

1

All Is Number

万物皆数

数字统治万物



万物皆数。

All is Number.

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



- ◀ % 求余数
- ◀ float() 将输入转化为浮点数
- ◀ input() 函数接受一个标准输入数据, 返回为 string 类型
- ◀ int() 将输入转化为整数
- ◀ is_integer() 判断是否为整数
- ◀ lambda 构造匿名函数; 匿名函数是指一类无需定义函数名的函数或子程序
- ◀ len() 返回序列或者数据帧的数据数量
- ◀ math.e math 库中的欧拉数
- ◀ math.pi math 库中的圆周率
- ◀ math.sqrt(2) math 库计算 2 的平方根
- ◀ mpmath.e mpmath 库中的欧拉数
- ◀ mpmath.pi mpmath 库中的圆周率
- ◀ mpmath.sqrt(2) mpmath 库计算 2 的平方根
- ◀ numpy.add() 向量或矩阵加法
- ◀ numpy.array() 构造数组、向量或矩阵
- ◀ numpy.cumsum() 计算累计求和
- ◀ numpy.linspace() 在指定的间隔内, 返回固定步长数组
- ◀ numpy.matrix() 构造二维矩阵
- ◀ print() 在 console 打印
- ◀ range() 返回的是一个可迭代对象, range(10) 返回 0 ~ 9, 等价于 range(0, 10); range(1, 11) 返回 1 ~ 10; range(0, -10, -1) 返回 0 ~ -9; range(0, 10, 3) 返回 [0, 3, 6, 9], 步长为 3
- ◀ zip(*) 将可迭代的对象作为参数, 让对象中对应的元素打包成一个个元组, 然后返回由这些元组组成的列表。* 代表解包, 返回的每一个都是元组类型, 而并非是原来的数据类型

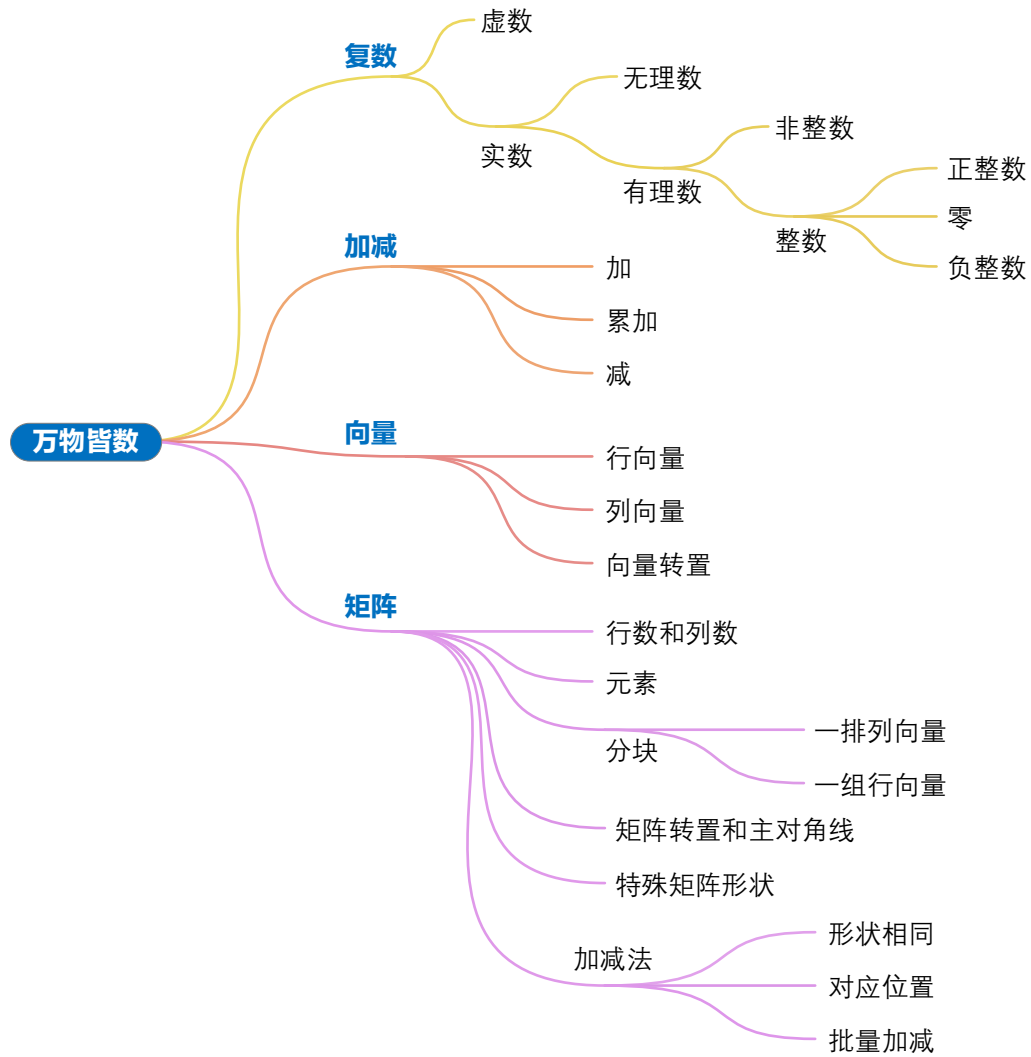
本 PDF 文件为作者草稿, 发布目的为方便读者在移动终端学习, 终稿内容以清华大学出版社纸质出版物为准。

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



1.1 数字和运算：人类思想的伟大飞跃

数字，就是人类思想的空气，无处不在，不可或缺。

大家不妨停止阅读，用一分钟时间，看看自己身边哪里存在数字。

举目四望，你会发现，键盘上有数字，书本印着数字，手机显示数字，交易媒介充满数字，食品有卡路里数值，时钟的数字提醒我们时间，购物扫码本质也是数字。一串串手机号码让人们联通，身份编号是我们的个体的标签 ...

当下，数字已经融合到人类生活的方方面面；多数时候，数字像是空气，我们认为它理所当然，甚至忽略了它的存在。

数字是万物的绝对尺度，数字更是一种高阶的思维方式。远古时期，不同地域、不同族群的人类突然意识到，2 只鸡，2 只兔，2 头猪，有一个共性，那就是——2。

2 和更多更多数字，以及它们之间加、减、乘、除以及更多复杂运算被抽象出来，这是人类思想的一次伟大飞跃。

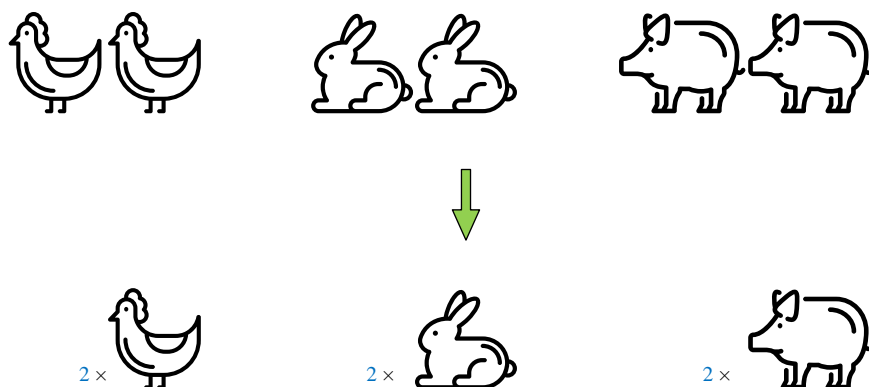


图 1. 数字是人类抽象思维活动的产物

数字这一宝贵的人类遗产，在不同地区、不同种族之间薪火相传。

5000 年前，古巴比伦人将各种数学计算，比如倒数、平方、立方，刻在泥板上。古埃及则是将大量数学知识记录在纸草上。

古巴比伦采用 60 进制；不谋而合，中国自古便发明使用天干地支六十甲子为一个周期来纪年。

今人所说的“阿拉伯数字”实际上是古印度人创造的。古印度发明了**十进制** (decimal system)，而古阿拉伯人将它们发扬光大；中世纪末期，十进制传入欧洲，而日后成为全世界的标准。中国古代也创造了十进制。

有学者认为，人类不约而同地发明并广泛使用十进制，是因为人类有十根手指；数数的时候，自然而然地用手指记录。

虽然十进制大行其道，但是其他进制依然广泛运用；比如，**二进制** (binary system)、**八进制** (octal system) 和 **十六进制** (hexadecimal system) 经常在电子系统使用。

日常生活中，我们不知不觉中也经常使用其他进制。**十二进制** (duodecimal system) 常常出现，比如十二小时制、一年十二个月、**黄道十二宫** (zodiac)、**十二地支** (Earthly Branches)、**十二生肖** (Chinese zodiac)。四进制也不罕见，比如一年四个季度。二十四进制用在一天 24 小时、一年 24 节气。六十进制也很常用，比如一分钟 60 秒、一小时 60 分钟。



图 2. 古巴比伦泥板 (图片来自 Wikipedia)

随着科学技术持续发展，人类的计算也日趋复杂。零、负数、分数、小数、无理数、虚数被发明创造出来。于此同时，人类也在发明改进计算工具，让计算更快、更准。

算盘，作为一种原始的计算工具，可能最先发源于公元前 2400 年的巴比伦。现如今，算盘已经基本绝迹。随着运算量和复杂度不断提高，对运算速度、准确度的需求激增，人类亟需摆脱手工运算，计算器应运而生。

1622 年，英国数学家**威廉·奥特雷德** (William Oughtred) 发明计算尺。早期计算尺主要用来四则运算，而后发展到可以用来求对数和三角函数。直到二十世纪后期被便携式计算器代替之前，计算尺一度是科学和工程重要的计算工具。

1642 年，法国数学家**帕斯卡** (Blaise Pascal) 发明机械计算器，这个机器可以直接进行加减运算。难以想象，帕斯卡设计自己第一台计算器时，未满 19 岁。

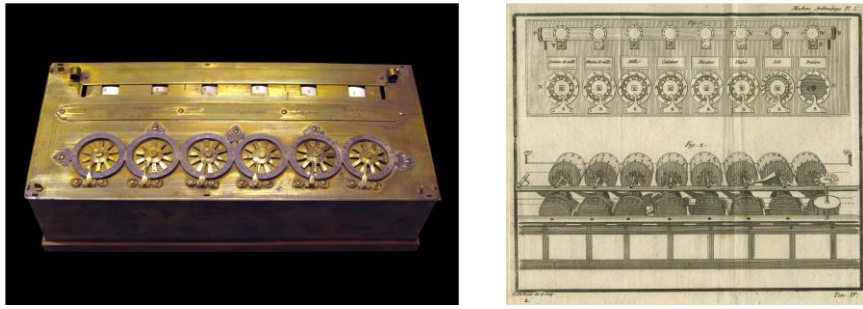


图 3. 帕斯卡机械计算器和原理图纸 (图片来自 Wikipedia)

1822 年前后，英国数学家**查尔斯·巴贝奇** (Charles Babbage)，设计完成**差分机** (difference engine)。第一台差分机重达 4 吨，最高可以存 16 位数。差分机是以蒸汽机为动力的自动机械计算机，它已经很接近第一台计算机；因此，也有很多人将查尔斯·巴贝奇称作“计算机之父”。

1945 年，“伊尼亚克”ENIAC 诞生。ENIAC 的全称为**电子数值积分计算机** (Electronic Numerical Integrator And Computer)。ENIAC 是一台真正意义上的电子计算机。ENIAC 重达 27 吨，占地 167 平方米。

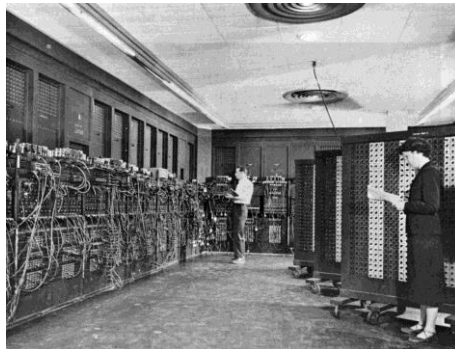


图 4. ENIAC 计算机 (图片来自 Wikipedia)

20 世纪 50 年代电子计算机主要使用真空管，而后开始使用半导体晶体管；半导体使得计算机体积变得更小、成本更低、耗电更少、性能更可靠。进入二十世纪，计算机器的更新迭代，让人目不暇接，甚至让人感觉窒息。

20 世纪 70 年代，集成电路和微处理器先后投入大规模使用，计算机和其他智能设备开始逐渐步入寻常百姓家。

现如今，计算的竞赛愈演愈烈，量子计算机的研究进展如火如荼。

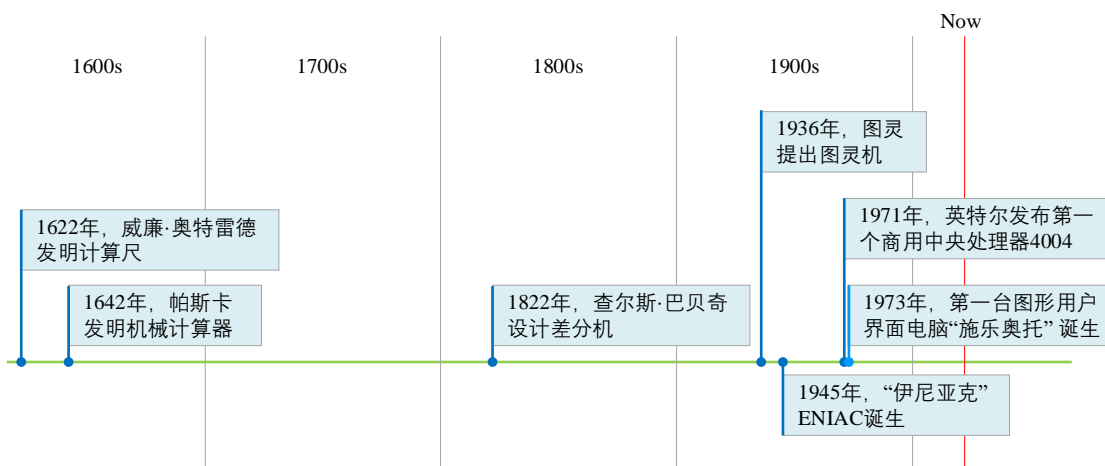


图 5. 计算器发展历史时间轴

到这里，我们不妨停下来，喘口气，回望来时的路；再去看看数字最朴素、最原始、最直观的形态。

1.2 数字分类：从复数到自然数

本节介绍数字分类，介绍的数字类型如图 6 所示。

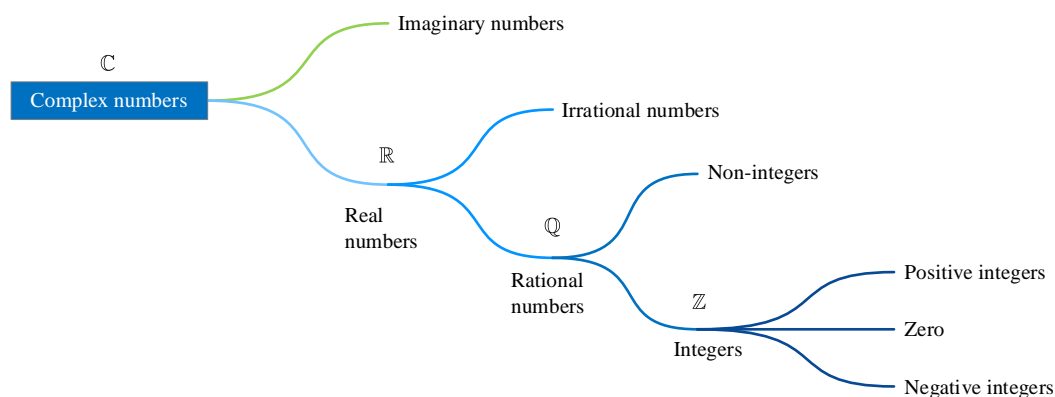


图 6. 数字分类

复数

复数 (complex number) 包括**实数** (real numbers) 和**虚数** (imaginary number)。**复数集** (the set of complex numbers) 的记号为 \mathbb{C} 。

集合 (set)，简称**集**，是指具有某种特定性质**元素** (element) 的总体；白话说，集合就是一堆东西构成的整体。因此，复数集是所有复数构成的总体。

复数的具体形式如下。

$$a + bi \quad (1)$$

其中， a 和 b 是实数。

(1) 中， i 是**虚数单位** (imaginary unit)， i 的平方等于 -1 ，即。

$$i^2 = -1 \quad (2)$$

笛卡尔 (René Descartes) 最先提出虚数这个概念；而后，**欧拉** (Leonhard Euler) 和**高斯** (Carl Friedrich Gauss) 等人对虚数做了深入研究。

有意思的是，不经意间，(1) 中便使用 a 和 b 来代表实数。用抽象字母来代表具体数值是代数的基础。**代数** (algebra) 的研究对象不仅是数字，而是各种抽象化的结构。

实数

实数集 (the set of real numbers) 记号为 \mathbb{R} ；实数包括**有理数** (rational numbers) 和**无理数** (irrational numbers)。

(1) 中， $b = 0$ 时，便得到的是实数。如图 7 所示，实数集可以用**实数轴** (real number line 或 real line) 来展示。

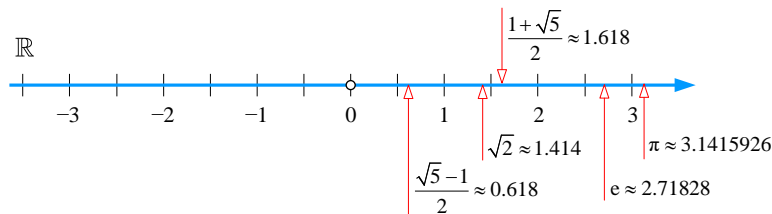


图 7. 实数轴

无意之间，我们又用到了解析几何中重要的工具之一——数轴。图 7 这个看似平凡无奇的一根轴，实际上是人类的一个伟大发明创造。

数轴描述一维空间，两根垂直并相交于原点的数轴张成了二维坐标系，在二维平面上升起一根垂直平面的数轴便张成了三维坐标系。数轴和坐标系的发明让代数和几何前所未有地结合在一起。

目前，数学被分割成一个个板块——算数、代数、几何、解析几何、线性代数、概率统计等等——这种安排虽然有利于学习，但是板块之间的联系被人为割裂；本书的重要任务之一就是强化各个板块之间的联系，让大家看见森林，而不是一棵棵树。

有理数

有理数集合用“ \mathbb{Q} ”表示，有理数可以表达为**两个整数的商** (quotient of two integers)，形如。

$$\frac{a}{b} \quad (3)$$

其中， a 为**分子** (numerator)， b 为**分母** (denominator)。

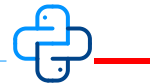
注意 (3) 中**分母不为零** (the denominator is not equal to zero)。有理数可以表达成**有限小数** (finite decimal 或 terminating decimal)，或者**无限循环小数** (repeating decimal 或 recurring decimal)；小数中的圆点叫做**小数点** (decimal separator)。

无理数

图 7 所示实数轴上除有理数以外，都是无理数。无理数不能用一个整数或两个整数的商来表示；无理数也叫**无限不循环小数** (non-repeating decimal)。

很多重要的数值都是无理数，比如图 7 所示数轴上的圆周率 π (pi)、 $\sqrt{2}$ (the square root of two)、**自然常数** e (exponential constant) 和**黄金分割比** (golden ratio)。自然常数 e 也叫**欧拉数** (Euler's number)。

执行如下代码，可以打印出 π 、 e 和 $\sqrt{2}$ 的精确值。代码使用了 `Math` 库中函数，`Math` 库是 Python 提供的内置数学函数库。



```
# Bk Ch1 01
# Import math library
import math

# Print the value of pi
print ('pi = ', math.pi)

# Print the value of e
print ('e = ', math.e)

# Print the value of square root of 2
print ('sqrt(2) = ', math.sqrt(2))
```

打印结果如下。

```
pi = 3.141592653589793
e = 2.718281828459045
sqrt(2) = 1.4142135623730951
```


下面，我们做一个有趣的实验——打印圆周率和自然常数 e 小数点后 1,000 位数字。图 8 所示为圆周率小数点后 1,000 位，图 9 采用热图的形式展示圆周率小数点后 1,024 位。图 10 自然常数 e 小数点后 1,000 位。

中国古代南北朝时期数学家祖冲之 (429 ~ 500) 曾刷新圆周率估算记录；他估算圆周率在 3.1415926 到 3.1415927 之间，这一记录在之后的约 1000 年内无人撼动。圆周率的估算是本书的一条重要线索，我们将追随前人足迹，用不同的数学工具估算圆周率。

3.1415926535	8979323846	2643383279	5028841971	6939937510	
5820974944	5923078164	0628620899	8628034825	3421170679	100 digits
8214808651	3282306647	0938446095	5058223172	5359408128	
4811174502	8410270193	8521105559	6446229489	5493038196	200 digits
4428810975	6659334461	2847564823	3786783165	2712019091	
4564856692	3460348610	4543266482	1339360726	0249141273	300 digits
7245870066	0631558817	4881520920	9628292540	9171536436	
7892590360	0113305305	4882046652	1384146951	9415116094	400 digits
3305727036	5759591953	0921861173	8193261179	3105118548	
0744623799	6274956735	1885752724	8912279381	8301194912	500 digits
9833673362	4406566430	8602139494	6395224737	1907021798	
6094370277	0539217176	2931767523	8467481846	7669405132	600 digits
0005681271	4526356082	7785771342	7577896091	7363717872	
1468440901	2249534301	4654958537	1050792279	6892589235	700 digits
4201995611	2129021960	8640344181	5981362977	4771309960	
5187072113	4999999837	2978049951	0597317328	1609631859	800 digits
5024459455	3469083026	4252230825	3344685035	2619311881	
7101000313	7838752886	5875332083	8142061717	7669147303	900 digits
5982534904	2875546873	1159562863	8823537875	9375195778	
1857780532	1712268066	1300192787	6611195909	2164201989	1000 digits

图 8. 圆周率小数点后 1000 位

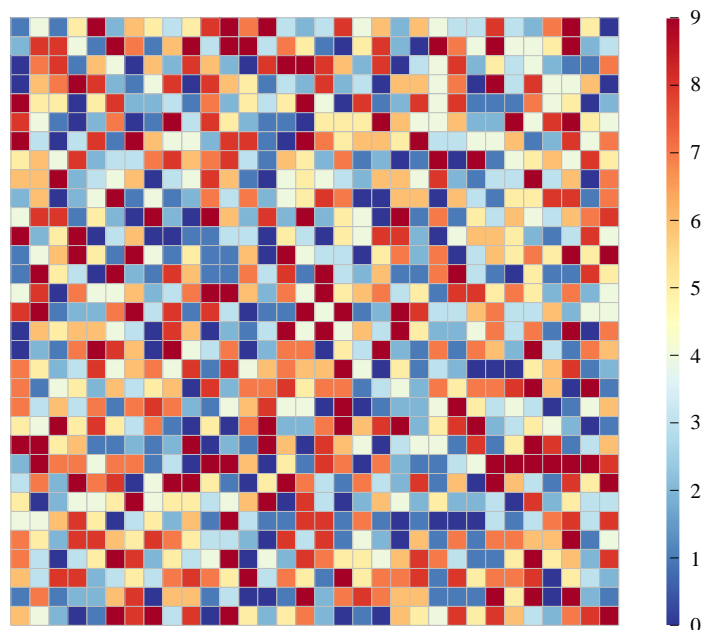


图 9. 圆周率小数点后 1024 位热图

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

2.7182818284	5904523536	0287471352	6624977572	4709369995	
9574966967	6277240766	3035354759	4571382178	5251664274	100 digits
2746639193	2003059921	8174135966	2904357290	0334295260	
5956307381	3232862794	3490763233	8298807531	9525101901	200 digits
1573834187	9307021540	8914993488	4167509244	7614606680	
8226480016	8477411853	7423454424	3710753907	7744992069	300 digits
5517027618	3860626133	1384583000	7520449338	2656029760	
6737113200	7093287091	2744374704	7230696977	2093101416	400 digits
9283681902	5515108657	4637721112	5238978442	5056953696	
7707854499	6996794686	4454905987	9316368892	3009879312	500 digits
7736178215	4249992295	7635148220	8269895193	6680331825	
2886939849	6465105820	9392398294	8879332036	2509443117	600 digits
3012381970	6841614039	7019837679	3206832823	7646480429	
5311802328	7825098194	5581530175	6717361332	0698112509	700 digits
9618188159	3041690351	5988885193	4580727386	6738589422	
8792284998	9208680582	5749279610	4841984443	6346324496	800 digits
8487560233	6248270419	7862320900	2160990235	3043699418	
4914631409	3431738143	6405462531	5209618369	0888707016	900 digits
7683964243	7814059271	4563549061	3031072085	1038375051	
0115747704	1718986106	8739696552	1267154688	9570350354	1000 digits

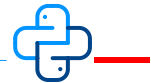
图 10. 自然常数 e 小数点后 1000 位

观察图 8 所示的圆周率小数点后 1,000 位数字，可以发现 0~9 这 10 个数字反复随机出现；这里，“随机”是指偶然、随意、无法预测，概率统计中将会大量使用“随机”这个概念。

给大家留一个问题，能否凭直觉猜一下哪个数字出现的次数最多？圆周率小数点后 10,000 位、100,000 位，乃至 1,000,000 位，0~9 这 10 个数字出现的次数又会怎样？

答案会在本系列丛书《概率统计》一册揭晓。

以下代码打印圆周率、 $\sqrt{2}$ 和自然常数 e，小数点后 1,000 位数字。Mpmath 是一个任意精度浮点运算库。



```
# Bk Ch1 02

from mpmath import mp

mp.dps = 1000 + 1

print('print 1000 digits of pi behind decimal point')
print(mp.pi)

print('print 1000 digits of e behind decimal point')
print(mp.e)

print('print 1000 digits of e behind decimal point')
print(mp.sqrt(2))
```

目前，背诵 pi 的小数点后最多位数的世界吉尼斯纪录是 70,000 位。该纪录由印度人在 2015 年创造，用时近 10 小时。这一纪录的需要背诵的数字量是图 8 的 70 倍，感兴趣的读者可以修改代码获取，并打印保存这些数字。注意，取决于个人电脑的算力，这一过程可能要用时很久。

整数

整数 (integers) 包括**正整数** (positive integers)、**负整数** (negative integers) 和零。正整数**大于零** (greater than zero); 负整数**小于零** (less than zero)。整数集用“ \mathbb{Z} ”表达。

整数重要性质之一是，整数相加、相减或相乘结果还是整数。

奇偶性 (parity) 是整数另外一个重要性质。**能被 2 整除的整数被称作偶数** (an integer is called an even integer if it is divisible by two); 否则，**该整数为奇数** (the integer is odd)。

利用以上原理，我们可以写一段 Python 代码，判断数字奇偶性。代码中，`%` 用来求余数。



```
# Bk Ch1_03

# A number is even if division by 2 gives a remainder of 0.
# If the remainder is 1, it is an odd number.

num = float(input("Enter a number: "))

if num.is_integer():

    if (num % 2) == 0:
        print("{0} is even ".format(int(num)))
    else:
        print("{0} is odd ".format(int(num)))

else:
    print("{0} is not an integer ".format(num))
```

自然数

自然数 (natural number 或 counting numbers) 有些时候指的是正整数，有些时候指的是**非负整数** (nonnegative integer)，这时自然数集合包括“0”。零，是否属于自然数尚未达成一致意见。

至此，我们回顾了数字类型；表 1 总结数字类型，并给出例子。

表 1. 不同种类数字及举例

英文表达	汉语表达	举例
Complex number	复数	$7 + 2i$
Imaginary number	虚数	$2i$
Real number	实数	7
Irrational number	无理数	π, e
Rational number	有理数	1.5
Integer	整数	-1, 0, 1
Natural number	自然数	9, 18

1.3 加减：最基本的数学运算

本节介绍加、减这两个最基本算数运算。

加法

加法 (addition) 的运算符为**加号** (plus sign, plus symbol); 加法运算式中，**等式** (equation) 的左边为**加数** (addend) 和**被加数** (augend, summand)，等式的右边是**和** (sum)。

加法的表达方式多种多样，比如说“**和** (summation)”、“**加** (plus)”、“**增长** (increase)”、“**小计** (subtotal)”和“**总数** (total)”等等。

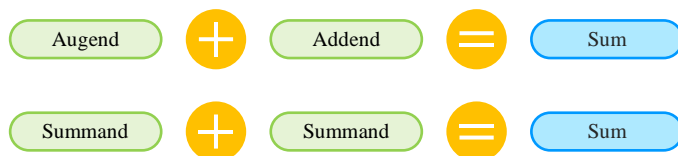


图 11. 加法运算

图 12 是在数轴上可视化 $2 + 3 = 5$ 这一加法运算。

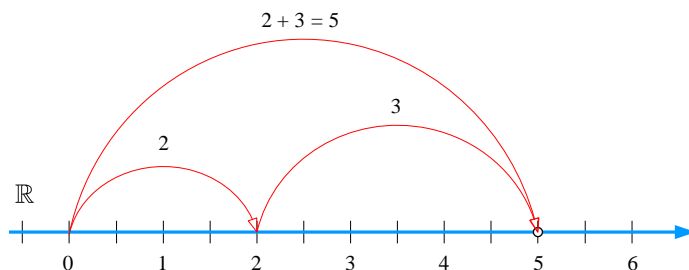


图 12. $2 + 3 = 5$ 在数轴上的可视化

如下代码完成图 12 所示加法运算。

```
# Bk Ch1 04
```

```
num1 = 2
num2 = 3
```

```
# add two numbers
```

```
sum = num1 + num2
```



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。

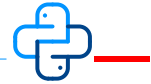
代码及 PDF 文件下载：<https://github.com/Visualize-ML>

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

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

对以上代码稍作调整，利用 `input()` 函数，让用户通过键盘输入数值。



```
# Bk Ch1 05

# user input numbers
num1 = input('Enter first number: ')
num2 = input('Enter second number: ')

# add two numbers
sum = float(num1) + float(num2)

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

结果打印如下。

```
Enter first number: 2
Enter second number: 3
The sum of 2 and 3 is 5.0
```

表 2 总结加法的英文表达。

表 2. 加法的英文表达

数学表达	英文表达
$1+1=2$	One plus one equals two. The sum of one and one is two. If you add one to one, you get two.
$2+3=5$	Two plus three equals five. Two plus three is equal to five. Three added to two makes five. If you add two to three, you get five.

累计求和

对于一行数字，**累计求和** (cumulative sum 或 cumulative total) 得到的结果不是一个总和，而是从左向右每加一个数值，得到的分步结果。比如，自然数 1 到 10 累计求和结果为。

1, 3, 6, 10, 15, 21, 28, 36, 45, 55 (4)

(4) 累计求和计算过程如下所示。

```

      1+2+3+4+5+6=21
      1+2+3+4+5=15
      1+2+3+4=10
      1+2+3=6
      1+2=3
      1
1, 2, 3, 4, 5, 6, 7, 8, ... (5)

```

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

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.cumsum()` 函数进行累计求和。NumPy 是一个开源的 Python 库，丛书的大量线性代数运算都离不开它。



```
# Bk Ch1 06

import numpy as np

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

a_i_cumsum = np.cumsum(a_i)
print(a_i_cumsum)
```

减法

减法 (subtraction) 是**加法的逆运算** (inverse operation of addition)，运算符为**减号** (minus sign)。如图 13 所示，减法运算过程是，**被减数** (minuend) 减去**减数** (subtrahend) 得到**差** (difference)。

减法其他名字包括“**减** (minus)”、“**少** (less)”、“**差** (difference)”、“**减少** (decrease)”、“**拿走** (take away)”和“**扣除** (deduct)”等等。

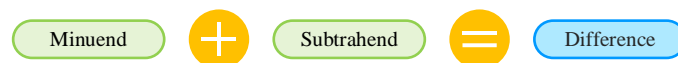


图 13. 减法运算

图 14 所示为在数轴上展示 $5 - 3 = 2$ 的减法运算。

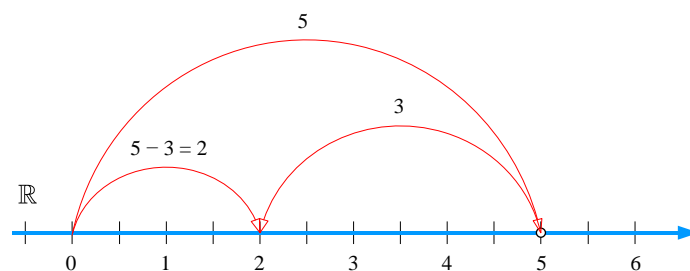


图 14. $5 - 3 = 2$ 在数轴上的可视化

以下代码完成图 14 给出的减法运算。



```
# Bk Ch1 07
num1 = 5
num2 = 3

# add two numbers
diff = num1 - num2

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

相反数

求**相反数** (inverse number 或 additive inverse number) 的过程是**改变符号** (reverses its sign), 这个操作常称作**变号** (sign change)。比如, 5 的相反数为-5 (negative five)。表 3 给出常用的减法英文表达。

表 3. 减法常见英文表达

数学表达	英文表达
5-3=2	Five minus three equals two.
	Five minus three is equal to two.
	Three subtracted from five equals two.
	If you subtract three from five, you get two.
4-6=-2	If you take three from five, you get two.
	Four minus six equals negative two.
	Four minus six is equal to negative two.

1.4 向量：数字排成行、列

本书的读者会问, 明明第一章讲的是算数, 怎么一下扯到向量, 这个线性代数的概念。

向量、矩阵等线性代数概念对于数据科学和机器学习至关重要。在机器学习中, 数据几乎都是以矩阵形式存储、运算。毫不夸张地说, 没有线性代数, 就没有现代计算机运算。

然而, 初学者对向量、矩阵等概念却展现出特别的抗拒, 甚至恐惧。

逐渐地, 大家会发现算数、代数、解析几何、微积分、概率统计、优化方法并不是一个个孤岛, 线性代数正是连接它们的重要桥梁之一。

基于以上考虑, 本系列丛书把线性代数知识穿插到数学各个板块, 加强读者对线性代数知识的理解。

书归正传。若干数字排成一行或一列, 并且用中括号括起来, 得到的数组叫做**向量** (vector)。

排成一行的叫做**行向量** (row vector), 排成一列的叫做**列向量** (column vector); 白话说, 行向量就是表格的一行数字, 列向量就是表格的一列数字。如下两例分别展示行向量和列向量。

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

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_{1 \times 3}, \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}_{3 \times 1} \quad (6)$$

注意 (6) 中下角标, 1×3 代表“1 行、3 列”, 3×1 代表“3 行、1 列”。本书在给出向量和矩阵时, 偶尔会以下角标形式展示其形状。

利用 NumPy 库, 可以用 `numpy.array([1, 2, 3])` 定义 (6) 中行向量。用 `numpy.array([[1], [2], [3]])` 定义 (6) 中列向量。

注意, 定义向量时如果只用一层中括号 [], 比如 `numpy.array([1, 2, 3])`, 得到的结果只有一个维度; 有两层中括号 [], `numpy.array([[1], [2], [3]])` 得到的结果有两个维度。这一点在矩阵运算中非常重要。

转置

行向量**转置**(transpose) 得到列向量; 同理, 列向量转置得到行向量。

如图 15 所示, 转置相当于镜像; 图 15 中红线就是镜像轴, 红线从第 1 行、第 1 列元素出发, 朝向右下方 45° 。

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^T = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}^T = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad (7)$$

本系列丛书采用的转置符号为上标 T 。

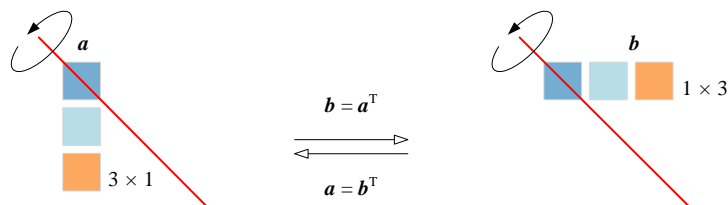


图 15. 向量转置

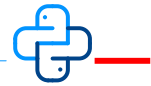
本系列丛书用加粗、斜体小写字母来代表向量, 比如图 15 中向量 \mathbf{a} 和向量 \mathbf{b} 。

给定如下行向量 \mathbf{a} , \mathbf{a} 有 n 个元素, 元素本身用小写字母表示。

$$\mathbf{a} = [a_1 \quad a_2 \quad \cdots \quad a_n] \quad (8)$$

其中, 下角标代表向量元素的序号; $[a_1, a_2, \dots, a_n]$ 读作“ n row vector, a sub one, a sub two, dot dot dot, a sub n ”。

以下代码定义行向量和列向量, 并展示如何通过转置将行向量和列向量相互转换。



```
# Bk_Ch1_08

import numpy as np

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

b = a_row.T

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

a = b_col.T
```

本书在介绍线性代数相关知识时，会尽量使用具体数字，而不是变量符号；这样做的考虑是，让读者构建向量和矩阵运算最直观的体验。

本系列丛书中《矩阵力量》一册则系统讲解线性代数知识，以及线性代数和代数、解析几何、微积分、概率统计、优化方法、数据科学等板块的联系。

1.5 矩阵：数字排列成长方形

矩阵 (matrix) 将一系列数字以长方形方式排列，比如。

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2 \times 3}, \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3 \times 2}, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{2 \times 2} \quad (9)$$

白话说，矩阵将数字排列成表格，有行、有列。(9) 给出三个矩阵，分别是 2 行 3 列 (记做 2×3)、2 行 3 列 (记做 2×3) 和 2 行 2 列 (记做 2×2)。丛书用大写、斜体字母代表矩阵，比如矩阵 **A** 和矩阵 **B**。

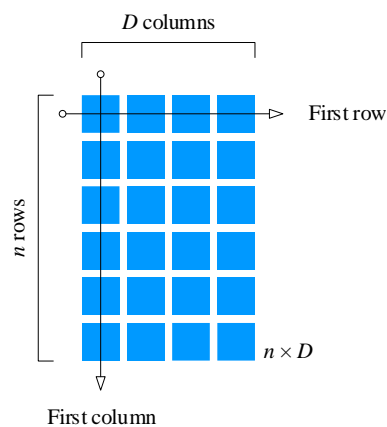


图 16. $n \times D$ 矩阵 **X**

图 16 所示为一个 $n \times D$ (n by capital D) 矩阵 X , n 是**矩阵的行数** (number of rows in the matrix), D 是**矩阵的列数** (number of columns in the matrix)。 X 可以展开写成如下表格形式。

$$X_{n \times D} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{bmatrix} \quad (10)$$

矩阵 X 中, **元素** (element) $x_{i,j}$ 被称作 ij 元素 (ij entry 或 ij element), 也可以说 $x_{i,j}$ 出现在 i 行 j 列 (appears in row i and column j)。比如, $x_{n,1}$ 是矩阵 X 的第 n 行、第 1 列元素。本系列丛中, 数据矩阵一般采用大写、粗体、斜体 X 表达。

表 4 总结如何用英文读矩阵和矩阵元素。

表 4. 矩阵有关英文表达

数学表达	英文表达
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	Two by two matrix, first row one two, second row three four
$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$	m by n matrix, first row a sub one one, a sub one two, dot dot dot, a sub one n second row a sub two one, a sub two two, dot dot dot, a sub one n dot dot dot last row a sub m one, a sub m two, dot dot dot a sub m n
$a_{i,j}$	Lowercase (small) a sub i comma j
$a_{i,j+1}$	Lowercase a double subscript i comma j plus one
$a_{i,j-1}$	Lowercase a double subscript i comma j minus one

如下代码利用 `numpy.array()` 定义矩阵, 并提取矩阵的某一列、某两列、某一行、某一个位置的具体值。



```
# Bk_Ch1_09

import numpy as np

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

A_first_col = A[:,0] # saved as one dimension row
A_first_col_V2 = A[:,[0]] # saved as a column
A_first_second_col_V2 = A[:,[0,1]] # extract first and second columns
A_first_third_col_V2 = A[:,[0,2]] # extract first and third columns
A_first_row = A[[0],:] # extract first row
A_second_row = A[[1],:] # extract second row
A_second_row_first_col = A[[1],[0]] # i = 2, j = 1
```

鸢尾花数据集

绝大多数情况，数据以矩阵形式存储、运算。

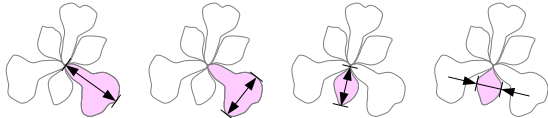
图 17 所示鸢尾花卉数据集，全称为安德森鸢尾花卉数据集 (Anderson's Iris data set)，是植物学家埃德加·安德森 (Edgar Anderson) 在加拿大魁北克加斯帕半岛上的采集的 150 个鸢尾花样本数据。这些数据都属于鸢尾属下的三个亚属。每一类鸢尾花收集了 50 条样本记录，共计 150 条。

图 17 中数据第一列是序号，不算做矩阵元素；但是它告诉我们，鸢尾花数据集有 150 个样本数据，即 $n = 150$ 。紧随其后的是被用作样本定量分析的四个特征——花萼长度 (sepal length)、花萼宽度 (sepal width)、花瓣长度 (petal length) 和花瓣宽度 (petal width)。

最后一列为分类，即标签 (label)。三个标签分别为——山鸢尾 (setosa)、变色鸢尾 (versicolor) 和维吉尼亚鸢尾 (virginica)。最后一列标签算在内，矩阵有 5 列，即 $D = 5$ 。

这个 150×5 的矩阵的每一列，即列向量，为鸢尾花一个特征的样本数据。矩阵的每一行，即行向量，代表某一个特定的鸢尾花样本。

鸢尾花数据集可以说是本系列丛书最重要的数据集，没有之一。我们将用各种数学工具从各种视角分析鸢尾花数据。



Index	Sepal length X_1	Sepal width X_2	Petal length X_3	Petal width X_4	Species C
1	5.1	3.5	1.4	0.2	Setosa C_1
2	4.9	3	1.4	0.2	
3	4.7	3.2	1.3	0.2	
...	
49	5.3	3.7	1.5	0.2	
50	5	3.3	1.4	0.2	Versicolor C_2
51	7	3.2	4.7	1.4	
52	6.4	3.2	4.5	1.5	
53	6.9	3.1	4.9	1.5	
...	
99	5.1	2.5	3	1.1	Virginica C_3
100	5.7	2.8	4.1	1.3	
101	6.3	3.3	6	2.5	
102	5.8	2.7	5.1	1.9	
103	7.1	3	5.9	2.1	
...	
149	6.2	3.4	5.4	2.3	
150	5.9	3	5.1	1.8	




图 17. 鸢尾花数据表格，单位为厘米 (cm)

矩阵形状记号

大部分数学教科书表达矩阵形状时采用 $m \times n$ ；本系列丛书表达矩阵形状时，一般用 $n \times D$ ， n 表达行数， D 表达列数。

采用 $n \times D$ 这种记号有几方面的考虑。

首先 m 和 n 这两个字母区分度不高；两者长相类似，而且发音相近，这让初学者辨别行、列时很大惑；而 n 和 D ，一个小写字母，一个大写字母，且发音有显著区别，很容易辨识。

此外，处理数据时大家会发现，`pandas.DataFrame` 定义的数据帧中，列代表特征，比如性别、身高、体重、年龄等等；行一般代表样本，比如小张、小王、小姜等等。而统计中，一般用 n 代表样本数，因此决定用 n 来代表矩阵的行数；字母 D 取自 dimension (维度) 的首字母，方便记忆。本系列丛书横跨代数、线性代数、概率统计几个板块， $n \times D$ 这种记法方便大家把矩阵运算和统计知识联系起来。

本书编写之初，也有考虑用 feature (特征) 的首字母 F 来表达矩阵的列数，但最终放弃。一方面，是因为丛书后续会用 F 代表一些特定函数；另一方面， n 和 F 的发音区分度不如 n 和 D 那么高。

基于以上考虑，本系列丛书后续在表达样本数据矩阵形状时都会默认采用 $n \times D$ 这一记法，除非特别说明。

1.6 矩阵：一排列向量，或一组行向量

矩阵可以看做是，若干列向量左右排列，或者若干行向量上下叠放。比如，形状为 2×3 的矩阵可以看成是 3 个列向量左右排列，也可以看成是上下叠放的 2 个行向量。

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2 \times 3} = \left[\begin{bmatrix} 1 \\ 4 \end{bmatrix} \begin{bmatrix} 2 \\ 5 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \end{bmatrix} \right] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad (11)$$

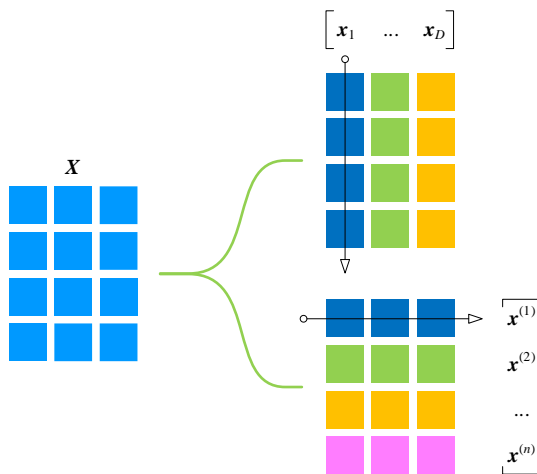


图 18. 矩阵可以分解成一系列行向量或列向量

一般情况，如图 18 所示，形状为 $n \times D$ 的矩阵 \mathbf{X} ，可以写成 D 个左右排列的列向量。

$$\mathbf{X}_{n \times D} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_D] \quad (12)$$

\mathbf{X} 也可以写成上下叠放的 n 个行向量。

$$\mathbf{X}_{n \times D} = \begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \vdots \\ \mathbf{x}^{(n)} \end{bmatrix} \quad (13)$$

实际上，(12) 和 (13) 蕴含着一种重要的思想——矩阵分块 (block matrix 或 partitioned matrix)。本系列丛书《矩阵力量》一册会详细介绍矩阵分块及相关运算规则。

注意，为了区分含序号的列向量和行向量，本系列丛书将列向量的序号写成下角标，比如 \mathbf{x}_1 、 \mathbf{x}_2 、 \mathbf{x}_D 等；将行向量的序号写成上角标加圆括号，比如 $\mathbf{x}^{(1)}$ 、 $\mathbf{x}^{(2)}$ 、 $\mathbf{x}^{(n)}$ 等。

矩阵转置

矩阵转置 (matrix transpose) 指的是将矩阵的行列互换得到的新矩阵，比如下例。

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}_{3 \times 2}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}_{2 \times 3} \quad (14)$$

(14) 中， 3×2 矩阵转置得到矩阵的形状为 2×3 。

图 19 为矩阵转置示意图，其中红色线为**主对角线** (main diagonal)；主对角线是从矩阵第 1 行、第 1 列元素出发，一条向右下方倾斜 45° 斜线。转置前后，矩阵主对角线元素位置不变，比如 (14) 的 1、4 两个元素。向量转置是矩阵转置的特殊形式。

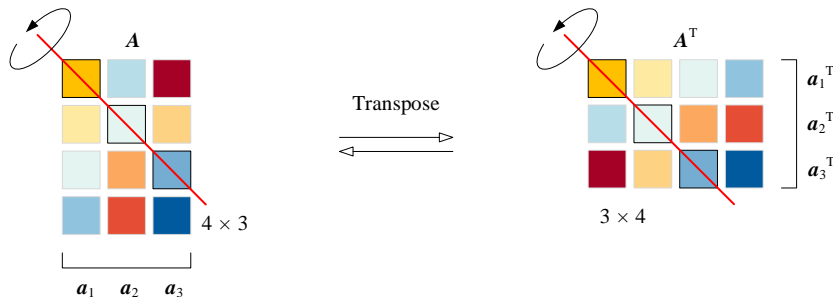


图 19. 矩阵转置

如图 19 所示，矩阵 A 可以写成三个列向量左右排列 $[a_1, a_2, a_3]$ ；对 A 转置得到的结果为。

$$A^T = [a_1 \quad a_2 \quad a_3]^T = \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \end{bmatrix} \quad (15)$$

这一点对于转置运算非常重要，再举个具体例子。给定如下矩阵，并将其写成左右排列的列向量。

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} & \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} & \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} \end{bmatrix} \quad (16)$$

(16) 矩阵转置结果为。

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}^T = \begin{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} & \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} & \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} \end{bmatrix}^T = \begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \\ \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \\ \begin{bmatrix} 7 & 8 & 9 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad (17)$$

1.7 矩阵形状：每种形状都有特殊性质和用途

矩阵的一般形状为长方形，但是矩阵还有很多特殊形状。图 20 所示为常见特殊形态矩阵。

很显然，列向量、行向量都是特殊的矩阵。

如果列向量的元素都为 1，一般记做 $\mathbf{1}$ ，它被称作全 1 列向量，简称**全 1 向量** (all-ones vector)。

如果列向量的元素都是 0，这种列向量叫做**零向量** (zero vector)，记做 $\mathbf{0}$ 。

行数和列数相同的矩阵叫**方阵** (square matrix)，比如 2×2 矩阵。

对角矩阵 (diagonal matrix) 一般是一个主对角线之外的元素皆为 0 的方阵。需要注意的是，读者会在本系列丛书《矩阵力量》一本中发现，对角矩阵也可以不是方阵。

单位矩阵 (identity matrix) 是主对角线元素为 1 其余元素均为 0 的方阵。

对称矩阵 (symmetric matrix) 是指以矩阵元素以主对角线为轴对称的方阵。

零矩阵 (null matrix) 是指所有元素皆为 0 的矩阵，记做 \mathbf{O} 。注意，零矩阵也未必都是方阵。

每一种特殊形状矩阵在线性代数舞台上都扮演特殊的角色，本系列丛书会慢慢讲给大家。

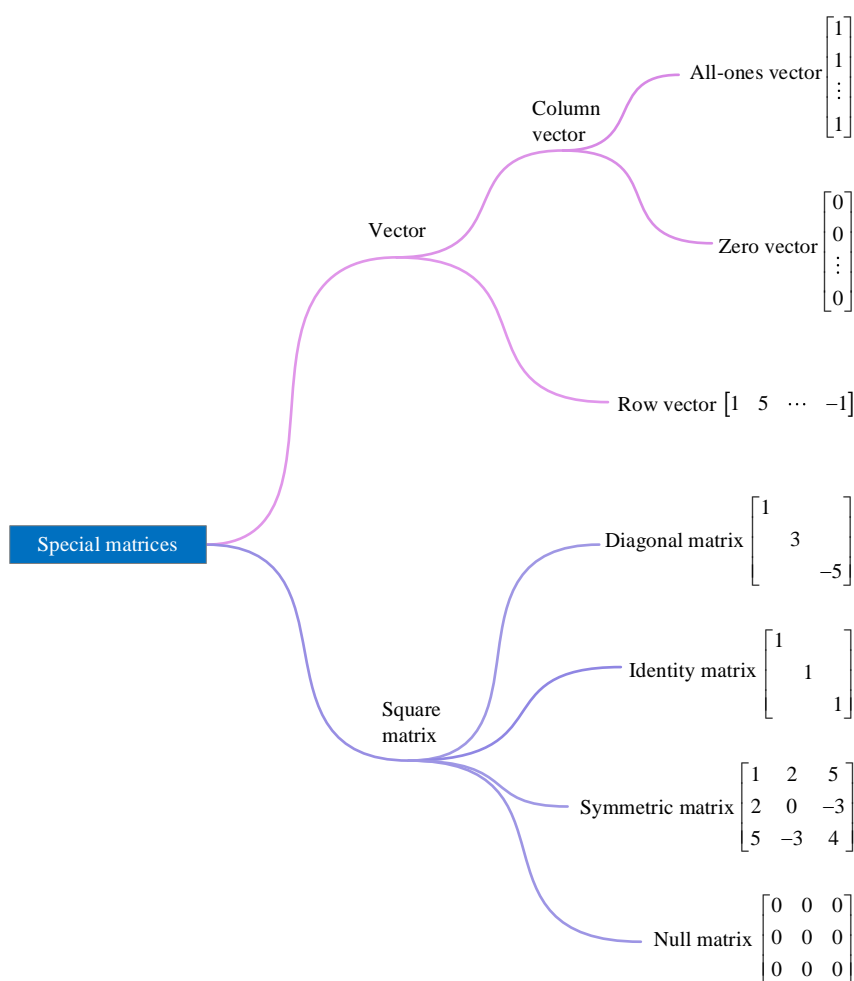


图 20. 常见特殊形态矩阵

1.8 矩阵加减：形状相同，对应位置，批量加减

本节介绍矩阵加减法；矩阵相加减就是批量化完成若干加减运算。矩阵加减可以视作四则运算中加减的高阶版本。

注意，两个矩阵能够完成加减运算的前提——形状相同。

上一节说过，行向量和列向量是特殊的矩阵。两个等长的行向量相加，为对应元素相加，得到还是一个行向量，比如下例。

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} + \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1+4 & 2+5 & 3+6 \end{bmatrix} = \begin{bmatrix} 5 & 7 & 9 \end{bmatrix} \quad (18)$$

同理，两个等长行向量相减，就是对应元素相减，得到的也是相同长度行向量。

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} - \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1-4 & 2-5 & 3-6 \end{bmatrix} = \begin{bmatrix} -3 & -3 & -3 \end{bmatrix} \quad (19)$$

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

(18) 和 (19) 相当于一次性，批量完成了三个加减法运算。

同理，两个等长的列向量相加，得到仍然是一个列向量。

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} \quad (20)$$

图 21 所示为两个数字相加的示意图；而图 22 所示为向量求和。



图 21. 数字求和

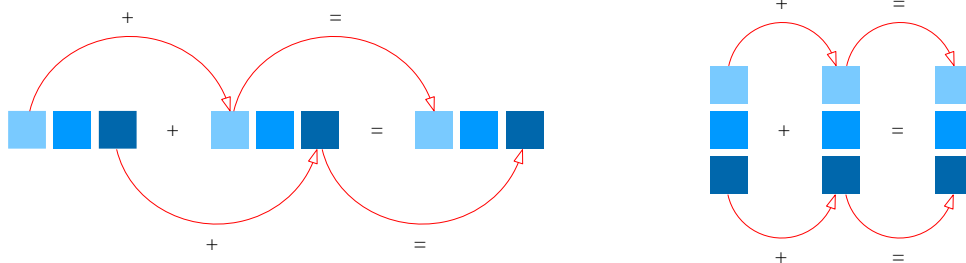


图 22. 向量求和

如下代码展示了四种计算行向量相加的方式。这四种方法中，当然首推用 NumPy。



```
# Bk Ch1 10

list1=[1, 2, 3]
list2=[4, 5, 6]
print([x + y for x, y in zip(list1, list2)])

print(list(map(lambda x,y:x+y, list1, list2)))

import numpy as np

x = np.array(list1)
y = np.array(list2)
print(x+y)

print(np.add(list1,list2))
```

形状相同的两个矩阵相加的结果还是矩阵；运算规则为，对应位置元素相加，形状不变。

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2 \times 3} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}_{2 \times 3} = \begin{bmatrix} 1+1 & 2 & 3 \\ 4 & 5+1 & 6 \end{bmatrix}_{2 \times 3} = \begin{bmatrix} 2 & 2 & 3 \\ 4 & 6 & 6 \end{bmatrix}_{2 \times 3} \quad (21)$$

两个矩阵相减的运算原理完全相同。

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2 \times 3} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}_{2 \times 3} = \begin{bmatrix} 1-1 & 2 & 3 \\ 4 & 5-1 & 6 \end{bmatrix}_{2 \times 3} = \begin{bmatrix} 0 & 2 & 3 \\ 4 & 4 & 6 \end{bmatrix}_{2 \times 3} \quad (22)$$

注意，用 for 循环来解决矩阵相加是最费力的办法，比如下例代码。为了让代码运算效率提高，常用的方法之一就是——向量化 (vectorize)。也就是说，尽量采用向量运算，避免循环。



```
# Bk Ch1 11
A = [[1, 2, 3],
      [4, 5, 6]]
B = [[1, 0, 0],
      [0, 1, 0]]
A_plus_B = [[A[i][j] + B[i][j]
              for j in range(len(A[0]))]
             for i in range(len(A))]
print(A_plus_B)
```

下例所示为利用 NumPy 完成矩阵加法。



```
# Bk Ch1 12
import numpy as np
A = [[1, 2, 3],
      [4, 5, 6]]
B = [[1, 0, 0],
      [0, 1, 0]]
print(np.array(A) + np.array(B))
print(np.add(A, B))
```



数字和数学是抽象的，它们是人类总结的规律，是人类思想的产物。

“双兔傍地走”中的“双”就是 2；2 这个数字对人类有意义，对兔子自身没有意义；两只兔子只顾自的玩耍，一旁暗中观察的某个人在大脑中思维活动抽象产生了“双”这个数字概念，而且要进一步“辨雄雌”。

试想一个没人类的自然界。那里，天地始交，万物并秀，山川巍峨，江河奔涌，雨润如酥，暗香浮动，芳草萋萋，鹿鸣呦呦，鹰击长空，鱼翔浅底。

试问，这般香格里拉的梦幻世界和数字有什么关系？

然而，本书的读者很快就知道，微观世界中，自然界中，天体运行中，人类通过几千年的观察研究发现，数字、数学规律无处不在；可惜天意从来高难问，大部分规律不为人所知罢了。

这让我们不禁追问，可感知世界万物是否仅仅是表象？世界万物创造动力和支配能量，是否就是数字和数学？我们听到的、看到的、触摸到的，是否都是数字化的，虚拟化的？整个物质世界仅仅是某个巨型计算机模拟的产物？这些问题让我不寒而栗。

老子说，“大道无形，生育天地；大道无情，运行日月。”老子是否真的参透了世间万物，它口中的“大道”是否就是数字、数学规律？