Symbolic Computation in SymPy

25 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

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

因式分解

图 1 所示为利用 SymPy 完成因式分解。

ⓐ 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。ⓑ 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。

ⓒ 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 x1, x2 = 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 = sympy.symbols('k', integer=True) 这一句定义符号变量 k，并假定 k 为整数。z = sympy.symbols('z', real=True) 定义了符号变量 z，并假定 z 为实数。

ⓓ 定义了 $x^2 - y^2$。ⓔ 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 $x^2 - y^2$。

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

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

图 1. 因式分解

替换

图 2 中 ⓐ 定义字符串，ⓑ 将字符串转化为符号表达式 x^3+x^2+x+1。

ⓒ 用符号 y 替代符号 x，符号表达式变为 y^3+y^2+y+1。

ⓓ 用 0 替代 x，结果为 1。

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

图 2. 用 sympy.sympify 将字符串转化为符号表达式

特殊符号数值

SymPy 还可以定义定义特殊符号数值，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。

表 1. 用 sympy 定义特殊符号数值

代码	结果

`from sympy import sympify`
`sympify(2).is_integer`
`sympify(2).is_real`
	True
	True

`from sympy import Rational` `Rational(1, 2)`	$\frac{1}{2}$
`from sympy import sqrt` `1 / (sqrt(2) + 1)`	$\frac{1}{1+\sqrt{2}}$
`from sympy import pi` `expr = pi ** 2`	π^2
`from sympy import exp` `exp(2)`	e^2
`from sympy import factorial` `factorial(5)`	5!
`from sympy import binomial` `binomial(5, 4)`	$C_5^4=5$
`from sympy import gamma` `gamma(5)`	$\Gamma(5)=(5-1)!=4\times3\times2\times1=24$

区间

表 2 总结如何用 sympy.Interval() 定义各种区间，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 intvl = Interval.Lopen(0, 1)，得到区间 (0, 1]。然后利用 intvl.has(0) 或 intvl.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

代码	结果
`from sympy import Interval, oo` `Interval(0, 1, left_open=False, right_open=False)`	[0, 1]
`from sympy import Interval, oo` `Interval(0, 1, left_open=True, right_open=True)`	(0, 0)

`from sympy import Interval, oo`
`Interval(0, 1, left_open=False, right_open=True)`
`# Interval.Ropen(0, 1)`
	[0, 1)

`from sympy import Interval, oo`
`Interval(0, 1, left_open=True, right_open=False)`
`# Interval.Lopen(0, 1)`
	(0, 1]

`from sympy import Interval, oo` `Interval(0, oo, left_open=False, right_open=True)`	[0, ∞)
`from sympy import Interval, oo` `Interval(-oo, 0, left_open=True, right_open=True)`	(−∞, 0)
`from sympy import Interval, S` `Interval(0, 1).complement(S.Reals)`	(−∞, 0) ∪ (1, ∞)

求解等式

图 3 代码介绍如何用 sympy.solve() 求解等式。ⓐ 定义等式 $x^2=1$，ⓑ 求解等式结果为 $[-1,1]$。

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

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

图 3. 用 sympy.solve() 求解等式

函数

图 4 代码用 sympy.lambdify() 将符号函数 $\exp\left(-x_1^2-x_2^2\right)$ 转化为 Python 函数，从而可以进行数值运算。图 5 所示为代码绘制的二元高斯函数曲面。

```
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
f_gaussian_x1x2 = exp(-x1**2 - x2**2)
# 将符号表达式转换为Python函数
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')

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')
```

图 4. 用 sympy.lambdify() 将符号表达式转化为 Python 函数

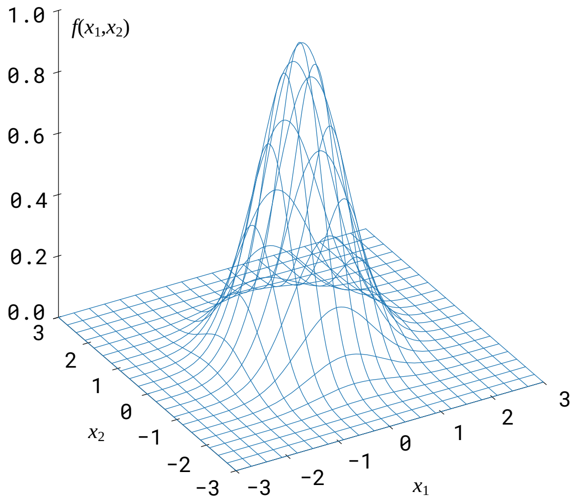


图 5. 二元高斯函数曲面

25.3 线性代数

NumPy 是 Python 科学计算中非常重要的一个库，它提供了快速、高效的多维数组对象及其操作方法，是众多其他科学计算库的基础。

矩阵

图 6 中代码用 sympy.Matrix() 定义矩阵、列向量。

ⓐ 从 sympy 导入 Matrix 函数。

ⓑ 定义 2 行、3 列矩阵 $\boldsymbol{A}$。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 $\boldsymbol{A}$ 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***.row(0) 也可以用来提取矩阵第 1 行。***A***[:,0] 提取矩阵第一列，***A***.col(0) 也可以提取矩阵第一列。

此外，***A***.row_del(0) 可以用来删除第 1 行元素。***A***.row_insert() 可以用来在特定位置插入行向量。类似地，***A***.col_del(0) 可以用来删除第 1 列元素。***A***.col_insert() 可以用来在特定位置插入列向量。

ⓒ 定义列向量 $\boldsymbol{a}$。

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

图 6. 用 sympy.matrix() 定义矩阵

图 7 定义的矩阵 $\boldsymbol{A}$ 为 $\boldsymbol{A}=\begin{bmatrix} a & b \\ c & d \end{bmatrix}$。

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

b

图 7. 用 sympy.matrix() 定义全符号矩阵

表 3 给出了几种产生特殊矩阵的方法。此外，$\boldsymbol{A}$.is_symmetric() 判断矩阵 $\boldsymbol{A}$ 是否为对称阵，$\boldsymbol{A}$.is_diagonal() 判断矩阵 $\boldsymbol{A}$ 是否为对角阵，$\boldsymbol{A}$.is_lower 判断矩阵 $\boldsymbol{A}$ 是否为下三角，$\boldsymbol{A}$.is_upper 判断矩阵 $\boldsymbol{A}$ 是否为上三角纠正，$\boldsymbol{A}$.is_square 判断矩阵 $\boldsymbol{A}$ 是否为方阵，$\boldsymbol{A}$.is_zero_matrix 判断矩阵 $\boldsymbol{A}$ 是否为全 0 矩阵，$\boldsymbol{A}$.is_diagonalizable() 判断矩阵 $\boldsymbol{A}$ 是否可以对角化。$\boldsymbol{A}$.is_positive_definite 判断矩阵 $\boldsymbol{A}$ 是否为正定。

表 3. 用 sympy 函数产生特殊矩阵

矩阵类型	代码	结果
单位矩阵	`from sympy import eye` `A = eye(3)`	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
全 0 矩阵	`from sympy import zeros` `A = zeros(3, 3)`	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
全 1 矩阵	`from sympy import ones` `A = ones(3, 3)`	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
对角方阵	`from sympy import diag` `A = diag(1, 2, 3)`	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

上三角矩阵	`from sympy import ones`
	`A = ones(3)`
	`A.upper_triangular()`
		$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

下三角矩阵	`from sympy import ones`
	`A = ones(3)`
	`A.lower_triangular()`
		$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

运算

图 8 代码给出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。

ⓒ和ⓓ给出两种矩阵逆运算符。

ⓔ将符号矩阵转化为浮点数 NumPy 数组。

ⓕ计算矩阵 $\boldsymbol{Q}$ 的逆，结果为 $\dfrac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$。

ⓖ计算矩阵 $\boldsymbol{Q}$ 的行列式，结果为 $ad-bc$。

ⓗ计算矩阵 $\boldsymbol{Q}$ 的迹，结果为 $a+d$。

```python
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 * B # 矩阵乘法
A @ B # 矩阵乘法

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

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

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

Q.inv()    # 矩阵逆
Q.det()    # 行列式
Q.trace() # 迹
```

图 8. 用 sympy 中常见矩阵运算

正定性

正定性是线性代数、优化方法、机器学习重要的数学概念。下面我们用一个 2×2 矩阵 $\boldsymbol{A}_{2\times 2}$ 介绍正定性。

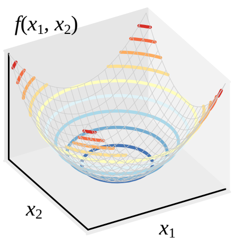
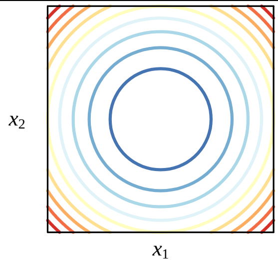
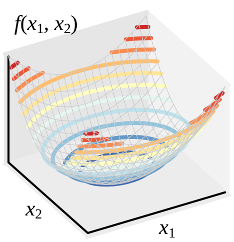
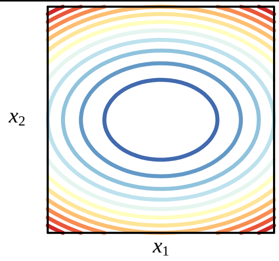
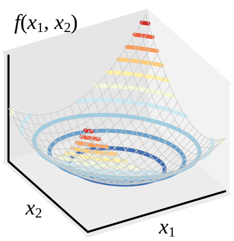
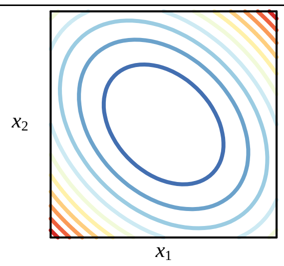
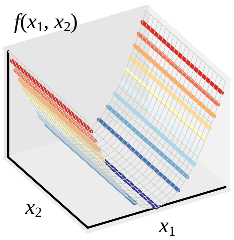
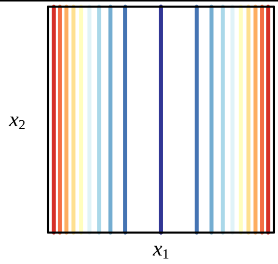
矩阵 $\boldsymbol{A}_{2\times 2}$ 是正定，意味着 $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times 2}$ @ $\boldsymbol{x}$ 是个开口朝上的抛物面，形状像是碗。除了 (0, 0)，$f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times 2}$ @ $\boldsymbol{x}$ 均大于 0。(0, 0) 为最小值，图中箭头都背离 (0, 0)。

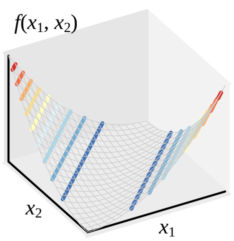
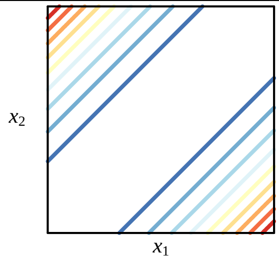
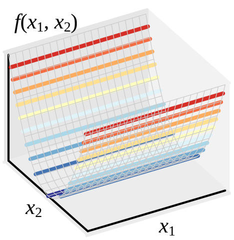
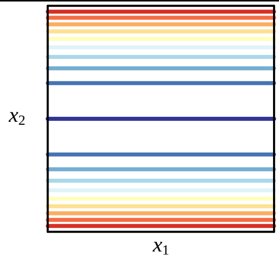
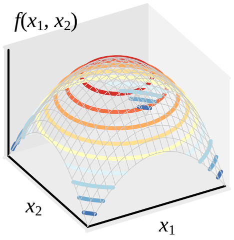
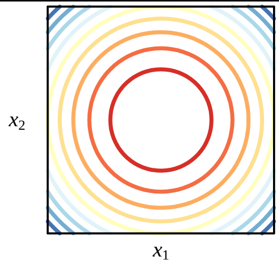
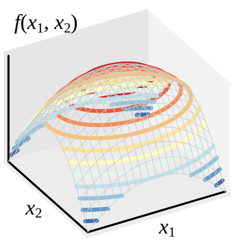
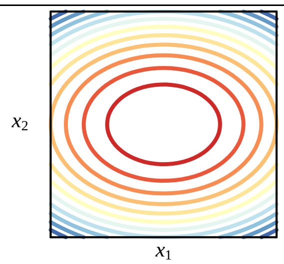
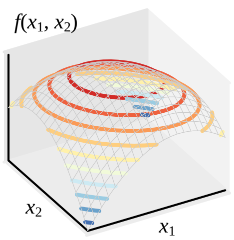
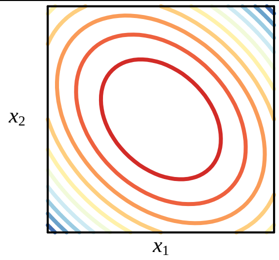
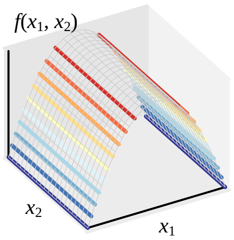
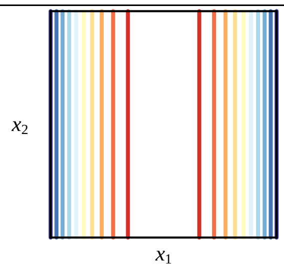
矩阵 $\boldsymbol{A}_{2\times 2}$ 是半正定，意味着 $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times 2}$ @ $\boldsymbol{x}$ 是个开口朝上的山谷面。除了 (0, 0)，$f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times 2}$ @ $\boldsymbol{x}$ 均大于等于 0。山谷的谷底都是极小值，图中箭头都背离谷底所在直线。

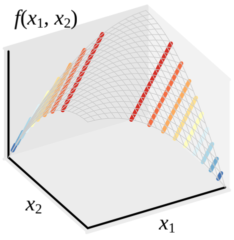
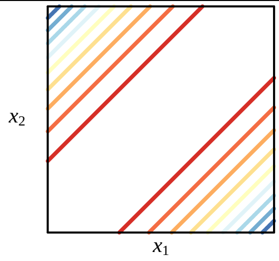
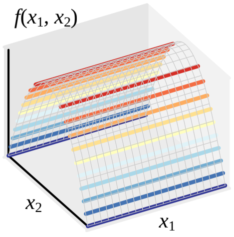
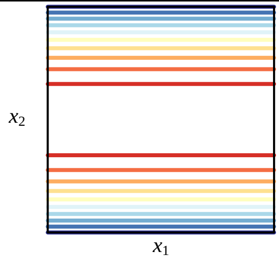
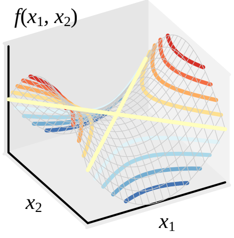
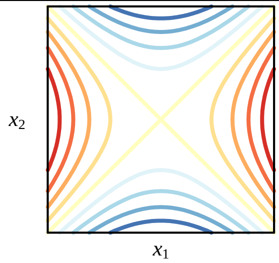
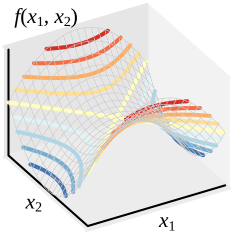
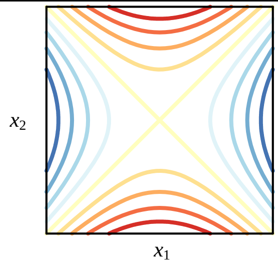
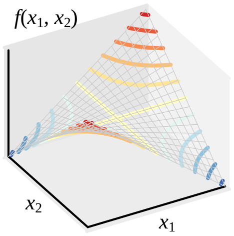
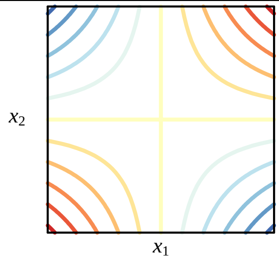
矩阵 $\boldsymbol{A}_{2\times2}$ 是负定，意味着 $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times2}$ @ $\boldsymbol{x}$ 是个开口朝下的抛物面。除了 (0, 0)，$f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times2}$ @ $\boldsymbol{x}$ 均小于 0。(0, 0) 为最大值，图中箭头都指向 (0, 0)。

矩阵 $\boldsymbol{A}_{2\times2}$ 是半负定，意味着 $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times2}$ @ $\boldsymbol{x}$ 是个开口朝下的山脊面。除了 (0, 0)，$f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times2}$ @ $\boldsymbol{x}$ 均小于等于 0。山脊的顶端都是极大值，图中箭头指向山脊顶端所在直线。

矩阵 $\boldsymbol{A}_{2\times2}$ 不定，意味着 $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times2}$ @ $\boldsymbol{x}$ 是个马鞍面，(0, 0) 为鞍点。$f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}$ @ $\boldsymbol{A}_{2\times2}$ @ $\boldsymbol{x}$ 符号不定。图中有些箭头背离 (0, 0)，有些指向 (0, 0)。

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

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

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

```
# 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))
      # 左子图，三维
c     ax = fig.add_subplot(1, 2, 1, projection='3d')

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

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

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

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

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

图 9. 定义可视化函数

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

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

# 定义二元函数

def fcn(A, xx1, xx2):

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

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

    return ff_x

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

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

图 10. 可视化正定性，使用时配合前文代码

矩阵分解

图 11 完成符号矩阵 $\boldsymbol{A}=\begin{bmatrix} a^2 & 2abc \\ 2abc & b^2 \end{bmatrix}$ 的特征值和特征向量。

```
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.eigenvals()
# 特征向量
A.eigenvects()
```

图 11. 用 sympy 完成符号矩阵的特征值分解

图 12 完成矩阵 $\boldsymbol{A}=\begin{bmatrix}0 & 1\\ 1 & 1\\ 1 & 0\end{bmatrix}$ 的奇异值分解。$\boldsymbol{U}$ 的结果为 $\boldsymbol{U}=\begin{bmatrix}\sqrt{2}/2 & \sqrt{6}/6\\ 0 & \sqrt{6}/3\\ -\sqrt{2}/2 & \sqrt{6}/6\end{bmatrix}$，$\boldsymbol{S}$ 的结果为 $\boldsymbol{S}=\begin{bmatrix}1 & \\ & \sqrt{3}\end{bmatrix}$，$\boldsymbol{V}$ 的结果为 $\boldsymbol{V}=\begin{bmatrix}-\sqrt{2}/2 & \sqrt{2}/2\\ \sqrt{2}/2 & \sqrt{2}/2\end{bmatrix}$。请大家分别计算 $\boldsymbol{V}$.T @ $\boldsymbol{V}$，$\boldsymbol{V}$ @ $\boldsymbol{V}$.T，$\boldsymbol{U}$.T @ $\boldsymbol{U}$。

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

图 12. 用 sympy 完成矩阵的奇异值分解

再聊聊矩阵乘法规则：尺寸匹配是前提

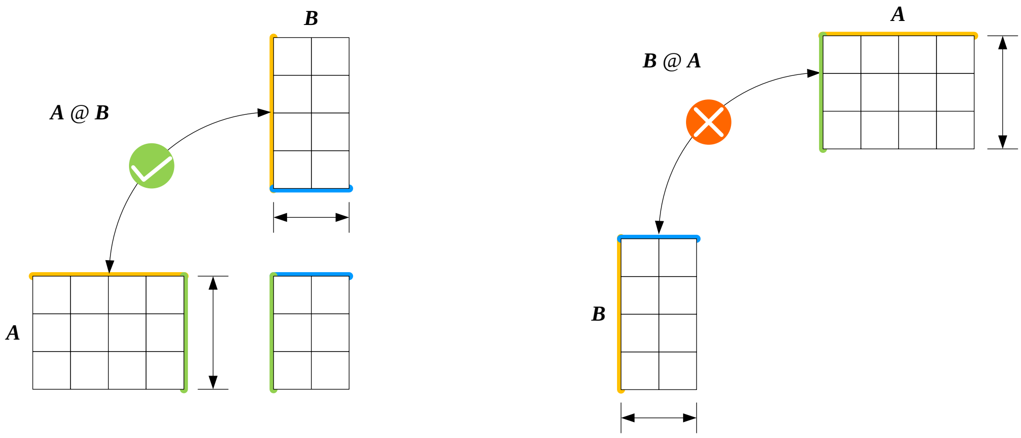


图 13. 矩阵乘法 $\boldsymbol{A}$ @ $\boldsymbol{B}$ 成立，$\boldsymbol{B}$ @ $\boldsymbol{A}$ 不成立

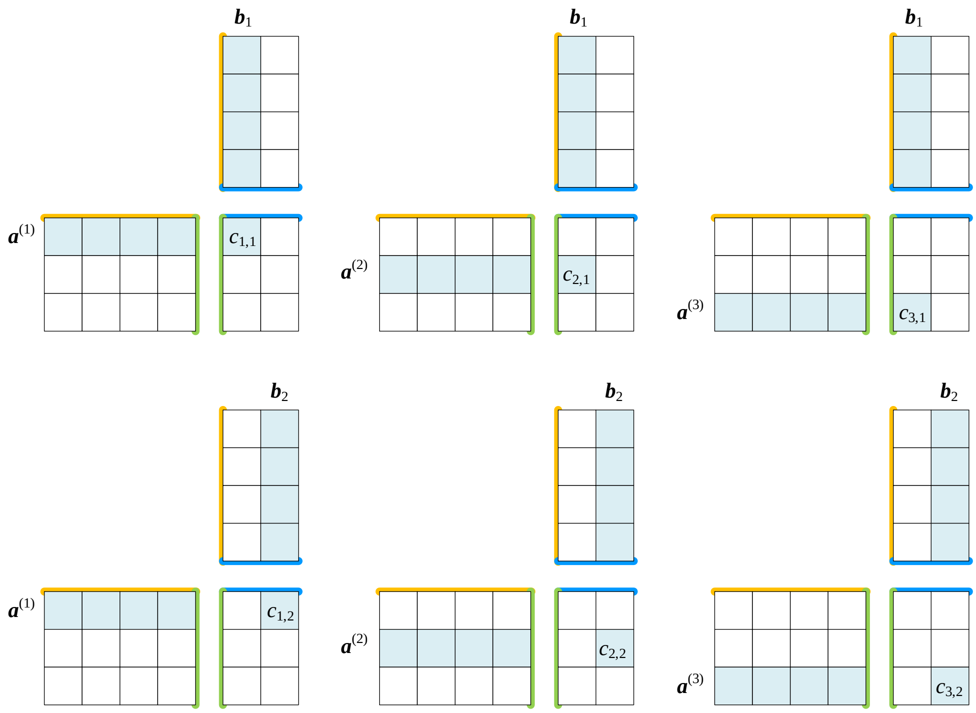


图 14. 如何获得矩阵乘法 A @ B 的每个元素

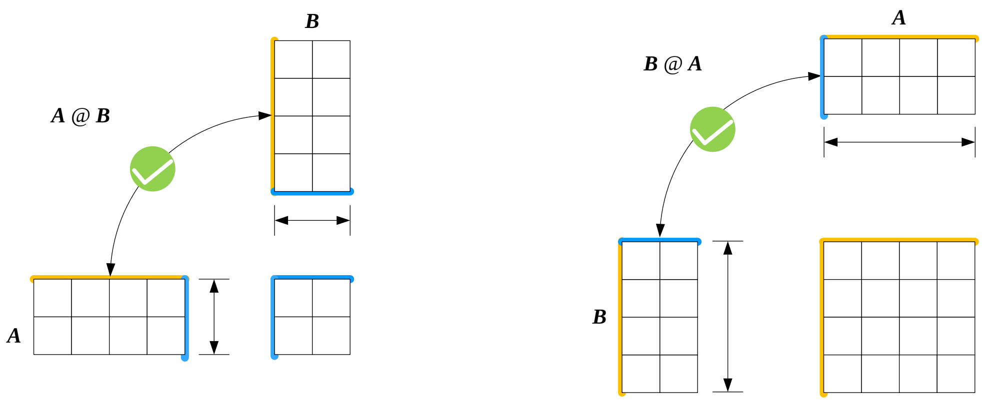


图 15. 矩阵乘法 A @ B 成立，B @ A 成立；但是，一般情况，A @ B ≠ B @ A

几何视角看矩阵乘法

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。本章最后从几何角度举几个例子和大家简单聊聊矩阵乘法。

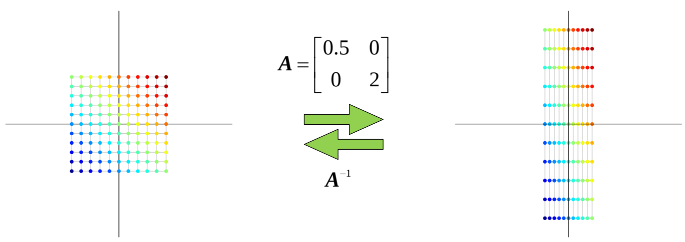


图 16. 平面缩放

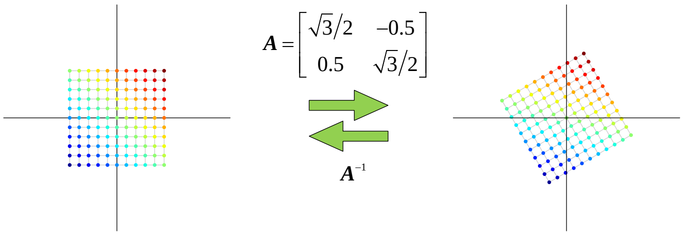


图 17. 平面旋转

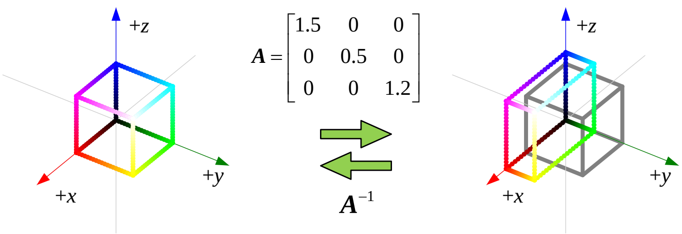


图 18. 三维缩放

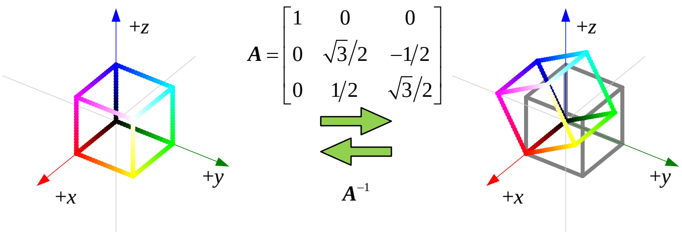


图 19. 三维旋转

请大家利用 SymPy 自行计算中图 16 ~ 图 19 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。请大家试着用 SymPy 计算图 20 和图 21 两个矩阵的逆，看看是否会报错。

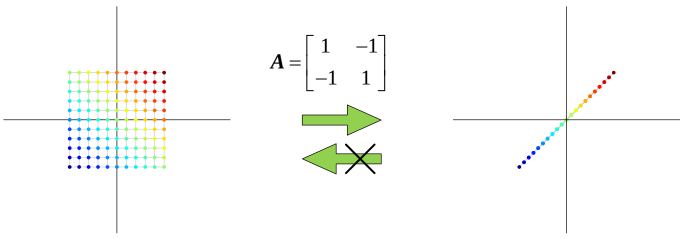


图 20. 平面投影，不可逆

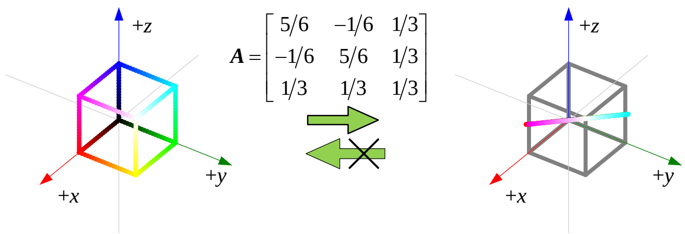


图 21. 三维投影，不可逆

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。

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。