

3 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
- ▶ `transpose()` 矩阵转置，比如 `A.transpose()`，等同于 `A.T`

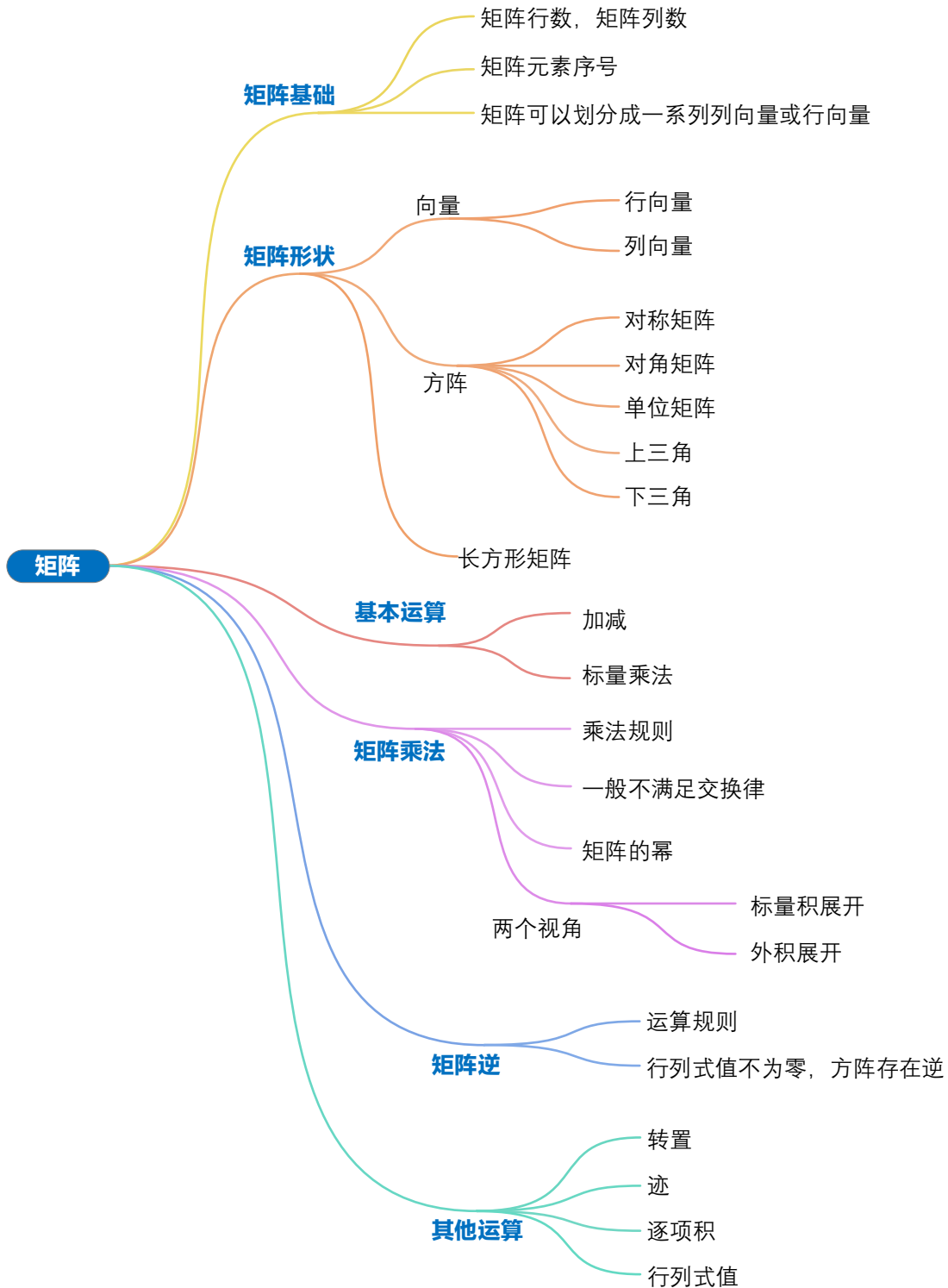
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

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



3.1 矩阵：元素按长方型阵列排列

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 \mathbf{X} 、 \mathbf{V} 、 \mathbf{A} 、 \mathbf{B} 等。一般用 \mathbf{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 $\boldsymbol{\chi}$ ，比如 D 维随机变量 $\boldsymbol{\chi} = [X_1, X_2, \dots, X_D]^T$ 。

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

$$\mathbf{X}_{n \times D} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{bmatrix} \quad (1)$$

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。

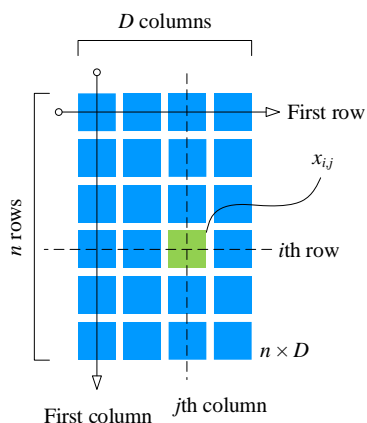


图 1. $n \times D$ 矩阵 \mathbf{X}

矩阵构造

矩阵 \mathbf{X} 中，**元素** (element) x_{ij} 被称作 (i, j) 元素 (ij entry 或 ij element)； x_{ij} 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 \mathbf{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 \mathbf{X} 可以写成一组上下叠放的行向量：

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

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

$$\mathbf{x}^{(i)} = [x_{i,1} \quad x_{i,2} \quad \cdots \quad x_{i,D}] \quad (3)$$

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。

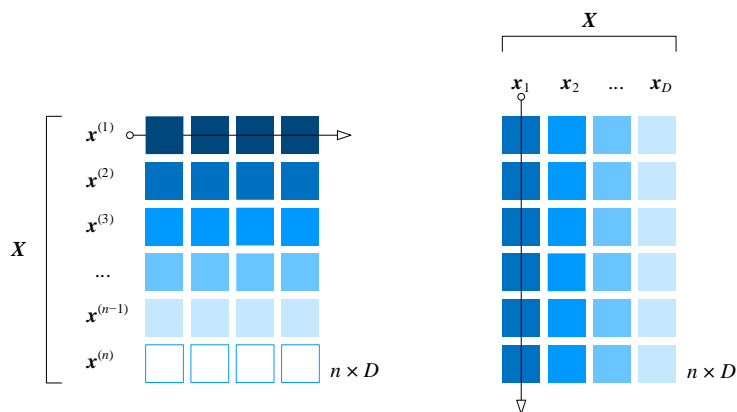


图 2. 矩阵可以看做是由行向量或列向量构造

矩阵 \mathbf{X} 也可以写成一组左右放置的列向量：

$$\mathbf{X}_{n \times D} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_D] = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{bmatrix} \quad (4)$$

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

$$\mathbf{x}_j = \begin{bmatrix} x_{1,j} \\ x_{2,j} \\ \vdots \\ x_{n,j} \end{bmatrix} \quad (5)$$

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_Ch3_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。

```
# Bk4_Ch3_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))
```

3.2 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。

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

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

图 3 总结几种常见矩阵形状，本节逐一讲解。

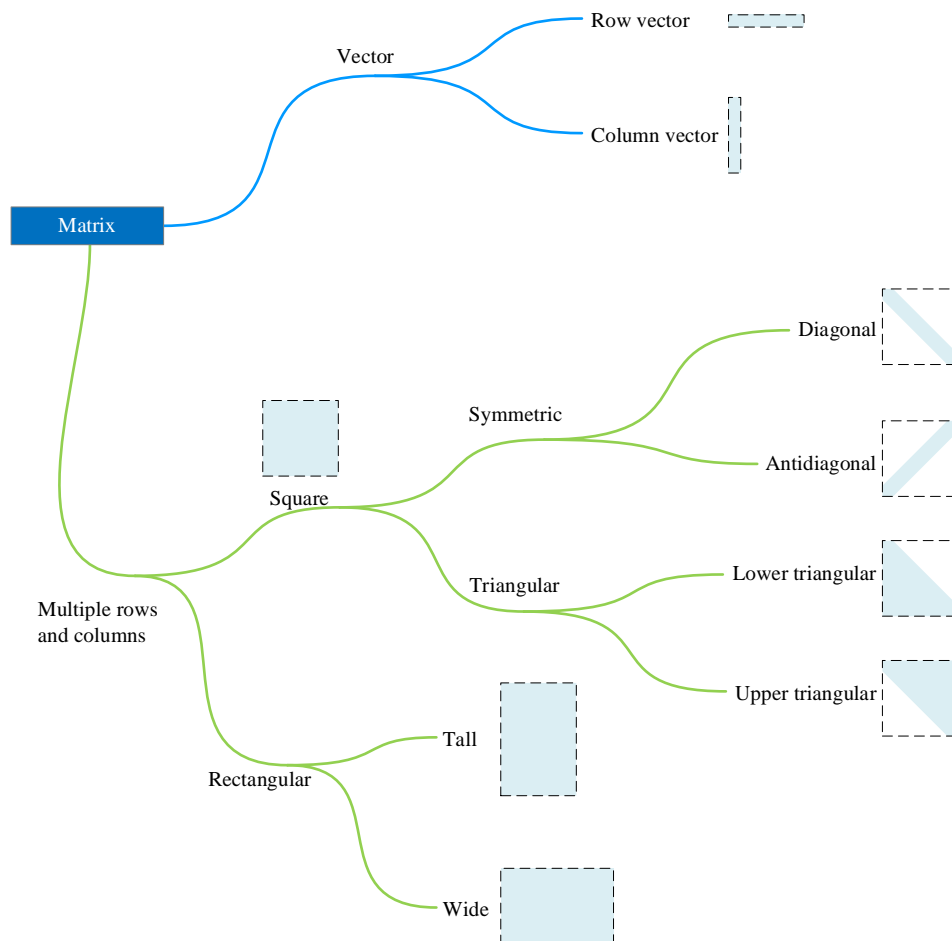


图 3. 几种常见矩阵形状

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 A 便是对称矩阵：

$$A = A^T \quad (6)$$

本章后续将详细介绍转置运算。

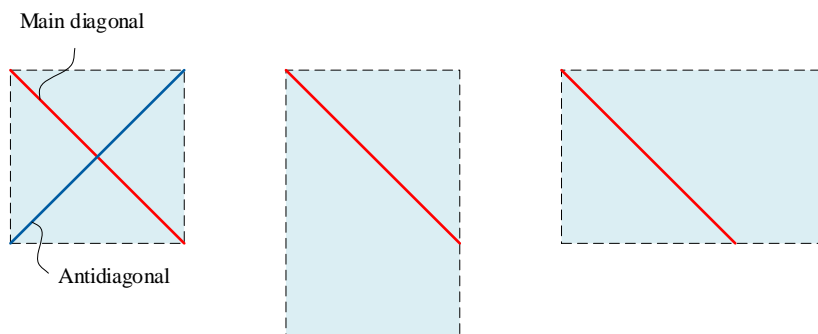


图 4. 主对角线和副对角线

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：

$$A_{n \times n} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \quad (7)$$

图 5 比较对称矩阵和对角矩阵。

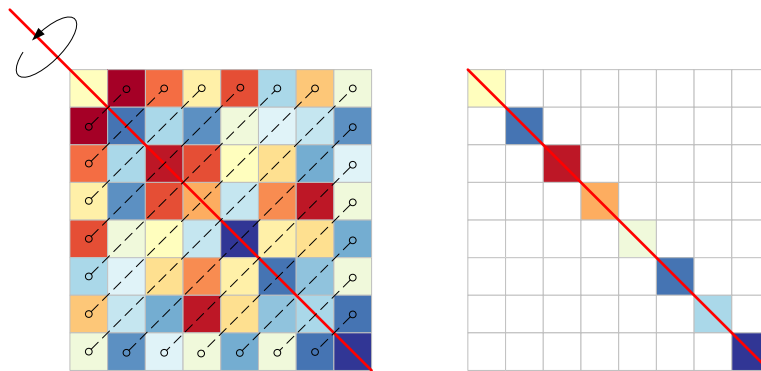


图 5. 对称矩阵和对角矩阵之间关系

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，比如下两例：

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

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(A)` 提取矩阵 **A** 主对角线元素, 结果为列向量。`diag(a)` 将向量 **a** 展成对角阵 **B**, **B** 主对角线元素依次为向量 **a** 元素。

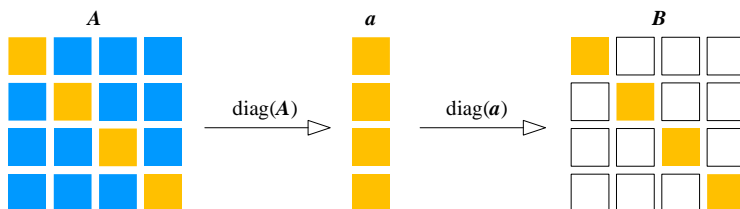


图 6. `diag()` 函数

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



Bk4_Ch3_02.py 展示如何使用 `numpy.diag()`。

```
# Bk4_Ch3_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 **I** 来表达:

$$\mathbf{I}_{n \times n} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \quad (9)$$

也有很多文献用 **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)：

$$\mathbf{U}_{n \times n} = \begin{bmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n} \\ 0 & u_{2,2} & \dots & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{n,n} \end{bmatrix} \quad (10)$$

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)：

$$\mathbf{L}_{n \times n} = \begin{bmatrix} l_{1,1} & 0 & \dots & 0 \\ l_{2,1} & l_{2,2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \dots & l_{n,n} \end{bmatrix} \quad (11)$$

提一嘴，如果矩阵 \mathbf{A} 为**可逆矩阵** (invertible matrix, non-singular matrix)， \mathbf{A} 可以通过 LU 分解变成一个下三角矩阵 \mathbf{L} 与一个上三角矩阵 \mathbf{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。

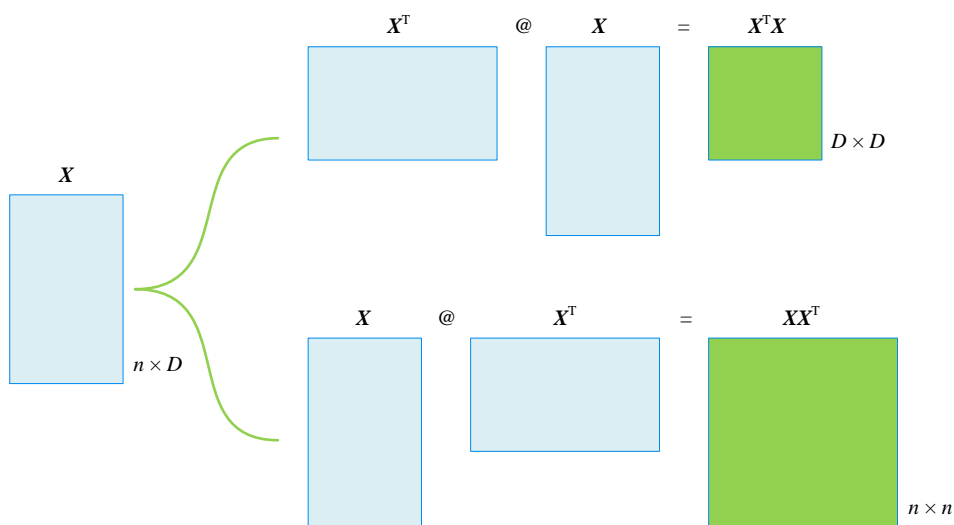


图 7. 将长方形矩阵变成方阵

图 7 所示为将细高数据矩阵 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。

3.3 基本运算：加减和标量乘法

矩阵加减

两个相同大小的矩阵 A 和 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：

$$A_{m \times n} + B_{m \times n} = \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & \dots & a_{1,n} + b_{1,n} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & \dots & a_{2,n} + b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} + b_{m,1} & a_{m,2} + b_{m,2} & \dots & a_{m,n} + b_{m,n} \end{bmatrix}_{m \times n} \quad (12)$$

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

$$A + B = B + A \quad (13)$$

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

$$A + B + C = A + (B + C) = (A + B) + C \quad (14)$$

矩阵减法的运算规则和加法一致。

零矩阵

丛书用 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：

$$A + O = O + A = A \quad (15)$$

`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_Ch3_03.py 介绍如何完成矩阵加减法运算。

```
# Bk4_Ch3_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 X 和标量 k 的乘积 (the product of the matrix X by a scalar k) 写作 kX :

$$kX = \begin{bmatrix} k \cdot x_{1,1} & k \cdot x_{1,2} & \cdots & k \cdot x_{1,D} \\ k \cdot x_{2,1} & k \cdot x_{2,2} & \cdots & k \cdot x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ k \cdot x_{n,1} & k \cdot x_{n,2} & \cdots & k \cdot x_{n,D} \end{bmatrix} \quad (16)$$



Bk4_Ch3_04.py 如何进行矩阵标量乘法。

```
# Bk4_Ch3_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)
```

3.4 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 A 和标量 k 之和，相当于矩阵 A 的每一个元素加 k 。比如，

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

上述运算规则也适用于减法。

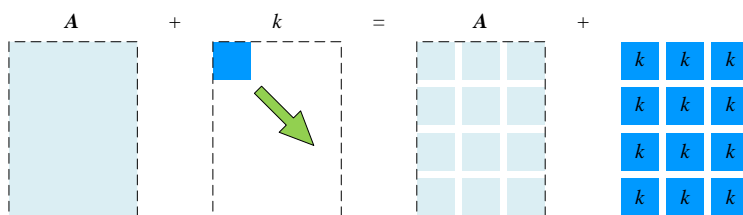


图 8. 广播原则，矩阵加标量

矩阵和列向量之和

一个矩阵 A 和列向量 c 相加，前提是的 A 行数和 c 行数相同。

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

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

上述规则也同样适用于减法。

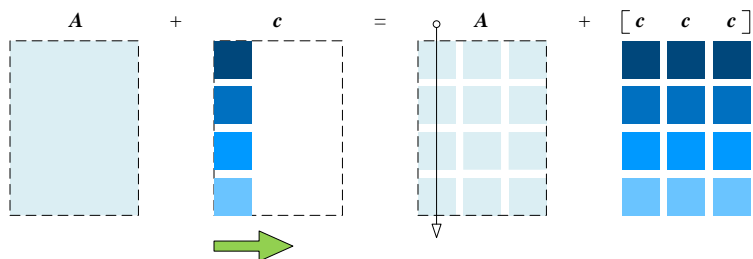


图 9. 广播原则，矩阵加列向量

矩阵和行向量之和

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

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

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + [2 \quad 1] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+2 & 2+1 \\ 3+2 & 4+1 \\ 5+2 & 6+1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 5 & 5 \\ 7 & 7 \end{bmatrix} \quad (19)$$

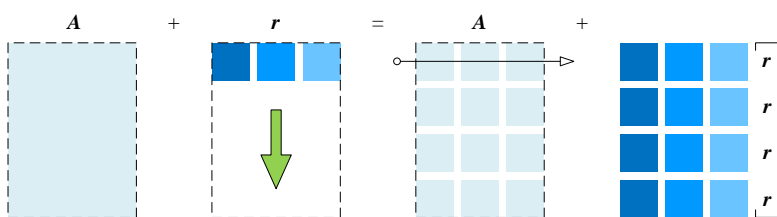


图 10. 广播原则，矩阵加行向量

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 c 自我复制，左右排列得到矩阵和 r 的列数一致；行向量 r 自我复制，上下叠加得到矩阵和 c 的行数一致。然后完成加法运算，

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} + [2 \quad 1] = \begin{bmatrix} 3 & 3 \\ 2 & 2 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 1 \\ 2 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 3+2 & 3+1 \\ 2+2 & 2+1 \\ 1+2 & 1+1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 4 & 3 \\ 3 & 2 \end{bmatrix} \quad (20)$$

调转行、列向量顺序，不影响结果。

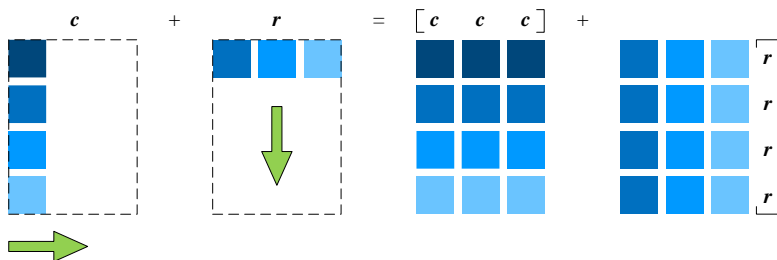


图 11. 广播原则，列向量加行向量



Bk4_Ch3_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。

```
# Bk4_Ch3_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
```

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

$$C_{n \times m} = A_{n \times D} B_{D \times m} = A_{n \times D} @ B_{D \times m} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} \end{bmatrix} \quad (21)$$

其中,

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

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

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

$$c_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \dots + a_{i,D}b_{D,j} \quad (23)$$

用矩阵乘法来表达上式, 即,

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

其中, $\mathbf{a}^{(i)}$ 是 \mathbf{A} 第 i 行元素构成的行向量, \mathbf{b}_j 是 \mathbf{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,

$$c_{i,j} = \mathbf{a}^{(i)\top} \mathbf{b}_j \quad (25)$$

图 12 所示为矩阵乘法规则示意图。

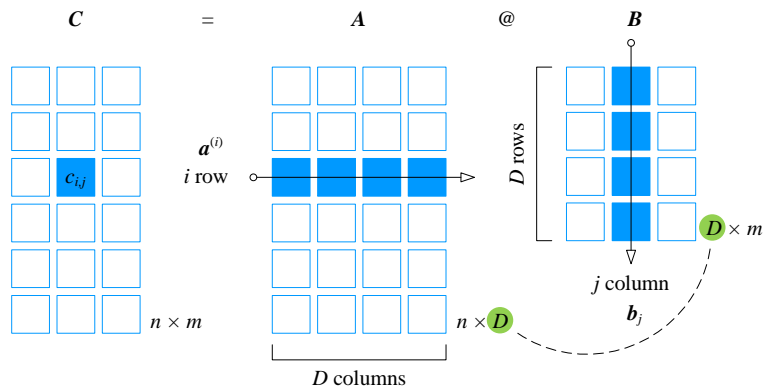


图 12. 矩阵乘法规则

一般情况, 矩阵乘法不满足交换律:

$$AB \neq BA \quad (26)$$

另外，请读者注意以下矩阵乘法规则：

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



Bk4_Ch3_06.py 介绍在 Numpy 中如何进行矩阵乘法运算。

```
# Bk4_Ch3_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_Ch3_07.py 给出的三个乘法例子，自行比较结果。

```
# Bk4_Ch3_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) A 的**矩阵的幂** (powers of matrices) 的幂为：

$$\begin{aligned} A^0 &= I \\ A^1 &= A \\ A^2 &= AA \\ A^{n+1} &= A^n A \end{aligned} \quad (28)$$



Bk4_Ch3_08.py 展示如何计算矩阵幂。

```
# Bk4_Ch3_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_Ch3_09.py 给出的两个例子。

```
# Bk4_Ch3_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)
```

3.6 两个视角解剖矩阵乘法

为了更好地理解矩阵乘法，我们用两个 2×2 矩阵相乘来讲解：

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

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。

$$\begin{array}{c}
 \begin{array}{ccc}
 \mathbf{A} & & \mathbf{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{array} \\
 \\
 \begin{array}{ccc}
 \mathbf{a}^{(1)} & & \mathbf{b}_1 \\
 \begin{bmatrix} a_{1,1} & a_{1,2} \\ \phantom{a_{2,1}} & \phantom{a_{2,2}} \end{bmatrix} & @ & \begin{bmatrix} b_{1,1} \\ b_{2,1} \end{bmatrix} \\
 & = & \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} \\ \phantom{a_{2,1}b_{1,1}} + \phantom{a_{2,2}b_{2,1}} \end{bmatrix} \\
 & & \mathbf{c}_{1,1}
 \end{array} \\
 \\
 \begin{array}{ccc}
 \mathbf{a}^{(1)} & & \mathbf{b}_2 \\
 \begin{bmatrix} a_{1,1} & a_{1,2} \\ \phantom{a_{2,1}} & \phantom{a_{2,2}} \end{bmatrix} & @ & \begin{bmatrix} \phantom{b_{1,1}} & b_{1,2} \\ \phantom{b_{2,1}} & b_{2,2} \end{bmatrix} \\
 & = & \begin{bmatrix} \phantom{a_{1,1}b_{1,1}} + \phantom{a_{1,2}b_{2,1}} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ \phantom{a_{2,1}b_{1,1}} + \phantom{a_{2,2}b_{2,1}} & \phantom{a_{2,1}b_{1,2}} + \phantom{a_{2,2}b_{2,2}} \end{bmatrix} \\
 & & \mathbf{c}_{1,2}
 \end{array} \\
 \\
 \begin{array}{ccc}
 \mathbf{a}^{(2)} & & \mathbf{b}_1 \\
 \begin{bmatrix} \phantom{a_{1,1}} & \phantom{a_{1,2}} \\ a_{2,1} & a_{2,2} \end{bmatrix} & @ & \begin{bmatrix} b_{1,1} \\ b_{2,1} \end{bmatrix} \\
 & = & \begin{bmatrix} \phantom{a_{1,1}b_{1,1}} + \phantom{a_{1,2}b_{2,1}} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} \end{bmatrix} \\
 & & \mathbf{c}_{2,1}
 \end{array} \\
 \\
 \begin{array}{ccc}
 \mathbf{a}^{(2)} & & \mathbf{b}_2 \\
 \begin{bmatrix} \phantom{a_{1,1}} & \phantom{a_{1,2}} \\ a_{2,1} & a_{2,2} \end{bmatrix} & @ & \begin{bmatrix} \phantom{b_{1,1}} & b_{1,2} \\ \phantom{b_{2,1}} & b_{2,2} \end{bmatrix} \\
 & = & \begin{bmatrix} \phantom{a_{1,1}b_{1,1}} + \phantom{a_{1,2}b_{2,1}} & \phantom{a_{1,1}b_{1,2}} + \phantom{a_{1,2}b_{2,2}} \\ \phantom{a_{2,1}b_{1,1}} + \phantom{a_{2,2}b_{2,1}} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix} \\
 & & \mathbf{c}_{2,2}
 \end{array}
 \end{array}$$

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。

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

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

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

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

第二视角

矩阵乘法的第二视角叫做外积展开。

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

$$\begin{aligned}\mathbf{AB} = \mathbf{A} @ \mathbf{B} &= \begin{bmatrix} a_{1,1} \\ a_{2,1} \end{bmatrix}_{2 \times 1} \begin{bmatrix} a_{1,2} \\ a_{2,2} \end{bmatrix}_{2 \times 1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix}_{1 \times 2} \\ &= \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 \end{bmatrix}_{1 \times 2} \begin{bmatrix} \mathbf{b}^{(1)} \\ \mathbf{b}^{(2)} \end{bmatrix}_{2 \times 1} = \mathbf{a}_1 \mathbf{b}^{(1)} + \mathbf{a}_2 \mathbf{b}^{(2)} = \begin{bmatrix} a_{1,1} \\ a_{2,1} \end{bmatrix}_{2 \times 1} @ \begin{bmatrix} b_{1,1} & b_{1,2} \end{bmatrix}_{1 \times 2} + \begin{bmatrix} a_{1,2} \\ a_{2,2} \end{bmatrix}_{2 \times 1} @ \begin{bmatrix} b_{2,1} & b_{2,2} \end{bmatrix}_{1 \times 2} \\ &= \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} \end{bmatrix}_{2 \times 2} + \begin{bmatrix} a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}_{2 \times 2} \\ &= \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}\end{aligned}\quad (31)$$

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。

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

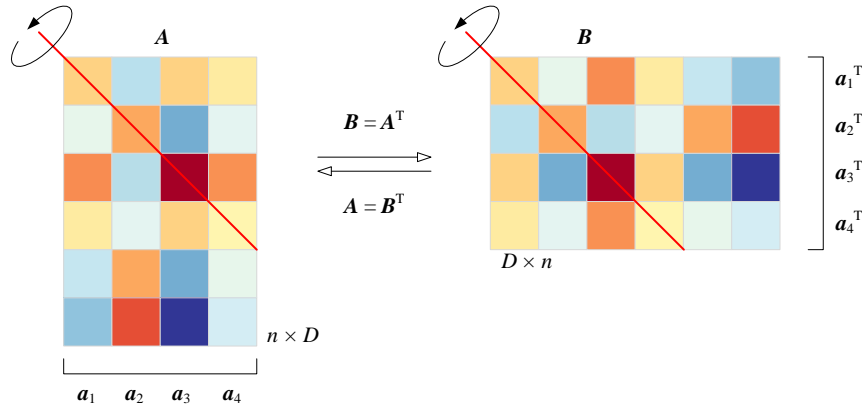


图 14. 矩阵转置

如图 14 所示，将矩阵 A 写成一组列向量：

$$A = [a_1 \ a_2 \ a_3 \ a_4] \quad (32)$$

矩阵 A 转置 A^T 可以展开写作：

$$A^T = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \\ a_4^T \end{bmatrix} \quad (33)$$

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

$$A = A^T \quad (34)$$

矩阵转置的几个重要性质如下：

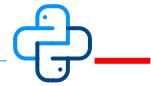
$$\begin{aligned} (A^T)^T &= A \\ (A+B)^T &= A^T + B^T \\ (kA)^T &= kA^T \\ (AB)^T &= B^T A^T \\ (ABC)^T &= C^T B^T A^T \end{aligned} \quad (35)$$

等长列向量 a 和 b 的标量积等价于 a 的转置乘 b ，或 b 的转置乘 a ：

$$a \cdot b = a^T b = b^T a = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \quad (36)$$

a 的模 (L^2 范数) 也可以写成 a 转置乘自身，再开方：

$$\|a\| = \sqrt{a \cdot a} = \sqrt{a^T a} \quad (37)$$



Bk4_Ch3_10.py 计算矩阵转置。

```
# Bk4_Ch3_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
```

3.8 矩阵逆：相当于除法运算

方阵 A 如果可逆 (invertible)，仅当存在矩阵 B 使得：

$$AB = BA = I \quad (38)$$

矩阵 A 的逆 (inverse) 写作 A^{-1} 。矩阵可逆也称非奇异 (non-singular)。

请读者注意以下和矩阵逆有关的运算规则：

$$\begin{aligned} (A^T)^{-1} &= (A^{-1})^T \\ (AB)^{-1} &= B^{-1}A^{-1} \\ (ABC)^{-1} &= C^{-1}B^{-1}A^{-1} \\ (kA)^{-1} &= \frac{1}{k}A^{-1} \end{aligned} \quad (39)$$

其中，假设 A 、 B 、 C 、 AB 和 ABC 逆运算存在。

一般情况，

$$(A+B)^{-1} \neq A^{-1} + B^{-1} \quad (40)$$

特别地，对于任意 2×2 矩阵 A ：

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad (41)$$

矩阵 A 的逆 A^{-1} 可以通过下式获得，

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

其中

$$|A| = ad - bc \quad (43)$$

$|A|$ 被称作矩阵 A **行列式** (determinant)。

注意，行列式值 $|A|$ 不为 0 时，矩阵 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 A 是**正交矩阵** (orthogonal matrix)：

$$A^T = A^{-1} \quad (44)$$

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_Ch3_11.py 展示如 Numpy 如何矩阵逆。



```
# Bk4_Ch3_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_Ch3_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()`。

```
# Bk4_Ch3_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)
```

3.9 迹：主对角元素之和

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

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^n a_{i,i} = a_{1,1} + a_{2,2} + \cdots + a_{n,n} \quad (45)$$

举个例子,

$$\text{tr}(\mathbf{A}) = \text{tr} \begin{pmatrix} 1 & -1 & 0 \\ 3 & 2 & 4 \\ -2 & 0 & 3 \end{pmatrix} = 1 + 2 + 3 = 6 \quad (46)$$

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_Ch3_13.py 介绍如何计算矩阵的迹。

```
# Bk4_Ch3_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:

$$\begin{aligned} \text{tr}(\mathbf{A} + \mathbf{B}) &= \text{tr}(\mathbf{A}) + \text{tr}(\mathbf{B}) \\ \text{tr}(k\mathbf{A}) &= k \cdot \text{tr}(\mathbf{A}) \\ \text{tr}(\mathbf{A}^T) &= \text{tr}(\mathbf{A}) \\ \text{tr}(\mathbf{AB}) &= \text{tr}(\mathbf{BA}) \end{aligned} \quad (47)$$

3.10 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:

$$\mathbf{A}_{n \times D} \odot \mathbf{B}_{n \times D} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,2}b_{1,2} & \cdots & a_{1,D}b_{1,D} \\ a_{2,1}b_{2,1} & a_{2,2}b_{2,2} & \cdots & a_{2,D}b_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}b_{n,1} & a_{n,2}b_{n,2} & \cdots & a_{n,D}b_{n,D} \end{bmatrix}_{n \times D} \quad (48)$$

图 15 所示为矩阵逐项积运算法则示意图。

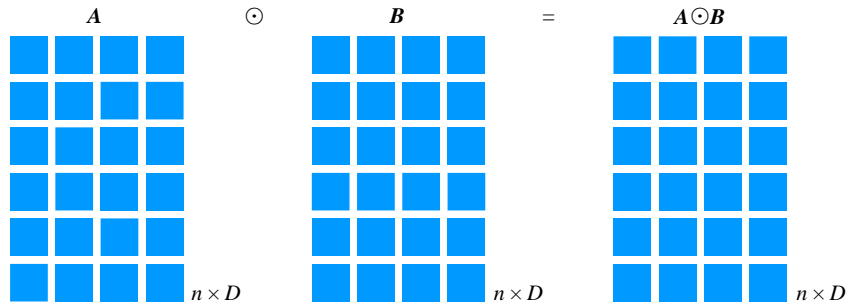
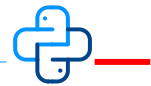


图 15. 矩阵逐项积



Bk4_Ch3_14.py 计算逐项积。

```
# Bk4_Ch3_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_pieewise = np.multiply(A,B)
A_times_B_pieewise_V2 = A*B
```

3.11 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 A 的行列式值可以表达为 $|A|$ 或 $\det(A)$ 。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 A 根据一定的规则映射到一个标量。

一阶方阵的行列式值：

$$|a_{11}| = a_{11} \quad (49)$$

二阶方阵的行列式值：

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21} \quad (50)$$

几何视角

图 16 给出的是二阶矩阵行列式的几何意义。图中蓝色平行四边形的面积就是 A 的行列式值，它可以通过整个矩形面积减去四个三角形面积得到，比如下例：

$$\det(A) = \det \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix} = 3 \times 4 - 2 \times 1 = 10 \quad (51)$$

注意，行列式值可以为零或负值。

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行列式值。因此，行列式值正、负、零都有其特定的几何内涵，本书后续将展开讲解。

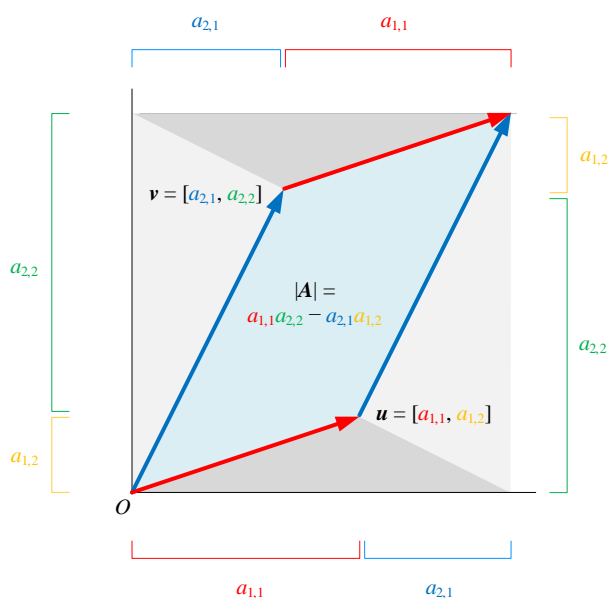


图 16. 二阶矩阵的行列式的几何意义



Bk4_Ch3_15.py 计算行列式值。

```
# Bk4_Ch3_15.py
import numpy as np
A = np.array([[3, 1],
              [2, 4]])

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

三阶方阵行列式值

三阶方阵的行列式值：

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = \begin{vmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} + \begin{vmatrix} 0 & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} + \begin{vmatrix} 0 & 0 & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \quad (52)$$

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

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

更多性质

特别地，对角阵的行列式值为：

$$\begin{vmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} \quad (53)$$

三角阵的行列式值为：

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} \quad (54)$$

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：

$$\begin{aligned} \det(\mathbf{AB}) &= \det(\mathbf{A}) \cdot \det(\mathbf{B}) \\ \det(c\mathbf{A}_{n \times n}) &= c^n \det(\mathbf{A}) \\ \det(\mathbf{A}^T) &= \det(\mathbf{A}) \\ \det(\mathbf{A}^n) &= (\det(\mathbf{A}))^n \\ \det(\mathbf{A}^{-1}) &= \frac{1}{\det(\mathbf{A})} \end{aligned} \quad (55)$$

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：

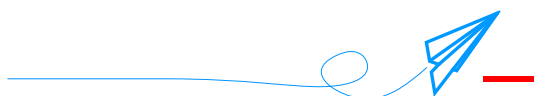
$$\begin{aligned}
 \mathbf{a} \times \mathbf{b} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \\
 &= \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} \mathbf{i} - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} \mathbf{j} + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \mathbf{k} \\
 &= (a_2 b_3 - a_3 b_2) \mathbf{i} + (a_3 b_1 - a_1 b_3) \mathbf{j} + (a_1 b_2 - a_2 b_1) \mathbf{k}
 \end{aligned} \tag{56}$$

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

$$\begin{aligned}
 \mathbf{a} &= -2\mathbf{i} + \mathbf{j} + \mathbf{k} \\
 \mathbf{b} &= \mathbf{i} - 2\mathbf{j} - \mathbf{k}
 \end{aligned} \tag{57}$$

$\mathbf{a} \times \mathbf{b}$ 结果如下：

$$\begin{aligned}
 \mathbf{a} \times \mathbf{b} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -2 & 1 & 1 \\ 1 & -2 & -1 \end{vmatrix} \\
 &= \begin{vmatrix} 1 & 1 \\ -2 & -1 \end{vmatrix} \mathbf{i} - \begin{vmatrix} -2 & 1 \\ 1 & -1 \end{vmatrix} \mathbf{j} + \begin{vmatrix} -2 & 1 \\ 1 & -2 \end{vmatrix} \mathbf{k} \\
 &= \mathbf{i} - \mathbf{j} + 3\mathbf{k}
 \end{aligned} \tag{58}$$



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。