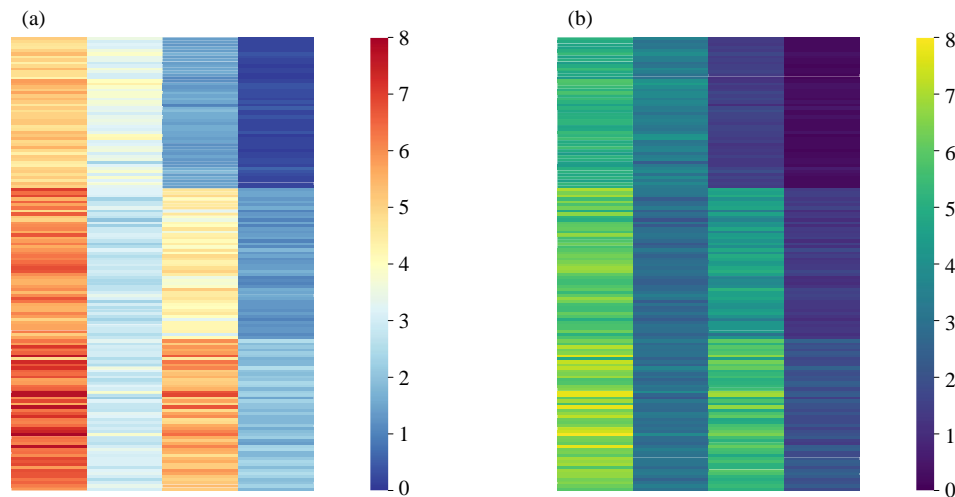


图 9. 使用 Seaborn、Plotly 热图可视化鸢尾花数据集

代码 5 绘制图 9 (a), 下面聊聊其中关键词句。

**a** 用 `seaborn.load_dataset()`, 简作 `sns.load_dataset()`, 导入鸢尾花数据。

**b** 用 `seaborn.heatmap()`, 简作 `sns.heatmap()`, 绘制热图, 展示鸢尾花数据。

`data=iris_sns.iloc[:,0:-1]` 指定了要传递给 `sns.heatmap()` 的数据。`iris_sns` 是数据集, 方法 `iloc[:,0:-1]` 选择了所有行 (:) 和除最后一列之外的所有列 (0:-1), 即选择了数据集中的量化特征部分。

`vmin = 0, vmax = 8` 设置了颜色映射的范围, 即最小值和最大值。在这个例子中, 颜色映射的范围被限制在 0 到 8 之间。

`ax = ax` 将图形绘制在预先定义的轴对象上。

`yticklabels = False` 关闭了 y 轴上的刻度标签, 即不显示 y 轴上的数值。

`xticklabels = ['Sepal length', 'Sepal width', 'Petal length', 'Petal width']` 设置 x 轴上的刻度标签, 即显示特征的名称。

`cmap = 'RdYlBu_r'` 设置热图的颜色映射。

```


# 导入包
import matplotlib.pyplot as plt
import seaborn as sns

# 从seaborn中导入鸢尾花样本数据
a iris_sns = sns.load_dataset("iris")

# 绘制热图
fig, ax = plt.subplots()

b sns.heatmap(data=iris_sns.iloc[:,0:-1],
              vmin = 0, vmax = 8,
              ax = ax,
              yticklabels = False,
              xticklabels = ['Sepal length', 'Sepal width',
                             'Petal length', 'Petal width'],
              cmap = 'RdYlBu_r')

```

代码 5. 用 Seaborn 生成热图;  Bk1\_Ch11\_05.ipynb

代码 6 绘制图 9 (b)，下面聊聊其中关键词句。

- a 将 `plotly.express` 模块导入，简作 `px`。
- b 用 `plotly.express.iris()`，简作 `px.iris()`，从 `Plotly` 库中导入鸢尾花数据集。数据类型也是数据帧。不同于 `Seaborn` 中的鸢尾花数据集，`Plotly` 的数据集多了一列鸢尾花分类标签 (0、1、2)。
- c 用 `plotly.express.imshow()`，简作 `px.imshow()`，创建热图对象 `fig`。  
`df.iloc[:,0:-2]` 通过 `iloc` 切片选择了数据帧所有行和除了倒数第一、二列之外的所有列。  
 参数 `text_auto=False` 禁用了自动生成文本标签。  
`width = 600`, `height = 600` 设置图形的宽度和高度为 600 像素。  
`x = None` 设置横轴的值 `None`，意味着横轴上不显示具体的数值。  
`zmin=0`, `zmax=8` 设置颜色映射的范围，即最小值和最大值。在这个例子中，颜色映射的范围被限制在 0 到 8 之间。  
`color_continuous_scale = 'viridis'` 设置颜色映射，这里使用的是 `viridis` 色谱。
- d 用 `update_layout()` 方法对 `fig` 对象更新布局设置。这一句的目标是隐藏纵轴刻度标签。将字典赋值给参数 `yaxis`。字典中参数，`tickmode='array'` 指定了刻度标签的显示模式为数组。  
`tickvals=[]` 将刻度标签的值设为空列表，即在 `y` 轴上不显示任何刻度标签。
- e 用列表设置横轴标签。
- f 先用 `len` 计算列表长度，然后用 `range()` 生成可迭代对象，最后用 `list()` 将 `range` 转化为列表，结果为 `[0, 1, 2, 3]`。
- g 用 `update_xaxes()` 更新 `fig` 对象横轴设置。其中，`tickmode='array'` 指定了 `x` 轴刻度标签的显示模式为数组。  
`tickvals=x_ticks` 指定在 `x` 轴上显示的刻度值。  
`ticktext=x_ticks` 用于指定在 `x` 轴上显示的刻度标签的文本。

```

# 导入包
import matplotlib.pyplot as plt
a import plotly.express as px

# 从Plotly中导入鸢尾花样本数据
b df = px.data.iris()


# 创建Plotly热图
c fig = px.imshow(df.iloc[:,0:-2], text_auto=False,
                  width = 600, height = 600,
                  x = None, zmin=0, zmax=8,
                  color_continuous_scale = 'viridis')

# 隐藏 y 轴刻度标签
d fig.update_layout(yaxis=dict(tickmode='array', tickvals=[]))

# 修改 x 轴刻度标签
e x_labels = ['Sepal length', 'Sepal width',
              'Petal length', 'Petal width']
f x_ticks = list(range(len(x_labels)))
g fig.update_xaxes(tickmode='array', tickvals=x_ticks,
                  ticktext=x_labels)

fig.show()

```

代码 6. 用 Plotly 生成热图;  Bk1\_Ch11\_06.ipynb

## 11.5 三维可视化方案

本章介绍常见四种三维空间可视化方案。图 10 所示为三维直角坐标系和三个平面。

**散点图** (scatter plot) 用于展示三维数据的离散点分布情况。每个数据点在三维空间中的位置由其对应的三个数值确定。通过散点图，可以观察数据点的分布、聚集程度和可能的趋势。

**线图** (line plot) 可用于表示在三维空间中的曲线或路径。通过将连续的点用线段连接，可以呈现数据的演变过程或路径的形态。线图在表示运动轨迹、时间序列数据等方面很有用。

**网格面图** (mesh surface plot) 展示了三维空间中表面或曲面的形状。通过将空间划分为网格，然后根据每个网格点的数值给予相应的高度或颜色，可以可视化复杂的三维数据，例如地形地貌、物理场、函数表面等。

**三维等高线图** (3D contour plot) 在三维空间中绘制了等高线的曲线。这种图形通过将等高线与垂直于平面的轮廓线相结合，可以同时显示三个维度的信息。它适用于表示等值线密度、梯度分布等。



鸢尾花书《数学要素》第 6 章专门介绍三维直角坐标系。

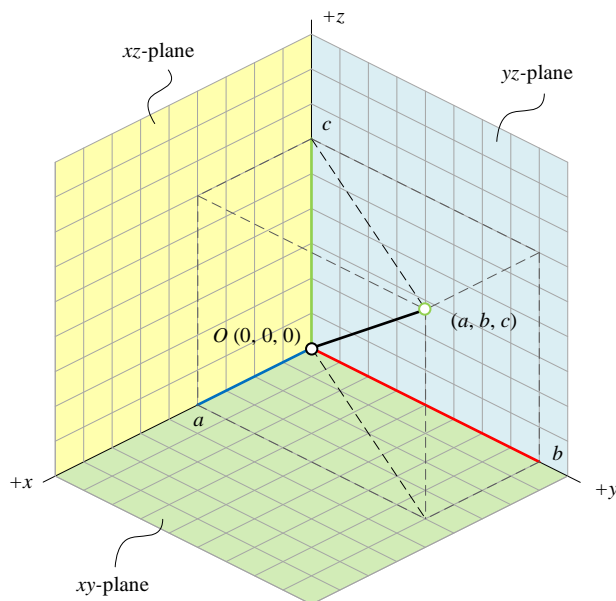


图 10. 三维直角坐标系和三个平面

### 三维视图视角

学过机械工程制图的同学知道，在三维空间中，我们可以将立体物体的投影投射到不同的平面上，以便更好地理解其形状和结构。

以下是常见的三维立体在不同面的投影方式：

- ▶ **俯视投影** (top view) 把立体物体在垂直于其底面的平面上投影的方式。这种投影显示了物体的顶部视图，可以揭示物体在水平方向上的外形和布局。
- ▶ **侧视投影** (side view) 将立体物体在垂直于其侧面的平面上投影的方式。这种投影显示了物体的侧面视图，可以展示物体在垂直方向上的外形和结构。
- ▶ **正视投影** (front view) 把立体物体在垂直于其正面的平面上投影的方式。这种投影显示了物体的正面视图，可以展示物体在前后方向上的外形和特征。
- ▶ **斜视投影** (isometric view) 将立体物体在等角度投射到平面上的方式。它显示了物体的斜面视图，保留了物体在三个维度上的比例关系，使观察者能够同时感知物体的长度、宽度和高度。

这些不同面的投影方式可以提供不同的视角，帮助我们 from 多个方面理解和分析立体物体。选择合适的投影方式取决于我们关注的特定方面和目的。特别是用 Matplotlib、Plotly 绘制三维图像时，选择合适的投影方式至关重要。

在 Matplotlib 中，`ax.view_init(elev, azim, roll)` 方法用于设置三维坐标轴的视角，也叫相机照相位置。这个方法接受三个参数：`elev`、`azim` 和 `roll`，它们分别表示仰角、方位角和滚动角。

- ▶ **仰角** (elevation): `elev` 参数定义了观察者与 `xy` 平面之间的夹角，也就是观察者与 `xy` 平面之间的旋转角度。当 `elev` 为正值时，观察者向上倾斜，负值则表示向下倾斜。

- **方位角 (azimuth):** `azim` 参数定义了观察者绕  $z$  轴旋转的角度。它决定了观察者在  $xy$  平面上的位置。`azim` 的角度范围是  $-180$  到  $180$  度，其中正值表示逆时针旋转，负值表示顺时针旋转。
- **滚动角 (roll):** `roll` 参数定义了绕观察者视线方向旋转的角度。它决定了观察者的头部倾斜程度。正值表示向右侧倾斜，负值表示向左侧倾斜。

通过调整这三个参数的值，可以改变三维图形的视角，从而获得不同的观察效果。例如，增加仰角可以改变观察者的俯视角度，增加方位角可以改变观察者在  $XY$  平面上的位置，增加滚动角可以改变观察者的头部倾斜程度。

类比的话，这三个角度和图 11 所示飞机的三个姿态角度类似。

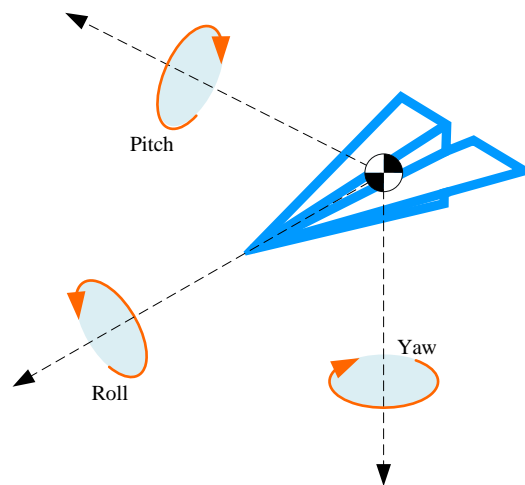


图 11. 飞机姿态的三个角度

如图 12 所示，鸢尾花书中调整三维视图视角一般只会用 `elev`、`azim`，几乎不使用 `roll`。

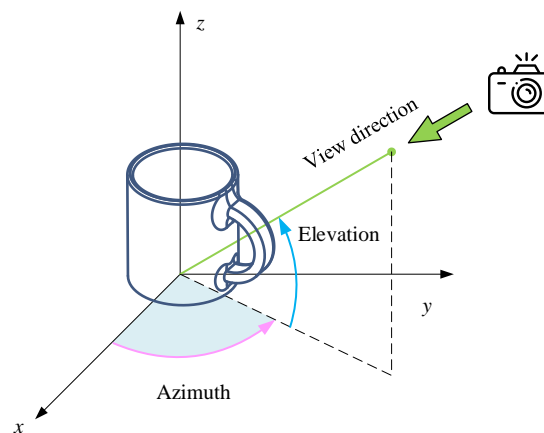


图 12. 仰角和方位角示意图

图 13 展示六个特殊视角，供大家参考。

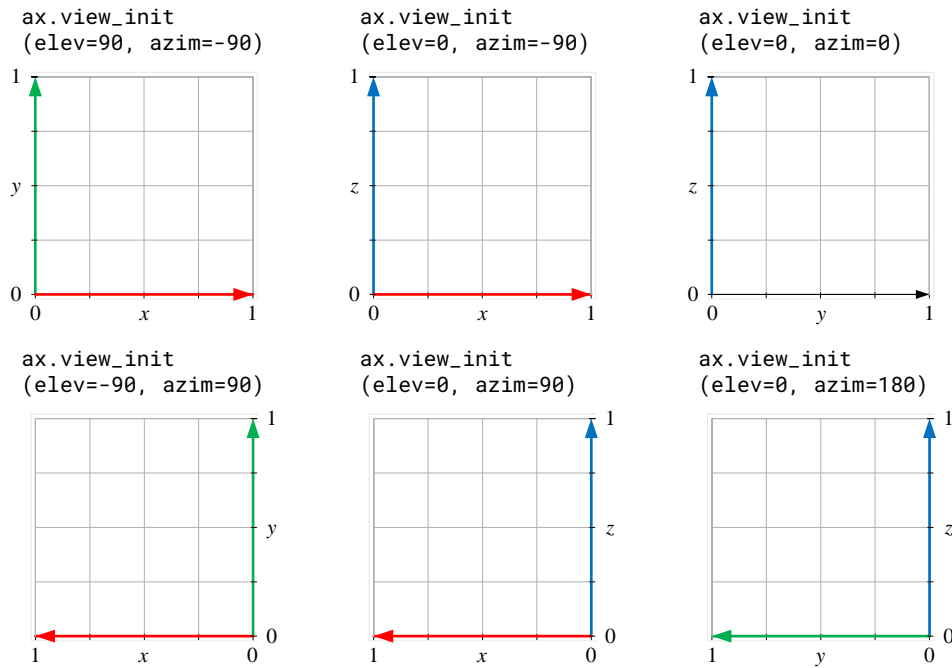


图 13. 几个特殊视角

下面，我们聊聊代码 7 中关键语句。

- a** 用 `matplotlib.pyplot.figure()`，简作 `plt.figure()`，创建图形对象 `fig`。
  - b** 在图形对象 `fig` 上，用 `add_subplot()` 方法通过设置 `projection = '3d'`，添加一个三维坐标轴对象。
  - c**、**d**、**e** 分别设置 `x`、`y`、`z` 轴标签。
  - f** 用 `set_proj_type()` 方法将投影类型设置为 `'ortho'`，即正交投影。我们马上就要了解不同的投影方法。
  - g** 通过 `view_init()` 设置三维轴对象的视角，两个关键字参数分别为 `elev = 30`（仰角 30 度），`azim = 30`（方位角 30 度）。
  - h** 用 `set_box_aspect()` 方法将三维直角坐标系的三个坐标轴比例设为一致。
- 请大家在 JupyterLab 中练习代码 7，并调整仰角、方位角大小观察图像变化。

**⚠ 注意**，`ax = fig.gca(projection='3d')` 已经被最新版本 Matplotlib 弃用，正确的语法为 `ax = fig.add_subplot(projection='3d')`。

```

import matplotlib.pyplot as plt
# 导入Matplotlib的绘图模块

a fig = plt.figure()
# 创建一个新的图形窗口

b ax = fig.add_subplot(projection='3d')
# 在图形窗口中添加一个3D坐标轴子图

c ax.set_xlabel('x')
d ax.set_ylabel('y')
e ax.set_zlabel('z')
# 设置坐标轴的标签


f ax.set_proj_type('ortho')
# 设置投影类型为正交投影 (orthographic projection)

g ax.view_init(elev=30, azim=30)
# 设置观察者的仰角为30度，方位角为30度，即改变三维图形的视角

h ax.set_box_aspect([1,1,1])
# 设置三个坐标轴的比例一致，使得图形在三个方向上等比例显示

plt.show()
# 显示图形

```

代码 7. 设置三维图像观察视角;  Bk1\_Ch11\_07.ipynb

有关 Matplotlib 三维视图视角，请参考：

[https://matplotlib.org/stable/api/toolkits/mplot3d/view\\_angles.html](https://matplotlib.org/stable/api/toolkits/mplot3d/view_angles.html)

## 两种投影方法

此外，大家还需要注意投影方法。上述代码采用的是正交投影。

在 Matplotlib 中，`ax.set_proj_type()` 方法用于设置三维坐标轴的投影类型。Matplotlib 提供了两种主要的投影类型：

- ▶ **透视投影** (perspective projection) 是默认的投影类型，如图 14 (a) 所示。简单来说就是近大远小，它模拟了人眼在观察远处物体时的视觉效果，使得远离观察者的物体显得较小。透视投影通过在观察者和图形之间创建一个虚拟的透视点，从而产生远近比例和景深感。设置方式为：`ax.set_proj_type('persp')`。
- ▶ **正交投影** (orthographic projection) 是另一种投影类型，如图 14 (b) 所示。它在观察者和图形之间维持固定的距离和角度，不考虑远近关系，保持了物体的形状和大小。正交投影在某些情况下可能更适合于一些几何图形的呈现，尤其是在需要准确测量物体尺寸或进行定量分析时。设置方式为：`ax.set_proj_type('ortho')`。

Plotly 的三维图像也是默认透视投影，想要改成正交投影对应的语法为：

```
fig.layout.scene.camera.projection.type = "orthographic"
```

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



图 15 展示了 3D 绘图时改变焦距对透视投影的影响。需要注意的是，Matplotlib 会校正焦距变化所带来的“缩放”效果。

透视投影中，默认焦距为 1，对应 90 度的**视场角** (Field of View, FOV)。增加焦距（1 至无穷大）会使图像变得扁平，而减小焦距（1 至 0 之间）则会夸张透视效果，增加图像的视觉深度。当焦距趋近无穷大时，经过缩放校正后，会得到正交投影效果。

⚠ 注意，鸢尾花书中三维图像绝大部分都是正交投影。

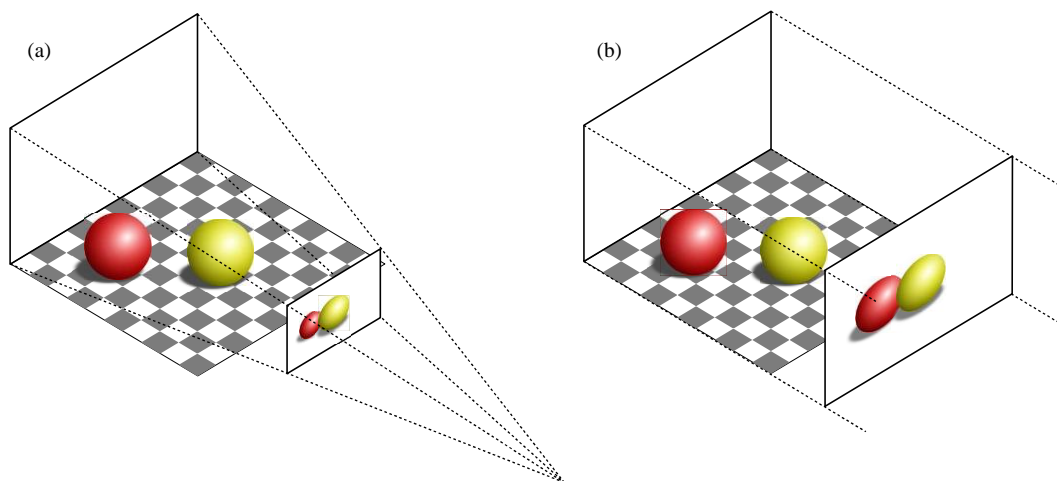


图 14. 透视投影和正交投影；来源：<https://github.com/rougier/scientific-visualization-book>

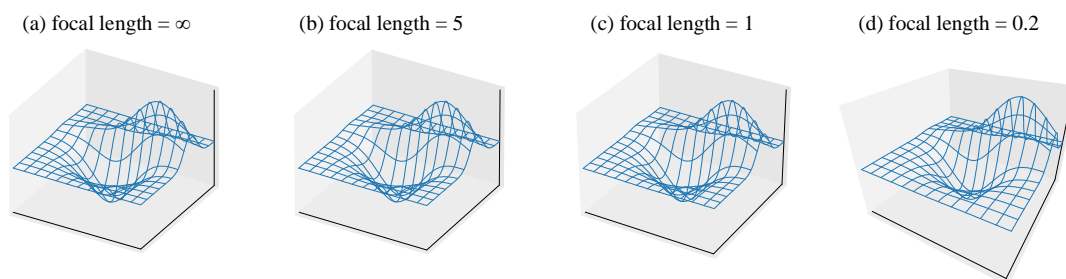


图 15. 投影焦距对结果影响；参考：<https://matplotlib.org/stable/gallery/mplot3d/projections.html>

## 11.6 三维散点

上一章我们利用平面散点可视化鸢尾花数据集，这一节将用三维散点图可视化这个数据集。图 16 所示为利用 Matplotlib 绘制的三维散点图，这幅图用不同颜色表征鸢尾花分类。类似图 13，几个特殊视角，请大家将图 16 投影到不同平面上。

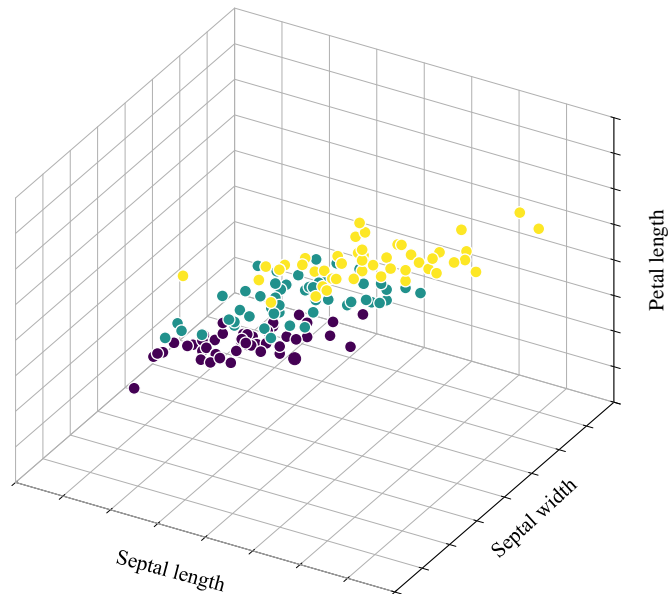


图 16. 用 Matplotlib 绘制三维散点图

代码 8 绘制图 16 三维散点图，下面聊聊其中关键语句。

- a** 取出 NumPy Array 前三列。第一个冒号 `:` 代表所有行，`:3` 代表索引为 0、1、2 的连续三列，分别代表鸢尾花花萼长度、花萼宽度、花瓣长度三个特征样本数据。
- b** 提取鸢尾花分类数据，数据类型也是 NumPy Array。
- c** 用 `matplotlib.pyplot.figure()`，简作 `plt.figure()`，创建图形对象 `fig`。
- d** 用 `add_subplot()` 方法在 `fig` 上增加一个三维轴对象。其中，111 分别代表 1 行 1 列编号为 1 的子图，`projection = '3d'` 设定投影为三维。
- e** 在三维轴对象上用 `scatter()` 绘制三维散点图。其中 `X[:,0]` 代表二维数组索引为 0 列，即第 1 列，以此类推，`c` 代表 `color`，设定值为 `y`，即用颜色渲染不同鸢尾花分类。
- f** 用 `axis()` 分割三个语句，用来设定 x、y、z 轴取值范围。
- g** 用 `set_proj_type('ortho')` 设定三维轴对象为正交投影。

代码 9 用 Plotly 绘制交互三维散点图，请大家自行分析并逐行注释。

```

# 导入包
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets

# 加载鸢尾花数据集
iris = datasets.load_iris()
# 取出前三个特征作为横纵坐标和高度
a X = iris.data[:, :3]
b y = iris.target

# 创建3D图像对象
c fig = plt.figure()
d ax = fig.add_subplot(111, projection='3d')

# 绘制散点图
e ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y)

# 设置坐标轴标签
ax.set_xlabel('Sepal length')
ax.set_ylabel('Sepal width')
ax.set_zlabel('Petal length')
# 设置坐标轴取值范围
f ax.set_xlim(4,8); ax.set_ylim(1,5); ax.set_zlim(0,8)
# 设置正交投影
g ax.set_proj_type('ortho')
# 显示图像
plt.show()

```



代码 8. 用 Matplotlib 和绘制三维散点图; Bk1\_Ch11\_08.ipynb

```

import plotly.express as px
# 导入鸢尾花数据
a df = px.data.iris()
b fig = px.scatter_3d(df,
                    x='sepal_length',
                    y='sepal_width',
                    z='petal_length',
                    size='petal_width',
                    color='species')

c fig.update_layout(autosize=False,width=500,height=500)
d fig.layout.scene.camera.projection.type = "orthographic"
fig.show()

```

代码 9. 用 Plotly 绘制三维散点图; Bk1\_Ch11\_09.ipynb

## 11.7 三维线图

图 17 所示为利用 Matplotlib 绘制“线图 + 散点图”可视化微粒的随机漫步。并且用散点的颜色渐进变化展示时间维度。



《数据有道》将专门介绍随机漫步。

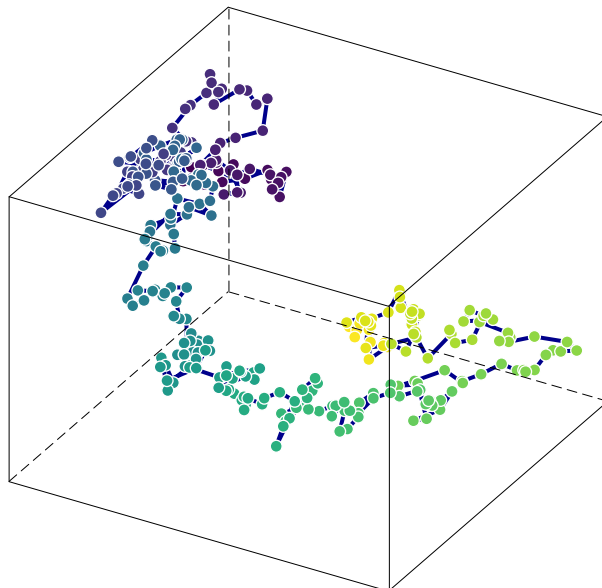


图 17. 用 Matplotlib 绘制微粒随机漫步线图



### 什么是随机漫步？

随机漫步是指一个粒子或者一个系统在一组离散的时间步骤中，按照随机的方向和大小移动的过程。每个时间步骤，粒子以随机的概率向前或向后移动一个固定的步长，而且每个时间步骤之间的移动是相互独立的。随机漫步模型常用于模拟不确定性和随机性的系统，例如金融市场、扩散过程、分子运动等。通过模拟大量的随机漫步路径，可以研究粒子或系统的统计特性和概率分布。

代码 10 绘制图 17，代码中也用 Plotly 绘制三维散点，请大家在 JupyterLab 中查看可视化结果。

代码 10 中得大家注意的是 <sup>b</sup>，其中用到了几个新函数。

其中，`numpy.random.standard_normal()` 用来产生服从**标准正态分布**（standard normal distribution）的随机数。这些随机数代表每步行走的步长。

`numpy.cumsum()` 用来计算累加，代表微粒随机行走轨迹。请大家自行分析代码 10 中剩余语句，并逐行注释。

```

# 导入包
import matplotlib.pyplot as plt
import numpy as np
a import plotly.graph_objects as go

# 生成随机游走数据
num_steps = 300
b t = np.arange(num_steps)
x = np.cumsum(np.random.standard_normal(num_steps))
y = np.cumsum(np.random.standard_normal(num_steps))
z = np.cumsum(np.random.standard_normal(num_steps))

# 用 Matplotlib 可视化
fig = plt.figure()
c ax = fig.add_subplot(111, projection='3d')


d ax.plot(x,y,z,color = 'darkblue')
e ax.scatter(x,y,z,c = t, cmap = 'viridis')

ax.set_xticks([]); ax.set_yticks([]); ax.set_zticks([])
# 设置正交投影
ax.set_proj_type('ortho')
# 设置相机视角
ax.view_init(elev = 30, azim = 120)
# 显示图像
plt.show()

# 用 Plotly 可视化
f fig = go.Figure(data=go.Scatter3d(
    x=x, y=y, z=z,
    marker=dict(size=4,color=t,colorscale='Viridis'),
    line=dict(color='darkblue', width=2)))

fig.layout.scene.camera.projection.type = "orthographic"
fig.update_layout(width=800,height=700)
fig.show() # 显示绘图结果

```

代码 10. 用 Matplotlib 和 Plotly 可视化三维随机行走;  Bk1\_Ch11\_10.ipynb

## 11.8 三维网格面

图 18 所示为利用 Axes3D.plot\_surface() 绘制的三维网格曲面。



请大家思考如何在图 18 中加入 colorbar。

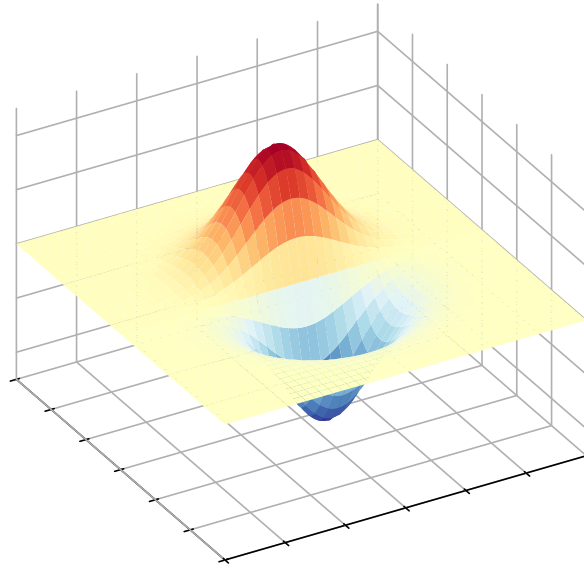


图 18. 用 Matplotlib 绘制三维网格曲面

代码 11 绘制图 18，这段代码还用 Plotly 绘制的三维曲面，请大家在 JupyterLab 中查看可视化结果。下面聊聊代码 11 中关键语句。

**a** 导入 Plotly 库中的 `graph_objects` 模块，简作 `go`。

**b** 用 `fig.add_subplot()` 在图形对象 `fig` 中添加了一个三维子图轴对象 `Axes3D`。

参数 `111` 表示将图形分成 1 行 1 列的子图，而数字 `1` 表示当前子图在这个网格中的位置。

参数 `projection='3d'` 指定子图的投影方式，这里是三维投影。这是用 Matplotlib 库绘制三维可视化方案的前提。

**c** 利用 `plot_surface()` 方法在三维轴对象 `ax` 上绘制三维网格曲面

`xx1`, `xx2`, `ff` 这三个参数是数据。`xx1` 和 `xx2` 是坐标网格，分别为横纵坐标，`ff` 是这个网格上的函数值。

参数 `cmap='RdYlBu_r'` 指定等高线颜色映射。在这里，使用了 `'RdYlBu_r'`，表示红、黄、蓝渐变色的颜色映射，`'_r'` 代表翻转。

**d** 用 `plotly.graph_objects.Figure()` 创建 Plotly 图形对象。

`plotly.graph_objects.Surface()`，简作 `go.Surface()`，用于创建三维曲面图的对象。

`z=ff`, `x=xx1`, `y=xx2` 这三个参数分别是 `z`、`x`、`y` 轴的数据，用于表示三维空间中的曲面。

参数 `colorscale='RdYlBu_r'` 指定曲面颜色映射的参数。

鸢尾花书经常用 `plot_wireframe()` 绘制网格曲面，请大家自行学习下例。

<https://matplotlib.org/stable/gallery/mplot3d/wire3d.html>

```

# 导入包
import matplotlib.pyplot as plt
import numpy as np
a import plotly.graph_objects as go

# 生成曲面数据
x1_array = np.linspace(-3,3,121)
x2_array = np.linspace(-3,3,121)

xx1, xx2 = np.meshgrid(x1_array, x2_array)
ff = xx1 * np.exp(- xx1**2 - xx2 **2)


# 用 Matplotlib 可视化三维曲面
fig = plt.figure()
b ax = fig.add_subplot(111, projection='3d')

c ax.plot_surface(xx1, xx2, ff, cmap='RdYlBu_r')

# 设置坐标轴标签
ax.set_xlabel('x1'); ax.set_ylabel('x2');
ax.set_zlabel('f(x1,x2)')
# 设置坐标轴取值范围
ax.set_xlim(-3,3); ax.set_ylim(-3,3); ax.set_zlim(-0.5,0.5)
# 设置正交投影
ax.set_proj_type('ortho')
# 设置相机视角
ax.view_init(elev = 30, azim = 150)
plt.tight_layout()
plt.show()

# 用 Plotly 可视化三维曲面
d fig = go.Figure(data=[go.Surface(z=ff, x=xx1, y=xx2,
                                colorscale='RdYlBu_r')])
fig.layout.scene.camera.projection.type = "orthographic"
fig.update_layout(width=800,height=700)
fig.show()

```

代码 11. 用 Matplotlib 和 Plotly 可视化三维网格面;  Bk1\_Ch11\_11.ipynb

## 11.9 三维等高线

图 19 所示为用 Matplotlib 绘制的三维等高线，这些等高线投影到水平面便得到上一章介绍的平面等高线。



鸢尾花书《可视之美》将介绍更多三维等高线的用法。



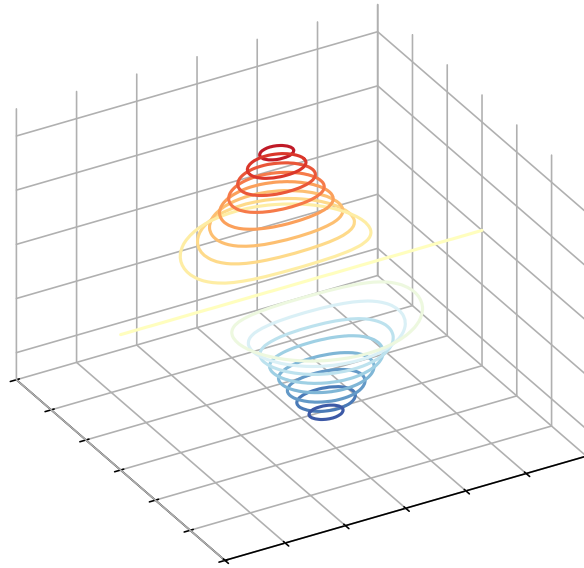


图 19. 用 Matplotlib 绘制三维等高线

代码 12 绘制图 19，这段代码也用 Plotly 绘制的三维“曲面 + 等高线”，请大家在 JupyterLab 中查看可视化结果。下面聊聊其中关键语句。

- a 导入 Plotly 库中的 `graph_objects` 模块，简作 `go`。
- b 在三维轴对象 `ax` 上用 `contour()` 添加三维等高线。大部分参数已经在本章前文提过。参数 `levels=20`：指定等高线的数量。
- c 创建 `contour_settings`，这是一个包含等高线设置的字典。在这里，设置了 `z` 轴的等高线参数，包括是否显示等高线 (`"show": True`)，开始值 (`"start": -0.5`)，结束值 (`"end": 0.5`)，和轮廓线之间的间隔 (`"size": 0.05`)。
- d 用 `plotly.graph_objects.Figure()`，简作 `go.Figure()`，创建 Plotly 图形对象 `fig`。  
`fig` 是一个 Plotly 图形对象，用于容纳图形的各种元素。  
`plotly.graph_objects.Surface()`，简作 `go.Surface()`，是 Plotly 中用于创建三维表面图的对象。  
`x=xx1`，`y=xx2`，`z=ff` 这三个参数分别是 `x`、`y`、`z` 轴的数据，用于表示三维空间中的表面。  
`colorscale='RdYlBu_r'` 指定表面颜色映射的参数。  
`contours=contour_settings` 指定等高线的参数，使用了之前定义的字典对象。

```

# 导入包
import matplotlib.pyplot as plt
import numpy as np
a import plotly.graph_objects as go


# 生成曲面数据
x1_array = np.linspace(-3,3,121)
x2_array = np.linspace(-3,3,121)
xx1, xx2 = np.meshgrid(x1_array, x2_array)
ff = xx1 * np.exp(- xx1**2 - xx2 **2)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
b ax.contour(xx1, xx2, ff, cmap='RdYlBu_r', levels = 20)
# 设置坐标轴标签
ax.set_xlabel('x1'); ax.set_ylabel('x2'); ax.set_zlabel('f(x1,x2)')
# 设置坐标轴取值范围
ax.set_xlim(-3,3); ax.set_ylim(-3,3); ax.set_zlim(-0.5,0.5)
# 设置正交投影
ax.set_proj_type('ortho')
# 设置相机视角
ax.view_init(elev = 30, azim = 150)
plt.tight_layout()
plt.show()

c contour_settings = {"z": {"show":True, "start":-0.5,
                           "end":0.5, "size": 0.05}}
d fig = go.Figure(data=[go.Surface(x=xx1,y=xx2,z=ff,
                                colorscale='RdYlBu_r',
                                contours = contour_settings)])

fig.layout.scene.camera.projection.type = "orthographic"
fig.update_layout(width=800, height=700)
fig.show() # 显示绘图结果

```

代码 12. 用 Matplotlib 和 Plotly 可视化三维等高线;  Bk1\_Ch11\_11.ipynb

## 11.10 箭头图

我们可以用 `matplotlib.pyplot.quiver()` 绘制**箭头图** (quiver plot 或 vector plot)。实际上, 我们在本书第 5 章用箭头图可视化二维向量、三维向量, 不知道大家是否有印象。

本节尝试利用 `matplotlib.pyplot.quiver()` 复刻第 5 章中看到的箭头图, 具体如图 20 所示。

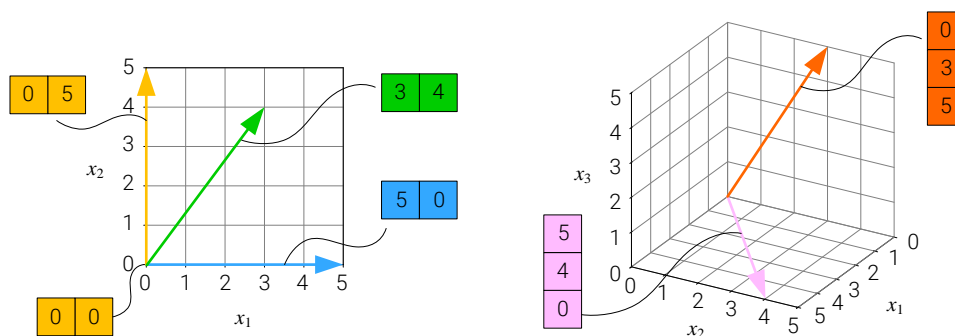


图 20. 二维、三维箭头图

代码 13 绘制图 20 平面箭头图。

- a 用 Python 列表创建了一个矩阵。这个矩阵有 3 行、2 列，形状为  $3 \times 2$ 。
- b 自定义函数，用来绘制二维平面箭头。这个函数有两个输入，向量和颜色 RGB 色号。
- c 用 `matplotlib.pyplot.quiver()`，简作 `plt.quiver()`，绘制平面箭头图。

参数 `0` 和 `0` 是箭头的起点坐标，这表示箭头起点在原点  $(0, 0)$  处。

参数 `vector[0]` 为箭头在  $x$  轴上的分量，`vector[1]` 是箭头在  $y$  轴上的分量。

参数 `angles='xy'` 指定箭头应该以  $x$  和  $y$  轴的角度来表示。

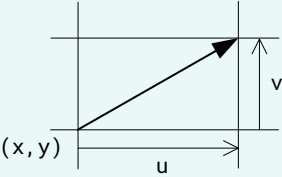
参数 `scale_units='xy'` 指定箭头的比例应该根据  $x$  和  $y$  轴的单位来缩放。

参数 `scale=1` 指定箭头的长度应该乘以的比例因子。在这里，箭头的长度将乘以 1，保持原始长度。

参数 `color=RGB` 指定箭头的颜色。RGB 可以是一个包含红、绿、蓝值的元组，也可以是字符串色号，或者是十六进制色号。

参数 `zorder` 用来指定“艺术家”图层序号。

- d 取出数组索引为 `0` 的元素，即矩阵的第 1 个行向量。
- e 调用自定义函数绘制平面箭头，颜色采用十六进制色号。



```

# 导入包
import matplotlib.pyplot as plt

# 定义二维列表
A = [[0,5],
      [3,4],
      [5,0]]

# 自定义可视化函数
def draw_vector(vector, RGB):
    plt.quiver(0, 0, vector[0], vector[1], angles='xy',
               scale_units='xy', scale=1, color = RGB,
               zorder = 1e5)

fig, ax = plt.subplots()
v1 = A[0] # 第一行向量
draw_vector(v1, '#FFC000')
v2 = A[1] # 第二行向量
draw_vector(v2, '#00CC00')
v3 = A[2] # 第三行向量
draw_vector(v3, '#33A8FF')

ax.axvline(x = 0, c = 'k')
ax.axhline(y = 0, c = 'k')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.grid()
ax.set_aspect('equal', adjustable='box')
ax.set_xbound(lower = -0.5, upper = 5)
ax.set_ybound(lower = -0.5, upper = 5)

```

代码 13. 绘制二维箭头图; Bk1\_Ch11\_12.ipynb

代码 14 绘制图 20 三维箭头图。

- a 自定义函数绘制三维箭头图。
- b 和前文类似，也是用 `matplotlib.pyplot.quiver()` 绘制箭头图，不同的是这段代码绘制三维箭头图。参数 `0, 0, 0` 是箭头的起点坐标，即三维空间中的原点  $(0, 0, 0)$  处。参数 `vector[0]`、`vector[1]`、`vector[2]` 是箭头在  $x$ 、 $y$ 、 $z$  轴上的分量。参数 `arrow_length_ratio=0` 指定箭头的长度应该乘以的比例因子。在这里，`0` 表示箭头的长度将被设置为一个合适的长度，不乘以比例因子。
- c 创建了一个新的 Matplotlib 图形对象 `fig`，并指定了它的大小为  $(6, 6)$ ，即宽度和高度都为 6 英寸。
- d 用 `fig.add_subplot()` 在图形对象 `fig` 中添加了一个三维子图轴对象 `Axes3D`。本章前文已经介绍其中大部分参数，请大家回顾。值得一提的是，参数 `proj_type='ortho'` 指定了三维投影的类型为正交投影。
- e 利用列表生成式提取矩阵第一列，即第一个列向量。
- f 调用自定义函数绘制三维箭头图。


```

# 自定义可视化函数
a def draw_vector_3D(vector, RGB):
    b plt.quiver(0, 0, 0, vector[0], vector[1], vector[2],
                arrow_length_ratio=0, color = RGB,
                zorder = 1e5)

c fig = plt.figure(figsize = (6,6))
d ax = fig.add_subplot(111, projection='3d', proj_type = 'ortho')
    # 第一列向量
e v_1 = [row[0] for row in A]
f draw_vector_3D(v_1, '#FF6600')
    # 第二列向量
v_2 = [row[1] for row in A]
draw_vector_3D(v_2, '#FFBBFF')

ax.set_xlim(0,5)
ax.set_ylim(0,5)
ax.set_zlim(0,5)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
ax.view_init(azim = 30, elev = 25)
ax.set_box_aspect([1,1,1])

```

代码 14. 绘制三维箭头图，使用时配合前文代码；  Bk1\_Ch11\_12.ipynb



鸢尾花书《可视之美》将专门介绍更多箭头图的可视化方案。



请大家完成下面 3 道题目。

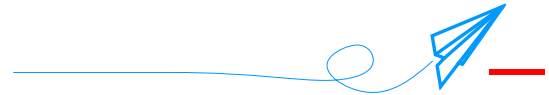
Q1. 分别用 Matplotlib、Seaborn、Plotly 绘制鸢尾花数据集，花瓣长度、宽度散点图，并适当美化图像。

Q2. 分别用 Matplotlib 和 Plotly 绘制如下二元函数等高线图，并用语言描述图像特点（等高线形状、疏密分布、增减、最大值、最小值等等）。

$$\begin{aligned}
 f(x_1, x_2) &= x_1 \\
 f(x_1, x_2) &= x_2 \\
 f(x_1, x_2) &= x_1 + x_2 \\
 f(x_1, x_2) &= x_1 - x_2 \\
 f(x_1, x_2) &= x_1^2 + x_2^2 \\
 f(x_1, x_2) &= -x_1^2 - x_2^2 \\
 f(x_1, x_2) &= x_1^2 + x_2^2 + x_1 x_2 \\
 f(x_1, x_2) &= x_1^2 - x_2^2 \\
 f(x_1, x_2) &= x_1^2 \\
 f(x_1, x_2) &= x_1 x_2 \\
 f(x_1, x_2) &= x_1^2 + x_2^2 + 2x_1 x_2
 \end{aligned}$$

Q3. 分别用 Matplotlib 和 Plotly 中网格面、三维等高线可视化以上几个二元函数。

\* 本章不提供答案。



本章介绍了几种常用的二维平面和三维空间的可视化方案。实现这些可视化方案时，我们混用 Matplotlib 和 Plotly。Matplotlib 特别擅长绘制静态图，这对于平面出版很适用；而 Plotly 绘制的图像具有交互属性，很适合用来现场演示。