19 Sequences
数列
用几何可视化手段展示数列和极限

我把心和灵魂投入到绘画中，并在这个过程中失去了理智。

I put my heart and my soul into my work, and have lost my mind in the process.

—— 文森特·梵高 (Vincent van Gogh) | 荷兰后印象派画家 | 1853—1890年

- `matplotlib.patches.Polygon` 多边形对象
- `matplotlib.pyplot.Rectangle()` 矩形对象
- `matplotlib.transforms.Affine2D` 完成对象的二维仿射变换，如平移、缩放、旋转等几何操作
- `numpy.cumsum()` 计算累计求和
- `numpy.insert()` 在数组特定位置插入数值

19.1 什么是数列？

数列是按照一定规律排列的数字集合。常见类型包括**等差数列** (arithmetic progression或arithmetic sequence)、**等比数列** (geometric progression或geometric sequence)、**斐波那契数列** (Fibonacci sequence) 等。

本质上，数列也是一种特殊函数。在机器学习中，数列常用于表示数据序列，如时间序列预测等。

一般情况下，我们可以通过折线图、火柴图、热图展示数列趋势和模式。而本章将用几何视角展示数列和数列求和极限。

19.2 斐波那契数列

《编程不难》介绍过斐波那契数列。我们可以在自然界中找到很多斐波那契数列的例子，如植物的分枝结构、蜂窝的排列方式等。图19.1所示为斐波那契数列的一种可视化方案。

Bk2_Ch19_01.ipynb中绘制了图19.1，下面讲解代码19.1中关键语句。

ⓐ用matplotlib.pyplot.cm，简写作plt.cm，生成一组渐变色。plt.cm.Blues_r根据np.linspace(0, 1, num + 1) 数值生成一组蓝色渐变色。

ⓑ为正方形的旋转角度。

ⓒ为正方形的边长。

ⓓ为正方形“锚点”的坐标。不考虑旋转的话，锚点为矩形左下角顶点的位置坐标。

ⓔ用plt.Rectangle() 矩形对象创建“正方形”实例。

rec_loc_idx 是矩形左下角的位置坐标。

width=side_idx 和 height=side_idx 分别设置了矩形的宽度和高度。

facecolor=colors[i + 1] 设置了矩形的填充颜色。

edgecolor='k' 设置了矩形的边框颜色为黑色。

transform 参数用于指定矩形的变换方式，这里使用了仿射变换 (affine transformation)。

Affine2D().rotate_deg_around(x_array[i], y_array[i], rotate_i) 是一个仿射变换，表示绕 (x_array[i], y_array[i]) 点旋转 rotate_i 度。

ax.transData 表示将变换应用于数据坐标系。

ⓕ在轴对象ax添加图形对象。

代码19.1　可视化斐波那契数列 | BK2_Ch19_01.ipynb

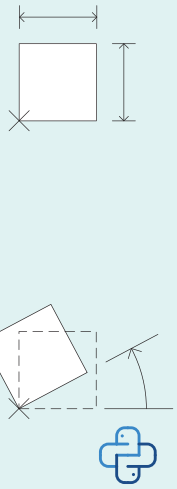
```
fig, ax=plt.subplots(figsize=(8,8))

colors=plt.cm.Blues_r(np.linspace(0,1,num+1))

for i in idx_array:
    rotate_i=rotation[i]
    side_idx=fibonacci_array_from_1[i]
    rec_loc_idx=np.array([x_array[i], y_array[i]])

    rec = plt.Rectangle(rec_loc_idx,
                        width=side_idx,
                        height=side_idx,
                        facecolor=colors[i + 1],
                        edgecolor='k',
                        transform=Affine2D().rotate_deg_around(
                           x_array[i], y_array[i], rotate_i)
                        + ax.transData)
    ax.add_patch(rec)

ax.set_aspect('equal')
ax.plot(0, 0,  color='r', marker='x', markersize=10)
ax.axis('off')
plt.show()
```



Bk2_Ch19_02.ipynb中绘制了图19.2。图19.2在图19.1基础上增加了螺旋线，请大家自行分析。

19.3 巴都万数列

图19.3可视化了**巴都万数列** (Padovan Sequence)。巴都万数列前15项为1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37, 49。

大家可以发现，这个数列的前3项均为1，之后的数值递推关系定义为$P(n) = P(n-2) + P(n-3)$。

$n > 6$时，巴都万数列也可以写成$P(n) = P(n-1) + P(n-5)$。这是本例可视化用的公式。

Bk2_Ch19_03.ipynb中绘制了图19.3。

这个代码中，作图技巧是采用matplotlib.patches.Polygon() 绘制正三角形。代码用到的重要数学工具是平面旋转。请大家自行分析这段代码。

19.4 雷卡曼数列

在数学和计算机科学中，**雷卡曼数列** (Recamán sequence) 通常使用递归的方式来定义。雷卡曼数列前20项为0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8, 25, 43, 62。

图19.4用圆弧方式展示了雷卡曼数列的前60列。Bk2_Ch19_04.ipynb中绘制了图19.4，并给出了通项公式。

19.5 数列求和极限

图19.5和图19.6则用几何方式展现了数列求和极限。当项数逐渐增加，且数列求和无限接近某个值时，这个值被称为数列求和极限。

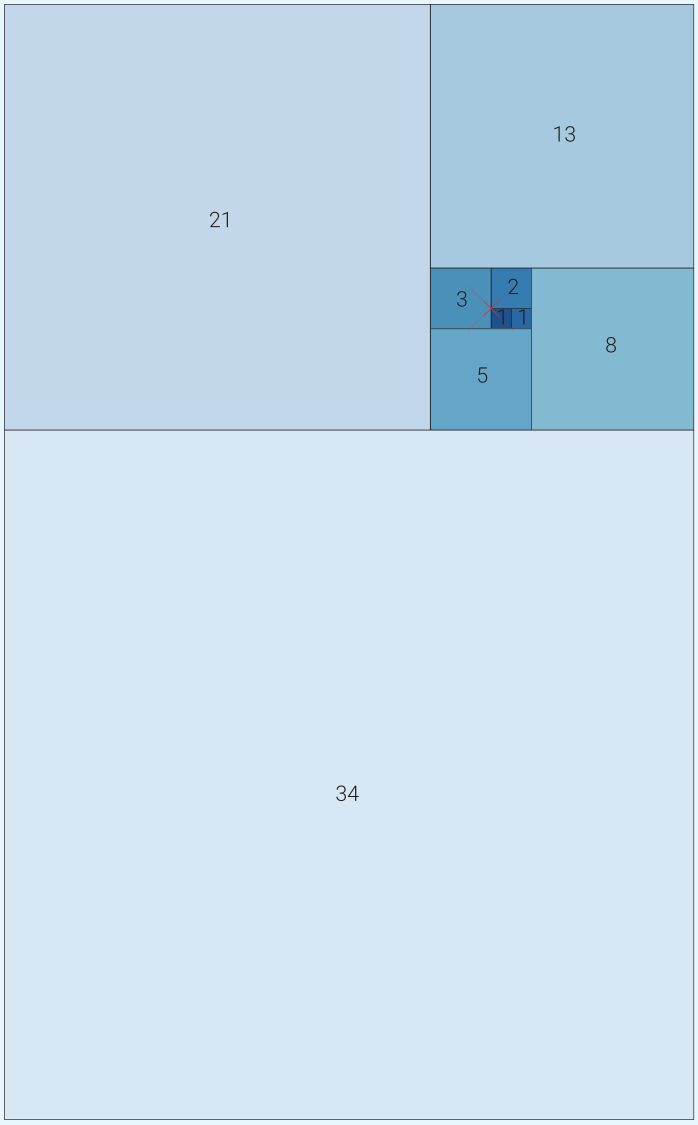
如图19.5所示，数列求和1/4 + 1/16 + 1/64 + 1/256 + … 趋向于1/3。

Bk2_Ch19_05.ipynb中绘制了图19.5，请大家自行分析代码。

如图19.6所示，数列求和1/2 + 1/4 + 1/8 + 1/16 + … 趋向于1。

Bk2_Ch19_06.ipynb中绘制了图19.6，请大家自行分析代码。

本章讲解了如何可视化斐波那契数列、巴都万数列、雷卡曼数列，以及数列求和极限。《数学要素》将介绍数列、极限背后的数学工具，并由其引出微积分相关知识点。

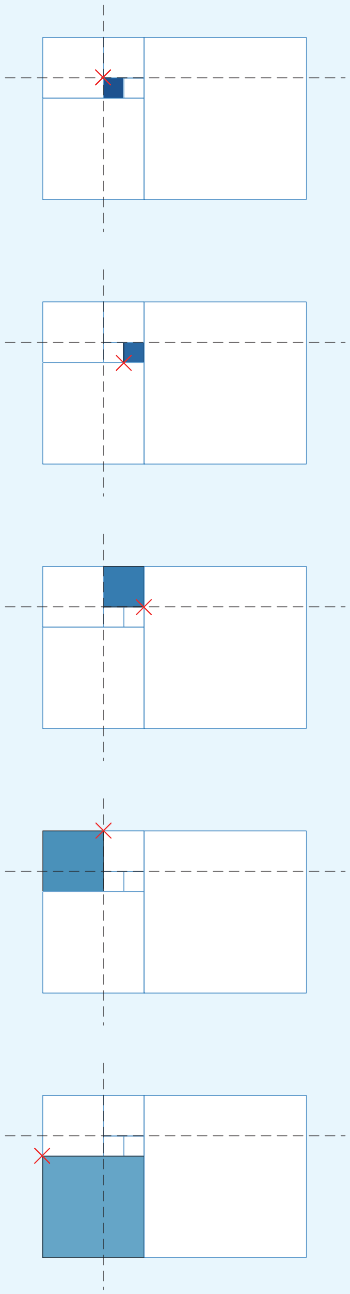


斐波那契数列(Fibonacci sequence)以递归的方式定义。它的前两个数字是0和1，从第三个数字开始，每个数字都是前两个数字的和。

斐波那契数列前几项为

0, 1, 1, 2, 3, 5, 8, 13, 21, 34。

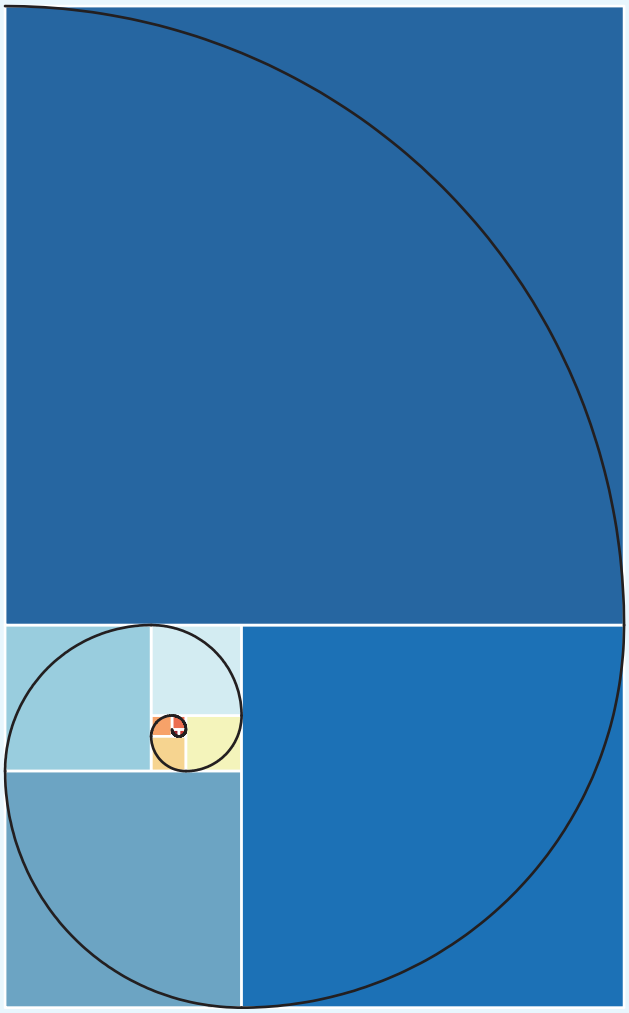
左图的正方形的边长展示的便是(0以外) 斐波那契数列。



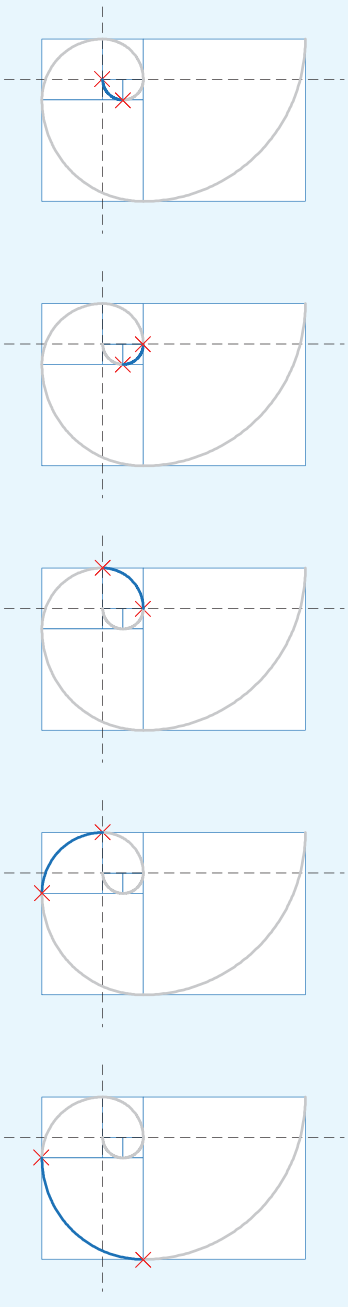
为了绘制这组正方形，需要确定四个量：①图中“×”的横坐标；②图中“×”的纵坐标；③正方形边长；④旋转角度。请大家根据以下表格数值寻找规律。配套的Jupyter Notebook中给出了具体的计算代码。

n	1	2	3	4	5	6	7
横坐标	0	1	2	0	-3	2	10
纵坐标	0	-1	0	2	-1	-6	2
边长	1	1	2	3	5	8	13
旋转角	-90	0	90	180	270	360	450

图19.1　可视化斐波那契数列 | BK2_Ch19_01.ipynb



斐波那契螺旋线基于斐波那契数列。它的构造方式基于上一页的正方形，用1/4圆弧(90°) 连接图中“x”，最终形成一个螺旋线。斐波那契螺旋线可以用来近似黄金螺旋(golden spiral)。

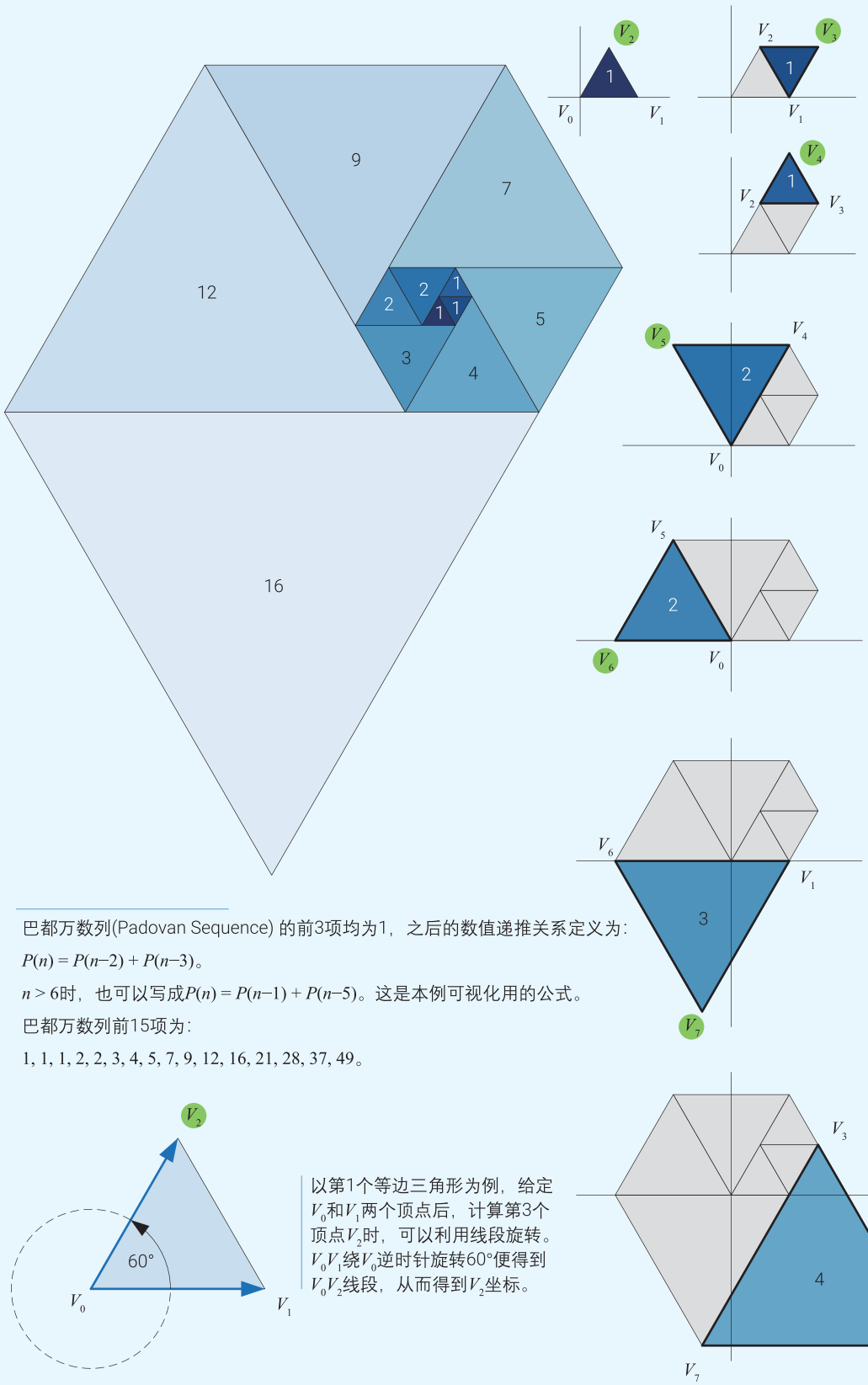


为了绘制斐波那契螺旋线，需要确定四个量：①圆心横坐标；② 圆心纵坐标；③ 半径；④ 1/4圆弧起始角。请大家根据以下表格数值寻找规律。配套的Jupyter Notebook中给出了具体的计算代码。

n	1	2	3	4	5	6	7
横坐标	1	1	0	0	2	2	-3
纵坐标	0	0	0	-1	-1	2	2
半径	1	1	2	3	5	8	13
起始角	2 × 90	3 × 90	4 × 90	5 ×90	6 × 90	7 × 90	8 ×90

代码中还用到了以下三角波。

图19.2　可视化斐波那契螺旋线 | BK2_Ch19_02.ipynb



巴都万数列(Padovan Sequence) 的前3项均为1，之后的数值递推关系定义为：

$P(n) = P(n-2) + P(n-3)$。

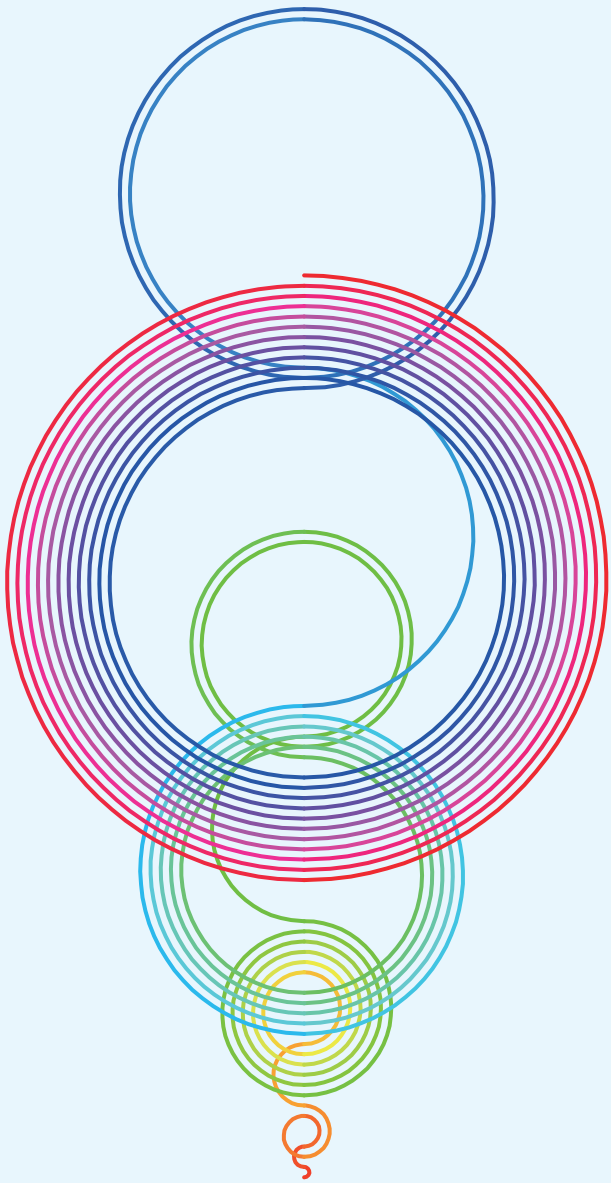
$n > 6$时，也可以写成$P(n) = P(n-1) + P(n-5)$。这是本例可视化用的公式。

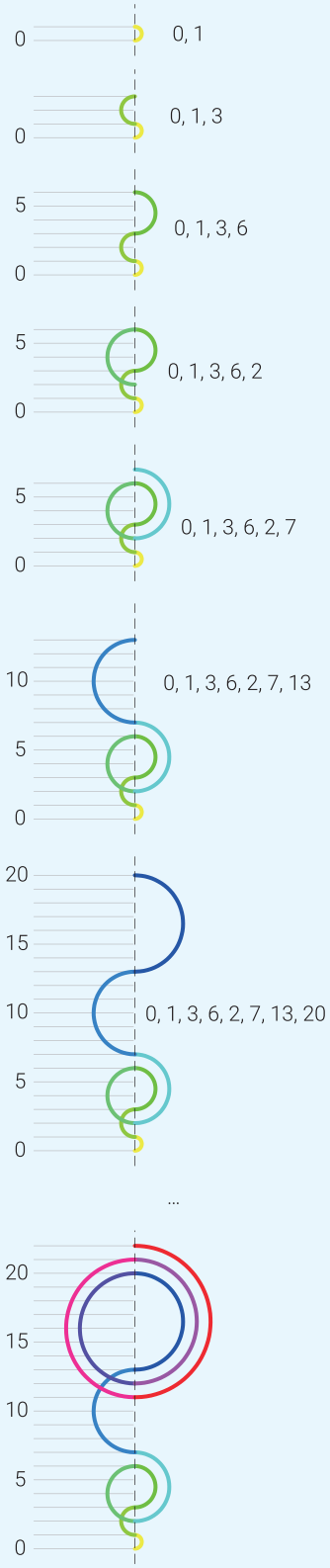
巴都万数列前15项为：

1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, 28, 37, 49。

以第1个等边三角形为例，给定V_0和V_1两个顶点后，计算第3个顶点V_2时，可以利用线段旋转。V_0V_1绕V_0逆时针旋转60°便得到V_0V_2线段，从而得到V_2坐标。

图19.3　可视化巴都万数列 | BK2_Ch19_03.ipynb





在数学和计算机科学中，雷卡曼(Recamán)数列通常使用递归的方式来定义。雷卡曼数列前20项为：

0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8, 25, 43, 62。

上图可视化雷卡曼数列前60项。下图展示雷卡曼数列前1000项的变化趋势。配套的Jupyter Notebook中给出了通项公式。

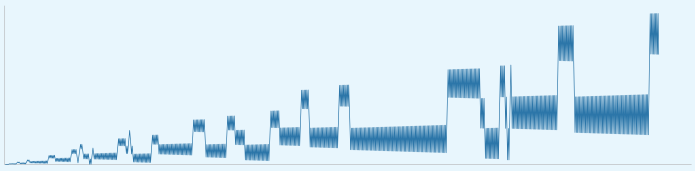
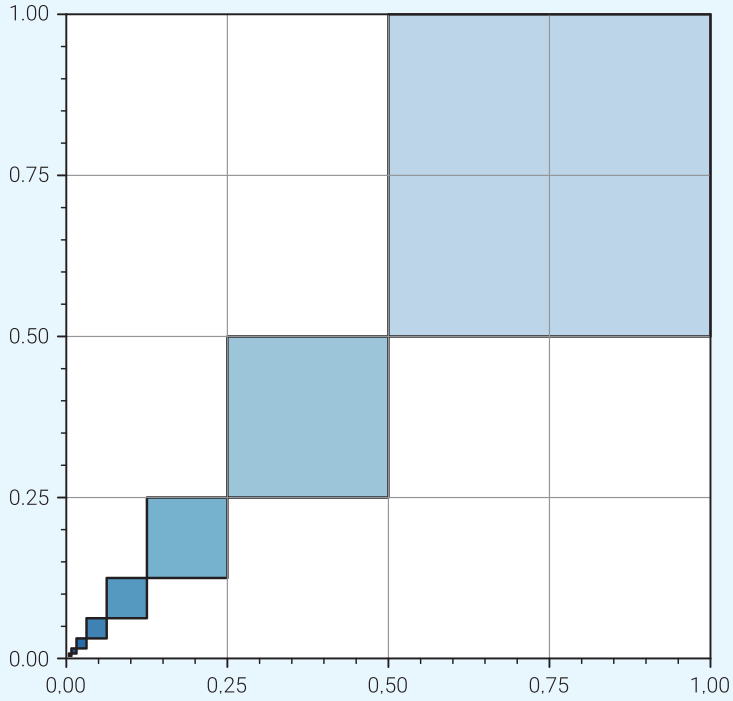
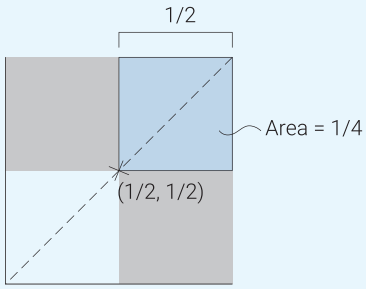
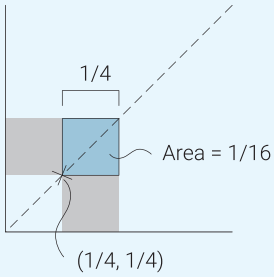
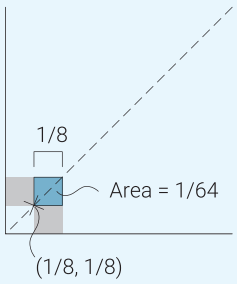


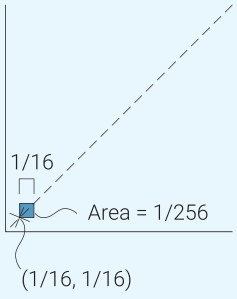
图19.4　可视化雷卡曼数列 | BK2_Ch19_04.ipynb











当项数逐渐增加，且数列求和无限接近某个值时，这个值被称为数列求和极限。比如，

$1/4 + 1/16 + 1/64 + 1/256 + \cdots \rightarrow 1/3$

也就是说

$2^{-2} + 2^{-4} + 2^{-8} + 2^{-16} + \cdots \rightarrow 1/3$

图解方法来看，上图的蓝色正方形的面积之和为1/3。

为了绘制上图中的正方形，需要确定三个量：①横坐标；②纵坐标；③边长。请大家根据以下表格数值寻找规律。配套的Jupyter Notebook中给出了具体的计算代码。

n	1	2	3	4	5	6	7
横坐标	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
纵坐标	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
边长	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}

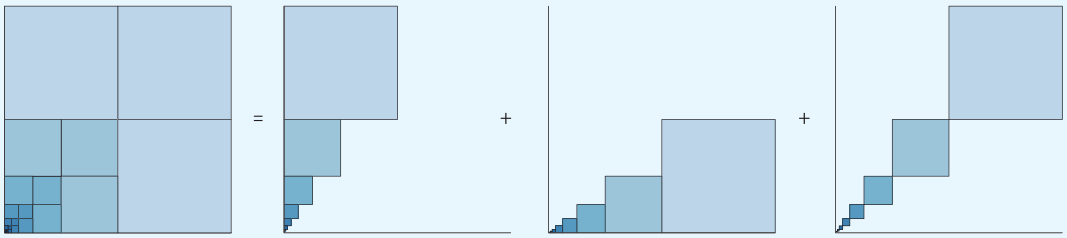
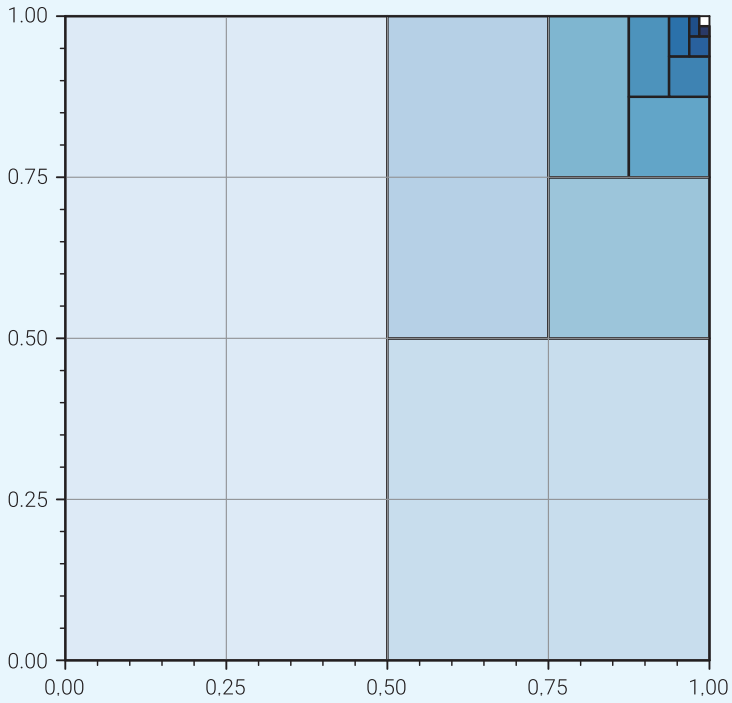
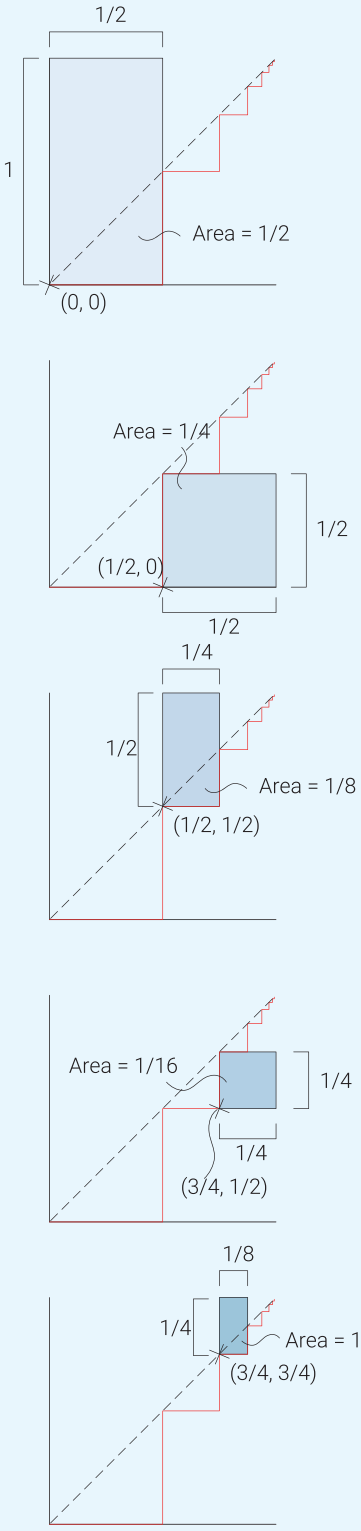


图19.5　可视化数列之和极限，第1组 | BK2_Ch19_05.ipynb





以下数列和的极限为

$1/2 + 1/4 + 1/8 + 1/16 + \cdots \to 1$

也就是说

$2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + \cdots \to 1$

图解方法来看，上图的蓝色正方形的面积之和为1。

反向来看，就是日取其半，万世不竭。

为了绘制上图矩形，需要确定四个量：①横坐标；②纵坐标；③矩形宽度；④矩形高度。请大家根据以下表格数值寻找规律。配套的Jupyter Notebook中给出了具体的计算代码。

n	1	2	3	4	5	6	7
横坐标	0	1/2	1/2	3/4	3/4	7/8	7/8
纵坐标	0	0	1/2	1/2	3/4	3/4	7/8
宽度	1/2	1/2	1/4	1/4	1/8	1/8	1/16
高度	1	1/2	1/2	1/4	1/4	1/8	1/8

图19.6　可视化数列之和极限，第2组 | BK2_Ch19_06.ipynb

20 Function 函数

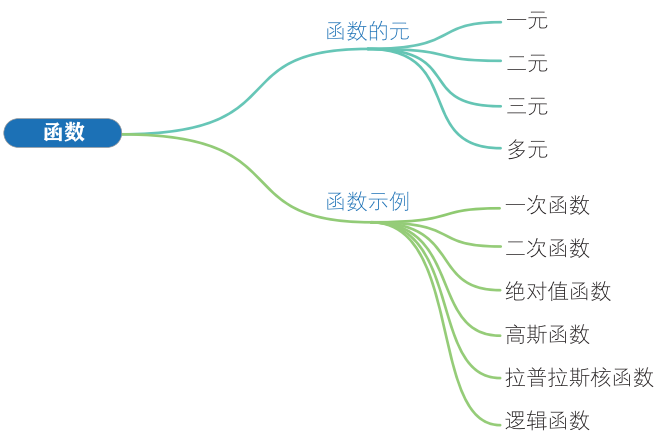
可视化一元、二元、三元函数

至暗深夜也必然结束，太阳终将升起。

Even the darkest night will end and the sun will rise.

—— 维克多 • 雨果 (Victor Hugo) | 法国文学家 | 1802—1885年

- `matplotlib.pyplot.contour()` 绘制等高线图
- `matplotlib.pyplot.contourf()` 绘制填充等高线图
- `matplotlib.pyplot.plot_wireframe()` 绘制线框图
- `numpy.linspace()` 在指定的间隔内，返回固定步长的数据
- `numpy.meshgrid()` 创建网格化数据



20.1 函数

在数学中，函数的**元** (arity) 指的是函数接受的参数个数。常见的函数元数包括以下几类。

- **一元函数** (unary function) 接受一个参数。例如，$f_1(x) = x$ 是一个一元函数，它接受一个参数 x。
- **二元函数** (binary function) 接受两个参数。例如，$f_2(x_1, x_2) = x_1 + x_2$ 是一个二元函数，它接受两个参数 x_1 和 x_2。
- **三元函数** (ternary function) 接受三个参数。例如，$f_3(x_1, x_2, x_3) = x_1 + x_2 + x_3$是一个三元函数，它接受三个参数 x_1、x_2 和 x_3。
- **多元函数** (n-ary function) 接受 n 个参数。多元函数的参数个数可以是任意多个，例如$f_n(x_1, x_2, \cdots, x_n) = x_1 + x_2 + \cdots + x_n$是一个多元函数，它接受任意$n$个参数 x_1、x_2、...、x_n。

《编程不难》第8章简述函数概念，请大家回顾。

一元、二元、三元、多元函数的映射如图20.1所示。

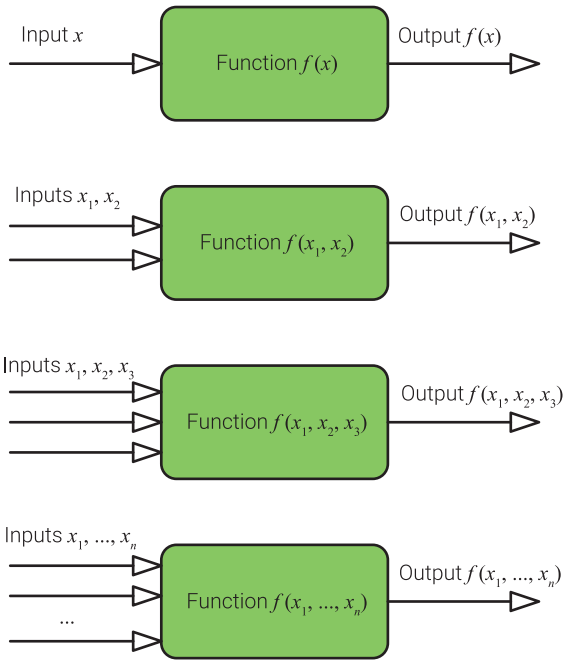


图20.1　一元、二元、三元、多元函数的映射

下面我们就来聊聊如何利用本书前文介绍的各种可视化方案展示一元、二元、三元函数。

20.2 一次函数

一元**一次函数** (linear function) 是一种形式为$f(x) = ax + b$的函数，a和b是实数常数，且a不为零。它是最简单的线性函数，描述了一个直线的斜率和截距。图20.3、图20.4所示为一元、二元、三元一次函数的几种可视化方案。

请大家思考如何可视化四元函数，比如$f(x_1, x_2, x_3, x_4) = x_1 + x_2 + x_3 + x_4$。

一次函数在数学和实际应用中非常有用。在统计学中，它被广泛用于线性回归分析，其中通过拟合一条直线来描述自变量与因变量之间的关系。这可以帮助我们预测和解释数据的变化趋势。

《数学要素》专门介绍一次函数。

此外，一次函数还在分类决策面中扮演重要角色。在机器学习和模式识别中，一次函数可以用来划分特征空间，将数据点分为不同的类别。

Bk2_Ch20_01.ipynb中绘制了图20.3中一元一次函数$f_1(x) = x$。代码很简单，下面讲解其中可视化语句 (见代码20.1)。

ⓐ在轴对象ax上用plot()绘制一元函数线图，x1_array为横坐标，f1_array为纵坐标。

ⓑ将坐标轴的纵横比设置为相等。

ⓒ隐藏四个图脊(spines)。

代码20.1　可视化一元一次函数 | Bk2_Ch20_01.ipynb

```
fig, ax=plt.subplots()

ax.plot(x1_array, f1_array)

ax.set_xlabel('x'); ax.set_ylabel('f(x)')
ax.set_xticks(ticks_array); ax.set_yticks(ticks_array);
ax.set_aspect('equal', adjustable='box')
ax.set_xlim(x1_array.min(),x1_array.max());
ax.set_ylim(x1_array.min(),x1_array.max());
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.grid(linestyle='--', linewidth=0.25, color=[0.5,0.5,0.5])
plt.show()
```

Bk2_Ch20_02.ipynb中绘制了图20.3中二元一次函数。代码20.2绘制网格面，下面讲解其中关键语句。

ⓐ在图形对象fig上用add_subplot(projection = '3d')方法增加一个三维轴对象。

ⓑ在三维轴对象ax上用plot_wireframe()绘制网格面，展示二元函数$f(x_1, x_2) = x_1 + x_2$。

rstride指定网格数据行之间的跨度，表示在行的方向上将线绘制多少行之后跳过一行。

cstride指定网格数据列之间的跨度，表示在列的方向上将线绘制多少列之后跳过一列。

请大家尝试将rstride、cstride设置为其他正整数，并观察效果。

linewidth指定网格面线宽，即线的粗细。

ⓒ利用set_proj_type()设置三维图形的投影方式为正投影。本书前文介绍过正投影，简单来说，正投影不考虑“近大远小”这种视觉效果。

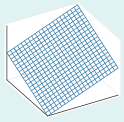
ⓓ利用view_init()设定视角。参数elev为仰角，参数azim为方位角。

代码20.2　可视化二元一次函数，网格面 | Bk2_Ch20_02.ipynb

```
fig=plt.figure(figsize=(5,5))
ax=fig.add_subplot(projection='3d')

ax.plot_wireframe(xx1, xx2, f2_array,
                  rstride=1, cstride=1,
                  linewidth=1)

ax.set_xlabel('$x_1$'); ax.set_ylabel('$x_2$')
ax.set_zlabel('$f(x_1,x_2)$'); ax.grid(False)
ax.set_proj_type('ortho')
ax.set_xticks(ticks_array); ax.set_yticks(ticks_array)
ax.view_init(azim=-120, elev=30)
ax.set_xlim(xx1.min(), xx1.max()); ax.set_ylim(xx2.min(), xx2.max())
plt.tight_layout()
```



代码20.2绘制的网格面，相当于用“经纬线”织布。而代码20.3则只绘制单一维度织线。我们在本书第15章介绍过这种可视化方案，请大家回顾。

ⓐ利用plot_wireframe()时，设置cstride=0，不显示数据column列方向织线。

ⓑ利用plot_wireframe()时，设置rstride=0，不显示数据row行方向织线。

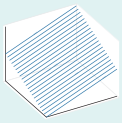
代码20.3　可视化二元一次函数，网格面单一维度织线 | Bk2_Ch20_02.ipynb

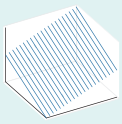
```
fig=plt.figure(figsize=(5,5))
ax=fig.add_subplot(projection='3d')

# 只绘制row方向织线
ax.plot_wireframe(xx1, xx2, f2_array,
                  rstride=1, cstride=0)

fig=plt.figure(figsize=(5,5))
ax=fig.add_subplot(projection='3d')

# 只绘制column方向织线
ax.plot_wireframe(xx1, xx2, f2_array,
                  rstride=0, cstride=1)
```





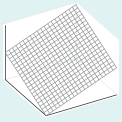
代码20.4采用“网格面 + 三维等高线”方式可视化二元函数。请大家自行分析代码。

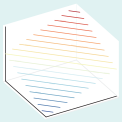
代码20.4　可视化二元一次函数，网格面 + 三维等高线 | Bk2_Ch20_02.ipynb

```
fig=plt.figure(figsize=(5,5))
ax=fig.add_subplot(projection='3d')
# 绘制网格面
ax.plot_wireframe(xx1, xx2, f2_array,
                  rstride=1, cstride=1,
                  color=[0.5,0.5,0.5],
                  linewidth=0.25)
# 绘制等高线
ax.contour(xx1, xx2, f2_array,
           levels=20,
           cmap='RdYlBu_r')
```

a

b





代码20.5分别采用二维等高线、二维填充等高线可视化二元函数，请大家自行分析。Bk2_Ch20_02.ipynb还给出其他几种可视化方案，请大家自行分析。

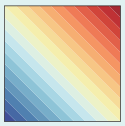
代码20.5　可视化二元一次函数，二维等高线，二维填充等高线 | Bk2_Ch20_02.ipynb

```
fig=plt.figure(figsize=(5,5))
ax=fig.add_subplot()
# 二维等高线
ax.contour(xx1, xx2, f2_array,
           levels=20, cmap='RdYlBu_r')

fig=plt.figure(figsize=(5,5))
ax=fig.add_subplot()
# 二维填充等高线
ax.contourf(xx1, xx2, f2_array,
           levels=20, cmap='RdYlBu_r')
```

a

b



Bk2_Ch20_03.ipynb主要用“切豆腐”方法可视化三元函数，我们已经在本书前文介绍过相关代码，请大家自行分析。

如图20.2所示，利用Plotly的Volume图 (体积图) 可视化三元函数。在JupyterLab中，大家可以旋转、缩放这幅图。

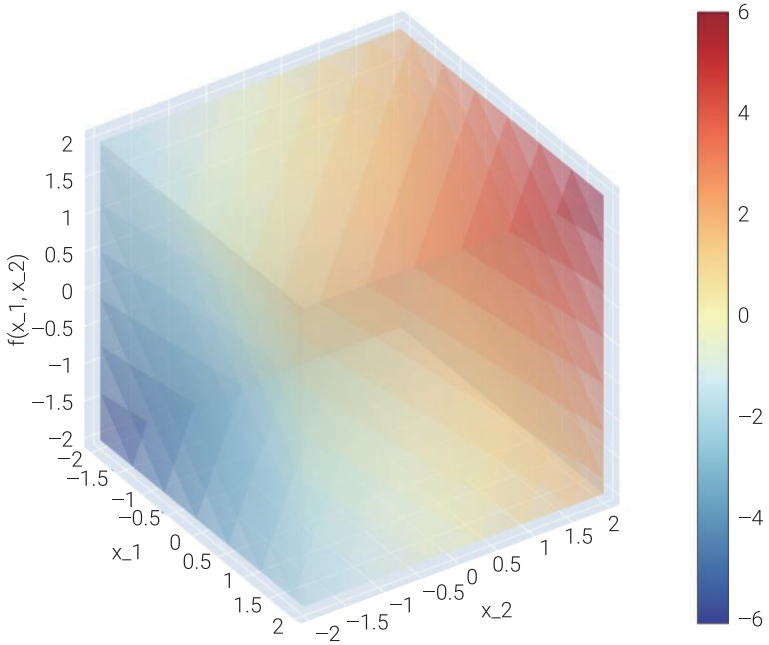


图20.2　用Plotly的Volume图可视化三元一次函数 | Bk2_Ch20_03.ipynb

下面，我们讲解绘制图20.2的核心代码 (见代码20.6)。

ⓐ用plotly.graph_objects.Volume()，简写作go.Volume()，绘制体积图。

x=xxx1.flatten()表示 x 轴上的坐标数据。其中，xxx1.flatten() 是将 xxx1 中的三维数组展平成一维数组。

y=xxx2.flatten()表示 y 轴上的坐标数据。

z=xxx3.flatten()表示 z 轴上的坐标数据。

value=f3_array.flatten()表示体积图上每个点的三元函数数值。

isomin=f3_array.min()指定体积图的数据范围的下限，对应三元函数最小值。

isomax=f3_array.max()指定体积图的数据范围的上限，对应三元函数最大值。

opacity=0.25设置体积图的透明度，0.25表示相对较透明。

colorscale='RdYlBu_r'指定体积图的颜色映射。

surface_count=20表示体积图中显示的曲面数量。

ⓑ用plotly.graph_objects.Layout()，简写作go.Layout()，设置图形属性。

width=800设定图形的宽度为800像素。

height=800设定图形的高度为800像素。

scene=dict()设置图形中场景，包含了 x 轴、y 轴、z 轴的标题等信息。

xaxis_title='x_1'设置 x 轴的标题。

yaxis_title='x_2'设置 y 轴的标题。

zaxis_title='f(x_1,x_2)'设置 z 轴的标题。

ⓒ用plotly.graph_objects.Figure()，简写作go.Figure()，创建图形对象。

data=data指定图形的数据，即之前创建的体积图数据。

layout=layout指定图形的布局。

ⓓ设置投影方式为正投影。

ⓔ和ⓕ设置视角位置。

代码20.6　用Plotly的volume | ♔Bk2_Ch20_03.ipynb

```
import plotly.graph_objects as go
data=go.Volume(                                    ⓐ
    x=xxx1.flatten(),
    y=xxx2.flatten(),
    z=xxx3.flatten(),
    value=f3_array.flatten(),
    isomin=f3_array.min(),
    isomax=f3_array.max(),
    opacity=0.25,
    colorscale='RdYlBu_r',
    surface_count=20)

# 布局设置
layout=go.Layout(                                  ⓑ
    width=800, height=800,
    scene=dict(
        xaxis_title='x_1',
        yaxis_title='x_2',
        zaxis_title='f(x_1,x_2)'))
```

```
# 创建图形对象
c fig=go.Figure(data=data, layout=layout)
d fig.layout.scene.camera.projection.type = "orthographic"
e camera=dict(eye=dict(x=0.6, y=-1, z=0.5))
f fig.update_layout(scene_camera=camera)
  fig.show()
```

20.3 其他几个函数示例

请大家自行修改Bk2_Ch20_01.ipynb、Bk2_Ch20_02.ipynb、Bk2_Ch20_03.ipynb，自行绘制以下几个函数的可视化方案。

二次函数

一元**二次函数** (quadratic function) 是一种形式为$f(x) = ax^2 + bx + c$的函数，其中a、b、c是实数常数，且a不为零。图20.5、图20.6所示为一元、二元、三元二次函数的几种可视化方案。

二次函数在数学和实际应用中有广泛的应用。在数学领域，二次函数是研究代数和解析几何的重要对象，它们具有丰富的性质和特征。

在实际应用中，二次函数可以用于建模和预测各种现象。例如，在物理学中，二次函数可以描述自由落体运动的高度随时间变化的关系。此外，二次函数还用于优化问题和曲线拟合。

例如，在最优化领域，通过分析二次函数的性质，可以找到最小值或最大值的位置；在曲线拟合中，可以利用二次函数来逼近实际数据，并进行预测和插值。

《数学要素》专门介绍二次函数。

绝对值函数

绝对值函数 (absolute value function) 是一种形式为$f(x) = a|x - b|$的函数，a和b是实数常数。图20.7、图20.8所示为一元、二元、三元绝对值函数的几种可视化方案。

绝对值函数在数学和实际应用中都具有重要的作用。在数学中，绝对值函数是一种特殊的分段函数，具有简单的图像和性质。它在不等式和绝对值相关的问题中经常出现。

绝对值函数可以用于测量误差或距离的绝对值，使得结果不受正负号影响。在优化问题中，绝对值函数常用于表示目标函数或约束条件，帮助找到最优解。此外，绝对值函数还在统计学和数据分析中发挥作用。它可以用于计算绝对误差或残差，评估模型的拟合程度。

《数学要素》专门介绍不同绝对值函数。

高斯函数

图20.9、图20.10所示为一元、二元、三元**高斯函数** (Gaussian function) 的几种可视化方案。

高斯函数常用于高斯分布，因此高斯函数在统计学和概率论中广泛应用。高斯函数在数据分析和模型拟合中非常有用。它可以用来描述数据集的分布情况，并进行概率推断和统计推断。

在机器学习中，高斯函数常用于聚类算法、异常检测和生成模型。此外，高斯函数还在信号处理中广泛应用，如图像处理和滤波。它可以用来平滑信号、降噪和边缘检测。

《统计至简》专门介绍高斯分布。

拉普拉斯核函数

拉普拉斯核函数 (Laplacian kernel function) 是一种常用的非线性核函数，用于**支持向量机** (Support Vector Machine，SVM) 和其他机器学习算法中的非线性分类和回归任务。图20.11、图20.12所示为一元、二元、三元拉普拉斯核函数的几种可视化方案。

拉普拉斯核函数可以将数据映射到高维特征空间，从而在低维空间中实现非线性分类。与其他核函数相比，拉普拉斯核函数在决策边界附近具有更陡峭的变化，因此对异常点更敏感，使其能够更好地捕捉数据中的局部结构。

《机器学习》专门介绍支持向量机。

拉普拉斯核函数在图像识别、文本分类、生物信息学等领域广泛应用。它具有良好的分类性能，并且在处理高维数据和处理小样本问题时表现出色。

逻辑函数

逻辑函数 (logistic function) 是一种常用的非线性函数，也称为Sigmoid函数或逻辑曲线。逻辑函数的取值范围在0 ~ 1之间，具有平滑的S形曲线。图20.13、图20.14所示为一元、二元、三元逻辑函数的几种可视化方案。

逻辑函数在机器学习、神经网络和统计学中广泛应用。它主要用于将输入值映射到概率值，将连续变量转换为概率分布。在二元分类任务中，逻辑函数可以将输入的线性组合转化为0 ~ 1之间的概率值，用于判断样本属于某一类别的概率。

逻辑函数在逻辑回归模型中扮演重要角色，用于建立分类模型和进行概率预测。它能够对数据进行非线性建模，拟合复杂的分类决策边界。

《数据有道》专门介绍逻辑回归。

此外，逻辑函数还用于激活函数，如在神经网络中，逻辑函数可用于将神经元的输出转化为非线性响应，提高模型的表达能力。

函数是代数、机器学习中极为重要的数学概念。通常，我们看到最多的是一元函数的图像；而本章利用各种可视化方案，向大家展示了二元、三元函数图像。

下一章，我们还会用类似本章的可视化方案展示二次型，二次型是在优化问题、机器学习算法中重要的数学概念。

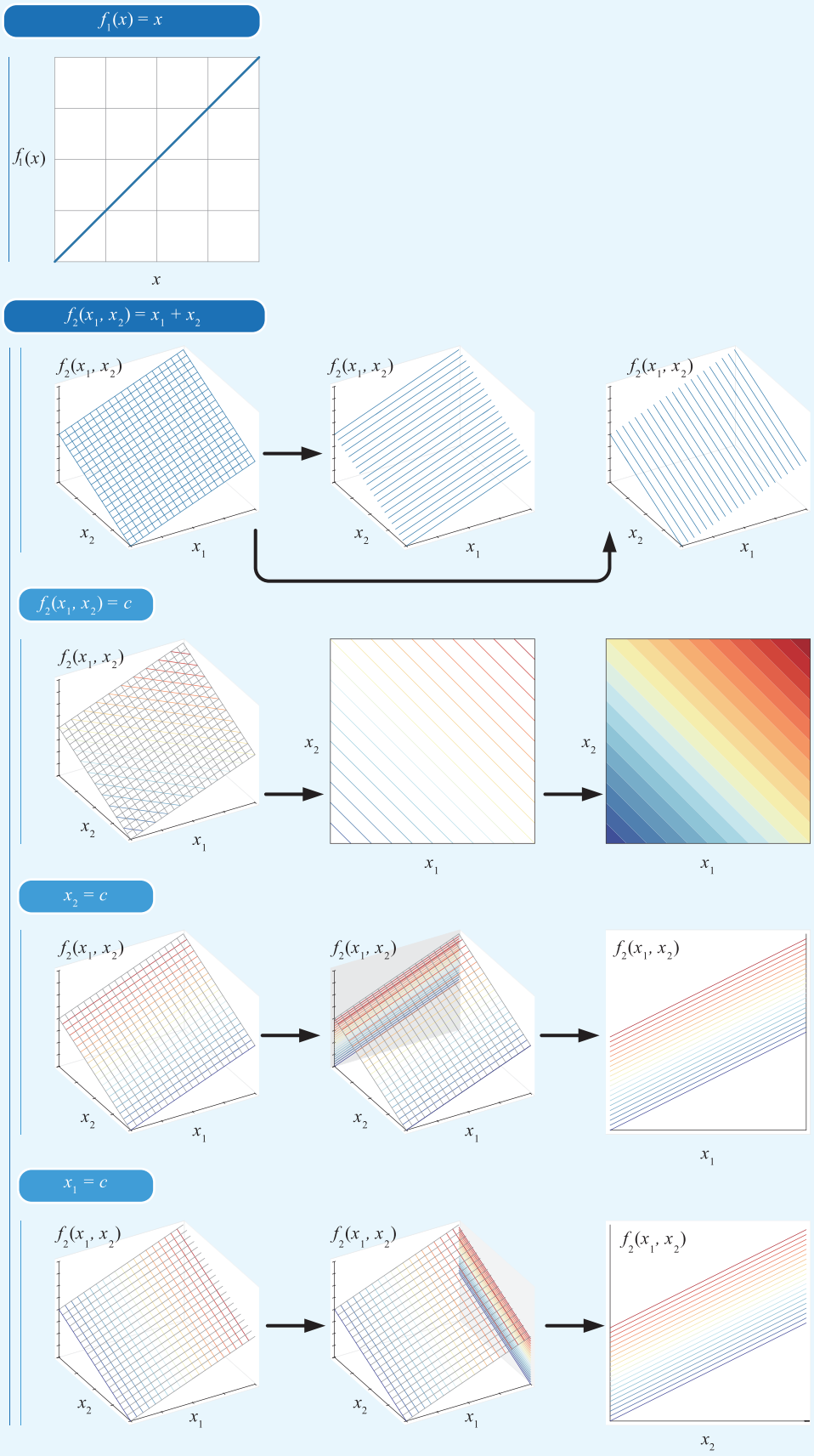


图20.3　一次函数，一元、二元 | Bk2_Ch20_01.ipynb | Bk2_Ch20_03.ipynb

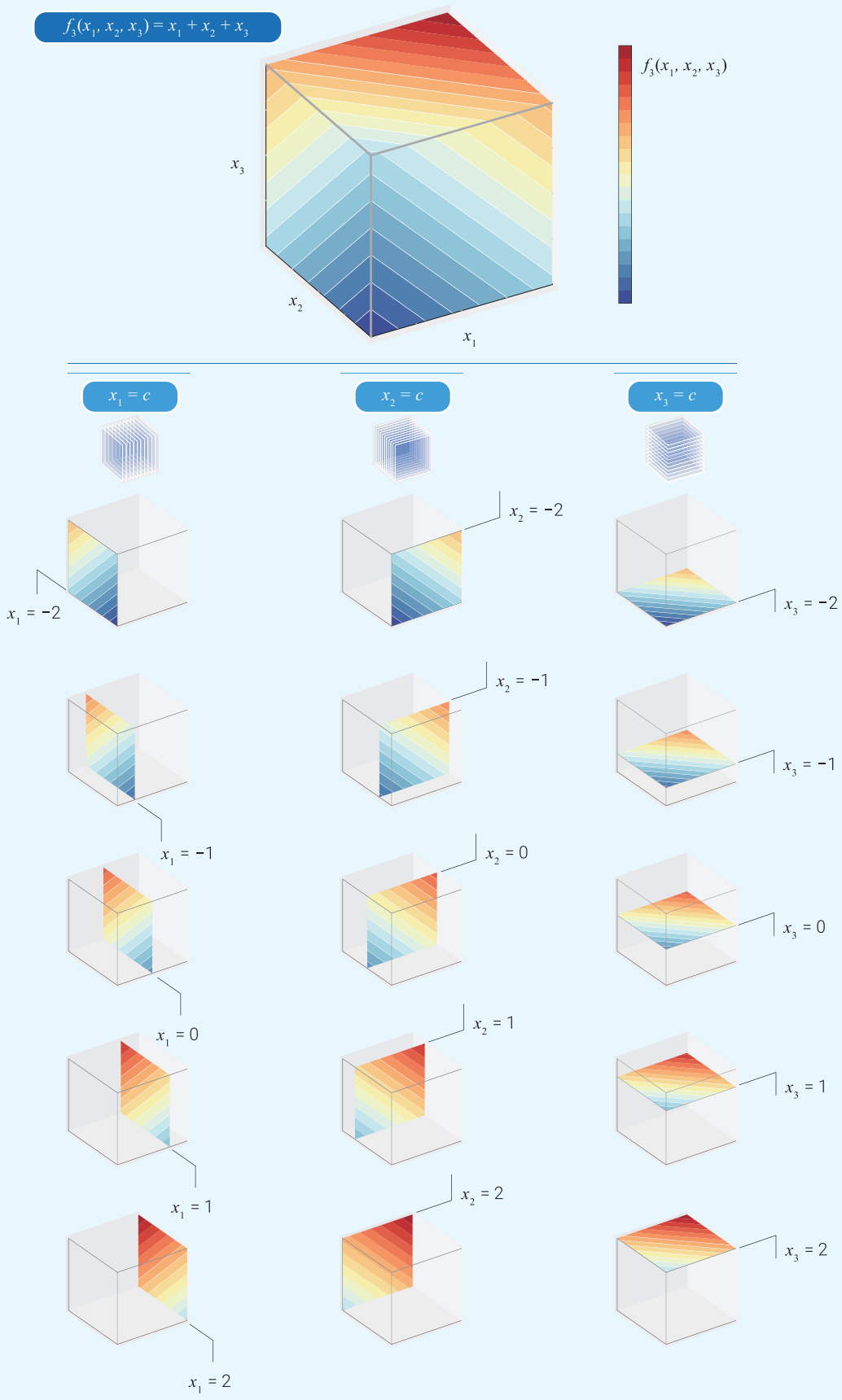


图20.4　一次函数，三元 | Bk2_Ch20_03.ipynb

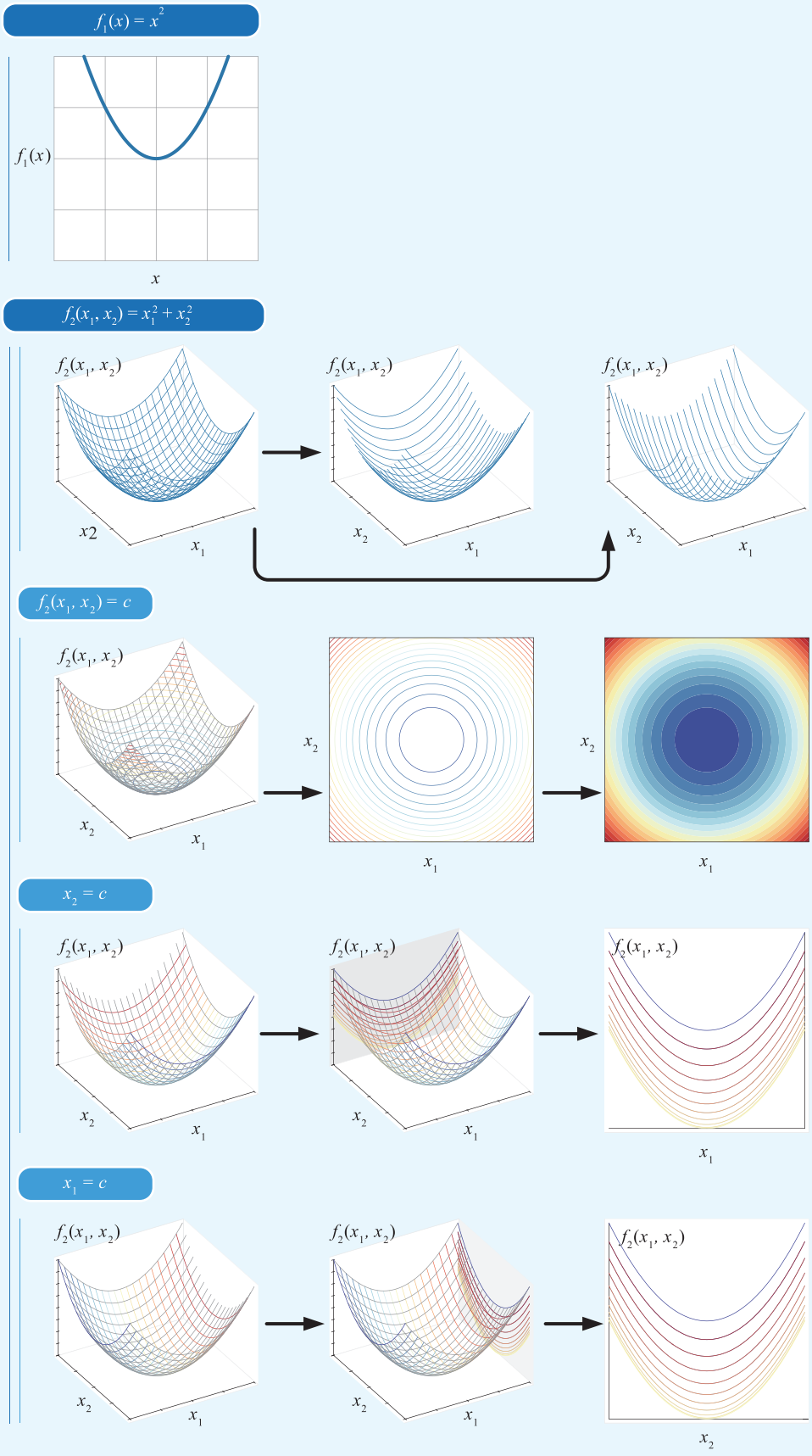


图20.5　二次函数，一元、二元