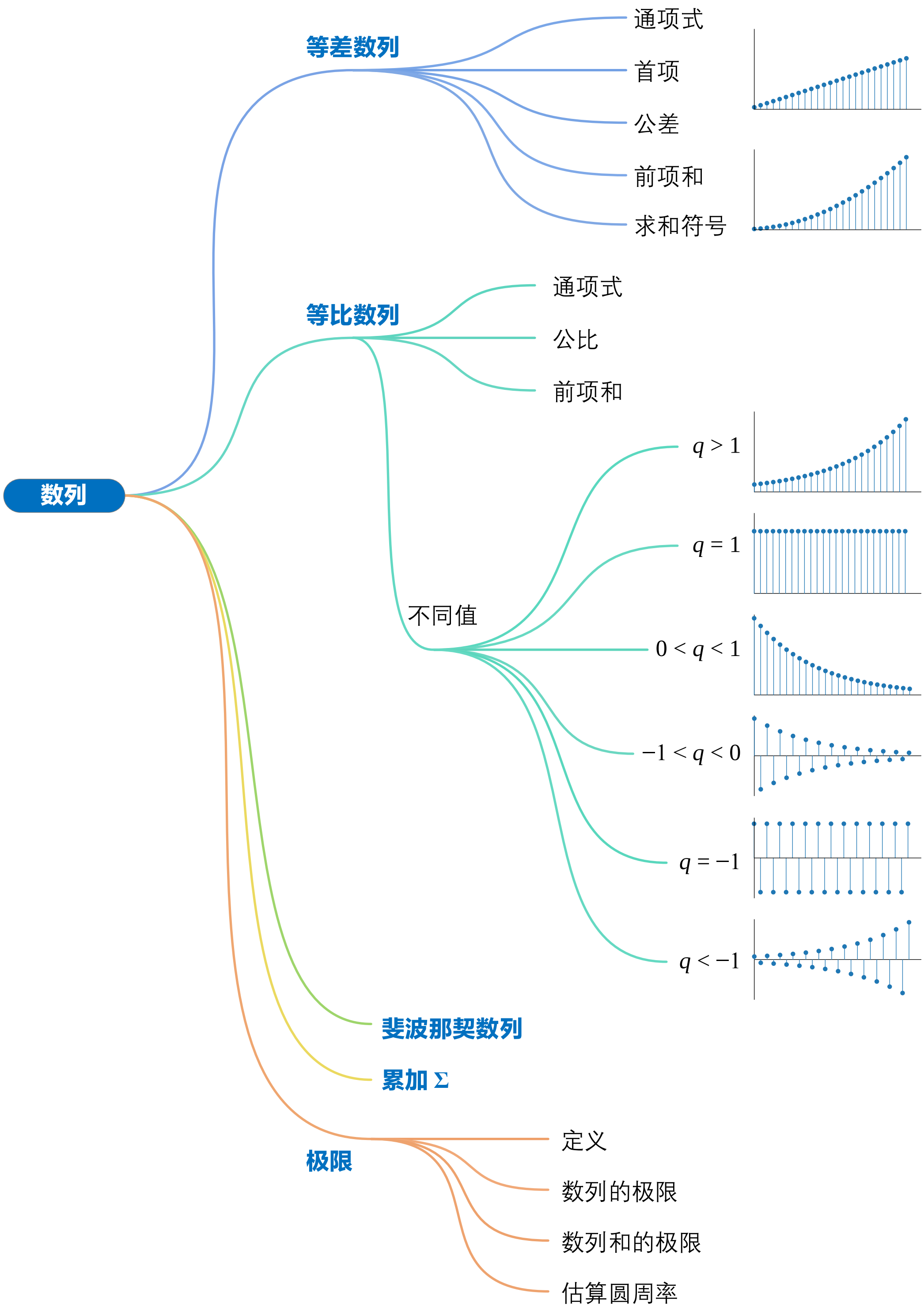
14 Sequences 数列

也是一种特殊函数

> 有数字的地方，就存在美。
>
> ***Wherever there is number, there is beauty.***
>
> —— 普罗克洛 (Proclus) | 古希腊哲学家 | 412 ~ 485

- ◀ `numpy.sum()` 计算数列和
- ◀ `numpy.cumsum()` 计算累积和
- ◀ `numpy.cumprod()` 计算累积乘积
- ◀ `numpy.arange()` 根据指定的范围以及设定的步长，生成一个等差数列，数据类型为数组
- ◀ `matplotlib.pyplot.stem()` 绘制火柴梗图


数列
等差数列
通项式
首项
公差
前项和
求和符号
等比数列
通项式
公比
前项和
不同值
$q > 1$
$q = 1$
$0 < q < 1$
$-1 < q < 0$
$q = -1$
$q < -1$
斐波那契数列
累加 Σ
极限
定义
数列的极限
数列和的极限
估算圆周率

14.1 芝诺悖论：阿基里斯追不上乌龟

芝诺悖论 (Zeno's paradoxes) 中最有名的例子莫过于“阿基里斯追乌龟”。

阿基里斯 (Achilles) 是古希腊神话中的勇士，可谓飞毛腿。而乌龟的奔跑速度仅仅是阿基里斯的 1/10。赛跑比赛时，阿基里斯让乌龟在自己前面 100 米处起跑，他自己在后面追。

根据芝诺悖论，阿基里斯不可能追上乌龟。

芝诺的逻辑是这样的，赛跑过程中，阿基里斯必须先追到 100 米处，而此时乌龟已经向前爬行了 10 米。此时，相当于乌龟还是领先阿基里斯 10 米，这算是一个新的起跑点。

阿基里斯继续追乌龟，当他跑了 10 米之后，乌龟则又向前爬了 1 米。

于是，阿基里斯还需要再追上 1 米，于此同时乌龟又向前爬了 1/10 米。

如此往复，结论是阿基里斯永远也追不上乌龟。

为了方便可视化，假设乌龟爬行速度是阿基里斯奔跑速度的 1/2。设定，阿基里斯奔跑速度 10 m/s，神龟爬 (飞) 行速度 5 m/s。

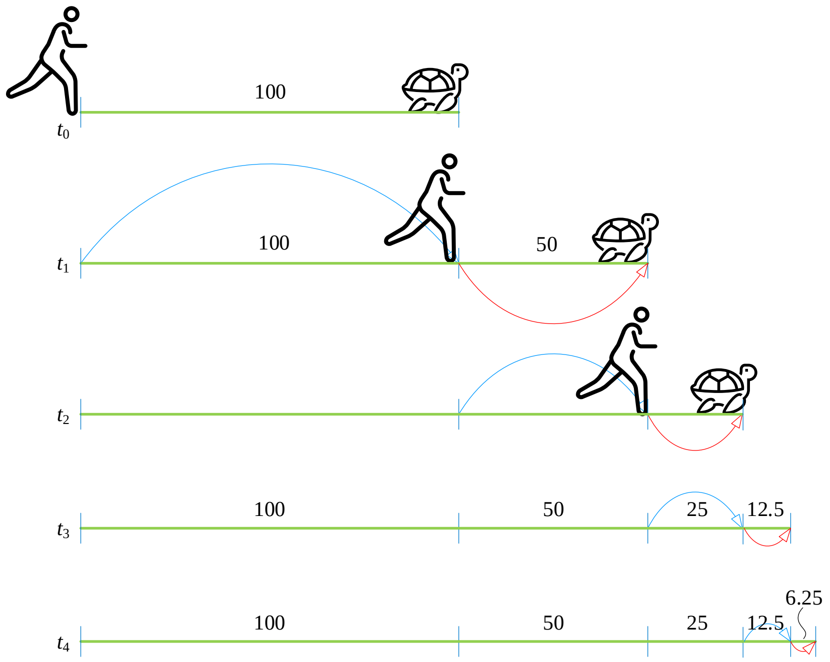


图 1. 阿基里斯追乌龟

t_0 = 0 s 时刻，乌龟在阿基里斯前方 100 米处，两者同时起跑。

t_1 = 10 s，阿基里斯跑了 10 s，追到 100 m；而这段时间，乌龟向前跑了 50 m。因此，此刻乌龟领先优势为 50 m。

t_2 = 10 + 5 s，阿基里斯又跑了 5 s，又追了 50 m。5 s 时间，乌龟跑了 25 m，而此时乌龟领先优势为 25 m。

t_3 = 10 + 5 + 2.5 s，阿基里斯又跑了 2.5 s，再追了 25 m。此刻乌龟领先优势为 12.5 m。

t_4 = 10 + 5 + 2.5 + 1.25 s，阿基里斯再追了 12.5 m，此刻乌龟领先优势为 6.25 m。

时间无限可分，距离无限可分，上述过程无穷尽也，似乎阿基里斯永远也追不上乌龟。

同向追赶问题

看到这里，大家一定会有不同意见。

这分明就是一道小学算数的"同向追赶问题"，距离之差 (100 m) 除以速度之差 (5 m/s) 就等于阿基里斯追上乌龟所需要的时间为 20 s。而 20 s 时间，阿基里斯一共跑了 200 m。

这种解题思路固然正确。但是，解题过程的前提条件是，假设阿基里斯恰好追上了乌龟！

解题技巧当然重要。实际上，看似无比荒诞的阿基里斯追乌龟问题，其中蕴含的数学思想才是内核。

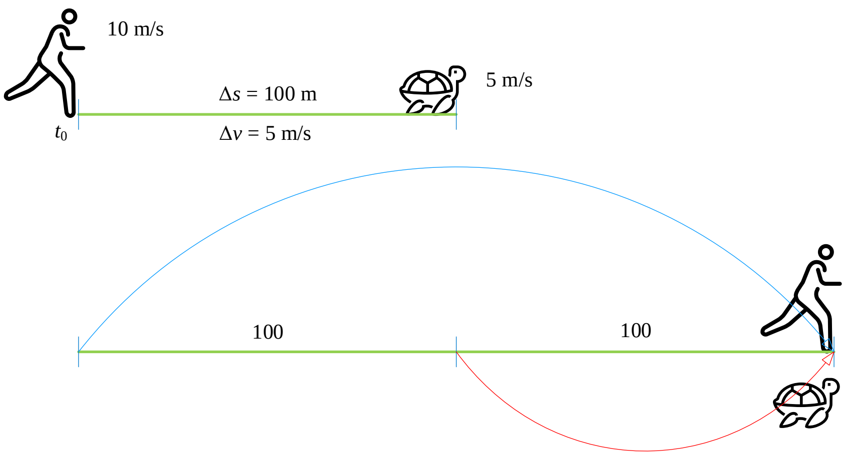


图 2. 小学数学同向追赶问题

一尺之棰，日取其半，万世不竭

下面用数列这个数学工具来分析上述问题。**数列** (sequence) 是指按照一定规则排列的一列数。

将图 1 中每段时间间隔写成一个数列：

$$10,\quad 5,\quad 2.5,\quad 1.25,\quad 0.625,\quad 0.3125,\quad 0.15625,\quad 0.078125,\ldots \tag{1}$$

图 3 (a) 用火柴梗图可视化 (1) 数列。

追赶时间的逐项和也写成一个数列累加：

$$10,\quad 15,\quad 17.5,\quad 18.75,\quad 19.375,\quad 19.6875,\quad 19.84375,\quad 19.921875,\ldots \tag{2}$$

图 3 (b) 所示火柴梗图为 (2) 数列。可以发现上述数列似乎逐渐趋向于 20，即阿基里斯追上乌龟所需要的时间。

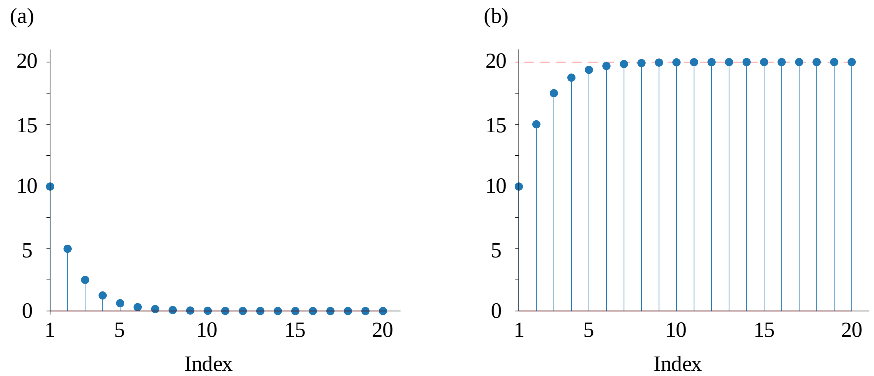


图 3. 时间间隔数列和时间逐项和数列

将阿基里斯在不同时刻之间奔跑的距离写成一列数：

$$100,\quad 50,\quad 25,\quad 12.5,\quad 6.25,\quad 3.125,\quad 1.5625,\quad 0.78125,\ldots \tag{3}$$

容易发现，这个数列就是大家熟悉的等比数列。上述数列的逐项和构成了一个新数列：

$$100,\quad 150,\quad 175,\quad 187.5,\quad 193.75,\quad 196.875,\quad 198.4375,\quad 199.21875,\ldots \tag{4}$$

可以发现上述数列似乎逐渐趋向于 200，即阿基里斯追上乌龟总共奔跑的距离。

这体现的正是庄子的哲学观点——“一尺之棰，日取其半，万世不竭。”如图 4 所示，面积为 1 的正方形，每次取一半，如此往复，没有尽头。

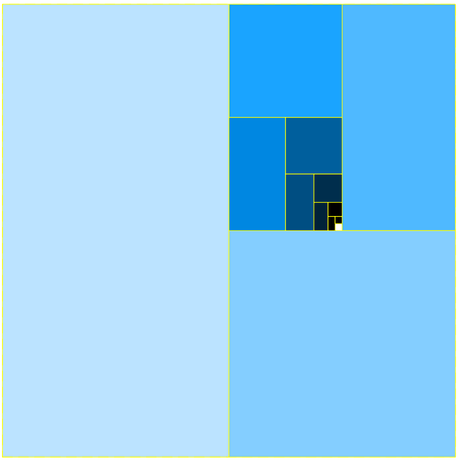


图 4. 日取其半，万世不竭

大家经常会遇到类似比较 1 和 0.99999... 大小之类的数学问题，其中蕴含的数学思想就是极限。本章就讲解数列、数列前 n 项和、极限等数学工具。

14.2 数列分类

几种常见的数列：

- **等差数列** (arithmetic sequence 或 arithmetic progression)，图 5 (a) 为递增等差数列，图 5 (b) 为递减等差数列；
- **等比数列** (geometric sequence 或 geometric progression)，图 5 (c) 为递增等比数列，图 5 (d) 为递减等比数列；
- **正负相间数列** (sign sequence 或 bipolar sequence)，如图 5 (e) 和 (f) 所示；
- **斐波那契数列** (Fibonacci sequence)，如图 5 (g) 所示；
- **随机数列** (random sequence)，如图 5 (h) 所示。

此外，根据数列项的数量，数列可以分为**有限项数列** (finite sequence) 和**无限项数列** (infinite sequence)。

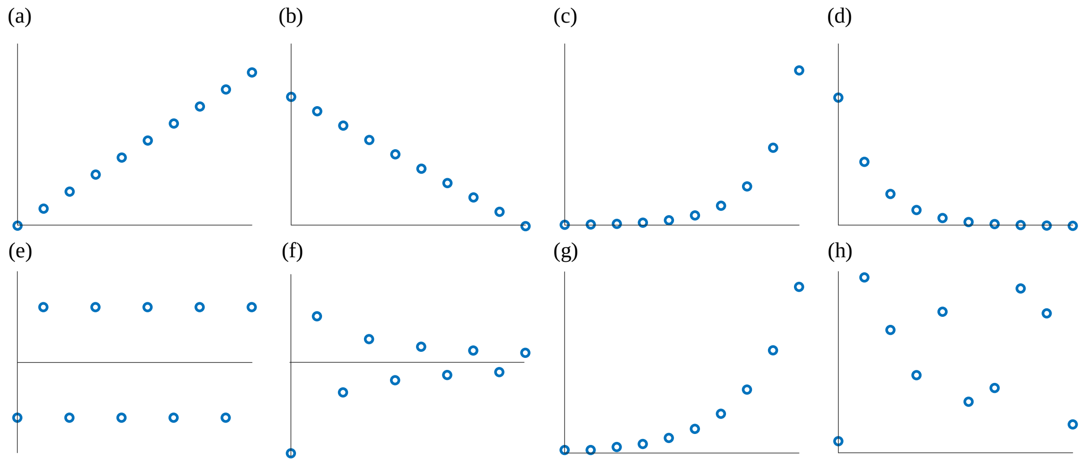


图 5. 几种数列

14.3 等差数列：相邻两项差相等

等差数列指的是数列中任何相邻两项的差相等，比如 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, …

等差数列中相邻两项差值常被称作**公差** (common difference)。将数列的第 k 项用一个具体含有参数 k 式子表示出来，称作该数列的通项公式。

等差数列通项公式 a_k 的一般式如下：

$$a_k = a + (k-1)\cdot d \tag{5}$$

a_1 (读作 a sub one) 为数列第一项，即**首项** (initial term) $a_1 = a$；d 为公差；k 为**项数** (number of terms)。`numpy.arange()` 和 `numpy.linspace()` 可以用来生成等差数列。

(5) 所示等差数列前 k 项之和为 S_k：

$$\begin{aligned} S_k &= a + (a+d) + (a+2d) + \ldots + (a+(k-1)\cdot d) \\ &= \sum_{i=1}^{k}(a+(i-1)\cdot d) = a\cdot k + \frac{k(k-1)}{2}\cdot d \end{aligned} \tag{6}$$

上式中，i 为**索引** (index) 也叫序号；Σ 是求和符号，是希腊字母 σ 的大写，读作 sigma。`numpy.sum()` 可以用来计算数列和。注意区分，Π 是求积符号，它是希腊字母 π 的大写。

欧拉 (Leonhard Euler) 最先使用 Σ 来表达求和。

相信读者还记得，等差数列求和的计算方法——首项加末项之和，乘以项数，然后除 2。相传，这个等差数列求和方法是**高斯** (Johann Carl Friedrich Gauss) 年仅 10 岁的时候发现的。

本书前文介绍，给定一列数，除了求和之外，还有**累计求和** (cumulative sum)。比如，等差数列 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 的累积和为 1, 2 (1 + 2), 6 (1 + 2 + 3), 10 (1 + 2 + 3 + 4), 15, 21, 28, 36, 45, 55。累积和的最后一项也是数列之和。`numpy.cumsum()` 可以用来计算数列累计求和。

下面，我们编写一段 Python 代码，计算等差数列 1, 2, 3, 4, 5, …, 99, 100 之和，以及数列累积和。并绘制图 6 两图；图 6 (a) 为 a_k 随序数变化，图 6 (b) 为 S_k 和随序数变化。

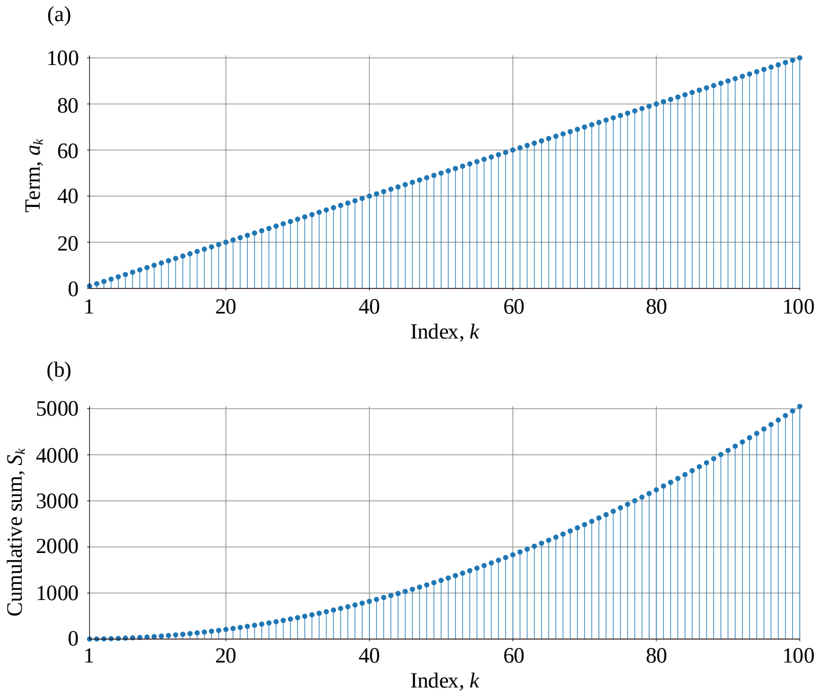


图 6. 等差数列 1,2,3,4,5,…,99,100 和数列累积和

函数视角

从函数角度，数列也是函数，k 为自变量，a_k 为因变量。需要特殊强调的是 k 的取值为正整数。如图 6 (a) 所示，(5) 可以看做特殊的一次函数。

也从函数角度来看，(6) 中 k 为自变量、取值同样为正整数，S_k 为因变量。如图 6 (b) 所示，(6) 可以看做特殊的二次函数。

数列作为一种特殊的函数，也具有各种函数性质。

$d > 0$，如图 5 (a) 所示数列 a_k 递增；$d < 0$，如图 5 (c) 所示数列 a_k 递减。

$d > 0$，如图 5 (b) 所示 S_k 图像开口向上，呈现出凸性；$d < 0$，数列递减，如图 5 (b) 所示 S_k 图像开口向下，呈现出凹性。

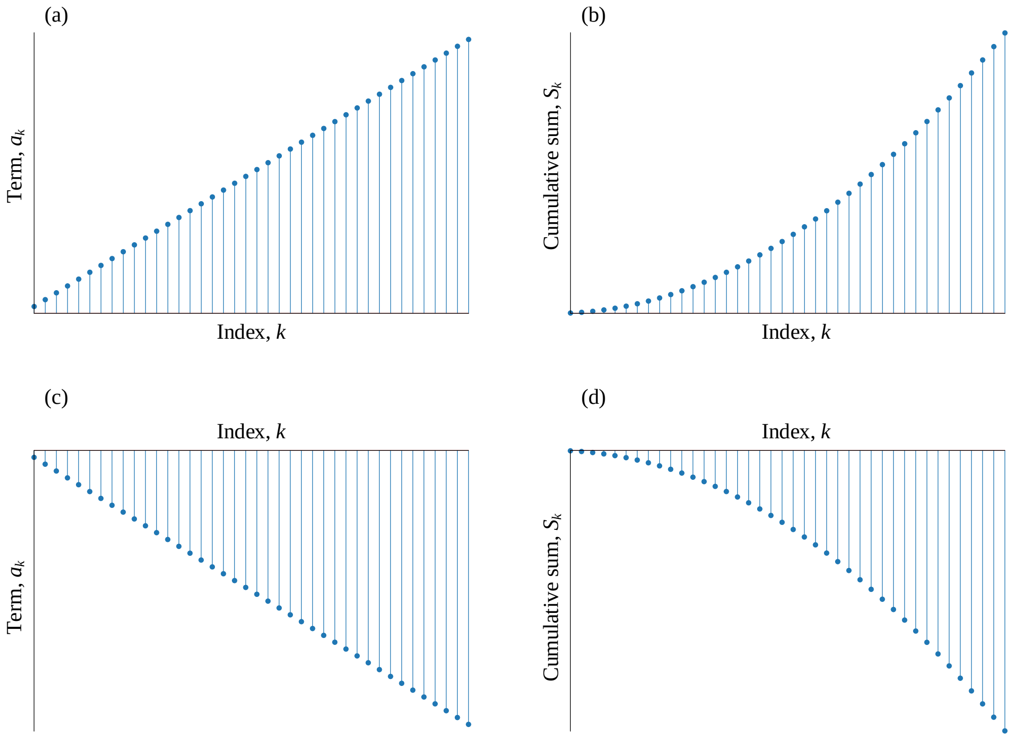


图 7. 公差为正、负两种情况

表 1. 数列英文表达

数学表达	英文表达
$a_n + a_{n-1} + \cdots + a_1 + a_0$	*a* sub n plus *a* sub *n* minus one plus dot dot dot plus *a* sub one plus *a* sub zero. *a* sub *n* plus *a* sub *n* minus one plus ellipsis plus *a* sub one plus *a* sub zero.
$a_n \cdot a_{n-1} \cdot \cdots \cdot a_1 \cdot a_0$	*a* sub *n* times a sub *n* times one plus dot dot dot times *a* sub one times a sub zero.
$a_0 + x\left(a_1 + x\left(a_2 + \ldots\right)\right)$	a sub zero plus x times quantity of a sub one plus x times quantity of a sub two plus dot dot dot
$\left(a_n - a_{n-1}\right)^2$	*a* sub *n* minus *a* sub quantity *n* minus one all squared.

Bk3_Ch14_01.py 计算并绘制图 6。

14.4 等比数列：相邻两项比值相等

等比数列指的是数列中任何相邻两项比值相等，比如 2, 4, 8, 16, 32, 64, 128, …

等比数列的比值被称作为**公比** (common ratio)。等比数列第 k 项 a_k 的一般式如下：

$$a_k = aq^{k-1} = \frac{a}{q}q^k \tag{7}$$

其中，a 为首项，q 为公比。注意 q 不为 0。

从函数角度，(7) 为特殊的指数函数——q 为底数，自变量 k 为指数，k 的取值范围为正整数。

(7) 所示等比数列前 k 项之和 S_k 为：

$$\begin{aligned} S_k &= a + aq + aq^2 + aq^3 + \cdots + aq^{k-1} \\ &= \sum_{i=0}^{k-1} a \cdot q^i = \frac{a\left(q^k - 1\right)}{(q-1)} = \frac{a}{q-1}q^k - \frac{a}{q-1} \end{aligned} \tag{8}$$

注意，q 不为 1。从函数角度，(7) 也是指数函数。

请大家回忆等比数列求和技巧。首先，计算 S_k 和 q 乘积：

$$S_k q = aq + aq^2 + aq^3 + \cdots + aq^{k-1} + aq^k \tag{9}$$

(9) 和 (8) 等式左右分别相减，并整理得到 S_k：

$$\begin{aligned} & S_k q - S_k = S_k(q-1) = aq^k - a = a\left(q^k - 1\right) \\ & \Rightarrow S_k = \frac{a\left(q^k - 1\right)}{(q-1)} \end{aligned} \tag{10}$$

函数视角

图 8 展示六种等比 q 取不同值时，等比数列的特点。

当 $q > 1$ 时，等比数列呈现出**指数增长** (exponential growth)，如图 8 (a) 所示。

当 $q = 1$ 时，等比数列蜕化成常数数列，如图 8 (b) 所示。

当 $0 < q < 1$ 时，等比数列呈现出**衰退** (decay)，如图 8 (c) 所示。

当$-1 < q < 0$ 时，等比数列呈现两种特性：**振荡** (oscillate) 和**收敛** (converge)，如图 8 (d) 所示。

当 $q = -1$ 时，等比数列只是反复振荡，也就是正负相间数列，如图 8 (e) 所示。

当 $q < -1$ 时，等比数列振荡**发散** (diverge)，如图 8 (e) 所示。

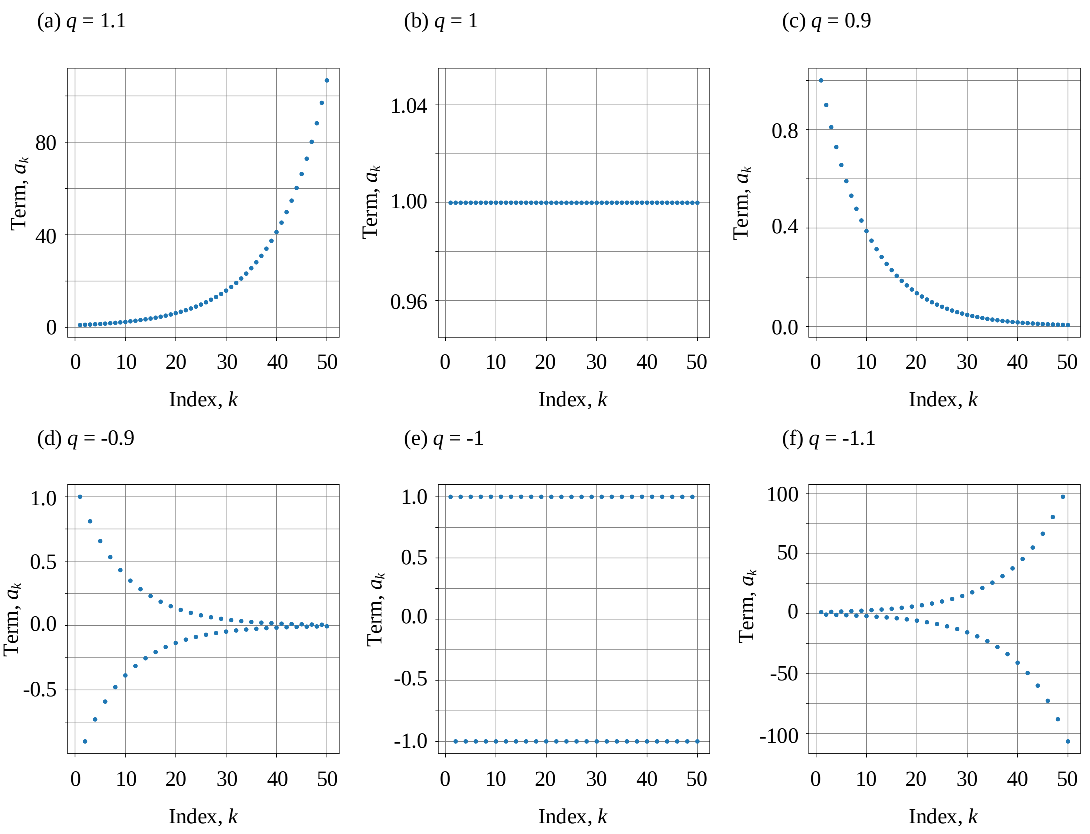


图 8. 六种等比数列趋势

Bk3_Ch14_02.py 绘制图 8 几幅子图。

在 Bk3_Ch14_02.py 基础上，我们做了一个 App 用来交互呈现 q 对数列的影响。并采用 Plotly 呈现交互散点图像。请参考 Streamlit_Bk3_Ch13_01.py。

数列在大数据和机器学习中有着广泛应用，下面举一个例子介绍等比数列在指数加权移动平均 EWMA (Exponentially Weighted Moving Average) 方法的应用。

一般情况，求解平均值 SA (Simple Average) 时，对不同时间点的观察值赋予相同的权重，比如下式：

$$SA=\frac{1}{n}\left(s_1+s_2+s_3+...+s_n\right) \tag{11}$$

其中，所有观察值中 s_1 为最旧的数据，s_n 为最新数据。

采用 EWMA 可以保证越新的观察值享有越高的权重，这样估算得到的平均值能够反映数据近期趋势：

$$EWMA=\frac{(1-\lambda)}{1-\lambda^n}\left(\lambda^{n-1}s_1+\lambda^{n-2}s_2+\lambda^{n-3}s_3+...+\lambda^0 s_n\right) \tag{12}$$

(12) 中的 λ 为衰减系数 (decay factor)，取值范围在 0 ~ 1 之间。λ 越小，衰减越明显。

可以发现索引为 i 的权重 w_i 计算式如下所示：

$$w_i=\frac{(1-\lambda)}{1-\lambda^n}\lambda^{n-i} \tag{13}$$

索引连续变化时，权重 w_i 便构成一个等比数列。图 9 所示为 EWMA 权重随衰减系数变化情况。

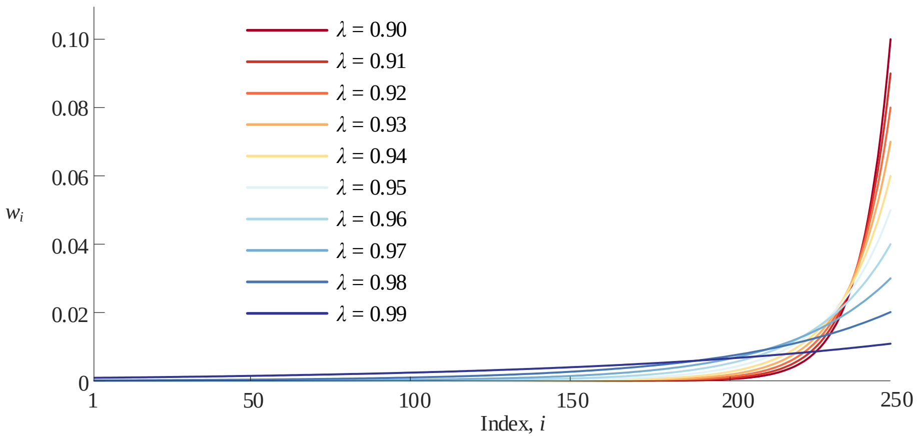


图 9. EWMA 权重随衰减系数变化

EWMA 权重一个重要的性质是，所有权重之和为 1，也就是，

$$\sum_{i=1}^{n}w_i=\frac{(1-\lambda)}{1-\lambda^n}\left(\lambda^{n-1}+\lambda^{n-2}+\lambda^{n-3}+...+\lambda^0\right)=1 \tag{14}$$

更多有关时间序列、移动平均 MA、EWMA 方法及其应用，请读者阅读本系列丛书《数据科学》一册。

14.5 斐波那契数列

本书前文介绍过斐波那契数列和杨辉三角的关系。**斐波那契数列** (Fibonacci sequence)，又被称作黄金分割数列。

斐波那契数列可以通过如下**递归** (recursion) 方法获得：

$$\begin{cases} F_0 = 0 \\ F_1 = F_2 = 1 \\ F_n = F_{n-1} + F_{n-2}, \quad n > 2 \end{cases} \tag{15}$$

于是，包括第 0 项，斐波那契数列的前 10 项为：

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55 \tag{16}$$

Bk3_Ch14_03.py 产生并打印斐波那契数列。

黄金分割

斐波那契数列和**黄金分割** (golden ratio) 有着密切联系。图 10 所示为利用斐波那契数列构造的矩形，这个矩形是对黄金分割矩形的近似。黄金矩形的长宽比例 φ 为：

$$\varphi = \frac{\sqrt{5}+1}{2} \approx 1.61803 \tag{17}$$

图 10 所示矩形的长宽比例为：

$$\frac{21+13}{21} \approx 1.61905 \tag{18}$$

图 10 所示的螺旋线叫做**斐波那契螺旋线** (Fibonacci spiral)，它是对**黄金螺旋线** (golden spiral) 的近似。

本系列丛书将在《矩阵力量》一本介绍如何用**特征值分解** (eigen decomposition) 求解斐波那契数列通项式。

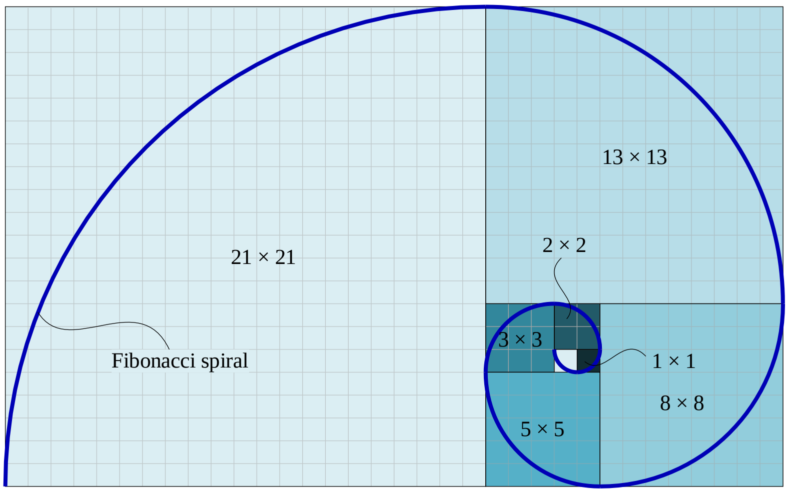


图 10. 斐波那契数列和黄金分割关系

14.6 累加：大写西格玛

求和符号 (summation symbol) ——大写西格玛 Σ (capital sigma) —— 是表达求和的便捷记法。

以下式为例，a_i 描述求和中的每一项，下角标 i 代表**索引** (index variable 或 index)，也叫序号：

$$\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_{n-1} + a_n \tag{19}$$

Σ 下侧和上侧的数字分别代表了**求和索引下限** (lower bound of summation) 和**求和索引上限** (upper bound of summation)，如图 11 所示。

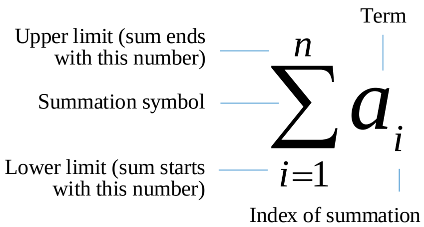


图 11. 大西格玛求和记号

常用表达索引的字母有 i、j、k、m、n 等，采用什么索引字母并不影响求和结果，比如：

$$\sum_{i=1}^{100} a_i = \sum_{j=1}^{100} a_j = \sum_{k=1}^{100} a_k \tag{20}$$

Σ 中索引上下限可以都是具体正整数，比如：

$$\sum_{i=1}^{5} a_i = a_1 + a_2 + a_3 + a_4 + a_5 \tag{21}$$

Σ 中索引上下限也可以都是代数符号，比如：

$$\sum_{i=m}^{n} a_i = a_m + a_{m+1} + \cdots + a_{n-1} + a_n \tag{22}$$

Σ 的索引也可以是满足集合运算的标签：

$$\sum_{i \in S} a_i \tag{23}$$

降维

如图 12 所示，当索引 i 在一定范围变化时，比如 $1 \sim n$，数列 $\{a_i\}$ $(i = 1 \sim n)$ 本身相当于一个数组，而索引 i 像是方向。

对 $\{a_i\}$ $(i = 1 \sim n)$ 求和，相当于在 i 方向上将数组“压扁”，得到一个标量 $\sum_i a_i$ 。$\sum_i a_i$ 除以 n (也就是数列元素个数)，便得到平均数。

从空间角度来看，数列 $\{a_i\}$ 是一维数组，索引 i 就是它的维度。而求和运算 $\sum_i a_i$ 相当于“降维”，得到的“和”只是一个数值，没有维度。

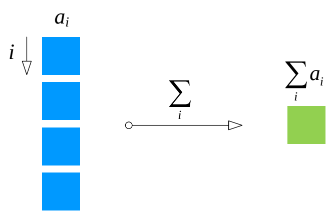


图 12. 从数组角度看求和

线性代数运算

看到这里，大家是否想得到，此处图 12 数列 $\{a_i\}$ 相当于一个列向量 $\boldsymbol{a}$，即：

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \tag{24}$$

本书前文在讲解向量和矩阵时提到过，对列向量 $\boldsymbol{a}$ 所有元素求和可以利用如下向量内积或矩阵运算得到，

$$\sum_{i=1}^{n} a_i = \boldsymbol{1} \cdot \boldsymbol{a} = \boldsymbol{a} \cdot \boldsymbol{1} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \boldsymbol{a}^{\mathrm{T}} \boldsymbol{1} = \boldsymbol{1}^{\mathrm{T}} \boldsymbol{a} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} @ \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \tag{25}$$

其中，$\boldsymbol{1}$ 就是全 1 列向量，和 $\boldsymbol{a}$ 等长。

这样，我们就把数列求和、向量、向量内积、矩阵乘法这几个概念联系起来。

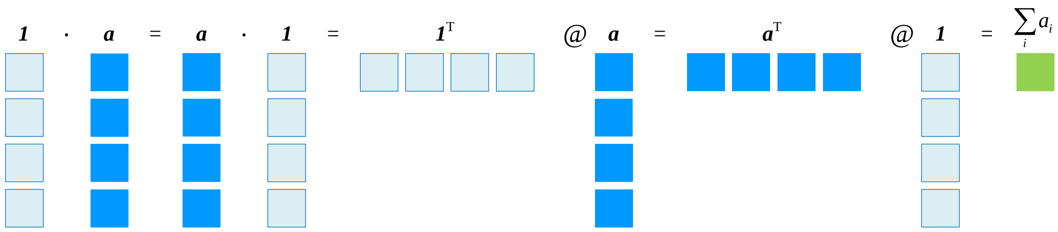


图 13. 从向量内积和矩阵乘法角度看求和

下面介绍几个有关求和符号的重要法则。

对数运算把连乘变连加

类似 Σ，Π (capital pi) 用来标记多项相乘，比如：

$$\prod_{i=1}^{n} a_i = a_1 a_2 \cdots a_{n-1} a_n \tag{26}$$

本书前文提到，对数运算将连乘转化为连加，比如：

$$\ln\left(\prod_{i=1}^{n} a_i\right) = \ln a_1 + \ln a_2 + \cdots + \ln a_{n-1} + \ln a_n = \sum_{i=1}^{n} \ln a_i \tag{27}$$

乘系数

常数 c 乘 a_i，再求和 $\sum_{i=1}^{n} c a_i$，等同于常数 c 乘 $\sum_{i=1}^{n} a_i$：

$$\sum_{i=1}^{n} c a_i = c\left(\sum_{i=1}^{n} a_i\right) = c \sum_{i=1}^{n} a_i \tag{28}$$

其中，$c \times$ 相当于“缩放”，$\sum_{i}$ 相当于“降维”。如图 14 所示，(28) 相当于在说，“缩放 → 降维”等价于“降维 → 缩放”。

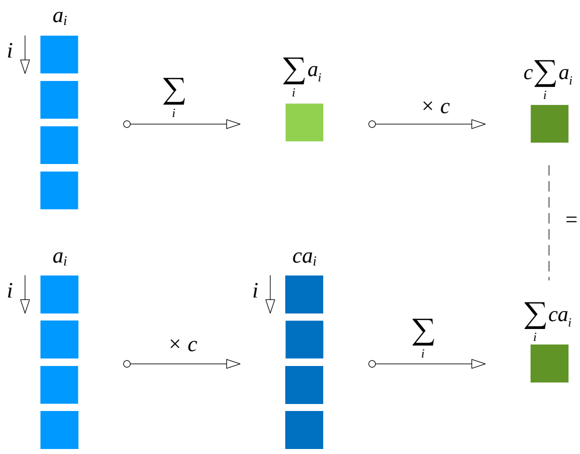


图 14. 乘系数

特别地，如果 Σ 内为一常数 a，则，

$$\sum_{i=1}^{n} a = na \tag{29}$$

分段求和

可以根据索引排列，将求和分割成几个部分分别求和，比如：

$$\begin{aligned}\sum_{i=1}^{n} a_i &= \left(a_1 + a_2 + \cdots + a_k\right) + \left(a_{k+1} + a_{k+2} + \cdots + a_n\right) \\ &= \sum_{i=1}^{k} a_i + \sum_{i=k+1}^{n} a_i\end{aligned} \tag{30}$$

分段求和常用在对齐不同长度数组，以便化简计算。

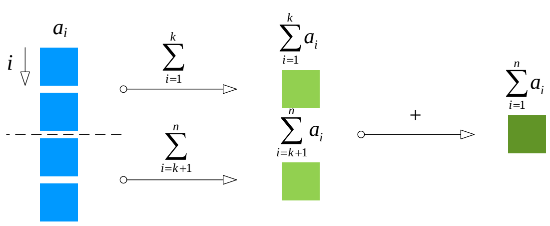


图 15. 分段求和

两项相加减

拥有相同索引的两项相加再求和，等于分别求和再相加，即，

$$\sum_{i=1}^{n}\left(a_i+b_i\right)=\sum_{i=1}^{n}a_i+\sum_{i=1}^{n}b_i \tag{31}$$

上述法则也适用于减法，即，

$$\sum_{i=1}^{n}\left(a_i-b_i\right)=\sum_{i=1}^{n}a_i-\sum_{i=1}^{n}b_i \tag{32}$$

平方

Σ 内为 a_i 的平方，则，

$$\sum_{i=1}^{n}\left(a_i^2\right)=\sum_{i=1}^{n}a_i^2=a_1^2+a_2^2+a_3^2+\cdots a_n^2 \tag{33}$$

如图 16 所示，利用前文 (24) 定义的列向量 $\boldsymbol{a}$，(33) 等价于：

$$\sum_{i=1}^{n}a_i^2=\boldsymbol{a}\cdot\boldsymbol{a}=\boldsymbol{a}^{\mathrm{T}}\boldsymbol{a} \tag{34}$$

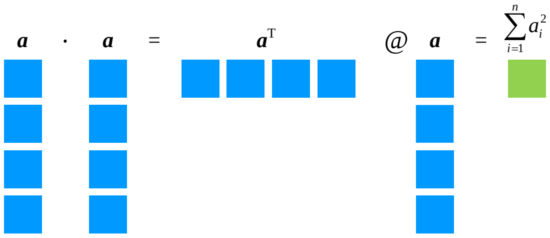


图 16. 从向量内积和矩阵乘法角度看 $\sum_{i=1}^{n}a_i^2$

$\sum_{i=1}^{n}a_i$ 的平方则为：

$$\left(\sum_{i=1}^{n}a_i\right)^2=\left(a_1+a_2+a_3+\cdots a_n\right)^2 \tag{35}$$

显然，(33) 和 (35) 不相同：

$$\sum_{i=1}^{n}\left(a_i^2\right)\neq\left(\sum_{i=1}^{n}a_i\right)^2 \tag{36}$$

乘法

拥有相同索引的 a_i 和 b_i 相乘，再求和：

$$\sum_{i=1}^{n}(a_i b_i) = a_1 b_1 + a_2 b_2 + a_3 b_3 + \cdots a_n b_n \tag{37}$$

定义列向量 $\boldsymbol{b}$，$\boldsymbol{b}$ 和 $\boldsymbol{a}$ 形状相同，$\boldsymbol{b}$ 的元素为 b_i。如图 17 所示，$\sum_{i=1}^{n}(a_i b_i)$ 等价于：

$$\sum_{i=1}^{n}(a_i b_i) = \boldsymbol{a}\cdot\boldsymbol{b} = \boldsymbol{b}\cdot\boldsymbol{a} = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b} = \boldsymbol{b}^{\mathrm{T}}\boldsymbol{a} \tag{38}$$

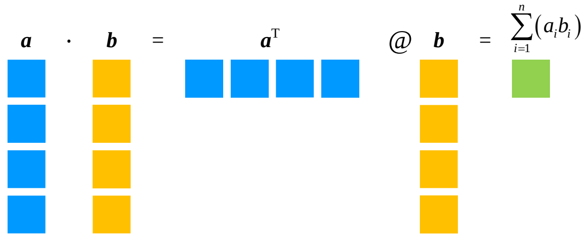


图 17. 从向量内积和矩阵乘法角度看 $\sum_{i=1}^{n}(a_i b_i)$

$\sum_{i=1}^{n} a_i$ 和 $\sum_{i=1}^{n} b_i$ 相乘展开得到：

$$\left(\sum_{i=1}^{n} a_i\right)\left(\sum_{i=1}^{n} b_i\right) = \sum_{i=1}^{n} a_i \sum_{i=1}^{n} b_i = (a_1 + a_2 + \cdots a_n)(b_1 + b_2 + \cdots b_n) \tag{39}$$

显然，(37) 和 (39) 不相同：

$$\sum_{i=1}^{n}(a_i b_i) \neq \left(\sum_{i=1}^{n} a_i\right)\left(\sum_{i=1}^{n} b_i\right) \tag{40}$$

二重求和

一些情况，我们需要用到二重求和记号 ΣΣ，比如：

$$\begin{aligned}\sum_{i=1}^{3}\sum_{j=2}^{4} a_i b_j &= \sum_{i=1}^{3} a_i b_2 + a_i b_3 + a_i b_4 \\ &= (a_1 b_2 + a_1 b_3 + a_1 b_4) + (a_2 b_2 + a_2 b_3 + a_2 b_4) + (a_3 b_2 + a_3 b_3 + a_3 b_4)\end{aligned} \tag{41}$$

注意，式中内层 Σ 索引为 j，外层 Σ 索引为 i。先对索引 j 求和，再对索引 i 求和。

注意上式中，每个元素只有一个索引，相当于只有一个维度。

两个索引

实践中，经常遇到的情况时一项有两个、甚至更多索引，比如 $a_{i,j}$ 有两个索引 i 和 j。

根据本节前文分析思路，举个例子，索引 i 的取值范围为 $1 \sim n$，索引 j 的取值范围为 $1 \sim m$，数组 $a_{i,j}$ 相当于有 i 和 j 两个维度。

这是否让大家想到了本书前文讲过的矩阵，如图 18 所示，$a_{i,j}$ 相当于是矩阵 $\boldsymbol{A}$ 的第 i 行、第 j 列元素。

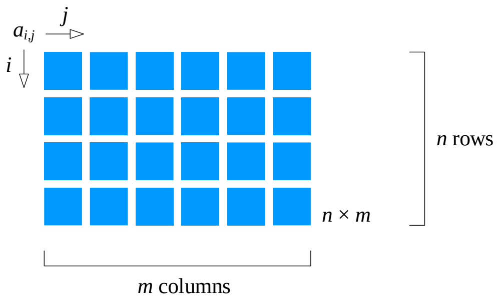


图 18. $n \times m$ 矩阵 $\boldsymbol{A}$

本节后续内容就围绕图 19 热图给出的二维数组展开，这个数组有 8 行、12 列。

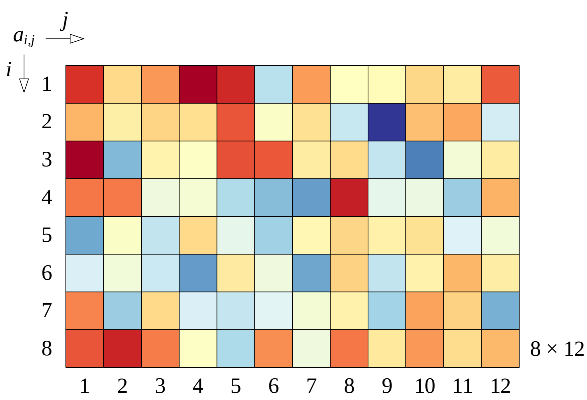


图 19. 8×12 数组

偏求和

下面先介绍单维求和，也就是沿着一个索引求和。我们给它取个名字，叫“偏求和”。这个“偏”字呼应本书第 16、18 章要介绍的“偏导数”、“偏微分”、“偏积分”等概念。

首先聊聊 $a_{i,j}$ 对索引 i 偏求和：

$$\underbrace{\sum_{i=1}^{n} a_{i,j}}_{\text{Sum over } i} \Rightarrow \sum_{i=1}^{n} a_{i,1}, \quad \sum_{i=1}^{n} a_{i,2}, \quad \cdots \quad \sum_{i=1}^{n} a_{i,m} \tag{42}$$

我们发现，得到的不是一个求和，而是 m 个和。也就是说，图 18 中每一列数值求和，每一列都有一个“偏求和”结果。

白话说，$a_{i,j}$ 就是个“表格”，$\sum_{i} a_{i,j}$ 就是按列求和，每列一个和。

如图 20 所示，$\sum_{i=1}^{n} a_{i,7}$ 代表对数组第 7 列元素求和。注意，为了简化数学表达，我们也常用 $\sum_{i} a_{i,j}$ 代表对索引 i 的求和，求和上下限不再给出。

$\sum_{i} a_{i,j}$ 除以 n，得到的就是每一列元素的平均数。

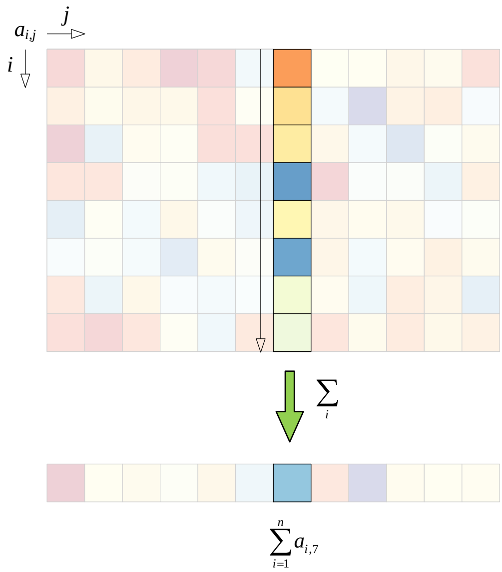


图 20. 将二维数组的第 7 列求和

这相当于矩阵 $a_{i,j}$ 沿着索引 i 被“压扁”。如图 21 所示，原来数据有两个维度——索引 i 和 j；而现在只剩一个维度——索引 j。

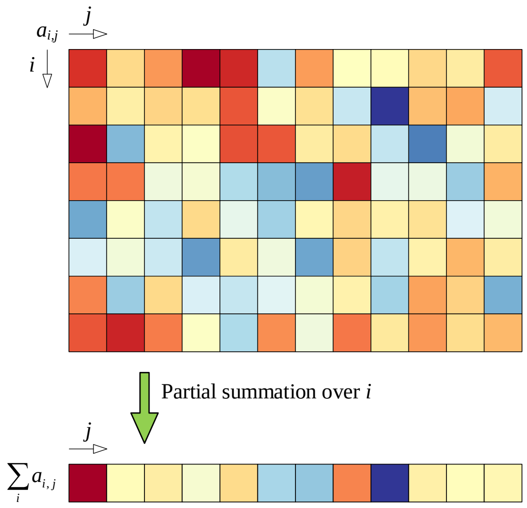


图 21. 将二维数组沿着索引 i 代表的方向“压扁”

同理，如图 22 所示，$a_{i,j}$ 对索引 j 偏求和 $\sum_j a_{i,j}$，相当于沿着索引 j 方向将数组“压扁”；$\sum_j a_{i,j}$ 只剩 i 这一个维度。

白话说，$a_{i,j}$ 就是个“表格”，$\sum_j a_{i,j}$ 就是按行求和，每行一个和。

$\sum_j a_{i,j}$ 除以 m，得到的就是每一行元素的平均数。

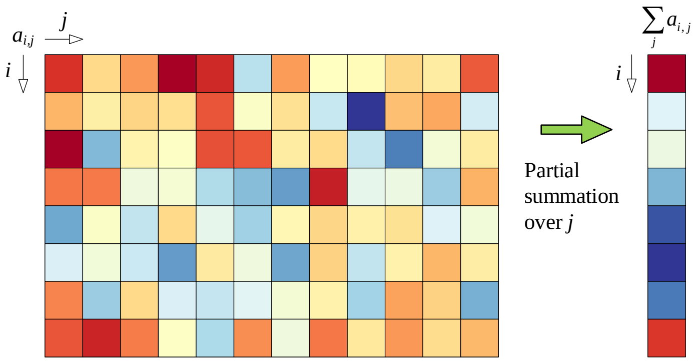


图 22. 将二维数组沿着索引 j 代表的方向“压扁”

多重求和

而 $\sum_i a_{i,j}$ 和 $\sum_j a_{i,j}$ 沿着各自剩余最后一个方向再次“压扁”，得到的就是 $a_{i,j}$ 所有元素的和。

这种情况，求和顺序不影响结果，即：

$$\underbrace{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{i,j}}_{\text{Sum over } j} = \underbrace{\sum_{j=1}^{m}\sum_{i=1}^{n} a_{i,j}}_{\text{Sum over } i} \tag{43}$$

上式也可以写作:

$$\sum_{j,i} a_{i,j} = \sum_{i,j} a_{i,j} \tag{44}$$

下标"j, i"表示先对 j 求和、再对 i 求和；下标"i, j"表示先对 i 求和、再对 j 求和。

图 23 所示为上述计算的过程分解。注意，只有数组是"方方正正"的结构时，上式才成立；否则，求和先后顺序会影响到结果。这一点和多重积分中积分先后顺序原理一致。

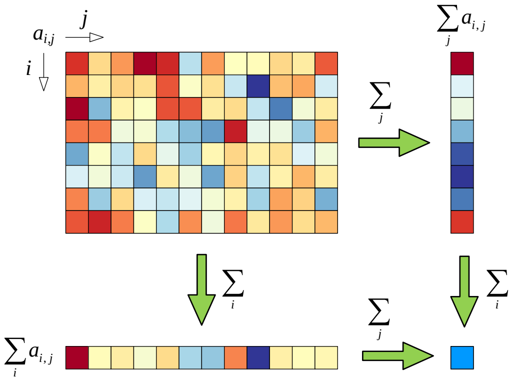


图 23. 求和顺序不影响结果

举个例子:

$$\begin{aligned}\sum_{i=1}^{3}\sum_{j=2}^{4} a_{i,j} &= \sum_{i=1}^{3} a_{i,2} + a_{i,3} + a_{i,4} \\ &= (a_{1,2} + a_{1,3} + a_{1,4}) + (a_{2,2} + a_{2,3} + a_{2,4}) + (a_{3,2} + a_{3,3} + a_{3,4})\end{aligned} \tag{45}$$

调换求和顺序，得到:

$$\begin{aligned}\sum_{j=2}^{4}\sum_{i=1}^{3} a_{i,j} &= \sum_{j=2}^{4} a_{1,j} + a_{2,j} + a_{3,j} \\ &= (a_{1,2} + a_{2,2} + a_{3,2}) + (a_{1,3} + a_{2,3} + a_{3,3}) + (a_{1,4} + a_{2,4} + a_{3,4})\end{aligned} \tag{46}$$

可以发现 (45) 和 (46) 两式相等。

矩阵运算视角

前文介绍的"偏求和"和多重求和都可以通过矩阵运算得到结果。

如图 24 所示，$a_{i,j}$ 对索引 i 偏求和等价于如下矩阵运算:

$$\boldsymbol{1}^{\mathrm{T}}\boldsymbol{A}=\sum_{i}a_{i,j} \tag{47}$$

注意，上式中全 1 列向量 $\boldsymbol{1}$ 的形状为 8×1，转置得到 $\boldsymbol{1}^{\mathrm{T}}$ 的形状为 1×8。

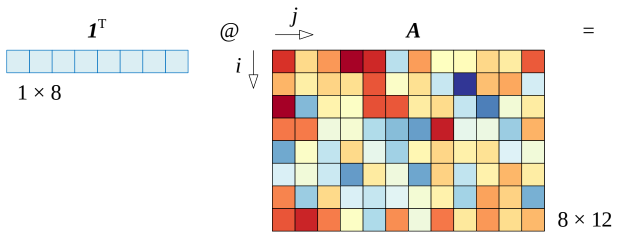


图 24. 计算矩阵 $\boldsymbol{A}$ 每列元素和

如图 24 所示，$a_{i,j}$ 对索引 j 偏求和等价于如下矩阵运算：

$$\boldsymbol{A}\boldsymbol{1}=\sum_{j}a_{i,j} \tag{48}$$

注意，上式中全 1 列向量 $\boldsymbol{1}$ 的形状为 12×1。

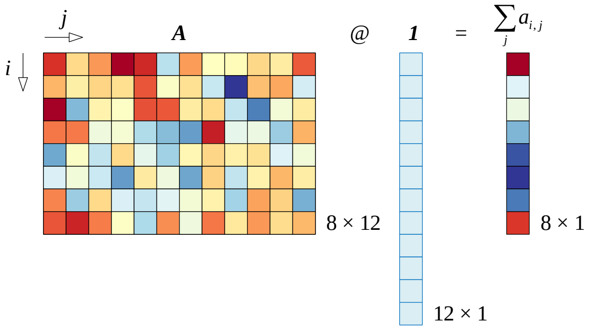


图 25. 计算矩阵 $\boldsymbol{A}$ 每行元素和

如图 26 所示，求矩阵 $\boldsymbol{A}$ 所有元素之和对应的矩阵运算为：

$$\sum_{i}\sum_{j}a_{i,j}=\sum_{j}\sum_{i}a_{i,j}=\boldsymbol{1}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{1} \tag{49}$$

再次强调，上式中两个全 $\boldsymbol{1}$ 向量形状不同。

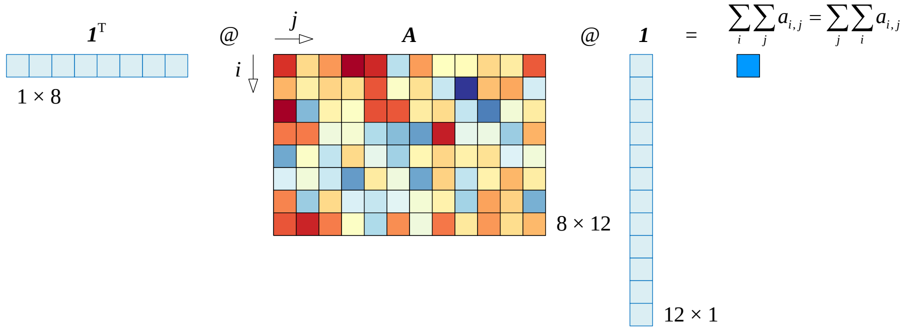


图 26. 计算二维矩阵 $\boldsymbol{A}$ 所有元素之和

两个以上索引

某一项可能有超过两个索引，比如 $a_{i,j,k}$ 有三个索引，数组 $\{a_{i,j,k}\}$ 相当于有三个维度。

图 27 所示为三维数组的多重求和运算，求和的顺序为"j, i, k"。

首先，$a_{i,j,k}$ 沿 j 索引求和，得到 $\sum_j a_{i,j,k}$。相当于一个立方体"压扁"为一个平面，三维降维到二维。

然后，再沿 k 索引求和，进一步将平面"压扁"得到一维数组。此时，数组只有一个索引 i。

最后，沿着 i 再求和，得到一个标量 $\sum_{j,i,k} a_{i,j,k}$。

请大家自行绘制按照"i, j, k"这个顺序求和得到 $\sum_{i,j,k} a_{i,j,k}$ 过程示意图。

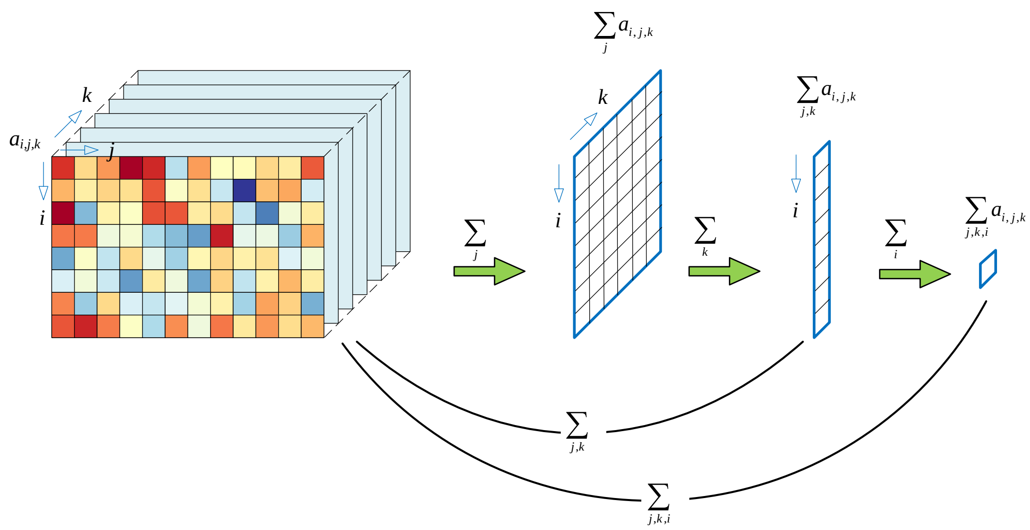


图 27. 三维数组的求和运算

对于多维数组，建议大家了解一下 Python 中 xarray 这个工具包。此外，Pandas 数据帧也是处理多维数组不错的工具。本系列丛书会介绍 xarray 和 Pandas 这两个工具。

爱因斯坦曾经提出过以自己名字命名的求和法则，叫做爱因斯坦求和约定 (Einstein summation convention)。Numpy 中 numpy.einsum() 函数的运算规则就是基于爱因斯坦求和约定。这个数学工具简化多维数组求和运算，本系列丛书《矩阵力量》将展开介绍。

表 2. 求和和求积的英文表达

数学表达	英文表达
$\sum_{i=1}^{n} a_i$	The sum of all the terms (small) *a* sub (small) *i*, where *i* takes the integers from one to (small) *n*. The sum from (small) *i* equals one to (small) *n*, (small) *a* sub *i*. The sum as (small) *i* runs from one to *n* of (small) *a* sub (small) *i*.
$\sum_{n=1}^{5} 2n$	The sum of 2 times *n* as *n* goes from 1 to 5. The summation of the expression 2*n* for integer values of *n* from 1 to 5.
$\prod_{i=1}^{n} a_i$	The multiplication of all the terms *a* sub *i*, where *i* takes the values from one to *n*. The product from *i* equals 1 to n of *a* sub *i*.
$\prod_{i=1}^{\infty} y_i$	The product from *i* equals one to infinity of *y* sub *i*.

Bk3_Ch14_04.py 绘制本节二维数组热图，并计算“偏求和”。请大家自行计算这个二维数组所有元素之和。

14.7 数列极限：微积分的一块基石

数列极限

数列 $\{a_n\}$ 极限存在的确切定义如下。

设 $\{a_n\}$ 为一数列，如果存在常数 C，对于任意给定的正数 ε，不管 ε 有多小，总存在正整数 N，使得 $n > N$ 时，如下不等式均成立，

$$|a_n - C| < \varepsilon \tag{50}$$

那么就称常数 C 是数列 $\{a_n\}$ 的极限；也可以说，数列 $\{a_n\}$ 收敛于 C：

$$\lim_{n\to\infty} a_n = C \tag{51}$$

其中，lim 是英文 limit 的缩写，$n \to \infty$ 表达 n 趋向无穷。

如果，极限 C 不存在，则称数列 $\{a_n\}$ 极限不存在。

几个例子

给定如下等比数列：

$$a_n = \frac{1}{2^n} \tag{52}$$

当 n 趋向于无穷，数列值趋向于零：

$$\lim_{n\to\infty} a_n = \lim_{n\to\infty} \frac{1}{2^n} = 0 \tag{53}$$

对于如下数列，当 n 趋向无穷时，数列值在两个定值之间震荡；因此，不存在极限：

$$a_n = (-1)^n \tag{54}$$

对于如下数列，当 n 趋向无穷时，数列值急速增加至无穷，即发散；因此，也不存在极限：

$$a_n = 2^n \tag{55}$$

收敛的数列，可以自下而上收敛、自上而下收敛、振荡收敛。

图 28 给出了三个收敛数列的例子。图 28 (c) 对应的数列如下：

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = \mathrm{e} \tag{56}$$

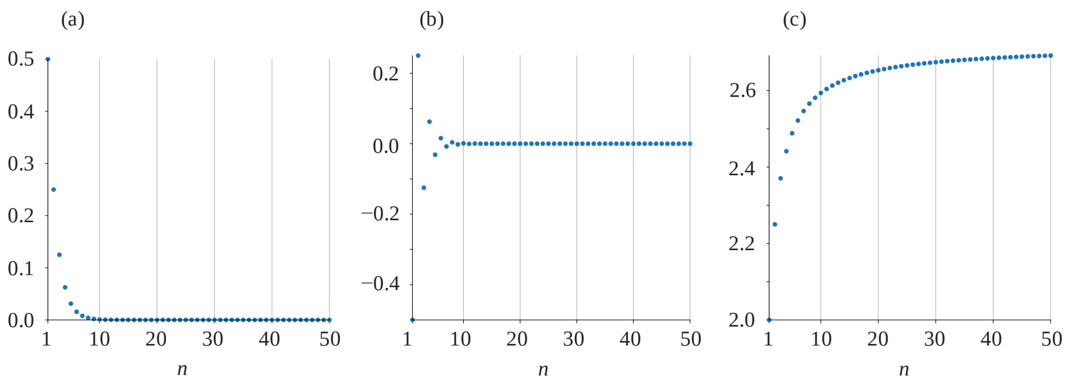


图 28. 收敛数列

数列和的极限

此外，数列之和也可以收敛。下式就是一个收敛的数列之和，数列之和随 n 变化趋势如图 29 (a) 所示：

$$1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\cdots=\sum_{n=0}^{\infty}\frac{1}{2^n}=2 \tag{57}$$

如图 29 (b) 所示，以下数列之和也是收敛于 1：

$$\frac{1}{1\times 2}+\frac{1}{2\times 3}+\frac{1}{3\times 4}+\frac{1}{4\times 5}+\frac{1}{5\times 6}+\cdots=\sum_{n=1}^{\infty}\frac{1}{n(n+1)}=1 \tag{58}$$

如图 29 (c) 所示，自然对数底数 e 也可以用数列和的极限来近似：

$$\sum_{k=0}^{\infty}\frac{1}{k!}=1+\frac{1}{1}+\frac{1}{1\times 2}+\frac{1}{1\times 2\times 3}+\frac{1}{1\times 2\times 3\times 4}+\cdots=e \tag{59}$$

但是 $1/n$ 这个数列之和并不收敛：

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\cdots=\sum_{n=1}^{\infty}\frac{1}{n} \tag{60}$$

如果在以上数列中每一项增加正负号交替，这个数列之和收敛：

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\cdots=\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{n}=\ln(2) \tag{61}$$

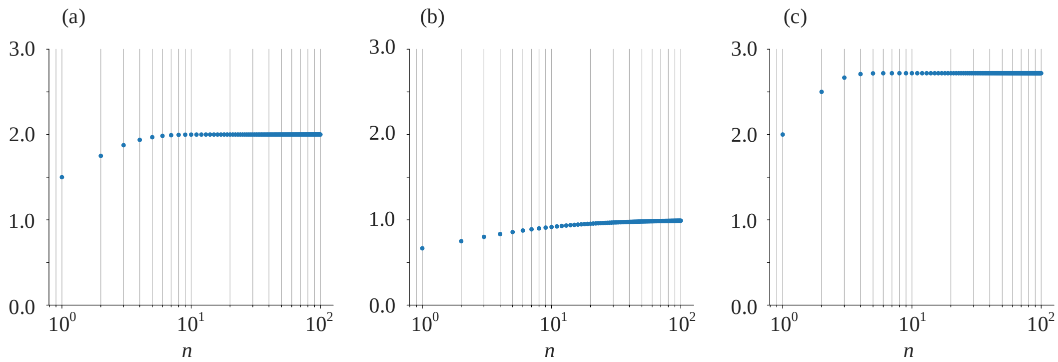


图 29. 数列之和收敛

Bk3_Ch14_05.py 绘制图 29。代码中利用 sympy.limit_seq() 函数计算极限值。

14.8 数列极限估算圆周率

本书前文介绍，古代数学家通过割圆术不断提高圆周率的估算精度。随着数学方法的发展，很多数学家发现可以用数列和来逼近圆周率。这是圆周率估算的一次颠覆性进步。

比如莱布尼兹发现，如下数列之和逼近 $\pi/4$：

$$\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots + \frac{(-1)^{n+1}}{2n-1} \tag{62}$$

即

$$\pi = 4\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{2k-1} \tag{63}$$

图 30 所示为上式随着 k 不断增加逼近圆周率值。

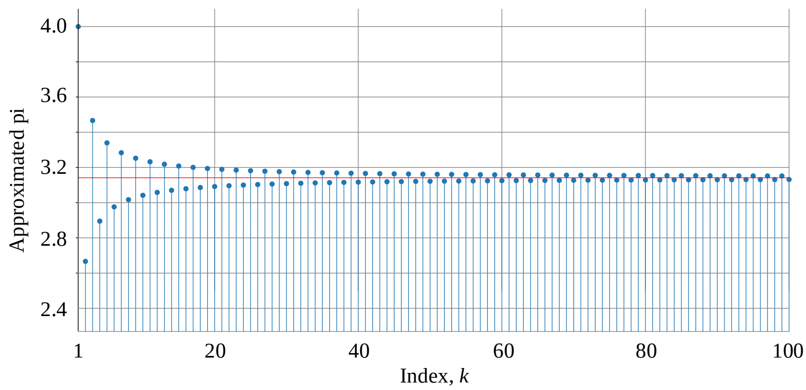


图 30. 数列之和逼近圆周率

Bk3_Ch14_06.py 绘制图 30。

本章介绍的大西格玛求和、极限这两个数学工具是微积分的基础。

大家在学习大西格玛求和时，请务必从几何、数据、维度、矩阵运算这几个视角分析求和运算；不然，复杂多层求和运算会让大家晕头转向。

此外，有了数列和极限这两个数学概念，我们在圆周率估算方法上又进一步。