第 1 章

数值分析的基本概念

1.1 数值算法的研究对象

1. 数值算法的研究对象

在解决现代工程技术问题时，常常需要首先建立问题的数学模型，然后合理地设计问题的算法，并通过计算机计算，最后获得问题的解答。本课程正以此为基本研究对象，是研究算法的学科。为使读者对本课程的基本目的和内容有一个大致的了解，我们先讨论下列两组数学模型的计算问题。

例 1.1 (1) 求解线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，其中 $\boldsymbol{A}$ 为 3 阶可逆方阵，$\boldsymbol{x}=(x_1,x_2,x_3)^{\mathrm{T}}$；

(2) 求代数方程 $3x^2+8x-3=0$ 在$[0,1]$上的根 x^*；

(3) 已知 $y=P(x)$为$[x_0,x_1]$上的直线，满足 $P(x_0)=y_0$，$P(x_1)=y_1$，$\bar{x}\in(x_0,x_1)$，求 $P(\bar{x})$；

(4) 计算定积分 $I=\int_a^b \frac{1}{x}\mathrm{d}x$ $(1<a<b)$；

(5) 解常微分方程初值问题 $\begin{cases} y'=x+y, \\ y(0)=0。 \end{cases}$

解 应用微积分学和线性代数的基本知识，不难求得问题的解。

(1) 由线性代数克莱姆(Cramer)法则，得 $x_1=\frac{D_1}{D}$，$x_2=\frac{D_2}{D}$，$x_3=\frac{D_3}{D}$，其中 $D=|\boldsymbol{A}|$，D_j 为由 $\boldsymbol{b}$ 置换 D 的第 j 列所得。所以问题归结为计算 4 个 3 阶行列式。

(2) 利用初等求根公式容易得到 $x^*=\frac{1}{3}$。

(3) 由解析几何知识，得 $P(\bar{x})=y_0+\frac{y_1-y_0}{x_1-x_0}(\bar{x}-x_0)$。

(4) 根据积分公式，得 $I=\ln\frac{b}{a}$。

(5) 根据线性常微分方程求解公式，得 $y(x)=-x-1+\mathrm{e}^x$。

例 1.2 (1) 求解线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，其中 $\boldsymbol{A}$ 为 30 阶可逆方阵，$\boldsymbol{x}=(x_1,x_2,\cdots,x_{30})^{\mathrm{T}}$；

(2) 求超越方程 $x\mathrm{e}^x=1$ 在$[0,1]$上的根 x^*；

(3) 已知 $y=f(x)$为$[x_0,x_1]$上的函数，满足 $f(x_0)=y_0$，$f(x_1)=y_1$，$\bar{x}\in(x_0,x_1)$，求 $f(\bar{x})$；

(4) 计算定积分 $I=\int_a^b \frac{1}{\ln x}\mathrm{d}x$ $(1<a<b)$;

(5) 解常微分方程初值问题 $\begin{cases} y'=x+y^2, \\ y(0)=0。 \end{cases}$

解 例 1.2 与例 1.1 初步看来“差不多”,是否也容易解决呢? 试讨论如下。

(1) 由克莱姆法则需要计算 31 个 30 阶行列式,如果按照行列式的展开式计算,由于每个 30 阶行列式有 30! 项,每一项为 30 个元素乘积,共需 30! $\times 29\times 31 \approx 2\times 10^{35}$ 次乘法,计算量非常大。

(2) 设 $f(x)=x\mathrm{e}^x-1$,由于 $f(0)<0$,$f(1)>0$,根据连续函数介值定理,方程在[0,1]上的根 x^* 存在,但无法求得 x^* 的解析形式,只能想办法求其近似值。

(3) 此问题看似无理,但又实实在在,借用例 1.1 相应问题的结果求解,$f(\bar{x})\approx P(\bar{x})=y_0+\frac{y_1-y_0}{x_1-x_0}(\bar{x}-x_0)$。

(4) 高等数学的牛顿-莱布尼茨公式对该问题失效,因为无法找到被积函数 $f(x)=\frac{1}{\ln x}$ 的原函数,我们可以考虑一些近似求解方法。

(5) 线性常微分方程比较容易得到解析解,而非线性常微分方程一般都没有解析解,主要依靠数值解法求取近似解。该问题与例 1.1 相应问题似乎只有“一点点”差别,但它没有解析解,只能依赖数值方法。

上述例子提示我们,在高等数学中学习的算法只能求解一些比较简单特殊的数学模型,工程实践中遇到的许多数学问题或者由于计算量太大或者由于没有解析方法,不能有效地使用手工计算,需要借助于计算机的计算能力。同时,我们必须认识到:第一,计算机的认识能力是有限的,这就要求我们设计其能接受的“算法”。例如,我们考虑用 C 语言解例 1.2(4),但 C 语言并不能识别“积分”这个数学概念,需要我们预先将积分转化为初等运算和初等函数构成的计算问题(在第 5 章我们将有多种方法求解此题)。第二,计算机的计算能力也是有限的,这就要求我们为其设计“好的算法”。例如对于例 1.2(1)的计算量,即使用 2013 年全球最快的超级计算机“天河二号”计算,每秒能做 33.86 千万亿次乘法也需要 591 亿年! 这就要求我们设计出更有效的算法。事实上,使用第 2 章的方法,我们用普通的计算机可以在 1s 内求解此题。

本课程将会给出解决例 1.2 这类问题的算法。这里必须指出的是,实际问题远比例 1.2 列举的问题复杂得多,本课程只是讨论一些常用的基本数值算法。实际工程中,不同的算法适合不同的问题,所以我们需要更有针对性的算法。课程中所介绍的数值计算的一些基本概念和基本思想是具有普遍意义的。读者在学习中应将着重点放在各种算法的基本思想方法上,而不是死记硬背一些计算公式。

2. 数值算法的特点

对于给定的问题和设备,一个**算法**(algorithm)是用该设备可理解的语言表示的,是对解决这个问题的一种方法的精确刻画。这里的设备主要指计算机软件系统,所以也称计算机算法。对于算法刻画的要求取决于设备的语言能力。如果使用汇编语言之类的低级语言,那么对算法的描述需要极为详尽,包括变量地址、算术运算方式都要表达详尽;如果使用

C、FORTRAN 等高级语言，那么算术运算和初等函数等就成为设备可理解的语言，所以其描述可以粗略一些；而如果使用 MATLAB 这类专业化数学软件，那么就连积分、微分方程等的计算也不必详述。本课程讨论的大部分算法是建立在 C、FORTRAN 等高级语言这个层次上，也有一些算法建立在 MATLAB 软件层次上。

计算机算法主要包含数值算法、非数值算法和软计算方法三类。数值算法主要指与连续数学模型有关的算法，如数值线性代数、方程求解、数值逼近、数值微积分、微分方程数值解和最优化计算方法等；非数值算法主要指与离散数学模型有关的算法，如排序、搜索、分类、图论算法等；软计算方法是近来发展的不确定性算法的总称，包括随机模拟、神经网络计算、模糊逻辑、遗传算法、模拟退火算法和 DNA 算法等。

本课程主要研究的是数值算法，它是计算机算法内容最为丰富、也是最基本的一部分内容。数值算法具有下列几个显著特点。

(1) 有穷性

由于计算机编码的离散性本质，数值算法和所有计算机算法一样，必须在有限步完成，同时其处理对象的规模也是有限的。

(2) 数值性

由于其研究对象的特点，数值算法所涉及的数据类型主要是数值型，特别是以实数型居多。字符类型数据在数值算法中很少出现，整数型也比较少，数值算法中使用的大部分变量都是浮点实数，而且通常都是双精度的。

(3) 近似性

近似性是数值算法最显著的特点，这是由其研究对象和计算机的离散性本质之间的矛盾决定的。与连续数学模型相关的很多概念(如微分、积分)都是数学上的无穷极限，而计算机算法无法确切地表达和执行这些概念，只能得出某种程度的有穷近似，这就决定了它们之间必然存在误差。

1.2　误差分析的概念

在数值计算中，误差往往是不可避免的，那么怎样估计计算误差，判断计算结果的有效性，就成为数值分析极其重要的内容。

1. 误差限和有效数字

首先给出一些误差分析术语的定义。这些术语常常被人们不加定义地使用，但在不同的文献中的含义却有细微差别，我们采用了教科书中比较标准的定义。

定义 1.1(误差和相对误差)　设 x^* 是某量的准确值，x 是 x^* 的近似值，称 $\delta(x)=x^*-x$ 为 x 的**误差**(error)或**绝对误差**(absolute error)。在数值计算问题中，x^* 是未知的，从而 $\delta(x)$ 也无法精确得到，但我们往往可以得到 $\delta(x)$ 的绝对值上界，即 $|x^*-x|\leqslant\varepsilon$，称 ε 为 x 的**误差限**(error bound)或**精度**(accuracy)。应用中常记为 $x\pm\varepsilon$。误差与准确值的比值 $\delta_r(x)=(x^*-x)/x^*$ 称为 x 的**相对误差**(relative error)。如果 $|(x^*-x)/x^*|\leqslant\varepsilon_r$，称 ε_r 为 x 的**相对误差限**(relative error bound)或**相对精度**(relative accuracy)。当 ε_r 很小时，$\varepsilon_r\approx\varepsilon/|x|$。

定义 1.2(准确位数和有效数字)　设 x^* 是某量的准确值，x 是 x^* 的近似值，即

$$x = \pm 0.a_1\ a_2 \cdots a_n \cdots \times 10^m, \tag{1.1}$$

其中,m 为整数,$a_1 \sim a_n$ 为 0～9 中一个数字且 $a_1 \neq 0$。如果

$$|x^* - x| \leqslant 0.5 \times 10^{-k}, \tag{1.2}$$

即 x 的误差不超过 10^{-k} 位的半个单位,则称近似数 x **准确到** 10^{-k} **位**,并说 x 有 $m+k$ 位**有效数字**(significant digit)。

例 1.3 设圆周率 $\pi = 3.1415926\cdots$,求下列近似数的绝对误差、相对误差和有效数字。

(1) $x_1 = 3.14$;

(2) $x_2 = 3.141$;

(3) $x_3 = 3.142$;

(4) $x_4 = 3.1414$。

解 (1) $\pi - x_1 = 0.15926\cdots \times 10^{-2}$,$(\pi - x_1)/\pi = 0.5072\cdots \times 10^{-3}$,由于 $x_1 = 0.314 \times 10^1$,$|\pi - x_1| \leqslant 0.5 \times 10^{-2}$,从而 x_1 有 3 位有效数字;

(2) $\pi - x_2 = 0.5926\cdots \times 10^{-3}$,$(\pi - x_2)/\pi = 0.1887\cdots \times 10^{-3}$,由于 $x_2 = 0.3141 \times 10^1$,$|\pi - x_2| \leqslant 0.5 \times 10^{-2}$,从而 x_2 有 3 位有效数字;

(3) $\pi - x_3 = -0.4073\cdots \times 10^{-3}$,$(\pi - x_3)/\pi = -0.1296\cdots \times 10^{-3}$,由于 $x_3 = 0.3142 \times 10^1$,$|\pi - x_3| \leqslant 0.5 \times 10^{-3}$,从而 x_3 有 4 位有效数字;

(4) $\pi - x_4 = 0.1926\cdots \times 10^{-3}$,$(\pi - x_4)/\pi = -0.613\cdots \times 10^{-4}$,由于 $x_4 = 0.31414 \times 10^1$,$|\pi - x_4| \leqslant 0.5 \times 10^{-3}$,从而 x_4 有 4 位有效数字。

有效数字概念的通俗定义是这样的:设 x^* 是某量的准确值,x 是 x^* 的近似值,如果在从第一个非零数字开始的第 n 位进行四舍五入,x^* 和 x 的结果完全一致,则称 x 有 n 位有效数字。但这一定义在数值分析中无法应用,因为 x^* 是未知的。定义 1.2 是该通俗定义的抽象,其出发点为绝对误差限,而有效数字应该理解为计算精度的一种简略的表达。在例 1.3 中,对于 x_1, x_2, x_3,定义 1.2 与通俗定义一致,而 x_4 不一致,应该说定义 1.2 能更好地表达精度的好坏,因为按照通俗定义,x_4 只有 3 位有效数字,但实际上,x_4 的误差明显比 x_3 的误差小,是 π 更好的近似值,所以通俗定义是不合理的。

2. 截断误差与收敛性

截断误差也称方法误差,是数值算法设计中最主要考虑的误差问题。许多数学概念(如微分、积分等)都具有极限上的意义,不可能经过有限次算术运算计算出来,我们只能构造某种算法用有限次算术运算作近似替代,由此产生的误差称为该数值算法的**截断误差**(truncation error)。

例如,已知 x 在 0 附近,要计算指数函数 e^x 的值。由于 e^x 并不是算术运算,根据微分学的泰勒(Taylor)公式

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + R_n(x), \tag{1.3}$$

考虑到一方面计算多项式只用到算术运算,另一方面当 n 充分大时,余项 $R_n(x)$ 的数值很小,这样便可以用式(1.3)右边前面的 n 次多项式来近似替代 e^x,从而得到近似计算公式为

$$e^x \approx S_n(x) = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}, \tag{1.4}$$

其截断误差可以用余项公式

$$e^x - S_n(x) = R_n(x) = \frac{x^{n+1}}{(n+1)!}e^{\xi} \quad (\xi \text{在 } 0 \text{ 与 } x \text{ 之间}) \tag{1.5}$$

来进行分析。根据微分学理论，对于固定的 x，当 $n \to \infty$ 时，余项 $R_n(x) \to 0$，即 $S_n(x) \to e^x$。据此我们称算法(1.4)是**收敛**的。一种算法是收敛的，说明该算法总可以通过提高计算量使得截断误差任意小。

需要指出的是，大多数高级语言已经将指数函数、对数函数、三角函数等许多初等数学函数做成软件的标准库函数，所以我们编程时不需要再作处理。本质上，计算机只能作算术运算，高级语言中的数学函数正是根据式(1.4)这类算法构造的。

3. 舍入误差和数值稳定性

计算机中采用二进制实数系统，并且表示成规格化浮点形式：

$$\pm 2^m \times 0.\beta_1\beta_2\cdots\beta_t,$$

称为**机器数**。其中整数 m 称为**阶码**，用二进制数

$$m = \pm \alpha_1\alpha_2\cdots\alpha_s$$

表示，$\alpha_i = 0$ 或 $1(i = 1,2,\cdots,s)$。小数 $0.\beta_1\beta_2\cdots\beta_t$ 称为**尾数**，$\beta_1 = 1$，$\beta_j = 0$ 或 $1(j = 2,3,\cdots,t)$。正整数 t 称为**字长**。s 决定了机器数的绝对值范围，而 t 决定了机器数的表示精度。由于 s 和 t 都是有限的，所以机器数是离散的而非连续的。

机器数有单精度和双精度之分。通常，单精度为 32 位，双精度为 64 位，它们是正负号、阶码和尾数所占二进制的总长度(见图 1-1)。

图 1-1　计算机中数的表示

单精度机器数，$t = 23$，大约相当于十进制 7 位有效数字；双精度机器数，$t = 52$，大约相当于十进制 15 位有效数字。单精度机器数和双精度机器数的绝对值范围分别为 $2^{-128} \sim 2^{128}$(即 $2.9 \times 10^{-39} \sim 3.4 \times 10^{38}$)和 $2^{-1024} \sim 2^{1024}$(即 $5.56 \times 10^{-309} \sim 1.79 \times 10^{308}$)。低于该范围的机器数视为 0，高于该范围的机器数视为无穷大。

由于机器字长的限制而产生的误差称为**舍入误差**(rounding error 或者 round-off error)。十进制数进入计算机运算时先转换成二进制机器数，并对 t 位后的数作舍入处理，使得尾数为 t 位，因此一般都有舍入误差。两个二进制机器数作算术运算时，也要作类似的舍入处理，使得尾数为 t 位，从而也有舍入误差。二进制数计算结果输出时再转换成十进制数。所以说，舍入误差遵循的是二进制操作规律(见上机实验题 1)。但为了符合通常的习惯，在以后的讨论中我们仍然采用十进制进行误差分析，而忽略十进制与二进制的差异。

设函数 $y = f(x_1, x_2, \cdots, x_n)$ 是一个算法或模型，x_i^* 是变量 x_i 的准确值，而 $\tilde{x}_i$ 是变量 x_i 的近似值，$i = 1,2,\cdots,n$。如果 $\tilde{x}_i$ 的精度为 ε_i，且 f 的计算过程中没有新的误差产生，那

么计算结果$\tilde{y}=f(\tilde{x}_1,\tilde{x}_2,\cdots,\tilde{x}_n)$具有怎样的精度呢?

如果$f(x_1,x_2,\cdots,x_n)=\sum_{i=1}^{n}a_ix_i$(线性函数),那么

$$|\delta(\tilde{y})|=|f(x_1^*,x_2^*,\cdots,x_n^*)-f(\tilde{x}_1,\tilde{x}_2,\cdots,\tilde{x}_n)|\leqslant\sum_{i=1}^{n}|a_i|\varepsilon_i。\tag{1.6}$$

又如果$f(x_1,x_2,\cdots,x_n)$是非线性函数,此时对于误差传播的严格估计是很困难的,而且不等式估计结果往往过于保守,所以一般采用一次近似估计法。根据微分学理论,绝对误差为

$$\delta(\tilde{y})=f(x_1^*,x_2^*,\cdots,x_n^*)-f(\tilde{x}_1,\tilde{x}_2,\cdots,\tilde{x}_n)\approx\sum_{i=1}^{n}\left(\frac{\partial f}{\partial x_i}\right)^*\delta(\tilde{x}_i)。\tag{1.7}$$

相对误差为

$$\delta_r(\tilde{y})=\frac{\delta(\tilde{y})}{y^*}\approx\sum_{i=1}^{n}\left(\frac{\partial f}{\partial x_i}\right)^*\frac{x_i^*}{y^*}\delta_r(\tilde{x}_i)。\tag{1.8}$$

其中,$y^*=f(x_1^*,x_2^*,\cdots,x_n^*)$,$\left(\frac{\partial f}{\partial x_i}\right)^*=\frac{\partial f}{\partial x_i}(x_1^*,x_2^*,\cdots,x_n^*)$,误差分析中$x_i^*$,$y^*$也可用其近似值代替。根据式(1.7)、式(1.8),绝对误差传播主要取决于$\left(\frac{\partial f}{\partial x_i}\right)^*$,相对误差传播主要取决于$\left(\frac{\partial f}{\partial x_i}\right)^*\frac{x_i^*}{y^*}$($i=1,2,\cdots,n$)。

两个数之间的算术运算是二元函数,根据式(1.7)、式(1.8)得到

$$\delta(a\pm b)=\delta(a)\pm\delta(b),\quad\delta_r(a\pm b)=[a/(a\pm b)]\delta_r(a)+[b/(a\pm b)]\delta_r(b);\tag{1.9}$$

$$\delta(ab)\approx b\delta(a)+a\delta(b),\quad\delta_r(ab)\approx\delta_r(a)+\delta_r(b);\tag{1.10}$$

$$\delta(a/b)\approx(1/b)\delta(a)-(a/b^2)\delta(b),\quad\delta_r(a/b)\approx\delta_r(a)-\delta_r(b)。\tag{1.11}$$

式(1.9)表明,在作加减法时,应尽量避免两个相近的数相减,否则会使相对误差增大,导致有效数字的严重损失。式(1.11)表明,在作除法时,应尽量避免用绝对值很小的数作除数,否则会使绝对误差增大。

例 1.4 设有长为L、宽为D的矩形场地,现测得L的近似值$\tilde{L}=(120\pm0.2)$m,D的近似值$\tilde{D}=(90\pm0.2)$m。求该场地面积的误差限和相对误差限。

解 设场地面积的近似值为$\tilde{S}=\tilde{L}\tilde{D}$,由于$|\delta(\tilde{L})|=|\delta(\tilde{D})|=0.2$m,根据式(1.10)得

$$|\delta(\tilde{S})|\approx|\tilde{L}\delta(\tilde{D})+\tilde{D}\delta(\tilde{L})|\leqslant|\tilde{L}||\delta(\tilde{D})|+|\tilde{D}||\delta(\tilde{L})|=(120+90)\text{m}\times0.2\text{m}=42\text{m}^2,$$

$$|\delta_r(\tilde{S})|\leqslant\frac{|\delta(\tilde{S})|}{|\tilde{S}|}=\frac{42}{120\times90}=0.0039=0.39\%。$$

舍入误差的影响很大程度上取决于计算设备的能力。对于通常的问题,如果我们采用双精度,舍入误差的影响一般不会太明显。但是在某些情况下,舍入误差的影响可能是至关重要的。

例 1.5 计算积分

$$I_n=\int_0^1x^n\mathrm{e}^{x-1}\mathrm{d}x,\quad n=0,1,\cdots,20。\tag{1.12}$$

解 这 21 个积分当然可以各自求解,但计算量比较大。现在我们构造一种递推算法,可以在求得其中一个积分的基础上非常简单地推出其他 20 个积分的值。根据分部积分法得

$$I_n = x^n \mathrm{e}^{x-1} \Big|_0^1 - \int_0^1 n x^{n-1} \mathrm{e}^{x-1} \mathrm{d}x = 1 - nI_{n-1},$$

从而得到递推公式

$$I_n = 1 - nI_{n-1}, \quad n = 1,2,\cdots,20。 \tag{1.13}$$

由于 $I_0 = 1 - 1/\mathrm{e}$，我们可以利用式(1.13)计算 $I_1, I_2, \cdots, I_{20}$。双精度计算结果见表 1-1。

因为当 $0 \leqslant x \leqslant 1$ 时，有

$$x^n/\mathrm{e} \leqslant x^n \mathrm{e}^{x-1} \leqslant x^n,$$

所以

$$\frac{1}{(n+1)\mathrm{e}} \leqslant I_n \leqslant \frac{1}{n+1}。 \tag{1.14}$$

虽然 I_0 有相当高的精度，并且递推公式(1.13)没有任何截断误差，$I_{18} \sim I_{20}$ 的计算结果却明显有很大误差。现在我们将递推公式(1.13)略作改造，成为

$$I_{n-1} = (1 - I_n)/n, \quad n = 20,19,\cdots,1。 \tag{1.15}$$

首先我们根据式(1.14)，取 $I_{20} = 0.5(1/\mathrm{e}+1)/(20+1)$，按式(1.15)递推计算结果见表 1-1。可以看到，尽管 I_{20} 的精度不算很高，递推计算结果 $I_2 \sim I_0$ 却有相当高的精度。表 1-1 告诉我们递推公式(1.13)和式(1.15)尽管在理论上完全等价，其实际计算效果却有天壤之别。为此我们有必要分析一下两种算法中误差的传播情况。

表 1-1　例 1.5 的计算结果

n	算法(1.13)		算法(1.15)	
0	0.632121		0.632121	
1	0.367879		0.367879	
2	0.264241		0.264241	
3	0.207277		0.207277	
4	0.170893		0.170893	
5	0.145533		0.145533	
6	0.126802	↓	0.126802	↑
7	0.112384		0.112384	
8	0.100932		0.100932	
9	0.091612		0.091612	
10	0.083877		0.083877	
11	0.077352		0.077352	
12	0.071773		0.071773	
13	0.066948		0.066948	
14	0.062731		0.062732	
15	0.059034		0.059018	
16	0.055459		0.055719	
17	0.057192		0.052773	
18	−0.02945		0.050086	
19	1.55962		0.048372	
20	−30.1924		0.032569	

设 I_n 有误差 δ_n，在算法(1.13)中，有 $\delta_n = -n\delta_{n-1}$，这样 $\delta_{20} = (20!)\,\delta_0 = 2.43 \times 10^{18}\delta_0$，

虽然 δ_0 很小,但 δ_{20} 却很大。而对于算法(1.15),有 $\delta_{n-1}=-\delta_n/n$,这样 $\delta_0=\delta_{20}/(20!)=4.1103\times10^{-19}\delta_{20}$,可见误差 δ_{20} 会在计算过程中逐步缩小。

如果在一个数值算法中,舍入误差会在计算过程中恶性增大,我们就称该算法**数值不稳定**(numerically unstable),否则称其**数值稳定**(numerically stable)。例如,当我们用绝对值很小的数作除数,则算法就是数值不稳定的。

4. 数据误差和病态问题

工程问题的数学模型中的参数往往由实验和测量数据取得,所以不可避免地存在数据误差。正常情况下,这些误差对于解的影响是在允许范围内的,但对于某些问题,数据误差可能会产生严重影响。

例 1.6 解线性方程组

$$\begin{cases}\dfrac{49}{36}x_1+\dfrac{3}{4}x_2+\dfrac{21}{40}x_3=\dfrac{949}{360},\\[2ex]\dfrac{3}{4}x_1+\dfrac{61}{144}x_2+\dfrac{3}{10}x_3=\dfrac{1061}{720},\\[2ex]\dfrac{21}{40}x_1+\dfrac{3}{10}x_2+\dfrac{769}{3600}x_3=\dfrac{3739}{3600}。\end{cases}\tag{1.16}$$

解 容易验证 $x_1=x_2=x_3=1$ 是方程组(1.16)的唯一准确解。如果我们将系数保留 4 位有效数字,得到方程组

$$\begin{cases}1.361x_1+0.7500x_2+0.5250x_3=2.636,\\0.7500x_1+0.4236x_2+0.3000x_3=1.474,\\0.5250x_1+0.3000x_2+0.2136x_3=1.039。\end{cases}\tag{1.17}$$

计算结果为 $x_1=1.2203$,$x_2=-0.3084$,$x_3=2.2981$,与原方程组的解相比已经面目全非。

与例 1.5 不同的是,这里的误差与算法无关,因为这个解对于方程组(1.17)有很高的精度(读者很容易直接验证)。问题出在方程组(1.16) 本身对数据的误差极其敏感,尽管数据只有很小的变化,却导致解产生了很大的变化,我们称这类问题为**病态问题**(ill-posed problem)。

病态问题也可以用一次近似来作分析。考虑函数 $y=f(x_1,x_2,\cdots,x_n)$,x_i^* 是模型参数 x_i 的准确值,$\widetilde{x}_i$ 是近似值,$i=1,2,\cdots,n$,$y^*=f(x_1^*,x_2^*,\cdots,x_n^*)$ 是模型的准确解,$\widetilde{y}=f(\widetilde{x}_1,\widetilde{x}_2,\cdots,\widetilde{x}_n)$是近似模型的准确解。根据式(1.7)、式(1.8),$\left(\dfrac{\partial f}{\partial x_i}\right)^*$ 和 $\left(\dfrac{\partial f}{\partial x_i}\right)^*\dfrac{x_i^*}{y^*}$ 的绝对值反映了解对参数数据误差的敏感性程度,我们称 $\left|\left(\dfrac{\partial f}{\partial x_i}\right)^*\right|$ 和 $\left|\left(\dfrac{\partial f}{\partial x_i}\right)^*\dfrac{x_i^*}{y^*}\right|$ 为该问题的**条件数**(condition number)。条件数很大的数值问题称为**病态问题**。

1.3 数值算法设计的注意事项

1. 数值算法评价的基本原则

设计一个数值算法,主要是要处理好计算精度和计算速度两个问题。计算精度问题就

是计算结果的可靠性问题，一般来说，可靠的算法应该具有收敛性和数值稳定性。计算速度问题就是算法的效率问题，它包含计算量和存储量两个方面。时空两个方面的复杂性都会影响计算速度。通常影响计算速度最关键的因素是算法的收敛速度。通俗地讲，一个好的算法就是又准又快的算法。

2. 数值算法设计的一些注意事项

(1) 要保证算法具有收敛性和较高的收敛速度。

(2) 要保证算法具有数值稳定性。

(3) 小心处理病态问题。

(4) 注意影响收敛速度的细节。

一般来说加减法比乘法快，乘法比除法快。如使用 $x+x, x\cdot x, 0.5x$ 就分别比 $2x, x^2, x/2$ 的计算速度快。

例 1.7(秦九韶算法或 Horner 法)　考虑多项式 $a_0+a_1x+\cdots+a_nx^n$ 的算法设计。

解　由于计算机作加减法比乘法快得多，我们只考虑乘法的计算量。如果直接计算，需要

$$0+1+\cdots+n=n(n+1)/2$$

次乘法。

如果使用递推算法

$$t_0=1,\quad p_0=a_0,\quad t_k=xt_{k-1},\quad p_k=p_{k-1}+a_kt_k,$$

其中，$k=1,2,\cdots,n$，只有 $2n$ 次乘法，计算量减少了 n 的一个阶次。

如果使用秦九韶算法(或称 Horner 算法)

$$p_0=a_n,\quad p_k=xp_{k-1}+a_{n-k},\quad k=1,2,\cdots,n,$$

则只有 n 次乘法，计算量进一步又减少了一半。

(5) 注意影响数值稳定性的细节。

如应尽量避免相差悬殊的数加减，避免两个相近的数相减，避免绝对值很小的数作除数等。

例 1.8(相近的数相减)　已知 $x=\pi\times10^{-8}$，考虑采用双精度(15 位十进制)计算下式的算法设计(真实解为 0.197392×10^{-14})。

$$y=\frac{1}{1+2x}-\frac{1-x}{1+x}。\tag{1.18}$$

解　直接用式(1.18)求解得 0.19984×10^{-14}，可见只有 2 位有效数字。这是由于 x 接近于 0，式(1.18)涉及两个近似数的减法，容易造成相对误差增大。我们将其改为

$$y=\frac{2x^2}{(1+2x)(1+x)},\tag{1.19}$$

计算得 $y=0.19739\times10^{-14}$，有 5 位有效数字，效果就比较好。

例 1.9(大数吃小数)　考虑采用单精度(7 位十进制)计算下式的算法设计：

$$y=123456+\sum_{i=1}^{1000}x_i,\quad 0.01\leqslant x_i\leqslant0.04。\tag{1.20}$$

解　显然不等式 $123466\leqslant y\leqslant123496$ 成立。但如果我们按式(1.20)的顺序求解，结果

为 123456,也就是后面 1000 项“小数”x_i 被“大数”123456“吃掉了”。为什么呢? 因为计算机在作加减法时,总是先将各项阶码统一到最大的阶码,即

$$123456 \to 0.1234560 \times 10^6,$$

$$0.04 \to 0.4 \times 10^{-1} \to 0.00000004 \times 10^6 \to 0.0000000 \times 10^6。$$

由于两个数的阶码相差很大,0.04 的阶码上升到 10^6 时,其尾数就很小而超出了字长范围,就被舍去了。改进的算法是先求 $\sum_{i=1}^{1000} x_i$。由于 x_i 数量级相近,就不会发生“大数吃小数”,而 1000 项小数的总和超过 10,也已经不太小了,不至于被 123456“吃掉”。

(6) 节省存储空间。

例如考虑迭代

$$p_0 = a_n, \quad p_k = xp_{k-1} + a_{n-k}, \quad k = 1,2,\cdots,n。$$

如果将 $p_0, p_1, \cdots, p_n$ 作为一个数组,需要 $n+1$ 个实数单元,但其实我们并不需要 $p_0, p_1, \cdots, p_{n-1}$,且可以更新,所以 $p_0, p_1, \cdots, p_n$ 可以共用一个实数变量 p,这样只需 1 个实数单元。

(7) 避免死循环。

数值算法很多都具有收敛性问题。一旦出现不收敛或收敛速度太慢,普通 while 循环语句往往会进入死循环,所以使用 for 语句更可靠。如果使用 while 语句,最好设置一个循环次数的上界。

(8) 不要作实数相等比较。

由于十进制与二进制转换以及舍入误差的影响,实数相等的比较一般都是不成功的。比如用数值方法验证等式

$$\sin^2 x = 1 - \cos^2 x,$$

你会发现总是不成功。正确做法是,设置一个很小的误差限 ε,去验证

$$|\sin^2 x - (1 - \cos^2 x)| \leqslant \varepsilon。$$

(9) 尽量使用双精度实数。

整数和单精度实数在数值计算中往往造成舍入误差影响很大,只要计算时间允许,应尽量使用双精度实数。

(10) 尽量减少中间结果的显示和输出。

由于输出和写屏都需要占用内存空间,从而影响速度。减少中间结果显示,可以提高运算速度。

习　　题

1. 下列各数:

$$x_1 = 1.1021, \quad x_2 = 0.031, \quad x_3 = 56.430$$

都是经过四舍五入得到的近似数,即误差限不超过最后一位的半个单位。试计算下列各式的误差限和有效数字:

(1) $2x_1 - x_2 + x_3$;

(2) $x_1 x_2 x_3$;

(3) x_2 / x_3。

2. 计算球体积，要使相对误差限为 1%。问度量半径 r 时允许的相对误差限是多少？

3. 设 $s=0.5gt^2$，假定 g 是准确的，而对 t 的测量有 ± 0.1s 的误差。证明当 t 增加时 s 的绝对误差的绝对值增加，而相对误差的绝对值却减少。

4. 序列$\{y_n\}$满足递推关系

$$y_n = 10y_{n-1} - 1, \quad n = 1,2,\cdots,$$

若 $y_0=\sqrt{2}$取三位有效数字，计算到 y_{10}时误差有多大？这个计算过程数值稳定吗？

5. 设函数 $f(x)=\dfrac{e^x-1-x}{x^2}$，计算 $f(0.01)$的真值。如果使用 6 位有效数字计算，结果的误差有多大？考虑其近似公式 $f(x)\approx\dfrac{1}{2}+\dfrac{x}{6}+\dfrac{x^2}{24}$，仍然用 6 位有效数字计算，结果的误差有多大？哪种算法更精确？为什么？

6. 设 $a\neq 0$，$b^2-4ac>0$，考虑二次方程 $ax^2+bx+c=0$ 的求根公式

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}, \quad x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a},$$

及其等价公式

$$x_1 = \frac{-2c}{b+\sqrt{b^2-4ac}}, \quad x_2 = \frac{-2c}{b-\sqrt{b^2-4ac}},$$

当$|b|\approx\sqrt{b^2-4ac}$时，怎样算法设计比较合理？

7. 用三位尾数的浮点数按下列两种方式计算 $\sum\limits_{k=1}^{5}\dfrac{1}{k^4}$，为什么答案不同？哪个更准确？

(1) 按 k 递增顺序；

(2) 按 k 递减顺序。

8. 要计算 $(\sqrt{2}-1)^6$的近似值，取$\sqrt{2}\approx 1.4$，代入下列方法计算，再比较哪一个得到的结果最好。

$$\frac{1}{(\sqrt{2}+1)^6}, \quad \frac{1}{(3+2\sqrt{2})^3}, \quad (3-2\sqrt{2})^3, \quad 99-70\sqrt{2}。$$

9. 改进下列式子，使得计算结果更精确：

(1) $\sqrt{1+x}-\sqrt{x}$，当 x 充分大；

(2) $\dfrac{1-\cos x}{\sin x}$，当 x 接近于 0；

(3) $\ln(x-\sqrt{x^2-1})$，当 x 充分大。

10. 改进下列式子，使其减少运算次数：

(1) $(x-1)^4+3(x-1)^3-4(x-1)^2+x+5$；

(2) x^{15}。

上机实验题

实验 1（二进制）　用 MATLAB 计算 $\sum\limits_{i=1}^{1000}0.1-100$，会发现居然有误差！虽然从十进制数角度分析，这一计算应该是准确的，但是实验反映了计算机内部的二进制本质。

实验 2　用 MATLAB 计算 0.48－0.5＋0.02 会出现什么问题？怎样做可以避免出现这样的问题？

实验 3(数值稳定性)　用 MATLAB 验算例 1.5。

实验 4(病态方程组)　用 MATLAB 验算例 1.6。

实验 5(秦九韶算法)　编写 1.3 节秦九韶算法程序，并用该程序计算多项式

$$f(x)=x^5+3x^3-2x+6$$

在 $x=1.1,1.2,1.3$ 的值。

实验 6　设 x 是一个维数为 n 的数组，它的均值和标准差公式为

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

$$s_1=\sqrt{\frac{1}{n-1}\left[\sum_{i=1}^{n}(x_i-\bar{x})^2\right]},$$

$$s_2=\sqrt{\frac{1}{n-1}\left[\sum_{i=1}^{n}x_i^2-n\bar{x}^2\right]},\quad n\geqslant 1。$$

试回答下列问题：

(1) 标准差公式 s_1 和 s_2 理论上是等价的，但数值计算上会有什么不同？哪一个公式可能会更准确？

(2) 选取 $x=10^9,10^9+1,10^9+2$，验证你对问题(1)的猜测。

第 2 章

数 值 代 数

考虑线性方程组

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1, \\ \vdots \\ a_{n1}x_1 + \cdots + a_{mn}x_n = b_n, \end{cases}$$

当它的系数行列式

$$D = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{mn} \end{vmatrix} \neq 0$$

时，由线性代数理论知，其解可由克莱姆法则计算。第 1 章例 1.2 已指出这种方法的问题在于运算量太大，以至于在数值计算上毫无价值。

本章讨论线性方程组数值计算常用的直接解法及相关问题，包括：顺序高斯消去法、选列主元高斯消去法、矩阵 LU 分解、平方根分解等算法及程序设计；基于范数和条件数的概念，研究线性方程组求解过程中的误差分析问题；利用 MATLAB 软件，介绍方阵求逆、特征值与特征向量、正交三角分解、奇异值分解等进一步的数值问题。

2.1 高斯消去法

1. 理论基础

解线性方程组最常用的是高斯消去法，其理论基础是线性代数的初等变换理论。我们先证明线性代数中的一个引理，它是高斯消去法的理论基础。

引理 2.1 设有线性方程组

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1, \\ \vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n = b_m, \end{cases} \tag{2.1}$$

若对它的增广矩阵施行行初等变换 $(\boldsymbol{A}, \boldsymbol{b}) \xrightarrow{\text{行初等变换}} (\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{b}})$，则方程组(2.1)与

$$\begin{cases} \widetilde{a}_{11}x_1 + \cdots + \widetilde{a}_{1n}x_n = \widetilde{b}_1, \\ \vdots \\ \widetilde{a}_{m1}x_1 + \cdots + \widetilde{a}_{mn}x_n = \widetilde{b}_m, \end{cases} \tag{2.2}$$

同解。其中 $\boldsymbol{A}=(a_{ij})_{m\times n}$,$\boldsymbol{b}=\begin{pmatrix}b_1\\ \vdots\\ b_m\end{pmatrix}$,$\widetilde{\boldsymbol{A}}=(\widetilde{a}_{ij})_{m\times n}$,$\widetilde{\boldsymbol{b}}=\begin{pmatrix}\widetilde{b}_1\\ \vdots\\ \widetilde{b}_m\end{pmatrix}$。

证明 由$(\boldsymbol{A},\boldsymbol{b})\xrightarrow{\text{行初等变换}}(\widetilde{\boldsymbol{A}},\widetilde{\boldsymbol{b}})$知,存在初等矩阵 $\boldsymbol{P}_1,\cdots,\boldsymbol{P}_r$ 使 $\boldsymbol{P}_r\cdots\boldsymbol{P}_1(\boldsymbol{A},\boldsymbol{b})=(\widetilde{\boldsymbol{A}},\widetilde{\boldsymbol{b}})$。记 $\boldsymbol{P}=\boldsymbol{P}_r\cdots\boldsymbol{P}_1$,则 $\boldsymbol{P}$ 可逆且 $\boldsymbol{P}(\boldsymbol{A},\boldsymbol{b})=(\boldsymbol{PA},\boldsymbol{Pb})=(\widetilde{\boldsymbol{A}},\widetilde{\boldsymbol{b}})$得 $\boldsymbol{PA}=\widetilde{\boldsymbol{A}},\boldsymbol{Pb}=\widetilde{\boldsymbol{b}}$。

若 $\boldsymbol{x}^*$ 为方程组(2.1)的解,即 $\boldsymbol{Ax}^*=\boldsymbol{b}$,则 $\boldsymbol{PAx}^*=\boldsymbol{Pb}$,从而 $\widetilde{\boldsymbol{A}}\boldsymbol{x}^*=\widetilde{\boldsymbol{b}}$,知 $\boldsymbol{x}^*$ 也是方程组(2.2)的解;反之,若 $\boldsymbol{x}^*$ 为方程组(2.2)的解,即 $\widetilde{\boldsymbol{A}}\boldsymbol{x}^*=\widetilde{\boldsymbol{b}}$,则 $\boldsymbol{PAx}^*=\boldsymbol{Pb}$,从而由 $\boldsymbol{P}$ 可逆可得 $\boldsymbol{Ax}^*=\boldsymbol{b}$,知 $\boldsymbol{x}^*$ 也是方程组(2.1)的解。证毕。

2. 顺序高斯消去法

高斯消去法的基本思想是:首先使用初等变换将方程转化为一个同解的三角形方程组(称为**消元过程**),再通过回代法求解该三角形方程组(称为**回代过程**)。按行原先的位置进行消元的高斯消去法称为**顺序高斯消去法**(Gaussian elimination method)。

例 2.1 用顺序高斯消去法解线性方程组

$$\begin{cases}x_1+x_2+x_3=6,\\ -x_1+3x_2+x_3=4,\\ 2x_1-6x_2+x_3=-5。\end{cases}\tag{2.3}$$

解 消元过程:

$$\left(\begin{array}{ccc|c}1&1&1&6\\-1&3&1&4\\2&-6&1&-5\end{array}\right)\xrightarrow[r_3-2r_1]{r_2+r_1}\left(\begin{array}{ccc|c}1&1&1&6\\0&4&2&10\\0&-8&-1&-17\end{array}\right)\xrightarrow{r_3+2r_2}\left(\begin{array}{ccc|c}1&1&1&6\\0&4&2&10\\0&0&3&3\end{array}\right)。$$

回代过程:

$$\begin{cases}x_1+x_2+x_3=6,\\ 4x_2+2x_3=10,\\ 3x_3=3\end{cases}\Rightarrow\begin{cases}x_3=1,\\ x_2=\dfrac{10-2x_3}{4}=2,\\ x_1=6-x_2-x_3=3。\end{cases}\tag{2.4}$$

由引理 2.1 知,由上述消元与回代两阶段运算得到的结果恰为所求线性方程组的解。

对于一般线性方程组,使用顺序高斯消去法求解

$$\begin{cases}a_{11}^{(1)}x_1+\cdots+a_{1n}^{(1)}x_n=a_{1,n+1}^{(1)},\\ \vdots\\ a_{n1}^{(1)}x_1+\cdots+a_{mn}^{(1)}x_n=a_{n,n+1}^{(1)}。\end{cases}\tag{2.5}$$

消元过程:

$$\left(\begin{array}{ccccc|c}a_{11}^{(1)}&a_{12}^{(1)}&a_{13}^{(1)}&\cdots&a_{1n}^{(1)}&a_{1,n+1}^{(1)}\\a_{21}^{(1)}&a_{22}^{(1)}&a_{23}^{(1)}&\cdots&a_{2n}^{(1)}&a_{2,n+1}^{(1)}\\a_{31}^{(1)}&a_{32}^{(1)}&a_{33}^{(1)}&\cdots&a_{3n}^{(1)}&a_{3,n+1}^{(1)}\\\vdots&\vdots&\vdots&&\vdots&\vdots\\a_{n1}^{(1)}&a_{n2}^{(1)}&a_{n3}^{(1)}&\cdots&a_{mn}^{(1)}&a_{n,n+1}^{(1)}\end{array}\right)\rightarrow\left(\begin{array}{ccccc|c}a_{11}^{(1)}&a_{12}^{(1)}&a_{13}^{(1)}&\cdots&a_{1n}^{(1)}&a_{1,n+1}^{(1)}\\0&a_{22}^{(2)}&a_{23}^{(2)}&\cdots&a_{2n}^{(2)}&a_{2,n+1}^{(2)}\\0&a_{32}^{(2)}&a_{33}^{(2)}&\cdots&a_{3n}^{(2)}&a_{3,n+1}^{(2)}\\\vdots&\vdots&\vdots&&\vdots&\vdots\\0&a_{n2}^{(2)}&a_{n3}^{(2)}&\cdots&a_{mn}^{(2)}&a_{n,n+1}^{(2)}\end{array}\right)\rightarrow$$

$$\cdots \rightarrow \left(\begin{array}{ccccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} & a_{1,n+1}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} & a_{2,n+1}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} & a_{3,n+1}^{(3)} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} & a_{n,n+1}^{(n)} \end{array}\right),$$

其中

$$a_{ij}^{(2)} = a_{ij}^{(1)} - l_{i1}a_{1j}^{(1)}, \quad l_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \quad i = 2,3,\cdots,n, j = 2,3,\cdots,n+1。$$

一般地，有

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)}, \quad l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \tag{2.6}$$

$$i = k+1,\cdots,n, \quad j = k+1,\cdots,n+1, \quad k = 1,2,\cdots,n-1。$$

这里，$a_{kk}^{(k)}$ 称为第 k 步的主元素($k=1,2,\cdots,n$)。

回代过程：

$$\begin{cases} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = a_{1,n+1}^{(1)}, \\ a_{22}^{(2)}x_2 + \cdots + a_{2n}^{(2)}x_n = a_{2,n+1}^{(2)}, \\ \vdots \\ a_{nn}^{(n)}x_n = a_{n,n+1}^{(n)} \end{cases} \Rightarrow \begin{cases} x_n = a_{n,n+1}^{(n)}/a_{nn}^{(n)}, \\ x_k = (a_{k,n+1}^{(k)} - \sum\limits_{j=k+1}^{n} a_{kj}^{(k)}x_j)/a_{kk}^{(k)}, \\ k = n-1,n-2,\cdots,1。 \end{cases} \tag{2.7}$$

现在我们来统计两阶段的计算量，由于加减法明显比乘除法快，所以我们只统计乘除法的次数。

消元过程：第 k 步有$(n-k)(n-k+1)+(n-k)=(n-k)(n-k+2)$次乘除法($k=1,2,\cdots,n-1$)，共

$$\sum_{k=1}^{n-1}(n-k)(n-k+2) = \sum_{i=1}^{n-1}(i^2+2i) = \frac{n(n-1)(2n-1)}{6} + n(n-1) = \frac{n(n-1)(2n+5)}{6}$$

次乘除法。

回代过程：计算 x_k 时，有 $n-k+1$ 次乘除法($k=n,n-1,\cdots,1$)，共

$$\sum_{k=1}^{n}(n-k+1) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

次乘除法。

两阶段合计共

$$\frac{n(n-1)(2n+5)}{6} + \frac{n(n+1)}{2} = \frac{n^3}{3} + n^2 - \frac{n}{3} \approx \frac{n^3}{3}\ (\text{当 } n \text{ 很大时})$$

次乘除法。

可见消元过程有 n^3 数量级计算量，而回代过程只有 n^2 数量级计算量，所以高斯消去法的计算量主要在消元过程部分。与克莱姆法则相比，计算量从$(n+2)!$ 数量级减少到 n^3 数量级，计算量过大的问题已得到了根本解决。对于 30 阶线性方程组，使用克莱姆法则在每秒运行 1 千万亿次的超级计算机上需要亿万年。而使用高斯消去法可以在普通的计算机上用不到 1s 就可以解决。

可以证明，如果 $\boldsymbol{A}=(a_{ij})_{n\times n}$的顺序主子式

$$D_1 = a_{11}, \quad D_2 = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}, \quad \cdots, \quad D_n = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{vmatrix}$$

均不为零,顺序高斯消去法求解 $\boldsymbol{Ax}=\boldsymbol{b}$ 可行。

3. 选列主元高斯消去法

高斯消去法的计算过程是不可靠的,一旦出现主元素 $a_{kk}^{(k)}=0$,计算就不能进行。即使对所有 $k=1,2,\cdots,n$,$a_{kk}^{(k)}\neq 0$,也不能保证计算过程数值稳定。

例 2.2 由高斯消去法取 3 位有效数字解线性方程组

$$\begin{cases} 0.001x_1 + x_2 = 1, \\ x_1 + x_2 = 2。 \end{cases} \tag{2.8}$$

解 消元过程:

根据 3 位浮点数运算规则 $1-1000=(0.0001-0.1)\times 10^4 \xlongequal{\text{舍入}} (0.000-0.1)\times 10^4 = -1000$,同理 $2-1000 \xlongequal{\text{舍入}} -1000$。

$$\left(\begin{array}{cc|c} 0.001 & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \xrightarrow{r_2-1000r_1} \left(\begin{array}{cc|c} 0.001 & 1 & 1 \\ 0 & 1-1000 & 2-1000 \end{array}\right) \xrightarrow{\text{舍入}} \left(\begin{array}{cc|c} 0.001 & 1 & 1 \\ 0 & -1000 & -1000 \end{array}\right)。$$

回代过程:

由

$$\begin{cases} 0.001x_1 + x_2 = 1, \\ -1000x_2 = -1000 \end{cases} \Rightarrow \begin{cases} x_2 = 1.00, \\ x_1 = 0.00, \end{cases}$$

代入原方程组验算,发现结果严重失真。

分析结果失真的原因,发现由于第 1 列的主元素 0.001 绝对值过于小,从而消元过程作分母时把中间过程数值放大 1000 倍,使中间结果"吃"掉了原始数据,造成数值不稳定。

针对以上问题,考虑选用绝对值大的数作为主元素。

消元过程:

$$\left(\begin{array}{cc|c} 0.001 & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \xrightarrow{r_1 \leftrightarrow r_2} \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0.001 & 1 & 1 \end{array}\right) \xrightarrow{r_2-0.001r_1} \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1-0.001 & 1-0.002 \end{array}\right)$$

$$\xrightarrow{\text{舍入}} \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 & 1 \end{array}\right)。$$

这里舍入过程 $1-0.001=(0.1-0.0001)\times 10^1 \xlongequal{\text{舍入}} 1$;同理 $1-0.002 \xlongequal{\text{舍入}} 1$。

回代过程:

由

$$\begin{cases} x_1 + x_2 = 2, \\ x_2 = 1 \end{cases} \Rightarrow \begin{cases} x_2 = 1.00, \\ x_1 = 1.00, \end{cases}$$

代入原方程组验算,发现结果基本合理。

这里尽管也有"大数吃小数"现象,但由于没有造成中间过程数值放大,被"吃掉"的是中间量,数值稳定,所以效果好了很多。

在高斯消去法的第 k 步消元时，将第 k 列中第 k 行至第 n 行绝对值最大的元素选为主元素 $a_{kk}^{(k)}$ 实施消元($k=1,2,\cdots,n$)，这样的方法称为**选列主元高斯消去法**(Gauss elimination method with partial pivoting)。

例 2.3　用选列主元高斯消去法解方程组(2.3)。

解　消元过程：

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 6 \\ -1 & 3 & 1 & 4 \\ 2 & -6 & 1 & -5 \end{array}\right) \xrightarrow[\text{(选主元)}]{r_1 \leftrightarrow r_3} \left(\begin{array}{ccc|c} 2 & -6 & 1 & -5 \\ -1 & 3 & 1 & 4 \\ 1 & 1 & 1 & 6 \end{array}\right) \xrightarrow[r_3-0.5r_1]{r_2+0.5r_1} \left(\begin{array}{ccc|c} 2 & -6 & 1 & -5 \\ 0 & 0 & 1.5 & 1.5 \\ 0 & 4 & 0.5 & 8.5 \end{array}\right)$$

$$\xrightarrow{r_2 \leftrightarrow r_3} \left(\begin{array}{ccc|c} 2 & -6 & 1 & -5 \\ 0 & 4 & 0.5 & 8.5 \\ 0 & 0 & 1.5 & 1.5 \end{array}\right)。$$

注意：在第二次选主元素时只需比较第二列中第二行以后的各元素。

回代过程：

由

$$\begin{cases} 2x_1 - 6x_2 + x_3 = -5, \\ 4x_2 + 0.5x_3 = 8.5, \\ 1.5x_3 = 1.5 \end{cases} \Rightarrow \begin{cases} x_3 = 1, \\ x_2 = 2, \\ x_1 = 3。 \end{cases}$$

可以证明，只要 $\boldsymbol{A}=(a_{ij})_{n\times n}$ 可逆，选列主元的高斯消去法求解 $\boldsymbol{Ax}=\boldsymbol{b}$ 可行(见习题 4)。

4. 三对角线性方程组的追赶法

考虑数值计算问题中常见的一类三对角线性方程组

$$\begin{cases} b_1x_1 + c_1x_2 = d_1, \\ a_2x_1 + b_2x_2 + c_2x_3 = d_2, \\ \quad\ddots \quad\ddots \quad\ddots \quad\vdots \\ a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n = d_{n-1}, \\ a_nx_{n-1} + b_nx_n = d_n, \end{cases} \tag{2.9}$$

我们将高斯消去法应用于三对角线性方程组得到所谓**追赶法**，其中消元过程为“追”，回代过程为“赶”。

例 2.4　用追赶法解三对角线性方程组

$$\begin{cases} 3x_1 + x_2 = 2, \\ 2x_1 + 3x_2 + x_3 = 1, \\ 2x_2 + 3x_3 + x_4 = 2, \\ x_3 + 3x_4 = -4。 \end{cases} \tag{2.10}$$

解　追：

$$\left(\begin{array}{cccc|c} 3 & 1 & 0 & 0 & 2 \\ 2 & 3 & 1 & 0 & 1 \\ 0 & 2 & 3 & 1 & 2 \\ 0 & 0 & 1 & 3 & -4 \end{array}\right) \to \left(\begin{array}{cccc|c} 3 & 1 & 0 & 0 & 2 \\ 0 & 7/3 & 1 & 0 & -1/3 \\ 0 & 2 & 3 & 1 & 2 \\ 0 & 0 & 1 & 3 & -4 \end{array}\right) \to \left(\begin{array}{cccc|c} 3 & 1 & 0 & 0 & 2 \\ 0 & 7/3 & 1 & 0 & -1/3 \\ 0 & 0 & 15/7 & 1 & 16/7 \\ 0 & 0 & 1 & 3 & -4 \end{array}\right)$$

$$\rightarrow \left(\begin{array}{cccc|c} 3 & 1 & 0 & 0 & 2 \\ 0 & 7/3 & 1 & 0 & -1/3 \\ 0 & 0 & 15/7 & 1 & 16/7 \\ 0 & 0 & 0 & 38/15 & -76/15 \end{array}\right)。$$

赶:

由

$$\begin{cases} 3x_1 + x_2 = 2, \\ \dfrac{7}{3}x_2 + x_3 = -\dfrac{1}{3}, \\ \dfrac{15}{7}x_3 + x_4 = \dfrac{16}{7}, \\ \dfrac{38}{15}x_4 = -\dfrac{76}{15} \end{cases} \Rightarrow \begin{cases} x_4 = -2, \\ x_3 = 2, \\ x_2 = -1, \\ x_1 = 1。 \end{cases}$$

用追赶法计算公式可直接求解式(2.9):

$$\left(\begin{array}{ccccc|c} b_1 & c_1 & & & & d_1 \\ a_2 & b_2 & c_2 & & & d_2 \\ & \ddots & \ddots & \ddots & & \vdots \\ & & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ & & & a_n & b_n & d_n \end{array}\right) \rightarrow \left(\begin{array}{ccccc|c} \widetilde{b}_1 & c_1 & & & & \widetilde{d}_1 \\ & \widetilde{b}_2 & c_2 & & & \widetilde{d}_2 \\ & & \ddots & \ddots & & \vdots \\ & & & \widetilde{b}_{n-1} & c_{n-1} & \widetilde{d}_{n-1} \\ & & & & \widetilde{b}_n & \widetilde{d}_n \end{array}\right)。$$

追:

$$\widetilde{b}_1 = b_1, \quad \widetilde{d}_1 = d_1, \quad l_k = \frac{a_k}{\widetilde{b}_{k-1}}, \quad \widetilde{b}_k = b_k - l_k c_{k-1},$$

$$\widetilde{d}_k = d_k - l_k \widetilde{d}_{k-1}, \quad k = 2,3,\cdots,n。 \tag{2.11}$$

赶:

$$x_n = \frac{\widetilde{d}_n}{\widetilde{b}_n}, \quad x_k = \frac{\widetilde{d}_k - c_k x_{k+1}}{\widetilde{b}_k}, \quad k = n-1, n-2, \cdots, 1。 \tag{2.12}$$

追赶法不需要对零元素计算,只有 $5n-4$ 次乘除法计算量,且当系数矩阵对角占优时数值稳定,是解三对角方程组的优秀算法。

5. 算法和程序

程序 2.1 根据式(2.6)和式(2.7)编写,使用 MATLAB,对于 i,j 的循环可直接简单用分块矩阵处理。

程序 2.1 (顺序高斯消去法)

```
function x=nagauss(a,b,flag)
%用途: 顺序高斯消去法解线性方程组 ax=b
%a: 系数矩阵,b:右端列向量
%flag: 若为 0,则显示中间过程,否则不显示,默认值为 0
%x: 解向量
```

```
if nargin<3,flag=0; end
n=length(b); a=[a,b];
%消元
for k=1:(n-1)
  a((k+1):n,(k+1):(n+1))=a((k+1):n,(k+1):(n+1))...
    -a((k+1):n,k)/a(k,k)*a(k,(k+1):(n+1));
  a((k+1):n,k)=zeros(n-k,1);
  if flag==0,a,end
end
%回代
x=zeros(n,1);
x(n)=a(n,n+1)/a(n,n);
for k=n-1:-1:1
  x(k)=(a(k,n+1)-a(k,(k+1):n)*x((k+1):n))/a(k,k);
end
```

用以解例 2.1,在 MATLAB 指令窗口执行:

```
>>A=[1 1 1;-1 3 1;2 -6 1]; b=[6; 4; -5];
>>x=nagauss(A,b)
a=
     1     1     1     6
     0     4     2    10
     0    -8    -1   -17
a=
     1     1     1     6
     0     4     2    10
     0     0     3     3
x=
     3
     2
     1
```

选列主元高斯消去法类似。不同之处为,对于任意 k,执行式(2.6)前先选列主元。方法是,设 $|a_{pk}^{(k)}|=\max\limits_{k\leqslant i\leqslant n}|a_{ik}^{(k)}|$,如果 $p>k$,则第 k 行与第 p 行交换。

选列主元高斯消去法流程图如图 2-1 所示。

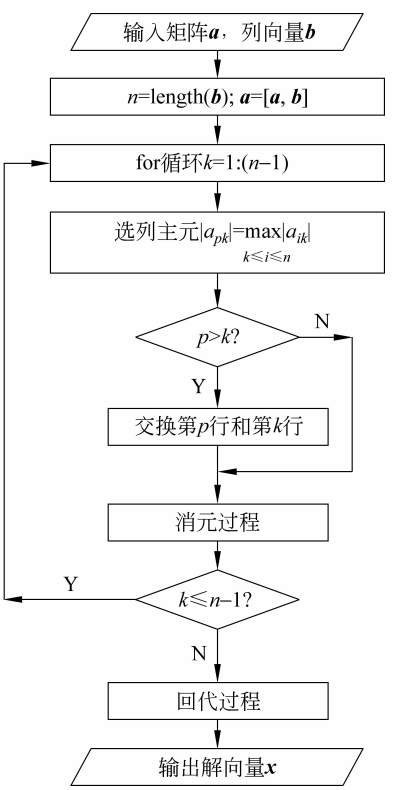

图 2-1　选列主元高斯消去法流程图

程序 2.2　(选列主元高斯消去法)

```
function x=nagauss2(a,b,flag)
%用途: 选列主元高斯消去法解线性方组 ax=b
%a:系数矩阵,b:右端列向量
%flag:若为 0,则显示中间过程,否则不显示。默认值为 0
%x:解向量
```

```
if nargin<3,flag=0; end
n=length(b); a=[a,b];
for k=1:(n-1)
%选列主元
[ap,p]=max(abs(a(k:n,k)));                    %找出绝对值最大数的相对位置
p=p+k-1;                                      %把相对位置转换成绝对位置
if p>k,
    t=a(k,:); a(k,:)=a(p,:); a(p,:)=t;        %交换行
end
%消元
a((k+1):n,(k+1):(n+1))=a((k+1):n,(k+1):(n+1))...
  -a((k+1):n,k)/a(k,k)*a(k,(k+1):(n+1));
a((k+1):n,k)=zeros(n-k,1);
if flag==0,a,end
end
%回代
x=zeros(n,1);
x(n)=a(n,n+1)/a(n,n);
for k=n-1:-1:1
  x(k)=(a(k,n+1)-a(k,(k+1):n)*x((k+1):n))/a(k,k);
end
```

用以解例 2.1,在 MATLAB 指令窗口执行:

```
>>A=[1 1 1;-1 3 1;2 -6 1]; b=[6;4;-5];
>>x=nagauss2(A,b)
a=
       2.0000  -6.0000  1.0000  -5.0000
            0         0  1.5000   1.5000
            0   4.0000  0.5000   8.5000
a=
       2.0000  -6.0000  1.0000  -5.0000
            0   4.0000  0.5000   8.5000
            0         0  1.5000   1.5000
x=
    3
    2
    1
```

对于大型线性方程组,高斯消去法可采用紧凑存储,采用一个增广矩阵($\boldsymbol{A}$,$\boldsymbol{b}$),消元系数 l 可利用 $\boldsymbol{A}$ 的下三角部分存储,向量 $\boldsymbol{x}$ 可利用 $\boldsymbol{b}$ 存储。而追赶法存储可仅采用 $\boldsymbol{a}$,$\boldsymbol{b}$,$\boldsymbol{c}$,$\boldsymbol{d}$ 四个列向量,$\boldsymbol{x}$ 可用 $\boldsymbol{d}$ 存储。