工业软件数学基础

导论

课程简介

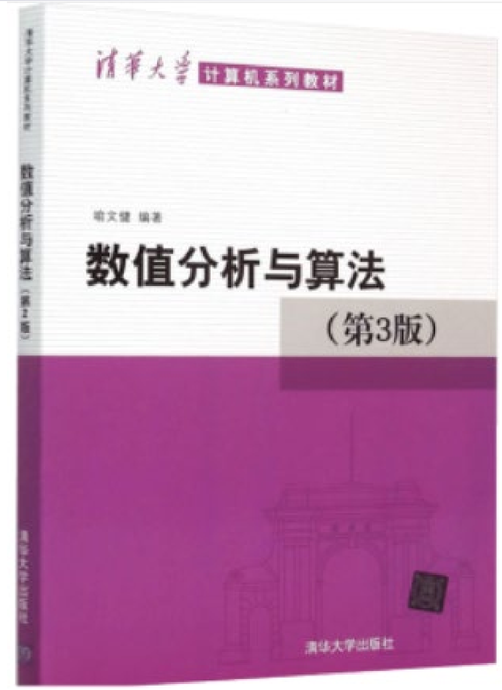
- 课程目标
  - 介绍广泛应用于工业软件的各种数值计算方法
  - 巩固连续数学基础知识、增强实际应用能力
- 考评方法
  - 平时：雨课堂/作业 (30%)
  - 期末：考试(70%)

教材

- 数值分析与算法(第3版)

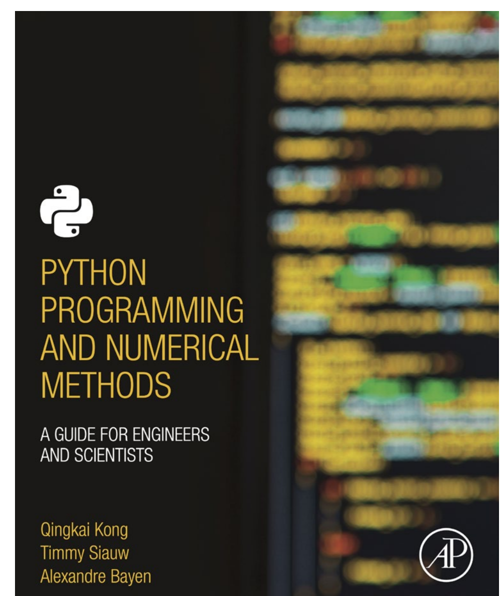
  喻文健 编著

  清华大学出版社, **2020**年

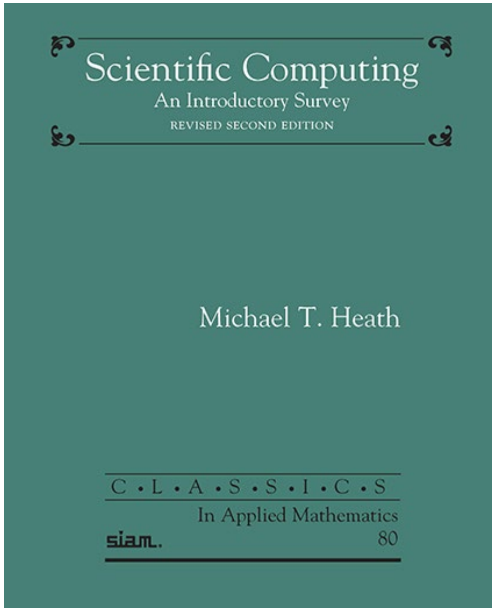


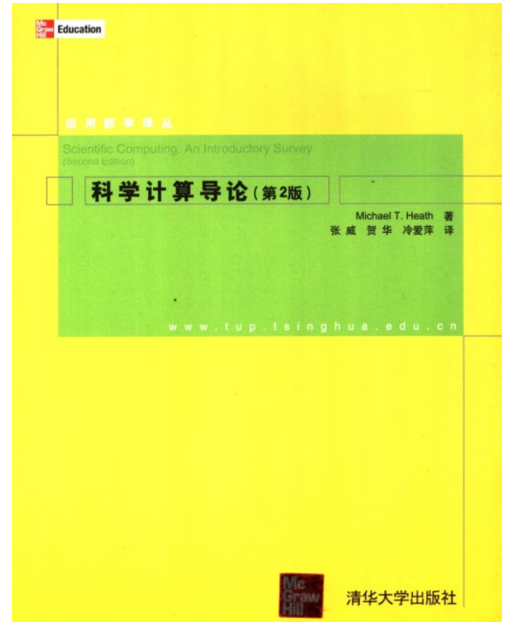
- 参考书

  https://pythonnumericalmethods.berkeley.edu/notebooks/Index.html



参考书





http://heath.cs.illinois.edu/scicomp/

主要教学内容

- 导论
- 非线性方程解法
- 线性方程组的直接解法
- 线性方程组的迭代解法
- 线性最小二乘问题
- 矩阵特征值计算
- 函数插值
- 数值积分与微分
- 常微分方程初值问题

学习建议

- 学习建议
  - 不缺课、按时完成作业
  - 重点理解问题背景、算法思路和具体步骤
  - 适当进行公式推导、算法复杂度分析与比较
  - 自己编程实验, 提升学习兴趣!

类似数学课：公式推导、理论证明

又像专业课：算法分析和设计

数值计算概况

数值分析、数值计算 、科学计算

当今科学研究的三种基本手段：

- 理论分析
- 科学实验
- 数值仿真 (数值计算，科学计算)

科学计算的发展涉及硬件和软件两个方面，这里只考虑软件方面

研究求解连续数学问题的算法.

数值计算与数值算法

- 数值计算的特点
  - 处理连续数学的量(实数量)，可能涉及微分、积分和非线性。被求解的问题一般没有解析解、或理论上无法通过有限步四则运算求解
    - 无解析解： $33x^5+3x^4-17x^3+2x^2+4x-39=0$
    - 有解析解，但需无限步计算：**sin(*x*)**
    - 更多的实际应用问题通过数值仿真(simulation)来解决
  - 目标：寻找迅速完成的(迭代)算法，评估结果的准确度
- 好数值算法的特点
  - 计算效率高、计算复杂度低
  - 可靠性好：在考虑实际计算的各种误差情况下，结果尽可能地准确

数值计算的步骤

- 建立数学模型（需要相关问题背景）
- 研究数值计算、求解方程的算法　（本课程重点）
- 通过计算机语言编程实现算法
- 在计算机上运行程序进行数值实验、仿真
- 将计算结果用较直观的方式输出，如图形可视化方法
- 解释和验证计算结果，如果需要重复上面的某些步骤

上述各步骤相互间紧密地关联，影响着最终的计算结果和效率（问题的实际背景和要求也左右着方法的选择）

设计数值方法(算法)的关键：将问题简化或加以近似，然后求解简化后的问题，估计带来的误差。

数值软件/程序包

- 数值计算的软件与程序包
  - 解决常见问题，促进各个科学和工程领域的科研
  - 了解基本原理，学习算法设计和实现技巧
  - 成为聪明的软件/程序包使用者
- 存在形式和资源
  - 免费(netlib, github, …)，付费(NR, …)
  - Fortran, C, C++, Matlab, Python, Julia, …
  - 源代码，API调用，集成开发环境

- MATLAB：www.mathworks.com MathWorks
  - 集成环境：交互式计算系统，高级编程语言
  - 数值计算、矩阵计算功能强大(包含很多先进算法)
  - 大量专题工具箱(**Toolbox**)，为专业应用提供便利
  - 开发环境、可视化功能等比较强

	Matlab(作为编程语言)	C, C++, Fortran
	第四代编程语言	第三代编程语言
编译方式	解释器，或JIT加速器(R2015b后)	编译器
申明变量？	不需要	需要
开发时间	较快	较慢
运行时间	较慢	较快
开发环境	集成环境(编辑器、调试器、命令历史、变量空间、profiler、编译器)	

数值软件/程序包

- 矩阵计算有关程序包
  - LAPACK, www.netlib.org/lapack/index.html
    - LAPACKE: C language APIs for LAPACK
    - BLAS: www.netlib.org/blas/ (ATLAS, GotoBLAS, OpenBLAS*)
  - ScaLAPACK: Distributed-memory variant
  - MAGMA: Matrix Algebra for GPU & Multicore Arch. https://icl.cs.utk.edu/magma/
- 更高层的程序库与环境
  - Intel公司**MKL(**Math Kernel Library, in C)
  - Eigen (a C++ template library): eigen.tuxfamily.org
  - Matlab, Octave, Python (Numpy), …

- 数值计算应用广泛
  - 人工智能、机器人、大数据、多媒体：矩阵计算、奇异值分解、数据拟合(回归) 、常微分方程数值解
  - 计算机图形学CAD：函数插值、逼近、微分方程数值解
  - 电子设计自动化（EDA）：大规模线性方程组求解、常微分方程、偏微分方程
  - 高性能计算：性能评测、算法实现与优化、电力系统仿真

误差分析基础

误差分析基础

- 误差的来源
- 误差及其分类
  - 误差与有效数字
  - 数据传递误差与计算误差
  - 截断误差与舍入误差
- 问题的敏感性与数据传递误差
- 算法的稳定性

误差的来源

计算前
- 模型误差　　忽略次要因素
- 数据误差　　常数或测量值、前一步计算的结果

计算中
- 截断误差　　方法误差　例：sin(x)= …
- 舍入误差　　计算时表示数的位数有限

“四舍五入” (默认)

例1.1 用球表面积公式计算地球表面积

$$A = 4\pi r^2$$

- ➢将地球近似成球体　　模型误差
- ➢取半径$r \approx 6370km$　　数据误差
- ➢将π的值取到有限位 (如3.14)　　截断误差
- ➢计算$4\pi r^2$ (计算乘法)　　舍入误差

误差及其分类

1.绝对误差与相对误差

- x ~准确值，$\hat{x}$ ~近似值，绝对误差$e(\hat{x})=\hat{x}-x$
  - (绝对)误差往往不能反映准确程度

  例: 测量长约1公里的物体，误差1cm; 测量长约1米的物体，误差也是1cm
- 相对误差$e_r(\hat{x})=\frac{\hat{x}-x}{x}$
  - 误差、相对误差都可正可负
  - 当准确值为0时，相对误差无意义
  - 一般准确值未知, 估计误差上限, 误差限$\varepsilon(\hat{x})$, $\varepsilon_r(\hat{x})$
  - 误差较小时, $e_r(\hat{x})\approx\frac{\hat{x}-x}{\hat{x}}$ $\Rightarrow$ $\varepsilon_r(\hat{x}) \approx \frac{\varepsilon(\hat{x})}{\hat{x}}$

例 “四舍五入”的绝对误差限

实数 x 的十进制标准表示式是 $x = \pm a_0.a_1a_2\cdots a_{n-1}a_n\cdots \times 10^m$,

其中 m 为整数, $a_0, a_1, a_2, \cdots, a_{n-1}, a_n, \cdots$ 都是 0 至 9 中的一个数, 且 $a_0 \neq 0$. 如果按“四舍五入”对 a_n 作舍入, 得到 x 的近似值

$$\hat{x} = \begin{cases} \pm a_0.a_1a_2\cdots a_{n-1} \times 10^m, a_n \leqslant 4, \\ \pm a_0.a_1a_2\cdots (a_{n-1}+1) \times 10^m, a_n \geqslant 5, \end{cases}$$

- 四舍时 $|e| = |\hat{x} - x| = (a_0.a_1a_2\cdots a_{n-1}a_n\cdots - a_0.a_1a_2\cdots a_{n-1}) \times 10^m$

$$\leqslant 0.\underbrace{00\cdots0}_{n-1\text{ 个}}5 \times 10^m = \frac{1}{2} \times 10^{m-(n-1)}$$

- 五入时 $|e| = |\hat{x} - x| = (a_0.a_1a_2\cdots(a_{n-1}+1) - a_0.a_1a_2\cdots a_{n-1}a_n\cdots) \times 10^m$

$$\leqslant 0.\underbrace{00\cdots0}_{n-1\text{ 个}}5 \times 10^m = \frac{1}{2} \times 10^{m-(n-1)}.$$

结论: 对 x 进行四舍五入后, 绝对误差限为被保留的数字中最后数位的半个单位.

误差及其分类

设 $\hat{x}$ 为 x 的近似值, 若 $\hat{x}$的绝对误差限是它的某一位的半个单位，而该位向左到$\hat{x}$的第一位非零数字共有n位，则称$\hat{x}$ 为 x 的近似值具有***n*位有效数字(significant digit)** 或者说近似值$\hat{x}$准确到该位。

设 $x = \pm 10^m \times \left(d_0 + \frac{d_1}{10} + \cdots + \frac{d_{n-1}}{10^{n-1}} + \cdots\right)$

其中, $d_i(i = 0,1,2,\cdots)$ 为 $0 \sim 9$的某个数字, 且 $d_0 \neq 0$

$$|\hat{x} - x| \leq \frac{1}{2} \times \frac{1}{10^{n-1}} \times 10^m = \frac{1}{2} \times 10^{m-(n-1)}$$

则$\hat{x}$ 为 x 的近似值具有n位**有效数字**$d_0, d_1, d_2, \cdots, d_{n-1}$

误差及其分类

保留p位有效数字与相对误差有何关系?

定理 设对x保留p位有效数字后得到的近似值$\hat{x}$, 则$\hat{x}$的相对误差 $|e_r(\hat{x})| \le \frac{1}{2d_0}\times 10^{-p+1}$, 其中$d_0$为$x$的第一位有效数字.

证明: 设 $x = \pm 10^m \times \left(d_0 + \frac{d_1}{10} + \cdots + \frac{d_{p-1}}{10^{p-1}} + \cdots\right)$

考虑四舍五入, $|\hat{x} - x| \le 10^m \times \frac{1}{2\times 10^{p-1}}$

而$|x| \ge 10^m \times d_0$

$$\Longrightarrow |e_r(\hat{x})| = \frac{|\hat{x}-x|}{|x|} \le \frac{1}{2d_0} \times 10^{-p+1}$$

误差及其分类

例 $x = \pi = 3.14159265\cdots$, 保留**3**位有效数字

$$\hat{x} = 3.14\ ,\ |e(\hat{x})| \le \frac{1}{2}\times 10^{-3+1},\ |e_r(\hat{x})| \le \frac{1}{2}\times\frac{1}{3}\times 10^{-3+1}$$

- 精度 (precision) 和准确度 (accuracy)
  - 准确度: 与误差大小有关
  - 精度: 与表示数的数字位数有关 (单/双精度)

3.123456有7位十进制精度, 但它近似 π 准确度并不高。

例 求 $\sqrt{5}$ 的近似值, 使其相对误差小于 1%, 问应取几位有效数字?

解: 设应取 n 位有效数字, 则因 $2<\sqrt{5}<3$, 故 n 应满足不等式

$$\frac{1}{2\times 2}10^{-(n-1)}<0.01.$$

$n>3-2\lg 2=3-2\times 0.3=2.4$, 因此可取 $n=3$, 即 $\sqrt{5}$ 的近似值取为 2.24 时, 其相对误差小于 1%.

误差及其分类

2.数据传递误差与计算误差

以简单的函数求值问题为例

- $x \to f(x)$, $\hat{x} \to \hat{f}(\hat{x})$
- 误差 $\hat{f}(\hat{x}) - f(x) = \underbrace{[\hat{f}(\hat{x}) - f(\hat{x})]}_{\text{计算误差}} + \underbrace{[f(\hat{x}) - f(x)]}_{\text{数据传递误差}}$

说明：这里的数据传递误差不考虑具体的计算方法, 认为计算过程精确，它仅受问题本身影响

3.截断误差与舍入误差 $\hat{f}(\hat{x}) - f(\hat{x})$

- 数值方法近似、有限精度运算 (计算误差的两部分)

误差及其分类

例 用差商近似一阶导数 $f'(x) \approx \dfrac{f(x+h)-f(x)}{h}$

- h为步长, 分析两种误差与h关系
- 截断误差$e_T = hf''(\xi)/2$
- $\varepsilon_T = Mh/2$, M是$|f''(\xi)|$上界
- 设计算$f(x)$ 舍入误差限为ϵ,

$$\varepsilon_{tot} = \frac{Mh}{2} + \frac{2\epsilon}{h}$$

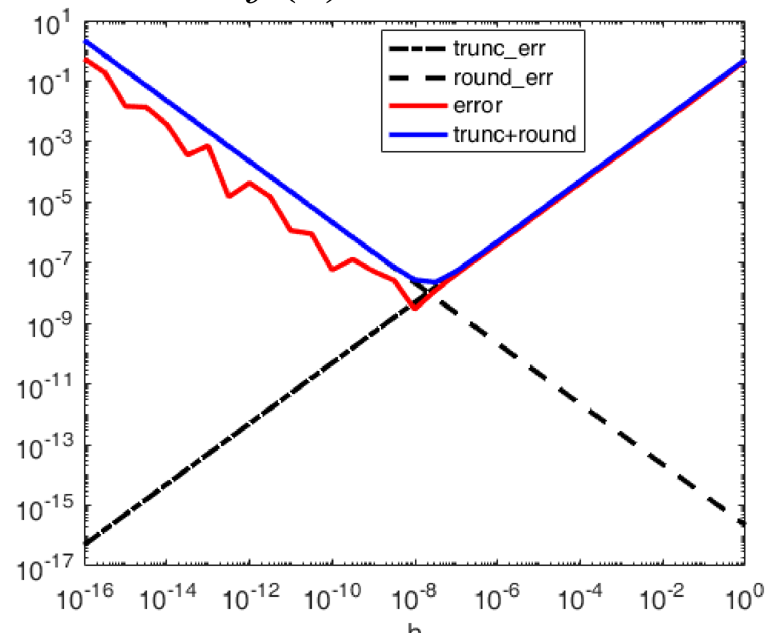
舍入误差

实验: 看h取何值使ε_{tot}最小

$\epsilon \approx 10^{-16}$, 最佳$h = 2\times10^{-8}$

$$h = 2\sqrt{\varepsilon/M}$$

$f(x)=\sin x$, $x=1$



问题的敏感性 (数据传递误差)

- 定义 问题的敏感性: 输入数据扰动对问题解的影响程度. 不敏感(良态， well-conditioned), 敏感(病态， ill-conditioned)
- 定义 用条件数(condition number)反映问题的敏感性

$$\mathrm{cond} = \frac{\|\text{问题的解的相对变化量}\|}{\|\text{输入数据的相对变化量}\|}$$

范数

即问题对数据误差的”放大因子”. cond越大问题越病态

例: 函数求值问题: $x \to f(x)$, $\hat{x} \to f(\hat{x})$

结果相对误差 $\frac{f(\hat{x})-f(x)}{f(x)}$, 数据相对变化 $\frac{\hat{x}-x}{x}$

$$\mathrm{cond} = \left|\frac{[f(\hat{x})-f(x)]/f(x)}{(\hat{x}-x)/x}\right| \approx \left|\frac{xf'(x)}{f(x)}\right| \quad \text{(近似公式)}$$

问题的敏感性 (数据传递误差)

- 也可定义绝对条件数

例如对函数求值问题，绝对条件数$\mathrm{cond}_A=\left|\frac{f(\hat{x})-f(x)}{\hat{x}-x}\right|$

- 条件数反映问题的特性, 与计算方法无关. 它受输入数据影响, 因而常考虑其上限

例(函数求值问题的敏感性): 假设 x 接近 $\pi/2$, 试用条件数估计式分析计算正切函数 $f(x) = \tan x$ 的问题敏感性。

$$\text{【解】} \operatorname{cond} \approx \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x(1+\tan^2 x)}{\tan x}\right| = \left|x\left(\frac{1}{\tan x} + \tan x\right)\right|$$

当 x 接近 $\pi/2$ 时, 条件数趋于无穷大, 此时计算 $\tan x$ 是**高度敏感**的。

例如, 对 $x = 1.57079(\pi/2 \approx 1.570796)$, 按这个公式算出的近似条件数为 2.48276×10^5 。

$$\tan(1.57079) = 1.58058 \times 10^5,$$
$$\tan(1.57078) = 6.12490 \times 10^4,$$

根据它们计算出 $x = 1.57079$ 处条件数为

$$\operatorname{cond} = \frac{(15.8058 - 6.12490) \times 10^4 / 1.58058 \times 10^5}{10^{-5}/1.57079} \approx 9.621 \times 10^4,$$

与上面近似条件数的结果很接近。如此大的条件数说明, 输入扰动经过计算过程在输出结果上放大了几十万倍, 因此在 $x = \pi/2$ 点附近计算 $\tan x$ 是很敏感的问题。

例(函数求值问题及其反问题的条件数)：函数 $f(x)=\sqrt{x}$, 分析该函数求值问题及其反问题的敏感性。

【解】由 $f'(x)=1/(2\sqrt{x})$，条件数

$$\text{cond} \approx \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x/2\sqrt{x}}{\sqrt{x}}\right| = \frac{1}{2}\ 。$$

它说明计算结果的相对变化是输入数据相对变化的 1/2, 所以求平方根问题是**不敏感(良态)**问题。

反问题 $g(y)=y^2$ 的条件数: $\left|\frac{yg'(y)}{g(y)}\right| = \left|\frac{y\cdot 2y}{y^2}\right| = 2$, 说明反问题也是**不敏感(良态)**问题。

算法的稳定性(数值稳定性)

有限字长的数的四舍五入, 或精度

- 结果对计算过程中的扰动不敏感的算法更稳定

例 对长度100的数组求和 (考虑2位数精度的计算)

算法1: 按存储顺序对这100个数直接累加

若实际数据为2.0, 0.01, …, 0.01 (99个), 则结果? sum=2.0

算法2: 先按元素绝对值递增的顺序排序, 再依次求和

对上述数据取值, sum=0.99+2.0= 3.0, 更准确!

算法2比算法1更稳定!

- 对包含一系列计算步骤的过程, 若中间步骤的小扰动不放大或放大不严重, 则该过程对应的算法更稳定

一般指中间结果的相对误差!

算法的稳定性

例 计算黄金分割比$\phi=\frac{\sqrt{5}-1}{2}$的n次幂(n=1, 2, ..., 20)

<u>算法1</u>: 直接乘法,$f(x)=x^n$, x=0.618034 (ϕ的近似值)

<u>算法2</u>: 利用递推式

$$\begin{cases}\phi^{n+1}=\phi^{n-1}-\phi^{n}\\ \phi^{0}=1,\ \phi^{1}=x\end{cases}$$

每步仅做一次减法

算法2的效果

双精度浮点运算

n	ϕ^n的计算值
2	0.381966
3	0.236068
…	…
18	0.000144
19	0.000154
20	-0.000010

分析误差传播趋势 (算ϕ^n误差为e_n)

$$\begin{cases}e_{n+1}=e_{n-1}-e_n\\ e_0=0, e_1=x-\phi\end{cases}$$

$e_2=-e_1$, $e_3=e_1-e_2=2e_1$, $e_4=-3e_1$, $e_5=5e_1$, …, $|e_{20}|=c|e_1|$, c是Fibonacci序列第 20 项

$|e_r(\hat{\phi}^n)|$>100%

算法的稳定性

- **算法2** 相对误差的放大更严重!

  $|e_{20}| = c|e_1|$, c是Fibonacci序列的第20项 $\sim 6.7 \times 10^3$

- 算法1

由于x<1, n ↗, 误差 ↘ $|e_{20}| \approx \phi^{19}|e_1|$ $\phi^{19}(\approx 1 \times 10^{-4})$

- 对比两种算法, 显然算法2非常不稳定
- 未考虑中间步的舍入误差(很小)

算法的稳定性

- 一般每次四则运算的舍入误差很小, 但一个算法含很多步, 从输入量开始“向前”分析舍入误差很难
- 向后误差分析——考察舍入误差影响算法稳定性
  - 以函数求值为例

    $y=f(x)$，计算结果为$\hat{y}=\hat{f}(x)$
  - 求$\hat{x}$使其满足$f(\hat{x})=\hat{y}$, 则$\Delta x=\hat{x}-x$称为向后误差
  - 向后误差大小反映算法过程的稳定性
- 对一些问题, 可通过向后误差分析来研究算法的稳定性, 例如对于求解线性方程组的高斯消去法。

计算机浮点数系统

计算机浮点数系统与舍入误差

- 浮点数的表示
- 机器精度（$\varepsilon_{\mathrm{mach}}$）
- 抵消现象 (cancellation)

计算机中的浮点数

- 实数x在计算机中的表示即浮点数$\mathrm{fl}(x)$ (**2进制**)

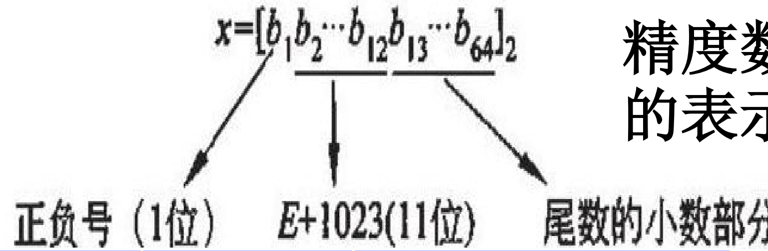
$$\mathrm{fl}(x) = \pm\left(d_0 + \frac{d_1}{2} + \frac{d_2}{2^2} + \cdots + \frac{d_{p-1}}{2^{p-1}}\right) \times 2^E$$

- 基数: β进制 (β=2); 指数E: 上限值U, 下限值L
- p位尾数, 也称p为精度位数 p位有效数字
- IEEE浮点数系统已成为标准, 分单精度和双精度
- 规范化的规则要求$d_0 = 1$
- 好处：数的表示唯一、尾数都是有效数字、d_0不用存储 (该位表示±的信息)

计算机中的浮点数

- 浮点数为有限个，且非均匀地分布在实数轴上
- 机器精度$\varepsilon_{\mathrm{mach}} = 2^{-p}$　(1与右边相邻数间隔的一半)
- 下溢值(underflow level, UFL) : 2^{L}
- 上溢值(overflow level, OFL) : $(2-2^{-p+1})\times 2^{U}$
- 规范化浮点数的总个数

$$\boldsymbol{2(\beta-1)\beta^{p-1}(U-L+1)+1}$$

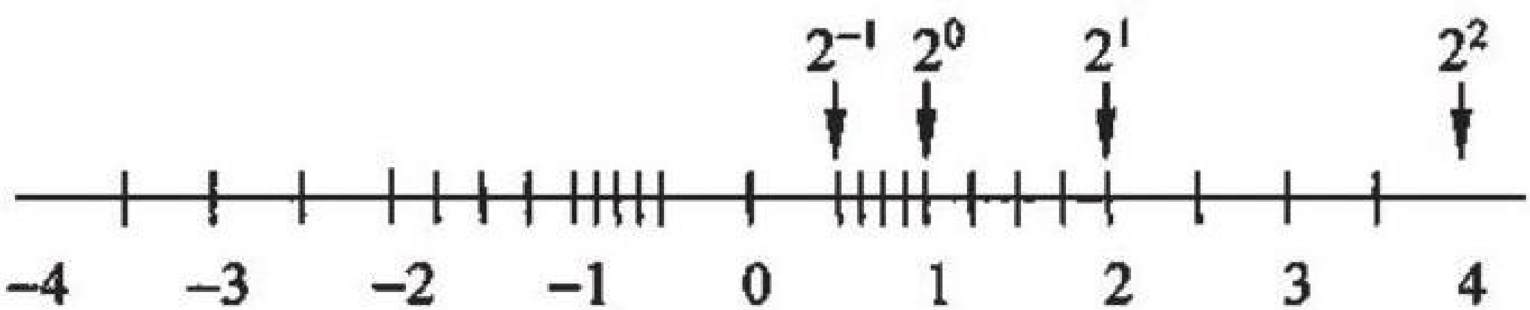


IEEE双精度数的表示

	浮点数系统	β	p	L	U	$\varepsilon_{\mathrm{mach}}$
32 bits	IEEE单精度	2	24	-126	127	5.960×10^{-8}
64 bits	IEEE双精度	2	53	-1022	1023	$\boxed{1.110\times10^{-16}}$

例 (浮点数系统): 一个简单的浮点数系统，$\beta = 2, p = 3, L = -1, U = 1$

- 这个系统中有 25 个浮点数。
- 最大的浮点数 $\mathrm{OFL} = (1.11)_2 \times 2^1 = (3.5)_{10}$，
- 最小的正浮点数为 $UFL = (1.00)_2 \times 2^{-1} = (0.5)_{10}$ 。
- 可以观察到浮点数不是等间隔分布的，但在相邻的 2 的整数次幂之间浮点数呈均匀分布。这是一般的浮点数系统都具有的特点。

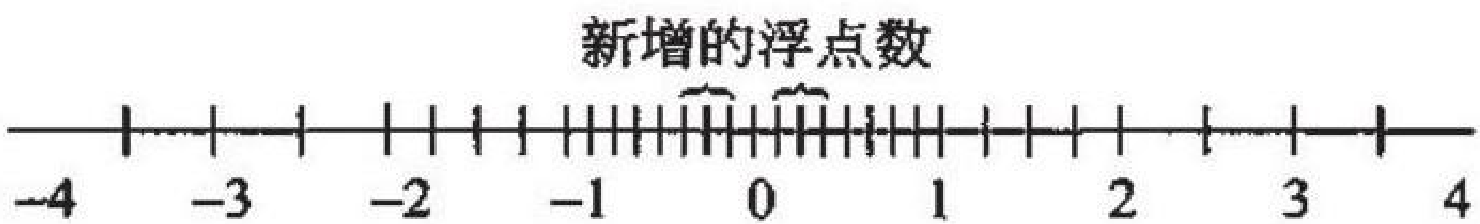


可以看到 0 与它附近的浮点数间隔较大, 这是由规范化造成的。由于尾数最小为 1.00 ⋯, 指数最小为 L, 所以 0 和 β^L 之间不可能有浮点数。IEEE 标准中定义了一种**次规范化规则**, 使得 0 和 β^L 之间的数得以表示。

次规范化放松了规范化的限制, 在指数域取一个特殊值的情况下允许尾数的首位为 0 ,指数固定位为 L，这样 0 周围的间隔就可以被新增的浮点数填充。

次规范化的机制使得下溢值变小, 这种现象也称为**渐进下溢**。

次规范化虽然增加了所表示数的范围, 但新增加的数的精度要低于其他浮点数, 因为它们的有效数字较少。



计算机中的浮点数

- 定理:设实数x在浮点数系统中的表示为浮点数$\mathrm{fl}(x)$，则$\left|\frac{\mathrm{fl}(x)-x}{x}\right| \le \varepsilon_{\mathrm{mach}}$ (默认四舍五入，或"最近舍入")

- 定理: $x_1, x_2 \in \mathbb{R}$, 若$\left|\frac{x_2}{x_1}\right| \le \frac{1}{2}\varepsilon_{\mathrm{mach}}$, 则$x_2$的值对浮点运算$x_1 + x_2$的结果毫无影响. ("大数吃掉小数")

若$\left|\frac{x_2}{x_1}\right| > \varepsilon_{\mathrm{mach}}$,一定不"吃小数"

- **+-*/运算, 以及简单函数的误差与$\varepsilon_{\mathrm{mach}}$同级别(sin, tan, atan, exp)**

抵消现象

- 两个符号相同、值相近的p位数相减使结果的有效数字远少于p位, 称为抵消(cancellation)

例：$x = 1.92305 \times 10^3$，$y = 1.92137 \times 10^3$，则 $x - y = ?$ 1.68

减法计算未发生误差, 但其结果仅有三位有效数字

- 结果的有效数字位数的减少, 意味着相对误差的放大, 往往会影响后续计算的准确度
- 抵消现象是发生<u>信息丢失</u>、<u>误差变大</u>的信号！

浮点加和浮点乘满足交换律, 但不满足结合律

例：假设 x 是一个略小于机器精度的正浮点数, 则在浮点运算体系下 $\mathrm{fl}((1+x)+x)=1$, 但 $\mathrm{fl}(1+(x+x))>1$ 。

保证计算结果的准确性

保证计算结果的准确性

- 变换病态问题的形式，改善敏感性
- 选择稳定性好的算法，避免计算中误差的扩大
- 选择截断误差小的算法
- 控制舍入误差，遵循减小舍入误差的几条建议

若初始数据误差较小，问题不敏感，采用稳定的算法一定能得出比较准确的结果。

控制舍入误差的几条建议

- 尽量采用双精度浮点数(提高精度, 增大p)
- 避免中间计算结果出现上(下)溢出
- 避免“大数吃掉小数”(加、减法)
- 避免符号相同的两个相近数相减 (抵消现象)
- 简化计算步骤，减少运算次数

避免中间计算结果出现上(下)溢出

例： 计算$y = \frac{x_1}{x_2 \cdot x_3 \cdots\cdots x_n}$

其中 x_2 比 x_1 小很多，$|x_1/x_2| > 3.403 \times 10^{38}$ 。

采用单精度浮点数计算 x_1/x_2 会发生上溢。一般情况下, y 的准确结果不会超出上溢值。

为避免上溢, 应先计算 $z = x_2 \cdot x_3 \cdots \cdots x_n$, 然后计算 $y = x_1/z$ 。

在实际的计算中, 应对各个操作数的大小有大体的了解, 然后通过调整计算次序避免可能出现的上溢和下溢。

避免"大数吃掉小数"(加、减法)

例 (级数求和)：在浮点算术系统中计算 $\sum_{n=1}^{\infty} \frac{1}{n}$，分析会得到什么结果。

如果精确计算，这个无穷级数的和是发散的，但在浮点算术系统中不是这样。粗略分析，进行有限精度计算可能会：

(1)部分和非常大并发生上溢，即达到 OFL；

(2) $1/n$ 逐渐变小并产生下溢，因此部分和结果不变化。

实际上在达到上述两种情形之前，一旦增加量 $1/n$ 与部分和 $\sum_{k=1}^{n-1} \frac{1}{k}$ 的值相差悬殊，它们的和就停止变化，这时，

$$\frac{1}{n} \leqslant \frac{1}{2} \varepsilon_{\text{mach}} \sum_{k=1}^{n-1} \frac{1}{k}$$

避免符号相同的两相近数相减 (抵消现象)

例 一元二次方程 $ax^2 + bx + c = 0$的求根公式

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- 如果 $4ac$ 相对于 b^2 很小, $b > 0$，则 $-b + \sqrt{b^2 - 4ac}$ 会发生抵消现象, 采用:

$$x_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- 如果 $4ac$ 相对于 b^2 很小, $b < 0$, 则 $-b - \sqrt{b^2 - 4ac}$ 会发生抵消现象,采用:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{2c}{-b + \sqrt{b^2 - 4ac}}$$

例 当 $x < 0$, 且 $|x|$ 较大时,利用公式

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

截断前 n 项计算 e^x,会发生严重的抵消, 使数值结果误差很大。

- 设 $x = -20$,前96项的和$S_{96}(x) = 5.62188 \times 10^{-9}$, 下一步加的项为 $x^{96}/96! = 7.98930 \times 10^{-26}$, 它与 $S_{96}(x)$ 的比值已经小于$\frac{1}{2}\varepsilon_{\mathrm{mach}}$, 求和运算都不会改变部分和的计算值, 因此$e^{-20}$的计算值为 5.62188×10^{-9} (取 6 位有效数字), 而 e^{-20}的准确值为 2.06115×10^{-9},误差很大。

- 当 $x > 0$ 时,求和式中每项都大于 0 ,不会有抵消现象, 计算是稳定的。当 $x = 20$ 时, 计算前 68 项后结果将不再变化, 部分和为 $S_{68}(x) = 4.85165 \times 10^8$, 与准确值完全一样。

- 对于 $x < 0$ 的情况, 通过式 $1/e^{-x}$ 计算 e^x 是有效、可行的算法。

简化计算步骤,减少运算次数

例 (多项式函数求值的算法): 计算多项式的值.

$$P_n(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$$

- 若直接计算 a_ix^{n-i} 再逐项相加,一共需要 $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ 次乘法和 n 次加法。
- **泰九韶算法**(我国宋代数学家秦九韶于 1247 年提出, 在国外称为 **Hernor 算法**, 1819 年才被提出),只要 n 次乘法和 n 次加法.

输入： x, 多项式系数a_i, $i = 0,1,2,\cdots,n$

输出： $P_n(x)$

$b := a_0$;

For $i = 1,2,\ldots,n$

    $b := bx + a_i$;

End

$P_n(x) := b$;