高等院校计算机系列教材

数值分析与应用程序

主　编　全惠云
副主编　邹秀芬　康立山
　　谢资清　何迎生

WUHAN UNIVERSITY PRESS
武汉大学出版社

高等院校计算机系列教材

数值分析与应用程序

主　编　全惠云

副主编　邹秀芬　康立山

谢资清　何迎生

WUHAN UNIVERSITY PRESS

武汉大学出版社

图书在版编目(CIP)数据

数值分析与应用程序/全惠云主编．--武汉：武汉大学出版社，2007.4
高等院校计算机系列教材
ISBN 978-7-307-05430-1

Ⅰ.数… Ⅱ.全… Ⅲ.数值计算—计算方法—程序设计—高等学校—教材 Ⅳ.O241

中国版本图书馆 CIP 数据核字(2007)第 023114 号

责任编辑：黄金文 史新奎 张敏 责任校对：程小宜 版式设计：支笛

出版发行：**武汉大学出版社** (430072 武昌 珞珈山)
(电子邮件：wdp4@whu.edu.cn 网址：www.wdp.com.cn)
印刷：武汉中科兴业印务有限公司
开本：787×1092 1/16 印张：11.25 字数：263 千字
版次：2007 年 4 月第 1 版 2007 年 4 月第 1 次印刷
ISBN 978-7-307-05430-1/O・353 定价：21.00 元(含配套光盘)

高等院校计算机系列教材

编　委　会

内容提要

本书介绍了科学计算中常用的计算方法，其内容包括误差的概念，插值方法，线性代数方程组的解法，非线性方程的求根，数值积分与数值微分，最小二乘法，特征值的计算，常微分方程初值问题的数值解法等. 该书重点突出，深入浅出，便于教学. 每种算法都附有 C 语言和 Matlab 语言程序（放入附在本书封底的光盘里），便于读者上机实习，也便于实际工作者查阅和上机使用。

前 言

把计算方法与相应的计算程序结合在一起编写的书籍虽然早有所见，但在计算机日益普及的今天，学习计算方法与学习相应的计算程序，两者缺一不可，这已成为人们的共识.

因此，编写一本把计算方法与相应的计算程序相结合的教材，实在是势在必行. 就我们的学识而言，写这本书是勉为其难，幸好有 S. Nakamura 和 David Kahaner 的著作及国内类似的书籍可作参考.

通过教学实践和调查，我们认为，非计算数学专业的学生只需要学习计算方法的一些基本内容，而不必在计算方法的理论上花费过多的时间. 此外，还应当增加上机算题的内容. 基于以上考虑，全书在内容安排上参考了目前许多大学讲授计算方法的实际情况，删去了偏微分方程与常微分方程边值问题数值解法的内容. 本书的内容包括：误差的概念，插值方法，线性代数方程组的解法，非线性方程的求根，数据拟合方法，数值微分和数值积分方法，特征值数值方法，常微分方程初值问题解法等最基本的内容.

本书的一个突出特点是在写法上偏重计算格式的构造和编程，以及使用时应注意的问题及技巧，并配有大量的计算例题. 例题的数值结果大部分由书上的程序计算所得，便于验证，一改以往计算方法教材论证多、例题少的情况. 另一个突出特点是每一个计算格式均配有用 C 语言和 Matlab 语言编制的程序，所有的程序都是完整独立的，每一个程序都有详细的使用说明，这些程序适用于各种规模的计算题，全部程序储存在一张安装在书封底内页上的光盘内. 这种书与光盘二合一的新形式的教材，便于学生和其他科研人员进行数值计算，有利于克服以往计算方法教学只讲理论不上机实习的弊端.

除此之外，我们还编撰了几位在计算数学领域做出突出贡献的著名科学家的生平事迹，分别编排在有关章节的末尾，也许对于读者了解计算数学的发展史有所裨益.

学习本书所需的数学基础是微积分、线性代数和常微分方程的基本概念，同时，应该熟悉 C 语言或 Matlab 语言之一. 全书需要 60 左右学时的讲授，20 左右机时的上机.

参加本书编写的有全惠云、邹秀芬、康立山、谢资清、何迎生等.

作者诚挚地感谢武汉大学出版社为本书的出版作出的努力，感谢硕士研究生李灿等为本书的电子文稿付出的辛勤劳动。

本书既可作为高等学校信息与计算科学专业的教材或参考书，也可作为计算机及相关专业的教材，也可供从事数值分析的有关人员参考.

限于水平，不当之处，承蒙指正，不胜感激.

作者

2006. 12.

目 录

第1章 科学计算中的误差

1.1 引　　言

在用计算机解决科学计算问题时，误差的来源主要有两种：截断误差和舍入误差. 在研究这两种误差之前，有必要弄清楚有关误差的一些基本概念.

首先的问题是误差如何定义. 若用 x 来表示准确值 x^* 的一个近似，则此近似值 x 和准确值 x^* 的差称为误差(也称绝对误差). 用 e^* 来表示：

$$e^* = x - x^*. \tag{1.1.1}$$

由于在一般情况下准确值 x^* 是不知道的，所以误差 e^* 的准确值也不可能求出. 但根据实际情况，往往可以估计出误差绝对值的上界 ε^*，称 ε^* 为误差限：

$$|e^*| = |x - x^*| \leqslant \varepsilon^*. \tag{1.1.2}$$

例 1.1　$x^* = \pi = 3.14159265\cdots$ 按四舍五入的原则如下：

取一位：$x_1 = 3$，　$e_3^* \approx -0.14$；

取三位：$x_3 = 3.14$，　$e_3^* \approx -0.0016$；

取五位：$x_5 = 3.1416$，$e_5^* \approx 0.000007$.

上面的例子中的 $x_3 = 3.14$ 是以三位有效数字来表示 π，它的误差限为：

$$|x_3 - \pi| \leqslant \frac{1}{2} \times 10^{-2}.$$

$x_5 = 3.1415$ 是以五位有效数字来表示 π，它的误差限为

$$|x_5 - \pi| \leqslant \frac{1}{2} \times 10^{-4}.$$

但误差的概念，不能说明近似的程度. 我们经常用的另一个尺度是相对误差. 定义近似值的误差与准确值的比值为相对误差，并记作 e_r^*：

$$e_r^* = \frac{e^*}{x^*} = \frac{x - x^*}{x^*}. \tag{1.1.3}$$

由于在实际中 x^* 往往是不知道的，所以通常取

$$e_r^* = \frac{e^*}{x^*} = \frac{x - x^*}{x} \tag{1.1.4}$$

为近似值 x 的相对误差. 并把相对误差绝对值的上界称为相对误差，记作：

$$e_r^* = \frac{\varepsilon^*}{|x|}, \tag{1.1.5}$$

其中：ε^* 是 x 的误差限.

由于相对误差不依赖于数的量纲，因此它是衡量准确度的一个更有效的工具. 例如在测量质量时用的单位有千克和克两种，两者的相对误差是相同的. 但前者的绝对误差却是后者的

1000 倍.

1.2 截断误差

数值解大部分是精确解的近似. 大部分数值方法基于有多项式表示的近似数,即使是更优良的算法,也是由基本的算法组合而成的. 因此,研究一个数值方法的误差,就必须考察一下多项式是怎样准确地逼近原函数的. 一个给定点的泰勒展式,在基本范围内准确地代表一个函数. 因此,通过数值解的多项式展示和准确解的泰勒级数比较,尤其是通过寻找在哪一个阶段产生差异,就可计算出误差,称之为截断误差,记为,$R_n(x)$.

例 1.2 函数 $f(x)$ 在 $x=0$ 点的泰勒展式为

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \cdots + \frac{f^{(n)}(0)}{n!}x^n + \cdots.$$

若取上式右端的前 $n+1$ 项

$$P_n(x) = f(0) + \frac{f'(0)}{1!}x + \cdots + \frac{f^{(n)}(0)}{n!}x^n$$

作为 $f(x)$ 的近似值,则此数值方法的截断误差是

$$R_n = f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}x^{(n+1)}.$$

上两式中 $f^{(n)}(c)$ 表示 $f(x)$ 的 n 阶导数在 c 点的值.

1.3 计算机中的数与舍入误差

由于计算机数位的限制,在用计算机进行数值计算时只能取有限位数字进行运算,由此而引起的误差称为舍入误差. 例如,$\pi = 3.14159\cdots$,$\sqrt{2} = 1.414214\cdots$,$\frac{1}{3} = 0.3333\cdots$,等等. 在计算机上运算时只能运用有限位小数,如果我们只保留小数点后四位数字,则

$$\rho_1 = 3.1416 - \pi = 0.0000074\cdots,$$

$$\rho_2 = 1.4142 - \sqrt{2} = -0.000013\cdots,$$

$$\rho_3 = 0.3333 - \frac{1}{3} = -0.000033\cdots$$

就是舍入误差. 在某些情形下舍入误差会产生很严重的影响,使得结果完全失真. 因此,了解计算机中的算术运算和明白在什么情况下会产生严重的舍入误差是至关重要的.

另外,数值常数的范围和精度随着不同的编程语言和不同的计算机而变化,自然,舍入误差的情况也会千差万别.

1.3.1 数的基

我们在日常生活中所用的数字系统称为十进制系统,十进制数系统的基是 10. 然而大部分的计算机在计算和内存中用的是二进制系统而不是十进制(有些计算机用的是八进制和十六进制系统,八进制、十六进制与二进制有密切的关系). 二进制系统的基是 2,八进制和十六进制系统的基分别是 8 和 16. 例如 $(3.224)_{10}$ 表示基为 10 的十进制数 3.224,$(1.0011)_2$ 表示

基为 2 的二进制数 1.0011.

一个基为 r 的数例如 $(abcdefg.hijk)_r$ 的十进制值可计算如下：

$$ar^6 + br^5 + cr^4 + dr^3 + er^2 + fr + g + hr^{-1} + ir^{-2} + jr^{-3} + kr^{-4}.$$

在本书中，若无特别声明，则省略下标的数都是十进制数.

1.3.2　数值常数的范围

计算机中的数主要分成两类：(*a*) 整数；(*b*) 实数(浮点数).

在 C 语言中，根据整数的范围和符号，整数定义为四种类型：无符号型、整型、长型和短型. 表 1.1A 列出几种不同计算机上有符号整数的范围. 也就是说，超出了此范围的数在计算机上是无法表示的.

表 1.1A　　C 语言中整数的范围

	短　型	整　型	长　型
Quick C (IBM PC)	N/A	$-2^{15} \sim 2^{15}-1$	$-2^{31} \sim 2^{31}-1$
IRIS(SG)	$-2^{15} \sim 2^{15}-1$	$-2^{31} \sim 2^{31}-1$	$-2^{31} \sim 2^{31}-1$
Cray	$-2^{31} \sim 2^{31}-1$	$-2^{63} \sim 2^{63}-1$	$-2^{63} \sim 2^{63}-1$

一个实数(十进制浮点数)一般用 $a \times 10^b$ 表示，a 叫做尾数，b 叫做指数. 通常 b 是一个整数，a 是小数，a 和 b 都只有有限位，相应的实数也只有有限位，因而存在最大和最小的实数.

在计算机上表示的一个实数的范围依赖于计算机硬件和软件的设计，在 IBM PC 机(Basic)上实数的范围从 2.9×10^{-39} 到 1.7×10^{38}，其他计算机上的数据参看表 1.1B.

我们必须认识到计算机上的实数是不连续的. 如果在零附近查找数，最小的正数是 2.9×10^{-39}(在 IBM PC(Basic) 机). 因此在 0 和 2.9×10^{-39} 以及 0 和 -2.9×10^{-39} 之间没有数能表示出来. 在 IBM PC 机上，1 和大于 1 的最小数之差大约为 1.19×10^{-7}，这个值称为机器小数，关于机器小数详见 1.3.3 节.

表 1.1B　　实数的范围(单精度)

IBM PC(Basic)	2.9E-39	1.7E+38
IBM PC(Quick C)	1.4E-45	3.4E+38
IBM 370	5.4E-79	7.2E+75
IRIS(SG)	1.4E-45	3.4E+38
Cray XMP	4.6E-2 476	5.4E+2 465
VAX	2.9E-39	1.7E+38

1.3.3　计算机中的舍入误差

1.3.3.1　机器小数和舍入误差

计算机中最基本的误差来源归咎于用有限数位来表示一个数时所产生的误差.

正如前面已经解释的,机器小数是 1 和另一个大于 1 的最小数之差,一般记为 ε_{mach}. 我们举一个简单的例子来说明.

假设我们用四个数位的实数进行运算:

$$1.000 \times 10^{0} + 1.000 \times 10^{-4} = 1.000 + 0.0001000 = 1.0001 \rightarrow 1.000 \times 10^{0},$$

$$1.000 \times 10^{0} + 1.000 \times 10^{-3} = 1.000 + 0.001000 = 1.001 \rightarrow 1.001 \times 10^{0}.$$

机器小数可用下列 C 语言程序求出来:

```
\vspace{5cm}
% \vskip{1 cm}
% 以下是机器小数的 C 语言程序.
% $/ * \ Machine Epsilon\ C1-1.C \ * /$
% $\# include\ < stdio.h >$
% $main( \)$
% $ \{$
% $float\ex,g = 1,eps;$
% \\$do $
% \\$\{$
% \\\$g = g/2;$
% \\\$ex = g * 0.98 + 1$
% \\\$ex = ex - 1;$
% \\\$printf( " g = \% 15.8e\ex = \% 15.8e\backslash n" ,g,ex);$
% \\\$if( ex > 0);$
% \\$\} while( ex > 0);$
% \\$printf( " \backslash n\ Machine\ epsilon = \% \ 16.8e\\backslash n\% \backslash n",
eps);$
% $ \}$
```

几个不同计算机的机器小数见表 1.2.

表 1.2　　机器小数

精度	IBM PC	IBM 370	VAX ll	Cray XMP
单	1.19E - 7	9.53E - 7	1.19E - 7	3.55E - 15
双	2.77E - 7	2.22E - 16	2.77E - 17	1.26E - 29

下面来看看机器小数与舍入误差的关系.

对任何数 $1+\alpha$, 若 $0 < \alpha < \frac{\varepsilon_{mach}}{2}$, 则舍弃 α, 数变成 1; 若 $\alpha \geqslant \frac{\varepsilon_{mach}}{2}$, 则舍入,数变成 $1+\alpha$ (相当于数学上的四舍五入). 这样 $\frac{\varepsilon_{mach}}{2}$ 被认为是 1 的最大可能的舍入误差. 对任何一个实数 R, 它与下一个实数之差大约为 $\varepsilon_{mach} \times R$. 在计算机内存储 R 时,舍入误差大约为 $\frac{\varepsilon_{mach}}{2}$.

机器小数是一个比较重要的概念,在以后数值解法的误差分析过程中会经常用到.

1.3.3.2　舍入误差的问题

当两个数相加或相减时需要相当多的数位才能准确地表示一个结果. 在下列两种情况下会有严重的舍入误差:

(a) 当一个很大的数加(或减去) 一个很小的数时;

(b) 当两个相近的数相减时.

对第一种情形,我们考察在单位 1 上加 10 000 次 0.000 01 的 C 程序:

```
\vspace{3cm}
%$/ * \Summation by Single Precision sum\_sing1. C\ * /$
%
%$\#\ include\ <stdio. h >$
%
%$main\( \)$
%
%$ \{$
%
%$float\ x,sum = 1.0;$
%
%$int\ i,k = 0;$
%
%\\$( i = 1;i < = 10000;i + + )$
%
%\\$\{ sum = sum + 0.00001;$
%
%\\$ \}$
%
%\\$printf( " \backslash n\sum = \% f\backslash n" ,sum) ;$
%
%$\}$
```

这个程序在 IBM PC 机上的结果是 sum = 1.100 136.

由于精确解是 1.1,因此这个计算的相对误差是

$$\frac{1.1 - 1.100\ 136}{1.1} = -0.000\ 124 \text{ 或 } -0.012\ 4\%.$$

下面我们来说明两数相减时舍入误差的影响,从微积分学中我们知道

$$\lim_{\theta \to 0} \frac{f(x + \theta) - f(x)}{\theta} = f'(x).$$

令 $f(x) = \sin x$, 计算

$$d = \frac{\sin(1 + \theta) - \sin 1}{\theta}.$$

在 IBM PC 机上用 Basic 语言对不同的 θ 计算结果如下：

θ	d	误差(实际值 $-d$)
0.1	0.49736	0.042938
0.01	0.53607	0.004224
0.001	0.53989	0.00403
0.0001	0.54061	-0.003111
0.00001	0.53644	0.003860
0.000001	0.53644	0.003860
0.0000001	0.59604	-0.055744

$$\text{实际值} = \cos(1) = 0.54030$$

很明显，当 $\theta = 0.001$ 时，d 接近于准确值；而当 θ 进一步减小时误差却开始增大，原因是随着 θ 的减小，$f(1+\theta)$ 与 $f(1)$ 变得越来越近，舍入误差相对地增大了.

1.3.3.3 舍入误差产生的原因

为了解释舍入误差是怎样发生的，首先让我们考察在 IBM PC 机上作 1 + 0.00001 的计算(用 C 或 Basic 语言).

1 与 0.00001 的二进制表示分别为

$$(1)_{10} = (0.1000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 \times 2^1,$$

$$(0.00001)_{10} = (0.1010\ 0111\ 1100\ 0101\ 1010\ 1100)_2 \times 2^{-16}.$$

这两个数的和 1 + 0.00001 变成

$$(1)_{10} + (0.00001)_{10} = (0.1000\ 0000\ 0000\ 0000\ 0101\ 0011\ 1110\ 0010\ 1101\ 0010)_2 \times 2^1.$$

因为尾数只能有 24 位，所以 * 以及 * 以后的数都被舍入，在内存中存储的计算结果是

$$(1)_{10} + (0.00001)_{10} \approx (0.1000\ 0000\ 0000\ 0000\ 0101\ 0100)_2 \times 2^1 = (1.0000100136)_{10}.$$

这样，无论什么时候 0.00001 加到 1，结果会增加 0.0000000136 作为误差. 当另外的 0.0000 加到 1 重复 10000 次时，误差的积累为 $10000 \times 0.0000000136 = 0.000136$，这就是舍入误差.

其次，让我们考察 1.00001 - 1 的计算.

1.00001 的二进制表示为

$$(1.00001)_{10} = (0.1000\ 0000\ 0000\ 0000\ 0101\ 0100)_2 \times 2^1.$$

因此，1.00001 - 1 变成

$$\begin{aligned}&(0.1000\ 0000\ 0000\ 0000\ 0101\ 0100)_2 \times 2^1 - (0.1)_2 \times 2^1\\ &= (0.0000\ 0000\ 0000\ 0000\ 0101\ 0100)_2 \times 2^{10}\\ &= (0.10101)_2 \times 2^{-16}\\ &= (1.00136)_{10} \times 10^{-5}.\end{aligned}$$

而准确值为 0.00001，所以相对误差为 $\dfrac{0.00001 - 0.0000100136}{0.00001} = -0.00136$ 或