科学计算讲义习题及解答

Jianguo Huang & Yue Yu

目　录

第一章　绪论

Ex 1.1　$\sqrt{7}$ 可以由下列迭代算法计算:

$$x_0 = 2;\ \ x_{k+1} = \frac{1}{2}\left(x_k + \frac{7}{x_k}\right),\ k = 0, 1, 2, \cdots$$

(1) 证明: $x_k \to x^* = \sqrt{7}$.

(2) 证明: 若 x_k 是 $\sqrt{7}$ 有 n 位有效数字的近似值, 则 x_{k+1} 是 $\sqrt{7}$ 具至少 $2n$ 位有效数字的近似值.

证明　(1) 可以先证明 $\{x_k\}$ 是收敛序列, 然后取极限. 下面采用单调收敛定理证明. 因 $x_0 = 2 > 0$, 故 $x_k > 0,\ k \geqslant 1$, 从而由基本不等式, 得

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{7}{x_k}\right) = \frac{1}{2}\left((\sqrt{x_k})^2 + \left(\frac{\sqrt{7}}{\sqrt{x_k}}\right)^2\right) \geqslant \sqrt{7},\ k = 0, 1, \cdots$$

即序列有下界. 由此又可得

$$x_{k+1} - x_k = \frac{1}{2}\left(x_k + \frac{7}{x_k}\right) - x_k = \frac{1}{2}\left(-x_k + \frac{7}{x_k}\right) = \frac{1}{2}\frac{7 - x_k^2}{x_k} \leqslant 0,\ \ k \geq 1,$$

即序列 $\{x_k : k \geqslant 1\}$ 还是单调递减的, 因此收敛. 取极限并注意到序列非负即证.

(2) 用定义证明. 由题意及第一问,

$$\left|x_k - \sqrt{7}\right| = x_k - \sqrt{7} \leqslant \frac{1}{2} \times 10^{-n+1}\ \ (m = 0),$$

于是

$$\begin{aligned}\left|x_{k+1} - \sqrt{7}\right| &= x_{k+1} - \sqrt{7} = \frac{1}{2}\left(x_k + \frac{7}{x_k}\right) - \sqrt{7} \\ &= \frac{\left(x_k - \sqrt{7}\right)^2}{2x_k} \leqslant \frac{1}{2x_k} \cdot \frac{1}{4} \times 10^{-2n+2} \\ &= \frac{1}{2} \cdot \frac{10}{4x_k} \times 10^{-2n+1} \leqslant \frac{1}{2} \cdot \frac{10}{4 \cdot \sqrt{7}} \times 10^{-2n+1} < \frac{1}{2} \times 10^{-2n+1},\end{aligned}$$

证毕. □

注 1.1　利用有效数字与相对误差限的关系似乎很难得到 $2n$ 位有效数字, 而只能得到 $2n - 1$ 位. 事实上, 由题意及第一问,

$$\varepsilon_r^* = \frac{\left|\sqrt{7} - x_k\right|}{|x_k|} = \frac{x_k - \sqrt{7}}{x_k} \leqslant \frac{1}{2a_1} \times 10^{-(n-1)} \quad (a_1 = 2).$$

直接计算, 有

$$\frac{\left|\sqrt{7}-x_{k+1}\right|}{|x_{k+1}|}=\frac{x_{k+1}-\sqrt{7}}{x_{k+1}}=1-\frac{\sqrt{7}}{x_{k+1}}=\frac{\left(x_k-\sqrt{7}\right)^2}{x_k^2+7}$$
$$=\frac{x_k^2}{x_k^2+7}\frac{\left(x_k-\sqrt{7}\right)^2}{x_k^2}=\frac{x_k^2}{x_k^2+7}(\varepsilon_r^*)^2\leqslant\frac{1}{4a_1^2}\times10^{-2(n-1)},$$

注意到对任意 $a_1\geqslant1$, 有 $\frac{1}{4a_1^2}\leqslant\frac{1}{2(a_1+1)}$, 因此至少有 $2n-1$ 位是正确的 ($10^{-2(n-1)}=10^{-(2n-1-1)}$). 若考虑 $2n$ 位, 则右端应改写为

$$\frac{10}{4a_1^2}\times10^{-(2n-1)},$$

但 $\frac{10}{4a_1^2}\leqslant\frac{1}{2(a_1+1)}$ 并不成立, 而且很难再实行其他放缩.

Ex 1.2 已知

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

如果取 $\sqrt{2}\doteq1.4$, 上面的哪个算式得到的结果最精确? 给出详细推导.

解答 因 $\sqrt{2}=1.414\cdots$, 故 $h=\sqrt{2}-1.4\approx0.01$. 对 $(\sqrt{2}-1)^6$, 有

$$\begin{aligned}&\left|(\sqrt{2}-1)^6-(1.4-1)^6\right|\\&=\left|(0.4+h)^6-(0.4)^6\right|=(0.4+h)^6-(0.4)^6\\&\approx(0.4+0.01)^6-(0.4)^6\approx6.5410\times10^{-4}.\end{aligned}$$

类似可得,

$$\left|(3-2\sqrt{2})^3-(3-2\times1.4)^3\right|=\left|(0.2-2h)^3-(3-2\times1.4)^3\right|\approx0.0022,$$
$$\left|(99-70\sqrt{2})-(99-70\times1.4)\right|\approx0.7,$$
$$\left|\frac{1}{(1+\sqrt{2})^6}-\frac{1}{(1+1.4)^6}\right|\approx1.2893\times10^{-4},$$
$$\left|\frac{1}{(3+2\sqrt{2})^3}-\frac{1}{(3+2\times1.4)^3}\right|\approx5.2656\times10^{-5},$$
$$\left|\frac{1}{99+70\sqrt{2}}-\frac{1}{99+70\times1.4}\right|\approx1.7973\times10^{-5}.$$

比较以上结果可以发现, 最后一个计算得最好. □

注 1.2 误差限只能给出误差的上界，上界很大不一定本身很大，因此用误差限评估不是一个理想的方案.

Ex 1.3 试改变下列表达式使计算结果比较精确:

(1)

$$\frac{1}{1+2x}-\frac{1-x}{1+x},\quad |x|\ll 1;$$

(2)

$$\sqrt{x+\frac{1}{x}}-\sqrt{x-\frac{1}{x}},\quad x\gg 1.$$

解答 (1) 避免相近的数相减

$$\frac{1}{1+2x}-\frac{1-x}{1+x}=\frac{2x^2}{(1+2x)(1+x)};$$

(2) 避免相近的数相减

$$\sqrt{x+\frac{1}{x}}-\sqrt{x-\frac{1}{x}}=\frac{\sqrt{x^2+1}}{\sqrt{x}}-\frac{\sqrt{x^2-1}}{\sqrt{x}}=\frac{2}{\sqrt{x}\left(\sqrt{x^2+1}+\sqrt{x^2-1}\right)}.$$

Ex 1.4 找至少两种方法有效计算

$$f(x)=\frac{1-\cos x}{x^2},\ \ x\neq 0,\ \ |x|\ll 1.$$

解答 方法一: 用三角恒等式提高分子的计算精度

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

方法二: 考虑 Taylor 展开

$$\cos x=1-\frac{1}{2}x^2+\frac{1}{4!}x^4-\cdots,\qquad \frac{1-\cos x}{x^2}\approx\frac{1}{2}-\frac{1}{4!}x^2$$

Ex 1.5 设 $Y_0=28$, 按递推公式

$$Y_n=Y_{n-1}-\frac{1}{100}\sqrt{783},\ \ n=1,2,\cdots$$

计算到 Y_{100}. 若取 $\sqrt{783}\approx 27.982$ (5 位有效数字), 试问计算 Y_{100} 将有多大误差? 如果将递推公式改为

$$Y_n=2Y_{n-1}-\frac{1}{100}\sqrt{783},$$

情况又怎样?

解答 (1)为了方便, 记 $Y=\sqrt{783}$, $Y^*=27.983$, 则

$$e=|Y-Y^*|\leqslant\frac{1}{2}\times10^{m-n+1}=\frac{1}{2}\times10^{1-5+1}=\frac{1}{2}\times10^{-3}.$$

由递推公式,

$$Y_n-Y_n^*=Y_{n-1}-Y_{n-1}^*-\frac{1}{100}(Y-Y^*),\ \ n=1,2,\cdots$$

其中 $Y_0=Y_0^*=28$, 这样错位相消可得

$$Y_n-Y_n^*=-\frac{n}{100}(Y-Y^*),\ \ n=1,2,\cdots$$

当 $n=100$ 时, 误差为

$$|Y_{100}-Y_{100}^*|\leqslant e=\frac{1}{2}\times10^{-3}.$$

(2) 此时有

$$Y_n-Y_n^*=2(Y_n-Y_n^*)-\frac{1}{100}(Y-Y^*),\ \ n=1,2,\cdots$$

从而

$$\begin{aligned}Y_n-Y_n^*&=2^n(Y_0-Y_0^*)-(1+2+\cdots+2^{n-1})\frac{1}{100}(Y-Y^*)\\&=-\frac{2^n-1}{100}(Y-Y^*),\quad n=1,2,\cdots\end{aligned}$$

因此误差将会放大 $(2^{100}-1)/100$ 倍.

Ex 1.6 给定 n 个矩阵 $A_1,A_1,\cdots,A_n$, 试给出最优方法 (算法) 获得 $A=A_1A_1\cdots A_n$. 换言之, 给出最佳的求矩阵乘积顺序以获得 A.

解答: 这是所谓的“矩阵的链乘法”问题. 矩阵的乘法是满足结合律的, 因此矩阵的连乘积可以任意加括号, 而对不同维数的矩阵相乘, 加括号会改变元素相乘的次数 (这里忽略加法次数). 下面以三个矩阵相乘为例说明这一点. 首先对两个矩阵 A,B, 若它们的维数分别为 $p\times q$ 和 $q\times r$, 令 $C=AB$, 则

$$c_{ij}=\sum_{k=1}^{q}a_{ik}b_{kj},\quad 1\leqslant i\leqslant p,\ \ 1\leqslant j\leqslant r,$$

显然计算完所有分量需要 pqr 次乘法. 设 A_1 维数为 $p_1\times p_2$, A_2 维数为 $p_2\times p_3$, A_3 维数为 $p_3\times p_4$, 用 $N(\cdot)$ 表示计算次数, 则

$$N\left((A_1A_2)A_3\right)=p_1p_2p_3+p_1p_3p_4,$$

$$N\left(A_1(A_2A_3)\right) = p_2p_3p_4 + p_1p_2p_4.$$

取 $p_1 = 10,\ \ p_2 = 100,\ \ p_3 = 5,\ \ p_4 = 50$, 有

$$N\left((A_1A_2)A_3\right) = 7500, \qquad N\left(A_1(A_2A_3)\right) = 75000,$$

可以看到两者差别很大.

现在需要给出一个加括号的方法, 使得计算次数最小, 我们分以下几步说明.

Step 1 给定加括号的结果计算矩阵乘积

在给出加括号的算法之前, 我们先考虑给定加括号后, 如何利用加括号的结果给出矩阵的连乘积. 为了方便表述, 用一个例子来说明. 设矩阵 A_i 的维数为 $p_i \times p_{i+1}$, 它们可用一个数组存储: $p = [p_1, \cdots, p_n, p_{n+1}]$. 以下取 $p = [5, 6, 2, 9, 7, 6]$, $A = A_1A_2A_3A_4A_5$.

- 给出的加括号为 $\left((A_1A_2)\left((A_3A_4)\,A_5\right)\right)$, 在计算中一般表示为 ((A1A2)((A3A4)A5)). 注意到 A1 在计算中一般视为两个字符, 为了方便处理, 我们直接写为 1, 从而加括号为 a = ((12)((34)5)). 显然该例子中的 ((34)5) 可以直接为 (345), 但后面的算法给出的是最小长度 (括起来的部分) 为 2 的, 这种加括号不妨称为加“全括号”.

- 思路是去括号与保形式.

  1) 初始化 A 为所有矩阵构造的元胞数组, 即 $A = \{A_1, A_2, A_3, A_4, A_5\}$.

  2) 对字符串循环, 找到第一个右括号, 并记下它的下标, 然后往左侧找第一个左括号, 记录下标 (由于最小长度为 2, 因此左括号的位置是立马可以知道的). 对所给例子, 左括号标号为 ind1 = 2, 右括号为 ind2 = 5.

  3) 更新 A. 我们需要删除括号内对应的矩阵, 而且把乘积放到对应位置上, 形成的新矩阵按原始位置重新排序. 对例子, 令 $B = A_1A_2$, 则新的元胞数组 $A = \{B, A_3, A_4, A_5\}$. 我们希望仍按 1,2,3,...这样排列, 从而可以按初始情况同样处理. 做法是用 B 代替 A_1, 给出 $A = \{B, A_2, A_3, A_4, A_5\}$, 然后删除 A_2, 从而得到 $A = \{B, A_3, A_4, A_5\}$. 元胞的索引就对应 1,2,3,...

  4) 在计算 B 的过程中, 可以给出乘法的计算次数.

  5) 更新 p. 对例子, $p = [p_1, p_2, p_3, p_4, p_5, p_6]$, 因为 B 要加入到链中, 所以只需要删除第二个, 即 $p = [p_1, p_3, p_4, p_5, p_6]$.

  6) 更新字符串 a. 我们要删除 (12), 但因为要加入 B, 这里可以把它变为 112), 然后删除后面的部分, 从而得到 a = (1((34)5)). 为了保持形式, 可以把所有的数字 1,3,4,5 替换成 1,2,3,4, 最终给出 (1((23)4)).

7) 经过以上处理, 我们获得了与初始情况完全类似的问题, 从而可重复以上步骤.

CODE 1.1. newstring.m (去括号与保形式)

```
function [p,A,a,sums] = newstring(p,A,a,sums)

%% 第一个右括号的标号
ind2 = find (a ==')'); % 找到所有右括号的下标
ind2 = ind2(1);  % 第一个为需要的

% 找到相应的左括号标号
% ind1 = find(a == '(');  % 找到所有左括号的下标
% tem = find(ind1 < ind2); % 在右括号旁边的所有左括号的索引
% ind1 = ind1(tem(end)); % 索引对应的最后一个即为所求
ind1 = ind2 - 3;

%% 新的 p 转化为( 1234...) 与数乘步数
n0 = double(a(ind1+1)) - '1' + 1; % 删除部分对应的第一个真实数字
sums = sums + p(n0)*p(n0+1)*p(n0+2);
p(n0+1) = []; % 只要删除第二个, 因为新的乘积矩阵还有维数


%% 新的矩阵转化为 ( 1234...)
B = A{n0}*A{n0+1}; % 因为一定是两个相乘
A{n0} = B;
A(n0+1) = [];

%% 新的字符
a(ind1) = '1'; % 删除部分第一个赋值为 '1'
a(ind1+1:ind2) = []; % 删除后面几位
ind = find(a ≥'1'); % 找到数字的位置
a(ind) = (0:length(ind)-1) + '1';
```

对以上提到的例子, 我们随机生成一个矩阵, 编程如下

CODE 1.2. MultiplicationA.m (计算矩阵乘积与乘法次数)

```
function [sume,sums,Ae,A] = MultiplicationA(p,a)
n = length(p)-1; sume = prod(p);
% 给出固定的随机矩阵
```

```
ss = rng;
for i = 1:n
    A{i} = rand(p(i),p(i+1));
end
rng(ss)

Ae = A{1};
for i = 2:n
    Ae = Ae*A{i};   % 直接相乘的结果
end

%% 计算
sums = 0;
while n ≥ 2 % n 至少为 2
    [p,A,a,sums] = newstring(p,A,a,sums);
    n = length(p)-1;
end
A = A{1};
```

计算结果为: 直接相乘次数为 sume = 22680, 加括号的计算次数为 sums = 330 (注意这个结果与随机矩阵无关), 而且经比较, 两个乘积结果完全一致.

Step 2 加括号算法

设矩阵 A_i 的维数为 $p_i \times p_{i+1}$, 称 $A[i,j] = A_iA_{i+1}\cdots A_j$ 为一个链, 则必存在加括号方式, 使得这个链的计算次数最少, 记为 $m(i,j)$. 当然整个加括号后的链必然从某个 k 处断裂成两部分: $(A_i\cdots A_k)(A_{k+1}\cdots A_j)$, 这里每个部分都可能存在其他括号. 为了方便说明, 我们称此时的 k 为链 $A[i,j]$ 的断点. 加括号算法的构造基于如下断言:

命题 1.1 对断点 k, 两个子部分的加括号方式给出的也是最优结果.

事实上, $(A_i\cdots A_k)(A_{k+1}\cdots A_j)$ 的计算次数由三部分组成: $A[i,k]$ 加括号后的计算次数, $A[k+1,j]$ 加括号后的计算次数, 以及它们两个相乘的计算次数. 第三部分的次数始终为 $p_ip_{k+1}p_{j+1}$, 而每部分总可以改变加括号的方式使得最小.

由这个命题, 我们可以给出加括号算法. 简单来说就是: 先计算出整个链的断点, 然后对每个子链分别求断点, 直至不能再分为止. 这个思路说起来简单, 但是编程的话并不容易, 因为不断有新的子链生成. 但注意到对子链的处理与整个链的处理完全一样, 而且它们是按顺序进行的, 因此可用递归编程.

Step 3 递推公式、最优矩阵与断点矩阵

基于上面的命题, 可以获得如下的递推公式:

$$m(i,j)=\begin{cases}0, & i=j,\\ m(i,k)+m(k+1,j)+p_ip_{k+1}p_{j+1}, & i<j.\end{cases}$$

要注意这个式子中 k 是未知的, 它只能逐一验证. 为了获得公式右端中的计算项, 我们必须按长度为 $1,2,\cdots$ 的链顺序计算. 实际上, 长度为 1 的链对应的最小数是知道的 (事实上长度为 2 的也知道), 即

$$m(i,i)=0,\quad 1\leqslant i\leqslant n,$$
$$m(i,i+1)=p_ip_{i+1}p_{i+2},\quad 1\leqslant i\leqslant n-1.$$

由此我们就可以计算 $m(i,i+2),\ 1\leqslant i\leqslant n-2$. 这只需要在

$$m(i,i+2)=m(i,k)+m(k+1,i+2)+p_ip_{k+1}p_{i+3}$$

逐一取 $k=i,\cdots,j-1$, 然后比较 $\{\eta(k)\}$ 的大小, 此时

$$\eta(i)=m(i,i)+m(i+1,i+2)+p_ip_{i+1}p_{i+3}$$

$$\eta(i+1)=m(i,i+1)+m(i+2,i+2)+p_ip_{i+2}p_{i+3}$$

都是知道的. 在计算 $m(i,i+3)$ 时可以发现需要其他长度为 2 的结果, 因此必须把所有长度为 2 的算出来, 才能算长度为 3 的. 计算中的断点用 s(i,j) 存储, 于是有程序

CODE 1.3. compute_ms.m (最优矩阵与断点矩阵)

```
function [m,s] = compute_ms(p)

n = length(p)-1;
%% 长度为 1 的
m = zeros(n,n); % 长度为 1 的为 0, 直接初始化(只有 i ≤ j 的是结果)
s = zeros(n,n); % 存储断点(只有  i ≤ j 的是结果)

%% 长度为 2 到 n 的
for L = 2:n
    for i = 1:n-L+1
        j = i+L-1; % 长度为 L
        step = 1;
        for k = i:j-1 % 公式循环
```

```
14              temp(step) = m(i,k) + m(k+1,j) + p(i)*p(k+1)*p(j+1);
15              step = step + 1;
16          end
17          tempk = (i:j-1);
18          [m(i,j),ind] = min(temp);
19          s(i,j) = tempk(ind);
20      end
21  end
```

对例子, 结果为

```
m =

    0    60   150   256   330
    0     0   108   210   282
    0     0     0   126   210
    0     0     0     0   378
    0     0     0     0     0

s =

    0     1     2     2     2
    0     0     2     2     2
    0     0     0     3     4
    0     0     0     0     4
    0     0     0     0     0
```

Step 4 加括号的实现 (打印并保存结果到文件中)

采用递归编程, 程序如下

CODE 1.4. print_optimal_parens.m (加括号程序)

```
1  function print_optimal_parens(fid,s,i,j)
2
3  % fid 是要把打印的东西保存在新建的文件 fid 中
4  if i == j
5      fprintf('A%d',i);
6      fprintf(fid,'%d',i);
```

```
else
    fprintf('(');    fprintf(fid,'(');
    print_optimal_parens(fid,s,i,s(i,j));
    print_optimal_parens(fid,s,s(i,j)+1,j);
    fprintf(')');    fprintf(fid,')');
end
```

可如下调用上面的程序

CODE 1.5. 打印的主程序

```
clc;clear;
%% 初始化
p = [5 6 2 9 7 6]; n = length(p)-1;
[m,s] = compute_ms(p);

fid = fopen('getstr.txt','wt'); % 新建名为 getstr.txt 的文本文件
fprintf('The printed result: \n\n');
print_optimal_parens(fid,s,1,n) % 递归打印字符串
fprintf('\n');

a = importdata('getstr.txt');  % 读取保存在 getstr 中的字符串
a = char(a);
```

结果为

```
The printed result:


  ((A1A2)((A3A4)A5))
a =
   ((12)((34)5))
```

Step 4 程序的整理

把前面提到的函数文件 compute_ms.m, print_optimal_parens.m, MultiplicationA.m, newstring.m 放在同一个文件夹中 (如 program_homework_1), 编写如下主程序即可

CODE 1.6. 打印的主程序

```
clc;clear;
%% 初始化
```

```
p = [5 6 2 9 7 6]; n = length(p)-1;

%% 计算最优矩阵 m 和断点矩阵 s
[m,s] = compute_ms(p);

%% 打印输出结果, 保存在文本文件, 并读取字符串
fid = fopen('getstr.txt','wt'); % 新建名为 getstr.txt 的文本文件
fprintf('The printed result: \n\n');
print_optimal_parens(fid,s,1,n) % 递归打印字符串
fprintf('\n');
a = importdata('getstr.txt');  % 读取保存在 getstr 中的字符串
a = char(a)

%% 随机生成矩阵, 计算连乘积和次数
[sume,sums,Ae,A] = MultiplicationA(p,a)
```

结果为

```
The printed result:  ((A1A2)((A3A4)A5))
a = ((12)((34)5))
sume =   22680        sums =   330
Ae =
   24.9645   17.6380   21.3810   10.6112   24.9248   15.4404
   33.0347   23.2988   28.2158   14.0315   32.9165   20.4418
   23.8875   16.7820   20.2802   10.1304   23.6973   14.7976
   33.5484   23.6396   28.6143   14.2445   33.3940   20.7650
   26.7800   18.8208   22.7484   11.3587   26.5775   16.5878
A =
   24.9645   17.6380   21.3810   10.6112   24.9248   15.4404
   33.0347   23.2988   28.2158   14.0315   32.9165   20.4418
   23.8875   16.7820   20.2802   10.1304   23.6973   14.7976
   33.5484   23.6396   28.6143   14.2445   33.3940   20.7650
   26.7800   18.8208   22.7484   11.3587   26.5775   16.5878
```

参考文献

[1] 李庆扬、王能超、易大义, 数值分析 (第 5 版), 清华大学出版社, 北京, 2008.

[2] 张平文、李铁军, 数值分析, 北京大学出版社, 北京, 2007.

[3] 王沫然, MATLAB 与科学计算 (第 3 版), 电子工业出版社, 北京, 2012.

[4] K. Atkinson and W. Han, Elementary Numerical Analysis (Third Edition), John Wiley & Sons, New York, 2004.

[5] R. Neapolitan and K. Naimipour, Foundations of Algorithms (Fourth Edition), Jones and Bartlett Publishers, Boston, 2011.

[6] J. Stoer and R. Bulirsch, Introduction to Numerical Analysis (Third Edition), Springer, New York, 2002.