

25

Symbolic Computation in SymPy

SymPy 符号运算

SymPy 是一个 Python 的符号数学计算库



等式仅仅是数学中无聊至极的那部分；我努力从几何角度观察万物。

Equations are just the boring part of mathematics. I attempt to see things in terms of geometry.

—— 斯蒂芬·霍金 (Stephen Hawking) | 英国理论物理学家和宇宙学家 | 1942 ~ 2018



- ◀ `sympy.abc import x` 定义符号变量 `x`
- ◀ `sympy.abc()` 引入符号变量
- ◀ `sympy.collect()` 合并同类项
- ◀ `sympy.cos()` 符号运算中余弦
- ◀ `sympy.diff()` 求解符号导数和偏导解析式
- ◀ `sympy.Eq()` 定义符号等式
- ◀ `sympy.evalf()` 将符号解析式中未知量替换为具体数值
- ◀ `sympy.exp()` 符号自然指数
- ◀ `sympy.expand()` 展开代数式
- ◀ `sympy.factor()` 对代数式进行因式分解
- ◀ `sympy.integrate()` 符号积分
- ◀ `sympy.is_decreasing()` 判断符号函数的单调性
- ◀ `sympy.lambdify()` 将符号表达式转化为函数
- ◀ `sympy.limit()` 求解极限
- ◀ `sympy.Matrix()` 构造符号函数矩阵
- ◀ `sympy.plot_implicit()` 绘制隐函数方程
- ◀ `sympy.plot3d()` 绘制函数的三维曲面
- ◀ `sympy.series()` 求解泰勒展开级数符号式
- ◀ `sympy.simplify()` 简化代数式
- ◀ `sympy.sin()` 符号运算中正弦
- ◀ `sympy.solve()` 求解符号方程组
- ◀ `sympy.solve_linear_system()` 求解含有符号变量的线型方程组
- ◀ `sympy.symbols()` 创建符号变量
- ◀ `sympy.sympify()` 化简符号函数表达式
- ◀ `sympy.utilities.lambdify.lambdify()` 将符号代数式转化为函数



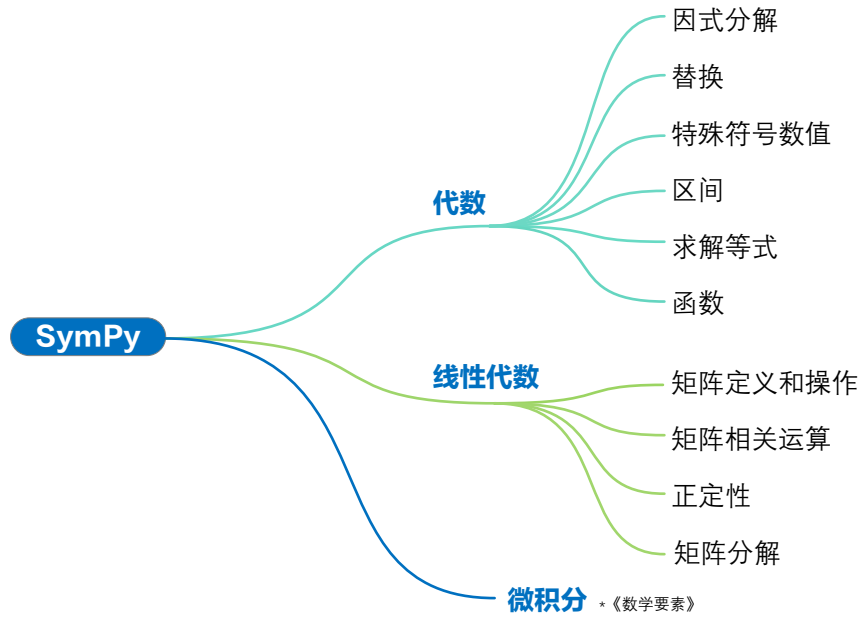
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

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



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

25.1 什么是 SymPy?

SymPy 是一个基于 Python 的符号数学库，它可以执行代数运算、解方程、微积分、离散数学以及其他数学操作。

与 NumPy、Pandas 等科学计算库不同，SymPy 主要关注的是符号计算而不是数值计算。具体来说，SymPy 可以处理未知变量和数学符号，而不仅仅是数值，这在一些数学研究和工程应用中非常有用。

本章主要介绍 SymPy 中代数、线性代数运算。此外，SymPy 还可以进行微积分运算，比如极限、导数、偏导数、泰勒展开、积分等。这部分内容需要一定的数学分析知识，我们将会在鸢尾花书《数学要素》一册展开讲解。

25.2 代数

本节举几个例子介绍如何用 SymPy 完成符号代数运算。

因式分解

代码 1 所示为利用 SymPy 完成因式分解。

a 从 `sympy` 导入 `symbols` 和 `factor`，其中 `symbols` 用来定义符号变量，`factor` 用来完成因式分解。

b 这两句的作用是将 SymPy 库中的数学符号以美观的形式打印出来。

c 定义了 x 和 y 两个符号变量。`symbols` 还可以定义带下角标的变量，比如 x_1 ， $x_2 = \text{symbols('x1 x2')}$ 。

也可以用 `from sympy.abc import x, y` 的形式定义符号变量。

此外，用 `sympy.symbols()` 定义变量时还可以提出符号的假设条件。

比如， $k = \text{sympy.symbols('k', integer=True)}$ 这一句定义符号变量 k ，并假定 k 为整数。

$z = \text{sympy.symbols('z', real=True)}$ 定义了符号变量 z ，并假定 z 为实数。

d 定义了 $x^2 - y^2$ 。

e 对 $x^2 - y^2$ 进行因式分解，结果为 $(x - y)(x + y)$ 。

反过来，可以用 `sympy.expand()` 展开 $(x - y)(x + y)$ ，结果为 $x^2 - y^2$ 。

```

a from sympy import symbols, factor
  # 从sympy中导入symbols, factor
b from sympy import init_printing
  init_printing("mathjax")

c x, y = symbols('x y')
  # 用sympy.symbols (简做symbols) 定义x和y两个符号变量
d f = x**2 - y**2
e f_factored= factor(f)

```

代码 1. 因式分解 |  Bk1_Ch25_01.ipynb

替换


代码 2 中 **a** 定义字符串。

- b** 将字符串转化为符号表达式 $x^3 + x^2 + x + 1$ 。
- c** 用符号 y 替代符号 x ，符号表达式变为 $y^3 + y^2 + y + 1$ 。
- d** 用 0 替代 x ，结果为 1。

```

from sympy import symbols, sympify
x, y = symbols('x y')
a str_expression = 'x**3 + x**2 + x + 1'
  # 将字符串转化为符号表达式
b str_2_sym = sympify(str_expression)
  # 将符号x替换为y
c str_2_sym.subs(x, y)
  # 将符号x替换为0
d str_2_sym.subs(x, 0)

```

代码 2. 用 sympy.sympify 将字符串转化为符号表达式 |  Bk1_Ch25_01.ipynb

特殊符号数值

SymPy 还可以定义特殊符号数值，表 1 给出几个例子。

比如，`sympy.sympify()` 将 2 转化为符号数值 2，然后进一步判断其是否为整数，是否为实数。

再比如，`from sympy import Rational; Rational(1, 2)` 这两句的结果为 $\frac{1}{2}$ 。

想要知道表格中结果的浮点数形式，可以用 `.evalf()`，比如 `exp(2).evalf()` 的结果为 7.38905609893065。

请大家在 JupyterLab 中练习表 1 给出的例子。

表 1. 用 sympy 定义特殊符号数值 |  Bk1_Ch25_01.ipynb

代码	结果
<code>from sympy import sympify sympify(2).is_integer sympify(2).is_real</code>	True True
<code>from sympy import Rational Rational(1, 2)</code>	$\frac{1}{2}$
<code>from sympy import sqrt 1 / (sqrt(2) + 1)</code>	$\frac{1}{1+\sqrt{2}}$
<code>from sympy import pi expr = pi ** 2</code>	π^2
<code>from sympy import exp exp(2)</code>	e^2
<code>from sympy import factorial factorial(5)</code>	5!
<code>from sympy import binomial binomial(5, 4)</code>	$C_5^4 = 5$
<code>from sympy import gamma gamma(5)</code>	$\Gamma(5) = (5-1)! = 4 \times 3 \times 2 \times 1 = 24$

区间

表 2 总结如何用 `sympy.Interval()` 定义各种区间，注意，默认区间左闭、右闭。`oo`（两个小写英文字母 o）代表正无穷。

注意，大家自己在同一个 Jupyter Notebook 练习时，`from sympy import Interval`，`oo` 只需要导入一次，不需要重复导入。

此外，用 `sympy.Interval()` 定义的区间还可以进行集合运算，比如 `Interval(0, 2) - Interval(0, 1)` 结果为 `(1, 2]`。再比如，`Interval(0, 1) + Interval(1, 2)` 的结果为 `[0, 2]`。

利用 `.has()` 还可以判断区间是否包含具体元素，比如先定义 `intv1 = Interval.Lopen(0, 1)`，得到区间 `(0, 1]`。

然后利用 `intv1.has(0)` 或 `intv1.contains(0)` 判断左开右闭区间是否包括元素 0，结果为 `False`。

表 2. 用 sympy.Interval() 定义区间 |  Bk1_Ch25_01.ipynb

代码	结果
<code>from sympy import Interval, oo Interval(0, 1, left_open=False, right_open=False)</code>	<code>[0, 1]</code>
<code>from sympy import Interval, oo Interval(0, 1, left_open=True, right_open=True)</code>	<code>(0, 0)</code>
<code>from sympy import Interval, oo Interval(0, 1, left_open=False, right_open=True) # Interval.Ropen(0, 1)</code>	<code>[0, 1)</code>
<code>from sympy import Interval, oo Interval(0, 1, left_open=True, right_open=False) # Interval.Lopen(0, 1)</code>	<code>(0, 1]</code>

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


<code>from sympy import Interval, oo</code> <code>Interval(0, oo, left_open=False, right_open=True)</code>	$[0, \infty)$
<code>from sympy import Interval, oo</code> <code>Interval(-oo, 0, left_open=True, right_open=True)</code>	$(-\infty, 0)$
<code>from sympy import Interval, S</code> <code>Interval(0, 1).complement(S.Reals)</code>	$(-\infty, 0) \cup (1, \infty)$

求解等式

代码 3 展示如何用 `sympy.solve()` 求解等式。

- a 定义等式 $x^2 = 1$ 。
- b 求解等式结果为 $[-1, 1]$ 。
- c 定义等式 $ax^2 + bx + c = 0$ 。
- d 求解等式结果为 $\left[\frac{-b - \sqrt{b^2 - 4ac}}{2a}, \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right]$ 。

```
from sympy import symbols, solve, Eq
x = symbols('x')
# 定义等式 x**2 = 1
a equation_1 = Eq(x**2, 1)
b solve(equation_1, x)
a, b, c = symbols("a, b, c", real=True)
# 定义等式 a*x**2 + b*x = -c
c equation_2 = Eq(a*x**2 + b*x + c, 0)
d solve(equation_2, x)
```

代码 3. 用 `sympy.solve()` 求解等式 |  Bk1_Ch25_01.ipynb

函数

图 1 所示为二元高斯函数 $f(x_1, x_2) = \exp(-x_1^2 - x_2^2)$ 曲面。

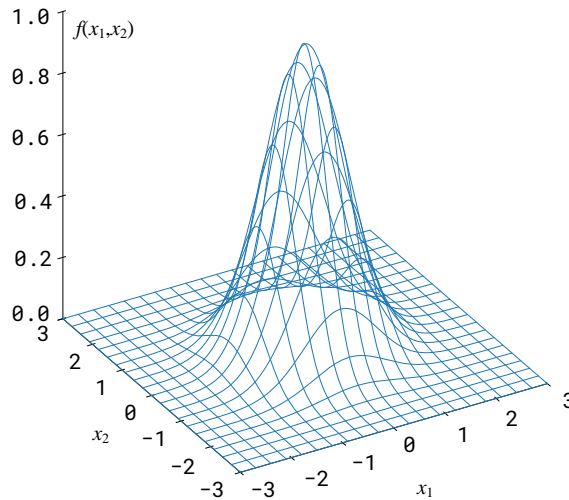


图 1. 二元高斯函数曲面

代码 4 绘制图 1，下面聊聊其中关键语句。

- a** 定义了符号函数 $\exp(-x_1^2 - x_2^2)$ 。
 - b** 用 `lambdify` 将符号函数 $\exp(-x_1^2 - x_2^2)$ 转化为 Python 函数，从而可以进行数值运算。其中，`[x1, x2]` 指定了符号变量。
 - c** 用 `plot_wireframe()` 在三维轴对象 `ax` 上绘制网格曲面来可视化二元高斯函数。
- 请大家自行分析代码 4 中剩余代码，并逐行注释。

```


from sympy import symbols, exp, lambdify
import numpy as np
import matplotlib.pyplot as plt

x1, x2 = symbols('x1 x2')
# 定义符号变量
a f_gaussian_x1x2 = exp(-x1**2 - x2**2)
# 将符号表达式转换为Python函数
b f_gaussian_x1x2_fcn = lambdify([x1, x2], f_gaussian_x1x2)
xx1, xx2 = np.meshgrid(np.linspace(-3, 3, 201),
                        np.linspace(-3, 3, 201))

ff = f_gaussian_x1x2_fcn(xx1, xx2)
# 可视化
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

c ax.plot_wireframe(xx1, xx2, ff,
                    rstride=10, cstride=10)
ax.set_proj_type('ortho')
ax.view_init(azim=-120, elev=30)
ax.grid(False)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('f(x1, x2)')
ax.set_xlim(-3, 3)
ax.set_ylim(-3, 3)
ax.set_zlim(0, 1)
ax.set_box_aspect(aspect = (1, 1, 1))
fig.savefig('二元高斯函数.svg', format='svg')

```

代码 4. 用 `sympy.lambdify()` 将符号表达式转化为 Python 函数 |  Bk1_Ch25_01.ipynb

25.3 线性代数

SymPy 也提供了一些线性代数工具，下面举例介绍。

矩阵定义和操作

代码 5 用 `sympy.Matrix()` 定义矩阵、列向量。

- a 从 `sympy` 导入 `Matrix` 函数。
- b 定义 2 行、3 列矩阵 A 。

函数 `sympy.shape()` 可以用来获取矩阵形状。举个例子，先用 `from sympy import shape` 导入 `shape`，然后 `shape(A)` 返回元组 $(2, 3)$ 即矩阵形状。 $A.T$ 可以完成矩阵转置。

对矩阵 A 的索引和切片方法和 NumPy 数组一致。

比如， $A[0, 0]$ 提取矩阵第 1 行、第 1 列元素。

$A[-1, -1]$ 提取矩阵最后一行、最后一列元素。

$A[0, :]$ 提取矩阵第一行, $A.\text{row}(0)$ 也可以用来提取矩阵第 1 行。

$A[:, 0]$ 提取矩阵第一列, $A.\text{col}(0)$ 也可以提取矩阵第一列。

此外, $A.\text{row_del}(0)$ 可以用来删除第 1 行元素。


$A.\text{row_insert}()$ 可以用来在特定位置插入行向量。

类似地, $A.\text{col_del}(0)$ 可以用来删除第 1 列元素。

$A.\text{col_insert}()$ 可以用来在特定位置插入列向量。

❷ 定义列向量 a 。

```
❶ from sympy import Matrix
# 定义矩阵
❷ A = Matrix([[1, 2, 3], [3, 2, 1]])
# 定义列向量
❸ a = Matrix([1, 2, 3])
```

代码 5. 用 `sympy.matrix()` 定义矩阵 |  Bk1_Ch25_02.ipynb

代码 6 定义的矩阵 A 为 $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ 。

```
❶ from sympy import Matrix, symbols
❷ A = Matrix(2, 2, symbols('a:d'))
```


代码 6. 用 `sympy.matrix()` 定义全符号矩阵 |  Bk1_Ch25_02.ipynb

表 3 给出了几种产生特殊矩阵的方法。

注意, `import numpy as np; np.array(A).astype(np.float64)` 可以把符号矩阵转化为 NumPy Array, 然后可以用 NumPy 函数进一步完成各种线性代数运算。

此外, $A.\text{is_symmetric}()$ 判断矩阵 A 是否为**对称阵** (symmetric matrix)。

$A.\text{is_diagonal}()$ 判断矩阵 A 是否为**对角阵** (diagonal matrix)。

$A.\text{is_lower}()$ 判断矩阵 A 是否为**下三角矩阵** (lower triangular matrix)。

$A.\text{is_upper}()$ 判断矩阵 A 是否为**上三角矩阵** (upper triangular matrix)。

$A.\text{is_square}()$ 判断矩阵 A 是否为**方阵** (square matrix)。

$A.\text{is_zero_matrix}()$ 判断矩阵 A 是否为**全 0 矩阵** (zero matrix)。

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>

本书配套微课程视频均发布在 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

`A.is_diagonalizable()` 判断矩阵 A 是否为可对角化矩阵 (diagonalizable matrix)。

`A.is_positive_definite()` 判断矩阵 A 是否为正定矩阵 (positive definite matrix)。

表 3. 用 sympy 函数产生特殊矩阵 |  Bk1_Ch25_02.ipynb

矩阵类型	代码	结果
单位矩阵	<pre>from sympy import eye A = eye(3)</pre>	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
全 0 矩阵	<pre>from sympy import zeros A = zeros(3, 3)</pre>	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
全 1 矩阵	<pre>from sympy import ones A = ones(3, 3)</pre>	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
对角方阵	<pre>from sympy import diag A = diag(1, 2, 3)</pre>	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$
上三角矩阵	<pre>from sympy import ones A = ones(3) A.upper_triangular()</pre>	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$
下三角矩阵	<pre>from sympy import ones A = ones(3) A.lower_triangular()</pre>	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

矩阵相关运算

代码 7 展示矩阵相关的常用运算。

- a 和 b 给出两种矩阵乘法运算符，建议大家使用 `@`，和 NumPy 矩阵乘法符号保持一致。
- c 和 d 给出两种矩阵逆运算符。
- e 将符号矩阵转化为浮点数 NumPy 数组。
- f 计算矩阵 Q 的逆，结果为 $\frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$ 。
- g 计算矩阵 Q 的行列式，结果为 $ad - bc$ 。
- h 计算矩阵 Q 的迹，结果为 $a + d$ 。

```

from sympy import Matrix, symbols

A = Matrix([[1, 3], [-2, 3]])
B = Matrix([[0, 3], [0, 7]])
A.T # 矩阵转置
A + B # 加法
A - B # 减法
3*A # 标量乘矩阵
A.multiply_elementwise(B) # 逐项积
a A * B # 矩阵乘法
b A @ B # 矩阵乘法

Matrix_2x2 = Matrix([[1.25, -0.75],
                     [-0.75, 1.25]])

c Matrix_2x2**(-1) # 矩阵逆
d Matrix_2x2.inv() # 矩阵逆
# 将符号矩阵转化为浮点数numpy数组
e np.array(Matrix_2x2).astype(np.float64)

a, b, c, d = symbols('a b c d')
Q = Matrix([[a, b],
            [c, d]])

f Q.inv() # 矩阵逆
g Q.det() # 行列式
h Q.trace() # 迹

```

代码 7. 用 sympy 中常见矩阵运算 | Bk1_Ch25_02.ipynb

正定性

正定性 (positive definiteness) 是线性代数、优化方法、机器学习重要的数学概念。表 4 用一组 2×2 矩阵 $A_{2 \times 2}$ 介绍正定性，下面具体来看。

矩阵 $A_{2 \times 2}$ 是**正定** (positive definite)，意味着 $f(x) = x^T @ A_{2 \times 2} @ x$ 是个开口朝上的抛物面，形状像是碗。除了 $(0, 0)$ ， $f(x) = x^T @ A_{2 \times 2} @ x$ 均大于 0 。 $(0, 0)$ 为最小值。

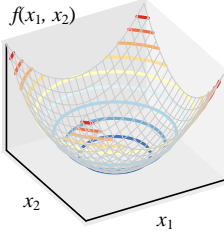
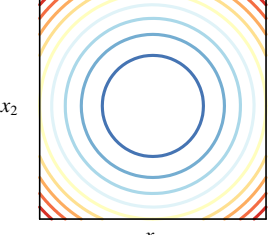
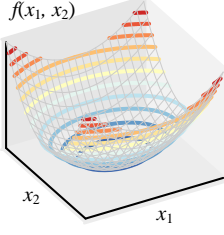
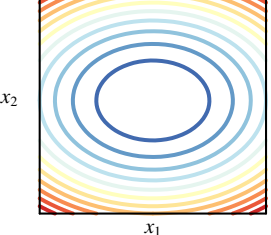
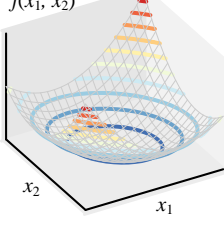
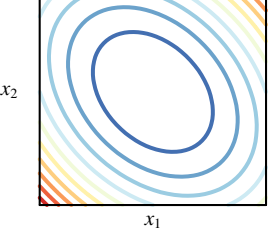
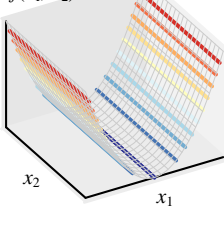
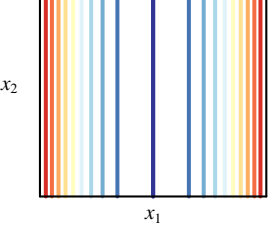
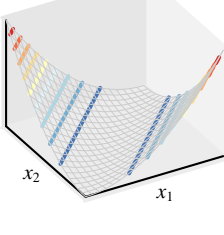
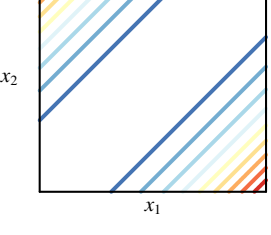
矩阵 $A_{2 \times 2}$ 是**半正定** (positive semi-definite)，意味着 $f(x) = x^T @ A_{2 \times 2} @ x$ 是个开口朝上的山谷面。除了 $(0, 0)$ ， $f(x) = x^T @ A_{2 \times 2} @ x$ 均大于等于 0 。山谷的谷底都是极小值。

矩阵 $A_{2 \times 2}$ 是**负定** (negative definite)，意味着 $f(x) = x^T @ A_{2 \times 2} @ x$ 是个开口朝下的抛物面。除了 $(0, 0)$ ， $f(x) = x^T @ A_{2 \times 2} @ x$ 均小于 0 。 $(0, 0)$ 为最大值。

矩阵 $A_{2 \times 2}$ 是**半负定** (negative semi-definite)，意味着 $f(x) = x^T @ A_{2 \times 2} @ x$ 是个开口朝下的山脊面。除了 $(0, 0)$ ， $f(x) = x^T @ A_{2 \times 2} @ x$ 均小于等于 0 。山脊的顶端都是极大值。

矩阵 $A_{2 \times 2}$ **不定** (indefinite)，意味着 $f(x) = x^T @ A_{2 \times 2} @ x$ 是个马鞍面， $(0, 0)$ 为鞍点。 $f(x) = x^T @ A_{2 \times 2} @ x$ 符号不定。

表 4. 几种 2×2 矩阵对应的不同正定性

正定性	矩阵 A 和函数	三维可视化	二维可视化
正定	$A = \begin{bmatrix} 1 & \\ & 1 \end{bmatrix}$ $f(x_1, x_2) = x_1^2 + x_2^2$		
正定	$A = \begin{bmatrix} 1 & \\ & 2 \end{bmatrix}$ $f(x_1, x_2) = x_1^2 + 2x_2^2$		
正定	$A = \begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 1.5 \end{bmatrix}$ $f(x_1, x_2) = 1.5x_1^2 + x_1x_2 + 1.5x_2^2$		
半正定	$A = \begin{bmatrix} 1 & \\ & 0 \end{bmatrix}$ $f(x_1, x_2) = x_1^2$		
半正定	$A = \begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{bmatrix}$ $f(x_1, x_2) = 0.5x_1^2 - x_1x_2 + 0.5x_2^2$		

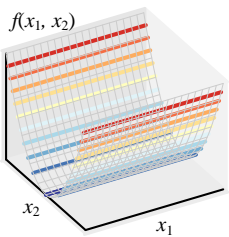
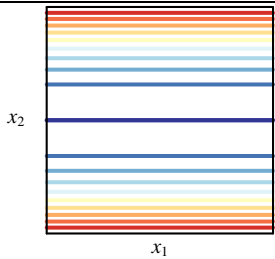
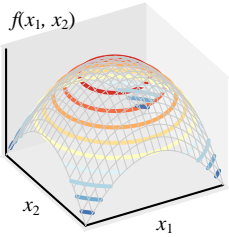
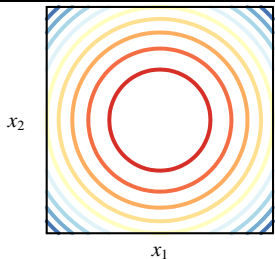
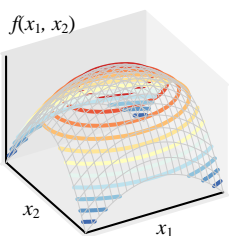
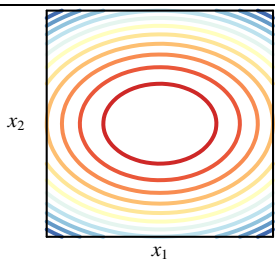
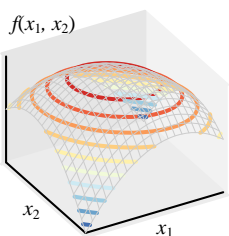
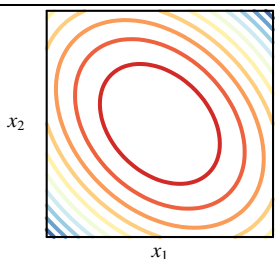
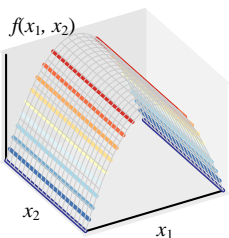
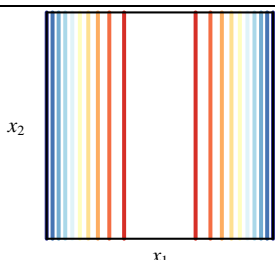
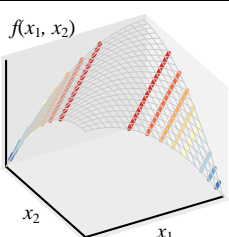
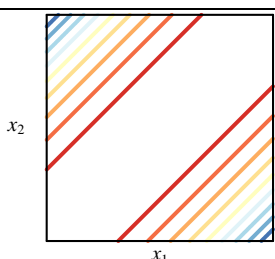
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

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

半正定	$A = \begin{bmatrix} 0 & \\ & 1 \end{bmatrix}$ $f(x_1, x_2) = x_2^2$		
负定	$A = \begin{bmatrix} -1 & \\ & -1 \end{bmatrix}$ $f(x_1, x_2) = -x_1^2 - x_2^2$		
负定	$A = \begin{bmatrix} -1 & \\ & -2 \end{bmatrix}$ $f(x_1, x_2) = -x_1^2 - 2x_2^2$		
负定	$A = \begin{bmatrix} -1.5 & -0.5 \\ -0.5 & -1.5 \end{bmatrix}$ $f(x_1, x_2) = -1.5x_1^2 - x_1x_2 - 1.5x_2^2$		
半负定	$A = \begin{bmatrix} -1 & \\ & 0 \end{bmatrix}$ $f(x_1, x_2) = -x_1^2$		
半负定	$A = \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix}$ $f(x_1, x_2) = -0.5x_1^2 + x_1x_2 - 0.5x_2^2$		

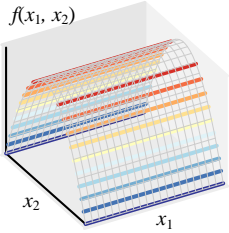
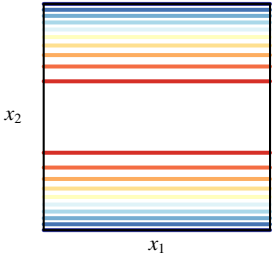
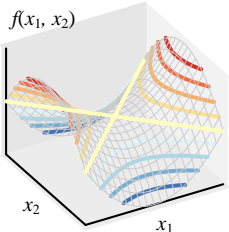
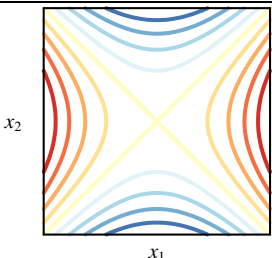
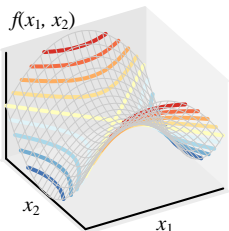
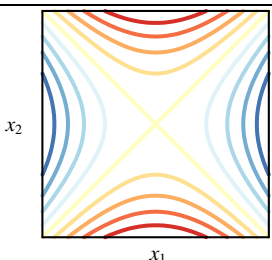
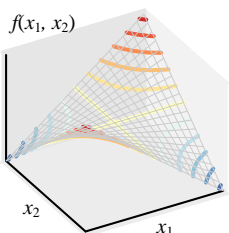
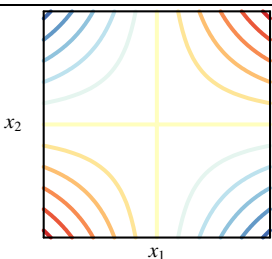
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

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

半负定	$A = \begin{bmatrix} 0 & \\ & -1 \end{bmatrix}$ $f(x_1, x_2) = -x_2^2$		
不定	$A = \begin{bmatrix} 1 & \\ & -1 \end{bmatrix}$ $f(x_1, x_2) = x_1^2 - x_2^2$		
不定	$A = \begin{bmatrix} -1 & \\ & 1 \end{bmatrix}$ $f(x_1, x_2) = -x_1^2 + x_2^2$		
不定	$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ $f(x_1, x_2) = 2x_1x_2$		

代码 8 和代码 9 绘制表 4 图像，下面讲解关键语句。

代码 8 首先自定义了一个可视化函数。

- a** 用 `def` 自定义函数 `visualize()`，这个函数有三个输入。
- b** 用 `matplotlib.pyplot.figure()`，简作 `plt.figure()`，创建了一个图形对象 `fig`。参数 `figsize=(6,3)` 代表图宽 6 英寸，图高 3 英寸。
- c** 用 `fig.add_subplot()` 在图形对象 `fig` 中添加一个子图，输出轴对象为 `ax_3D`。
参数 `(1, 2, 1)` 表示将图形划分为 1 行 2 列的子图布局，并选择第 1 个子图。参数第一个数字 1 表示行数，第二个数字 2 表示列数，第三个数字 1 表示选择的子图位置。
`projection='3d'` 表示使用 3D 投影坐标系，用于呈现三维可视化方案。
- d** 在三维轴对象 `ax_3D` 中用 `plot_wireframe()` 绘制网格图。
`xx1` 和 `xx2` 是网格数据，用于表示 `x` 和 `y` 坐标的网格点。
`f2_array` 包含了与网格点对应的二元函数坐标的数值。
`rstride` 和 `cstride` 分别表示行和列的步幅，控制网格之间的间隔，这里设置为 10。

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

`color=[0.8, 0.8, 0.8]` 指定了线框的颜色，这里是一个灰色。

`linewidth=0.25` 控制线框的线宽。

e 在三维轴对象 `ax_3D` 增加了第二个可视化方案——等高线。请大家自己解释函数输入。

f 用 `fig.add_subplot()` 在图形对象 `fig` 中添加第二个子图，子图位于右侧，默认为平面；输出轴对象为 `ax_2D`。

g 在平面轴对象 `ax_2D` 上绘制平面等高线。

```
# 导入包
import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, lambdify, expand, simplify

# 定义可视化函数
a def visualize(xx1, xx2, f2_array):

    b     fig = plt.figure(figsize=(6,3))
    c     # 左子图，三维
    c     ax_3D = fig.add_subplot(1, 2, 1, projection='3d')

    d     ax_3D.plot_wireframe(xx1, xx2, f2_array,
                             rstride=10, cstride=10,
                             color = [0.8,0.8,0.8],
                             linewidth = 0.25)

    e     ax_3D.contour(xx1, xx2, f2_array,
                      levels = 12, cmap = 'RdYlBu_r')

    ax_3D.set_xlabel('$x_1$'); ax_3D.set_ylabel('$x_2$')
    ax_3D.set_zlabel('$f(x_1,x_2)$')
    ax_3D.set_proj_type('ortho')
    ax_3D.set_xticks([]); ax_3D.set_yticks([])
    ax_3D.set_zticks([])
    ax_3D.view_init(azim=-120, elev=30)
    ax_3D.grid(False)
    ax_3D.set_xlim(xx1.min(), xx1.max());
    ax_3D.set_ylim(xx2.min(), xx2.max())

    # 右子图，平面等高线
    f     ax_2D = fig.add_subplot(1, 2, 2)
    g     ax_2D.contour(xx1, xx2, f2_array,
                      levels = 12, cmap = 'RdYlBu_r')

    ax_2D.set_xlabel('$x_1$'); ax_2D.set_ylabel('$x_2$')
    ax_2D.set_xticks([]); ax_2D.set_yticks([])
    ax_2D.set_aspect('equal'); ax_2D.grid(False)
    ax_2D.set_xlim(xx1.min(), xx1.max());
    ax_2D.set_ylim(xx2.min(), xx2.max())
    plt.tight_layout()
```

代码 8. 定义可视化函数 |  Bk1_Ch25_03.ipynb

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

代码 9 ^a 利用 `numpy.meshgrid()` 生成网格化数据，代表横纵轴坐标点。

^b 自定义函数，用来更方便地生成表 4 中不同 $f(\mathbf{x}) = \mathbf{x}^T @ \mathbf{A}_{2 \times 2} @ \mathbf{x}$ 。

^c 定义符号变量 `x1` 和 `x2`，分别代表 x_1 和 x_2 。

^d 相当于构造了符号列向量 $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ 。

^e 计算 $\mathbf{x}^T @ \mathbf{A}_{2 \times 2} @ \mathbf{x}$ ，虽然只有一个元素，但是结果为二维数组。

^f 从上述结果中提取符号表达式。

^g 打印 $\mathbf{x}^T @ \mathbf{A}_{2 \times 2} @ \mathbf{x}$ 解析式。

^h 用 `math.lambdify()` 将符号解析式转化为 Python 函数。

ⁱ 计算给定网格坐标下的二元函数 $f(\mathbf{x}) = \mathbf{x}^T @ \mathbf{A}_{2 \times 2} @ \mathbf{x}$ 值，结果也是二维数组。

^j 举了一个例子。

^k 调用自定义函数 `fcx()` 计算函数值。

^l 调用自定义函数 `visualize()` 可视化二元函数。

请大家在 JupyterLab 中练习计算并可视化表 4 所有示例。

```
# 生成数据
x1_array = np.linspace(-2,2,201)
x2_array = np.linspace(-2,2,201)

a xx1, xx2 = np.meshgrid(x1_array, x2_array)

# 定义二元函数

b def fcn(A, xx1, xx2):


c     x1,x2 = symbols('x1 x2')
d     x = np.array([[x1,x2]]).T
e     f_x = x.T@A@x
f     f_x = f_x[0][0]
g     print(simplify(expand(f_x)))

h     f_x_fcn = lambdify([x1,x2], f_x)
i     ff_x = f_x_fcn(xx1,xx2)

    return ff_x

# 不定矩阵
j A = np.array([[0, 1],
                [1, 0]])

k f2_array = fcn(A, xx1, xx2)
l visualize(xx1,xx2,f2_array)
```

代码 9. 可视化正定性，使用时配合前文代码 |  Bk1_Ch25_03.ipynb

矩阵分解

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

代码 10 完成符号矩阵 $A = \begin{bmatrix} a^2 & 2abc \\ 2abc & b^2 \end{bmatrix}$ 的特征值和特征向量，请在 JupyterLab 查看结果。


```
from sympy import Matrix, symbols
a, b, c, d = symbols('a b c d')
A = Matrix([[a**2, 2*a*b*c],
            [2*a*b*c, b**2]])
# 特征值
a A.eigenvals()
# 特征向量
b A.eigenvects()
```

代码 10. 用 sympy 完成符号矩阵的特征值分解 |  Bk1_Ch25_02.ipynb

代码 11 完成矩阵 $A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}$ 的奇异值分解。 U 的结果为 $U = \begin{bmatrix} \sqrt{2}/2 & \sqrt{6}/6 \\ 0 & \sqrt{6}/3 \\ -\sqrt{2}/2 & \sqrt{6}/6 \end{bmatrix}$ ， S 的结果为 $S = \begin{bmatrix} 1 & \\ & \sqrt{3} \end{bmatrix}$ ， V 的结果为 $V = \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$ 。

? 请大家分别计算 $V.T @ V$ ， $V @ V.T$ ， $U.T @ U$ ，并说明结果特点。

```
from sympy import Matrix
A = Matrix([[0, 1], [1, 1], [1, 0]])
# 奇异值分解
a U, S, V = A.singular_value_decomposition()
```

代码 11. 用 sympy 完成矩阵的奇异值分解 |  Bk1_Ch25_02.ipynb

本章最后从数据和几何角度再聊聊矩阵乘法规则。

再聊聊矩阵乘法规则：尺寸匹配是前提

矩阵乘法 $A @ B$ 的前提条件是，左侧矩阵 A 的列数必须等于右侧矩阵 B 的行数。如图 2 所示，如果矩阵乘法 $A @ B$ 成立，不代表 $B @ A$ 成立。

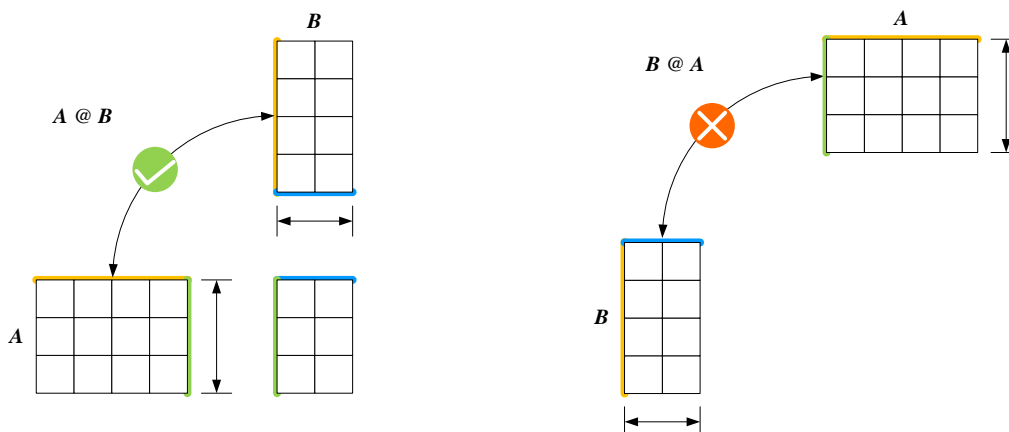
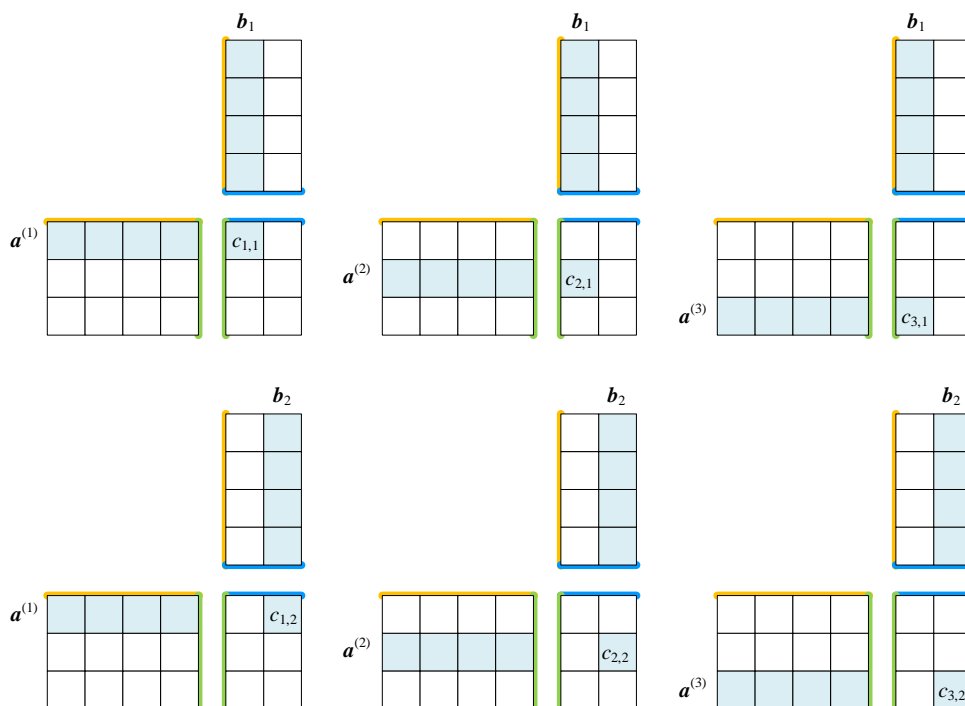
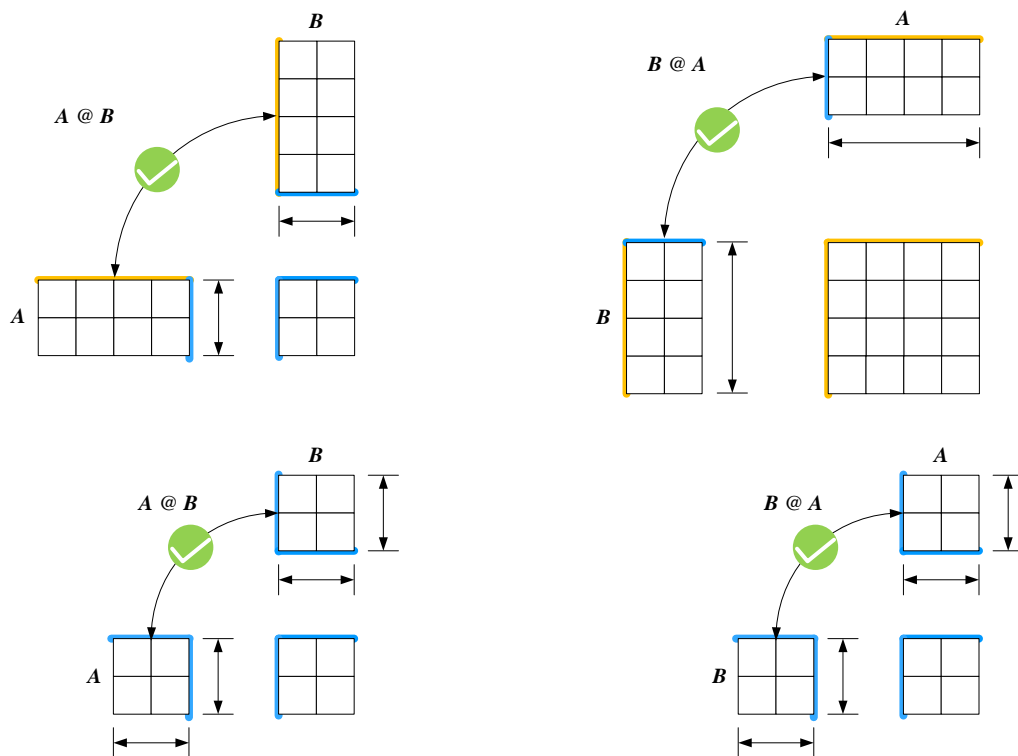
图 2. 矩阵乘法 $A @ B$ 成立, $B @ A$ 不成立

图 3 展示了如何获得矩阵乘法 $A @ B$ 的每一个元素。从这幅图中, 我们可以更清楚知道为什么要求 A 的列数必须等于 B 的行数。

图 4 告诉我们, 即便有些情况矩阵乘法 $A @ B$ 成立, $B @ A$ 也成立; 但是, 通常 $A @ B \neq B @ A$ 。

? 请大家思考, 什么条件下 $A @ B = B @ A$? 鸢尾花书《矩阵力量》会给出答案。

图 3. 如何获得矩阵乘法 $A @ B$ 的每个元素

图 4. 矩阵乘法 $A @ B$ 成立, $B @ A$ 成立; 但是, 一般情况, $A @ B \neq B @ A$

几何视角看矩阵乘法

下面我们从几何角度举几个例子和大家简单聊聊矩阵乘法和矩阵的逆。

如图 5 所示, 图中矩阵 A 完成的是缩放, 逆矩阵 A^{-1} 则相当于这个缩放的逆操作。

类似地, 图 6 中 A 完成平面旋转, A^{-1} 则向反方向旋转, 将图形恢复原貌。

请大家用 SymPy 在 JupyterLab 中计算图 5 和图 6 这两个矩阵的逆。

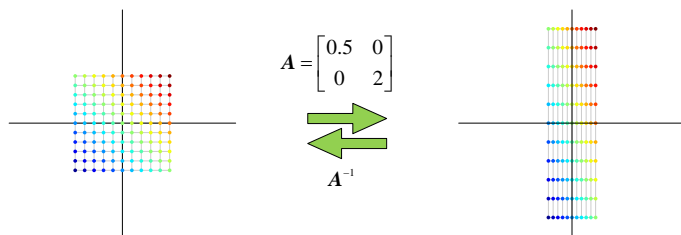


图 5. 平面缩放

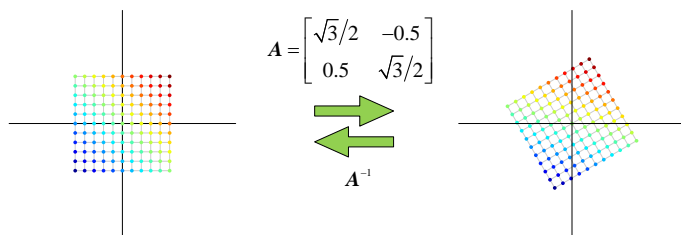


图 6. 平面旋转

在三维空间，矩阵也可以用来完成各种几何变换。

比如图 7，矩阵 A 完成三维空间缩放，每个轴方向的缩放比例显然不一致，和矩阵对角线元素有关。

图 8 则完成三维空间的旋转。这两种情况的逆矩阵也是完成反向几何操作。

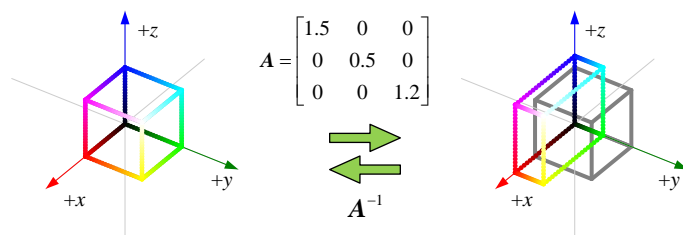


图 7. 三维缩放

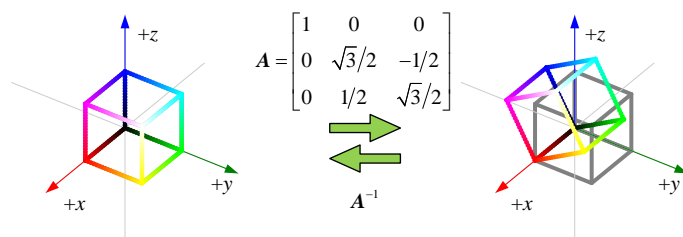


图 8. 三维旋转

请大家利用 SymPy 自行计算中图 5 ~ 图 8 逆矩阵具体值。

简单来说，如果逆矩阵不存在说明几何变换不可逆。

图 9 中矩阵 A 将平面图形“拍扁”成直线，数据信息发生丢失。这个几何操作叫做投影 (projection)。单从图 9 右图来看，我们不能将其恢复成左图。

类似地，图 10 中矩阵 A 将三维图形拍扁成平面。请大家试着用 SymPy 计算图 9 和图 10 两个矩阵的逆，看看是否会报错。

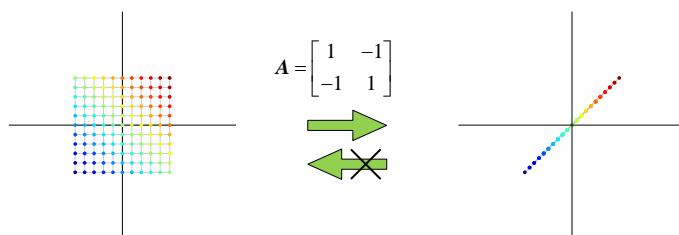


图 9. 平面投影，不可逆

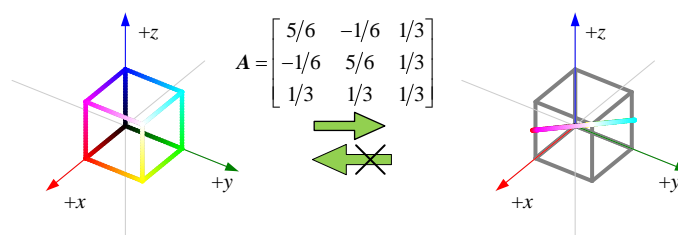


图 10. 三维投影，不可逆



请大家完成如下题目。

Q1. 在 JupyterLab 复刻本章所有代码和结果。

* 题目很基础，本书不给答案。



对用 SymPy 求解微积分问题感兴趣的读者可以参考。

<https://docs.sympy.org/latest/tutorials/intro-tutorial/calculus.html>



请大家注意，SymPy 目前很多功能还不够完善。大家想要处理更为复杂的符号运算，建议使用 Mathematica 或 MATLAB Symbolic Math Toolbox。

本章最后这些几何变换（旋转、缩放、投影）用途极为广泛，比如计算机视觉、机器人运动。在鸢尾花书中，大家会发现这些几何变换还帮助我们理解特征值分解、奇异值分解、随机数模拟、协方差矩阵、多元高斯分布、主成分分析等等。

看似枯燥无味、单调呆板的矩阵乘法实际上多姿多彩、妙趣横生。《矩阵力量》一册将为大家展现一个生机勃勃的矩阵乘法世界。