高等数值算法与应用（一）

Advanced Numerical Algorithms & Applications

计算机科学与技术系 喻文健

Today

- About the Course
- Introduction of Scientific Computing and Numerical Algorithm
- Mathematical Software and Matlab
- *Application Example: ***PageRank***^{TM}
- Source of Errors
- Floating Point Arithmetic
- Assignment

课程概要

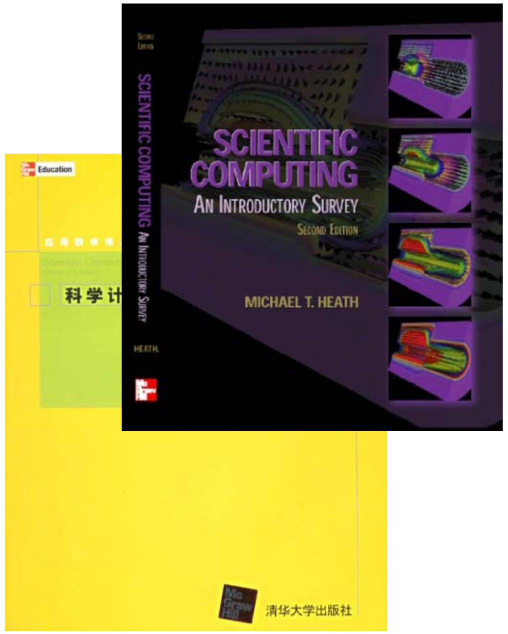
- 授课教师：喻文健
  - **E-mail:** **yu-wj@tsinghua.edu.cn**
  - [redacted]
- 网络学堂：提供课件、课程作业、阅读材料、**Project**信息等
- 答疑
  - 主要通过**E-mail**、网络学堂答疑

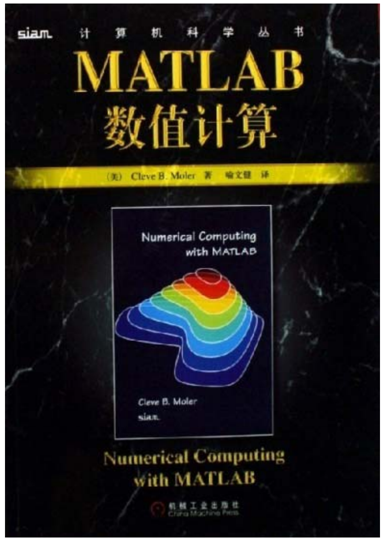
课程简介

- 数值算法、应用、高等（***scientific computing***）
- 课程目标
  - 本科数值分析课程的扩展
  - 重要数值算法及其实现, **Matlab**应用
  - 较新的理论知识和实践锻炼 $\longrightarrow$ 大规模稀疏矩阵 一些应用实例
  - 培养科研能力：在课题研究中应用、改进甚至创造有关数值算法的能力；文献、表达等

教材与参考书

- M. T. Heath，*Scientific Computing: An Introductory Survey*（第二版），清华大学出版社（影印版），2002年版。或：张威等译，科学计算导论，清华大学出版社，2005年版。
- 喻文健 译, "Matlab数值算法", 机械工业出版社，2006年
- 课件，以及其他补充资料
- 网络学堂：*课本勘误表；参考书的英文版电子书和Matlab程序包*




siam 计算机科学丛书
MATLAB
数值计算
Numerical Computing with MATLAB
Cleve B. Moler
Numerical Computing with MATLAB

授课方式与考核

- 授课方式
  - 以讲授为主，课堂报告和讨论
- 考评方法
  - 课程参与：**5%** 出勤、互动
  - 平时成绩：**60%** 作业通过网络学堂布置，下一周上课时交纸件
  - 期末**Project：40%** **(**多种形式**)**

主要教学内容

- 一.科学计算导论
  - 误差分析、计算机浮点算术精度体系
- 二.矩阵分解及其应用
  - **QR**分解、**SVD**分解以及应用；稀疏矩阵直接解法
- 三.线性方程组求解的迭代解法
  - **CG**算法、**GMRES**算法、预条件技术
- 四.常微分方程的数值解法
  - 常微分方程组、初值问题、刚性、边值问题
- 五.偏微分方程数值解法基础
- 六.快速算法及其应用
  - **FFT**，以及基于**FFT**的快速**PDE**解法

主要教学内容

- **七.课程报告(数值算法的应用)**
- **说明：**
  - 在教学和实践环节，使用**Matlab**
  - *课本"**Scientific Computing**"的交互式演示程序网站：* http://www.cse.illinois.edu/iem/
  - 本课程涉及第**1-4, 9-11**章内容。

Introduction of Scientific Computing and Numerical Algorithm

Wenjian Yu

数值分析、科学计算、数值计算

Numerical analysis = Scientific computing

数值计算作为当今科学研究的三种基本手段之一，是数学和计算机应用于其他学科的桥梁，因此它的发展受到广泛关注。有些发达国家甚至将科学计算作为衡量国家综合实力的一个重要方面，大力推动其发展。

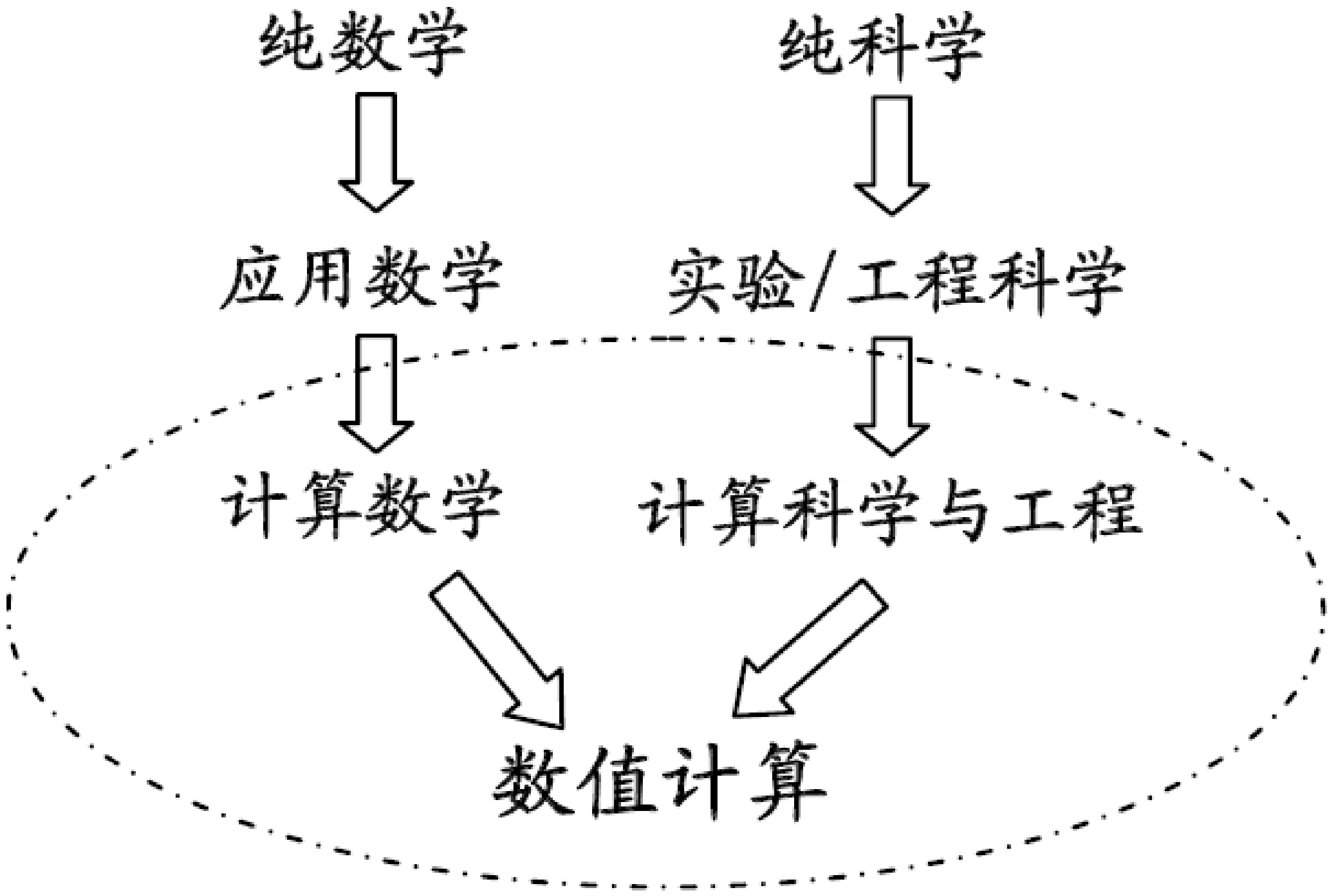
—— *英文影印版序言*

科学计算的含义也可包括软件和硬件两方面，这里的重点是软件方面，即数值计算的有关算法 **(数值仿真软件)**

“数值计算”主要是研究求解连续数学问题的算法的学科（而不仅仅局限于计算误差的研究）

对象　　核心

Evolution of scientific computing from other sciences and engineering disciplines

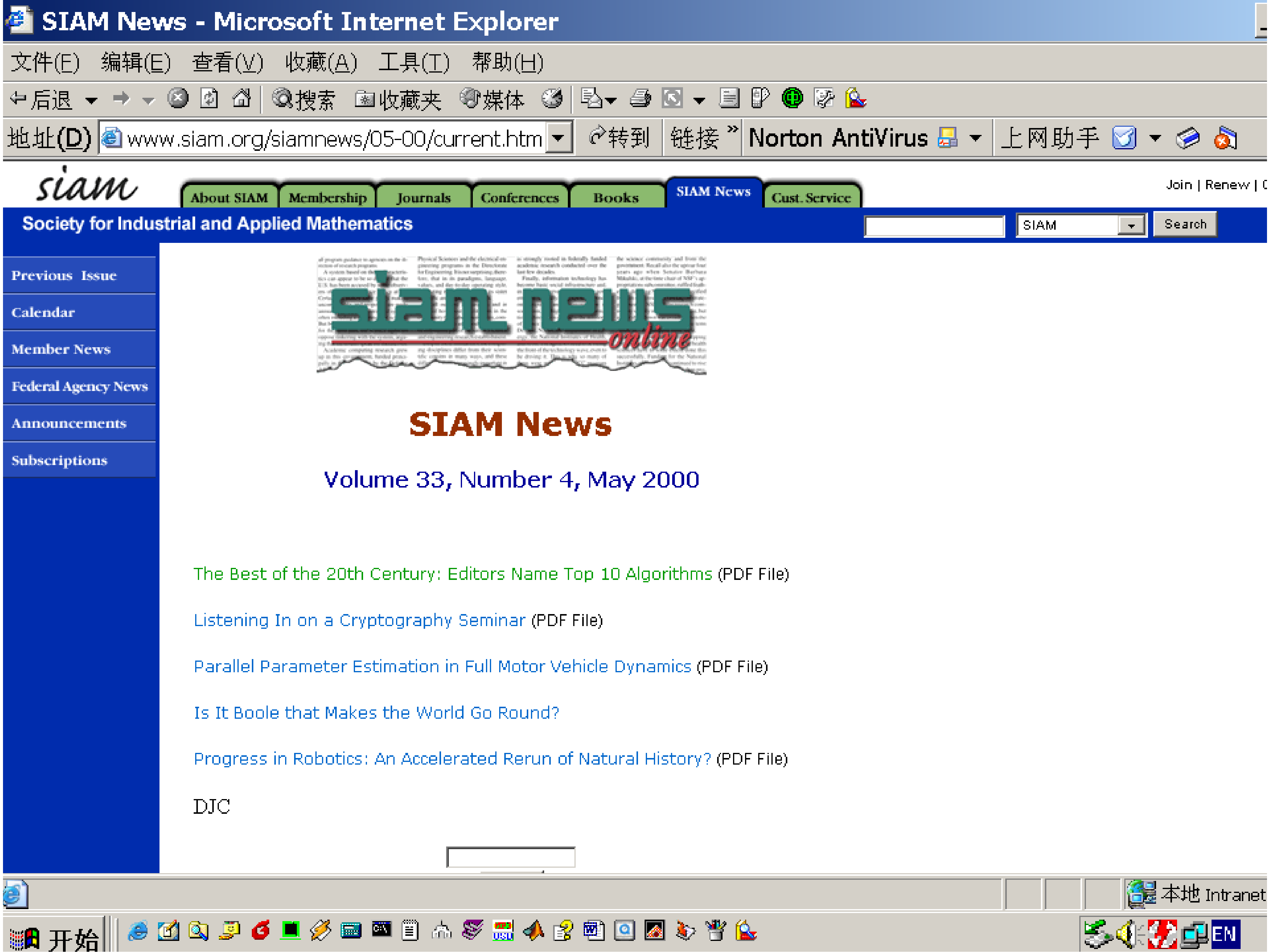


何谓“算法” - Algorithm

- **“Algorithm”**
  - 源于九世纪阿拉伯学者的名字**Al-Khwarizmi**
  - 其著作“**al-jabr wa’l muqabalah**”是现代高中代数**(algebra)**课本的雏形
  - 他强调解决问题时，系统的过程的重要性
  - *计算机程序＝数据结构＋算法*
  - **“Computing in Science & Engineering”(IEEE Computer Society**期刊**)**在**2000**年第**1**期公布了其评选的**20**世纪“十大算法”

见网络学堂“教学资源”

IEEE Computational Science and Engineering



SIAM News - Microsoft Internet Explorer

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

后退 搜索 收藏夹 媒体

地址(D) www.siam.org/siamnews/05-00/current.htm 转到 链接 Norton AntiVirus 上网助手

About SIAM | Membership | Journals | Conferences | Books | SIAM News | Cust. Service

Join | Renew |

Society for Industrial and Applied Mathematics

SIAM Search

Previous Issue
Calendar
Member News
Federal Agency News
Announcements
Subscriptions

SIAM News

Volume 33, Number 4, May 2000

The Best of the 20th Century: Editors Name Top 10 Algorithms (PDF File)

Listening In on a Cryptography Seminar (PDF File)

Parallel Parameter Estimation in Full Motor Vehicle Dynamics (PDF File)

Is It Boole that Makes the World Go Round?

Progress in Robotics: An Accelerated Rerun of Natural History? (PDF File)

DJC

本地 Intranet

开始 EN

Top ten algorithms of the century

"We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century"

—— *Editors*

- **1.1946 Los Alamos**科学实验室的***J. von Neumann, S. Ulam***和***N. Metropolis***编的**Metropolis**算法**,**即**Monte Carlo** 方法**(**"随机漫步"**)**
- **2.1947** 兰德**(RAND)**公司的***G. Dantzig***创造的线性规划的单纯型算法 **(simplex method)**
- **3.1950** 美国国家标准局数值分析所的***M. Hestenes, E. Stiefel***和***C. Lanczos***开创的**Krylov**子空间迭代法 **(*Lanczos*** *过程*, ***CG*** *算法*)
- **4.1950's** 橡树岭**(Oak Ridge)**国家实验室的***A. Householder*** 形式化的矩阵分解方法 **(***表示成矩阵分解起了革命性作用***)**

Top ten algorithms of the century

- **5.1957 IBM**由***J. Backus***领导的小组研制**Fortran**优化编译器
- **6.1959-61** 伦敦**Ferranti Ltd.**的***J.G.F. Francis***发明**QR**算法，*稳定地计算中、小规模矩阵的所有特征值*
- **7.1962** 伦敦**Elliot Brothers, Ltd.**的***Tony Hoare***提出快速排序算法**(Quicksort)**
- **8.1965 IBM Watson**研究中心的***J. Cooley***与**Princeton**大学及**AT&T Bell**实验室的***J. Tukey***共同提出了的**FFT**算法
- **9.1977 Brigham Young**大学的***H. Ferguson***和***R. Forcade***提出的整数关系侦察算法**(***应用于实验数学, 物理, 量子场理论***)**
- **10.1987 Yale**大学的***L. Greengard***和***V. Rokhlin***发明了快速多极算法**(*****fast multipole algorithm,*** *多体问题, 线性复杂度***)**

除了**No. 2, 5, 7**外，都属于或涉及数值计算的范畴**!**

数值算法与非数值算法

*The art of computer programming*系列

……

We might call the subject of these books "nonnumerical analysis." Computers have traditionally been associated with the solution of numerical problems such as …… Numerical computer programming is an extremely interesting and rapidly expanding field, and many books have been written about it.

From D. E. Knuth, ***The art of computer programming***, *Vol. 1* (《计算机程序设计艺术》)

- 算法分为"数值算法"和"非数值算法"
- 数值算法用途非常广泛，发展迅速，具有跨学科特点；处理连续数学的量，问题中经常涉及微分、积分或非线性。
- "非数值算法"的研究则通常归于计算机科学
- 数值算法：寻找快速收敛的(迭代)算法，评估准确度.

数值计算的问题与步骤

- 数值计算（科学计算、计算模拟、数值模拟）
  - 没有解析解的问题：Solve $33x^5+3x^4-17x^3+2x^2+4x-39=0$
  - 有解析解，但很难计算、或代价太大:
  - 模拟通常难以达到的实验条件 (时间、金钱成本)
    - 天体物理研究
    - 汽车碰撞实验，芯片投片前的性能仿真
- 数值模拟的步骤
  - 建立数学模型（需要相关学科背景）
  - 研究数值求解方程的算法
  - 通过计算机软件实现算法
  - 在计算机上运行软件进行数值模拟

课程重点

数值计算的问题

- 数值模拟的步骤（续）
  - 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（问题的背景和应用要求也左右着方法的选择）
- **Well-posed**问题（**well defined,** 适定）
  - 解存在、唯一、连续地依赖于问题数据
  - 问题数据的小改变不会导致解的突然变化
  - *在数值计算中该条件非常重要，因为数据扰动必然存在*
  - *对**well-posed**问题，解仍可能对数据扰动非常敏感。*

反例：地震的物理模型

Stable算法：算法本身不增加问题的敏感度

数值计算的策略

- 用简单问题近似复杂问题（具有相同或非常接近的解）
  - 无限维空间→有限维空间
  - 无限的过程→有限的过程　　　（数值积分）
  - 微分方程→代数方程
  - 非线性问题→线性问题
  - 高阶系统→低阶系统
  - 复杂函数→ 简单函数　　　（多项式函数）
  - 复杂形式的矩阵←简单形式的矩阵
- 求解过程：将问题简化　　（估计简化的误差）
  求解简化后的问题

Numerical Software / Package and Matlab

Wenjian Yu

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
  - 互联网，免费/商业代码
  - **Fortran, C, C++, Matlab**
  - 源代码使用，或**API**调用
  - 交互式集成环境的软件

名称	内容说明	商业/免费	网址
CMLIB	美国国家标准技术协会(NIST)的数学与统计软件虚拟仓库和检索系统	免费	gams.nist.gov
FMM	Computer Methods for Mathematical Computations 一书[2]所附软件	免费	www.netlib.org
HSL	英国 Science and Technology Facilities Council 提供的科学计算程序包	免费	www.hsl.rl.ac.uk
IMSL	Visual Numerics 公司提供的数学与统计程序软件库	商业	www.vni.com
NAG	NAG 公司提供的数值算法程序集	商业	www.nag.com
NAPACK	Applied Numerical Linear Algebra 一书[3]所附软件	免费	www.netlib.org
NETLIB	汇集各种免费数值计算软件的网站	免费	www.netlib.org
NR	Numerical Recipes 系列书[4]所附软件	部分免费	www.nr.com
NUMERALGO	期刊 Numerical Algorithms 中的软件	免费	www.netlib.org
MATLAB	MathWorks 公司出品的著名数学软件	商业	www.matlab.com
PORT	贝尔实验室开发的软件	部分免费	www.netlib.org
SLATEC	美国政府实验室联合汇编的软件	免费	www.netlib.org
SOL	美国斯坦福大学系统优化实验室开发的优化及相关问题的软件	免费	www.stanford.edu/group/SOL/
TOMS	期刊 ACM Transactions on Mathematical Software 中的算法程序，用名字和数字编号两种方式标识	免费	www.netlib.org

表 1-2 Matlab 与第三代编程语言的比较

	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(v 6.5 以后版本)	编译器
是否申明变量	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)	--

高质量数学软件追求的目标

- 可靠：对一般的问题总能正确运行；
- 准确：根据问题和输入数据产生精确的结果，并能对达到的准确度进行评估；
- 高效率：求解问题所用的时间和存储空间尽可能地小；
- 方便使用：具有方便、友好的用户界面；
- 可移植：在各种计算机平台下都（或经少量修改后）能使用；
- 可维护：程序易于理解和修改（开放源代码的软件）；
- 适用面广：可求解的问题广泛；
- 鲁棒：能解决大部分问题，偶尔失败的情况下能输出提示信息、及时地终止。

Some useful Matlab commands

- Start: `matlab`
- Constants: `pi, i, j`
- Arithmetic operators: `+, -, *, /, ^, .+, .-, .*, ./`
- Relational operators: `==, ~=, >, >=, <, <=`
- Logical operators: `and, or, not`
- Help: `help,` **`doc,`** `lookfor, demo`
- Outputs: `disp, fprintf, ;` (supress output), `format`
- Elementary junctions: `sin, cos, tan, sinh, asin, exp, log, log10, sqrt`
- Variables: `who, whos, clear, save, load, ans, diary`
- Vectors: `[…, …], rand, length`
- Matrices: `[…, … ; …, …], ones, eye, rand, size, diag, tril, triu`
- Graphs: `plot, subplot, loglog, ezplot, hold , plot3, figure, close`
- Files: `edit, type, ls, path`
- Programming: `function, if, for, while, end, inline, @`
- End: `quit`

Source of Errors

数值计算的误差

- 误差的来源
- 计算结果的误差
  - 绝对误差和相对误差
  - 数据误差的传递和计算误差
  - 截断误差和舍入误差
- 向后误差分析
- 问题的敏感性(病态性)
- 算法的稳定性

Source of approximation

- Which occur *before* a computation begins
  - Modeling: with simplified or omitted physical features (friction, viscosity, air resistance) 摩擦，黏滞性
  - Empirical measurements: laboratory measurements have finite precision
  - Previous computations: Input data may be the results of a previous computational step

Usually beyond our control, although they are important to determine the accuracy of computation

Source of approximation

- Which occur *during* computation
  - Truncation or discretization: omitting or simplifying some mathematical features (derivative to finite differen -ce, using a finite number of terms in an infinite series)
  - Rounding: representation of real numbers and arithmetic operations is ultimately limited to some finite amount of precision

Final accuracy is affected by <u>all approx</u>. Uncertainty in input may be amplified by the <u>nature of problem</u>; Perturbations during computation may be amplified by <u>algorithm</u>.

Surface area of the Earth:

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

- **The Earth is idealized as a sphere**
- **r ≈ 6370 km, based on empirical measurements**
- **Value of π must be truncated at some point**
- **The numerical values of input and the arithmetic operations are rounded**

Absolute Error & Relative Error

Absolute error = approximate value – true value

$$\text{Relative error} = \frac{\text{absolute error}}{\text{true value}}$$

Final error involving all approximations

- The absolute error is not its “absolute value”
- Note the relative error is undefined if the true value is 0
- A completely error approx. ~ a relative error of at least 100%
- E_r of 10^{-p} ~ p correct significant digits in 10-representation
- Precision: number of digits with which to be expressed
- Accuracy: number of correct significant digits
- We estimate or bound the error *rather than* compute it exactly, because the true value is unknown

Data Error and Computational Error

Typical problem: compute value of function $f: \mathbb{R} \to \mathbb{R}$ for given argument

x = true value of input, $f(x)$ = desired result

$\hat{x}$ = approximate (inexact) input

$\hat{f}$ = approximate function computed

Total error $= \hat{f}(\hat{x}) - f(x) =$

$$\underbrace{(\hat{f}(\hat{x}) - f(\hat{x}))}_{\text{computational error}} + \underbrace{(f(\hat{x}) - f(x))}_{\text{propagated data error}} =$$

computational error + propagated data error

数据误差的传递

- Algorithm has no effect on propagated data error

Truncation Error and Rounding Error

截断误差：方法误差、不考虑计算机有限精度的影响

Truncation error: difference between true result (for actual input) and result produced by given algorithm using exact arithmetic

Due to approximations such as truncating infinite series or terminating iterative sequence before convergence

舍入误差：计算机有限精度体系的影响

Rounding error: difference between result produced by given algorithm using exact arithmetic and result produced by same algorithm using limited precision arithmetic

Due to inexact representation of real numbers and arithmetic operations upon them

$\boxed{(\hat{f}(\hat{x}) - f(\hat{x}))}$

Computational error is sum of truncation error and rounding error, but one of these usually dominates

Example: Finite Difference Approx.

Error in finite difference approximation

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

exhibits tradeoff between rounding error and truncation error

Taylor's series: $f(x+h) = f(x) + f'(x)h + f''(\theta)h^2/2$

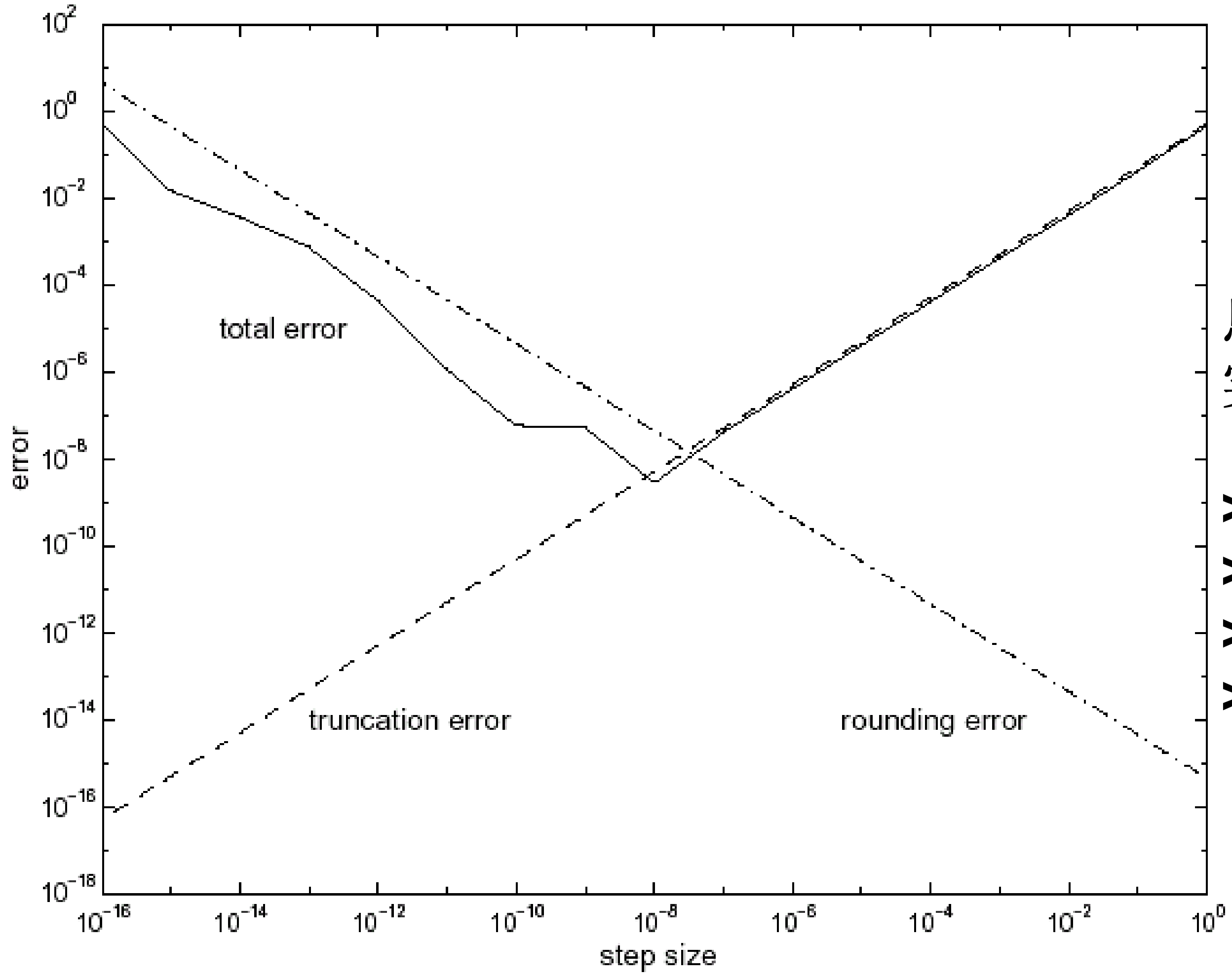
Truncation error bounded by $Mh/2$, where M bounds $|f''(t)|$ for t near x

Rounding error bounded by $2\epsilon/h$, where error in function values bounded by ϵ

$$\frac{Mh}{2} + \frac{2\epsilon}{h} \longleftarrow$$ Total error minimized when $h \approx 2\sqrt{\epsilon/M}$

Error increases for smaller h because of rounding error and increases for larger h because of truncation error

Example: Finite Difference Approx.



思考：怎么做**Matlab**
实验画出这个图**?**

```
>> h=logspace(-16, 0, 17);
>> nd=(sin(1+h)-sin(1))./h;
>> loglog(h, abs(nd-cos(1)))
>> axis([1e-16 1 1e-18 1e2])
```

Function: $f(x) = \sin(x), \text{ at } x = 1$ **M=1, $\epsilon \approx 10^{-16}$**

Forward and Backward Error

Suppose we want to compute $y = f(x)$, where $f\colon \mathbb{R} \to \mathbb{R}$, but obtain approximate value $\hat{y}$

Forward error $= \Delta y = \hat{y} - y$ 传统意义上讨论的误差

Backward error $= \Delta x = \hat{x} - x$, where $f(\hat{x}) = \hat{y}$ 折合到初始数据的误差

向后数据误差

例子： As approximation to $y = \sqrt{2}$, $\hat{y} = 1.4$ has absolute forward error

$$|\Delta y| = |\hat{y} - y| = |1.4 - 1.41421\ldots| \approx 0.0142,$$

or relative forward error about 1 percent

Since $\sqrt{1.96} = 1.4$, absolute backward error is

$$|\Delta x| = |\hat{x} - x| = |1.96 - 2| = 0.04,$$

or relative backward error 2 percent

向后误差分析

- 对向后误差的估计
- Idea
  - 将计算结果看成是一个“邻近”问题的准确解，或者完全由输入误差导致的结果
  - 那么，如果计算结果是非常邻近的问题的准确解，或对应于很小的向后数据误差，则计算结果就是好的（算法是稳定的）
- Why
  - 分析误差的向前传播往往很困难(计算步骤多, 误差限放得很大)
  - 向后误差分析相对容易，用于判断算法的稳定性

Example: Backward Error Analysis

To approximate cosine function $f(x) = \cos(x)$, truncating Taylor series after two terms gives

$$\hat{y} = \hat{f}(x) = 1 - x^2/2$$

Forward error:

$$\Delta y = \hat{y} - y = \hat{f}(x) - f(x) = 1 - x^2/2 - \cos(x)$$

To determine backward error, need value $\hat{x}$ such that $f(\hat{x}) = \hat{f}(x)$

For cosine function,

$$\hat{x} = \arccos(\hat{f}(x)) = \arccos(\hat{y})$$

Example, continuted

For $x = 1$,

$$y = f(1) = \cos(1) \approx 0.5403,$$

$$\hat{y} = \hat{f}(1) = 1 - 1^2/2 = 0.5,$$

$$\hat{x} = \arccos(\hat{y}) = \underline{\arccos(0.5)} \approx 1.0472$$

较容易计算

Forward error:

$$\Delta y = \hat{y} - y \approx 0.5 - 0.5403 = -0.0403,$$

Backward error:

$$\Delta x = \hat{x} - x \approx 1.0472 - 1 = 0.0472$$

Sensitivity and Conditioning

$\xleftarrow{\quad}$ 等价 $\xrightarrow{\quad}$

Problem *insensitive*, or *well-conditioned*, if relative change in input causes similar relative change in solution

$\xleftarrow{\quad}$ 等价 $\xrightarrow{\quad}$ $\longrightarrow$ 病态

Problem *sensitive*, or *ill-conditioned*, if relative change in solution can be much larger than that in input data

Condition number:

- $$\text{cond} = \frac{|\text{relative change in solution}|}{|\text{relative change in input data}|}$$

$$= \frac{|[f(\hat{x}) - f(x)]/f(x)|}{|(\hat{x} - x)/x|} = \frac{|\Delta y/y|}{|\Delta x/x|}$$

Problem sensitive, or ill-conditioned, if

$$\text{cond} \gg 1$$ 条件数远大于**1**

Condition Number

Condition number is "amplification factor" 放大因子

$$\left| \begin{matrix}\text{数据传递的}\\ \text{相对误差}\end{matrix} \right| = \text{cond} \times \left| \begin{matrix}\text{输入数据的}\\ \text{相对误差}\end{matrix} \right|$$

Condition number usually not known exactly and may vary with input, so <u>rough estimate or upper bound</u> used for cond, yielding 即:

$$\left| \begin{matrix}\text{数据传递的}\\ \text{相对误差}\end{matrix} \right| \lessapprox \text{cond} \times \left| \begin{matrix}\text{输入数据的}\\ \text{相对误差}\end{matrix} \right|$$

Example: Evaluating Function

函数求值问题

Evaluating function f for approximate input $\hat{x} = x + \Delta x$ instead of true input x gives

$$\text{数据传递的误差} = f(x+\Delta x) - f(x) \approx f'(x)\Delta x,$$

$$\text{数据传递的相对误差} = \frac{f(x+\Delta x) - f(x)}{f(x)} \approx \frac{f'(x)\Delta x}{f(x)},$$

$$\text{cond} \approx \left| \frac{f'(x)\Delta x / f(x)}{\Delta x / x} \right| = \left| \frac{x f'(x)}{f(x)} \right|$$

函数求值问题
的条件数估计式

Relative error in function value can be much larger or smaller than that in input, depending on particular f and x

依赖于具体的函数与**x**的取值

Example: Sensitivity　　敏感的问题

Tangent function for arguments near $\pi/2$: **1.570796**

$$\tan(1.57079) \approx 1.58058 \times 10^5$$

$$\tan(1.57078) \approx 6.12490 \times 10^4$$

Relative change in output quarter million times greater than relative change in input

For $x = 1.57079$, cond $\approx 2.48275 \times 10^5$

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

准确条件数：$\dfrac{\mathbf{9.6809\times10^4/1.58058\times10^5}}{\mathbf{10^{-5}/1.57079}} \approx \mathbf{1x10^5}$

算法的稳定性

Stability of algorithm analogous to conditioning of problem 可类比于

解同一个问题有多种算法

Algorithm *stable* if result relatively insensitive to perturbations *during* computation

计算数据误差、舍入误差，等等

From point of view of backward error analysis, algorithm stable if result produced is exact solution to nearby problem

向后误差分析

For stable algorithm, effect of computational error no worse than effect of small data error in input

求解同一问题的不同算法也可看成是的等价的不同问题。可能原始问题不病态，但某算法代表的数学问题（计算过程）病态，则该算法也是不稳定的。

Accuracy

Accuracy refers to closeness of computed solution to true solution of problem

准确度：对结果的最终评价

Stability alone does not guarantee accuracy

- Accuracy depends on conditioning of problem as well as stability of algorithm

Inaccuracy can result from applying stable algorithm to ill-conditioned problem or unstable algorithm to well-conditioned problem

Applying stable algorithm to well-conditioned problem yields accurate solution

小结

- 误差来源：计算之前的、计算过程中的
- Absolute Error & Relative Error
- Data Error & Computational Error
- Truncation Error & Rounding Error
- Forward Error & Backward Error
- Sensitivity (Conditioning) vs. Stability

思考典型的实际应用：数值计算过程包括多个步骤，怎么最大化最终结果的准确性？

	总误差		
	计算误差		输入数据传递误差
	截断误差	舍入误差	
如何评估大小?	对不同类型问题进行理论分析	向后误差分析；很难定量	问题病态性的概念，条件数
如何减小误差?	算法选择	选稳定的算法；避免误差危害原则；双精度计算	变换问题形式，改善病态性

Floating Point Arithmetic

Wenjian Yu

- 计算机的有限精度算术体系
  - 浮点数
  - 正规化
  - 浮点系统的特点
  - 舍入规则
  - 机器精度
  - 此正规化与逐渐下溢
  - 例外值
  - 浮点运算
  - 抵消现象(Cancellation)

Floating-Point Numbers

Floating-point number system characterized by four integers:

β	base or radix	基数
p	precision	
$[L, U]$	exponent range	

Number x represented as

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

科学计数法

where

$0 \le d_i \le \beta - 1, \; i = 0, \ldots, p-1, \;$ and $L \le E \le U$

$d_0 d_1 \cdots d_{p-1}$ called *mantissa* 尾数

E called *exponent*

$d_1 d_2 \cdots d_{p-1}$ called *fraction* 分数、小数

Typical Floating-Point Systems

Most computers use binary ($\beta = 2$) arithmetic

Parameters for typical floating-point systems shown below

	system	β	p	L	U	
单精度	IEEE SP	2	24	−126	127	**32**位字长
双精度	IEEE DP	2	53	−1022	1023	**64**位字长
	Cray	2	48	−16383	16384	
	HP calculator	10	12	−499	499	
	IBM mainframe	16	6	−64	63	

IEEE standard floating-point systems almost universally adopted for personal computers and workstations

Normalization 正规化

Floating-point system *normalized* if leading digit d_0 always nonzero unless number represented is zero

In normalized system, mantissa m of nonzero floating-point number always satisfies 尾数$d_0d_1 \ldots d_{p\text{-}1}$

$$1 \le m < \beta$$

基数

Reasons for normalization:

- representation of each number unique
- no digits wasted on leading zeros
- leading bit need not be stored (in binary system)

总是1

Properties of Floating-Point Systems

Floating-point number system finite and discrete

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

Number of normalized floating-point numbers:

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

x=0

< 2总字长

Smallest positive normalized number:

下溢值 underflow level = UFL = β^L

Largest floating-point number:

上溢值 overflow level = OFL = $\beta^{U+1}(1-\beta^{-p})$

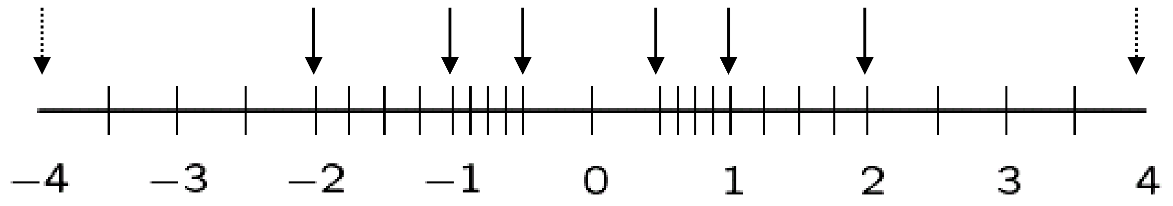
Floating-point numbers equally spaced only between powers of β 不均匀分布，仅在β^E与β^{E+1}之间均匀分布

Not all real numbers exactly representable; those that are are called *machine numbers* 能精确表示实数的称为机器数

Example: Floating-Point System

Tick marks indicate all 25 numbers in floating-point system having $\beta = 2$, $p = 3$, $L = -1$, and $U = 1$

3位尾数，
指数从-1到1



$$\text{OFL} = (1.11)_2 \times 2^1 = (3.5)_{10}$$

$$\text{UFL} = (1.00)_2 \times 2^{-1} = (0.5)_{10}$$

放大率

At sufficiently high magnification, all normalized floating-point systems look grainy and unequally spaced like this

粒状

仅在β^E与β^{E+1}之间均匀分布

Rounding Rules

舍入规则

If real number x not exactly representable, then approximated by "nearby" floating-point number fl(x)

例：3位尾数的系统

fl(1.75)=$(1.11)_2$

fl(1.5)=$(1.10)_2$

fl(1.625)=$(?)_2$

1.10

Process called *rounding*, and error introduced called *rounding error*

Two commonly used rounding rules:

"砍" • *chop*: truncate base-β expansion of x after $(p-1)$st digit; also called *round toward zero* 截断舍入（类似"下取整"）

• *round to nearest*: fl(x) nearest floating-point number to x, using floating-point number whose last stored digit is even in case of tie; also called *round to even* 最近舍入（类似"四舍五入"）

居中，不分胜负

Round to nearest most accurate, and is default rounding rule in IEEE systems

（另有误差源是输入输出中10/2进制转换）

Machine Precision

Accuracy of floating-point system characterized by *unit roundoff*, *machine precision*, or *machine epsilon*, denoted by ϵ_{mach}

E=0时，舍入带来的误差上限

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

With rounding by chopping, $\epsilon_{\text{mach}} = \beta^{1-p}$

With rounding to nearest, $\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-p}$

Alternative definition is smallest number ϵ such that $\text{fl}(1+\epsilon) > 1$

fl: 取最近的浮点数

Maximum *relative error* in representing real number x in floating-point system given by

$$\left|\frac{\text{fl}(x) - x}{x}\right| \le \epsilon_{\text{mach}}$$

一个浮点数系统相对间隔尺度（粒度）的度量

Machine Precision, continued

For toy system illustrated earlier, $\boxed{\beta = 2,\ p = 3,\ L = -1, \text{ and } U = 1}$

$\epsilon_{\text{mach}} = 0.25$ with rounding by chopping

$\epsilon_{\text{mach}} = 0.125$ with rounding to nearest β^{-p}

For IEEE floating-point systems,

$\epsilon_{\text{mach}} = 2^{-24} \approx 10^{-7}$ in single precision 表示实数的相对误差限

$\epsilon_{\text{mach}} = 2^{-53} \approx \boxed{10^{-16}}$ in double precision

演示程序：
floatshow.m

IEEE single and double precision systems have about 7 and 16 decimal digits of precision

Though both are "small," unit roundoff error ϵ_{mach} should not be confused with underflow level UFL 最小的正数

In all practical floating-point systems,

$$\underbrace{0 < \text{UFL}}_{\text{与最小指数有关}} < \overset{\text{与尾数长度有关}}{\epsilon_{\text{mach}}} < \text{OFL}$$

overflow, 最大的数

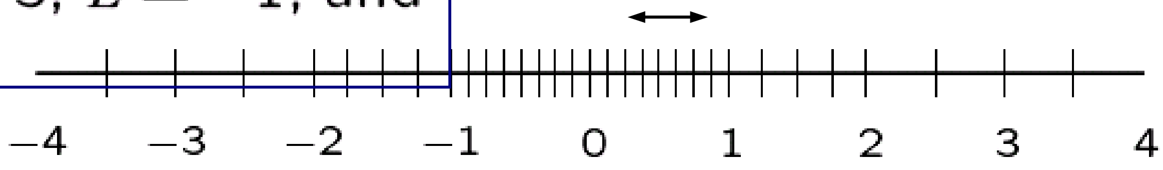
Subnormals and Gradual Underflow

Normalization causes gap around zero in floating-point system　　$0 \sim \beta^L$，UFL

If leading digits allowed to be zero, but only when exponent at its minimum value, then gap "filled in" by additional *subnormal* or *denormalized* floating-point numbers　次正规化浮点数

在IEEE标准中，实际由一个特殊的指数域值完成(由于不存储首位)

$\beta = 2$, $p = 3$, $L = -1$, and $U = 1$



Subnormals extend range of magnitudes representable, but have less precision than normalized numbers, and unit roundoff is no smaller ϵ_{mach}

较少的有效数字

增大的 Augmented system exhibits *gradual underflow*
逐渐下溢

思考：能表示的最小正数变为什么？

Exceptional Values

例外值

IEEE floating-point standard provides special values to indicate two exceptional situations:

- `Inf`, which stands for "infinity," results from dividing a finite number by zero, such as $1/0$
- `NaN`, which stands for "not a number," results from undefined or indeterminate operations such as $0/0$, $0 * \texttt{Inf}$, or $\texttt{Inf}/\texttt{Inf}$

`Inf` and `NaN` implemented in IEEE arithmetic through special reserved values of exponent field

用户是否可见，还依赖于操作系统、编程语言和编译器。
如果用户可见，则有利于调试程序和软件

Floating-Point Arithmetic

前面讲数的表示的误差

现在考虑浮点运算造成的误差

Addition or subtraction: Shifting of mantissa to make exponents match may cause loss of some digits of smaller number, possibly all of them

Multiplication: Product of two p-digit mantissas contains up to $2p$ digits, so result may not be representable

商

Division: Quotient of two p-digit mantissas may contain more than p digits, such as nonterminating binary expansion of $1/10$

有误差！

Result of floating-point arithmetic operation may differ from result of corresponding real arithmetic operation on same operands

Example: Floating-Point Arithmetic

Assume $\beta = 10$, $p = 6$

Let $x = 1.92403 \times 10^2$, $y = 6.35782 \times 10^{-1}$

Floating-point addition gives

$$x + y = 1.93039 \times 10^2,$$

assuming rounding to nearest

Last two digits of y do not affect result, and with even smaller exponent, y could have had no effect on result

Floating-point multiplication gives

$$x * y = 1.22326 \times 10^2,$$

which discards half of digits of true product

Floating-Point Arithmetic, continued

Real result may also fail to be representable because its <u>exponent is beyond available range</u>

<u>Overflow usually more serious than underflow</u> because there is *no* good approximation to arbitrarily large magnitudes in floating-point system, whereas <u>zero is often reasonable approximation</u> for arbitrarily small magnitudes

On many computer systems overflow is <u>fatal</u>, but an underflow may be silently set to zero

Example: Summing a Series

Infinite series

$$\sum_{n=1}^{\infty} \frac{1}{n}$$

has finite sum in floating-point arithmetic even though real series is divergent

发散的

Possible explanations:

- Partial sum eventually overflows
- $1/n$ eventually underflows
- Partial sum <u>ceases to change</u> once $1/n$ becomes negligible relative to partial sum:

$$1/n < \epsilon_{\text{mach}} \sum_{k=1}^{n-1} (1/k)$$

此情形比前两者都先发生！

应防止相差悬殊的两数加减

ε是小于$\varepsilon_{\text{mach}}$的数

$\text{fl}(1+\varepsilon)$= 1

$\text{fl}(a+\varepsilon\cdot a)=a$

Floating-Point Arithmetic, continued

Ideally, x flop y = fl(x op y), i.e., floating-point arithmetic operations produce correctly rounded results

意味着结果的相对误差 ~ $\varepsilon_{\text{mach}}$

Computers satisfying IEEE floating-point standard achieve this ideal as long as x op y is within range of floating-point system

注意：

1.这只是一步运算
2.一些熟悉的规则在浮点系统中无效

But some familiar laws of real arithmetic not necessarily valid in floating-point system

Floating-point addition and multiplication commutative but *not* associative 可结合
可交换

Example: if ϵ is positive floating-point number slightly smaller than ϵ_{mach},

$$(1+\epsilon)+\epsilon = 1, \text{ but } 1+(\epsilon+\epsilon) > 1$$

因此，要按运算顺序依次分析每一步计算的误差，这导致对实际问题的舍入误差分析非常困难。

由于$\varepsilon_{\text{mach}}$是很小的数，若采用较稳定的数值算法，最终结果的误差不会很大。

Cancellation

“抵消”

Subtraction between two p-digit numbers having same sign and similar magnitudes yields result with *fewer* than p digits, so it is usually exactly representable

Reason is that leading digits of two numbers *cancel* (i.e., their difference is zero)

Example:

$$1.92403\times10^2 - 1.92275\times10^2 = 1.28000\times10^{-1},$$

which is correct, and exactly representable, but has only three significant digits

仅三位有效数字

Cancellation, continued

Despite exactness of result, cancellation often implies serious loss of information

操作数

Operands often uncertain due to rounding or other previous errors, so relative uncertainty in difference may be large 两数之差的相对误差变大！

Example: if ϵ is positive floating-point number slightly smaller than ϵ_{mach},

舍入误差

$$(1+\epsilon)-(1-\epsilon) = 1-1 = 0$$

in floating-point arithmetic, which is correct for actual operands of final subtraction, but true result of overall computation, 2ϵ, has been completely lost

减法自身没有问题

相近数相减问题的条件数很大、很病态

作业题**1.5**

Cancellation, continued

Digits lost to cancellation are most significant, leading digits, whereas digits lost in rounding are least significant, trailing digits

Because of this effect, it is generally bad idea to compute any small quantity as difference of large quantities, since rounding error is likely to dominate result

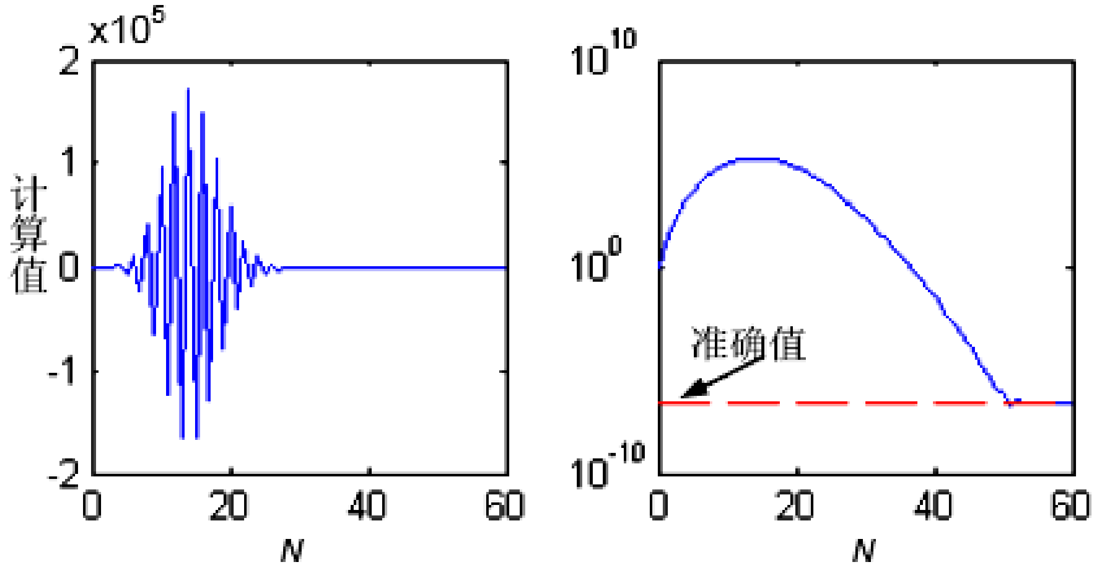
For example, summing alternating series, such as

正负交替序列

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

for $x < 0$, may give disastrous results due to catastrophic cancellation

由于 **“cancellation”**带来严重后果：
计算效率很低



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

图 1.5 $x = -15$时部分和的计算值随项数 N 的变化曲线，右侧图中纵轴表示计算值的绝对值，并采用对数坐标。

只有取**N=53**项或更多项进行求和，才能使结果误差小于**5%**

如果**x=15**，则不会发生抵消现象，项数**N=23**时结果的误差就小于**5%**，主要误差源是截断误差

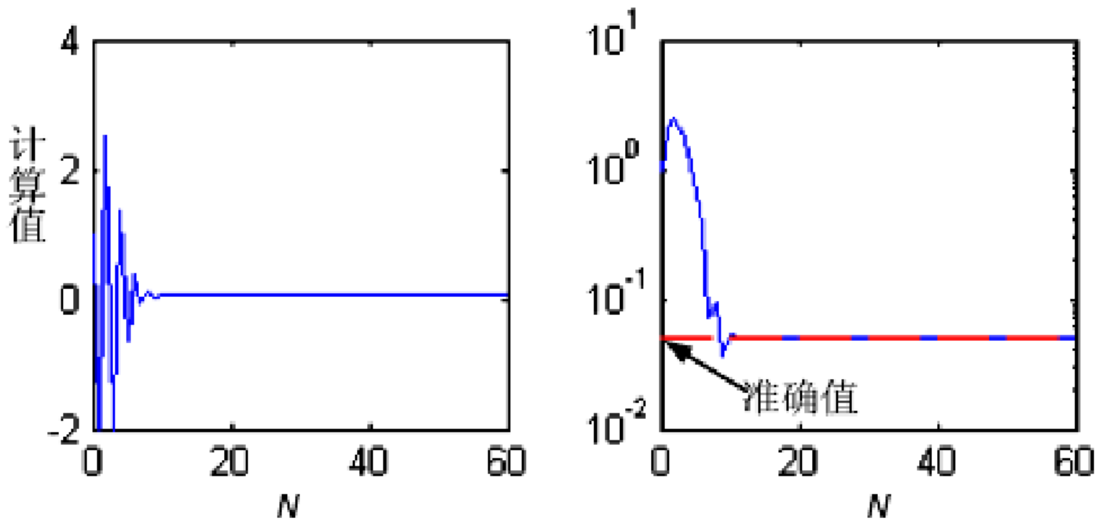


图 1.7 $x = -3$时部分和的计算值随项数 N 的变化曲线，

取项数**N=12**，结果与准确值的误差就已降至**1.8%**

如果**x=3**，则不会发生抵消现象，项数**N=8**时误差就小于**1.2%**

Example: Quadratic Formula

Two solutions of quadratic equation

$$ax^2 + bx + c = 0$$

given by

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

Naive use of formula can suffer overflow, or underflow, or severe cancellation

当系数**a, b, c**都很大，或都很小时，计算**b²**和**4ac**

Rescaling coefficients can help avoid overflow and harmful underflow

Cancellation between $-b$ and square root can be avoided by computing one root using alternative formula

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

当系数**a, c**相对于**b**很小时

Cancellation inside square root cannot be easily avoided without using higher precision

Example: Standard Deviation

标准差

Mean of sequence x_i, $i = 1, \ldots, n$, is given by

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i,$$

and standard deviation by

$$\sigma = \left[\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right]^{\frac{1}{2}}$$

Mathematically equivalent formula

$$\sigma = \left[\frac{1}{n-1} \left(\sum_{i=1}^{n} x_i^2 - n\bar{x}^2 \right) \right]^{\frac{1}{2}}$$

仅需遍历一次

avoids making two passes through data

Unfortunately, single cancellation error at end of one-pass formula is more damaging numerically than all of cancellation errors in two-pass formula combined

因为相减的两数都非常大，则被抵消的位数也多

- 小结
  - 基本概念：**OFL**、**UFL**、舍入规则、机器精度$\varepsilon_{\mathrm{mach}}$；浮点运算的舍入误差、“抵消”
  - 避免误差危害的建议
    - 避免相差悬殊的数做加减法（“大数吃掉小数”），相差的倍数达到$\varepsilon_{\mathrm{mach}}$
    - 避免符号相同的两相近数相减（cancellation）
    - 注意计算步骤安排，使每步都是良态的问题
    - 简化步骤，减少运算次数

Assignment

- 阅读课本第一章有关内容
- 浏览网站: http://www.cse.illinois.edu/iem/
- 练习题：**1.5, 1.11, 1.19, 1.20**
- 补充题：**floatshow.m**程序
- （详见网络学堂上的要求）