2 乘除

Multiplication and Division

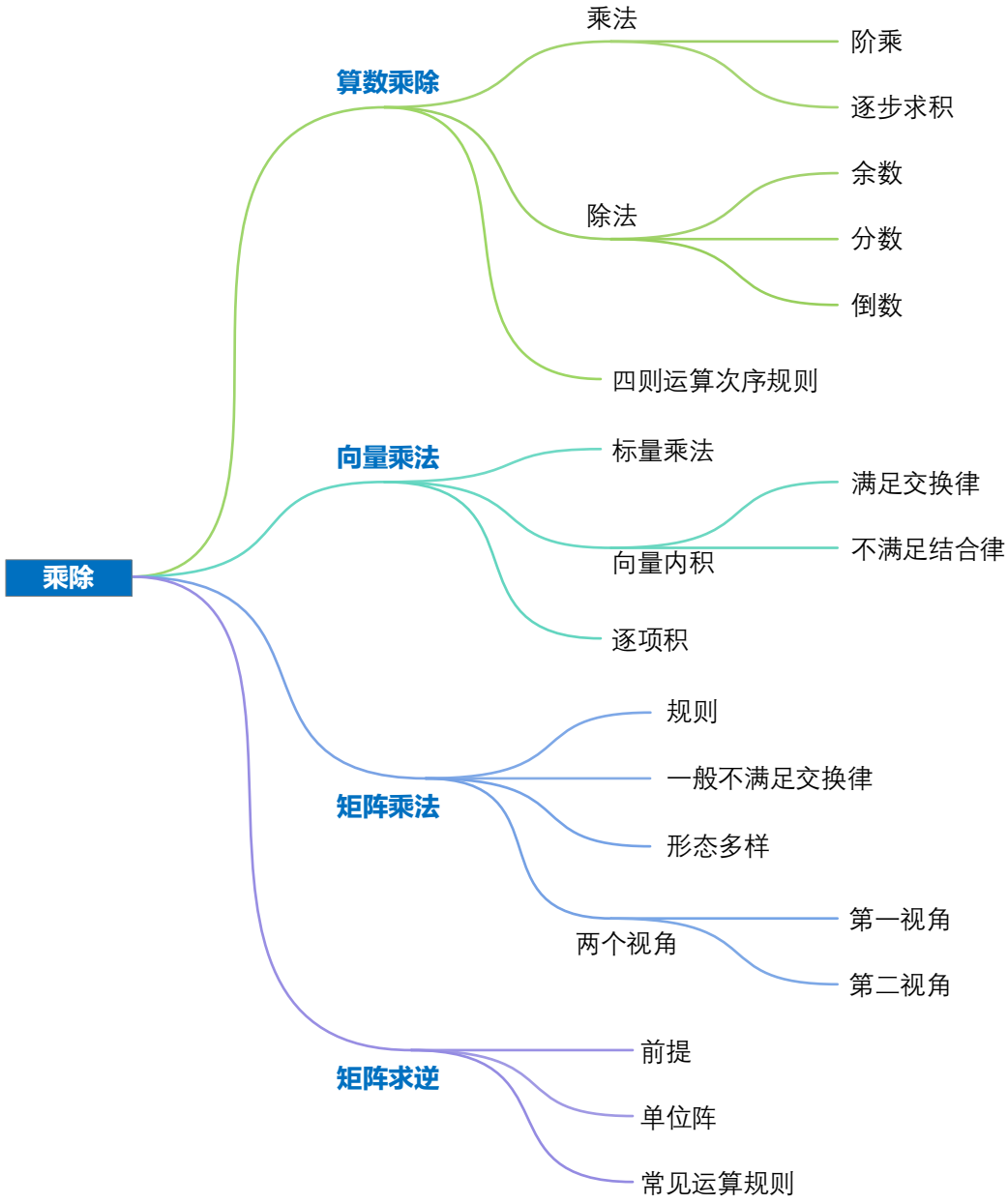
从九九乘法到矩阵乘法

大自然只使用最长的线来编织她的图景；因此，每根织线都能洞见整个大自然的锦绣图景。

Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry.

—— 理查德·费曼 (Richard P. Feynman) | 美国理论物理学家 | 1918 ~ 1988

- `input()` 函数接受一个标准输入数据，返回为字符串 `str` 类型
- `int()` 将输入转化为整数
- `math.factorial()` 计算阶乘
- `numpy.cumprod()` 计算累计乘积
- `numpy.inner()` 计算行向量的内积；函数输入为两个列向量时得到的结果为张量积
- `numpy.linalg.inv()` 计算方阵的逆
- `numpy.linspace()` 在指定的间隔内，返回固定步长的数组
- `numpy.math.factorial()` 计算阶乘
- `numpy.random.seed()` 固定随机数发生器种子
- `numpy.random.uniform()` 产生满足连续均匀分布的随机数
- `numpy.sum()` 求和
- `scipy.special.factorial()` 计算阶乘
- `seaborn.heatmap()` 绘制热图



2.1 算术乘除：先乘除，后加减，括号内先算

本节回顾算术中乘除法。

乘法

乘法 (multiplication) 算式等号左端是**被乘数** (multiplicand) 和**乘数** (multiplier)，右端是**乘积** (product)。乘法运算符读作**乘** (times 或 multiplied by)。**乘法表** (multiplication table 或 times table) 是数字乘法运算的基础。

图 1. 乘法运算

图 2 所示为数轴上可视化 $2 \times 3 = 6$。

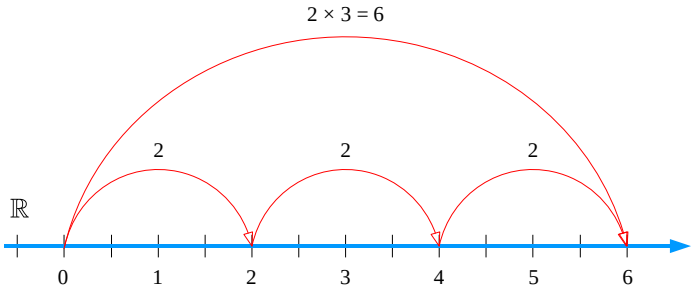


图 2. $2 \times 3 = 6$ 在数轴上的可视化

介绍几个常用乘法符号。乘法符号 × 用于数字相乘，一般不用在两个变量相乘。在线性代数中，× 则表示**叉乘** (cross product 或 vector product)，完全另外一回事。

在代数中，两个变量 a 和 b 相乘，可以写成 ab；这种记法被称作**隐含乘法** (implied multiplication)。ab 也可以写成 $a \cdot b$。

通常，圆点 · 不用在数字相乘，因为它容易和**小数点** (decimal point) 混淆。线性代数中，$\boldsymbol{a} \cdot \boldsymbol{b}$ 表示 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 两个向量的**标量积** (scalar product)，这是本章后续要介绍的内容。

多提一嘴，乘法计算时，请大家多留意数值单位。举个例子，正方形的边长为 1 m，其面积可以通过乘法运算 $1 \times 1 = 1$ 获得；而结果单位为平方米 m^2。有一些数值本身无单位 (unitless)，比如个数、z 分数。z 分数也叫标准分数 (standard score)，是概率统计中的一个概念，z 分数是一个

数与平均数的差再除以标准差的过程。本系列丛书会在《概率统计》一册详细介绍 z 分数这个概念。

表 1 给出和乘法相关的常用英文表达。

表 1. 乘法相关英文表达

数学表达	英文表达
$2 \times 3 = 6$	Two times three equals six. Two multiplied by three equals six. Two cross three equals six. The product of two and three is six. If you multiply two by three, you get six.
$a \cdot b = c$	*a* times *b* equals *c*. *a* multiplied by *b* equals *c*. *a* dot *b* equals *c*. The product of *a* and *b* is *c*.

以下代码完成两个数乘法。Python 中两个数字相乘用 * (asterisk 或 star)。

```
# Bk_Ch2_01

num1 = 2
num2 = 3

# add two numbers
prod = num1*num2

# display the computation
print('The product of {0} and {1} is {2}'.format(num1, num2, prod))
```

阶乘

某个正整数的**阶乘** (factorial) 是所有小于及等于该数的正整数的积。比如，5 的阶乘记做 5!，对应的运算为。

$$5! = 5 \times 4 \times 3 \times 2 \times 1 \tag{1}$$

特别地，定义 0 的阶乘为 0! = 1。本书有两个重要的数学概念需要用到阶乘——排列组合和泰勒展开。

Python 中可以用 `math.factorial()`、`scipy.special.factorial()`、`numpy.math.factorial()` 计算阶乘。为了帮助大家理解，如下代码自定义函数求解阶乘。

```
# Bk_Ch2_02

# define a function to calculate factorial
```

```
num = int(input("Enter an integer: "))

factorial = 1

# check if the number is negative, positive or zero
if num < 0:
   print("Factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is ", factorial)
else:
   for i in range(1,num + 1):
       factorial = factorial*i
   print("The factorial of",num," is ",factorial)
```

累计乘积

对于一组数字，**累计乘积** (cumulative product) 也叫**累积乘积**，得到的结果不仅仅是一个乘积，而是从左向右每乘一个数值得到的分步结果。比如，自然数 1 到 10 求累计求积结果为。

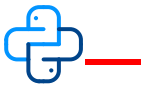
$$1,\quad 2,\quad 6,\quad 24,\quad 120,\quad 720,\quad 5040,\quad 40320,\quad 362880,\quad 3628800 \tag{2}$$

(2) 对应的累计乘积过程如下。

$$\begin{matrix} \overbrace{1\times2\times3\times4\times5\times6=720} \\ \overbrace{1\times2\times3\times4\times5=120} \\ \overbrace{1\times2\times3\times4=24} \\ \overbrace{1\times2\times3=6} \\ \overbrace{1\times2=2} \\ 1 \end{matrix}$$

$$1,\quad 2,\quad 3,\quad 4,\quad 5,\quad 6,\quad 7,\quad 8,\quad \ldots \tag{3}$$

以下代码利用 `numpy.linspace(1, 10, 10)` 产生 1 ~ 10 这 10 个自然数，然后利用 `numpy.cumprod()` 函数来求累计乘积。



```
# Bk_Ch2_03

import numpy as np

a_i = np.linspace(1,10,10)
print(a_i)

a_i_cumprod = np.cumprod(a_i)
np.set_printoptions(suppress=True)
print(a_i_cumprod)
```

除法

除法 (division) 是**乘法的逆运算** (reverse operation of multiplication)。**被除数** (dividend 或 numerator) **除以** (over 或 divided by) **除数** (divisor 或 denominator) 得到**商** (quotient)。

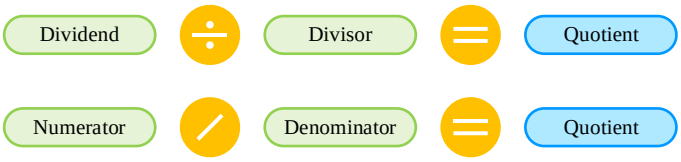


图 3. 除法运算

除法运算有时可以**除尽** (divisible)，比如，**6 可以被 3 除尽** (six is divisible by three)；有时也得到**余数** (remainder)，比如 7 除 2 余 1。除法的结果也可以用**分数** (fraction) 或**小数** (decimal) 来表达。

表 2. 除法英文表达

数学表达	英文表达
$6 \div 3 = 2$	Six divided by three equals two. If you divide six by three you get two.
$7 \div 2 = 3R1$	Seven over two is three and the remainder is one. Seven divided by two equals three with a remainder of one. Seven divided by two equals three and the remainder is one.

以下代码完成两个数的除法运算，除法运算符为正斜杠 /。

```
# Bk_Ch2_04

num1 = 6
num2 = 3

# add two numbers
division = num1/num2

# display the computation
print('The division of {0} over {1} is {2}'.format(num1, num2, division))
```

以下代码用来求余，求余数的运算符为 %。

```
# Bk_Ch2_05

num1 = 7
num2 = 2

# add two numbers
remainder = num1%num2

# display the computation
print('The remainder of {0} over {1} is {2}'.format(num1, num2, remainder))
```

分数

最常见的**分数** (fraction) 是**普通分数** (common fraction 或 simple fraction)，由**分母** (denominator) 和**分子** (numerator) 组成，分隔两者的是**水平线** (fraction bar) 或**正斜杠** (forward slash) /。

非零整数 (nonzero integer) a 的**倒数** (reciprocal) 是 $1/a$。分数 b/a 的倒数是 a/b。

表 3 总结常用分数英文表达。

表 3. 分数相关英文表达

数学表达	英文表达
$\frac{1}{2}$, $1/2$	One half A half One over two
$1:2$	One to two
$-\frac{3}{2}$	Minus three-halves Negative three-halves
$\frac{1}{3}$, $1/3$	One over three One third
$\frac{1}{4}$, $1/4$	One over four One fourth One quarter One divided by four
$1\frac{1}{4}$	One and one fourth
$1/5$	One fifth
$3/5$	Three fifths
$\frac{1}{n}$, $1/n$	One over n
$\frac{a}{b}$, a/b	a over b a divided by b The ratio of a to b The numerator is a while the denominator is b

2.2 向量乘法：标量乘法、向量内积、逐项积

这一节介绍三种重要的向量乘法：(1) **标量乘法** (scalar multiplication)；(2) **向量内积** (inner product)；(3) **逐项积** (piecewise product)。

标量乘法

标量乘法运算中，标量乘向量结果还是向量，相当于缩放。标量乘法运算规则很简单，向量 $\boldsymbol{a}$ 乘以 k，$\boldsymbol{a}$ 的每一个元素均与 k 相乘，比如下例标量 2 乘行向量 [1, 2, 3]。

$$2\times\begin{bmatrix}1 & 2 & 3\end{bmatrix}=\begin{bmatrix}2\times1 & 2\times2 & 2\times3\end{bmatrix}=\begin{bmatrix}2 & 4 & 6\end{bmatrix} \tag{4}$$

再如标量乘列向量。

$$2\times\begin{bmatrix}1\\2\\3\end{bmatrix}=\begin{bmatrix}2\times1\\2\times2\\2\times3\end{bmatrix}=\begin{bmatrix}2\\4\\6\end{bmatrix} \tag{5}$$

图 4 所示为标量乘法示意图。

图 4. 标量乘法

同理，标量 k 乘矩阵 $\boldsymbol{A}$ 结果是 k 与矩阵 $\boldsymbol{A}$ 每一个元素相乘，比如下例。

$$2\times\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}_{2\times3}=\begin{bmatrix}2\times1&2\times2&2\times3\\2\times4&2\times5&2\times6\end{bmatrix}_{2\times3}=\begin{bmatrix}2&4&6\\8&10&12\end{bmatrix}_{2\times3} \tag{6}$$

以下代码完成向量和矩阵标量乘法。

```
# Bk_Ch2_06

import numpy as np

# define a column vector
a_col = np.array([[1], [2], [3]])

b_col = 2*a_col

# define a row vector
a_row = np.array([[1, 2, 3]])

b_row = 2*a_row

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

B = 2*A
```

向量内积

向量内积 (inner product)，又叫**标量积** (scalar product) 或**点积** (dot product)，结果为标量。向量内积的运算规则是，两个形状相同向量，对应位置元素一一相乘，再求和。比如下例计算两个行向量内积。

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\cdot\begin{bmatrix}4 & 3 & 2\end{bmatrix}=1\times4+2\times3+3\times2=4+6+6=16 \tag{7}$$

计算两个列向量内积。

$$\begin{bmatrix}1\\2\\3\end{bmatrix}\cdot\begin{bmatrix}-1\\0\\1\end{bmatrix}=1\times(-1)+2\times0+3\times1=-1+0+3=2 \tag{8}$$

图 5 所示为向量内积规则示意图。

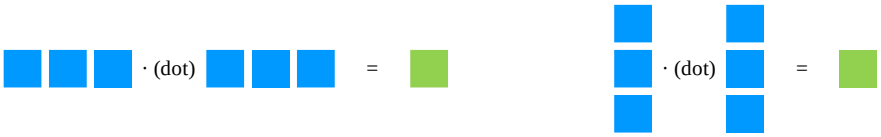


图 5. 向量内积示意图

显然，向量内积满足**交换律** (commutative)，即。

$$\boldsymbol{a}\cdot\boldsymbol{b}=\boldsymbol{b}\cdot\boldsymbol{a} \tag{9}$$

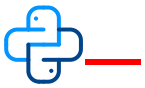
向量内积**对向量加法满足分配律** (distributive over vector addition)。

$$\boldsymbol{a}\cdot(\boldsymbol{b}+\boldsymbol{c})=\boldsymbol{a}\cdot\boldsymbol{b}+\boldsymbol{a}\cdot\boldsymbol{c} \tag{10}$$

显然，向量内积不满足**结合律** (associative)。

$$(\boldsymbol{a}\cdot\boldsymbol{b})\cdot\boldsymbol{c}\neq\boldsymbol{a}\cdot(\boldsymbol{b}\cdot\boldsymbol{c}) \tag{11}$$

注意，`numpy.inner()` 可以用来计算行向量的内积；但是，`numpy.inner()` 函数输入为两个列向量时得到的结果为**张量积** (tensor product)。机器学习和深度学习中，张量积是非常重要的向量运算，本系列丛书将在《矩阵力量》一册会详细介绍。



```
# Bk_Ch2_07

import numpy as np

# row vector dot product

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

a_dot_b = np.inner(a_row, b_row)
print(a_dot_b)

print(np.sum(a_row * b_row))

#%% column vector dot product
```

```
a_col = np.array([[1],  [2], [3]])
b_col = np.array([[-1], [0], [1]])

a_dot_b = np.inner(a_col, b_col)
print(a_dot_b) # tensor product

print(np.sum(a_col * b_col))
```

下面举几个例子，让大家管窥标量积的用途。

给定如下五个数字。

$$1,\quad 2,\quad 3,\quad 4,\quad 5 \tag{12}$$

这五个数字求和，可以用如下标量积计算得到。

$$\begin{bmatrix}1 & 2 & 3 & 4 & 5\end{bmatrix}^{T}\cdot\begin{bmatrix}1 & 1 & 1 & 1 & 1\end{bmatrix}^{T}=1\times1+2\times1+3\times1+4\times1+5\times1=15 \tag{13}$$

前文提过，$[1, 1, 1, 1, 1]^T$ 叫做全 1 向量。

这五个数字的平均值，也可以通过标量积得到。

$$\begin{bmatrix}1 & 2 & 3 & 4 & 5\end{bmatrix}^{T}\cdot\begin{bmatrix}1/5 & 1/5 & 1/5 & 1/5 & 1/5\end{bmatrix}^{T}=1\times1/5+2\times1/5+3\times1/5+4\times1/5+5\times1/5=3 \tag{14}$$

计算五个数字的平方和。

$$\begin{bmatrix}1 & 2 & 3 & 4 & 5\end{bmatrix}^{T}\cdot\begin{bmatrix}1 & 2 & 3 & 4 & 5\end{bmatrix}^{T}=1\times1+2\times2+3\times3+4\times4+5\times5=55 \tag{15}$$

此外，标量积还有重要的几何意义。本书后续将介绍这方面内容。

逐项积

逐项积 (piecewise product)，也叫**阿达玛乘积** (Hadamard product)；两个形状相同向量的逐项积为对应位置元素分别相乘，结果为相同形状的向量。逐项积的运算符为 ⊙。逐项积相当于算术乘法的批量运算。

举个例子，两个行向量逐项积。

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\odot\begin{bmatrix}4 & 5 & 6\end{bmatrix}=\begin{bmatrix}1\times4 & 2\times5 & 3\times6\end{bmatrix}=\begin{bmatrix}4 & 10 & 18\end{bmatrix} \tag{16}$$

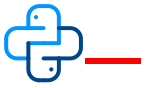
图 6 所示为向量逐项积运算示意图。

图 6. 向量逐项积

同理，两个矩阵逐项积的运算前提——矩阵形状相同；矩阵逐项积运算规则为对应元素相乘，结果形状不变。

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times3} + \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 1 \end{bmatrix}_{2\times3} = \begin{bmatrix} 1\times1 & 2\times2 & 3\times3 \\ 4\times(-1) & 5\times0 & 6\times1 \end{bmatrix}_{2\times3} = \begin{bmatrix} 1 & 4 & 9 \\ -4 & 0 & 6 \end{bmatrix}_{2\times3} \tag{17}$$

Python 中，对于 `numpy.array()` 定义的形状相同的向量或矩阵，逐项积可以通过 `*` 计算得到。



```
# Bk_Ch2_08

import numpy as np
a = np.array([[1, 2, 3]])
b = np.array([[4, 5, 6]])

# calculate element-wise product of row vectors
a_times_b = a*b

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

B = np.array([[1, 2, 3],
              [-1, 0, 1]])

# calculate element-wise product of matrices
A_times_B = A*B
```

2.3 矩阵乘法：最重要的线性代数运算规则

矩阵乘法是最重要线性代数运算，没有之一——这句话并不夸张。

矩阵乘法规则可以视作算术“九九乘法表”的进阶版本。

矩阵乘法规则

A 和 ***B*** 两个矩阵相乘的前提是，矩阵 ***A*** 的列数和矩阵 ***B*** 的行数相同。***A*** 和 ***B*** 的乘积一般写作 ***AB***；注意，***A*** 在左边，***B*** 在右边，不能随意改变顺序。

A 和 ***B*** 两个矩阵相乘 ***AB*** 读作“matrix boldface capital ***A*** times matrix boldface capital ***B***”或“the matrix product boldface capital ***A*** and boldface capital ***B***”。

NumPy 中，两个矩阵相乘的运算符为 @，本系列丛书公式会采用同样矩阵乘法运算符；因此，***AB*** 也常记做 ***A@B***。

$$\boldsymbol{C}_{m\times n} = \boldsymbol{A}_{m\times p}\boldsymbol{B}_{p\times n} = \boldsymbol{A}_{m\times p} @ \boldsymbol{B}_{p\times n} \tag{18}$$

其中，矩阵 ***A*** 的形状为 m 行、p 列，矩阵 ***B*** 的形状为 p 行、n 列。

$\boldsymbol{A}$ 和 $\boldsymbol{B}$ 相乘得到矩阵 $\boldsymbol{C}$，$\boldsymbol{C}$ 的形状为 m 行、n 列，相当于消去了 p。

特别注意，矩阵乘法不满足交换律；也就是，一般情况下式不成立。

$$\boldsymbol{A}_{m\times p}\boldsymbol{B}_{p\times n} \neq \boldsymbol{B}_{p\times n}\boldsymbol{A}_{m\times p} \tag{19}$$

首先，$\boldsymbol{B}$ 的列数和 $\boldsymbol{A}$ 的行数很可能不匹配；即便 $m = n$，也就是 $\boldsymbol{B}$ 的列数等于 $\boldsymbol{A}$ 的行数，$\boldsymbol{BA}$ 结果很可能不等于 $\boldsymbol{AB}$。再次强调，在矩阵乘法中不能随意改变矩阵顺序。

两个 2 × 2 矩阵相乘

下面，用两个 2 × 2 矩阵相乘讲解矩阵乘法运算规则；下式中，矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 相乘结果为矩阵 $\boldsymbol{C}$。

$$\boldsymbol{C} = \boldsymbol{AB} = \underbrace{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}}_{\boldsymbol{B}} = \underbrace{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}_{\boldsymbol{A}} @ \underbrace{\begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}}_{\boldsymbol{B}} \tag{20}$$

图 7 所示为两个 2 × 2 矩阵相乘如何得到矩阵 $\boldsymbol{C}$ 的每一个元素。

矩阵 $\boldsymbol{A}$ 的第一行元素和矩阵 $\boldsymbol{B}$ 第一列对应元素分别相乘，再相加，结果为矩阵 $\boldsymbol{C}$ 的第一行、第一列元素 $c_{1,1}$。

矩阵 $\boldsymbol{A}$ 的第一行元素和矩阵 $\boldsymbol{B}$ 第二列对应元素分别相乘，再相加，得到 $c_{1,2}$。

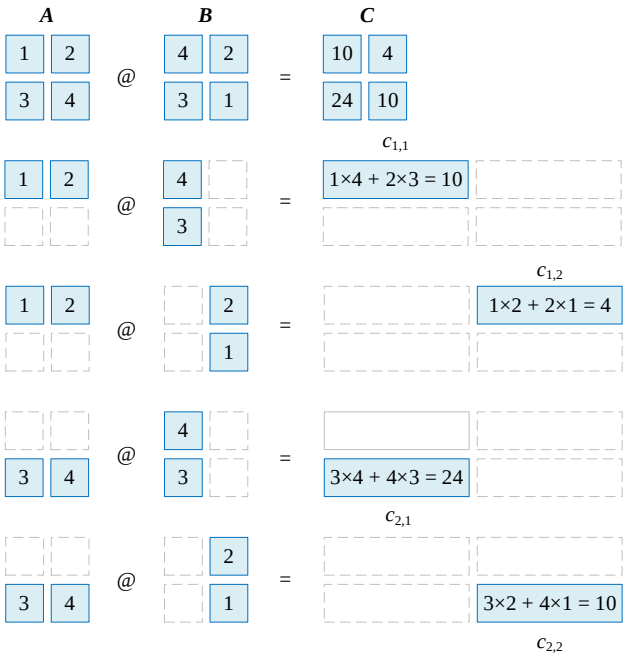


图 7. 矩阵乘法规则，两个 2 × 2 矩阵相乘为例

同理，依次获得矩阵 $\boldsymbol{C}$ 的 $c_{2,1}$ 和 $c_{2,2}$ 两个元素。

总结来说，矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 乘积 $\boldsymbol{C}$ 的第 i 行第 j 列的元素等于矩阵 $\boldsymbol{A}$ 的第 i 行的元素与矩阵 $\boldsymbol{B}$ 的第 j 列对应元素乘积再求和。

注意，这个矩阵运算规则既是一种发明创造，也是一种约定成俗；也就是说，这种乘法规则在被法国数学家雅克·菲利普·玛丽·比内 (Jacques Philippe Marie Binet, 1786 ~ 1856) 提出之后，在长期的数学实践中被广为接受。

就像大家儿时背诵九九乘法表时，这里建议大家先把矩阵乘法规则背下来。慢慢的，大家就会通过不断学习认识到这个乘法规则的精妙之处。

以下代码展示如何完成 (20) 矩阵乘法运算。

```
# Bk_Ch2_09

import numpy as np

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

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

# calculate matrix multiplication
C = A@B
```

矩阵乘法形态

图 8 给出了常见的多种矩阵乘法形态，每一种形态对应一类线性代数问题。图 8 特别高亮矩阵乘法中左侧矩阵的列和右侧矩阵的行。

本系列丛书《矩阵力量》一册将会详细介绍图 8 每一种乘法形态。这里格外提醒大家，初学者对线性代数会产生一种错误印象，这些千奇百怪的矩阵乘法规则就是“奇技淫巧”；这是极其错误的想法。在不断学习中，大家会逐渐领略到每种矩阵乘法形态的力量所在。

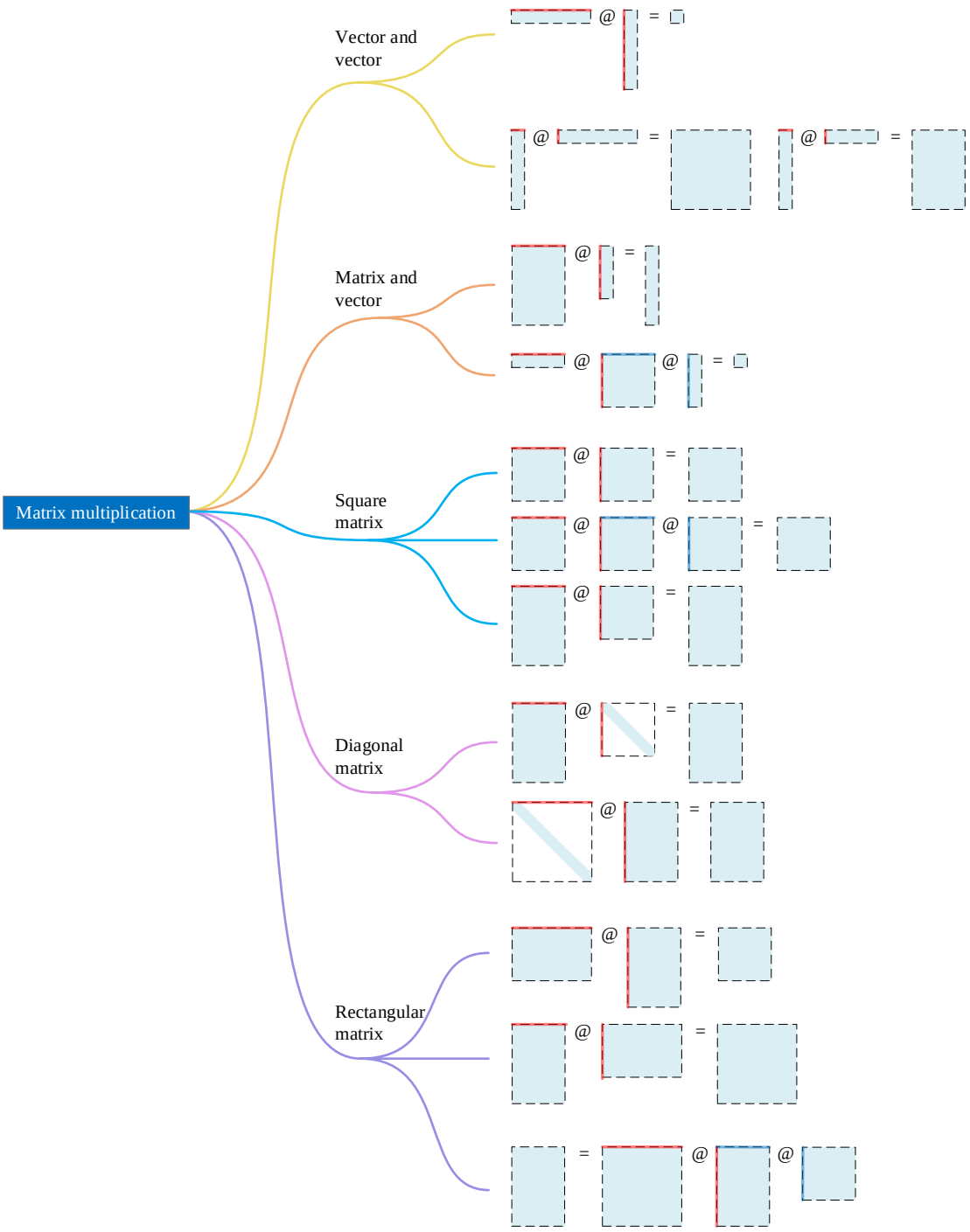


图 8. 矩阵乘法形态多样性

两个向量相乘

本节最后着重讲一下图 8 最顶上两种向量乘积。

向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 为等长列向量，$\boldsymbol{a}$ 转置 ($\boldsymbol{a}^T$) 乘 $\boldsymbol{b}$ 为标量，等价于 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的标量积。

$$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}=\boldsymbol{a}\cdot\boldsymbol{b} \tag{21}$$

举个例子。

$$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}=\begin{bmatrix}1\\2\\3\end{bmatrix}_{3\times1}^{\mathrm{T}}@\begin{bmatrix}4\\3\\2\end{bmatrix}_{3\times1}=\begin{bmatrix}1&2&3\end{bmatrix}_{1\times3}@\begin{bmatrix}4\\3\\2\end{bmatrix}_{3\times1}=\begin{bmatrix}1&2&3\end{bmatrix}\cdot\begin{bmatrix}4&3&2\end{bmatrix}=16 \tag{22}$$

列向量 $\boldsymbol{a}$ 乘 $\boldsymbol{b}$ 转置 ($\boldsymbol{b}^{\mathrm{T}}$)乘积为方阵，也就是行数和列数相同的矩阵。

$$\boldsymbol{a}\boldsymbol{b}^{\mathrm{T}}=\begin{bmatrix}1\\2\\3\end{bmatrix}_{3\times1}@\begin{bmatrix}4\\3\\2\end{bmatrix}_{3\times1}^{\mathrm{T}}=\begin{bmatrix}1\\2\\3\end{bmatrix}_{3\times1}@\begin{bmatrix}4&3&2\end{bmatrix}_{1\times3}=\begin{bmatrix}4&3&2\\8&6&4\\12&9&6\end{bmatrix}_{3\times3} \tag{23}$$

这两种特殊形态的矩阵乘法正是理解矩阵乘法规则的两个重要视角，这是下两节要介绍的内容。

2.4 矩阵乘法第一视角

这一节探讨矩阵乘法的第一视角。

上一节最后介绍，$\boldsymbol{a}$ 和 $\boldsymbol{b}$ 为等长列向量，$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}$ 结果标量，相当于标量积 $\boldsymbol{a}\cdot\boldsymbol{b}$。我们可以把 (20) 中 $\boldsymbol{A}$ 写成两个行向量 $\boldsymbol{a}^{(1)}$ 和 $\boldsymbol{a}^{(2)}$，把 $\boldsymbol{B}$ 写成两个列向量 $\boldsymbol{b}_1$ 和 $\boldsymbol{b}_2$，即。

$$\boldsymbol{A}=\begin{bmatrix}\begin{bmatrix}1&2\end{bmatrix}\\ \underbrace{\begin{bmatrix}3&4\end{bmatrix}}_{\boldsymbol{a}^{(2)}}\end{bmatrix}=\begin{bmatrix}\boldsymbol{a}^{(1)}\\ \boldsymbol{a}^{(2)}\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}\underbrace{\begin{bmatrix}4\\3\end{bmatrix}}_{\boldsymbol{b}_1}&\underbrace{\begin{bmatrix}2\\1\end{bmatrix}}_{\boldsymbol{b}_2}\end{bmatrix}=\begin{bmatrix}\boldsymbol{b}_1&\boldsymbol{b}_2\end{bmatrix} \tag{24}$$

这样 $\boldsymbol{AB}$ 矩阵乘积可以写成。

$$\boldsymbol{A}@\boldsymbol{B}=\underbrace{\begin{bmatrix}\begin{bmatrix}1&2\end{bmatrix}\\ \underbrace{\begin{bmatrix}3&4\end{bmatrix}}_{\boldsymbol{a}^{(2)}}\end{bmatrix}}_{\boldsymbol{A}}@\underbrace{\begin{bmatrix}\underbrace{\begin{bmatrix}4\\3\end{bmatrix}}_{\boldsymbol{b}_1}&\underbrace{\begin{bmatrix}2\\1\end{bmatrix}}_{\boldsymbol{b}_2}\end{bmatrix}}_{\boldsymbol{B}}=\begin{bmatrix}\begin{bmatrix}1&2\end{bmatrix}@\begin{bmatrix}4\\3\end{bmatrix}&\begin{bmatrix}1&2\end{bmatrix}@\begin{bmatrix}2\\1\end{bmatrix}\\ \begin{bmatrix}3&4\end{bmatrix}@\begin{bmatrix}4\\3\end{bmatrix}&\begin{bmatrix}3&4\end{bmatrix}@\begin{bmatrix}2\\1\end{bmatrix}\end{bmatrix}=\begin{bmatrix}10&4\\24&10\end{bmatrix} \tag{25}$$

也就是说，将位于矩阵乘法左侧的 $\boldsymbol{A}$ 写成行向量，右侧的 $\boldsymbol{B}$ 写成列向量；然后，行向量和列向量逐步相乘，得到乘积每个位置的元素。

用符号代替具体数字，(25) 可以写成。

$$\begin{aligned}\boldsymbol{A} \text{@} \boldsymbol{B} &= \begin{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} \end{bmatrix}_{1\times 2} \\ \begin{bmatrix} a_{2,1} & a_{2,2} \end{bmatrix}_{1\times 2} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} b_{1,1} \\ b_{2,1} \end{bmatrix}_{2\times 1} & \begin{bmatrix} b_{1,2} \\ b_{2,2} \end{bmatrix}_{2\times 1} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{a}^{(1)} \\ \boldsymbol{a}^{(2)} \end{bmatrix}_{2\times 1} \begin{bmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 \end{bmatrix}_{1\times 2} = \begin{bmatrix} \boldsymbol{a}^{(1)}\boldsymbol{b}_1 & \boldsymbol{a}^{(1)}\boldsymbol{b}_2 \\ \boldsymbol{a}^{(2)}\boldsymbol{b}_1 & \boldsymbol{a}^{(2)}\boldsymbol{b}_2 \end{bmatrix}_{2\times 2} = \begin{bmatrix} \left(\boldsymbol{a}^{(1)}\right)^{\mathrm{T}} \cdot \boldsymbol{b}_1 & \left(\boldsymbol{a}^{(1)}\right)^{\mathrm{T}} \cdot \boldsymbol{b}_2 \\ \left(\boldsymbol{a}^{(2)}\right)^{\mathrm{T}} \cdot \boldsymbol{b}_1 & \left(\boldsymbol{a}^{(2)}\right)^{\mathrm{T}} \cdot \boldsymbol{b}_2 \end{bmatrix}_{2\times 2}\end{aligned} \tag{26}$$

这是矩阵乘法的基本视角，它直接体现的是矩阵乘法规则。

更一般地，矩阵乘积 $\boldsymbol{AB}$ 中，左侧矩阵 $\boldsymbol{A}$ 的形状为 $m \times p$，将矩阵 $\boldsymbol{A}$ 写成一组上下叠放的行向量；行向量 $\boldsymbol{a}^{(i)}$ 列数为 p，即有 p 个元素。

$$\boldsymbol{A}_{m\times p} = \begin{bmatrix} \boldsymbol{a}^{(1)} \\ \boldsymbol{a}^{(1)} \\ \vdots \\ \boldsymbol{a}^{(m)} \end{bmatrix}_{m\times 1} \tag{27}$$

矩阵乘积 $\boldsymbol{AB}$ 中，右侧矩阵 $\boldsymbol{B}$ 的形状为 $p \times n$ 列，将矩阵 $\boldsymbol{B}$ 写成左右排列的列向量。列向量 $\boldsymbol{a}^{(i)}$ 行数为 p，也有 p 个元素。

$$\boldsymbol{B}_{p\times n} = \begin{bmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_n \end{bmatrix}_{1\times n} \tag{28}$$

$\boldsymbol{A}$ 和 $\boldsymbol{B}$ 相乘，可以展开写成。

$$\boldsymbol{A}_{m\times p} \text{@} \boldsymbol{B}_{p\times n} = \begin{bmatrix} \boldsymbol{a}^{(1)} \\ \boldsymbol{a}^{(2)} \\ \vdots \\ \boldsymbol{a}^{(m)} \end{bmatrix}_{m\times 1} \begin{bmatrix} \boldsymbol{b}_1 & \boldsymbol{b}_2 & \cdots & \boldsymbol{b}_n \end{bmatrix}_{1\times n} = \begin{bmatrix} \boldsymbol{a}^{(1)}\boldsymbol{b}_1 & \boldsymbol{a}^{(1)}\boldsymbol{b}_2 & \cdots & \boldsymbol{a}^{(1)}\boldsymbol{b}_n \\ \boldsymbol{a}^{(2)}\boldsymbol{b}_1 & \boldsymbol{a}^{(2)}\boldsymbol{b}_2 & \cdots & \boldsymbol{a}^{(2)}\boldsymbol{b}_n \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{a}^{(m)}\boldsymbol{b}_1 & \boldsymbol{a}^{(m)}\boldsymbol{b}_2 & \cdots & \boldsymbol{a}^{(m)}\boldsymbol{b}_n \end{bmatrix}_{m\times n} = \boldsymbol{C}_{m\times n} \tag{29}$$

图 9 所示为**热图** (heatmap) 可视化矩阵乘法。

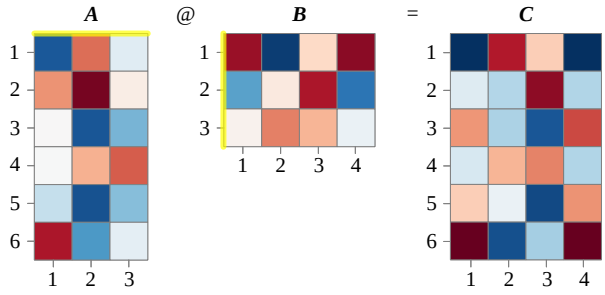


图 9. 矩阵乘法热图展示

具体如图 10 所示，$\boldsymbol{A}$ 中的第 i 行向量 $\boldsymbol{a}^{(i)}$ 乘以 $\boldsymbol{B}$ 中第 j 列向量 $\boldsymbol{b}_j$，得到标量 $\boldsymbol{a}^{(i)}\boldsymbol{b}_j$，对应乘积矩阵 $\boldsymbol{C}$ 中第 i 行、第 j 列元素 $c_{i,j}$。

$$c_{i,j} = \boldsymbol{a}^{(i)}\boldsymbol{b}_j \tag{30}$$

这就是矩阵乘法的第一视角。

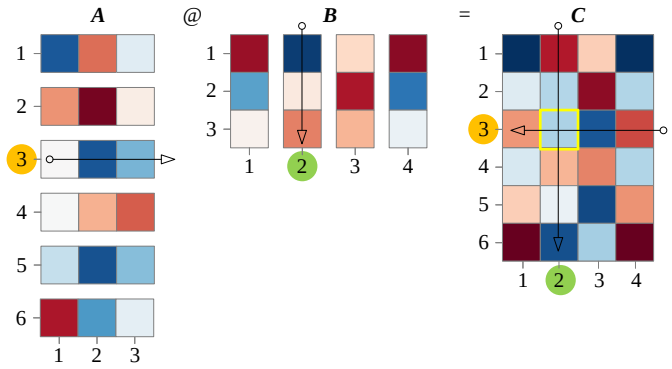


图 10. 矩阵乘法第一视角

以下代码可以用来绘制图 9。代码用 `numpy.random.uniform()` 函数产生满足连续均匀分布的随机数；并用 `seaborn.heatmap()` 绘制热图。

```
# Bk_Ch2_10_A

import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

# Repeatability
np.random.seed(7)

# Generate matrix A and B
m = 6
p = 3
n = 4

A = np.random.uniform(-1,1,m*p).reshape(m, p)
B = np.random.uniform(-1,1,p*n).reshape(p, n)

C = A@B

all_max = 1
all_min = -1

#%% matrix multiplication, first perspective

fig, axs = plt.subplots(1, 5, figsize=(12, 3))

plt.sca(axs[0])
ax = sns.heatmap(A,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,m+1), xticklabels=np.arange(1,p+1))
ax.set_aspect("equal")
plt.title('$A$')
plt.yticks(rotation=0)
```

```
plt.sca(axs[1])
plt.title('$@$')
plt.axis('off')

plt.sca(axs[2])
ax = sns.heatmap(B,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,p+1), xticklabels=np.arange(1,n+1))
ax.set_aspect("equal")
plt.title('$B$')
plt.yticks(rotation=0)

plt.sca(axs[3])
plt.title('$=$')
plt.axis('off')

plt.sca(axs[4])
ax = sns.heatmap(C,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,m+1), xticklabels=np.arange(1,n+1))
ax.set_aspect("equal")
plt.title('$C$')
plt.yticks(rotation=0)
```

2.5 矩阵乘法第二视角

下面，我们聊一聊矩阵乘法的第二视角。

还是以 (20) 为例，$\boldsymbol{A}$ 和 $\boldsymbol{B}$ 相乘，左侧矩阵 $\boldsymbol{A}$ 写成两个列向量 $\boldsymbol{a}_1$ 和 $\boldsymbol{a}_2$，把 $\boldsymbol{B}$ 写成两个列向量 $\boldsymbol{b}^{(1)}$和 $\boldsymbol{b}^{(2)}$。

$$\boldsymbol{A}=\begin{bmatrix}\underbrace{\begin{bmatrix}1\\3\end{bmatrix}}_{\boldsymbol{a}_1} & \underbrace{\begin{bmatrix}2\\4\end{bmatrix}}_{\boldsymbol{a}_2}\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}\underbrace{\begin{bmatrix}4 & 2\end{bmatrix}}_{\boldsymbol{b}^{(1)}}\\ \underbrace{\begin{bmatrix}3 & 1\end{bmatrix}}_{\boldsymbol{b}^{(2)}}\end{bmatrix} \tag{31}$$

这样 $\boldsymbol{AB}$ 乘积可以展开写成。

$$\boldsymbol{A}@\boldsymbol{B}=\underbrace{\begin{bmatrix}\begin{bmatrix}1\\3\end{bmatrix} & \begin{bmatrix}2\\4\end{bmatrix}\end{bmatrix}}_{\boldsymbol{A}}@\underbrace{\begin{bmatrix}\underbrace{\begin{bmatrix}4 & 2\end{bmatrix}}_{\boldsymbol{b}^{(1)}}\\ \underbrace{\begin{bmatrix}3 & 1\end{bmatrix}}_{\boldsymbol{b}^{(2)}}\end{bmatrix}}_{\boldsymbol{B}}=\underbrace{\begin{bmatrix}1\\3\end{bmatrix}}_{\boldsymbol{a}_1}@\underbrace{\begin{bmatrix}4 & 2\end{bmatrix}}_{\boldsymbol{b}^{(1)}}+\underbrace{\begin{bmatrix}2\\4\end{bmatrix}}_{\boldsymbol{a}_2}@\underbrace{\begin{bmatrix}3 & 1\end{bmatrix}}_{\boldsymbol{b}^{(2)}}=\begin{bmatrix}4 & 2\\12 & 6\end{bmatrix}+\begin{bmatrix}6 & 2\\12 & 4\end{bmatrix}=\begin{bmatrix}10 & 4\\24 & 10\end{bmatrix} \tag{32}$$

在这个视角下，我们惊奇发现矩阵乘法竟然变成了“加法”。

用符号代替数字，(32) 可以写成。

$$\begin{aligned}\boldsymbol{A}@\boldsymbol{B}&=\begin{bmatrix}\begin{bmatrix}a_{1,1}\\a_{2,1}\end{bmatrix}_{2\times1} & \begin{bmatrix}a_{1,2}\\a_{2,2}\end{bmatrix}_{2\times1}\end{bmatrix}\begin{bmatrix}\begin{bmatrix}b_{1,1} & b_{1,2}\end{bmatrix}_{1\times2}\\ \begin{bmatrix}b_{2,1} & b_{2,2}\end{bmatrix}_{1\times2}\end{bmatrix}\\ &=\begin{bmatrix}\boldsymbol{a}_1 & \boldsymbol{a}_2\end{bmatrix}_{1\times2}\begin{bmatrix}\boldsymbol{b}^{(1)}\\ \boldsymbol{b}^{(2)}\end{bmatrix}_{2\times1}=\boldsymbol{a}_1\boldsymbol{b}^{(1)}+\boldsymbol{a}_2\boldsymbol{b}^{(2)}\end{aligned} \tag{33}$$

更一般地，将矩阵 $\boldsymbol{A}_{m\times p}$ 写成一系列左右排列的列向量；这些列向量元素数量为 m，即行数为 m。

$$\boldsymbol{A}_{m\times p}=\begin{bmatrix}\boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_p\end{bmatrix}_{1\times p} \tag{34}$$

将矩阵 $\boldsymbol{B}_{p\times n}$ 写成上下叠放的行向量；这些行向量元素数量为 m，即列数为 m。

$$\boldsymbol{B}_{p\times n}=\begin{bmatrix}\boldsymbol{b}^{(1)}\\ \boldsymbol{b}^{(2)}\\ \vdots\\ \boldsymbol{b}^{(p)}\end{bmatrix}_{p\times 1} \tag{35}$$

矩阵 $\boldsymbol{A}$ 和矩阵 $\boldsymbol{B}$ 相乘，可以展开写成 p 个 $m\times n$ 矩阵相加。

$$\boldsymbol{A}_{m\times p}@\boldsymbol{B}_{p\times n}=\begin{bmatrix}\boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_p\end{bmatrix}_{1\times p}\begin{bmatrix}\boldsymbol{b}^{(1)}\\ \boldsymbol{b}^{(2)}\\ \vdots\\ \boldsymbol{b}^{(p)}\end{bmatrix}_{p\times 1}=\underbrace{\boldsymbol{a}_1\boldsymbol{b}^{(1)}+\boldsymbol{a}_2\boldsymbol{b}^{(2)}+\cdots\boldsymbol{a}_p\boldsymbol{b}^{(p)}}_{p\text{ matrices with shape of }m\times n}=\boldsymbol{C}_{m\times n} \tag{36}$$

我们可以把 $\boldsymbol{a}_k\boldsymbol{b}^{(k)}$ 结果矩阵写成 $\boldsymbol{C}_k$，这样 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 乘积 $\boldsymbol{C}$ 可以写成 $\boldsymbol{C}_k$ (k = 1, 2, …, p) 之和。

$$\boldsymbol{a}_1\boldsymbol{b}^{(1)}+\boldsymbol{a}_2\boldsymbol{b}^{(2)}+\cdots\boldsymbol{a}_p\boldsymbol{b}^{(p)}=\boldsymbol{C}_1+\boldsymbol{C}_2+\cdots\boldsymbol{C}_p=\boldsymbol{C} \tag{37}$$

图 11 给出的是图 10 所示矩阵乘法第二视角的热图。图中三个形状相同矩阵 $\boldsymbol{C}_1$、$\boldsymbol{C}_2$、$\boldsymbol{C}_3$ 相加得到 $\boldsymbol{C}$。

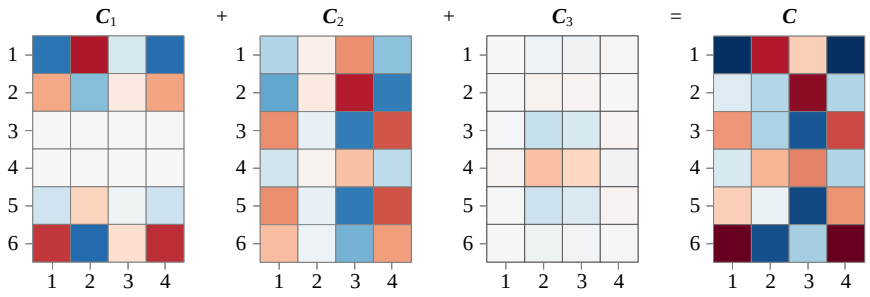


图 11. 矩阵乘法第二视角

如图 12 所示，从图像角度 (37) 好比若干形状相同的图片，经过层层叠加，最后获得一幅完整热图。

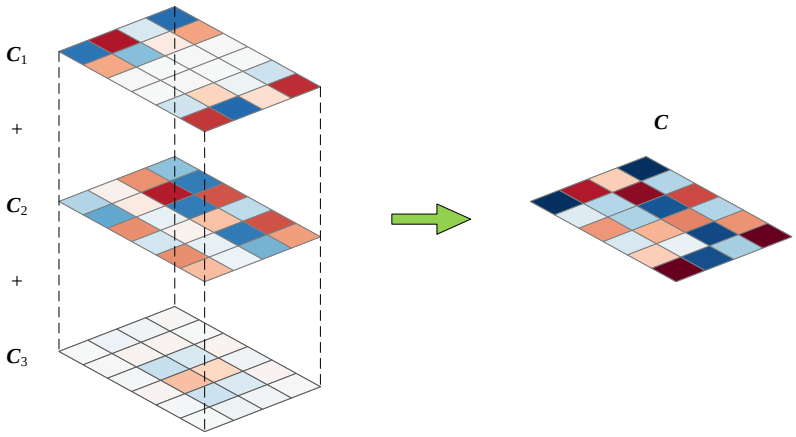


图 12. 三幅图像叠加得到矩阵 C 热图

图 13、图 14、图 15 分别展示如何获得图 11 中矩阵 C_1、C_2、C_3 热图。观察热图可以发现一个有意思的现象，列向量乘行向量得到一个矩阵；打个比方，两个向量相乘好像张起了一幅平面。张量积用的就是类似图 13、图 14、图 15 的运算思路。本系列丛书《矩阵力量》一册会讲解张量积。

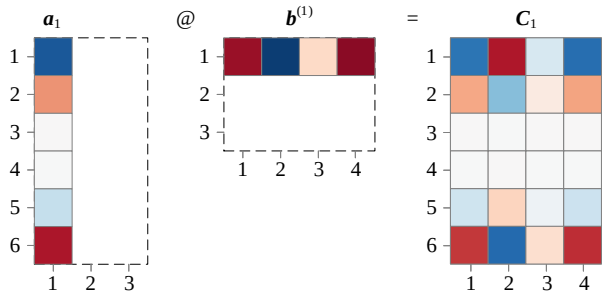


图 13. 获得 C_1

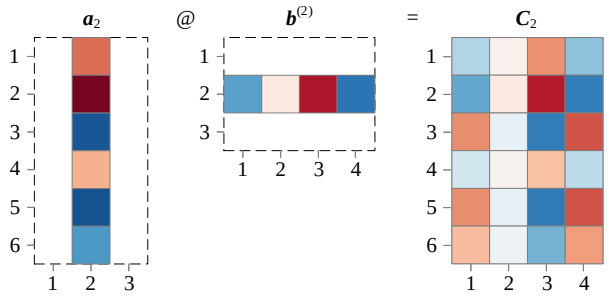


图 14. 获得 C_2

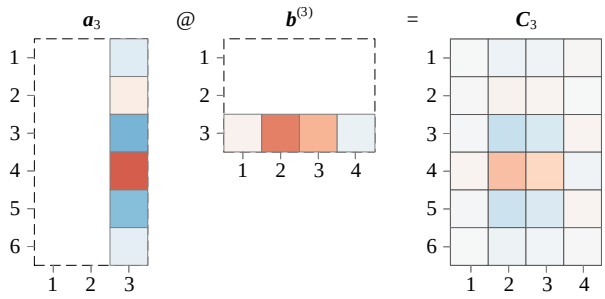


图 15. 获得 C_3

在这个视角下，矩阵的乘法变成若干矩阵的叠加；这是一个非常重要的视角，数学科学和机器学习很多算法都会离不开它。

配合前文代码，如下代码绘制图 11。

```
# Bk_Ch2_10_B

#%% matrix multiplication, second perspective

C1 = A[:,[0]]@B[[0],:]

C2 = A[:,[1]]@B[[1],:]

C3 = A[:,[2]]@B[[2],:]

fig, axs = plt.subplots(1, 7, figsize=(12, 3))

plt.sca(axs[0])
ax = sns.heatmap(C1,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,m+1), xticklabels=np.arange(1,n+1))
ax.set_aspect("equal")
plt.title('$C_1$')
plt.yticks(rotation=0)

plt.sca(axs[1])
plt.title('$@$')
plt.axis('off')

plt.sca(axs[2])
ax = sns.heatmap(C2,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,m+1), xticklabels=np.arange(1,n+1))
ax.set_aspect("equal")
plt.title('$C_2$')
plt.yticks(rotation=0)

plt.sca(axs[3])
plt.title('$@$')
plt.axis('off')

plt.sca(axs[4])
ax = sns.heatmap(C3,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,m+1), xticklabels=np.arange(1,n+1))
ax.set_aspect("equal")
```

```
plt.title('$C_3$')
plt.yticks(rotation=0)

plt.sca(axs[5])
plt.title('$=$')
plt.axis('off')

plt.sca(axs[6])
ax = sns.heatmap(C,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,m+1), xticklabels=np.arange(1,n+1))
ax.set_aspect("equal")
plt.title('$C$')
plt.yticks(rotation=0)
```

2.6 矩阵除法：计算逆矩阵

实际上，并不存在矩阵除法；所谓矩阵 $\boldsymbol{B}$ 除以矩阵 $\boldsymbol{A}$，实际上将矩阵 $\boldsymbol{A}$ 先转化逆矩阵 $\boldsymbol{A}^{-1}$，然后计算 $\boldsymbol{B}$ 和逆矩阵 $\boldsymbol{A}^{-1}$ 乘积。

$$\boldsymbol{B}\boldsymbol{A}^{-1} = \boldsymbol{B} @ \boldsymbol{A}^{-1} \tag{38}$$

$\boldsymbol{A}$ 如果**可逆** (invertible)，仅当 $\boldsymbol{A}$ 为方阵且存在矩阵 $\boldsymbol{A}^{-1}$ 使得下式成立。

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I} \tag{39}$$

$\boldsymbol{A}^{-1}$ 叫做**矩阵 A 的逆** (the inverse of matrix $\boldsymbol{A}$)。

(39) 中的 $\boldsymbol{I}$ 就是前文介绍过的**单位阵** (identity matrix)。**n 阶单位矩阵** (n-square identity matrix 或 n-square unit matrix) 的特点是对角线上的元素为 1，其他为 0。

$$\boldsymbol{I}_{n\times n} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{40}$$

可以用 `numpy.linalg.inv()` 计算方阵的逆。图 16 所示为方阵 $\boldsymbol{A}$ 和逆矩阵 $\boldsymbol{A}^{-1}$ 相乘得到单位矩阵 $\boldsymbol{I}$ 的热图。

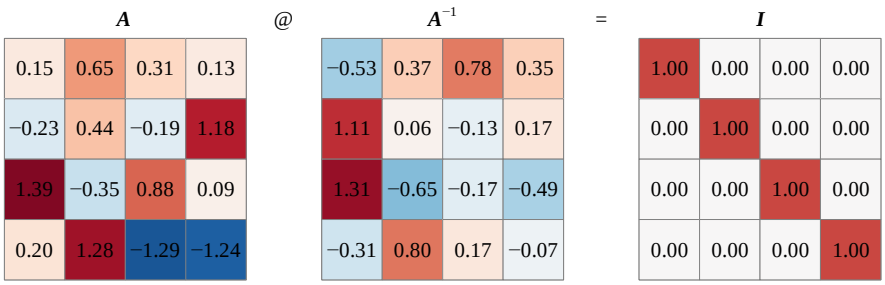


图 16. 方阵 $\boldsymbol{A}$ 和逆矩阵 $\boldsymbol{A}^{-1}$ 相乘

一般情况。

$$\left(\boldsymbol{A}+\boldsymbol{B}\right)^{-1} \neq \boldsymbol{A}^{-1}+\boldsymbol{B}^{-1} \tag{41}$$

请大家注意以下和矩阵逆有关的运算规则。

$$\begin{aligned}
\left(\boldsymbol{A}^{\mathrm{T}}\right)^{-1} &= \left(\boldsymbol{A}^{-1}\right)^{\mathrm{T}} \\
\left(\boldsymbol{AB}\right)^{-1} &= \boldsymbol{B}^{-1}\boldsymbol{A}^{-1} \\
\left(\boldsymbol{ABC}\right)^{-1} &= \boldsymbol{C}^{-1}\boldsymbol{B}^{-1}\boldsymbol{A}^{-1} \\
\left(k\boldsymbol{A}\right)^{-1} &= \frac{1}{k}\boldsymbol{A}^{-1}
\end{aligned} \tag{42}$$

其中，假设 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$、$\boldsymbol{AB}$ 和 $\boldsymbol{ABC}$ 逆运算存在。表 4 总结常见矩阵逆相关的英文表达。

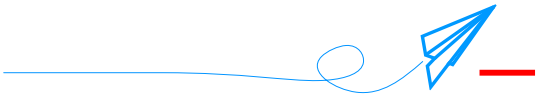
表 4. 和矩阵逆相关的英文表达

数学表达	英文表达
$\boldsymbol{A}^{-1}$	Inverse of the matrix boldface capital $\boldsymbol{A}$ Matrix boldface capital $\boldsymbol{A}$ inverse
$(\boldsymbol{A}+\boldsymbol{B})^{-1}$	Left parenthesis bold face capital $\boldsymbol{A}$ plus boldface capital $\boldsymbol{B}$ right parenthesis superscript minus one Inverse of the matrix sum boldface capital $\boldsymbol{A}$ plus boldface capital $\boldsymbol{B}$
$(\boldsymbol{AB})^{-1}$	Left parenthesis boldface capital $\boldsymbol{A}$ times boldface capital $\boldsymbol{B}$ right parenthesis superscript minus one Inverse of the matrix product boldface capital $\boldsymbol{A}$ and boldface capital $\boldsymbol{B}$
$\boldsymbol{ABC}^{-1}$	The product boldface capital $\boldsymbol{A}$ boldface capital $\boldsymbol{B}$ and boldface capital $\boldsymbol{C}$ inverse

以下代码计算并绘制图 16。

```
# Bk_Ch2_11

import numpy as np
from numpy.linalg import inv
from matplotlib import pyplot as plt
import seaborn as sns

# Repeatability
np.random.seed(0)
# Generate matrix A
n = 4
A = np.random.uniform(-1.5,1.5,n*n).reshape(n, n)

all_max = 1.5
all_min = -1.5

# matrix inverse
```

```
A_inverse = inv(A)

fig, axs = plt.subplots(1, 5, figsize=(12, 3))

plt.sca(axs[0])
ax = sns.heatmap(A,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,n+1), xticklabels=np.arange(1,n+1),
                 annot = True,fmt=".2f")
ax.set_aspect("equal")
plt.title('$A$')
plt.yticks(rotation=0)

plt.sca(axs[1])
plt.title('$@$')
plt.axis('off')

plt.sca(axs[2])
ax = sns.heatmap(A_inverse,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,n+1), xticklabels=np.arange(1,n+1),
                 annot = True,fmt=".2f")
ax.set_aspect("equal")
plt.title('$A^{-1}$')
plt.yticks(rotation=0)

plt.sca(axs[3])
plt.title('$=$')
plt.axis('off')

plt.sca(axs[4])
ax = sns.heatmap(A@A_inverse,cmap='RdBu_r',vmax = all_max,vmin = all_min,
                 cbar_kws={"orientation": "horizontal"},
                 yticklabels=np.arange(1,n+1), xticklabels=np.arange(1,n+1),
                 annot = True,fmt=".2f")
ax.set_aspect("equal")
plt.title('$I$')
plt.yticks(rotation=0)
```



如果学完这一章，大家对矩阵乘法规则还是一头雾水，我只有一个建议——死记硬背！

先别问为什么。就像背诵九九乘法表一样，把矩阵乘法规则背下来。此外，再次强调矩阵乘法等运算不是“奇技淫巧”。后面，大家会逐步意识到矩阵乘法的洪荒伟力。