北京化工大学

Beijing University of Chemical Technology

计算方法讲义 (一)

绪论

Cheng Yong

目录

第 1 章 绪论 **1**

1.1 数值计算问题 1
1.2 数学求解一般步骤 5
1.3 算法 6
1.4 数值算法设计思想 8
1.4.1 化大为小的缩减技术 8
1.4.2 化难为易的校正技术 13
1.4.3 化粗为精的松弛技术 15
1.5 误差 17
1.5.1 误差的来源 17
1.5.2 误差的度量 17
1.6 有效数字 18
1.7 误差度量之间的联系 19
1.8 算法选用原则 19
1.9 本章小结 20

创建日期：2019 年 7 月 5 日
更新日期：2020 年 2 月 12 日

第 1 章　绪论

1.1　数值计算问题

例 1. 引力波例子。

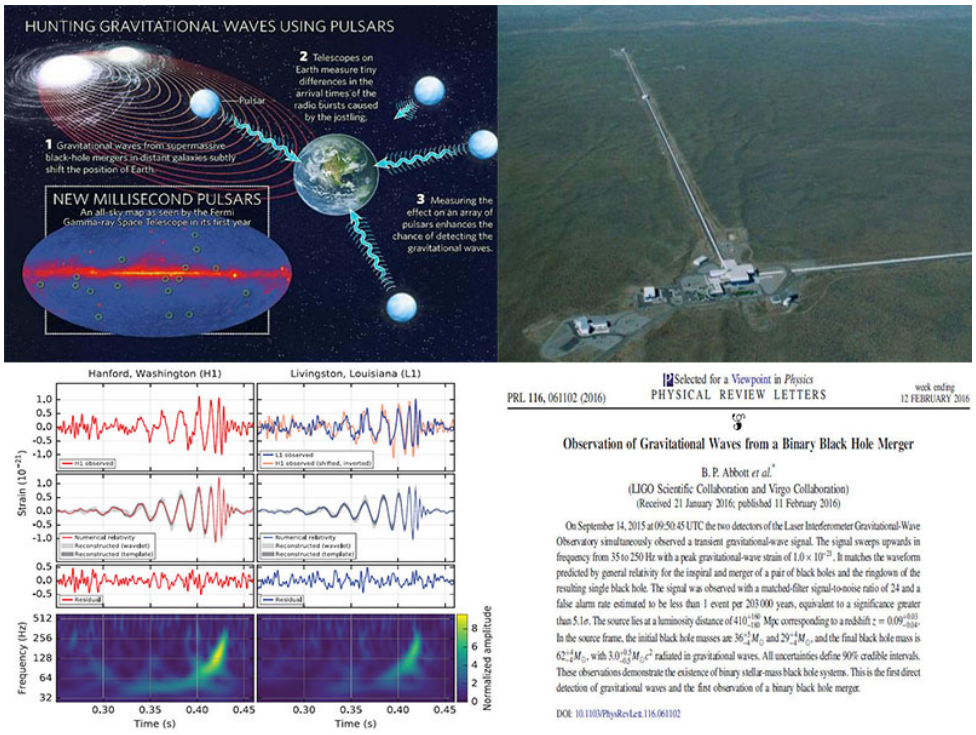
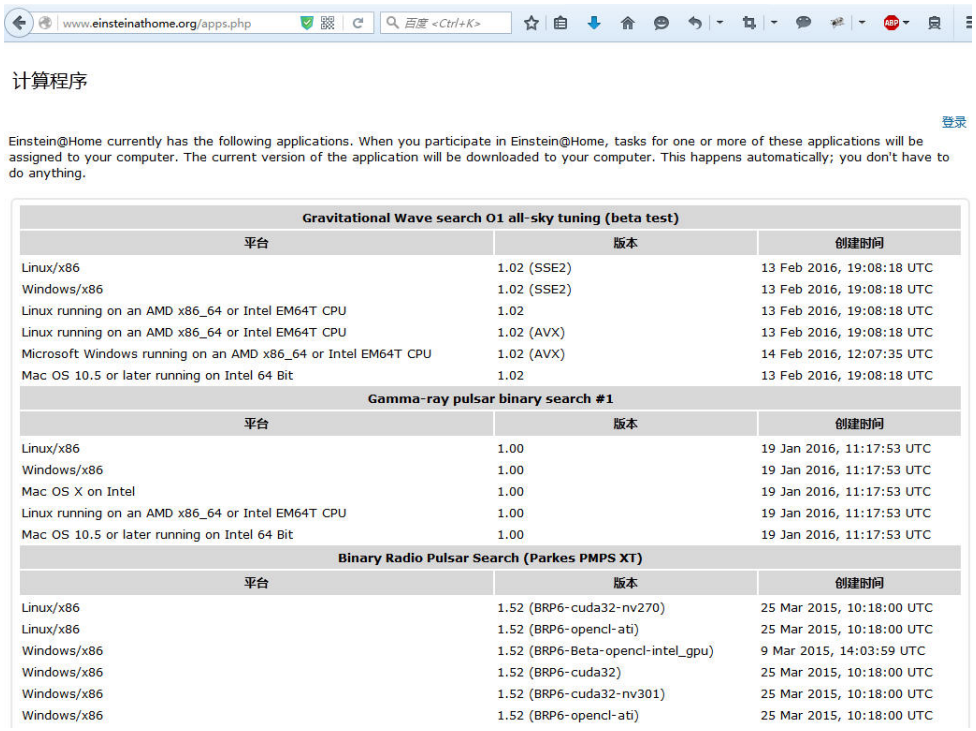


图 1.1: 引力波

例 2. 引力波发现中的科学计算。



www.einsteinathome.org/apps.php

计算程序

登录

Einstein@Home currently has the following applications. When you participate in Einstein@Home, tasks for one or more of these applications will be assigned to your computer. The current version of the application will be downloaded to your computer. This happens automatically; you don't have to do anything.

Gravitational Wave search O1 all-sky tuning (beta test)		
平台	版本	创建时间
Linux/x86	1.02 (SSE2)	13 Feb 2016, 19:08:18 UTC
Windows/x86	1.02 (SSE2)	13 Feb 2016, 19:08:18 UTC
Linux running on an AMD x86_64 or Intel EM64T CPU	1.02	13 Feb 2016, 19:08:18 UTC
Linux running on an AMD x86_64 or Intel EM64T CPU	1.02 (AVX)	13 Feb 2016, 19:08:18 UTC
Microsoft Windows running on an AMD x86_64 or Intel EM64T CPU	1.02 (AVX)	14 Feb 2016, 12:07:35 UTC
Mac OS 10.5 or later running on Intel 64 Bit	1.02	13 Feb 2016, 19:08:18 UTC
Gamma-ray pulsar binary search #1		
平台	版本	创建时间
Linux/x86	1.00	19 Jan 2016, 11:17:53 UTC
Windows/x86	1.00	19 Jan 2016, 11:17:53 UTC
Mac OS X on Intel	1.00	19 Jan 2016, 11:17:53 UTC
Linux running on an AMD x86_64 or Intel EM64T CPU	1.00	19 Jan 2016, 11:17:53 UTC
Mac OS 10.5 or later running on Intel 64 Bit	1.00	19 Jan 2016, 11:17:53 UTC
Binary Radio Pulsar Search (Parkes PMPS XT)		
平台	版本	创建时间
Linux/x86	1.52 (BRP6-cuda32-nv270)	25 Mar 2015, 10:18:00 UTC
Linux/x86	1.52 (BRP6-opencl-ati)	25 Mar 2015, 10:18:00 UTC
Windows/x86	1.52 (BRP6-Beta-opencl-intel_gpu)	9 Mar 2015, 14:03:59 UTC
Windows/x86	1.52 (BRP6-cuda32)	25 Mar 2015, 10:18:00 UTC
Windows/x86	1.52 (BRP6-cuda32-nv301)	25 Mar 2015, 10:18:00 UTC
Windows/x86	1.52 (BRP6-opencl-ati)	25 Mar 2015, 10:18:00 UTC

图 1.2: 引力波发现中的科学计算

例 3. *AlphaGo by Google*。

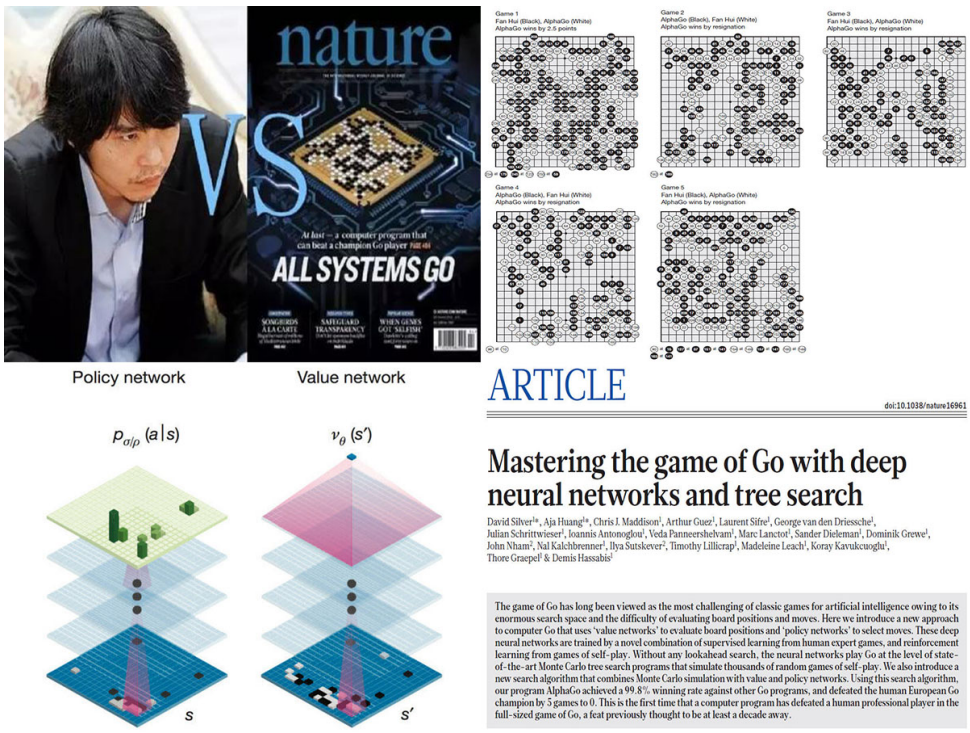


图 1.3: AlphaGo

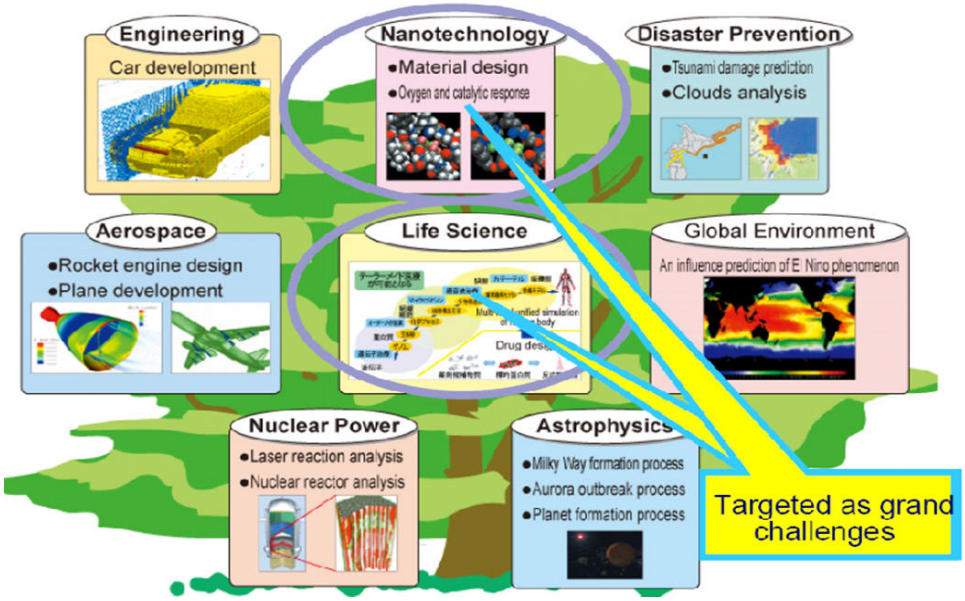


图 1.4: 科学计算领域

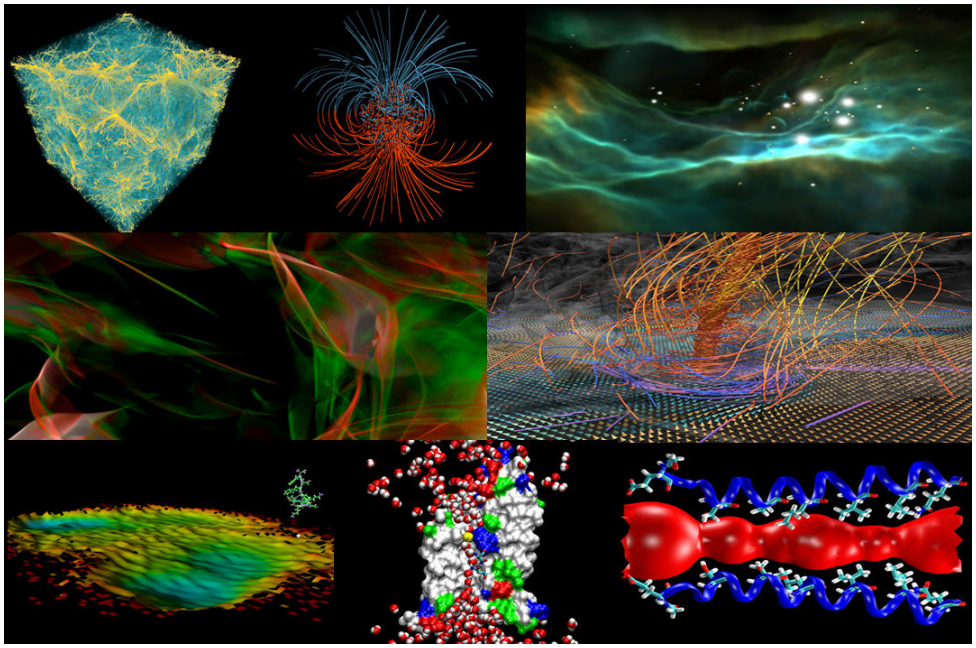


图 1.5: 科学计算需求

例 4. 计算多项式的值。

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \tag{1.1}$$

解：在计算过程中，需要执行乘法次数为

$$n + (n-1) + (n-2) + \cdots + 1 = \frac{(1+n)\cdot n}{2} \tag{1.2}$$

执行加法为 n 次。

如果对上述多项式进行如下变换：

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \tag{1.3}$$

$$= (a_n x + a_{n-1})x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \tag{1.4}$$

$$= ((a_n x + a_{n-1})x + a_{n-2})x^{n-2} + \cdots + a_1 x + a_0 \tag{1.5}$$

$$= (\cdots(a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_1)x + a_0 \tag{1.6}$$

此时进行多项式计算，需要执行乘法次数为 n 次，执行加法为 n 次，由此可见大大地减少了乘法计算次数，减少了计算时间。

例 5. 计算二阶行列式 $D = \begin{vmatrix} 1 & 3 \\ 2 & 4 \end{vmatrix}$ 的值。

$$D = \begin{vmatrix} 1 & 3 \\ 2 & 4 \end{vmatrix} = 1 \times 4 - 3 \times 2 = -2 \tag{1.7}$$

例 6. 计算三阶行列式 $D = \begin{vmatrix} 1 & 2 & 0 \\ 4 & -3 & 8 \\ 0 & -1 & 2 \end{vmatrix}$ 的值。

$$D = \begin{vmatrix} 1 & 2 & 0 \\ 4 & -3 & 8 \\ 0 & -1 & 2 \end{vmatrix} \tag{1.8}$$

$$= 1 \times (-3) \times 2 + 2 \times 8 \times 0 + 0 \times 4 \times (-1) \tag{1.9}$$

$$- 0 \times (-3) \times 0 - 2 \times 4 \times 2 - 1 \times 8 \times (-1) \tag{1.10}$$

$$= -14 \tag{1.11}$$

例 7. 计算如下行列式的值。

行列式表示如下：

$$\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \tag{1.12}$$

解：n 阶行列式

$$D = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_m \end{vmatrix} = \sum_{(p_1 p_2 \cdots p_n)} (-1)^{\tau} a_{1p_1} a_{2p_2} \cdots a_{np_n} \tag{1.13}$$

其中，$\tau = \tau(p_1 p_2 \cdots p_n)$，求和式中共有 $n!$ 项。

$n = 1$ 时，需要 0 次乘法，2 阶需要 2 次。

假设 n 阶需要 $f(n)$ 次乘法运算。则 $f(n) = n * f(n-1)$。

很容易就可以看出，n 阶需要 $n!$ 次乘法运算 ($n \geq 2$ 时)。

当 $n = 24$ 时，$2^{24} = 16777216$，而 $24! \approx 6.20449 * 10^{23}$，现在的世界最强的天河计算机每秒执行 5.49 亿亿次乘法，执行 24! 次乘法大约需要 1 年。

当 $n = 64$ 时，需要计算约 $5.0895 * 10^{54}$ 个宇宙年龄。

1.2 数学求解一般步骤

笛卡尔曾提出过被后世尊称为 “万能法则” 的一般模式：

1. 将实际问题化归为数学问题；
2. 将数学问题化归为代数问题；
3. 将代数问题化归为解方程；

科学计算求解步骤：

1. 提出实际问题；
2. 建立数学模型；
3. 提出数值问题；
4. 设计可靠、高效的算法；
5. 程序设计、上机实践计算结果；
6. 计算结果的可视化；

本课程主要讲解五类数值求解问题，分别是：

1. 插值问题；
2. 数值求积问题；
3. 非线性方程求根问题；
4. 线性方程组求根问题；
5. 常微分方程求解问题；

1.3 算法

科学计算离不开算法设计。所谓算法 (Algorithm) 是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令，算法代表着用系统的方法描述解决问题的策略机制。也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出。

算法的主要性质：

- 有穷性。算法的有穷性是指算法必须能在执行有限个步骤之后终止；
- 确切性。算法的每一步骤必须有确切的定义；
- 输入项。一个算法有 0 个或多个输入，以刻画运算对象的初始情况，所谓 0 个输入是指算法本身定出了初始条件；
- 输出项。一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；
- 可行性。算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步；

算法的描述方法包括：

- 自然语言。自然语言就是用人们日常使用的语言描述解决问题的方法和步骤，这种描述方法通俗易懂，在语法和语义上往往具有多义性；
- 伪代码。伪代码是介于自然语言和计算机语言之间的文字和符号；
- 传统流程图。传统流程图，使用不同的几何图形来表示不同性质的操作，使用流程线来表示算法的执行方向，比起前两种描述方式，其具有直观形象、逻辑清楚、易于理解等特点，但它占用篇幅较大，流程随意转向，较大的流程图不易读懂；

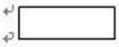
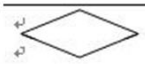
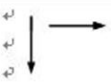
流程图符号	名称	说明
	起止框	表示算法的开始和结束
	处理框	表示完成某种操作，如初始化或运算赋值等
	判断框	表示根据一个条件成立与否，决定执行两种不同操作的其中一个
	输入输出框	表示数据的输入输出操作
	流程线	用箭头表示程序执行的流向
	连接点	用于流程分支的连接

图 1.6: 传统流程图

- N-S 结构化流程图。N-S 结构化流程图是 1973 年美国学者 Nassi 和 Shneiderman 提出的一种符合结构化程序设计原则的描述算法的图形方法。

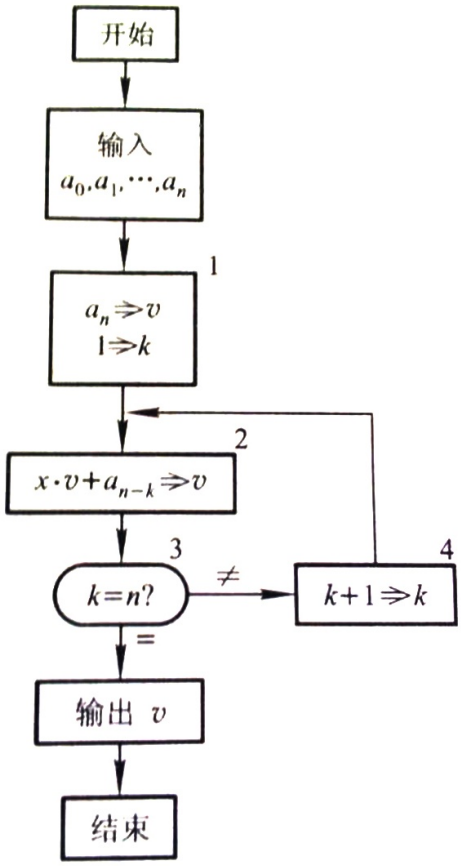


图 1.7: 秦九韶算法流程图

```
1
2  #include "stdio.h"
3  #define N 100
4  int main() {
5      int n,k,i;
6      float a[N],x;
7      double v;
8
9      /*输入数据*/
10     printf("please input n= ");
11     scanf("%d",&n);
12     printf("please input fator a0,a1,...,a%d= ",n);
13     for(i=0; i<=n; i++) {
14         scanf("%f",&a[i]);
15     }
16     printf("please input x= ");
17     scanf("%f",&x);
```

```
18      v=a[0];
19
20      /*判断 k是否等于 n*/
21      for(k=1; k<=n; k++)
22          v=x*v+a[k]; // 这一步非常关键
23
24      /*输出数据*/
25      printf("the answer is v=%lf",v);
26      getch();
27      return 0;
28  }
```

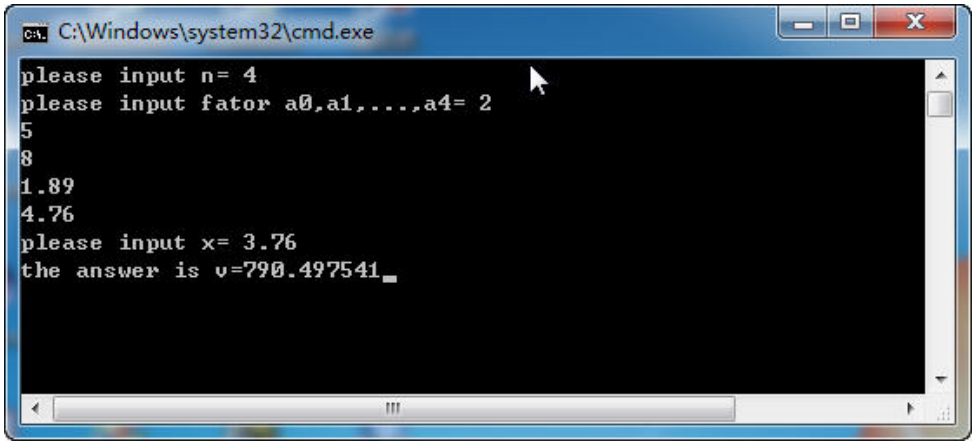


图 1.8: 秦九韶算法结果

1.4 数值算法设计思想

数值算法主要设计思想：

- 化大为小的缩减技术；
- 化难为易的校正技术；
- 化粗为精的松弛技术；

1.4.1 化大为小的缩减技术

缩减技术是数值算法设计的一种基本思想。

许多数值计算问题可以引进某个实数—即所谓问题的规模来刻画其“大小”，而问题的解恰好是其规模为足够小的退化情形。求解这类问题的一种行之有效的方法是通过某种简单的运算手续逐步缩减问题的规模，直到加工出问题的解。

所谓大事化小即指逐步压缩问题的规模，这个处理过程应具有结构递归和规模递减两项基本特征。小事化了即是指当问题的规模变得足够小时即可直接或方便得出问题的

解。通常对于规模大但有限的问题如上述数列求和问题设计出来的一类算法统称直接法。

在运用缩减技术的过程中是如何实现“大事化小，小事化了”的呢？这个处理过程主要有两项基本特征：

结构递归

大事化小是逐步完成的，其每一步将所考察的计算模型加工成同样类型的计算模型，因而这类算法具有明晰的递归结构。

规模递减

每一步加工前后的计算模型虽然属于同一类型，但是问题的规模已经被压缩了。压缩系数越小，则算法的效率就越高。

算法设计的这种技术称为规模缩减技术，简称缩减技术。下面将举例说明这种设计思想的具体运用。

例 8. *Zeno* 悖论

古希腊哲学家 Zeno 在两千多年前提出过一个耸人听闻的命题：一个人不管跑得多快，也永远赶不上一只爬在他 (她) 前面的乌龟，这就是著名的 Zeno 悖论。

设人与龟的速度分别为 V 与 v，S_k 表示逼近过程中的第 k 步人与龟的间距，t_k 表示相应的时间，相邻两步的时间差为 $\Delta t_k = t_k - t_{k-1}$。Zeno 悖论将人与龟的追赶问题分为追和赶两个过程：

1. 追的过程　先令乌龟不动，计算人追赶龟所需要的时间：

$$\Delta t_k = \frac{S_{k-1}}{V} \tag{1.14}$$

2. 赶的过程　再令人不动，计算龟在这段时间内爬行的时间：

$$S_k = v\Delta t_k \tag{1.15}$$

经过这两步之后，加工后的新问题的规模被压缩了 v/V 倍，即

$$S_k = \frac{v}{V}S_{k-1} \tag{1.16}$$

实际上，设 $S_0 = S$ 为已知，令 $t_0 = 0$，按上述步骤做不了几步，问题的规模 S_k 就可以忽略不计了，从而得到人追赶龟所花费的时间 t_k，这一过程可用如下算法来描述：

$$\begin{cases} S_0 = S \\ S_k = \frac{v}{V}S_{k-1}, \quad k = 1, 2, \cdots \end{cases} \tag{1.17}$$

上述算法称为 Zeno 算法，这也是 Zeno 悖论的算法形式描述。

例 9. 数列求和的累加算法

数列求和问题

$$S = a_0 + a_1 + \cdots + a_n \tag{1.18}$$

两种退化形式，当 $n = 0$ 时，$S = a_0$；当 $n = 1$ 时，$S = a_0 + a_1$。我们再考察上面的累加求和问题，设 b_k 表示前 $k + 1$ 项的部分和 $a_0 + a_1 + \cdots + a_k$，则有：

$$\begin{cases} b_0 = a_0 \\ b_k = b_{k-1} + a_k, \quad k = 1, 2, \cdots, n \end{cases} \tag{1.19}$$

则计算结果 b_n 即为所求的和值 S。

累加求和算法所刻画的计算模型的一个重要特征是：它具有多层嵌套结构：

$$S = (((((a_0) + a_1) + a_2) + \cdots) + a_{n-1}) + a_n \tag{1.20}$$

多项式求值的秦九韶算法

对给定的点 x，计算下列多项式的值：

$$P = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n = \sum_{k=0}^{n} a_k x^{n-k} \tag{1.21}$$

如果使用逐项计算的方法，所需要计算的乘法次数为：

$$Q = \sum_{k=0}^{n} (n - k) \approx \frac{n^2}{2} \tag{1.22}$$

显然，当 n 充分大的时候，计算量是非常大的。

我们可以将上述多项式表示为如下的嵌套结构

$$P = (((a_0 x + a_1)x + a_2)x + \cdots + a_{n-1})x + a_n \tag{1.23}$$

因此，我们有如下计算方法：令 $v_0 = a_0$，对 $k = 1, 2, \cdots, n$ 执行如下算式：

$$v_k = x \cdot v_{k-1} + a_k \tag{1.24}$$

则结果 $P = v_n$ 即为说给多项式 (1.21) 的值。

例 10. 二分法例子。

对于区间 $[a, b]$ 上连续不断且 $f(a) \cdot f(b) < 0$ 的函数 $y = f(x)$，通过不断地把函数 $f(x)$ 的零点所在的区间一分为二，使区间的两个端点逐步逼近零点，进而得到零点近似值的方法叫二分法。

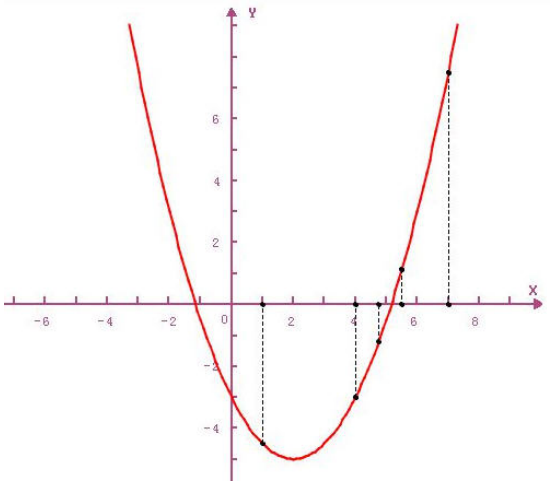


图 1.9: 二分法算法思想

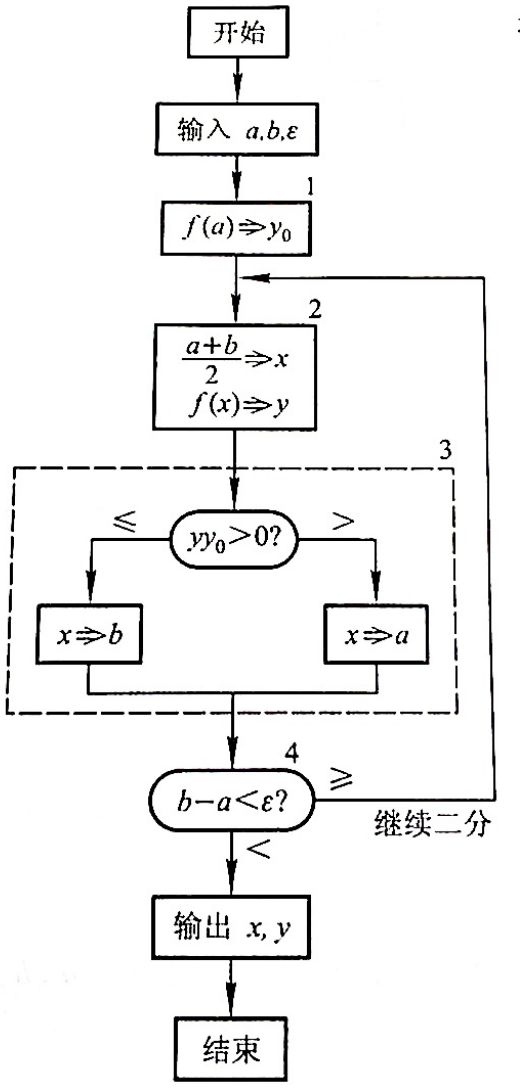


图 1.10: 二分法算法流程图

```c
#include <stdio.h>

double f(double x) {
    return x*x - 4; /*计算并返回函数值f(x)*/
}

void main(void) {
    double a,b,epsilon,x,middle;
    printf("\n请输入x的精度要求：");
    scanf("%lf",&epsilon);
    printf("\n请输入有根区间的边界a,b：");
    scanf("%lf,%lf",&a,&b);
    if(f(a)==0)x=a;
    else if(f(b)==0) x=b;
    else {
        while((b-a)/2>epsilon) {
            middle=(a+b)/2;
            if(f(middle)==0) break;
            else if(f(a)*f(middle)>0) a=middle;
            else b=middle;
        }
        x=(a+b)/2;
    }
    printf("\n方程f(x)=0的根x=%lf。",x);
}
```

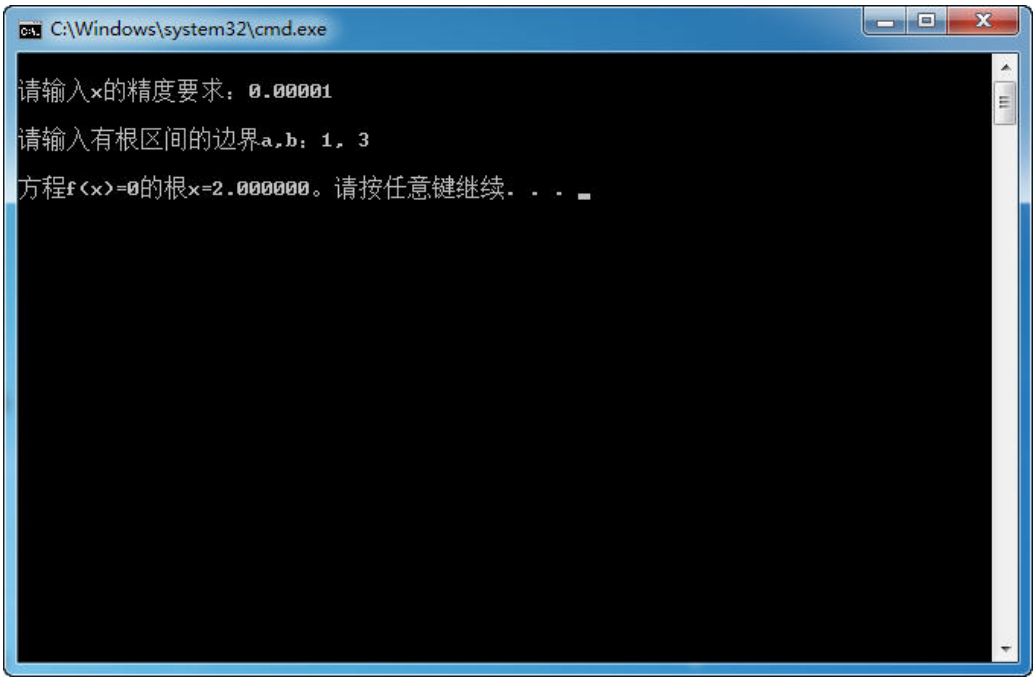


图 1.11: 二分法算法运行结果

1.4.2 化难为易的校正技术

校正技术得到的迭代法突出地体现了删繁就简，逐步求精的设计思想。“删繁就简”的含义是，删去所给复杂方程中某些高阶小量而简化生成所谓的校正方程以确定所给预报值的校正量，其中校正方程应满足逼近性和简单性两项要求。利用校正方程获得所给复杂方程解的一种行之有效的途径就是递推化，反复预报校正直到达到精度要求为止。

缩减技术主要是采用大事化小，小事化了的方法解决问题的，有些问题的 “大事化小” 过程似乎无法了结。这是一类无限逼近的过程，适于用所谓预报校正技术来处理。

对于迭代法来说，“删繁为简” 的含义是删除所给复杂方程中某些高阶的小量，简化生成所谓的校正方程，以确定所给预报值的校正量。关于校正方程有以下两项基本要求：

1. 逼近性　它与所给方程式近似的。逼近程度越高，所获得的校正量就越准确。

2. 简单性　校正方程越简单，则计算量就越小。

应当指出的是：在设计校正方程的时候，逼近性和简单性往往是顾此失彼的两个矛盾因素。逼近性越高也导致校正方程越复杂，使得计算量增加。因此，在设计校正方程时要权衡彼此的关系。

例 11. *Zeno* 悖论

在 Zeno 悖论中，人和龟的初始距离为 S，两者的速度分别为 V 和 v，容易列出方程如下：

$$Vt - vt = S \tag{1.25}$$

容易求得人追赶龟实际所花的时间为：

$$t^* = \frac{S}{V - v} \tag{1.26}$$

现在运用预报校正技术来处理这一问题，假设有预报值 t_0，希望通过某个校正值 Δt 使得校正值

$$t_1 = t_0 + \Delta t \tag{1.27}$$

更好地满足方程 (1.25)，即使方程更准确地成立：

$$V(t_0 + \Delta t) - v(t_0 + \Delta t) \approx S \tag{1.28}$$

从上述方程中略去 $v\Delta t$ 项，求得校正值为：

$$\Delta t = \frac{S - Vt_0 + vt_0}{V} \tag{1.29}$$

重复这一过程，得到如下的迭代公式：

$$t_{k+1} = \frac{S + vt_k}{V}, \qquad k = 0, 1, 2, \cdots \tag{1.30}$$

Zeno 悖论中表述的逼近过程正是这种迭代过程。当 $k \to \infty$ 时，t_k 逐步逼近人追赶龟所需要的时间 t^*。

例 12. *开方法例子。*

给定 $a > 0$，求开方值 $\sqrt{a}$ 的问题就是要解方程：

$$x^2 - a = 0 \tag{1.31}$$

设给定某个预报值 x_0，希望借助于某种简单方法确定校正量 Δx，使校正值 $x_1 = x_0 + \Delta x$ 能够比较准确地满足所给方程，即使

$$(x_0 + \Delta x)^2 \approx a \tag{1.32}$$

成立，设校正值 Δx 是个小量，舍去上式中的高阶小量 $(\Delta x)^2$，令

$$x_0^2 + 2x_0\Delta x = a \tag{1.33}$$

从中解出 Δx，可得校正值

$$\Delta x = \frac{1}{2}(\frac{a}{x_0} - x_0) \tag{1.34}$$

反复实施这种预报校正方法，即可求出开方公式：

$$x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k}) \quad k = 1, 2, 3, \cdots \tag{1.35}$$

直到 $x_{k+1} - x_k < \varepsilon(\varepsilon$ 为给定阈值)，此时 x_{k+1} 即为所求的根。
当 $a = 2, x_0 = 1, \varepsilon = 10^{-6}$ 时，计算结果如下：

k	x_k
0	1
1	1.500000000
2	1.416666667
3	1.414215686
4	1.414213562
5	1.414213562

表 1.1: 开方法迭代过程

求得 $\sqrt{2} \approx 1.414214$。

例 13. *求倒数的迭代算法*

给定一个值 $a(a \neq 0)$，不用除法来计算倒数值 $x^* = 1/a$。

设给定某个预报值 $x_0 \approx 1/a$，则有 $ax_0 \approx 1$，因此有

$$\overline{x}_0 = ax_0^2 \approx \frac{1}{a} \qquad \text{(两边同时乘以 } x_0\text{)} \tag{1.36}$$

由此可见，$\overline{x}_0$ 是与 x_0 相伴的另一个 $1/a$ 的近似值，并且两者位于 x^* 的同一侧。

考虑到 $ax_0 \approx 1$，我们有

$$\overline{x}_0 - x_0 = ax_0^2 - x_0 = ax_0(x_0 - \frac{1}{a}) \tag{1.37}$$

$$\approx x_0 - \frac{1}{a} \tag{1.38}$$

$$= x_0 - x^* \tag{1.39}$$

因此有：

$$x^* \approx 2x_0 - \overline{x}_0 \tag{1.40}$$

从而获得如下的迭代公式：

$$x_{k+1} = 2x_k - ax_k^2, \qquad k = 0, 1, 2, \cdots \tag{1.41}$$

1.4.3 化粗为精的松弛技术

在实际计算中常常可以获得目标值 F^* 的两个相伴的近似值 F_0 与 F_1 ，将它们加工成更高精度的结果的方法之一就是取两者的某种加权平均作为改进值：

$$F^{'} = (1 - \omega)F_0 + \omega F_1 \tag{1.42}$$

$$= F_0 + \omega(F_1 - F_0) \tag{1.43}$$

即通过适当选取权系数 ω 来调整校正量 $\omega(F_1 - F_0)$，以加工得到更高精度的 $F^{'}$，这种基于校正量的调整与松动的方法通常称为松弛技术。

超松弛

有一种情况：所提供的一对近似值与 F_1 和 F_0 有优劣之分，譬如 F_1 优而 F_0 劣，这时所采用如下松弛方式为：

$$F^{'} = (1 + \omega)F_1 - \omega F_0 \qquad \omega > 0 \tag{1.44}$$

即在松弛过程中张扬 F_1 的优势而抑制 F_0 的劣势，这种设计策略称作外推松弛技术，简称超松弛。

总之，超松弛的设计机理是优劣互补，化粗为精。松弛技术的关键在于松弛因子的选取，而这往往是相当困难的。

例 14. *Zeno 算法*

在考察 Zeno 算法，按照校正公式 (1.30)，计算 t_1 得到

$$t_1 = \frac{S + vt_0}{V} \tag{1.45}$$

据此有 $Vt_1 - vt_0 = S$，将式两端同时处以 $V - v$，得到如下等式

$$\frac{V}{V-v}t_1 - \frac{v}{V-v}t_0 = \frac{S}{V-v} = t^* \tag{1.46}$$

由此可见，精确解 t^* 等于任意给定的预报值 t_0 和它的校正值 t_1 两者的加权平均

$$t^* = (1+\omega)t_1 - \omega t_0, \qquad \omega = \frac{v}{V-v} \tag{1.47}$$

从上式可以看出，给定任意一对迭代值 t_0 和 t_1，按照某种方式进行松弛，结果总可以得到所给方程的精确解 t^*，这种加工效果是非常有趣的。

例 15. 割圆术例子

刘徽 (约公元 225 年-295 年，魏晋期间)，是中国数学史上非常伟大的数学家之一。他提出了 “割圆术”，这是将圆周用内接或外切正多边形穷竭的一种求圆面积和圆周长的方法。他用割圆术，从直径为 2 尺的圆内接正六边形开始割圆，依次得正 12 边形、正 24 边形，$\cdots$，割得越细，正多边形面积和圆面积之差越小。他计算了 3072 边形面积，得到 $\pi \approx 3927/1250 = 3.1416$。

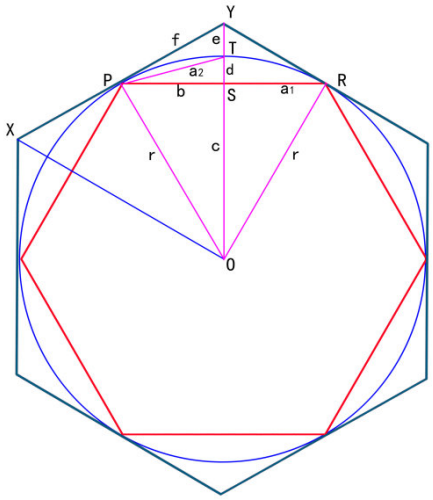


图 1.12: 割圆术

他发现利用 $S_{96} = 313\frac{584}{625}$ 与 $S_{192} = 314\frac{64}{625}$ 两个粗糙的数据进行松弛，可以得到高

精度的值 S_{3072}，加工过程如下：

$$S^* \approx S_{192} + \frac{36}{105}(S_{192} - S_{96}) \tag{1.48}$$

$$= 314\frac{64}{625} + \frac{36}{105} \times \left(314\frac{64}{625} - 313\frac{584}{625}\right) \tag{1.49}$$

$$= 314\frac{64}{625} + \frac{36}{105} \times \frac{105}{625} \tag{1.50}$$

$$= 314\frac{4}{25} \tag{1.51}$$

$$= S_{3072} \tag{1.52}$$

1.5 误差

1.5.1 误差的来源

- 模型误差 (描述误差)。反映实际问题有关量之间的计算公式 (数学模型) 通常是近似的；
- 观测误差。数学模型中包含的某些参数是通过观测得到的；
- 截断误差 (方法误差)。数值方法精确解与待求解模型的理论分析解之间的差异；
- 舍入误差。由于计算机表示浮点数采用的是有限字长；这样在计算机中表示的原始输入数据、中间计算数据、以及最终输出结果必然产生误差，称此类误差为舍入误差；
- 初值误差。由于初始值选取不合理所造成的最终结果的差异；

在计算方法中不研究前两类误差，总是假定数学模型是正确合理的反映了客观实际问题。

1.5.2 误差的度量

定义 1. 绝对误差定义 设 x 是某个量的准确值，x^* 是其近似值，则两者的差值称为近似值 x^* 的绝对误差，简称误差。

$$e(x^*) \triangleq x - x^* \tag{1.53}$$

在不引起混淆的情况下，可以简记 $e(x^*)$ 为 e。

定义 2. 绝对误差限 如果存在正数 ε，使得有绝对误差

$$|e(x^*)| = |x - x^*| \leq \varepsilon \tag{1.54}$$

则称 ε 为 x^* 近似 x 的一个绝对误差限。

$$x \in [x^* - \varepsilon, x^* + \varepsilon], \quad x = x^* \pm \varepsilon \tag{1.55}$$

绝对误差限虽然能够刻画对同一真值不同近似的好坏，但它不能刻画对不同真值近似程度的好坏。

定义 3. 相对误差定义 设 x^* 是对准确值 $x(\neq 0)$ 的一个近似，则称：

$$e_r(x^*) \triangleq \frac{e(x^*)}{x} = \frac{x - x^*}{x} \approx \frac{e(x^*)}{x^*} \tag{1.56}$$

为 x^* 近似 x 的相对误差，不引起混淆时简记 $e_r(x^*)$ 为 e_r。

定义 4. 相对误差限 数值 $|e_r|$ 的上限，记为 ε_r，相对误差限也可通过 $\varepsilon_r = \frac{\varepsilon}{|x^*|}$ 来计算。

为了规定一种近似数的表示法，使得用它表示的近似数自身就指示出其误差的大小。这里介绍有效数字和有效数的概念。

1.6 有效数字

定义 5. 有效数字 设 x 的近似值 x^* 有如下标准形式：

$$x^* = \pm 10^m \times \underline{0.x_1x_2x_3\cdots x_nx_{n+1}\cdots x_p} \tag{1.57}$$

其中 m 为整数，$\{x_i\} \subset \{0, 1, 2, 3, \cdots 9\}$ 且 $x_1 \neq 0, p \geq n$。如果有

$$|e| = |x - x^*| \leq \frac{1}{2} \times 10^{m-n} \tag{1.58}$$

则称 x^* 为 x 的具有 n 位有效数字的近似数，或称 x^* 准确到 10^{m-n} 位，其中数字 $x_1, x_2, x_3, \cdots, x_n$ 分别被称为 x^* 的第 $1, 2, 3, \cdots, n$ 个有效数字。

定义 6. 有效数 当 x^* 准确到末位，即 $n = p$，则称 x^* 为有效数。

例 16. $x = \pi, x_1^* = 3.141, x_2^* = 3.142, x_3^* = 3.1416, x_4^* = \frac{22}{7}, x_5^* = \frac{355}{113}$，分别计算 $x_1^*, x_2^*, x_3^*, x_4^*, x_5^*$ 分别具有几位有效数字，是否为有效数。

解：这里取 $\pi = 3.14159265358979323846\cdots$。因此有：

$$|x - x_1^*| = 0.00059\cdots \leq 0.005 = \frac{1}{2} \cdot 10^{1-3} \tag{1.59}$$

$$|x - x_2^*| = 0.00040\cdots \leq 0.0005 = \frac{1}{2} \cdot 10^{1-4} \tag{1.60}$$

$$|x - x_3^*| = 0.0000073\cdots \leq 0.00005 = \frac{1}{2} \cdot 10^{1-5} \tag{1.61}$$

$$|x - x_4^*| = 0.0012\cdots \leq 0.005 = \frac{1}{2} \cdot 10^{1-3} \tag{1.62}$$

$$|x - x_5^*| = 0.00000026\cdots \leq 0.0000005 = \frac{1}{2} \cdot 10^{1-7} \tag{1.63}$$

因此，x_1^* 具有 3 位有效数字，为非有效数；x_2^* 具有 4 位有效数字，是有效数；x_3^* 具有 5 位有效数字，是有效数；x_4^* 具有 3 位有效数字，不是有效数；x_5^* 具有 7 位有效数字，也不是有效数。

- 有效数的误差限是末位数单位的一半，可见有效数本身就体现了误差界；
- 对真值进行四舍五入得到有效数；
- 准确数字有无穷多位有效数字；
- 从实验仪器所读的近似数 (最后一位是估计位) 不是有效数，估计最后一位是为了确保对最后一位进行四舍五入得到有效数；

1.7 误差度量之间的联系

1. 绝对误差与相对误差

$$\frac{e(x^*)}{x} = e_r(x^*) \tag{1.64}$$

2. 绝对误差与有效数字

$$|e(x^*)| \leq \frac{1}{2} \cdot 10^{m-n} \tag{1.65}$$

3. 相对误差与有效数字

定理 1. *若 x^* 具有 n 位有效数字，则相对误差为 $|e_r| \leq \frac{1}{2x_1} \times 10^{1-n}$*

定理 2. *若相对误差为 $|e_r| \leq \frac{1}{2(x_1+1)} \times 10^{1-n}$，则 x^* 至少具有 n 位有效数字。*

该定理实质上给出了一种求相对误差限的方法。仅从 $|e_r| \leq \frac{1}{2x_1} \times 10^{1-n}$ 并不能保证 x^* 一定具有 n 位有效数字。

1.8 算法选用原则

- 简化计算步骤以减少运算次数；
- 避免大数 “吃掉” 小数；
- 尽量避免相近的数相减；
- 避免绝对值很小的数作为分母，防止出现溢出；
- 选用数值稳定性好的算法；

定义 7 (算法稳定性). 一个算法，如果在运算过程中舍入误差在一定条件下能够得到控制，或者舍入误差的增长不影响产生可靠的结果，则称该算法是数值稳定的，否则称其为数值不稳定。

例 17. 计算如下积分近似值。

$$I_n = \int_0^1 \frac{x^n}{x+5} dx \tag{1.66}$$

方案一：

$$I_0 = \int_0^1 \frac{1}{x+5} dx = \ln \frac{6}{5} \approx 0.1823 \tag{1.67}$$

$$I_n = \frac{1}{n} - 5I_{n-1} \quad (n = 1, 2, 3, \cdots) \tag{1.68}$$

方案二：

$$\frac{1}{6(n+1)} = \int_0^1 \frac{x^n}{6} dx \leq I_n \leq \int_0^1 \frac{x^n}{5} dx = \frac{1}{5(n+1)} \tag{1.69}$$

$$I_{n-1} = \frac{1}{5n} - \frac{1}{5} I_n \quad (n = n-1, n-2, n-3, \cdots, 1) \tag{1.70}$$

1.9 本章小结

本章主要介绍了算法设计的三种基本思想：即缩减技术、校正技术、松弛技术，并配合例子进行了阐述，这些思想是算法设计的瑰宝。

参考文献

[1] 靳天飞，杜忠友等. *计算方法*. 北京：清华大学出版社, 2010.

[2] 李桂成. *计算方法 (第 3 版)*. 北京：电子工业出版社. 2019.8.

[3] 安妮 • 戈林鲍姆. *数值方法：设计、分析和算法实现*. 北京：机械工业出版社, 2016.4.