

7

Functions Meet Coordinate Systems

函数

从几何图形角度探究



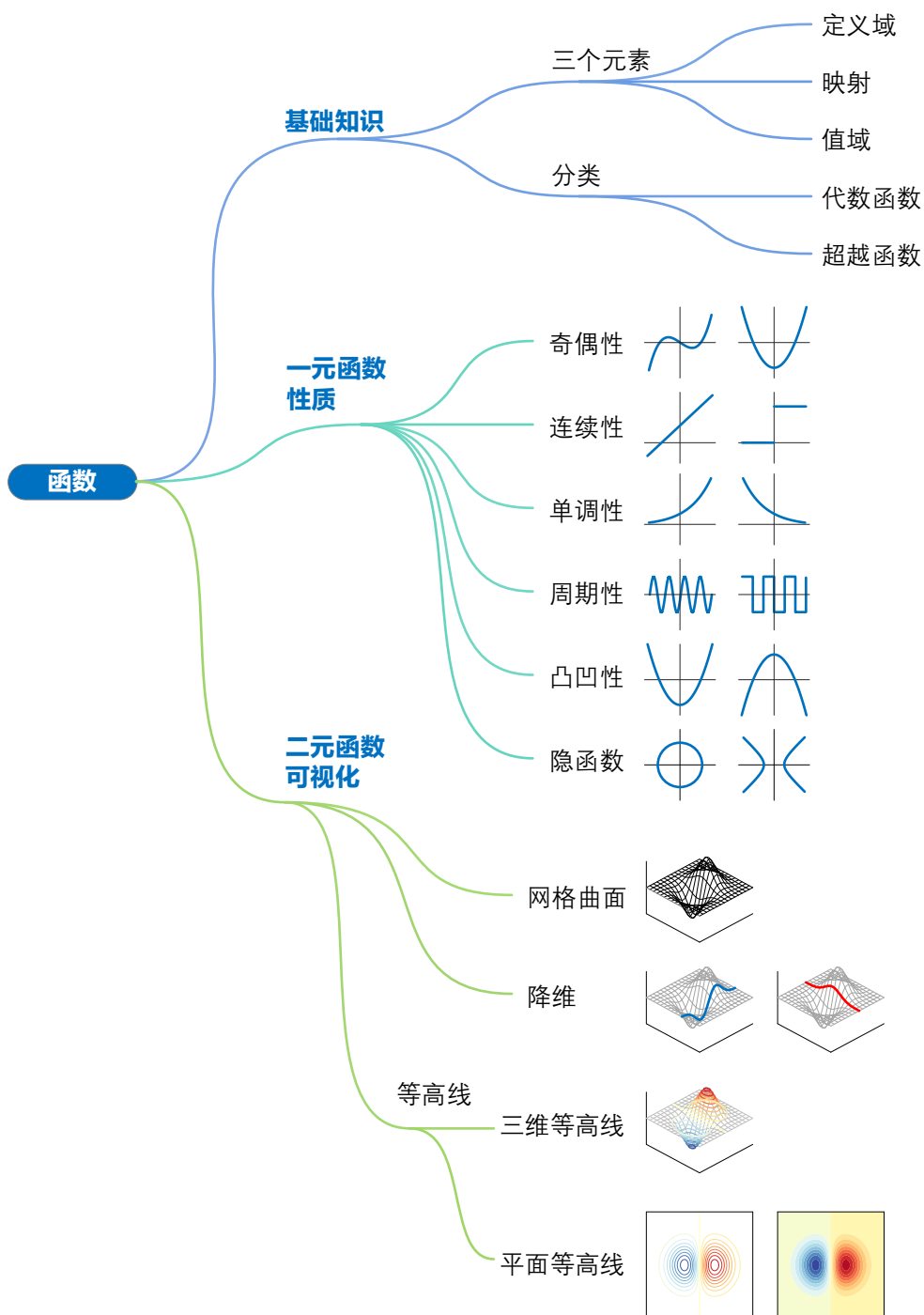
音乐是一种隐藏的数学实践，它是大脑潜意识下的计算。

Music is the hidden arithmetical exercise of a mind unconscious that it is calculating.

——戈特弗里德·莱布尼茨 (Gottfried Wilhelm Leibniz) | 德意志数学家、哲学家 | 1646 ~ 1716



- ▶ `matplotlib.pyplot.axhline()` 绘制水平线
- ▶ `matplotlib.pyplot.axvline()` 绘制竖直线
- ▶ `matplotlib.pyplot.contour()` 绘制等高线图
- ▶ `matplotlib.pyplot.contourf()` 绘制填充等高线图
- ▶ `numpy.linspace()` 在指定的间隔内，返回固定步长的数据
- ▶ `numpy.meshgrid()` 获得网格数据
- ▶ `plot_wireframe()` 绘制三维单色线框图
- ▶ `sympy.abc` 引入符号变量
- ▶ `sympy.diff()` 对符号函数求导
- ▶ `sympy.exp()` 符号运算中以 e 为底的指数函数
- ▶ `sympy.Interval` 定义符号区间
- ▶ `sympy.is_increasing` 判断符号函数的单调性
- ▶ `sympy.lambdify()` 将符号表达式转化为函数



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

7.1 当代数式遇到坐标系

图 1 ~ 图 3 给出了九幅图像，他们多数是函数，也有隐函数和参数方程。

坐标系给每个冷冰冰的代数式赋予生命；建议大家盯着每幅图像看一会就会惊奇地发现，坐标系给他们插上了翅膀，让他们在空间腾跃、讲述自己的故事。

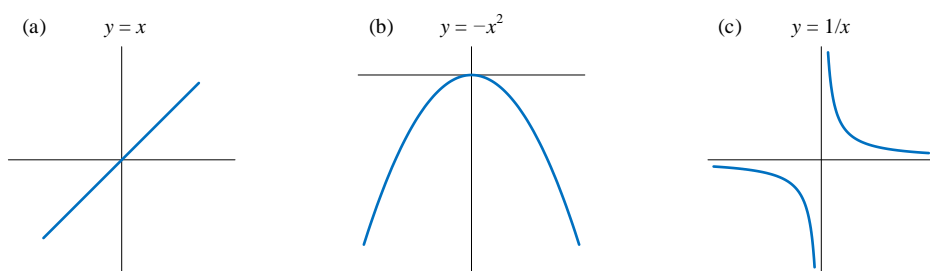


图 1. 一次函数、二次函数和反比例函数

线性函数 $y = x$ 是个坚毅果敢、埋头苦干的家伙。你问他，你要去哪？他莫不做声，自顾自地向着正负无穷，无限延伸，直到世界尽头。

抛物线 $y = -x^2$ 像一条腾出水面的锦鲤，在空中划出一道优美的弧线，他飞跃龙门、修成正果；从此岸到彼岸，离家越远，心就离家越近。

反比例函数 $y = 1/x$ 像一个哲学家，他在讲述——太极者，无极而生，动静之机，阴阳之母也。物极必反，任何事物都有两面，而且两面会互相转化。

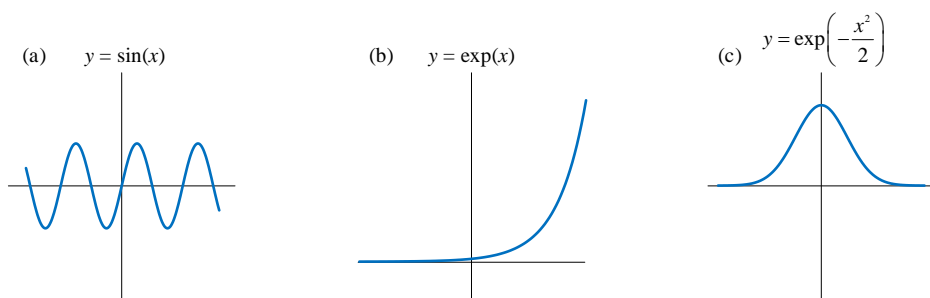


图 2. 正弦函数、指数函数和高斯函数

海水无风时，波涛安悠悠。正弦函数 $y = \sin(x)$ 像是海浪，永远波涛澎湃；它代表着生命的律动，你仿佛能够听到脉搏砰砰作响。

指数函数 $y = \exp(x)$ 就是那条巨龙。起初，他韬光养晦、潜龙勿用；万尺高楼起于累土，他不知疲倦、从未停歇；你看他，越飞越快，越升越高，如飞龙在天。

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

君不见黄河之水天上来，奔流到海不复回。优雅而神秘，高斯函数 $y = \exp(-x^2/2)$ 好比高山流水；上善若水，涓涓细流，利万物而不争。

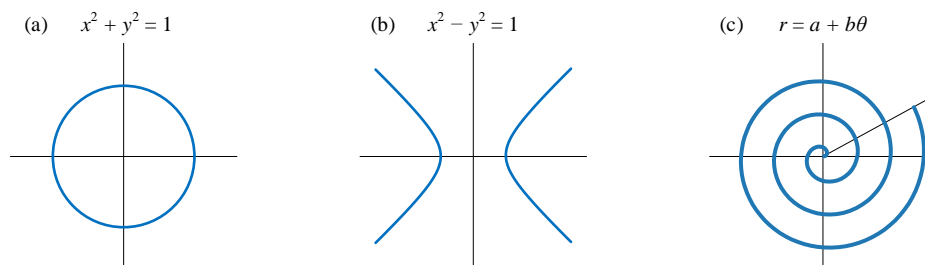


图 3. 正圆、双曲线和阿基米德螺旋线，非函数

海上生明月，天涯共此时。 $x^2 + y^2 = 1$ 是挂在天上的白玉盘，是家里客厅的圆饭桌，是捧在手里的圆月饼。转了一圈，起点是家。

可惜可叹，人有悲欢离合，月有阴晴圆缺，此事古难全。 $x^2 + y^2 = 1$ 正号 + 改为负号 -，就变成两条双曲线 $x^2 - y^2 = 1$ 。他们隔空相望，像牛郎和织女，盈盈一水间，脉脉不得语。

阿基米德螺旋线好似夜空中的银河星系，把我们的目光从人世的浮尘，拉到深蓝的虚空，让我们片刻间忘却了这片土地的悲欢离合。

7.2 一元函数：一个自变量

如果函数 f 以 x 作为唯一输入值，输出值写作 $y = f(x)$ ，函数就是一元函数；也就是说，一个自变量的函数叫做**一元函数** (univariate function)。

本书前文介绍过，函数输入值 x 构成的集合叫做定义域，函数输出值 y 构成的集合叫做值域。注意，定义域中任一 x 在值域中有唯一对应的 y ；当然，不同 x 可以对应一样的函数值 $f(x)$ 。

图 4 展示的便是一元函数的映射关系，以及几种一元函数示例。平面直角坐标系中，一般用线图 (line plot 或 line chart) 作为函数的可视化方案。

白话说，函数就是一种数值转化；上一章讲解不等式时，我们做过这样一个实验，给满足不等式条件的变量一个标签——1 (True)；不满足不等式的变量对应 0 (False)。这实际上也是函数映射，输入为定义域内自变量的取值，输出为两值之一 0 或 1。

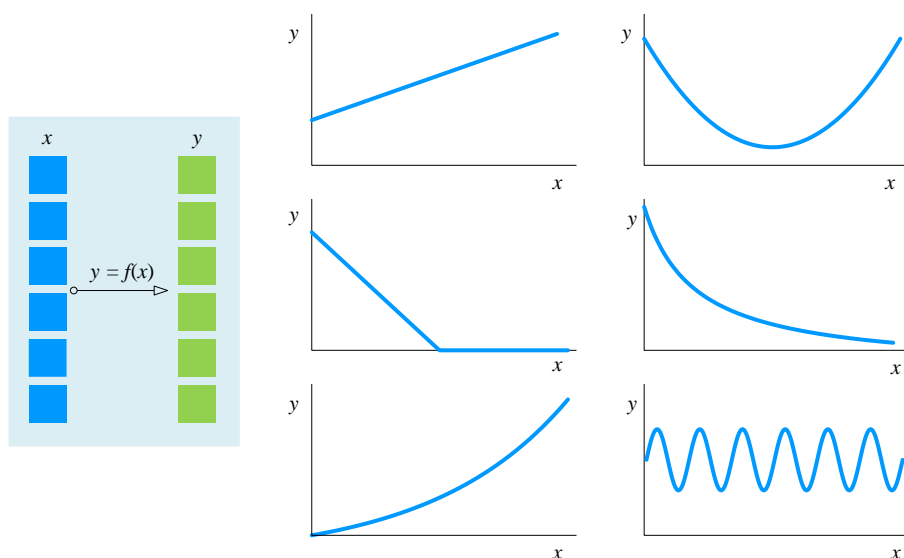


图 4. 一元函数

数据科学和机器学习中常用的函数一般分为**代数函数** (algebraic function) 和**超越函数** (transcendental function)。代数函数是指通过常数与自变量相互之间有限次的加、减、乘、除、有理指数幂和开方等运算构造的函数。超越函数指的是“超出”代数函数范畴的函数，比如对数函数、指数函数、三角函数等等。

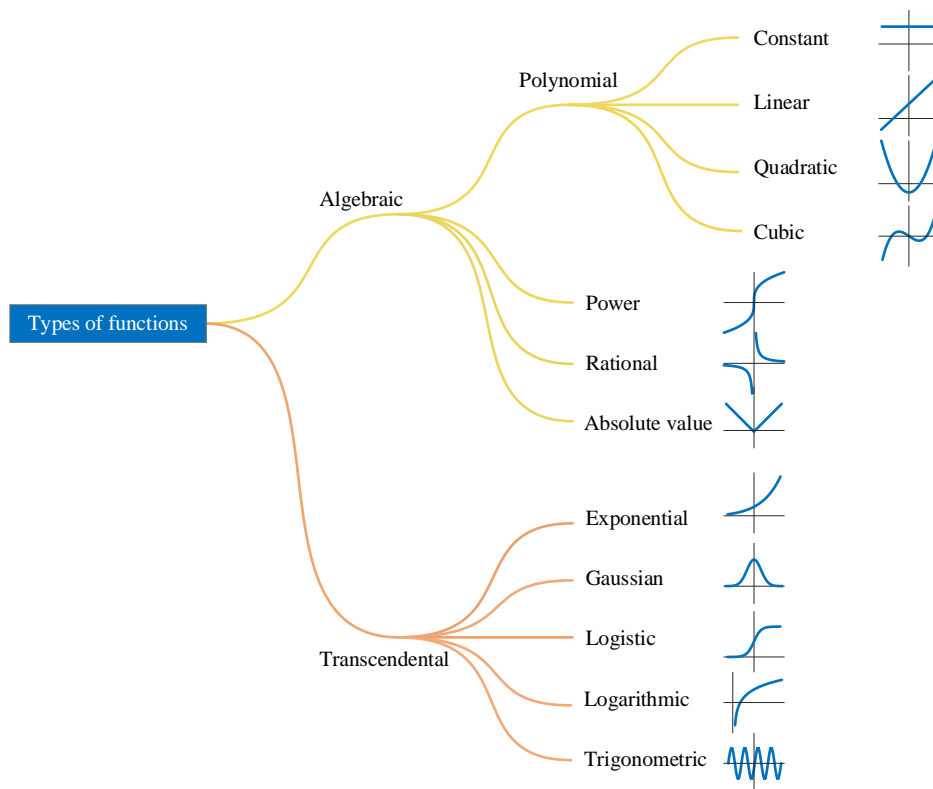


图 5. 常见函数分类



函数在机器学习中扮演重要作用。下面以神经网络 (neural network) 为例，简单介绍函数的作用。

神经网络的核心思想是模拟人脑**神经元** (neuron) 的工作原理。图 6 展示神经元基本生物学结构。神经元**细胞体** (cell body) 的核心是**细胞核** (nucleus)；细胞核周围围绕着**树突** (dendrite)。树突接受外部刺激，并将信号传递至神经元内部。

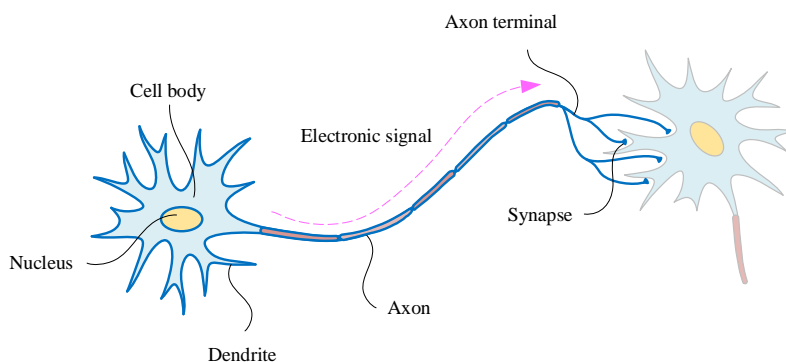


图 6. 神经元结构

细胞体汇总不同树突刺激，当刺激达到一定程度时，激发细胞兴奋状态；否则，细胞处于抑制状态。**轴突** (axon) 则负责将兴奋状态通过**轴突末端** (axon terminal) 的**突触** (synapse) 等结构传递到另一个神经元或组织细胞。

图 7 可看作是对神经元简单模仿。神经元模型的输入 x_1, x_2, \dots, x_D 类似于神经元的树突， x_i 取值为简单的 0 或 1。这些输入通过求和函数汇集到一起得到 x 。接着， x 值再通过一个如下的判别函数 $f()$ 得到最终的值 y 。

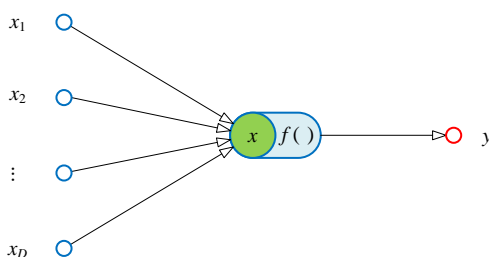


图 7. 最简单的神经网络模型

图 8 展示的便是几种常见的判别函数 $f()$ 及其对应图像。

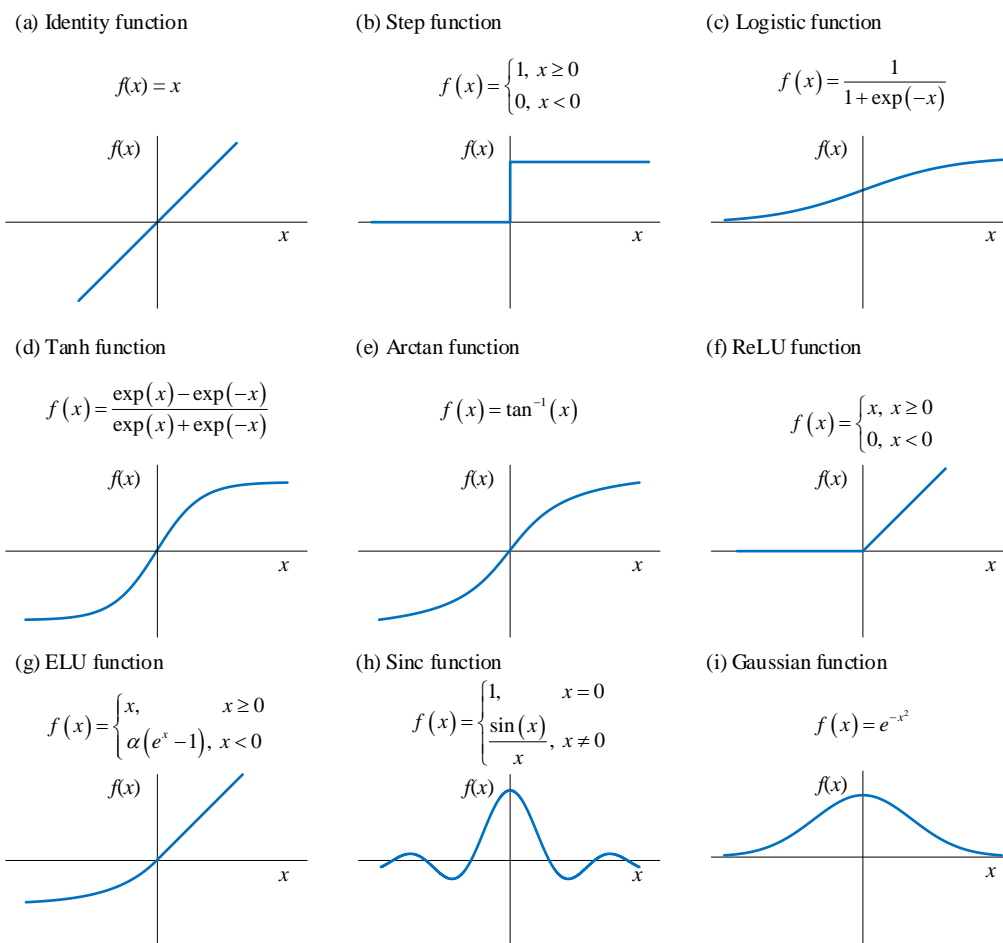


图 8. 几种常见的判别函数

7.3 一元函数性质

学习函数时，请大家关注函数这几个特征：形状及变化趋势、自变量取值范围、函数值取值范围、函数性质等。

下面，本节利用图 9 介绍一元函数常见性质。

奇偶性

图 9 (a) 展示**偶函数** (even function) 性质。 $f(x)$ 若为偶函数，对于定义域内任意 x 如下关系都成立。

$$f(x) = f(-x) \quad (1)$$

如图 9 (b) 所示，如果 $f(x)$ 为**奇函数** (odd function)，对于定义域内任意 x 如下关系都成立。

$$f(-x) = -f(x) \quad (2)$$

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

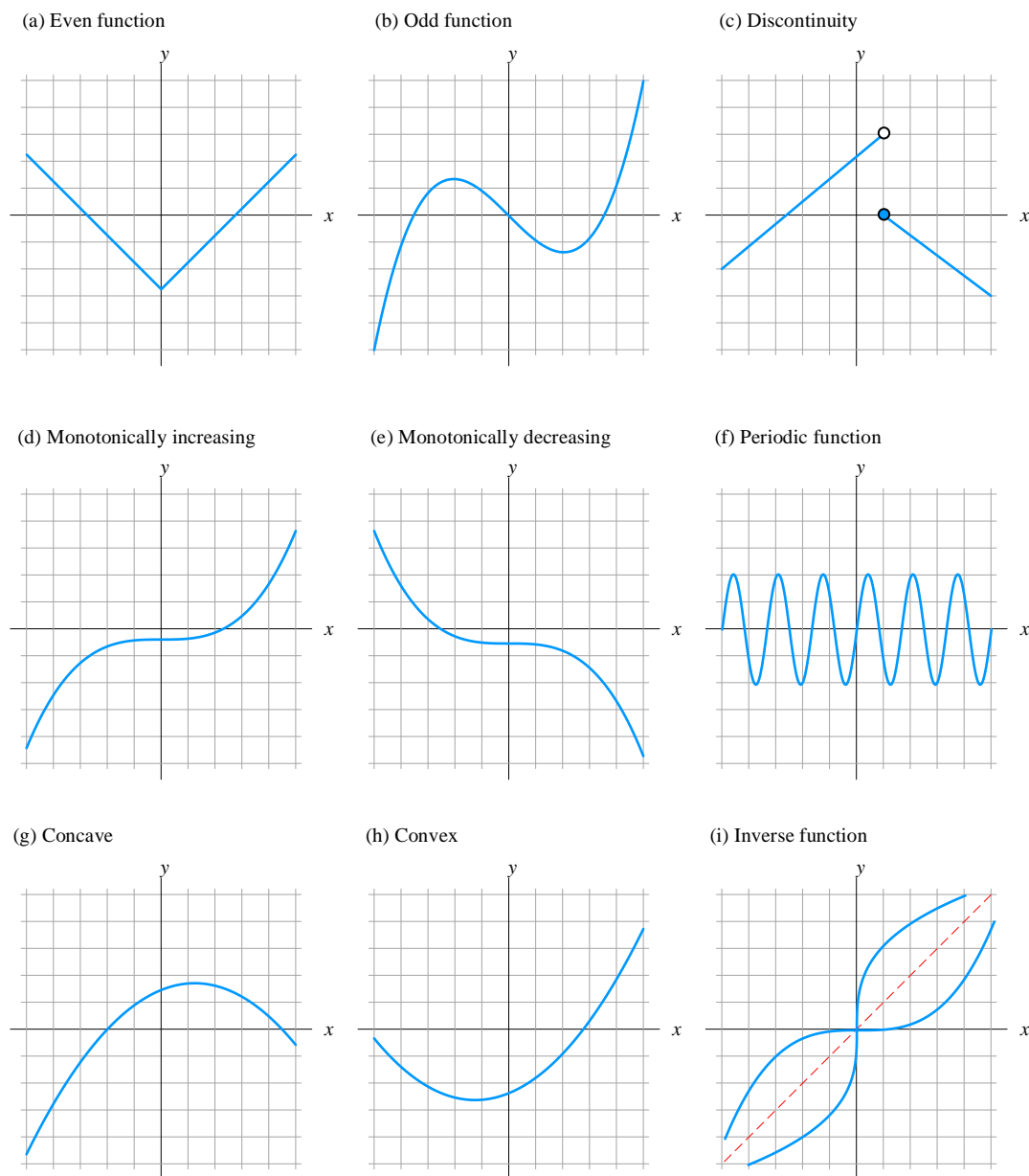


图 9. 一元函数常见性质

连续性

简单来说，**连续函数** (continuous function) 是指函数 $y = f(x)$ 当自变量 x 的变化很小时，所引起的因变量 y 的变化也很小；与之相对的就是**不连续函数** (discontinuous function)。图 9 (c) 给出的则是一个反例，函数存在不连续点 (discontinuity)。

如图 10 所示，不连续函数有几种：**渐近线间断** (asymptotic discontinuity)，**点间断** (point discontinuity)，还有**跳跃间断** (jump discontinuity)。

在学习**极限** (limit) 之后，函数的**连续性** (continuity) 更容易被定义。此外，函数的连续性和**可导性** (differentiability) 有着密切联系。这是本书后续要讨论的内容。

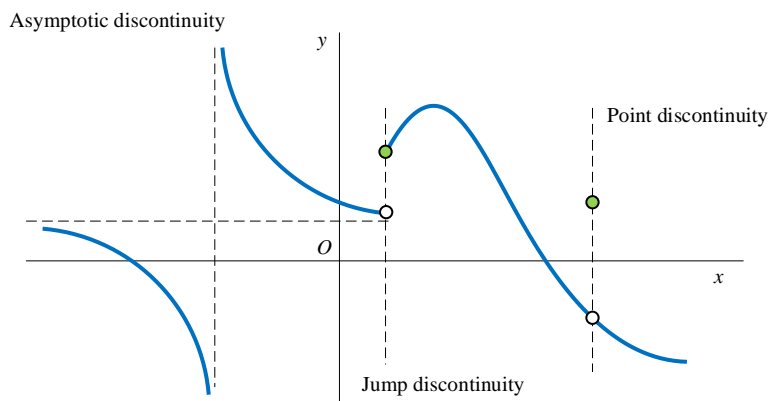


图 10. 几种不连续函数特征

单调性

图 9 (d) 和(e) 描述的是函数**单调性** (monotonicity)。图 9 (d) 给出的函数为**单调递增** (monotonically increasing)；图 9 (e) 函数则是**单调递减** (monotonically decreasing)。

`sympy.is_decreasing()` 可以用来判断符号函数的单调性；以下代码判断在不同区间内，特定函数的单调性。



```
# Bk Ch7 01

from sympy import is_increasing
from sympy.abc import x, y
from sympy import Interval, oo

expr_1 = -x**2
print(is_increasing(expr_1, Interval(-oo, 0)))

print(is_increasing(expr_1, Interval(0, oo)))

expr_2 = -x**2 + y
print(is_increasing(expr_2, Interval(-1, 2), x))
```

周期性

图 9 (f) 所示为函数**周期性** (periodicity)。如果函数 f 中不同位置 x 满足下式，则函数为周期函数。

$$f(x+T) = f(x) \quad (3)$$

其中， T 为**周期** (period)。

凸凹性

图 9 (g) 所示为**凹函数** (concave function)，图 9 (h) 所示为**凸函数** (convex function)。注意，国内数学教材对凸凹的定义，可能和本书正好相反。

函数的凸凹性和极值有着密切联系，本书后文会逐步介绍。

反函数

反函数 (inverse function) $x = f^{-1}(y)$ 的定义域、值域分别是函数 $y = f(x)$ 的值域、定义域。图 9 (i) 给出的是函数 f 和其反函数 f^{-1} ，两者关系如下。

$$f^{-1}(f(x)) = x \quad (4)$$

并不是所有函数都存在反函数。

隐函数

隐函数 (implicit function) 是由**隐式方程** (implicit equation) 所隐含定义的函数。比如，隐式方程 $F(x_1, x_2) = 0$ 描述 x_1 和 x_2 两者关系。

不同于一般函数，对于很多隐函数，很难分离自变量和因变量，比如图 11 所示四个例子。和函数一样，隐函数可以扩展到多元，比如图 12 所示为三元隐函数的例子。后续，我们会专门介绍如何用 Python 绘制如图 11 所示的隐函数图像。

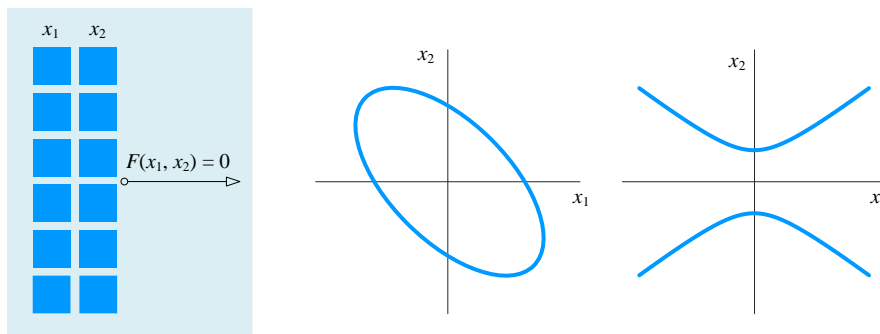


图 11. 二元隐函数

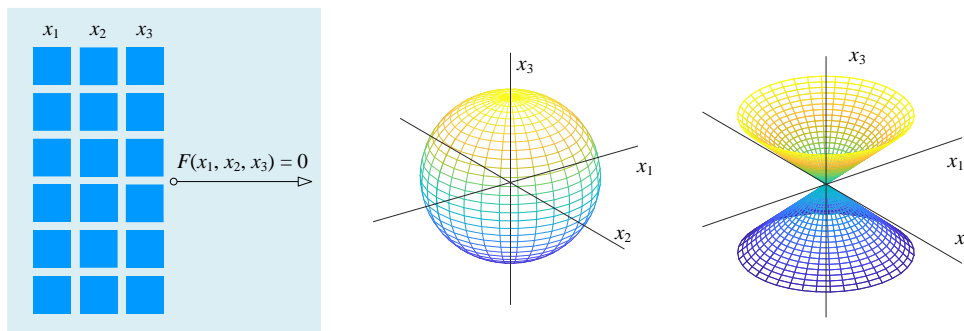


图 12. 三元隐函数

变化率和面积

很多数学问题要求我们准确地计算函数的变化率。如何量化函数变化率？或者说，如何确定函数曲线某一点的变化快慢？

从几何角度，如图 13 (a) 所示，函数上某一点切线的斜率正是函数的变化率；微积分中，这个变化率叫做导数 (derivative)。

如图 13 (b) 所示，一些数学问题求解面积、位移等，这就需要计算某个函数图形在一定取值范围和横轴围成几何图形的面积；这就需要大家了解积分 (integral) 这个数学工具。

本书后续会着重介绍导数和积分这两个数学工具。

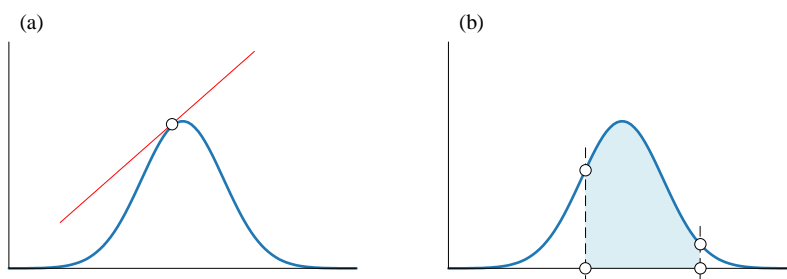


图 13. 函数的变化率和面积

7.4 二元函数：两个自变量

有两个自变量函数叫做**二元函数** (bivariate function)，比如 $y = f(x_1, x_2)$ 。本书常常借助三维直角坐标系可视化二元函数。图 14 所示为二元函数映射关系以及几个示例。

此外，有多个自变量函数叫做**多元函数** (multivariate function)，比如 $y = f(x_1, x_2, \dots, x_D)$ 。

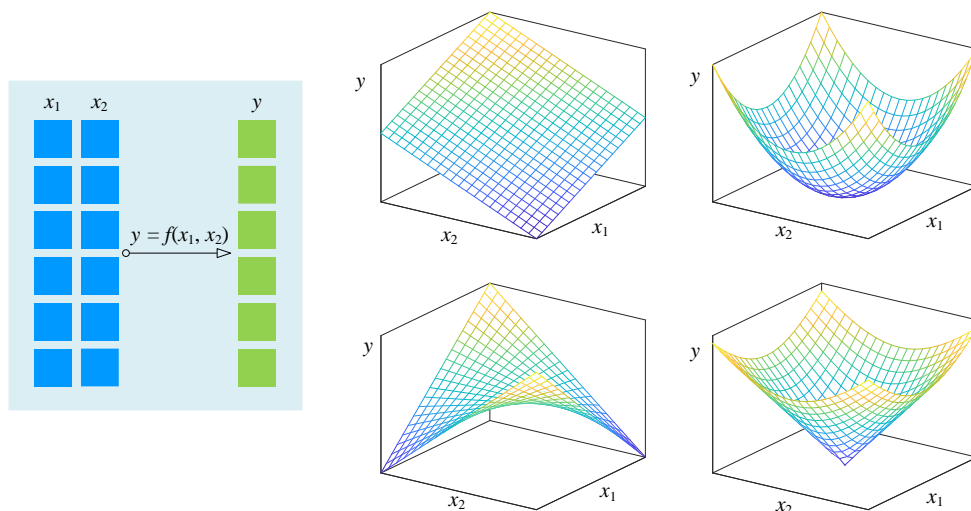


图 14. 二元函数

举个例子，二元一次函数 $y = f(x_1, x_2) = x_1 + x_2$ 有 x_1 和 x_2 两个自变量。当 x_1 和 x_2 取值分别为 $x_1 = 2$, $x_2 = 4$ ，函数值 $f(x_1, x_2) = 2 + 4 = 6$ 。

网格化数据

为了获得 $f(x_1, x_2)$ 在三维空间的图形，需要提供 x_1 和 x_2 一系列坐标值，

$$(x_1, x_2) = \begin{bmatrix} (-4, -4) & (-2, -4) & (0, -4) & (2, -4) & (4, -4) \\ (-4, -2) & (-2, -2) & (0, -2) & (2, -2) & (4, -2) \\ (-4, 0) & (-2, 0) & (0, 0) & (2, 0) & (4, 0) \\ (-4, 2) & (-2, 2) & (0, 2) & (2, 2) & (4, 2) \\ (-4, 4) & (-2, 4) & (0, 4) & (2, 4) & (4, 4) \end{bmatrix} \quad (5)$$

采用网格化数据，将上述坐标点 x_1 和 x_2 分离并写成矩阵形式。

$$x_1 = \begin{bmatrix} -4 & -2 & 0 & 2 & 4 \\ -4 & -2 & 0 & 2 & 4 \\ -4 & -2 & 0 & 2 & 4 \\ -4 & -2 & 0 & 2 & 4 \\ -4 & -2 & 0 & 2 & 4 \end{bmatrix}, \quad x_2 = \begin{bmatrix} -4 & -4 & -4 & -4 & -4 \\ -2 & -2 & -2 & -2 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 \end{bmatrix} \quad (6)$$

式中， x_1 的每个值代表点的横坐标值， x_2 的每个值代表点的纵坐标值。`numpy.meshgrid()` 可以用来获得网格化数据。

$y = f(x_1, x_2) = x_1 + x_2$ 这个二元函数便是将 (6) 相同位置的数值相加得到函数值 $f(x_1, x_2)$ 的矩阵。

$$f(x_1, x_2) = x_1 + x_2 = \begin{bmatrix} -4 & -2 & 0 & 2 & 4 \\ -4 & -2 & 0 & 2 & 4 \\ -4 & -2 & 0 & 2 & 4 \\ -4 & -2 & 0 & 2 & 4 \\ -4 & -2 & 0 & 2 & 4 \end{bmatrix} + \begin{bmatrix} -4 & -4 & -4 & -4 & -4 \\ -2 & -2 & -2 & -2 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 \end{bmatrix} = \begin{bmatrix} -8 & -6 & -4 & -2 & 0 \\ -6 & -4 & -2 & 0 & 2 \\ -4 & -2 & 0 & 2 & 4 \\ -2 & 0 & 2 & 4 & 6 \\ 0 & 2 & 4 & 6 & 8 \end{bmatrix} \quad (7)$$

图 15 所示为 $f(x_1, x_2) = x_1 + x_2$ 对应的三维空间平面。函数 $f()$ 则代表某种规则，将网格化数据从 x_1x_2 平面映射到三维空间。

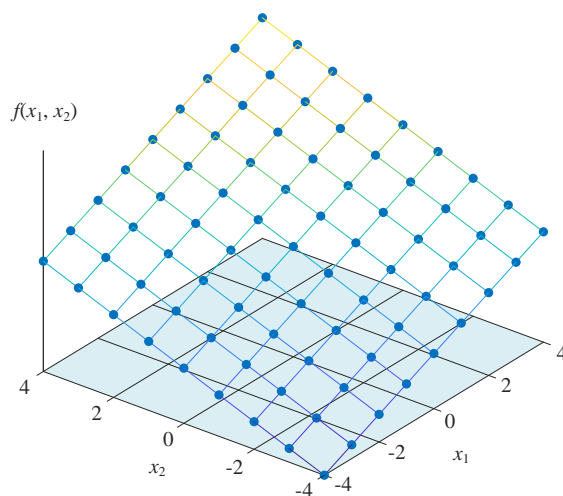


图 15. $f(x_1, x_2) = x_1 + x_2$ 对应的三维空间平面

这就是前文说的，在绘制函数图像时，比如二元函数曲面，实际上输入的函数值都是离散的、网格化的；当然，网格越密，函数曲面越精确。

实际应用中，网格的疏密可以根据函数的复杂度调整。比如图 15 这幅平面图像很简单，因此可以用比较稀疏的网格来呈现图像；但是，对于比较复杂的函数，网格则需要设置的密一些，也就是步长小一些。

下面看一个复杂二元函数 $f(x_1, x_2)$ 对应的曲面。图 16 对应的函数解析式为。

$$f(x_1, x_2) = 3(1 - x_1)^2 \exp(-x_1^2 - (x_2 + 1)^2) - 10\left(\frac{x_1}{5} - x_1^3 - x_2^5\right) \exp(-x_1^2 - x_2^2) - \frac{1}{3} \exp(-(x_1 + 1)^2 - x_2^2) \quad (8)$$

相对图 15，图 16 的网格更为密集。本章后续有关二元函数的可视化方案，都是以上述二元函数作为例子。

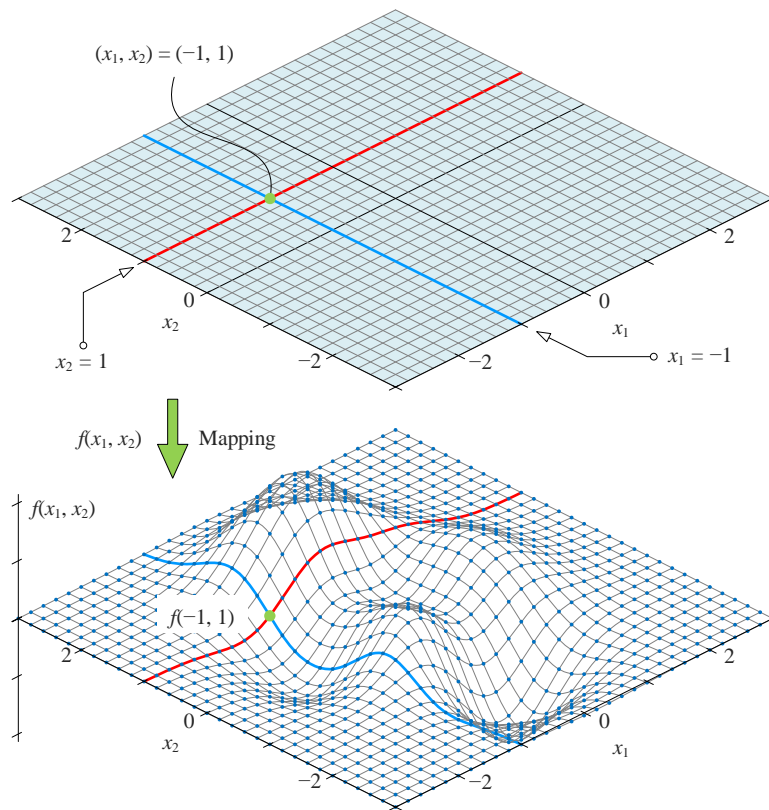
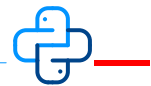


图 16. 网格化数据与二元函数映射

以下代码绘制图 16 二元函数 $f(x_1, x_2)$ 对应的网格曲面。



```
# Bk_Ch7_02_A

import numpy as np
from sympy import lambdify, diff, exp, latex, simplify
from sympy.abc import x, y
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm

num = 301; # number of mesh grids
x_array = np.linspace(-3,3,num)
y_array = np.linspace(-3,3,num)
xx,yy = np.meshgrid(x_array,y_array)

plt.close('all')
# f_xy = x*exp(- x**2 - y**2);
f_xy = 3*(1-x)**2*exp(-(x**2) - (y+1)**2) \
        - 10*(x/5 - x**3 - y**5)*exp(-x**2-y**2) \
        - 1/3*exp(-(x+1)**2 - y**2)

f_xy_fcn = lambdify([x,y],f_xy)
f_xy_zz = f_xy_fcn(xx,yy)

%% visualize the surface
```

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

```
fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

ax.plot_wireframe(xx,yy, f_xy_zz,
                  color = [0.5,0.5,0.5],
                  rstride=5, cstride=5,
                  linewidth = 0.25)

ax.set_proj_type('ortho')
ax.set_xlabel('$x$'); ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
ax.set_xlim(xx.min(), xx.max()); ax.set_ylim(yy.min(), yy.max())
ax.view_init(azim=-135, elev=30)
plt.tight_layout()
ax.grid(False)
plt.show()
```

7.5 降维：二元函数切一刀得到一元函数

如图 17 所示为二元函数两种可视化工具——剖面线、等高线。

本节介绍剖面线，它相当于在曲面上沿着横轴或纵轴切一刀；我们关注的是曲面截面处曲线的变化趋势，这相当于降维。

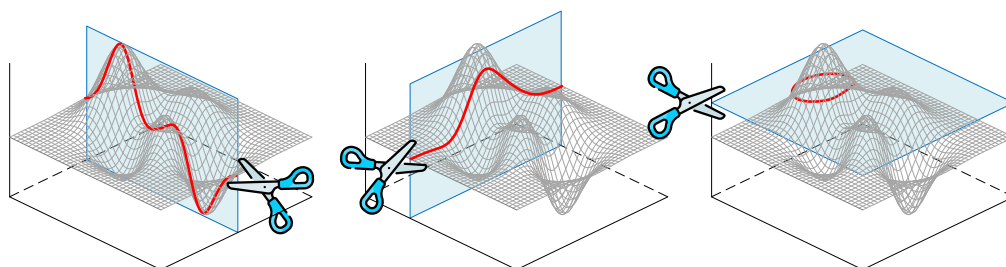


图 17. 函数降维

x_1y 平面方向剖面线

以 $f(x_1, x_2)$ 二元函数为例，如果自变量 x_2 固定 $x_2 = c$ ，只有自变量 x_1 变化，则 $f(x_1, x_2 = c)$ 相当于是 x_1 的一元函数。

图 18 中彩色曲线所示为 x_2 固定在几个具体值 c 时， $f(x_1, x_2 = c)$ 随 x_1 变化的剖面线。这些剖面线就是一元函数；利用一元函数性质，我们可以分析曲面在不同位置的变化趋势。

此外，本书后文介绍的偏导数 (partial derivative) 就是研究这些剖面线的变化率的数学工具。

注意，通过剖面线得出的结论不能推广到整个二元函数。

如图 19 所示，将一系列 $f(x_1, x_2 = c)$ 剖面线投影在 x_1y 平面上，给每条曲线涂上不同颜色，可以得到图 20。

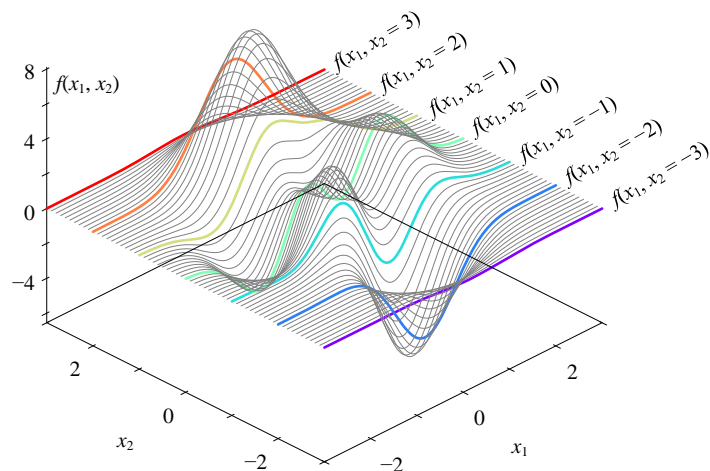
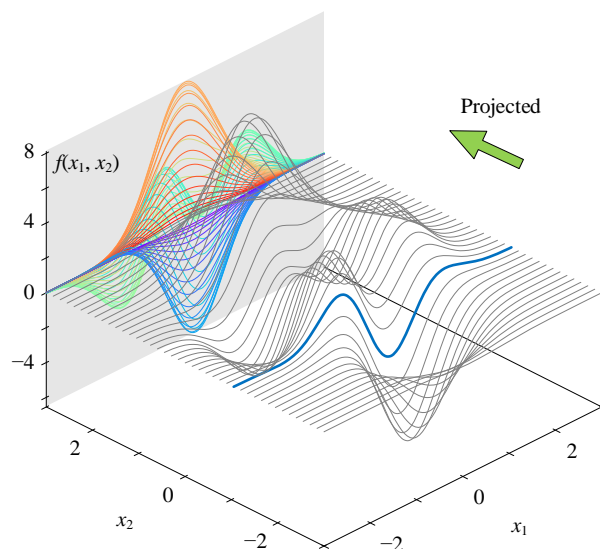
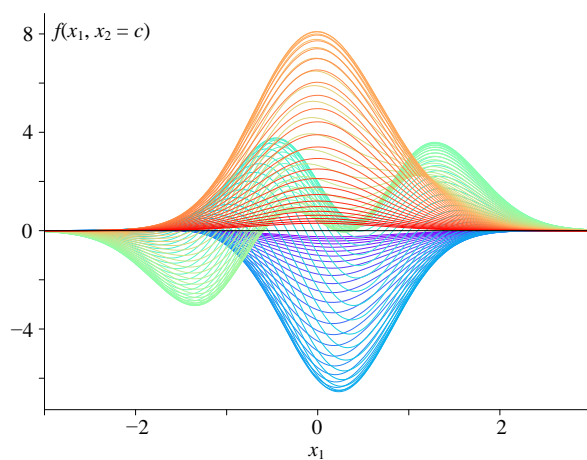
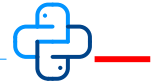
图 18. 自变量 x_2 固定，自变量 x_1 变化图 19. 将 $f(x_1, x_2)$ 剖面线投影到 x_1y 平面

图 20. 剖面线在 x_1y 平面投影

配合前文代码，以下代码绘制图 18、图 19、图 20。



```
# Bk_Ch7_02_B

### evaluate f(x1,x2) with x2 = c

fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

ax.plot_wireframe(xx,yy, f_xy_zz,
                  color = [0.5,0.5,0.5],
                  rstride=5, cstride=0,
                  linewidth = 0.25)

# x2 = b

colors = plt.cm.rainbow(np.linspace(0,1,7))

i = 0

for b in [-3,-2,-1,0,1,2,3]:
    f_xy_b = f_xy.subs(y, b)

    print('=====' )
    print('x_2 = %0.0f'%b)
    print('f(x1,x2 = %0.0f) = %s'% (b, str(simplify(f_xy_b))))
    print('=====' )

    f_xy_b_fcn = lambdify([x],f_xy_b)
    f_xy_b_zz = f_xy_b_fcn(x_array)

    ax.plot(x_array,x_array*0 + b,f_xy_b_zz,
            color = colors[i,:],
            label = '$x_2$ = %0.0f'%b)

    i = i + 1

plt.legend()
ax.set_proj_type('ortho')
ax.set_xlabel('$x$'); ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.view_init(azim=-135, elev=30)
plt.tight_layout()
ax.grid(False)
plt.show()

### surface projected along Y to X-Z plane

fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

ax.plot_wireframe(xx,yy, f_xy_zz,
                  rstride=5, cstride=0,
                  color = [0.5,0.5,0.5],
                  linewidth = 0.25)

ax.contour(xx,yy, f_xy_zz,
           levels = 60, zdir='y', \
           offset= yy.max(), cmap='rainbow')

ax.set_xlabel('$x$'); ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
```

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

```

ax.set_proj_type('ortho')

ax.view_init(azim=-135, elev=30)
ax.grid(False)
ax.set_xlim(xx.min(), xx.max()); ax.set_ylim(yy.min(), yy.max())
# ax.set_zlim(0, 0.7)
plt.tight_layout()

plt.show()

# project down-sampled surface

down_step = 2;
y_array_downsample = y_array[0::down_step]

fig, ax = plt.subplots()

colors = plt.cm.rainbow(np.linspace(0,1,len(y_array_downsample)))

for i in np.linspace(1,len(y_array_downsample),len(y_array_downsample)):
    plt.plot(x_array,f_xy_zz[(int(i)-1)*down_step,:],
             color = colors[int(i)-1])

plt.axhline(y=0, color='k', linestyle='-')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.xlabel('$x_1$')
plt.ylabel('$f(x_1, x_2 = b)$')
ax.set_xlim(xx.min(), xx.max())

```

x_2y 平面方向剖面线

自变量 x_1 固定，只有自变量 x_2 变化，则 $f(x_1, x_2)$ 相当于是 x_2 的一元函数。图 21 中彩色曲线所示为 x_1 固定在具体值 c 时， $f(x_1 = c, x_2)$ 随 x_2 变化。

如图 22 所示，将 $f(x_1 = c, x_2)$ 剖面线投影在 x_2y 平面上；给每条曲线涂上不同颜色，可以得到图 23。

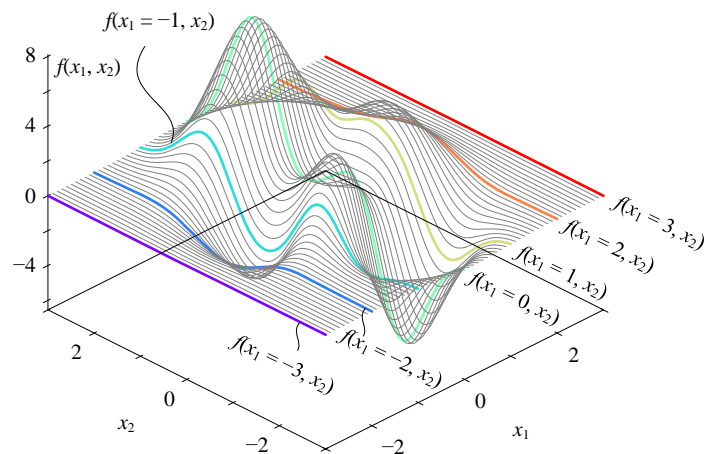
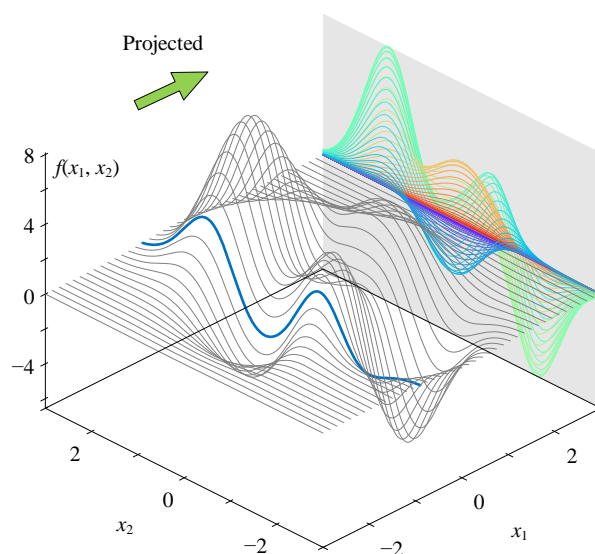
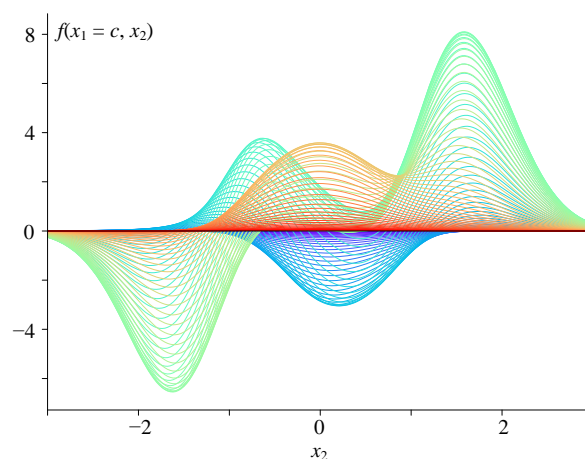


图 21. 自变量 x_1 固定，自变量 x_2 变化

图 22. 将 $f(x_1, x_2)$ 剖面线投影到 x_2y 平面图 23. 剖面线在 x_2y 平面投影

配合前文代码，以下代码绘制图 21、图 22、图 23。



```
# Bk_Ch7_02_C

#%% evaluate f(x1,x2) with x1 = a

fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

ax.plot_wireframe(xx,yy, f_xy_zz,
                  color = [0.5,0.5,0.5],
                  rstride=0, cstride=5,
                  linewidth = 0.25)
```

```

# x1 = a

colors = plt.cm.rainbow(np.linspace(0,1,7))

i = 0

for a in [-3,-2,-1,0,1,2,3]:
    f_x_a_y = f_xy.subs(x, a)

    print('=====' )
    print('x_1 = %0.0f'%a)
    print('f(x1 = %0.0f,x2) = %s)'%(a,str(simplify(f_x_a_y))))
    print('=====' )

    f_x_a_y_fcn = lambdify([y],f_x_a_y)
    f_x_a_y_zz = f_x_a_y_fcn(y_array)

    ax.plot(y_array*0 + a,y_array,f_x_a_y_zz,
            color = colors[i,:],
            label = '$x_1$ = %0.0f'%a)

    i = i + 1

plt.legend()
ax.set_proj_type('ortho')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.view_init(azim=-135, elev=30)
plt.tight_layout()
ax.grid(False)
plt.show()

%% surface projected along X to Y-Z plane

fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

ax.plot_wireframe(xx,yy, f_xy_zz,
                  rstride=0, cstride=5,
                  color = [0.5,0.5,0.5],
                  linewidth = 0.25)

ax.contour(xx,yy, f_xy_zz,
           levels = 60, zdir='x', \
           offset= xx.max(), cmap='rainbow')

ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
ax.set_proj_type('ortho')
ax.view_init(azim=-135, elev=30)
ax.grid(False)
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
plt.tight_layout()
plt.show()

# project down-sampled surface

down_step = 2;
x_array_downsample = x_array[0::down_step]

fig, ax = plt.subplots()

colors = plt.cm.rainbow(np.linspace(0,1,len(x_array_downsample)))

for i in np.linspace(1,len(x_array_downsample),len(x_array_downsample)):

```

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

```
plt.plot(y_array, f_xy_zz[:, (int(i)-1)*down_step],
        color = colors[int(i)-1])
plt.axhline(y=0, color='k', linestyle='-')
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.xlabel('$x_2$')
plt.ylabel('$f(x_1 = a, x_2)$')
ax.set_xlim(xx.min(), xx.max())
```

7.6 等高线：由函数值相等点连成

把图 24 所示 $f(x_1, x_2)$ 曲面比作一座山峰，函数值越大，相当于山峰越高。图中用暖色色块表达山峰，用冷色色块表达洼地。

等高线

三维等高线 (contour line) 和平面等高线是研究二元函数重要的手段之一。上一章在讲不等式时，我们简单提过等高线。简单来说，曲面某一条等高线就是函数值 $f(x_1, x_2)$ 相同，即 $f(x_1, x_2) = c$ 的相邻点连接构成的曲线。

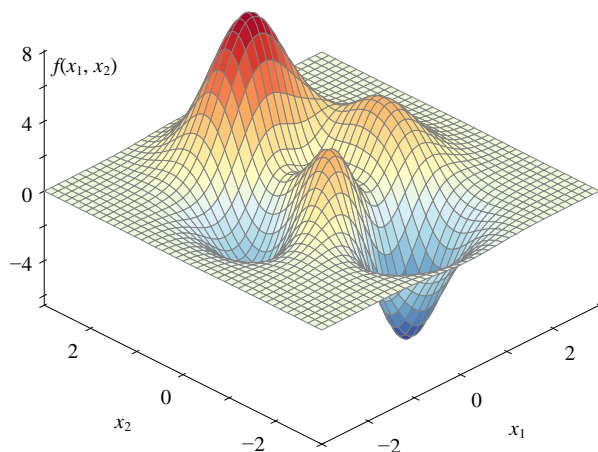


图 24. 用冷暖色表示函数的不同高度取值

当 c 取不同值时，便可以得到一系列对应不同高度的等高线，获得的图像便是三维等高线图，如图 25 所示线。这些曲线可以是闭合曲线，也可以非闭合。

将这些曲线垂直投影到水平面上，得到平面等高线图。

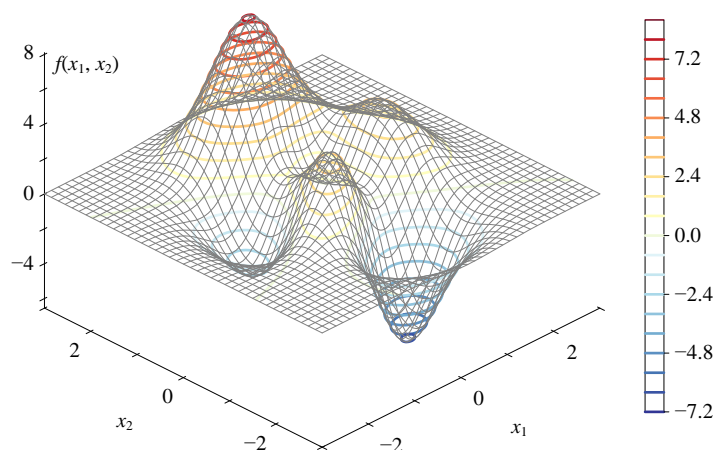


图 25. 二元函数三维等高线

图 26 所示 $f(x_1, x_2)$ 曲面等高线在 x_1x_2 平面上的投影结果。平面等高线图中，每条不同颜色的曲线代表一个具体函数取值；把二元函数比作山峰的话，等高线越密集的区域，坡度越陡峭。

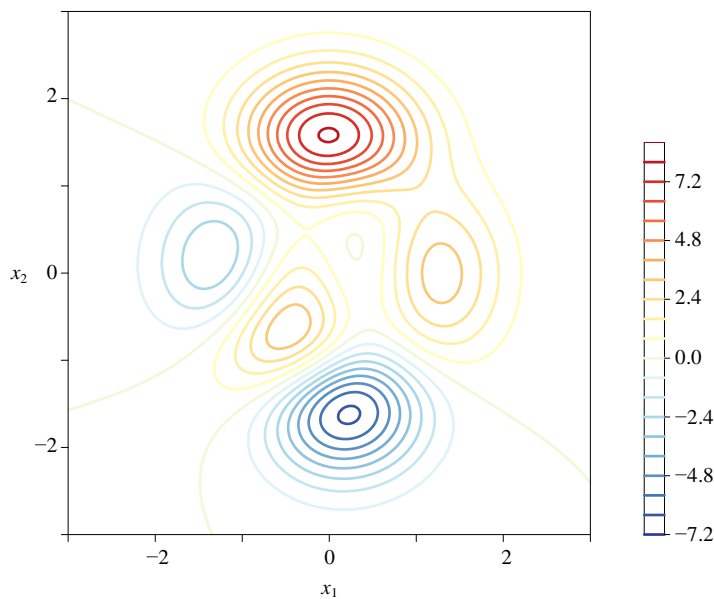


图 26. 二元函数的平面等高线

填充等高线

还可以使用填充等高线来可视化二元函数。

图 27 所示为 $f(x_1, x_2)$ 三维坐标系中在水平面上得到平面填充等高线。图 28 就是填充等高线在 x_1x_2 平面上的投影结果。注意，同一个颜色色块代表函数取值在某一特定区间内 $[c_i, c_{i+1}]$ 。

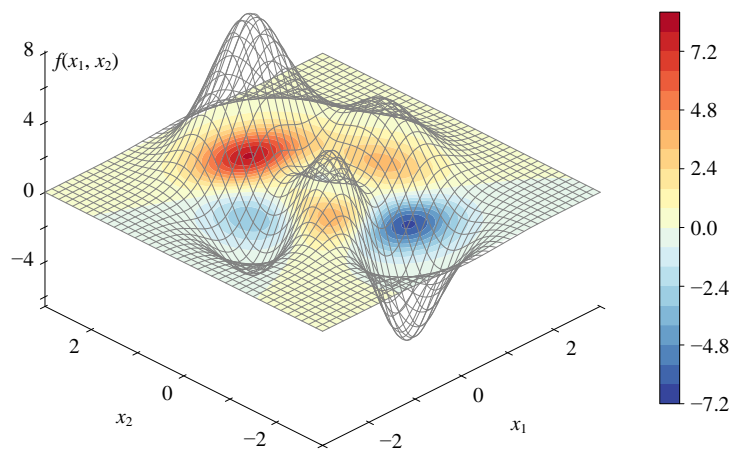


图 27. 三维曲面投影在水平面上得到平面填充等高线

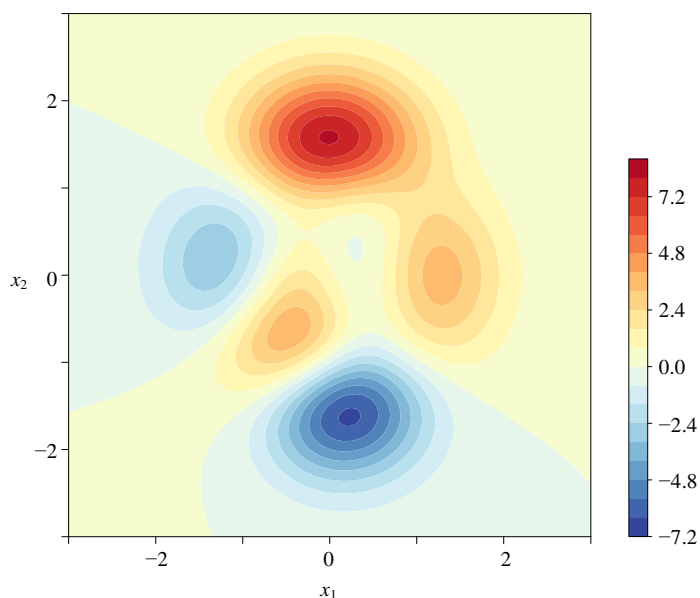


图 28. 平面填充等高线图

配合前文代码，以下代码绘制图 24 ~ 图 28。



```
# Bk_Ch7_02_D
#%% surface projected along Z to X-Y plane, contour
fig, ax = plt.subplots(subplot_kw={'projection': '3d'})
ax.plot_surface(xx,yy, f_xy_zz,
```

```

        cmap=cm.RdYlBu_r,
        rstride=5, cstride=5,
        linewidth = 0.25,
        edgecolors = [0.5,0.5,0.5])

ax.set_proj_type('ortho')

ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.view_init(azim=-135, elev=30)
plt.tight_layout()
ax.grid(False)
plt.show()

fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

ax.plot_wireframe(xx,yy, f_xy_zz,
                  color = [0.5,0.5,0.5],
                  linewidth = 0.25)

colorbar = ax.contour(xx,yy, f_xy_zz,20,
                      cmap = 'RdYlBu_r')

fig.colorbar(colorbar, ax=ax)

ax.set_proj_type('ortho')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
plt.tight_layout()
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.view_init(azim=-135, elev=30)
ax.grid(False)
plt.show()

fig, ax = plt.subplots()

colorbar = ax.contour(xx,yy, f_xy_zz, 20, cmap='RdYlBu_r')
fig.colorbar(colorbar, ax=ax)

ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())

ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
plt.gca().set_aspect('equal', adjustable='box')

plt.show()

fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

ax.plot_wireframe(xx,yy, f_xy_zz,
                  rstride=5, cstride=5,
                  color = [0.5,0.5,0.5],
                  linewidth = 0.25)

colorbar = ax.contourf(xx,yy, f_xy_zz,
                      levels = 20, zdir='z', \
                      offset= 0, cmap='RdYlBu_r')

fig.colorbar(colorbar, ax=ax)

ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$f(x,y)$')
ax.set_proj_type('ortho')
# ax.set_xticks([])

```

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


```

# ax.set_yticks([])
# ax.set_zticks([])
ax.view_init(azim=-135, elev=30)
ax.grid(False)
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
# ax.set_zlim(0, 0.7)
plt.tight_layout()

plt.show()

fig, ax = plt.subplots()

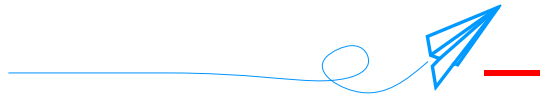
colorbar = ax.contourf(xx, yy, f_xy_zz, 20, cmap='RdYlBu_r')
fig.colorbar(colorbar, ax=ax)

ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())

ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
plt.gca().set_aspect('equal', adjustable='box')

plt.show()

```



没有坐标系，就没有函数。坐标系给函数以生命。希望大家，在学习任何函数时，首先想到的是求助于坐标系。

特别强调，在描绘函数形状和变化趋势时，千万不能按自己审美偏好“手绘”函数图像！函数图像必须通过编写代码绘制！

作者在很多数学教科书中，看到很多不负责任的“手绘”函数图像；特别不能容忍的是手绘高斯函数或高斯分布概率密度函数曲线，这简直就是暴殄天物！

哪怕技艺精湛，手绘函数也不能准确描绘函数的每一处细节。

即便是编程绘制的图像也不是百分之百准确无误，因为这些图像是散点连接而成的；只不过当这些点和点之间步长较小时，图像看上去连续光滑罢了。