

微分方程数值解

陈文斌 程 晋 吴新明 李立康 编著



博学·数学系列



復旦大學 出版社

www.fudanpress.com.cn

微分方程数值解

陈文斌 程 晋 吴新明 李立康 编著



博学·数学系列



復旦大學出版社

www.fudanpress.com.cn

图书在版编目(CIP)数据

微分方程数值解/陈文斌、程晋、吴新明、李立康编著. —上海:复旦大学出版社,2014.8
(复旦博学·数学系列)
ISBN 978-7-309-10786-9

I. 微… II. ①陈…②程…③吴…④李… III. 微分方程-数值计算-高等学校-教材
IV. 0241.8

中国版本图书馆 CIP 数据核字(2014)第 139876 号



微分方程数值解

陈文斌 程 晋 吴新明 李立康 编著
责任编辑/范仁梅

复旦大学出版社有限公司出版发行
上海市国权路 579 号 邮编:200433
网址:fupnei@fudanpress.com http://www.fudanpress.com
门市零售:86-21-65642857 团体订购:86-21-65118853
外埠邮购:86-21-65109143
上海肖华印务有限公司

开本 787 × 960 1/16 印张 19.75 字数 327 千
2014 年 8 月第 1 版第 1 次印刷

ISBN 978-7-309-10786-9/O · 540
定价: 38.00 元

如有印装质量问题, 请向复旦大学出版社有限公司发行部调换。
版权所有 侵权必究

内 容 提 要

本书主要介绍了常微分方程和偏微分方程的数值解法，具体包括：数值分析基础、常微分方程数值方法、椭圆型方程的差分方法、发展方程的差分方法、有限元方法简介以及有限元方法误差分析。

本书在编写过程中注重由浅入深、理论和数值实验结合；着重培养学生掌握基本的数值格式，并能对模型问题进行数值模拟和对数值结果进行一定的分析。

本书可以作为数学类各专业微分方程数值解课程的教学用书或参考书，对其他理工科学生学习常微分方程和偏微分方程数值解法也具有参考价值。

前 言

“微分方程数值解”作为计算数学方向的三大主干课程之一(其他两门课程为“数值逼近”和“数值线性代数”),各高校对这门课程一直非常重视,并先后编写了多本教材,例如 [27, 56, 78, 91–95, 127, 141]等.

由于教育部和复旦大学的教学改革,原有的数学五大专业方向合并为两个大的方向:数学和应用数学专业、信息和计算专业,原有的计算数学合并在大的信息和计算方向.同时在复旦大学,“微分方程数值解”课程也作为数学和应用数学的重要选修课程,每年有许多数学和应用数学方向的学生选修.这就需要调整课程内容,使得没有选修过“数值逼近”和“数值线性代数”的学生也能顺利完成本课程的学习.另一方面,“微分方程数值解”研究常微分方程和偏微分方程的数值方法,而相应的“数理方程”课程与“微分方程数值解”课程在同一学期进行,这要求我们对偏微分方程部分的解的定性需要作一些简单的介绍.因此在本教材中,我们一方面增加了“数值分析基础”一章,以方便非信息和计算方向的学生,特别是没有修习过“数值逼近”的学生学习本课程;另一方面,在介绍偏微分方程数值格式的时候,增加了一些简单的定性分析.

在本教材中,我们比较强调培养学生的动手能力和分析能力,特别是比较注重对数值结果的分析.在学习一些基本的离散方程和数值格式的基础上,要求读者能实现相应的算法,并对计算结果有一定的定性分析.为此,我们增加了相应的数值算例,希望学生在学习算例的基础上,能培养自己解决问题、分析问题的能力,为后续研究生阶段的进一步学习打下基础.在此要强调的是,由于Matlab软件的普及,本文的例子主要在Matlab软件环境中完成,我们鼓励学生在Fortran和C环境下重新完成相应的算例.

本书在复旦大学“微分方程数值解”的教材([56, 91, 92])上进行修改和扩充,在教材编写过程中,也参考了国内外有关的教材和论文,在此对他们一并感谢.本书的有限元部分由吴新明博士完成.

在本书的编写过程中,复旦大学数学科学学院的许多教师和学生提出了宝贵的意见和建议,编者在此谨向他们表示感谢,特别是林伟教授无偿地提供“常微分方程”的TEX模板作为参考,这里表示衷心感谢;还要感谢

现在微软工作的罗勳同学, 本书的部分算例由他在校时帮助实现; 另外要感谢研究生严阅同学和本科生何凡琛、许灏同学, 指出了许多笔误之处. 在本教材编写过程中, 得到了自然科学基金(11171077, 11331004, 91130004)支持, 也得到了复旦大学数学科学学院人才计划的支持(国家基础科学人才培养基金J1103105). 吴新明感谢青年科学基金项目(11301089)和重大研究计划(91330202)的支持. 同时, 编者也诚挚地感谢复旦大学出版社的范仁梅女士在本书编写过程中所给予的关心和支持, 以及她的细心校对, 极大地改善了本书的质量.

由于时间以及作者水平所限, 书中的错误以及不足之处在所难免, 我们衷心期望读者们给予批评与指正, 下面是email联系方式: wbchen@fudan.edu.cn.

陈文斌 程晋 吴新明 李立康
2013年12月于复旦园

目 录

第一章 数值分析基础	1
1.1 一个简单的递推格式	1
1.1.1 0.1不能被双精度精确表示	3
1.1.2 函数求值	8
1.1.3 对于初始扰动的分析	10
1.2 基本迭代格式	14
1.2.1 不动点迭代	15
1.2.2 Newton-Raphson方法	20
1.2.3 Logistic方程	24
1.3 离散范数和连续范数	27
1.4 函数的逼近	29
1.4.1 函数的插值	34
1.4.2 插值多项式的Newton表示	39
1.5 数值积分	42
1.5.1 复化求积公式	45
1.5.2 Gauss求积公式	47
1.5.3 自适应Simpson求积公式	49
第二章 常微分方程数值方法	55
2.1 常微分方程	55
2.1.1 线性系统	56
2.1.2 适定性	59
2.2 计算格式的导出	63
2.2.1 数值微分—导数的近似	63
2.2.2 Euler格式的收敛性	67
2.2.3 稳定和绝对稳定区域	70
2.3 高阶单步方法	73
2.3.1 Taylor级数法	73

2.3.2	Runge-Kutta方法	74
2.3.3	Runge-Kutta-Fehlberg格式和自适应步长调整	79
2.3.4	高阶单步方法中的基本概念	82
2.4	线性多步方法	85
2.4.1	Adams格式	85
2.4.2	Gear格式(BDF格式)	91
2.5	线性多步方法的性态分析	94
2.5.1	局部截断误差估计和相容性	94
2.5.2	线性多步方法的零稳定性	97
2.5.3	非齐次情形	102
2.5.4	收敛= 稳定+ 相容	104
2.5.5	绝对稳定性和绝对稳定区域	108
2.6	刚性问题	112
2.7	其他稳定性	118
2.8	二阶系统的求解	121
2.8.1	Newton-Störmer-Verlet-leapfrog方法	121
2.8.2	Newmark格式	123
2.8.3	Runge-Kutta方法	124
2.8.4	线性多步方法	125
第三章 椭圆型方程的差分方法		128
3.1	两点边值问题的差分方法	129
3.1.1	两点边值问题	129
3.1.2	能量意义下的稳定性	132
3.1.3	三点差分格式	136
3.1.4	紧致差分格式	144
3.1.5	收敛性分析	145
3.1.6	特征值问题	151
3.2	高维情况	155
3.3	求解器	165
3.3.1	迭代方法	166
3.3.2	多重网格	171
3.3.3	FFT算法	175
3.3.4	区域分解	178

第四章 发展方程的差分方法	187
4.1 抛物型方程	187
4.2 抛物型方程的基本差分格式	192
4.3 稳定性分析	195
4.3.1 直接法	196
4.3.2 分离变量法	201
4.3.3 传播因子法	203
4.3.4 按最大模范数稳定	207
4.3.5 交替方向方法	209
4.4 对流方程	213
4.5 波动方程	222
第五章 有限元方法简介	232
5.1 有限元方法	232
5.1.1 有限元离散	232
5.1.2 线性三角形元	234
5.1.3 单元刚度矩阵和质量矩阵	236
5.1.4 边界条件处理	237
5.2 Lagrange型单元	238
5.2.1 Lagrange型三角形元	239
5.2.2 Lagrange型矩形元	242
5.2.3 有限元定义	246
5.3 Hermite型单元	247
5.3.1 Hermite型三角形元	247
5.3.2 Hermite型矩形元	250
5.4 数值算例	252
5.4.1 一维边值问题	252
5.4.2 二维边值问题	255
5.5 时间相关问题的计算	258
5.5.1 抛物型方程	258
5.5.2 双曲型方程	262
第六章 有限元方法误差分析	266
6.1 变分问题适定性	266
6.1.1 Sobolev空间初步	266

6.1.2	Lax-Milgram引理	270
6.1.3	Poisson方程边值问题适定性	271
6.2	有限元误差估计	274
6.2.1	有限元逼近	274
6.2.2	H^1 -模估计	276
6.2.3	L^2 -模估计	278
6.3	其他类型有限元	280
6.3.1	数值积分的影响	280
6.3.2	等参有限元	283
6.3.3	非协调有限元	285
6.4	自适应有限元方法	289
6.4.1	后验误差分析	289
6.4.2	自适应算法	294
	参考文献	298

第一章 数值分析基础

这一章将从简单的递推格式开始, 介绍数值分析的一些基础知识, 说明计算什么时候是可靠的和可信的, 什么时候是会出问题的. 函数逼近和数值积分作为两个最基本的离散方法, 这里也将简单介绍. 如果前期已经选读过“数值逼近”的读者, 可以跳过本章, 或作为选读.

1.1 一个简单的递推格式

回忆一下在高中, 曾经做过的非常简单的练习:

例1.1.1 设序列 $u(n)$ 满足

$$u(n+2) - 2u(n+1) + u(n) = 0. \quad (1.1.1)$$

当 $u(1) = u(2) = 0.1$ 时, 求 $u(n)$.

显然当 $u(1) = u(2) = 0.1$ 的时候, 容易求得这个序列的每一项都等于0.1. 下面用MATLAB程序来验证这个简单的结果.

```
function u = Ch1Exam1(u0,u1,a,b,N) 1
% 用三项递推公式求序列:  $u(n+2) = a*u(n+1) + b*u(n)$  2
% 输入: u0 u1: 初始值 3
%      a,b: 系数 4
%      N: 递推步数 5
% 输出: u(1:N) 6
% 例如: u=Ch1Exam1(0.1,0.1,2,-1,100); 7
%      u=Ch1Exam1(0.1,0.1,3,-2,100); 8
u = [u0; u1]; 9
for i=1:N-2 10
    u2 = a*u1 + b*u0; 11
    u0 = u1; u1 = u2; 12
    u = [u; u2]; 13
end 14
```

如果在MATLAB中执行这个程序

```
>>u=Ch1Exam1(0.1, 0.1, 2, -1, 100); plot(u);
```

那么就会形成一个100维的向量 \mathbf{u} , 其元素全部是0.1(见图1.1).

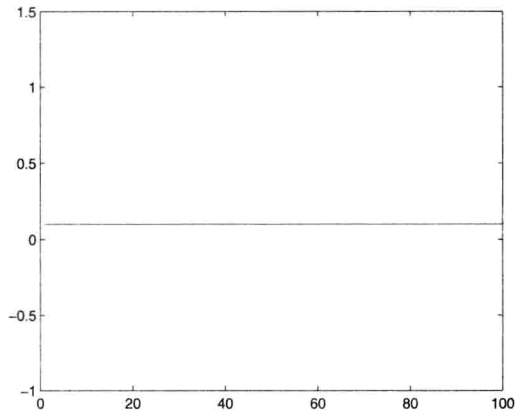


图 1.1: 100维的向量 \mathbf{u}

注意到利用上面的程序, 可以计算如下的三项递推公式:

$$u(n+2) = au(n+1) + bu(n). \quad (1.1.2)$$

下面再做个试验, 取 $a = 3, b = -2$, 也就是说求如下的问题.

例1.1.2 用程序Ch1Exam1.m计算 $u(n)$, 这里 $u(n)$ 满足

$$u(n+2) - 3u(n+1) + 2u(n) = 0, \quad (1.1.3)$$

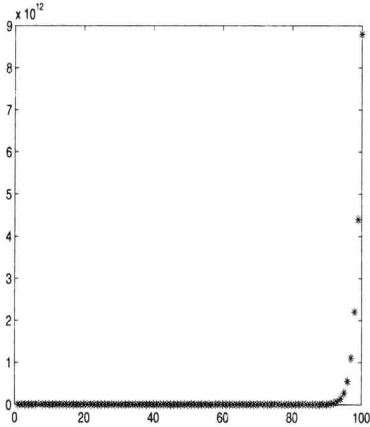
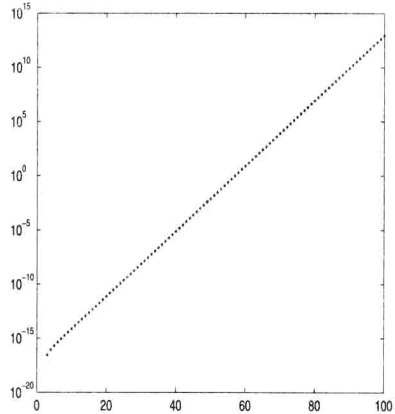
$$u(1) = 0.1, \quad u(2) = 0.1. \quad (1.1.4)$$

显然如果把初始值代到递推公式, 容易知道这里 $u(n)$ 也总是等于0.1. 现在通过给定初始值, 可以调用如下程序Ch1Exam1:

```
>>[u]=Ch1Exam1(0.1,0.1,3,-2,100);figure(1);plot(u,'*')
```

图1.2(a)的计算结果与我们的想象差得实在太远, 这里算出来的 $u(n)$ 并不是常数0.1, 而是随着迭代步数 n 的增大, 增长得非常快(指数增长)!

现在来观察误差的半对数图, 也就是用半对数图来画计算得到的 $u(n)$ 和精确解的差:

(a) 计算解 $u(n)$ 

(b) 计算解与精确解误差的半对数图

图 1.2: 例子1.1.2

```
>>e=u-0.1; semilogy(e, '.')
```

从误差的半对数图图1.2(b)可以看到: 随着求解的进行, 在半对数图里误差近似于一条直线, 也就是说误差随迭代步数指数增长. 如果看 $n = 100$ 的误差, 有 $e(100) \approx 8.7961 \times 10^{12}$, 精确解完全被误差淹没.

这个例子告诉我们: 哪怕简单的计算格式都可能产生问题. 那么在例子1.1.2中, 问题到底是如何引起的呢? 误差为什么会指数增长呢? 这要从计算机中数的表示和存储说起.

1.1.1 0.1不能被双精度精确表示

下面做一个简单的练习.

例1.1.3 在MATLAB中检查一下 $0.3 - 0.2$ 和 $0.3 - 0.2 - 0.1$.

```
>>0.3 - 0.2
    0.1000
>>0.3 - 0.2 - 0.1
   -2.7756e-017
```

从这里知道, 在MATLAB实际计算中:

$$0.3 - 0.2 - 0.1 \neq 0 \quad (\text{MATLAB}).$$

这种现象是由于数在计算机中的二进制存储形式引起的, 这种误差被称为舍入误差. 这里 $0.3 - 0.2 - 0.1$ 得到的舍入误差是 $-2.7756e - 017$, 这是一个很小的数, 但是不为0.

在计算机中, 实数以二进制的形式存储, 那么以2为底(或以16为底), 可以把0.1写成级数的形式:

$$\begin{aligned} 0.1 &= \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{0}{2^{10}} + \frac{0}{2^{11}} + \frac{1}{2^{12}} + \cdots \\ &= 2^{-4} \left(1 + \frac{9}{16} + \frac{9}{16^2} + \cdots \right). \end{aligned}$$

由于这个级数不能展开成有限项, 这就意味着:

0.1不能用有限位的寄存器精确存储.

在工业上, IEEE754-1985标准(新的标准为IEEE754-2008, 等价的国际标准为ISO/ IEC/IEEE 60559:2011)用来规范浮点数的存储和运算. 在存储之前, 0.1首先要按照IEEE754-1985标准进行规范化:

$$0.1 = +2^e \cdot (1 + f)_2, \quad e = -4, \quad f = 10011001 \cdots$$

在实际存储的时候, 需要存3个信息(见表1.1):

表 1.1: 数的存储

符号位	指数部分	尾数部分
s	e	f

- 符号位 s : $s = 0$ 表示正数, $s = 1$ 表示负数.
 - 指数部分 e : 如果是单精度, 用8位表示, 并存储成 $e + 127$; 如果是双精度, 用11位表示, 并存储成 $e + 1023$.
 - 尾数部分 f : 如果是单精度, 则存储23位; 如果是双精度, 则存储52位.
- 这样可以得到数0.1的两种存储格式:

- 单精度存储(single precision). 0.1表示成规范的(normalized)单精度数为

$$0.1 = (-1)^s \cdot (2^{e-127}) \cdot (1.f)_2.$$

$s = 0$, 这里

$$\begin{aligned} e &= -4 + 127 = 123 = (01111011)_2, \\ 1.f &= 1.10011001100110011001101, \end{aligned}$$

见表1.2, 注意这里 $1.f$ 中的首1不需存储以节省空间. 因此0.1的单精度表示为

$$(0\ 01111011\ \underline{10011001100110011001101})_2 = (3dcccccd)_{16}. \quad (1.1.5)$$

这里左边是2进制表示, 右边是16进制表示. 注意到最后一位是从1001中舍入而成101.

表 1.2: 0.1的单精度表示

b_{31}	$b_{30} \sim b_{23}$	$b_{22} \sim b_0$
s	e	f
0	01111011	10011001100110011001101

- 双精度存储(double precision). 同样0.1表示成规范的双精度数为

$$0.1 = (-1)^s \cdot (2^{e-1023}) \cdot (1.f)_2,$$

这里

$$\begin{aligned} s &= 0, \quad e = -4 + 1023 = 1019 = (01111111011)_2, \\ 1.f &= 1.\underline{1001100110011001100110011001100110011001100110011010}. \end{aligned}$$

这里最后一位也有进位. 同时注意这里的 f 和由单精度得到的 f 从小数点后的第23位是不同的:

$$\text{double}(0.1) - \text{single}(0.1) \approx -1.4901161 \times 10^{-9}.$$

因此, 0.1表示成双精度为

$$\begin{aligned} (0\ 01111111011\ 1001100110011001100110011001100110011001100110011010)_2 \\ = (3fb9999999999999a)_{16}. \end{aligned} \quad (1.1.6)$$

从这个表示可以知道:

$$0.1 < \text{double}(0.1) < 0.1 + 2^{-4} \cdot \frac{\text{eps}}{2}, \quad \text{eps} = 2^{-52}, \quad (1.1.7)$$

这里eps就是浮点表示相对精度(floating-point relative accuracy): 在计算机中下一个比1.0大的浮点数离1.0的距离. 在单精度时: $\text{eps} = 2^{-23} \approx 1.1921 \times 10^{-7}$; 在双精度时: $\text{eps} = 2^{-52} \approx 2.2204 \times 10^{-16}$.

注1.1.4 严格来讲, IEEE754标准中并没有硬性规定最后位如何舍入, 而是建议了五种舍入的方式, 感兴趣的读者可以查阅IEEE754的文档.

下面在MATLAB中看一下, 0.1存储格式是否如上面所述.

例1.1.5 查看0.1在MATLAB中的表示.

```
>>x=0.1;y=single(x); %在版本号大于7.0时
>>format hex;
>>x
      x = 3fb9999999999999a
>>y
      y = 3dcccccd
```

在MATLAB7.0及以上版本中, 提供了single函数, 用来从双精度到单精度的转化(相应地, double函数得到双精度表示). 这里用format hex来得到16进制的表示(用format short返回MATLAB默认的5位有效数表示). 可以看到, 这里用16进制表示时, x 与双精度表示(1.1.6)式一致, 而 y 与单精度表示(1.1.5)式一致.

例1.1.6 检查在MATLAB中

$$0.3 - 0.2 - 0.1 \quad \text{和} \quad 0.3 - (0.2 + 0.1)$$

分别等于多少.

来看一下MATLAB中的运算结果:

```
>>0.3 - 0.2 - 0.1
      -2.7756e-017
>>0.3 - (0.2 + 0.1)
      -5.5511e-017
```

从这里例子可以发现一些有趣的现象:

(1) 在MATLAB中, 计算结果与运算次序有关:

$$0.3 - 0.2 - 0.1 \neq 0.3 - (0.2 + 0.1). \quad (1.1.8)$$

(2) 即使最简单的加、减法都有可能产生误差:

$$0.3 - 0.2 - 0.1 \neq 0, \quad 0.3 - (0.2 + 0.1) \neq 0. \quad (1.1.9)$$

从对数0.1表示的分析容易知道, 这里出现的两个现象都是因为0.1, 0.2和0.3不能被双精度精确表示, 从而在表示这3个数的时候就产生了误

差(在第53位舍入或舍去),同时在计算过程中又会产生新的舍入问题,这就造成了计算次序不同,从而得到的结果不同.这种由于有限位表示而导致的舍入和舍去引起的误差称为舍入误差(rounding error, 或round-off error).

一般地,由于舍入误差的存在,对最基本的算术运算,假设有如下的模型 [75]:

$$fl(x \text{ op } y) = (x \text{ op } y)(1 + \delta), \quad |\delta| \leq \frac{1}{2}\text{eps}, \quad \text{op} = +, -, \times, /.$$
 (1.1.10)

这里 $fl(x)$ 表示 x 的浮点数表示, $x \text{ op } y$ 表示 x 与 y 精确运算的结果.从这个模型可以看到,最基本的四则运算的相对误差是

$$\text{reerror} = \frac{|fl(x \text{ op } y) - x \text{ op } y|}{|x \text{ op } y|} = |\delta| \leq \frac{\text{eps}}{2}.$$
 (1.1.11)

在计算过程中,常常用解和计算解的相对误差来度量精确的程度.对于更复杂的常用基本函数(比如指数函数和三角函数)的计算,一般认为其相对误差可以被某个常数 C 乘以浮点数表示的相对精度 eps 来控制.

在MATLAB中, eps 可以由直接输入 eps 命令得到.而按照 eps 的定义:在计算机中下一个比1.0大的浮点数离1.0的距离.一个常见的计算 eps 的程序如下:

```
function x=findeps      1
x=1;                    2
while (1+x/2>1)        3
    x=x/2;              4
end                      5
```

注意到在MATLAB中, $1 + \text{eps} > 1$ 而 $1 + \text{eps}/2 = 1$.这也就是通常说的“大数吃小数”现象:当一个大的数和一个小的数相加,小数可能会不起作用.

另外一个值得特别注意的是抵消(cancellation)现象:相近的两个数相减,得到的数只有很少的有效位数.一个例子是计算

$$f(x) = \frac{1 - \cos(x)}{x^2}.$$

当 x 趋于0的时候,容易知道 $f(x)$ 接近0.5,但是由于抵消现象的存在,计算结果是令人惊讶的,这个留作练习.

注1.1.7 (0.1引起的灾难) 在海湾战争中,爱国者导弹里时钟系统寄存器的间隔为0.1,而设计中0.1被存储为

$$(0.00011001100110011001100)_2 = \frac{209715}{2097152}.$$