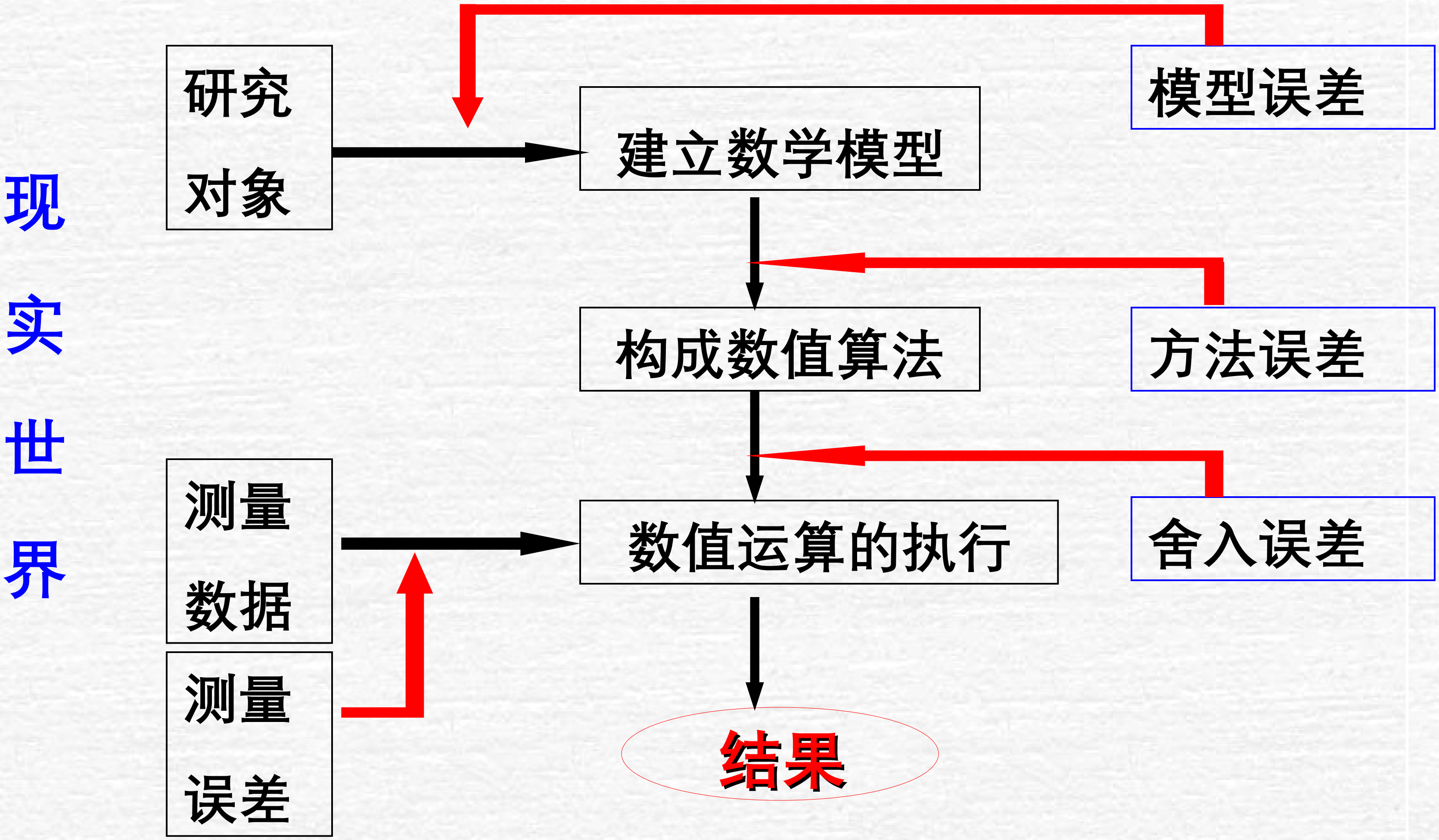
1. 3. 误差的种类及来源



以计算机为工具解决实际问题需经历三个过程：

I. 总体设计 (含数学模型的建立与模型细化等)

II. 详细设计 (主要是算法设计), 包括：

(1) 连续系统的离散化；

(2) 离散型方程的数值求解 .

I. 实验验证

(1) 模型误差

在建立数学模型过程中，要将复杂的现象抽象归结为数学模型，往往要忽略一些次要因素的影响，而对问题作一些简化，因此和实际问题有一定的区别。

(2) 观测误差

在建模和具体运算过程中所用的数据往往是通过观察和测量得到的，由于精度的限制，这些数据一般是近似的，即有误差。

(3) 截断误差

由于计算机只能完成有限次算术运算和逻辑运算，因此要将有些需用极限或无穷过程进行的运算有限化，对无穷过程进行截断，这就带来误差。

(4) 舍入误差

在数值计算过程中还会遇到无穷小数，因计算机受到机器字长的限制，它所能表示的数据只能有一定的有限位数，如按四舍五入规则取有限位数，由此引起的误差。

截断误差

如：

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

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

Taylor 展开

$$\ln(1+x) = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \cdots$$

若将前若干项的部分和作为函数值的近似公式，由于以后各项都舍弃了，自然产生了误差。

舍入误差

$$\pi = 3.14159265\cdots \qquad \pi \approx 3.1415927$$

$$\sqrt{2} = 1.414213562\cdots \qquad \sqrt{2} \approx 1.4142136$$

$$\frac{1}{3!} = \frac{1}{6} = 0.166666666\cdots \qquad \frac{1}{3!} \approx 0.16666667$$

数学模型一旦建立，进入具体计算时所考虑和分析的就是**截断误差和舍入误差**.

2. 误差与有效数字

◆ 绝对误差

$e^{*} = x - x^{*}$　其中 x^{*} 为精确值，x 为 x^{*} 的近似值。

$|e^{*}|$的上限记为 ε^{*}，称为绝对误差限。

工程上常记为 $x^{*} = x\varepsilon\square^{*}$

例如：$\int_0^1 e^{-x^2}dx = 0.743 \pm 0.006$

◆ 相对误差

$$e_r^{*} = \frac{e^{*}}{x^{*}}$$

x 的相对误差上限定义为 $\varepsilon_r^{*} = \dfrac{\varepsilon^{*}}{|x|}$

◆ 有效数字

用科学计数法，记

$$x = \square 0.a_1a_2\cdots a_n \square 10^m$$

其中 $a_1 \neq 0$，a_n 的截取按四舍五入规则

若

$$|x - x^*| \leq 0.5 \times 10^{m-n}$$

则称 x 有 n 位有效数字，精确到10^{m-n} 。

例 4 已知近似值为：$e^* = 2.718\,28$,

精确值为：$e = 2.718\,281\,82\cdots$,

求e^*的绝对误差、相对误差

解： 绝对误差 $|\varepsilon| = |e^* - e| = 0.000\,001\,82\cdots$

$$\leq 0.000\,002$$

$$= 2\times 10^{-6}$$

相对误差

$$\varepsilon_r^* = \frac{\varepsilon}{|e^*|} = \frac{2\times 10^{-6}}{2.718\,28} = \frac{2\times 10^{-6}}{2.718\,28}$$

或

$$\varepsilon_r^* = \frac{\varepsilon}{|e|} \approx 0.71\times 10^{-6}$$

例 5 $\pi = 3.14159265358979 32\cdots\cdots;$

$\pi^* = 3.1415$

问：π^* 有几位有效数字？请证明你的结论。

证明：

$\because \pi^* = 0.31415 \times 10^1$,

$|\pi - \pi^*| < 0.0000926\ldots\ldots \le 0.0005 = 0.5 * 10^{-3}. = 0.5 \times 10^{1-4}$

$\therefore \pi^*$ **有 4 位有效数字，**

精确到小数点后第 3 位。

返回

§ 1.4 题型分析与小结 算法和计算量

数值算法是从给定的已知量出发，经过有限次四则运算及规定的运算顺序，最后求出未知量的数值解，这样构成的完整计算步骤称为算法。

数值算法有四个特点：

(1) 目的明确	算法必须有明确的目的，其条件和结论均应有清楚的规定
(2) 定义精确	对算法的每一步都必须有精确的定义
(3) 算法可执行	算法中的每一步操作都是可执行的
(4) 步骤有限	算法必须在有限步内能够完成解题过程

计算量：一个算法所需的乘除运算总次数，单位是 flop.

计算量是衡量算法好坏的一个重要标准。

例 1 求 $Ax=b$， $Det(A)\neq 0$， $A=(a_{ij})_{20\times 20}$ 的计算量。

解：1. 用 Cramar 法则求解，总的计算量

$$N = ((n+1)(n-1)n!+n) \text{ 次}$$

当 $n=20$, $N\approx 9.7\times 10^{20}$ 次 .

以一台 10 亿 / 秒的计算机需约 3 万年 .

解：2. 使用 Gauss 消去法，

$n=20$， $N\approx 3060$ 次 $= O(n^3/3)$ 次 .

结论：分析算法的效率，选择算法非常重要。

例 2　计算 n 次多项式 $P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ 的值。

算法： 采用**秦九韶算法** (1247)（又称为 **Horner 算法** (1819)）

计算

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

$$\begin{cases} s_n = a_n \\ s_k = x s_{k+1} + a_k \quad (k = n-1, \cdots, 2, 1, 0) \\ P_n(x) = s_0 \end{cases}$$

说明： 算法需乘法 n 次，加法 n 次，存储单元 n+3 个。

上述秦九韶算法的结构是递归的，它通过一次式

$$s_k = x s_{k+1} + a_k$$

的反复计算，逐步降低多项式的次数，直到归结为零次式为止。

若以多项式的次数（或项数）定义为求值问题的规模，

秦九韶算法的特点： 在递归计算的过程中问题的规模逐次减 1

算法

1. 输入多项式系数 $a_0, a_1, \cdots, a_n$ 及 x；

2. 迭代计算

$$\begin{cases} s_n = a_n \\ s_k = x s_{k+1} + a_k \quad (k = n-1, \cdots, 2, 1, 0) \\ P_n(x) = s_0 \end{cases}$$

3. 输出结果

程序 QinJiushao.m

```
function s=QinJiushao(a,x)
   n=length(a);
   s=a(1);
   for k=2:n
        s=s*x+a(k);
   end
```

实例

求多项式

$$p(x) = x^5 - 3x^4 + 4x^2 - x + 1$$

在 $x=3$ 时的值．

运行及结果

```
>> a=[1 -3 0 4 -1 1];
>> s=QinJiushao(a,3)
s =
   34
```

注：编写程序时多项式系数按降幂排列．

例 3 计算 x^{255} 的复杂度

1. 计算 $X^{255} = \underbrace{X \times X \times X \times X}_{254\text{个乘法}} \times \ldots X$

工作量： N ＝ 254 flop

2. 计算 $X^{255} = X \times X^2 \times X^4 \times X^8 \times X^{16} \times X^{32} \times X^{64} \times X^{128}$

工作量： N ＝ 14 flop ， 8 个储存空间

返回

二．求解线性方程组

例 1 求解线性方程组解：

$$\begin{cases} x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3 = \frac{11}{6} \\ \frac{1}{2}x_1 + \frac{1}{3}x_2 + \frac{1}{4}x_3 = \frac{13}{12} \\ \frac{1}{3}x_1 + \frac{1}{4}x_2 + \frac{1}{5}x_3 = \frac{47}{60} \end{cases}$$

方程组的准确解为 $x_1=x_2=x_3=1$

若把方程组的系数舍入为两位有效数字，变为

$$\begin{cases} x_1 + 0.50x_2 + 0.33x_3 = 1.8 \\ 0.50x_1 + 0.33x_2 + 0.25x_3 = 1.1 \\ 0.33x_1 + 0.25x_2 + 0.20x_3 = 0.78 \end{cases}$$

则其解为 $x_1=-6.222\ldots$, $x_2=38.25\ \ldots$, $x_3=-33.65$,

可见这是一个病态问题（初始数据的的微小变化（扰动），导致计算结果的相对误差很大）与解法无关．

例 2: 求解 $x2 + (-10^9-1)\ x + 10^9 = 0$

解： $x_1 = 10^9,\ x_2 = 1.$

算法 A: $x_{1,2} = (- b \pm \mathrm{sqrt}\ (b^2 - 4ac)) / 2a$, □□□□□□□,

$- b = 10^9+1 = 0.10000000 \times 10^{10} + 0.0$

大数”吃掉”小数

$= 0.10000000 \times 10^{10}$

而 $b^2 - 4ac \approx b^2$, $\mathrm{sqrt}\ (b^2 - 4ac) \approx |b|$

□□ $x_1 = (- b + |b|)/2 = 0.10000000 \times 10^{10}$,

$x_2 = (- b - |b|)/2 = 0$

算法 B: $x_1 = (- b - \mathrm{sign}(b) \times \mathrm{sqrt}(b^2 - 4ac)) / 2a$

$x_2 = c/ax_1$

□ $x_1 = 0.10000000 \times 10^{10} = 10^9$

$x_2 = 0.10000000 \times 10^1 = 1$

算法 A 不稳定；算法 B 稳定，算法 B 准确

结论：良态问题选择稳定算法，才能得到满意解

例 3 已知$e = 2.718\,281\,82\cdots$，其近似值为$e^* = 2.718\,28$，求e^*的绝对误差限ε和相对误差限ε_r^*.

解： 绝对误差 $E = e^* - e$

$$= -0.000\,001\,82\cdots$$

$$|E| = 0.000\,001\,82\cdots \le 0.000\,002 = 2 \times 10^{-6}$$

$$\varepsilon = 2 \times 10^{-6}$$

$$\varepsilon_r^* = \frac{\varepsilon}{|e^*|} = \frac{2 \times 10^{-6}}{2.718\,28}$$

ε和ε_r^*并不是唯一的

$$= \frac{2 \times 10^{-6}}{2.718\,28} \approx 0.71 \times 10^{-6}$$

例 4 若π经四舍五入取小数点后3,5,7位数的近似值，求绝对误差限ε.

解：$\pi = 3.141\,592\,65\cdots$ $\quad |\pi - \pi^*| \le \varepsilon$

$\pi^* = 3.142 \qquad 0.000\,407\cdots \qquad \le 0.5\times 10^{-3}$

$\pi^* = 3.141\,59 \qquad 0.000\,002\,65\cdots \qquad \le 0.5\times 10^{-5}$

$\pi^* = 3.141\,592\,7 \qquad 0.000\,000\,04\cdots \qquad \le 0.5\times 10^{-7}$

可见，经四舍五入取近似值，其绝对误差限将不超过其末位数字的半个单位

例 5 ：为使π*的相对误差小于 0.001%, 至少应取几位有效数字？

解：假设 π* 取到 n 位有效数字，则其相对误差上限为

$$\varepsilon_r^* \le \frac{1}{2a_1} \times 10^{-n+1}$$

要保证其相对误差小于 0.001%，只要保证其上限满足

$$\varepsilon_r^* \le \frac{1}{2a_1} \times 10^{-n+1} < 0.001\%$$

已知 $a_1 = 3$，则从以上不等式可解得 $n > 6 - \log 6$，即 $n \ge 6$，应取 $\pi^* = 3.14159$。

例：计算 $I_n = \frac{1}{e}\int_0^1 x^n e^x dx$ ， $n = 0, 1, 2, \ldots\ldots$

公式一 $I_n = 1 - n\, I_{n-1}$

注意此公式精确成立

：$I_0 = \frac{1}{e}\int_0^1 e^x dx = 1 - \frac{1}{e} \approx 0.63212056 \overset{记为}{=\!=} I_0^*$

则初始误差 $|E_0| = |I_0 - I_0^*| < 0.5 \times 10^{-8}$

$$\frac{1}{e}\int_0^1 x^n \cdot e^0\, dx < I_n < \frac{1}{e}\int_0^1 x^n \cdot e^1\, dx \qquad \therefore \frac{1}{e(n+1)} < I_n < \frac{1}{n+1}$$

$I_1^* = 1 - 1 \cdot I_0^* = 0.36787944$

…… …… …… ……

$I_{10}^* = 1 - 10 \cdot I_9^* = 0.08812800$

$I_{11}^* = 1 - 11 \cdot I_{10}^* = 0.03059200$

$I_{12}^* = 1 - 12 \cdot I_{11}^* = 0.63289600$?

$I_{13}^* = 1 - 13 \cdot I_{12}^* = -7.2276480$??

$I_{14}^* = 1 - 14 \cdot I_{13}^* = 94.959424$? !

$I_{15}^* = 1 - 15 \cdot I_{14}^* = -1423.3914$! !

考察第 n 步的误差 $|E_n|$

$$|E_n| = |I_n - I_n^*| = |(1 - nI_{n-1}) - (1 - nI_{n-1}^*)| = n\,|E_{n-1}| = \cdots = n!\,|E_0|$$

可见初始的小扰动 $|E_0| < 0.5\times 10^{-8}$ 迅速积累误差呈递增走势。

造成这种情况的是不稳定的算法 /* unstable algorithm */

我们有责任改变。

公式二： $I_n = 1 - n\,I_{n-1} \quad \Rightarrow \quad I_{n-1} = \dfrac{1}{n}(1 - I_n)$

方法：先估计一个 I_N，再反推要求的 I_n ($n \ll N$) 。

注意此公式与公式一在理论上等价。

$$\because \quad \frac{1}{e(N+1)} < I_N < \frac{1}{N+1}$$

可取 $I_N^* = \dfrac{1}{2}\left[\dfrac{1}{e(N+1)} + \dfrac{1}{N+1}\right] \approx I_N$

当 $N \to +\infty$ 时, $|E_N| = |I_N - I_N^*| \to 0$

$$\text{取}\quad I_{15}^{*}=\frac{1}{2}\left[\frac{1}{e\cdot 16}+\frac{1}{16}\right]\approx 0.042746233$$

$$\Rightarrow I_{14}^{*}=\frac{1}{15}(1-I_{15}^{*})\approx 0.063816918$$

$$I_{13}^{*}=\frac{1}{14}(1-I_{14}^{*})\approx 0.066870220$$

$$I_{12}^{*}=\frac{1}{13}(1-I_{13}^{*})\approx 0.071779214$$

$$I_{11}^{*}=\frac{1}{12}(1-I_{12}^{*})\approx 0.077351732$$

$$I_{10}^{*}=\frac{1}{11}(1-I_{11}^{*})\approx 0.083877115$$

$$\vdots$$

$$I_{1}^{*}=\frac{1}{2}(1-I_{2}^{*})\approx 0.36787944$$

$$I_{0}^{*}=\frac{1}{1}(1-I_{1}^{*})\approx 0.63212056$$

考察反推一步的误差：

$$|E_{N-1}| = \left|\frac{1}{N}(1-I_N) - \frac{1}{N}(1-I_N^*)\right| = \frac{1}{N}|E_N|$$

以此类推，对 $n < N$ 有：

$$|E_n| = \frac{1}{N(N-1)\dots(n+1)}|E_N|$$

误差逐步递减，这样的算法称为稳定的算法 /* stable algorithm */

在我们今后的讨论中，误差将不可回避，

算法的稳定性会是一个非常重要的话题。