数值分析与算法 第二次大作业 实验报告

自 45 柳荫 2014011858

要求：

编写求 ln(x)的函数
要求：
（1）采用方法：
  a. Taylor 展开（最佳或近似最佳逼近）；
  b. 数值积分；
  c. 非 Taylor 展开的函数逼近方法（选作）；
（2）给出小数点后 20 位精度的结果（需要自行编写能够达到指定任意精度的四则运算）；
（3）分析选用方法的计算代价、收敛速度等；
（4）分析选用方法的方法误差和存储误差对最终结果的影响（除法带来误差忽略不计）。
说明：
（1）？？的取值范围为[1,100]， 输入最多 5 位有效数字；
（2）具体要求参见第一次大作业要求；
（3）自行编写全部算法；
（4）报告内容应完整，包括：上一页“要求”中所列内容、程序框图、实验结果及分析等。
（5）对于大作业抄袭与被抄袭者，以 0 分处理。

程序编译运行环境：

java jdk : 1.8.0_102。

需求分析：

要用多种方法求大数的对数值，无论是 Taylor 展开、数值积分还是其他方法，势必涉及到许多大数之间的加减乘除，于是得自行编写大数之间的四则运算，有了任意精度的四则运算，之后泰勒展开、数值积分等都只是调用公式了。

方案设计：

写了一个大数类 HugeNumber， 有 3 个成员变量，分别是 String 类存储的整数部分和小数部分以及 boolean 类的正负。 其构造函数有 3 种，空参数对应着大数 0，一个 String 的参数对应着一个大数写出来的样子，还有 3 个形参分别是整数部分、小数部分、正负的构造函数。

其他的准备工作的成员函数是以下几个：获取整数部分 getInteger、获取小数部分 getDecimal、获取正负 isPos、比较 2 个整数的大小 compareInteger、比较 2 个纯小数的大小 compareDecimal、比较 2 个大数绝对值的大小 compareAbs、对一个大数取其相反数 inverseHugeNumber、整数加法主要算法 addCoreInteger、正整数加法 addPosInteger、小数加法主要算法 addCoreDecimal、整数减法主要算法 subCoreInteger、正整数减法（大减小）subPosInteger、小数减法主要算法 subCoreDecimal、乘法主要算法 multiplyCore、除法求一位商 divOneBit、判断是否是 10 的整数幂次 isPowerOfTen、求倒数（指定有效数字个数）getReciprocal、大数加大数 jia、大数减大数 jian、大数乘大数 cheng、大数乘幂 chengMi。

最后 5 个，前三个是最关键的求取 ln(x)的算法的成员函数，分别为精确到小数点后 n 位的 taylor 展开求 lnx lnTaylor，用复化辛普森公式的把区间分为 n 段小区间的数值积分法来求 lnx lnIntegration，用连分数展开的函数逼近来求 lnx lnSeriesFraction。还有 2 个函数是用来对结果的后处理：精确到小数点后 n 位的截断函数 getFirstNBits 以及 显示结果的函数 display。

Taylor 展开：

因为测试的数的范围在[1,100]，因此而 ln(t)利用代换 t = 1 + x 转化为 ln(1 + x)后求泰勒展开时，x 的取值范围是(-1,1)，所以利用 taylor 展开并收敛到无穷级数时，t 的范围应该在（0，2）。因此为了先利用了 ln(ab) = ln(a) + ln(b)的对数性质，不断迭代，把要算的 t 的对数化为计算另一个数的对数，在迭代过程中，这个数不断减小，现实降到<2， 然后降到<1.25、<1.024、<1.01、<1.001、<1.0001，这样一来，所要求的 taylor 展开就变成了求一个<1.0001 的数 p 的自然对数。由于仅当比这几个标识数大时才继续迭代，每次减小的结果仍然大于 1，因此最终所得到的数 p 仍然大于 1，但它又小于 1.0001（假如迭代过程中没有遇到恰好等于标识数的情况），因此 在式

ln(p) = - ln[1 - (1 - 1/p)]

中，

p∈(1,1.0001)， x = 1 - 1/p > 0 且 x --> 0， 可以算出 x < 1 - 1/1.0001 < 10^{-4}

从而 在式

- ln(1 - x) = - (- (x + x^2 / 2 + x^3 / 3 + ...)) = x + x^2 / 2 + x^3 / 3 + ...

中，

由于规定了精确到小数点后 n 位，程序中的处理办法是当向后加的项 $x^k/k < 10^{-(n+1)}$时就不加了，n 取 30 时（精确到小数点后 30 位），前面得到 x < 10^{-4} ，所以 k 最多 = 8 时，就不用加了，即最多只要加到前 7 项。

这里加快计算速度的关键是在于之前的迭代减小过程，拿最大的 100 来举例，

ln 100 = ln2 + ln2 + ln2 + ln2 + ln2 + ln2 + ln1.25 + ln1.25

已经得到结果，不具代表性。

换用较大的随机5位有效数字99.999来举例，

$\ln 99.999 = 6\ln(2) + \ln(1.25) + 9\ln(1.024) + 9\ln(1.001) + 6\ln(1.0001) + \ln(u)$

其中 $1 < u < 1.0001$，可见迭代了 6+1+9+9+6+1 = 32 次，前 31 次都只是比较大小然后做一次乘法做一次加法，速度还是很快的。最后的趋近于 1 的数的自然对数由上面推导可以看出最多算 7 项就能得到精确到小数点后 30 位的结果，7 项当中，乘幂中的乘法一共是 2+3+4+5+6+7 = 27 次（包括与倒数相乘），求倒数一共是 6 次，加法一共是 6 次。

2 部分结合起来，一共加法不超过 31+6 = 37 次，乘法不超过 31+27 = 58 次，求倒数不超过 6 次。其中求 $\ln(u)$ 一共花费 6 次加法，27 次乘法，6 次倒数。综合看来，其收敛速度还是很快的，计算速度也非常可观，在笔者的电脑上不用 1s，小数点后 30 位就可以算出。

注：这里以及下面 2 种方法所讨论的加法、乘法、倒数的求取等，都是基于大数类本身的函数。

数值积分：

由于

$$ln(x) = \int_1^x \frac{1}{t} dt$$

故把区间 $[1, x]$ 分成 n 等份，在每个子区间 $[x_k, x_{k+1}]$ 上采用辛普森公式，最终用的是复合辛普森求积公式

$$Sn = \frac{h}{6}\ [f(1) +\ 4\sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right)\ +\ 2\sum_{k=1}^{n-1} f(x_k)\ +\ f(x)\ ]$$

其中 $f(x) = \frac{1}{x}$， 与 taylor 展开一样，先用迭代把要求的 $\ln(t)$ 的 t 压缩到 $(1, 1.0001)$ 内，$[1, x]$ 整个的长度 $<10^{-4}$，然后再用插值积分，进一步减小误差。

精确到小数点后 30 位时多次实验得出一般要 100 个等分段，进而以 n=100 估计计算代价如下：

先算出约 100 个插值节点，要 100 次乘法和 100 次加法（$x_i = i * h + 1$）；

同样算 $x_{k+\frac{1}{2}}$ 的时候，也要 100 次乘法和 100 次加法；

然后是上式的中间 2 部分，每一部分都要算 100 个倒数，再加到一起，最后乘以 2 或 4，加到总和上，一共主要是 200 次倒数和 200 次加法。

最后的乘以 $\frac{h}{6}$ 可以忽略不计。

综合起来，除去和泰勒展开一样的迭代部分，还需要 400 次加法、200 次乘法、200 次倒数。复杂度也不是很大，在本人电脑上计算到小数点后 30 位时间在 5s 以内，但是收敛速度比起泰勒展开较慢。

连分式：

参考了维基百科上的自然对数（要翻墙），使用了如下图的连分数的函数来逼近 ln(x)：

连分数 [编辑]

尽管自然对数没有简单的连分数，但有一些广义连分数如：

$$\ln(1+x) = \frac{x^1}{1} - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots$$
$$= \cfrac{x}{1 - 0x + \cfrac{1^2x}{2 - 1x + \cfrac{2^2x}{3 - 2x + \cfrac{3^2x}{4 - 3x + \cfrac{4^2x}{5 - 4x + \ddots}}}}}$$

其中右下角的那一部分无限迭代，并不能无止尽地算下去，又发现当 x→0 时，其分式值趋近于 0。因此，同样地利用前述 2 种方法的“预处理”方法，得到一个在(1,1.0001)的 t，因此其对应的 x∈(0,0.0001)，从而可以从某一个较大的 n 开始往前算，第 n+1 个分式近似的取为 0，所以最后一个分母取为

n - (n - 1) * x

然后依次从右下角算到左上角，逼近出 ln(t)的值。

在[1,100]内输入最多 5 位有效数字时，多次尝试发现只需要 n = 7 就可以实现精确到小数点后 30 位，此时最后一个分母(lastNum)为 7-6x，然后依次往前，从后一个 lastNum 算到前一个 lastNum 时，要经历以下步骤：先取倒数;然后乘 i^2x，是 3 次乘法;再加上 i，减去(i-1)x，减法看成加法，是 2 次加法，1 次乘法。整个一轮一共 2 次加法，4 次乘法，1 次倒数。

I 一开始是 n-1，为 6，最后一轮 i 为 1，一共 6 轮，12 次加法，24 次乘法，6 次求倒数。最后再取个倒数，乘以 x。

综合起来，除去和泰勒展开以及数值积分一样的预处理的迭代部分，连分数算法精确小数点后 30 位还需要 12 次加法，25 次乘法，7 次求倒数，比泰勒展开相当，都比用复合辛普森公式的数值积分算法快不少，在本人的电脑上运行也几乎不到 1s，收敛速度很快。

误差分析：

讨论算法的舍入误差：

具体的 3 种算法上面都已经详细说明，现声明：在本人的程序中，大数的加法、减法、乘法、乘幂都是没有舍入误差的，有几位就存几位，仅在求倒数的 getReciprocal 中存在舍入误差，但是其函数有一个参数 n 代表求取的结果的有效数字的个数（在后面用到的地方，其倒数都是纯小数（小于 1），所以小数位数至少取到 3 前 31 位了），可以自行控制，这样当把 n 设置为 31 时，用 getFirstNBits 函数的四舍五入求其前 30 位的有效数字时就只有 $0.5*10^{-30}$ 的舍入误差了。

另外在 3 种算法的“预处理”迭代部分中，已经对若干标识数的自然对数做了大于 40 位的截断赋值，与求倒数的 10^{-30} 相比，可以忽略不计。

由于要求中说明除法对结果的误差可以忽略不计，因此上述倒数的舍入误差对结果的影响不在此讨论。

讨论算法的截断误差：

泰勒展开截断误差分析：

由于我用的泰勒展开最后形式如下：

$- \ln(1 - x) = - (- (x + x^2 / 2 + x^3 / 3 + ...)) = x + x^2 / 2 + x^3 / 3 + ...$

上面讲到 $x < 10^{-4}$，而我 lnTaylor 函数中的选定精度的参数设置的是 30，指当 $x^n/n < 10^{-31}$ 后就不加了，此后则截断误差为

$x^n/n + x^{n+1}/(n+1) + x^{n+2}/(n+2) + \cdots$

$< x^n/n(1 + x^2 + x^3 + \cdots)$

$< x^n/n * \frac{1}{1-x}$

$< 10^{-31} * 2$

$< 0.2 * 10^{-30}$

数值积分截断误差分析：

由复合辛普森求积公式余项公式，$b-a < 1.0001 - 1 < 10^{-4}$，分为 100 段时其 $h < 10^{-4}/100$，得：

$|Rn(f)| = \frac{b-a}{180} * \left(\frac{h}{2}\right)^4 * f^{(4)}(\eta)$， ($f(x) = 1/x$ ， $1 < x < 1.0001$，$1 < \eta < x$)

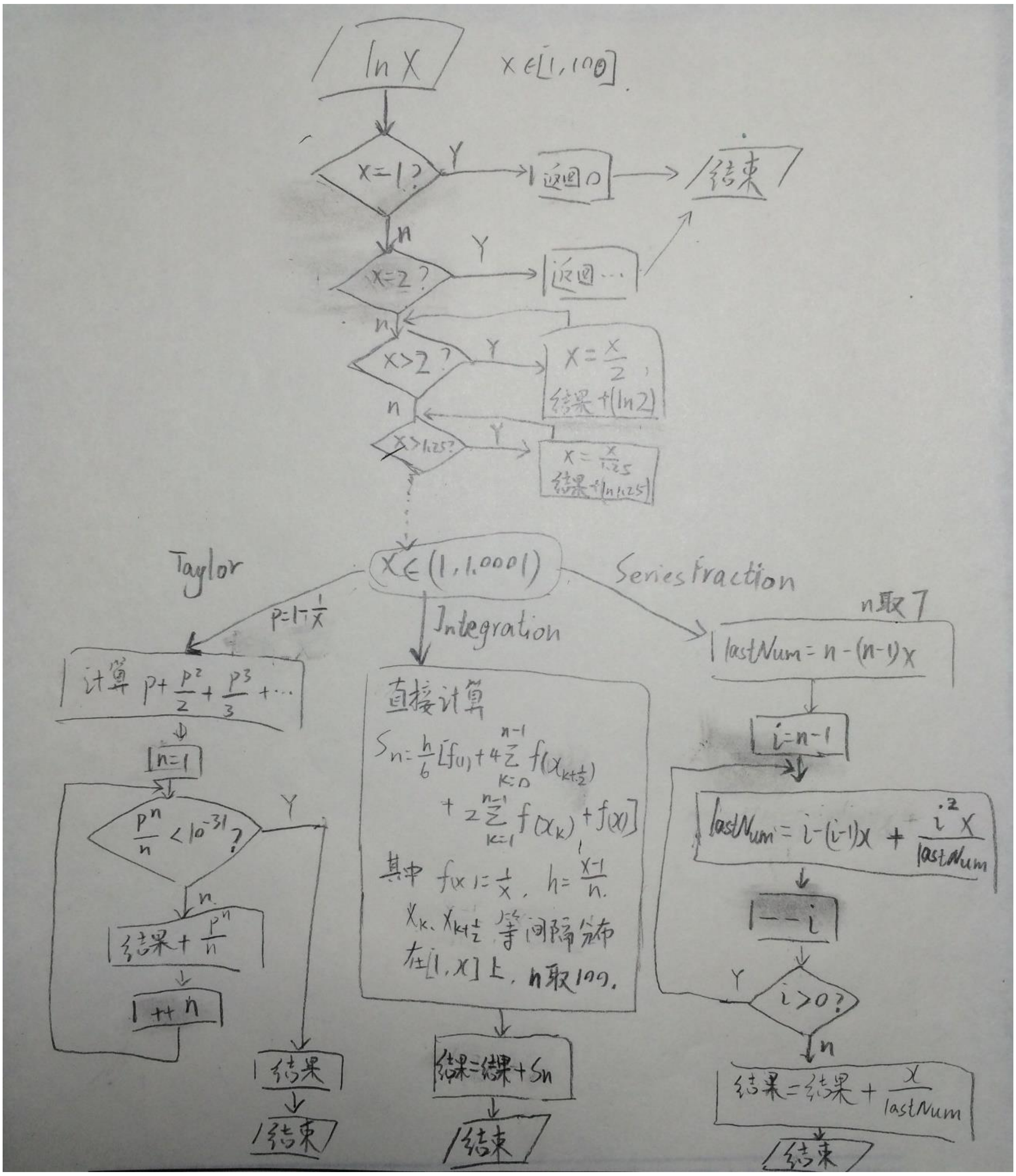
$< 10^{-4}/200 * (10^{-6}/2)^4 * 1/(24\eta^5)$

$< 10^{-30}/(32*24)$

$< 10^{-32}$

连分数截断误差由于涉及到大量除法，不便于分析，所以暂时略去。

程序框图：


ln X
x ∈[1, 100]
x=1?
Y
n
返回0
结束
x=2?
返回…
x>2?
x = x/2 ,
结果+ln2
x>1.25?
x = x/1.25
结果+ln1.25
x∈(1, 1.0001)
Taylor
p=1-1/x
计算 p+p²/2+p³/3+…
n=1
pⁿ/n < 10⁻³¹ ?
结果+pⁿ/n
++n
结果
Integration
直接计算
其中 f(x)=1/x, h=(x-1)/n.
n取100.
结果=结果+Sn
Series fraction
n取7
lastNum = n-(n-1)x
i=n-1
lastNum = i-(i-1)x + i²x/lastNum
--i
i>0?
结果=结果+x/lastNum

实验感悟：

本次实验从前到后花了不少时间，大数类的四则运算比较繁琐，从写的时候就时而出现一些 bug，到最后运用到核心算法中时又多次发现小的 bug，比如纯小数最前面的 0 在什么时候去掉运算比较方便等。

由于本人本次实验选用了 java 语言，而 java 中没有运算符重载，所以我的大数类的运算都要靠函数实现，有一些不便但也无伤大雅。

另外，从开始思考如何用 taylor 展开求大于 2 的数的自然对数值，到后来思考怎么样在不损失精度的前提下提高运算速度（技巧就是 3 个方法中的预处理部分，迭代减小，给出一些标识数的高位 ln 值），都不断是我有新的发现和惊喜。

总的来说，此次大实验不光大大锻炼了我对 java 语言的驾驭能力，同时也使我对函数逼近以及数值积分的方法有了更深刻的认识，包括各种方法实现起来的效率和收敛速度以及误差等等。这是一次非常好的学习提升经历。

参考网站：

https://zh.wikipedia.org/wiki/%E8%87%AA%E7%84%B6%E5%B0%8D%E6%95%B8