4 Matrix 矩阵

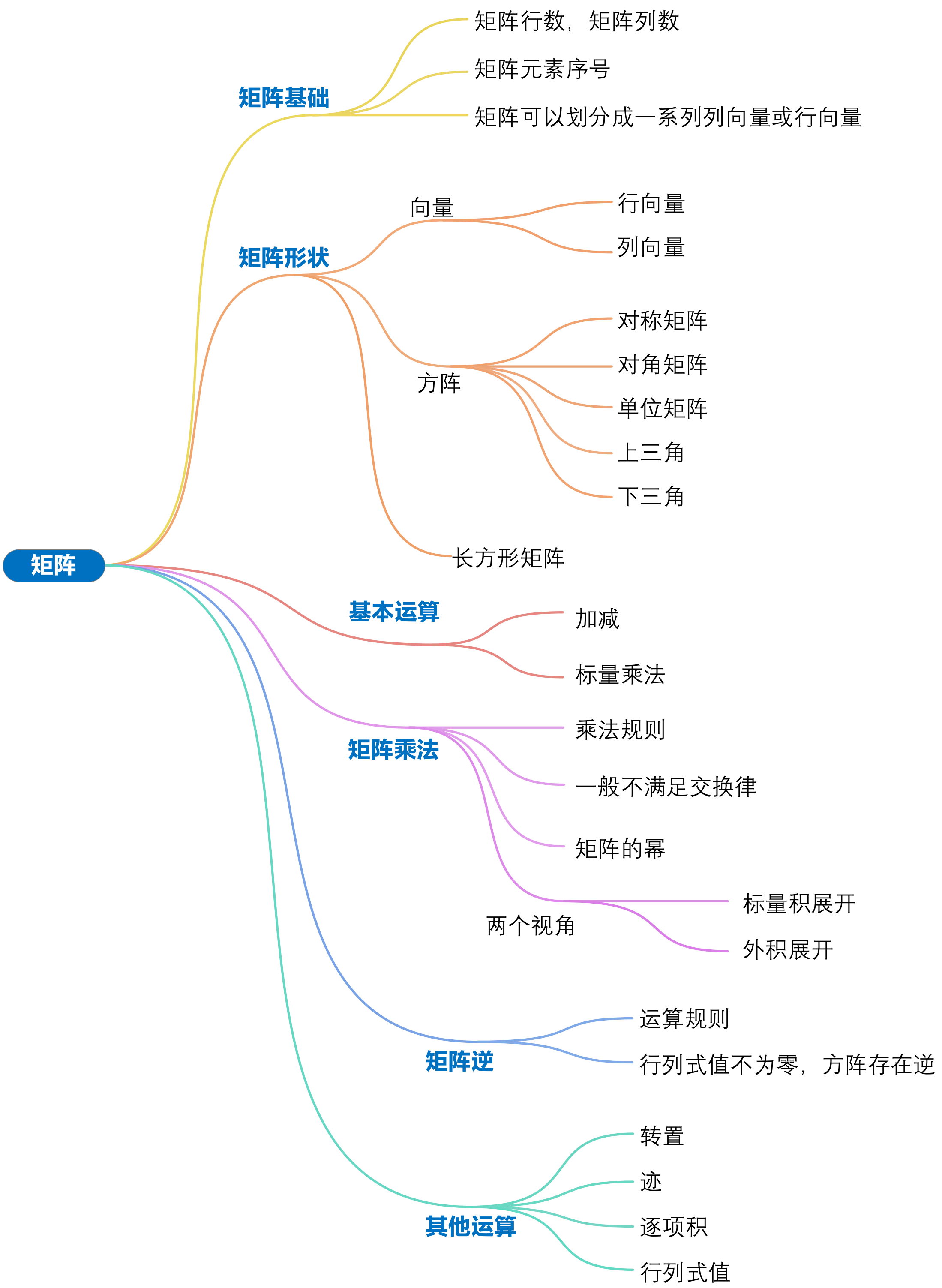
所有矩阵运算都是重要数学工具，都有应用场景

数字统治万物。

Number rules the universe.

—— 毕达哥拉斯 (Pythagoras) | 古希腊哲学家、数学家 | 570 ~ 495 BC

- numpy.add() 矩阵加法运算，等同于 +
- numpy.array() 构造多维矩阵/数组
- numpy.linalg.det() 计算行列式值
- numpy.linalg.inv() 计算矩阵逆
- numpy.linalg.matrix_power() 计算矩阵幂
- numpy.matrix() 构造二维矩阵
- numpy.multiply() 矩阵逐项积
- numpy.ones() 生成全 1 矩阵，输入为矩阵形状
- numpy.ones_like() 用来生成和输入矩阵形状相同的全 1 矩阵
- numpy.subtract() 矩阵减法运算，等同于 -
- numpy.trace() 计算矩阵迹
- numpy.zeros() 生成零矩阵，输入为矩阵形状
- numpy.zeros_like() 用来生成和输入矩阵形状相同的零矩阵
- tranpose() 矩阵转置，比如 A.transpose()，等同于 A.T


矩阵
矩阵基础
矩阵行数，矩阵列数
矩阵元素序号
矩阵可以划分成一系列列向量或行向量
矩阵形状
向量
行向量
列向量
方阵
对称矩阵
对角矩阵
单位矩阵
上三角
下三角
长方形矩阵
基本运算
加减
标量乘法
矩阵乘法
乘法规则
一般不满足交换律
矩阵的幂
两个视角
标量积展开
外积展开
矩阵逆
运算规则
行列式值不为零，方阵存在逆
其他运算
转置
迹
逐项积
行列式值

4.1 矩阵：元素按长方型阵列排列

再次强调，矩阵无非就是一个表格！

一般来说，矩阵是由标量组成的矩形阵列。但是，矩阵内的元素不仅仅局限于标量，也可以是虚数、符号，乃至数学式。

丛书矩阵通常由粗体斜体大写字母表示，比如 $\boldsymbol{X}$、$\boldsymbol{V}$、$\boldsymbol{A}$、$\boldsymbol{B}$ 等。一般用 $\boldsymbol{X}$ 来表达原始样本数据矩阵。注意，如果是随机变量构成的列向量，本系列丛书会用希腊字母 χ，比如 D 维随机变量 $\chi = [X_1, X_2, \ldots, X_D]^{\mathrm{T}}$。

如图 1 所示，一个 $n \times D$ (n by capital D) 矩阵 $\boldsymbol{X}$，具体如下：

$$\boldsymbol{X}_{n\times D} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{bmatrix} \tag{1}$$

其中，n 是**矩阵行数** (number of rows in the matrix)，D 是**矩阵列数** (number of columns in the matrix)。

从数据角度，n 是样本个数，D 是样本数据特征数。比如，鸢尾花数据集，不考虑标签 (即鸢尾花三大类 setosa、versicolor、virginica)，数据集本身 $n = 150$，$D = 4$。本系列丛书《数学要素》一册专门聊过为什么会选择 n 和 D 这两个字母，这里就不再重复。

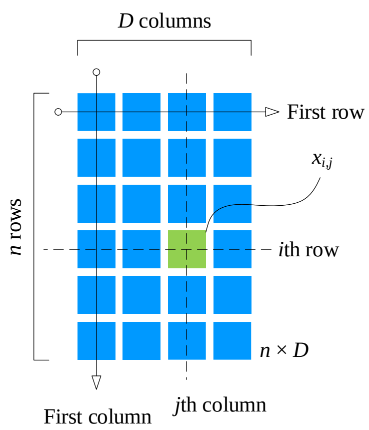


图 1. $n \times D$ 矩阵 $\boldsymbol{X}$

矩阵构造

矩阵 $\boldsymbol{X}$ 中，**元素** (element) $x_{i,j}$ 被称作 (i, j) 元素 ($i\ j$ entry 或 $i\ j$ element)；$x_{i,j}$ 出现在 i 行 j 列 (appears in row i and column j)。注意，i 和 j 的先后次序，先说行，再说列。

如图 2 所示，矩阵 $\boldsymbol{X}$ 可以看做是由一系列行向量或列向量按照一定规则构造而成。比如，矩阵 $\boldsymbol{X}$ 可以写成一组上下叠放的行向量：

$$\boldsymbol{X}_{n\times D}=\begin{bmatrix}\boldsymbol{x}^{(1)}\\ \boldsymbol{x}^{(2)}\\ \vdots\\ \boldsymbol{x}^{(n)}\end{bmatrix}=\begin{bmatrix}x_{1,1} & x_{1,2} & \cdots & x_{1,D}\\ x_{2,1} & x_{2,2} & \cdots & x_{2,D}\\ \vdots & \vdots & \ddots & \vdots\\ x_{n,1} & x_{n,2} & \cdots & x_{n,D}\end{bmatrix} \tag{2}$$

其中，行向量 $\boldsymbol{x}^{(i)}$ 为矩阵 $\boldsymbol{X}$ 第 i 行，具体为：

$$\boldsymbol{x}^{(i)}=\begin{bmatrix}x_{i,1} & x_{i,2} & \cdots & x_{i,D}\end{bmatrix} \tag{3}$$

重要的事情多说几遍，以鸢尾花数据集为例，它的每一行代表一朵花。

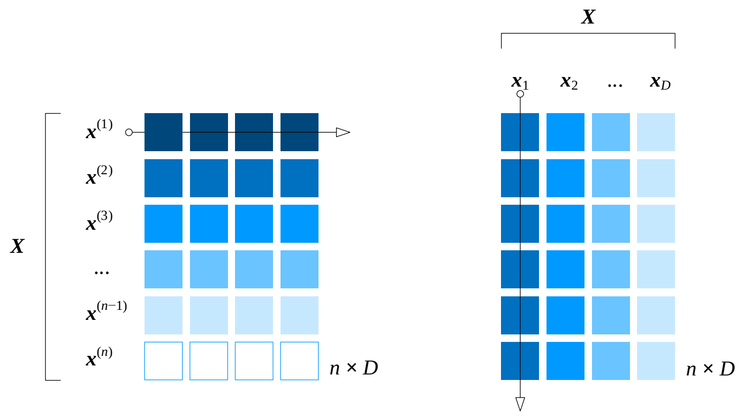


图 2. 矩阵可以看做是由行向量或列向量构造

矩阵 $\boldsymbol{X}$ 也可以写成一组左右放置的列向量：

$$\boldsymbol{X}_{n\times D}=\begin{bmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_D\end{bmatrix}=\begin{bmatrix}x_{1,1} & x_{1,2} & \cdots & x_{1,D}\\ x_{2,1} & x_{2,2} & \cdots & x_{2,D}\\ \vdots & \vdots & \ddots & \vdots\\ x_{n,1} & x_{n,2} & \cdots & x_{n,D}\end{bmatrix} \tag{4}$$

其中，列向量 $\boldsymbol{x}_j$ 为矩阵 $\boldsymbol{X}$ 第 j 列：

$$
\boldsymbol{x}_j = \begin{bmatrix} x_{1,j} \\ x_{2,j} \\ \vdots \\ x_{n,j} \end{bmatrix} \tag{5}
$$

还是以鸢尾花数据集为例，它的每一列代表一个特征，比如花萼长度。

再次强调，一般情况，本书默认向量为列向量，除非具体说明。

实际上，上述思路是用纵线或横线将矩阵划分成**分块矩阵** (block matrix)。分块矩阵有助于简化矩阵运算，本书后续会深入介绍分块矩阵相关内容。

Bk4_Ch4_01.py 用不同方式构造矩阵。

注意，numpy.matrix() 和 numpy.array() 都可以构造矩阵。但是两者结果有显著区别。numpy.matrix() 产生的数据类型是严格的 2 维<class 'numpy.matrix'>；而 numpy.array() 产生的数据可以是 1 维、2 维、乃至 n 维，类型统称为<class 'numpy.ndarray'>。

此外，在乘法和乘幂运算时，这两种不同方式构造的矩阵也会有明显差别，本章后续将逐步介绍。

```python
# Bk4_Ch4_01.py

import numpy as np

# 2d matrix
A_matrix = np.matrix([[2,4],
                      [6,8]])
print(A_matrix.shape)
print(type(A_matrix))

# 1d array
A_1d = np.array([2,4])
print(A_1d.shape)
print(type(A_1d))

# 2d array
A_2d = np.array([[2,4],
                 [6,8]])
print(A_2d.shape)
print(type(A_2d))

# 3d array
A1 = [[2,4],
      [6,8]]

A2 = [[1,3],
      [5,7]]

A3 = [[1,0],
      [0,1]]
A_3d = np.array([A1,A2,A3])
print(A_3d.shape)
print(type(A_3d))
```

4.2 矩阵形状：每种形状都有特殊用途

矩阵形状对于矩阵运算至关重要。本书之前介绍的**行向量** (row vector) 和**列向量** (column vector) 也是特殊形状的矩阵。稍作回顾，行向量可以看做一行多列的矩阵，列向量是一列多行矩阵。

图 3 总结几种常见矩阵形状，本节逐一讲解。

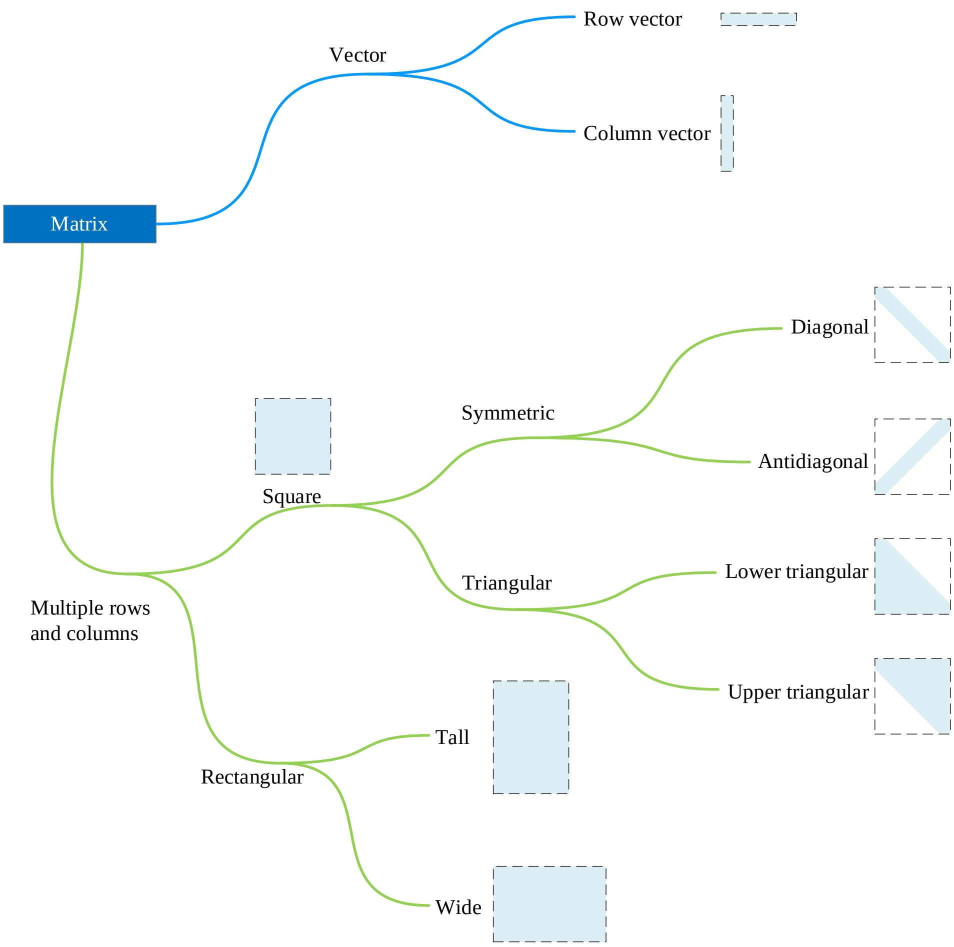


图 3. 几种常见矩阵形状

方阵

方阵 (square matrix) 指的是行列数相等的矩阵。$n \times n$ 矩阵被称作 n **阶方阵** (n-square matrix)。

对称矩阵 (symmetric matrix) 是一种特殊方阵。对阵矩阵的右上和左下方向元素以**主对角线** (main diagonal) 镜像对称。主对角线和**副对角线** (antidiagonal, secondary diagonal, minor diagonal) 的位置如图 4 所示。

对称矩阵**转置** (transpose) 结果为本身，比如满足下式的矩阵 $\boldsymbol{A}$ 便是对称矩阵：

$$\boldsymbol{A} = \boldsymbol{A}^{\mathrm{T}} \tag{6}$$

本章后续将详细介绍转置运算。

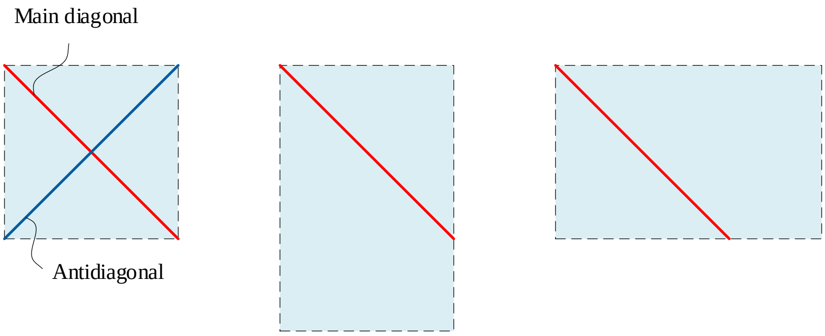


图 4. 主对角线和副对角线

对角矩阵

对角矩阵 (diagonal matrix) 是主对角线之外的元素皆为 0 (its non-diagonal entries of a square matrix are all zero) 的矩阵，比如下例：

$$\boldsymbol{\Lambda}_{n\times n} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \tag{7}$$

图 5 比较对称矩阵和对角矩阵。

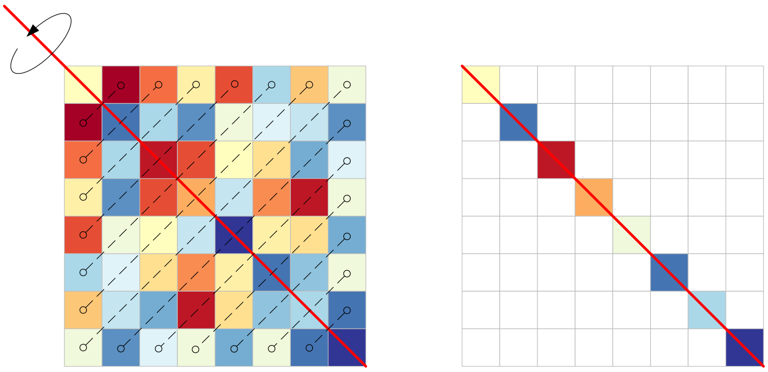


图 5. 对称矩阵和对角矩阵之间关系

请读者注意，不加说明时，本书中的对角矩阵都是方阵；但是，对角矩阵也可以是长方形矩阵，比如如下两例：

$$
\begin{bmatrix} 1 & & & \\ & 2 & & \\ & & 3 & \end{bmatrix}, \quad \begin{bmatrix} 1 & & \\ & 2 & \\ & & 3 \\ & & \end{bmatrix} \tag{8}
$$

副对角矩阵 (anti-diagonal matrix) 是副对角线之外元素皆为 0 的矩阵。

本书还常用 diag() 函数。如图 6 所示，diag($\boldsymbol{A}$) 提取矩阵 $\boldsymbol{A}$ 主对角线元素，结果为列向量。diag($\boldsymbol{a}$) 将向量 $\boldsymbol{a}$ 展成对角阵 $\boldsymbol{B}$，$\boldsymbol{B}$ 主对角线元素依次为向量 $\boldsymbol{a}$ 元素。

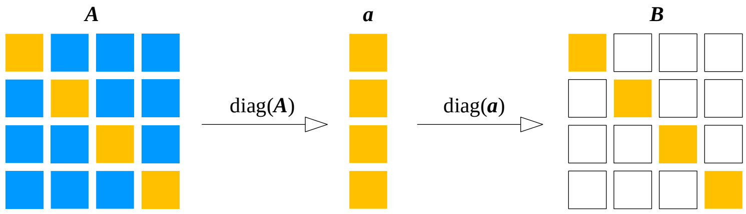


图 6. diag() 函数

Python 中，完成 diag() 函数为 `numpy.diag()`。注意，`numpy.diag(A)` 提取矩阵 $\boldsymbol{A}$ 对角线元素，结果为一维数组，形似行向量。

`Bk4_Ch4_02.py` 展示如何使用 `numpy.diag()`。

```python
# Bk4_Ch4_02.py

import numpy as np

A = np.matrix([[1,2,3],
               [4,5,6],
               [7,8,9]])

# extract diagonal elements
a = np.diag(A)

# construct a diagonal matrix
A_diag = np.diag(a)
```

单位矩阵

单位矩阵 (identity matrix) 是一种特殊对角矩阵。n 阶单位矩阵 (n-square identity matrix) 的特点是 $n \times n$ 方阵对角线上的元素为 1，其他为 0；本书中，单位矩阵用 $\boldsymbol{I}$ 来表达：

$$
\boldsymbol{I}_{n\times n}=\begin{bmatrix}1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1\end{bmatrix} \tag{9}
$$

也有很多文献用 $\boldsymbol{E}$ 代表单位矩阵。

三角矩阵

三角矩阵 (triangular matrix) 也是特殊的方阵。如果方阵对角线以下元素均为零，这个矩阵被称作**上三角矩阵** (upper triangular matrix)：

$$
\boldsymbol{U}_{n\times n}=\begin{bmatrix}u_{1,1} & u_{1,2} & \dots & u_{1,n}\\ 0 & u_{2,2} & \dots & u_{2,n}\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \dots & u_{n,n}\end{bmatrix} \tag{10}
$$

如果方阵对角线以上元素均为零，这个矩阵被称作**下三角矩阵** (lower triangular matrix)：

$$
\boldsymbol{L}_{n\times n}=\begin{bmatrix}l_{1,1} & 0 & \dots & 0\\ l_{2,1} & l_{2,2} & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ l_{n,1} & l_{n,2} & \dots & l_{n,n}\end{bmatrix} \tag{11}
$$

提一嘴，如果矩阵 $\boldsymbol{A}$ 为**可逆矩阵** (invertible matrix, non-singular matrix)，$\boldsymbol{A}$ 可以通过 LU 分解变成一个下三角矩阵 $\boldsymbol{L}$ 与一个上三角矩阵 $\boldsymbol{U}$ 的乘积。本书后续将介绍包括 LU 分解在内的各种常见矩阵分解。

长方形矩阵

长方形矩阵 (rectangular matrix) 是指行数和列数不相等的矩阵，可以是“细高”或“宽矮”。常见的数据矩阵几乎都是“细高”长方形矩阵，形状类似图 1。

计算时，长方形矩阵的形状并不“友好”。大家将会在矩阵乘法等内容中看到，我们会通过数学运算现将长方形矩阵“变成”方阵，再进行下一步运算，比如矩阵分解。

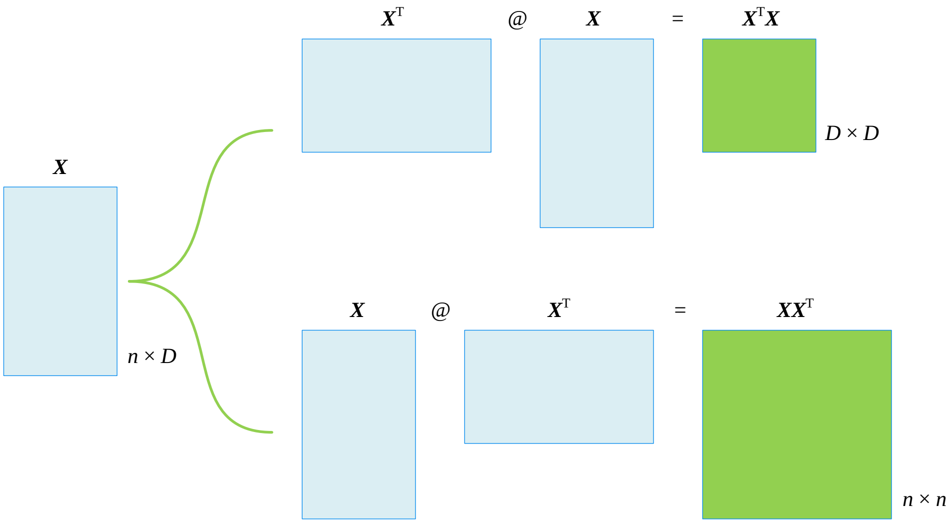


图 7. 将长方形矩阵变成方阵

图 7 所示为将细高数据矩阵 $\boldsymbol{X}$ 变成两个不同方阵的矩阵乘法运算过程。得到的结果叫格拉姆矩阵 (Gram matrix)，这是本书后文要专门讲解的内容。

多说一嘴，处理长方形矩阵有一个利器，这就是宇宙无敌的奇异值分解 (Singular Value Decomposition, SVD)，也称 SVD 分解。SVD 分解时本书后面最重要的矩阵分解，没有之一。请大家格外关注。

4.3 基本运算：加减和标量乘法

矩阵加减

两个相同大小的矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 相加，指的是把这两个矩阵对应位置元素分别相加，具体如下：

$$\boldsymbol{A}_{m\times n}+\boldsymbol{B}_{m\times n}=\begin{bmatrix} a_{1,1}+b_{1,1} & a_{1,2}+b_{1,2} & \cdots & a_{1,n}+b_{1,n} \\ a_{2,1}+b_{2,1} & a_{2,2}+b_{2,2} & \cdots & a_{2,n}+b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1}+b_{m,1} & a_{m,2}+b_{m,2} & \cdots & a_{m,n}+b_{m,n} \end{bmatrix}_{m\times n} \tag{12}$$

矩阵**加法交换律** (commutative property) 指的是：

$$\boldsymbol{A}+\boldsymbol{B}=\boldsymbol{B}+\boldsymbol{A} \tag{13}$$

矩阵**加法结合律** (associative property) 指的是：

$$\boldsymbol{A}+\boldsymbol{B}+\boldsymbol{C}=\boldsymbol{A}+(\boldsymbol{B}+\boldsymbol{C})=(\boldsymbol{A}+\boldsymbol{B})+\boldsymbol{C} \tag{14}$$

矩阵减法的运算规则和加法一致。

零矩阵

丛书用 $\boldsymbol{O}$ 表示元素全为 0 的矩阵，即**零矩阵** (zero matrix)。

零矩阵具有以下性质：

$$\boldsymbol{A}+\boldsymbol{O}=\boldsymbol{O}+\boldsymbol{A}=\boldsymbol{A} \tag{15}$$

`numpy.zeros()` 用来生成零矩阵，输入为矩阵形状；`numpy.zeros_like()` 用来生成和输入矩阵形状相同的零矩阵。

类似地，`numpy.ones()` 可以生成全 1 矩阵，输入为矩阵形状；`numpy.ones_like()` 用来生成和输入矩阵形状相同的全 1 矩阵。

`Bk4_Ch4_04.py` 介绍如何完成矩阵加减法运算。

```python
# Bk4_Ch4_03.py

import numpy as np

# define matrix
A = np.matrix([[1, 2], [3, 4]])
B = np.matrix([[2, 6], [4, 8]])

# matrix addition
A_plus_B = np.add(A,B)
A_plus_B_2 = A + B


# matrix subtraction
A_minus_B = np.subtract(A,B)
A_minus_B_2 = A - B
```

矩阵标量乘法

矩阵的**标量乘法** (scalar multiplication) 指的是，矩阵乘以某一标量，矩阵的每一个元素均乘以该标量。

矩阵 $\boldsymbol{X}$ 和标量 k 的乘积 (the product of the matrix $\boldsymbol{X}$ by a scalar k) 写作 $k\boldsymbol{X}$：

$$k\boldsymbol{X}=\begin{bmatrix} k\cdot x_{1,1} & k\cdot x_{1,2} & \cdots & k\cdot x_{1,D} \\ k\cdot x_{2,1} & k\cdot x_{2,2} & \cdots & k\cdot x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ k\cdot x_{n,1} & k\cdot x_{n,2} & \cdots & k\cdot x_{n,D} \end{bmatrix} \tag{16}$$

Bk4_Ch4_04.py 如何进行矩阵标量乘法。

```python
# Bk4_Ch4_04.py

import numpy as np

k = 2
X = [[1,2],
     [3,4]]

# scalar multiplication
k_times_X = np.dot(k,X)
k_times_X_2 = k*np.matrix(X)
```

4.4 广播原则

NumPy 中的矩阵运算常常使用**广播原则** (broadcasting)。当两个数组的形状并不相同的时候，可以通过扩展数组的方法来实现相加、相减等操作。

矩阵和标量之和

图 8 所示为，一个矩阵 $\boldsymbol{A}$ 和标量 k 之和，相当于矩阵 $\boldsymbol{A}$ 的每一个元素加 k。比如，

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + 2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 2 \\ 2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 1+2 & 2+2 \\ 3+2 & 4+2 \\ 5+2 & 6+2 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \tag{17}$$

上述运算规则也适用于减法。

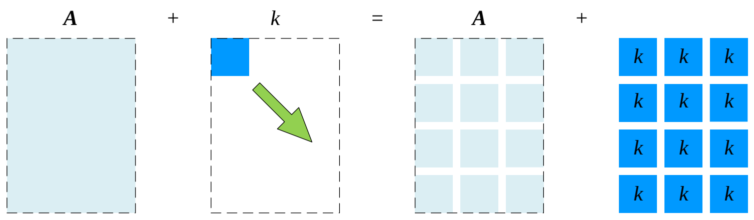


图 8. 广播原则，矩阵加标量

矩阵和列向量之和

一个矩阵 $\boldsymbol{A}$ 和列向量 $\boldsymbol{c}$ 相加，前提是的 $\boldsymbol{A}$ 行数和 $\boldsymbol{c}$ 行数相同。

如图 9 所示，矩阵 $\boldsymbol{A}$ 和列向量 $\boldsymbol{c}$ 相加，相当于 $\boldsymbol{A}$ 的每一列和 $\boldsymbol{c}$ 相加。另外一个视角，列向量 $\boldsymbol{c}$ 首先自我复制，左右排列得到和 $\boldsymbol{A}$ 形状相同的矩阵，再和 $\boldsymbol{A}$ 相加。

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 3 & 3 \\ 2 & 2 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1+3 & 2+3 \\ 3+2 & 4+2 \\ 5+1 & 6+1 \end{bmatrix} = \begin{bmatrix} 4 & 5 \\ 5 & 6 \\ 6 & 7 \end{bmatrix} \tag{18}$$

上述规则也同样适用于减法。

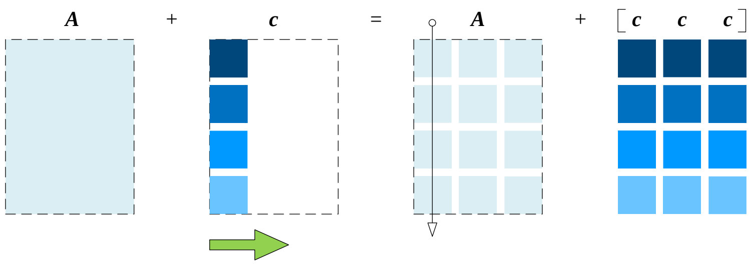


图 9. 广播原则，矩阵加列向量

矩阵和行向量之和

同理，一个矩阵 $\boldsymbol{A}$ 和行向量 $\boldsymbol{r}$ 相加，前提是的 $\boldsymbol{A}$ 列数和 $\boldsymbol{r}$ 列数相同。如图 10 所示，矩阵 $\boldsymbol{A}$ 和行向量 $\boldsymbol{r}$ 相加，相当于 $\boldsymbol{A}$ 的每一行和 $\boldsymbol{r}$ 相加。

另外一个视角，行向量 $\boldsymbol{r}$ 首先自我复制，上下叠加得到和 $\boldsymbol{A}$ 形状相同的矩阵，再和 $\boldsymbol{A}$ 相加：

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+2 & 2+1 \\ 3+2 & 4+1 \\ 5+2 & 6+1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 5 & 5 \\ 7 & 7 \end{bmatrix} \tag{19}$$

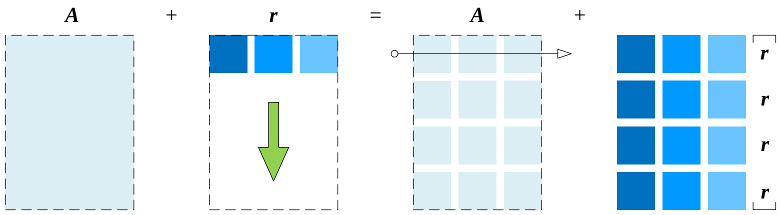


图 10. 广播原则，矩阵加行向量

列向量和行向量之和

利用广播原则，列向量可以和行向量相加。

如图 11 所示，列向量 $\boldsymbol{c}$ 自我复制，左右排列得到矩阵和 $\boldsymbol{r}$ 的列数一致；行向量 $\boldsymbol{r}$ 自我复制，上下叠加得到矩阵和 $\boldsymbol{c}$ 的行数一致。然后完成加法运算，

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 2 & 2 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 3+2 & 3+1 \\ 2+2 & 2+1 \\ 1+2 & 1+1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 4 & 3 \\ 3 & 2 \end{bmatrix} \tag{20}$$

调转行、列向量顺序，不影响结果。

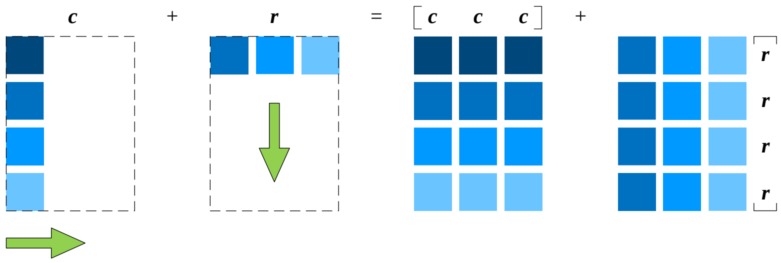


图 11. 广播原则，列向量加行向量

`Bk4_Ch4_05.py` 完成上述所示广播原则计算。此外，请大家把加号改成减号，验证广播原则在减法上的运算。

```python
# Bk4_Ch4_05.py

import numpy as np

# define matrix
A = np.matrix([[1, 2],
               [3, 4],
               [5, 6]])

# scaler
k = 2;

# column vector c
c = np.array([[3],
              [2],
              [1]])

# row vector r
r = np.array([[2,1]])

# broadcasting principles

# matrix A plus scalar k
A_plus_k = A + k

# matrix A plus column vector c
A_plus_a = A + c

# matrix A plus row vector r
A_plus_r = A + r

# column vector c plus row vector r
c_plus_r = c + r
```

4.5 矩阵乘法：线性代数的核心运算规则

法国数学家，雅克·菲利普·玛丽·比奈 (Jacques Philippe Marie Binet) 在 1812 年首先提出矩阵乘法运算规则。

毫不夸张地说，**矩阵乘法** (matrix multiplication) 在各种矩阵运算中居于核心地位，规则本身就是一项人类伟大创造！

矩阵 $\boldsymbol{A}$ 的列数等于矩阵 $\boldsymbol{B}$ 的行数，$\boldsymbol{A}$ 和 $\boldsymbol{B}$ 两个矩阵可以相乘。如果，矩阵 $\boldsymbol{A}$ 的形状是 $n \times D$，矩阵 $\boldsymbol{B}$ 的形状是 $D \times m$，两个矩阵的乘积 $\boldsymbol{C} = \boldsymbol{AB}$ 的形状是 $n \times m$：

$$\boldsymbol{C}_{n\times m} = \boldsymbol{A}_{n\times D}\boldsymbol{B}_{D\times m} = \boldsymbol{A}_{n\times D} @ \boldsymbol{B}_{D\times m} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} \end{bmatrix} \tag{21}$$

其中，

$$\boldsymbol{A}_{n\times D} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,D} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,D} \end{bmatrix}, \quad \boldsymbol{B}_{D\times m} = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,m} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{D,1} & b_{D,2} & \cdots & b_{D,m} \end{bmatrix} \tag{22}$$

为了配合 NumPy 计算，丛书利用 @ 表达矩阵乘法运算符。

矩阵乘法规则

$\boldsymbol{C}$ 的 (i, j) 元素通过 $\boldsymbol{A}$ 第 i 行元素分别乘以 $\boldsymbol{B}$ 的第 j 列元素，再求和得到：

$$c_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \ldots + a_{i,D}b_{D,j} \tag{23}$$

用矩阵乘法来表达上式，即，

$$c_{i,j} = \boldsymbol{a}^{(i)}\boldsymbol{b}_j \tag{24}$$

其中，$\boldsymbol{a}^{(i)}$ 是 $\boldsymbol{A}$ 第 i 行元素构成的行向量，$\boldsymbol{b}_j$ 是 $\boldsymbol{B}$ 的第 j 列元素构成的列向量。它俩的元素个数都是 D 个。这是理解矩阵乘法的第一视角，下一节我们会从两个不同视角来看矩阵乘法。此外，本书在分块矩阵一章中，会介绍更多视角。

(24) 也可以写成两个列向量的向量内积，即，

$$c_{i,j} = \boldsymbol{a}^{(i)\mathrm{T}}\boldsymbol{b}_j \tag{25}$$

图 12 所示为矩阵乘法规则示意图。

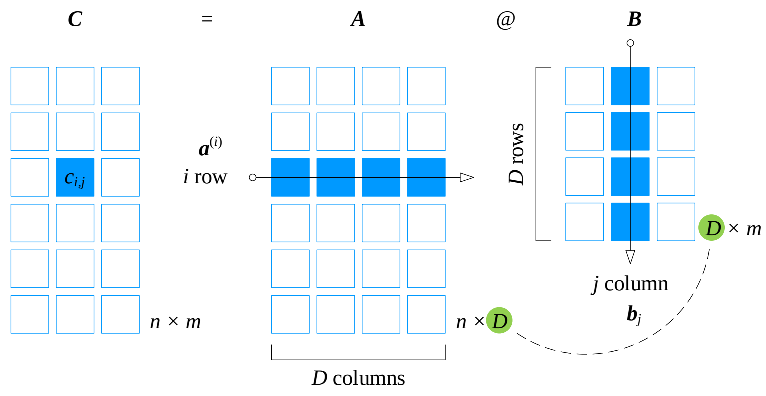


图 12. 矩阵乘法规则

一般情况，矩阵乘法不满足交换律：

$$\boldsymbol{AB} \neq \boldsymbol{BA} \tag{26}$$

另外，请读者注意以下矩阵乘法规则：

$$\begin{aligned}
&\boldsymbol{ABC} = \boldsymbol{A}(\boldsymbol{BC}) = (\boldsymbol{AB})\boldsymbol{C} \\
&k(\boldsymbol{AB}) = (k\boldsymbol{A})\boldsymbol{B} = \boldsymbol{A}(k\boldsymbol{B}) = (\boldsymbol{AB})k \\
&\boldsymbol{A}(\boldsymbol{B}+\boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC}
\end{aligned} \tag{27}$$

Bk4_Ch4_06.py 介绍在 Numpy 中如何进行矩阵乘法运算。

```python
# Bk4_Ch4_06.py

import numpy as np

A = np.array([[1, 2],
              [3, 4]])

B = np.array([[2, 4],
              [1, 3]])

# matrix multiplication
A_times_B = np.matmul(A, B)
A_times_B_2 = A@B
```

值得注意的是，对于两个由 numpy.array() 产生的数据，使用*相乘，得到的乘积是对应元素分别相乘，广播法则有效；而两个由 numpy.matrix() 产生的数据，使用*相乘，则得到结果等同于@。如果，分别由 numpy.array() 和 numpy.matrix() 产生的数据，使用*相乘，则等同于@。

请读者运行 Bk4_Ch4_07.py 给出的三个乘法例子，自行比较结果。

```
# Bk4_Ch4_07.py

import numpy as np

A = np.array([[1, 2]])

B = np.array([[5, 6],
              [8, 9]])

print(A*B)

A = np.array([[1, 2]])

B = np.matrix([[5, 6],
              [8, 9]])

print(A*B)

A = np.matrix([[1, 2]])

B = np.matrix([[5, 6],
              [8, 9]])

print(A*B)
```

矩阵的幂

n 阶方阵 (n-square matrix) $\boldsymbol{A}$ 的**矩阵的幂** (powers of matrices) 的幂为：

$$\begin{aligned} \boldsymbol{A}^0 &= \boldsymbol{I} \\ \boldsymbol{A}^1 &= \boldsymbol{A} \\ \boldsymbol{A}^2 &= \boldsymbol{A}\boldsymbol{A} \\ \boldsymbol{A}^{n+1} &= \boldsymbol{A}^n\boldsymbol{A} \end{aligned} \tag{28}$$

Bk4_Ch4_08.py 展示如何计算矩阵幂。

```
# Bk4_Ch4_08.py

from numpy.linalg import matrix_power as pw
A = np.array([[1., 2.],
              [3., 4.]])

# matrix inverse
A_3 = pw(A,3)
A_3_v3 = A@A@A

# piecewise power
A_3_piecewise = A**3
```

乘幂运算符**对 numpy.array() 和 numpy.matrix() 生成的数据有不同的运算规则。numpy.matrix() 生成矩阵 A，A**2，是矩阵乘幂；numpy.array() 生成的矩阵 B，B**2 是对元素分别平方。请读者比较 Bk4_Ch4_09.py 给出的两个例子。

```python
# Bk4_Ch4_09.py

import numpy as np

A = np.matrix([[1,3],
               [2,4]])

print(A**2)

B = np.array([[1,3],
              [2,4]])

print(B**2)
```

4.6 两个视角解剖矩阵乘法

为了更好理解矩阵乘法，我们用两个 2 × 2 矩阵相乘来讲解：

$$
\begin{aligned}
\boldsymbol{A}\boldsymbol{B} &= \boldsymbol{A}@\boldsymbol{B} \\
&= \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\
&= \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix}
\end{aligned} \tag{29}
$$

图 13 所示为两个 2 × 2 矩阵相乘如何得到结果的每一个元素。

下面，我们从两个视角来剖析矩阵乘法。这部分内容虽然在本系列丛书《数学要素》一册已经讲过一遍，为了加强大家对矩阵乘法理解，请学过的读者也耐心把本节内容扫读一遍。

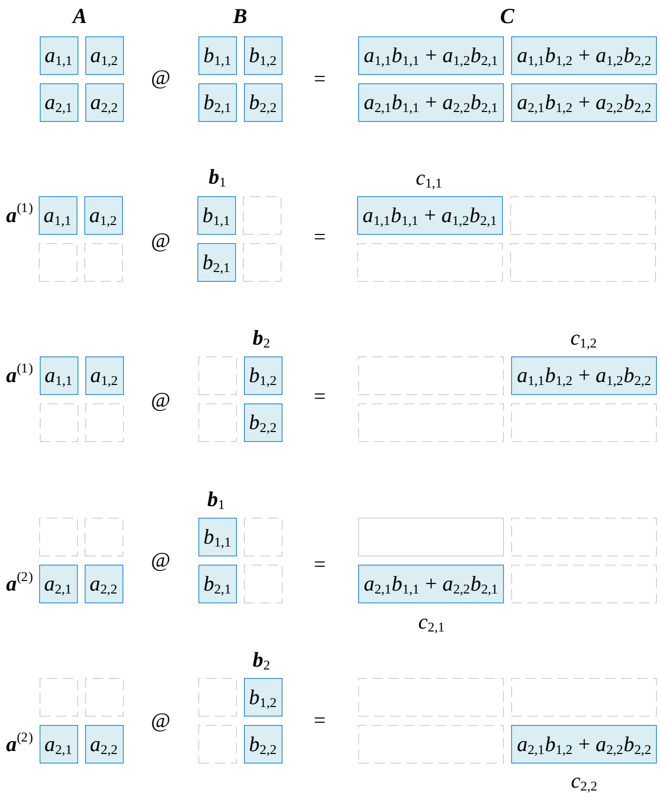


图 13. 矩阵乘法规则，两个 2 × 2 矩阵相乘为例

第一视角

第一视角是矩阵运算的常规视角，也叫做标量积展开。

如图 13 所示，矩阵乘法 $\boldsymbol{AB}$ 中，位于左侧的 $\boldsymbol{A}$ 写成一组行向量；位于右侧的 $\boldsymbol{B}$ 写成一组列向量。

$\boldsymbol{A}$ 的第 i 行 $\boldsymbol{a}^{(i)}$ 乘以 $\boldsymbol{B}$ 的第 j 列 $\boldsymbol{b}_j$，得到乘积 $\boldsymbol{C}$ 的 (i, j) 元素 $c_{i,j}$：

$$
\begin{aligned}
\boldsymbol{AB} = \boldsymbol{A} @ \boldsymbol{B} &= \begin{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} \end{bmatrix}_{1\times 2} \\ \begin{bmatrix} a_{2,1} & a_{2,2} \end{bmatrix}_{1\times 2} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} b_{1,1} \\ b_{2,1} \end{bmatrix}_{2\times 1} & \begin{bmatrix} b_{1,2} \\ b_{2,2} \end{bmatrix}_{2\times 1} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{a}^{(1)} \\ \boldsymbol{a}^{(2)} \end{bmatrix}_{2\times 1} \begin{bmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 \end{bmatrix}_{1\times 2} = \begin{bmatrix} \boldsymbol{a}^{(1)}\boldsymbol{b}_1 & \boldsymbol{a}^{(1)}\boldsymbol{b}_2 \\ \boldsymbol{a}^{(2)}\boldsymbol{b}_1 & \boldsymbol{a}^{(2)}\boldsymbol{b}_2 \end{bmatrix}_{2\times 2} \\
&= \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}
\end{aligned} \tag{30}
$$

第二视角

矩阵乘法的第二视角叫做外积展开。

将矩阵乘法 $\boldsymbol{AB}$ 中，位于左侧的 $\boldsymbol{A}$ 写成一组列向量；位于右侧的 $\boldsymbol{B}$ 写成一组行向量。如下所示，我们把 $\boldsymbol{AB}$ 展开写成矩阵加法：

$$\begin{aligned}\boldsymbol{AB} = \boldsymbol{A}@\boldsymbol{B} &= \left[\begin{bmatrix} a_{1,1} \\ a_{2,1} \end{bmatrix}_{2\times1} \quad \begin{bmatrix} a_{1,2} \\ a_{2,2} \end{bmatrix}_{2\times1}\right] \begin{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} \end{bmatrix}_{1\times2} \\ \begin{bmatrix} b_{2,1} & b_{2,2} \end{bmatrix}_{1\times2} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 \end{bmatrix}_{1\times2} \begin{bmatrix} \boldsymbol{b}^{(1)} \\ \boldsymbol{b}^{(2)} \end{bmatrix}_{2\times1} = \boldsymbol{a}_1\boldsymbol{b}^{(1)} + \boldsymbol{a}_2\boldsymbol{b}^{(2)} = \begin{bmatrix} a_{1,1} \\ a_{2,1} \end{bmatrix}_{2\times1} @ \begin{bmatrix} b_{1,1} & b_{1,2} \end{bmatrix}_{1\times2} + \begin{bmatrix} a_{1,2} \\ a_{2,2} \end{bmatrix}_{2\times1} @ \begin{bmatrix} b_{2,1} & b_{2,2} \end{bmatrix}_{1\times2} \\ &= \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} \end{bmatrix}_{2\times2} + \begin{bmatrix} a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}_{2\times2} \\ &= \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}\end{aligned} \tag{31}$$

观察矩阵乘法的这两个视角十分重要，本书下文还将深入探讨，并介绍更多视角。

4.7 转置：绕主对角线镜像

矩阵的行列互换得到的新矩阵的操作为**矩阵转置** (matrix transpose)。

如图 14 所示，一个 $n \times D$ 矩阵 $\boldsymbol{A}$ 转置得到 $D \times n$ 矩阵 $\boldsymbol{B}$，整个过程相当于矩阵 $\boldsymbol{A}$ 绕主对角线镜像。矩阵 $\boldsymbol{A}$ 的转置 (the transpose of a matrix $\boldsymbol{A}$) 记作 $\boldsymbol{A}^{\mathrm{T}}$ 或 $\boldsymbol{A}'$。注意，为了和求导记号区分，本书仅采用 $\boldsymbol{A}^{\mathrm{T}}$ 记法。

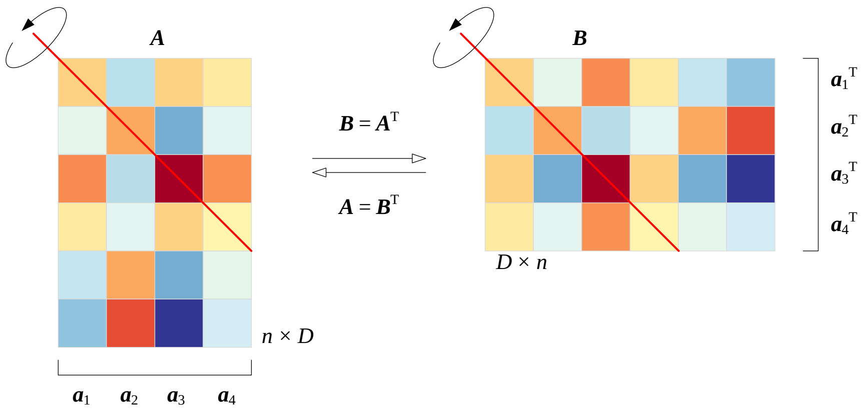


图 14. 矩阵转置

如图 14 所示，将矩阵 $\boldsymbol{A}$ 写成一组列向量：

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \boldsymbol{a}_3 & \boldsymbol{a}_4 \end{bmatrix} \tag{32}$$

矩阵 $\boldsymbol{A}$ 转置 $\boldsymbol{A}^{\mathrm{T}}$ 可以展开写作:

$$\boldsymbol{A}^{\mathrm{T}}=\begin{bmatrix}\boldsymbol{a}_1{}^{\mathrm{T}}\\\boldsymbol{a}_2{}^{\mathrm{T}}\\\boldsymbol{a}_3{}^{\mathrm{T}}\\\boldsymbol{a}_4{}^{\mathrm{T}}\end{bmatrix}\tag{33}$$

如上文所述，一个 $n\times D$ 矩阵 $\boldsymbol{A}$ 转置结果为自身，则称矩阵**对称** (symmetric):

$$\boldsymbol{A}=\boldsymbol{A}^{\mathrm{T}}\tag{34}$$

我们常用的一个对称矩阵就是列向量和自身的张量积，比如 $\boldsymbol{a}\otimes\boldsymbol{a}$ 。

矩阵转置的几个重要性质如下:

$$\begin{aligned}&\left(\boldsymbol{A}^{\mathrm{T}}\right)^{\mathrm{T}}=\boldsymbol{A}\\&\left(\boldsymbol{A}+\boldsymbol{B}\right)^{\mathrm{T}}=\boldsymbol{A}^{\mathrm{T}}+\boldsymbol{B}^{\mathrm{T}}\\&\left(k\boldsymbol{A}\right)^{\mathrm{T}}=k\boldsymbol{A}^{\mathrm{T}}\\&\left(\boldsymbol{AB}\right)^{\mathrm{T}}=\boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\\&\left(\boldsymbol{ABC}\right)^{\mathrm{T}}=\boldsymbol{C}^{\mathrm{T}}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\\&\left(\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_3\cdots\boldsymbol{A}_k\right)^{\mathrm{T}}=\boldsymbol{A}_k{}^{\mathrm{T}}\cdots\boldsymbol{A}_3{}^{\mathrm{T}}\boldsymbol{A}_2{}^{\mathrm{T}}\boldsymbol{A}_1{}^{\mathrm{T}}\end{aligned}\tag{35}$$

等长列向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的标量积等价于 $\boldsymbol{a}$ 的转置乘 $\boldsymbol{b}$，或 $\boldsymbol{b}$ 的转置乘 $\boldsymbol{a}$:

$$\boldsymbol{a}\cdot\boldsymbol{b}=\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}=\boldsymbol{b}^{\mathrm{T}}\boldsymbol{a}=a_1b_1+a_2b_2+\cdots+a_nb_n\tag{36}$$

$\boldsymbol{a}$ 的模 (L^2 范数) 也可以写成 $\boldsymbol{a}$ 转置乘自身，再开方:

$$\|\boldsymbol{a}\|=\sqrt{\boldsymbol{a}\cdot\boldsymbol{a}}=\sqrt{\boldsymbol{a}^{\mathrm{T}}\boldsymbol{a}}\tag{37}$$

Bk4_Ch4_10.py 计算矩阵转置。

```python
# Bk4_Ch4_10.py

import numpy as np

A = np.array([[1, 2],
              [3, 4],
              [5, 6]])

# matrix transpose
A_T = A.transpose()
A_T_2 = A.T
```

4.8 矩阵逆：相当于除法运算

方阵 $\boldsymbol{A}$ 如果**可逆** (invertible)，仅当存在矩阵 $\boldsymbol{B}$ 使得：

$$\boldsymbol{AB}=\boldsymbol{BA}=\boldsymbol{I} \tag{38}$$

矩阵 $\boldsymbol{A}$ 的**逆** (inverse) 写作 $\boldsymbol{A}^{-1}$。矩阵可逆 (invertible) 也称**非奇异** (non-singular)；否则就称矩阵不可逆 (non-invertible)，或奇异 (singular)。

请读者注意以下和矩阵逆有关的运算规则：

$$\begin{aligned}&\left(\boldsymbol{A}^{\mathrm{T}}\right)^{-1}=\left(\boldsymbol{A}^{-1}\right)^{\mathrm{T}}\\&\left(\boldsymbol{AB}\right)^{-1}=\boldsymbol{B}^{-1}\boldsymbol{A}^{-1}\\&\left(\boldsymbol{ABC}\right)^{-1}=\boldsymbol{C}^{-1}\boldsymbol{B}^{-1}\boldsymbol{A}^{-1}\\&\left(k\boldsymbol{A}\right)^{-1}=\frac{1}{k}\boldsymbol{A}^{-1}\end{aligned} \tag{39}$$

其中，假设 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$、$\boldsymbol{AB}$ 和 $\boldsymbol{ABC}$ 逆运算存在。

一般情况，

$$\left(\boldsymbol{A}+\boldsymbol{B}\right)^{-1}\neq\boldsymbol{A}^{-1}+\boldsymbol{B}^{-1} \tag{40}$$

特别地，对于任意 2 × 2 矩阵 $\boldsymbol{A}$：

$$\boldsymbol{A}=\begin{bmatrix}a&b\\c&d\end{bmatrix} \tag{41}$$

矩阵 $\boldsymbol{A}$ 的逆 $\boldsymbol{A}^{-1}$ 可以通过下式获得，

$$\boldsymbol{A}^{-1}=\begin{bmatrix}a&b\\c&d\end{bmatrix}^{-1}=\frac{1}{|\boldsymbol{A}|}\begin{bmatrix}d&-b\\-c&a\end{bmatrix} \tag{42}$$

其中

$$|\boldsymbol{A}|=ad-bc \tag{43}$$

$|\boldsymbol{A}|$ 被称作矩阵 $\boldsymbol{A}$ **行列式** (determinant)。

注意，行列式值 $|\boldsymbol{A}|$ 不为 0 时，矩阵 $\boldsymbol{A}$ 才存在逆。本章后续将详细讲解行列式值计算。

若下式成立，如果矩阵 $\boldsymbol{A}$ 是**正交矩阵** (orthogonal matrix)：

$$\boldsymbol{A}^{\mathrm{T}}=\boldsymbol{A}^{-1} \tag{44}$$

本书后续还将深入探讨正交矩阵的性质和应用，本节不做展开。

`Bk4_Ch4_11.py` 展示如 `Numpy` 如何矩阵逆。

```python
# Bk4_Ch4_11.py

from numpy.linalg import inv
A = np.array([[1., 2.],
              [3., 4.]])

# matrix inverse
A_inverse = inv(A)
A_times_A_inv = A@A_inverse
```

注意，对于 numpy.matrix() 产生的矩阵 A，可以通过 A.I 计算矩阵 A 的逆，比如 Bk4_Ch4_12.py 给出的例子。但是，这一方法不能使用在 numpy.array() 生成的矩阵。numpy.array() 生成的矩阵求逆，可以使用 numpy.linalg.inv()。

```python
# Bk4_Ch4_12.py

import numpy as np

A = np.matrix([[1, 2],
               [3, 4]])

print(A.I)

B = np.array([[1, 2],
              [3, 4]])

print(B.I)
```

4.9 迹：主对角元素之和

$n \times n$ 矩阵 $\boldsymbol{A}$ 的**迹** (trace) 为其主对角线元素之和：

$$\operatorname{tr}(\boldsymbol{A}) = \sum_{i=1}^{n} a_{i,i} = a_{1,1} + a_{2,2} + \cdots + a_{n,n} \tag{45}$$

举个例子，

$$\operatorname{tr}(\boldsymbol{A}) = \operatorname{tr}\left(\begin{bmatrix} 1 & -1 & 0 \\ 3 & 2 & 4 \\ -2 & 0 & 3 \end{bmatrix}\right) = 1 + 2 + 3 = 6 \tag{46}$$

一个矩阵的迹是其特征值的总和，这个重要的性质，我们会在主成分分析中看到它的应用。

Bk4_Ch4_13.py 介绍如何计算矩阵的迹。

```
# Bk4_Ch4_13.py

import numpy as np
A = np.array([[1, -1, 0],
              [3,  2, 4],
              [-2, 0, 3]])

# calculate trace of A
tr_A = np.trace(A)
```

请大家注意以下有关矩阵迹的性质：

$$\begin{aligned}&\operatorname{tr}(\boldsymbol{A}+\boldsymbol{B})=\operatorname{tr}(\boldsymbol{A})+\operatorname{tr}(\boldsymbol{B})\\&\operatorname{tr}(k\boldsymbol{A})=k\cdot\operatorname{tr}(\boldsymbol{A})\\&\operatorname{tr}(\boldsymbol{A}^{\mathrm{T}})=\operatorname{tr}(\boldsymbol{A})\\&\operatorname{tr}(\boldsymbol{AB})=\operatorname{tr}(\boldsymbol{BA})\end{aligned}\tag{47}$$

本系列丛书后续会介绍协方差矩阵 (covariance matrix) 可以用椭圆来表达。举个例子，给定一个协方差矩阵为：

$$\boldsymbol{\Sigma}=\begin{bmatrix}2.5 & 1.5\\1.5 & 2.5\end{bmatrix}\tag{48}$$

图 15 左图就是代表上述协方差矩阵的椭圆，即旋转椭圆。

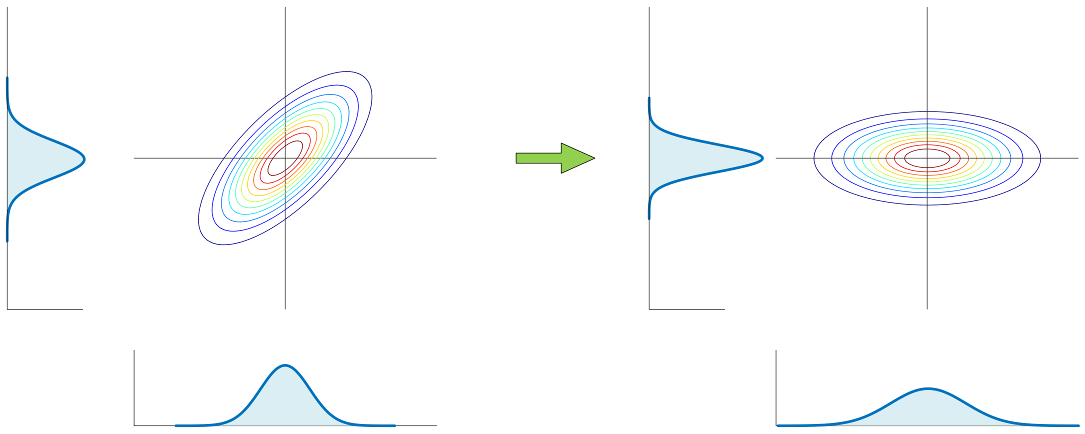


图 15. 协方差矩阵和椭圆关系

经过旋转操作，椭圆的长轴和横轴重合，得到图 15 右图椭圆，即正椭圆，对应的协方差矩阵为：

$$\Sigma_{\text{rotated}}=\begin{bmatrix}4 & 0\\ 0 & 1\end{bmatrix} \tag{49}$$

相信大家已经注意到，两个协方差矩阵的迹相同，都是 5！这一点非常重要，本系列丛书后续会在不同板块中探讨。

大家可能会问，两个协方差矩阵之间有怎样的联系？或者说，如何从 (48) 计算得到 (49)？椭圆之间的旋转角度怎么确定？本书后续介绍的特征值分解将回答这些疑问。

4.10 逐项积：对应元素相乘

在讲解向量运算时，我们介绍过**元素乘积** (element-wise multiplication)，也称为**阿达玛乘积** (Hadamard product) 或**逐项积** (piecewise product)。

逐项积常常用在矩阵上。两个形状相同的矩阵的逐项积是矩阵对应元素相乘，结果形状不变：

$$A_{n\times D}\odot B_{n\times D}=\begin{bmatrix}a_{1,1}b_{1,1} & a_{1,2}b_{1,2} & \cdots & a_{1,D}b_{1,D}\\ a_{2,1}b_{2,1} & a_{2,2}b_{2,2} & \cdots & a_{2,D}b_{2,D}\\ \vdots & \vdots & \ddots & \vdots\\ a_{n,1}b_{n,1} & a_{n,2}b_{n,2} & \cdots & a_{n,D}b_{n,D}\end{bmatrix}_{n\times D} \tag{50}$$

图 16 所示为矩阵逐项积运算法则示意图。

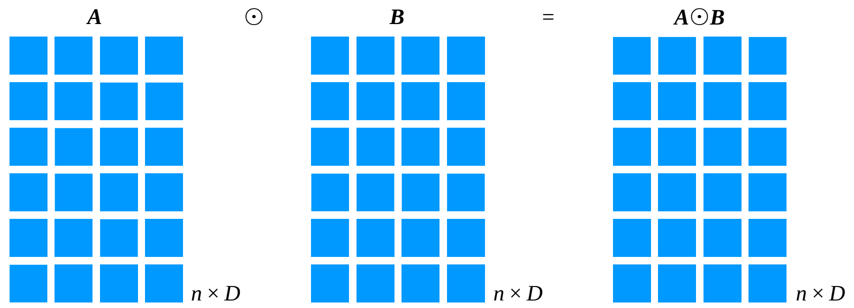


图 16. 矩阵逐项积

Bk4_Ch4_14.py 计算逐项积。

```
# Bk4_Ch4_14.py

import numpy as np
```

```
A = np.array([[1,2],
              [3,4]])

B = np.array([[5,6],
              [7,8]])

# Hadamard product
A_times_B_piecewise = np.multiply(A,B)
A_times_B_piecewise_V2 = A*B
```

4.11 行列式：将矩阵映射到标量值

每个方阵都有自己的**行列式** (determinant)，比如方阵 $\boldsymbol{A}$ 的行列式值可以表达为 $|\boldsymbol{A}|$ 或 $\det(\boldsymbol{A})$。如果行列式值非零，方阵则可逆或非奇异。

白话说，行列式是将一个方阵 $\boldsymbol{A}$ 根据一定的规则映射到一个标量。

一阶方阵的行列式值：

$$|a_{11}| = a_{11} \tag{51}$$

二阶方阵的行列式值：

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21} \tag{52}$$

三阶方阵的行列式值：

$$\begin{aligned} \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} &= \begin{vmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} + \begin{vmatrix} 0 & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} + \begin{vmatrix} 0 & 0 & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \\ &= a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \end{aligned} \tag{53}$$

根据以上规律，可以发现 $n \times n$ 矩阵 $\boldsymbol{A}$ 的行列式值可以通过递归计算得到。

更多性质

特别地，对角阵的行列式值为：

$$\begin{vmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} \tag{54}$$

三角阵的行列式值为：

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} \tag{55}$$

上述规则也适用于计算下三角矩阵的行列式值。

请读者注意以下行列式性质：

$$\begin{aligned} \det(\boldsymbol{AB}) &= \det(\boldsymbol{A}) \cdot \det(\boldsymbol{B}) \\ \det(c\boldsymbol{A}_{n\times n}) &= c^n \det(\boldsymbol{A}) \\ \det(\boldsymbol{A}^{\mathrm{T}}) &= \det(\boldsymbol{A}) \\ \det(\boldsymbol{A}^n) &= \det(\boldsymbol{A})^n \\ \det(\boldsymbol{A}^{-1}) &= \frac{1}{\det(\boldsymbol{A})} \end{aligned} \tag{56}$$

几何视角

图 17 给出的是二阶矩阵行列式的几何意义。

给定 2×2 方阵 $\boldsymbol{A}$，具体为：

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{57}$$

$\boldsymbol{A}$ 可以写成左右排列的两个列向量：

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 \end{bmatrix} \tag{58}$$

即：

$$\boldsymbol{a}_1 = \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix}, \quad \boldsymbol{a}_2 = \begin{bmatrix} a_{12} \\ a_{22} \end{bmatrix} \tag{59}$$

如图 17 所示，以 $\boldsymbol{a}_1$ 和 $\boldsymbol{a}_2$ 为两条边构造得到一个平行四边形。这个平行四边形的面积就是 $\boldsymbol{A}$ 的行列式值。下面我们推导一下。

如图 18 所示，矩形和三角形的面积很容易计算。平行四边形的面积，就是矩形面积减去两倍的绿色三角形面积，再减去两倍的橙色三角形面积，即：

$$\begin{aligned} \text{Area} &= (a_{11}+a_{12})(a_{21}+a_{22}) - a_{12}(a_{21}+a_{22}) - a_{21}(a_{11}+a_{12}) \\ &= a_{11}a_{22} - a_{12}a_{21} \end{aligned} \tag{60}$$

这和 (52) 行列式结果一致。

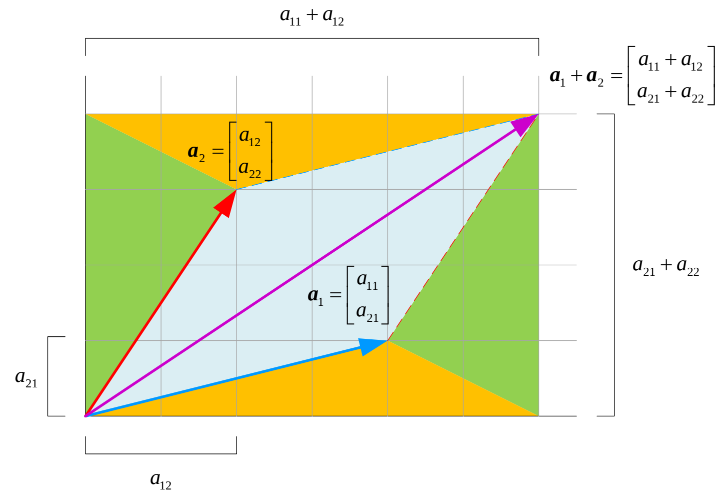


图 17. 二阶矩阵的行列式的几何意义

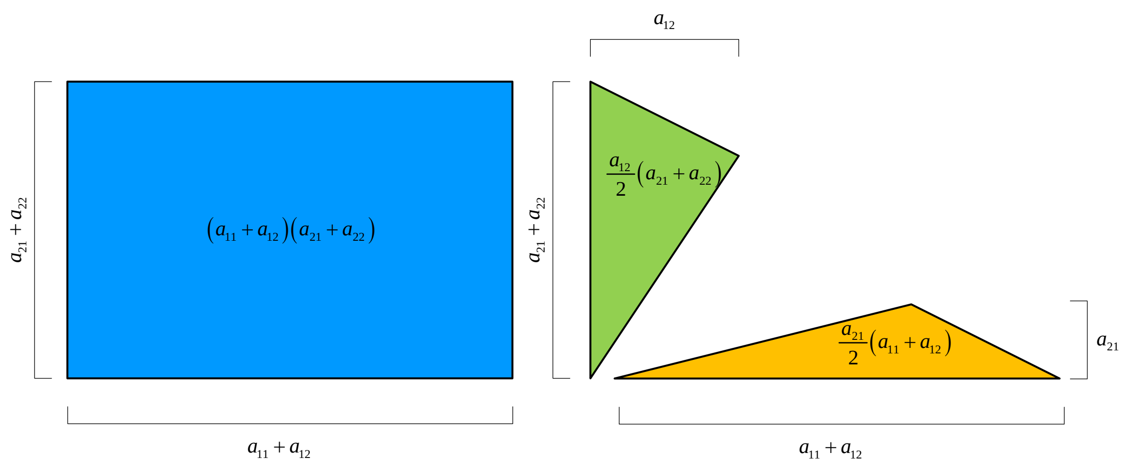


图 18. 三个几何形状的面积

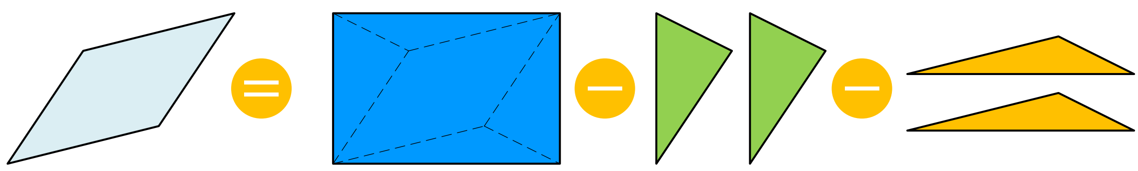
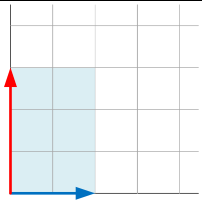
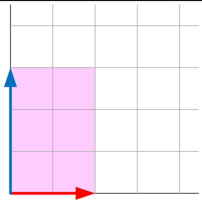
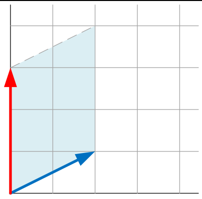
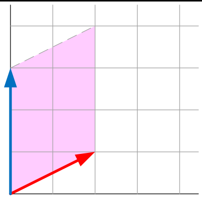
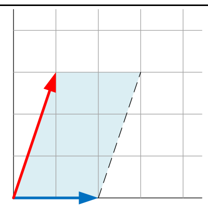
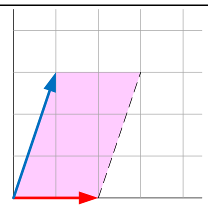
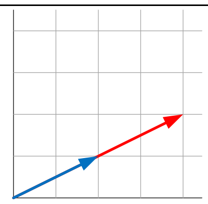


图 19. 求平行四边形面积

表 1 给出了几个特殊 2 × 2 方阵的行列式值和对应的平行四边形形状及面积。希望大家仔细对比表中几幅图，很容发现行列式值正、负、零都有其特定的几何内涵。

表 1. 几个特殊 2 × 2 方阵的行列式值

行列式值	向量	图形

$\begin{vmatrix} 2 & 0 \\ 0 & 3 \end{vmatrix} = 6$	$\boldsymbol{a}_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix},\quad \boldsymbol{a}_2 = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$	
$\begin{vmatrix} 0 & 2 \\ 3 & 0 \end{vmatrix} = -6$	$\boldsymbol{a}_1 = \begin{bmatrix} 0 \\ 3 \end{bmatrix},\quad \boldsymbol{a}_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$	
$\begin{vmatrix} 2 & 0 \\ 1 & 3 \end{vmatrix} = 6$	$\boldsymbol{a}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix},\quad \boldsymbol{a}_2 = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$	
$\begin{vmatrix} 0 & 2 \\ 3 & 1 \end{vmatrix} = -6$	$\boldsymbol{a}_1 = \begin{bmatrix} 0 \\ 3 \end{bmatrix},\quad \boldsymbol{a}_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$	
$\begin{vmatrix} 2 & 1 \\ 0 & 3 \end{vmatrix} = 6$	$\boldsymbol{a}_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix},\quad \boldsymbol{a}_2 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$	
$\begin{vmatrix} 1 & 2 \\ 3 & 0 \end{vmatrix} = -6$	$\boldsymbol{a}_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix},\quad \boldsymbol{a}_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$	
$\begin{vmatrix} 2 & 4 \\ 1 & 2 \end{vmatrix} = 0$	$\boldsymbol{a}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix},\quad \boldsymbol{a}_2 = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$	

Bk4_Ch4_15.py 计算行列式值。

```
# Bk4_Ch4_15.py

import numpy as np
A = np.array([[4, 2],
              [1, 3]])

# calculate determinant of A
det_A = np.linalg.det(A)
```

从面积到体积

本节前文讲解行列式值用的例子中矩阵都是 2×2，现在聊一聊 3×3 方阵的行列式值的几何意义。

我们先看一个最简单例子，给定如下 3×3 对角方阵：

$$\begin{bmatrix} 1 & & \\ & 2 & \\ & & 3 \end{bmatrix} \tag{61}$$

如图 20 所示，上式代表三维空间中边长分别为 1、2、3 的立方体，而行列式值为 6 则说明它的体积为 6。

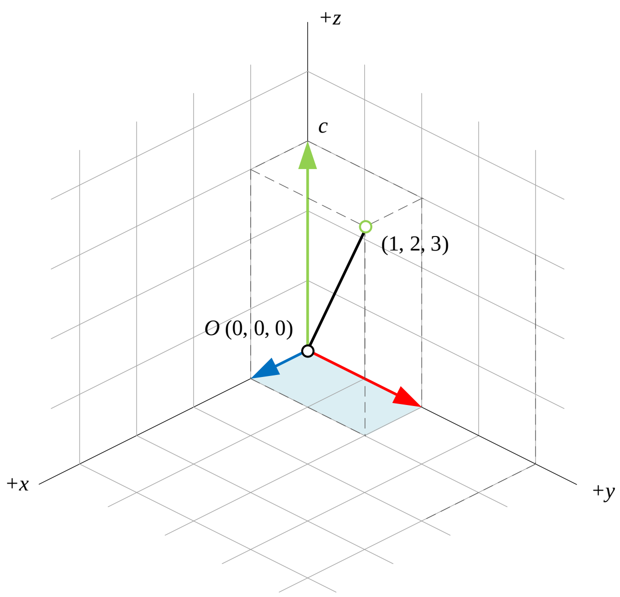


图 20. 立方体的体积为 6

对 (61) 稍作修改，将三个对角元素值改为 0，得到矩阵：

$$\begin{bmatrix} 1 & & \\ & 2 & \\ & & 0 \end{bmatrix} \tag{62}$$

这时，上式矩阵的行列式值为 0。从图 20 上来看，这个立方体“趴”在 xy 平面上，对应浅蓝色阴影，显然它的体积为 0。

行列式中某行或某列全为 0，行列式值为 0。从几何角度很容易理解，因为这个平行体的某条边长为 0，因此它的体积就是 0。

如图 21 (a) 所示，而对于任意 3 × 3 方阵 $\boldsymbol{A}$，它的行列式值的几何含义就是由其三个列向量 $\boldsymbol{a}_1$、$\boldsymbol{a}_2$、$\boldsymbol{a}_3$ 构造的平行六面体的体积。注意这个体积值有正负。特别地，如果 $\boldsymbol{a}_3$ 在 $\boldsymbol{a}_1$、$\boldsymbol{a}_2$ 构造的平面中，也就是 $\boldsymbol{a}_3$ 躺在图 21 (b) 中浅蓝色平面上，对应体积为 0，即方阵 $\boldsymbol{A}$ 行列式值为 0。

图 21 (b) 这种情况下，$\boldsymbol{a}_1$、$\boldsymbol{a}_2$、$\boldsymbol{a}_3$ 线性相关，$\boldsymbol{A}$ 的秩为 2，这是本书后文要介绍的内容。

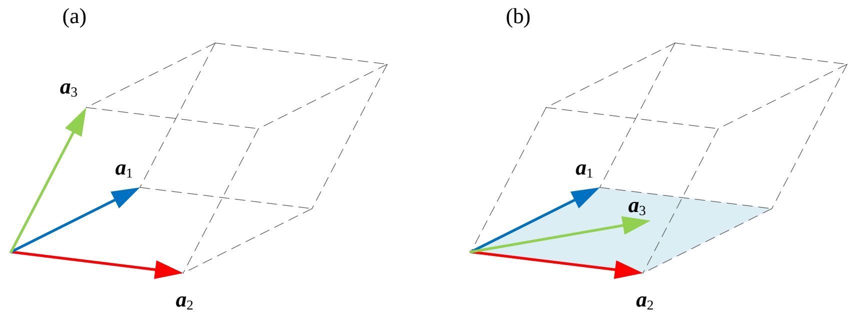


图 21. 3 × 3 方阵 $\boldsymbol{A}$ 行列式值的几何含义

此外，行列式值和线性变换有着密切关系，线性变换对面积/体积产生改变的比例就是变换矩阵的行列式值。本书后续将展开讲解。

多维

再进一步，给定如下 $D \times D$ 对角方阵：

$$\begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_D \end{bmatrix}_{D \times D} \tag{63}$$

上式说明，在 D 维空间中，这个“长方体”的边长分别为 λ_1、λ_2、… λ_D。而这个长方体的体积就是这些值连乘。

举个例子，在多元高斯分布的概率密度函数中，我们可以在分母上看到矩阵的行列式值 $|\boldsymbol{\Sigma}|^{\frac{1}{2}}$，$|\boldsymbol{\Sigma}|^{\frac{1}{2}}$ 起到的作用就是体积缩放：

$$f_{\chi}(\boldsymbol{x})=\frac{\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \tag{64}$$

几何变换

大家会逐渐发现，我们平时遇到的矩阵大部分不是对角方阵，计算其面积或体积显然不容易。有没有一种办法能够将这些矩阵，还原成其原本面貌，也就是将变换矩阵转化成对角方阵？

从几何视角来看，就是将形状各异的几何体，通过几何操作转换成"矩阵"或"立方体"，这样更容易计算面积或体积。

举个例子，如图 22 所示把平行四边形变成一个长方形。

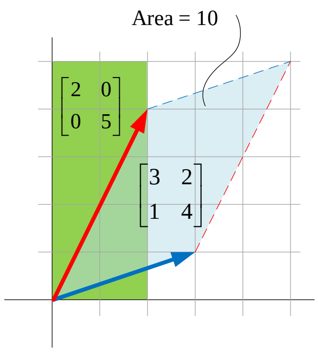


图 22. 把平行四边形变成长方形

答案是肯定的，这就是我们本书后续在矩阵分解中要着重讲解的特征值分解。此外，图 22 的 2 和 5 不是随心所欲定的，大家很快就会发现 2 和 5 叫做矩阵的特征值。

向量积

本书前文介绍的向量积也可以通过行列式计算得到，比如：

$$\begin{aligned}\boldsymbol{a}\times\boldsymbol{b}&=\begin{vmatrix}\boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k}\\a_1 & a_2 & a_3\\b_1 & b_2 & b_3\end{vmatrix}\\&=\begin{vmatrix}a_2 & a_3\\b_2 & b_3\end{vmatrix}\boldsymbol{i}-\begin{vmatrix}a_1 & a_3\\b_1 & b_3\end{vmatrix}\boldsymbol{j}+\begin{vmatrix}a_1 & a_2\\b_1 & b_2\end{vmatrix}\boldsymbol{k}\\&=(a_2b_3-a_3b_2)\boldsymbol{i}+(a_3b_1-a_1b_3)\boldsymbol{j}+(a_1b_2-a_2b_1)\boldsymbol{k}\end{aligned} \tag{65}$$

还用上一章的例子，给定 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 向量：

$$
\begin{aligned}
\boldsymbol{a} &= -2\boldsymbol{i} + \boldsymbol{j} + \boldsymbol{k} \\
\boldsymbol{b} &= \boldsymbol{i} - 2\boldsymbol{j} - \boldsymbol{k}
\end{aligned} \tag{66}
$$

$\boldsymbol{a} \times \boldsymbol{b}$ 结果如下：

$$
\begin{aligned}
\boldsymbol{a} \times \boldsymbol{b} &= \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ -2 & 1 & 1 \\ 1 & -2 & -1 \end{vmatrix} \\
&= \begin{vmatrix} 1 & 1 \\ -2 & -1 \end{vmatrix} \boldsymbol{i} - \begin{vmatrix} -2 & 1 \\ 1 & -1 \end{vmatrix} \boldsymbol{j} + \begin{vmatrix} -2 & 1 \\ 1 & -2 \end{vmatrix} \boldsymbol{k} \\
&= \boldsymbol{i} - \boldsymbol{j} + 3\boldsymbol{k}
\end{aligned} \tag{67}
$$

本章走马观花地介绍了常见矩阵运算规则，显然不能用四幅图概括主要内容。

本章介绍的每一种矩阵运算规则都是重要的数学工具，都有自己的应用场景。而在所有线性代数的运算法则中，矩阵乘法居于核心地位。

就像儿时背诵九九乘法表一样，矩阵乘法规则就是我们的“成人乘法表”——必须要熟练掌握！

随着本书对线性代数知识抽丝剥茧，大家会由浅入深认识到矩阵乘法的伟力。