《数值代数》第一次上机作业 实验报告

匡亚明学院 211240021 田铭扬

摘要

笔者使用 C++语言（利用 OpenBLAS 库和 lapack 库）编程，用 Gauss 列主元消元法、Cholesky 分解法、Crout 算法、追赶法等算法，分别解决一个“示例问题”，并藉此讨论了上述算法的运行效率和误差。

正文

前言

线性方程组的求解是线性代数中最基本的需求之一，因而也成为了数值计算的重要课题，其中诞生了 Gauss 消元法（以及列主元 Gauss 消元法和 Gauss-Jordan 消元法）、Cholesky 分解法（LL^T与 LDL^T算法）、LU 分解法（Crout 算法和 Doolittle 算法，追赶法为其特例）等经典算法。

本次数值实验的目的即是“实机”应用上述其中几个算法解决问题，并对它们的运行效率(CPU 时间)、误差等进行讨论。因为这些算法的经典性，此次实验对于《数值代数》课程的深入学习有重要的意义。

问题

$$T_n=\begin{bmatrix}2&-1&&\\-1&2&\ddots&\\&\ddots&\ddots&-1\\&&-1&2\end{bmatrix}\quad A_{n^2}=\begin{bmatrix}T_n+2I_n&-I_n&&\\-I_n&T_n+2I_n&\ddots&\\&\ddots&\ddots&-I_n\\&&-I_n&T_n+2I_n\end{bmatrix}$$

第 1 题 利用列主元 Gauss 消元法、LL^T 法和 LDL^T 法求解线性方程组 $A_{n^2}\boldsymbol{x}=\boldsymbol{c}$，真解取为 $\boldsymbol{x}=(1,1,\dots,1)^T$，右端向量由利用真解计算出来。

1.1 绘制数值误差同矩阵阶数 n 的关系，其中数值误差采用对数坐标;

1.2 绘制 CPU 时间同 n 的关系;

1.3 绘制矩阵条件数与 n 的关系。请问：摄动理论给出的右式是否完美刻画了相对误差的大小？

$$\boxed{\frac{\|\delta\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty}\le Cn^3\vartheta\eta(\mathbb{A})\kappa_\infty(\mathbb{A})}$$

第 2 题 考虑行列重排后相等的两个 n 阶矩阵

$$B_1=\begin{bmatrix}1&&&a\\&1&&a\\&&\ddots&\vdots\\a&a&\dots&1\end{bmatrix}\quad B_2=\begin{bmatrix}1&a&\dots&a\\a&1&&\\\vdots&&\ddots&\\a&&&1\end{bmatrix}$$

执行相应的 Crout 算法；利用 Matlab 命令 spy() 绘制它们在三角分解后的非零元素分布（或结构图），并比较相应的 CPU 时间。

利用矩阵的元素分布特点，修改 Crout 算法，删除那些无用的运算时间。重复上述操作，观察 CPU 时间是否得到节省？

第 3 题 计算三对角阵 T_n 或块三对角阵 A_{n^2} 的逆矩阵，并观测它们的运行效率（关于 n 的计算复杂度）。

第 4 题 设 $D_n = diag\{2^{-i}\}_{i=1}^{n}$，定义 $\tilde{T}_n = D_n\, T_n$；考虑两个同解的三对角线性方程组 $\tilde{T}_n\, \boldsymbol{x} = \widetilde{\boldsymbol{b}}$，$T_n\, \boldsymbol{x} = \boldsymbol{b}$，其右端项均由真解 $\boldsymbol{x} = (1,1,\ldots,1)^T$ 生成。用追赶法求解它们，观测数值误差同 n 的关系。

程序设计

第 1 题用到的 LL^T 法和 LDL^T 法、第 2 题用到的 Crout 算法、第 3 题用到的 Gauss-Jordan 算法和第 4 题用到追赶法，分别参照讲义（参考文献[1]）P15、P16(右下)、P14、P11 和 P19 的“代码块”进行实现，不再详述。部分算法在讲义中并未完全实现（如追赶法仅实现了矩阵变形，而未实现右端项变形和回代求解；LL^T 及 LDL^T 法仅实现了矩阵分解，而未实现回代求解），笔者进行了相应的补全。此外，讲义中的代码是基于朴素的“点对点”操作实现的，读者利用 OpenBLAS 库，对“axpy”、向量点乘、求向量 2-范数等操作进行了优化。

第 1 题还使用了列主元 Gauss 消元法，笔者在讲义 P2 的 Gauss 算法“代码块”基础上进行改动，实现了此算法。为保证代码易读性，笔者并未使用数组保存行交换情况，而是略微牺牲效率，用 OpenBLAS 库中的向量交换操作实现。

第 2 题要求针对题中矩阵，对 Crout 算法进行特别优化，为保证行文连贯，此部分留待“实验结果分析”段落详述。

最后，第 1 题需要计算矩阵的条件数，使用了 OpenBLAS 库自带指令实现。

实验环境

使用 VMWare 17 虚拟机运行 deepin 20.9（基于 Debian10）操作系统，并为其分配 4 个 CPU 核心及 6GB 内存。

使用 OpenBLAS 库实现的 CBLAS，并使用 lapack 库。

使用 GCC 8.3.0-1 版本编译器，未开启编译优化选项。

实验结果分析

第 1 题

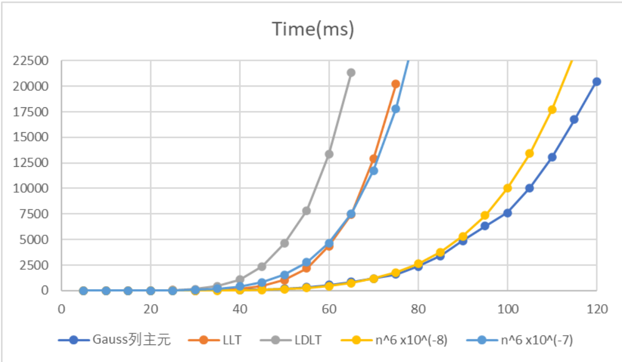
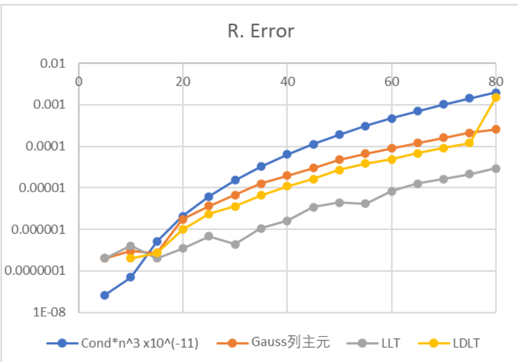


图 1.1 第 1 题运行效率(CPU 时间)图表

如图表所示，三种算法运行耗时均与n^6同阶。注意到矩阵A_{n^2}的阶数为n^2，故实验结果证明了三种算法的时间复杂度与理论一致，均为$O(N^3)$。

注意到 LL^{T}算法较 Gauss 列主元算法慢 10 倍左右，可能是因为 LL^{T}法涉及较慢的开方运算。但是不涉及开方运算的 LDL^{T}算法却比 LL^{T}的表现还差，可能是因为笔者是按照讲义中的“代码块”实现的算法：讲义 P15 中 LL^{T}算法是基于“逐列次序”实现的，笔者使用了 OpenBLAS 库中的向量点乘指令，而该库会对其进行并行优化，故速度较快；讲义 P16 中 LDL^{T}算法是基于“逐行次序”实现的，无法进行这一优化，故速度较慢。预计若基于“逐列次序”重写 LDL^{T}算法并进行相应优化，运行速度将会快于 LL^{T}算法。



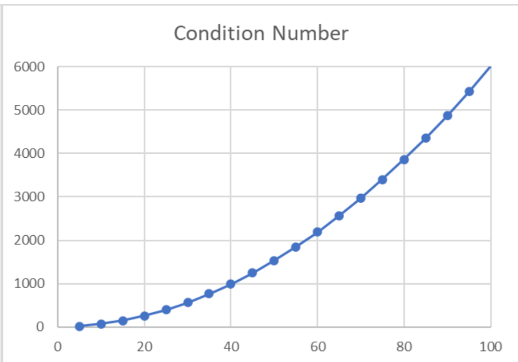


图 1.2 第 1 题相对误差图表-对数坐标 / 图 1.3 第 1 题条件数图表

如图表所示，注意矩阵的阶数为n^2，故三种算法的相对误差均与系数矩阵的条件数$\times$阶数$^{1.5}$同阶。由于课上并未学习主元增长因子的相关内容，可以“反向”讨论第 3 小问，即：假设公式 $\frac{\|\delta \boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} \le Cn^3\vartheta\eta(\mathbb{A})\kappa_\infty(\mathbb{A})$ 成立，则矩阵A_{n^2}的主元增长因子$\eta(A_{n^2}) \approx n^{-3}$。

第 2 题

对两个矩阵，分别取 a=0.001 与 a=1000 执行 Crout 算法，运行时间如下表：

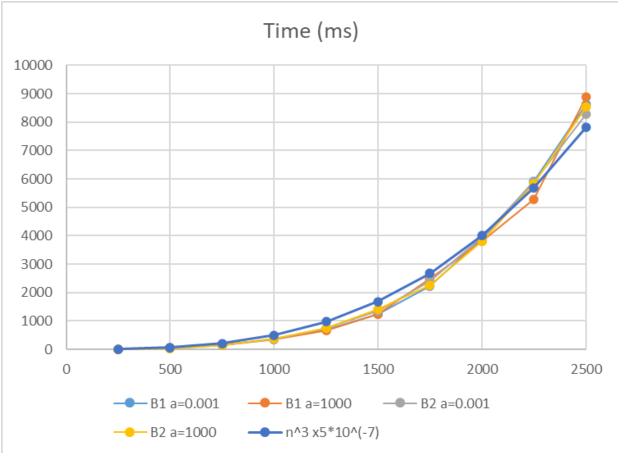


图 2.1 第 2 题运行效率(CPU 时间)图表

如表所示，Crout 算法对两个矩阵、两种 a 的取值的运行效率相近，且均与n^3同阶，与理论相符。而为了观察两者分解后“0”的分布情况，取 n=20, a=10 并输出执行 Crout 算法后的结果，得到如下图所示的输出：

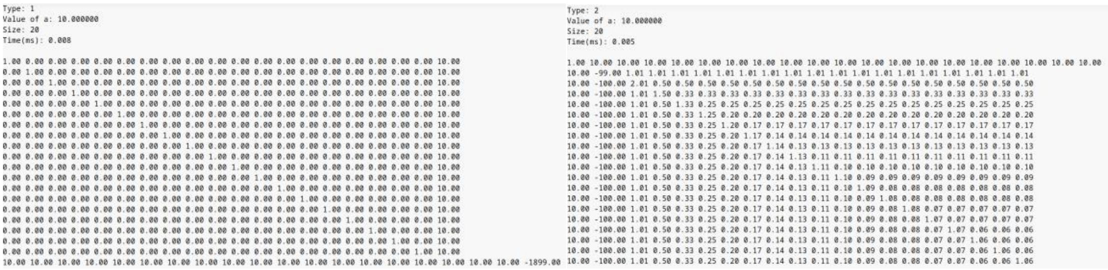


图 2.2/2.3 B_1 及 B_2 矩阵分解后的 0 元素分布

注意到 B_1 矩阵分解后，除对角线元素与最后一行、最后一列元素外全为 0，而 B_2 矩阵分解后几乎没有 0 元素。故可针对 B_1 进行优化：右图是讲义中的伪代码，第 3 行改为当且仅当(i==k||i==n||k==n)时才执行，否则置 a_{ik}=0，第 7 行同理。如此优化后运行结果如下表，实际的时间复杂度介于$O(N)$与$O(N^2)$之间。

1. For $k = 1, 2, \ldots, n$, Do
2. For $i = k, k+1, \ldots, n$, Do
3. $a_{ik} := a_{ik} - \sum_{r=1}^{k-1} a_{ir} a_{rk}$;
4. Enddo
5. For $j = k+1, k+2, \ldots, n$, Do
6. $a_{kj} := (a_{kj} - \sum_{r=1}^{k-1} a_{kr} a_{rj}) / a_{kk}$;
7. Enddo
8. Enddo

Size	500	1000	1500	2000	2500
未优化	50.873	381.984	1240.887	3908.282	8619.499
优化	0.212	1.037	4.992	23.328	49.293

表 2.4 B_1 优化前后对比（取 a=0.001）

第 3 题

由于矩阵T_n阶数较少，效果不明显，笔者在此选用A_{n^2}来测试 Gauss-Jordan 算法求逆矩阵的效率。运行结果如下：

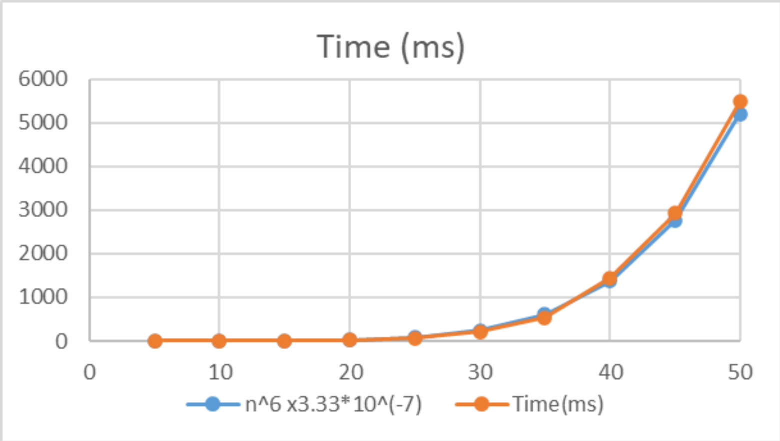


图 3.1 第 3 题运行效率(CPU 时间)图表

如图表所示，运行时间与n^6同阶，即时间复杂度为$O(N^3)$，与理论相符。

第 4 题

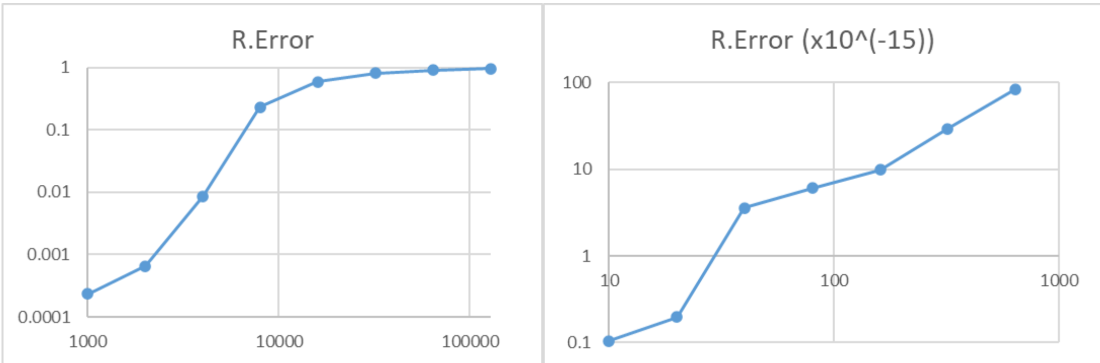


图 4.1/4.2 第 4 题误差图表-双对数坐标

上左是追赶法求解 $T_n\,\boldsymbol{x}=\boldsymbol{b}$ 的误差图表，上右是求解 $\tilde{T}_n\,\boldsymbol{x}=\tilde{\boldsymbol{b}}$ 的误差图表

注意到由于T_n不是严格对角占优矩阵，追赶法并不稳定，在阶数较大时相对误差接近 1，表现很差。

另一方面，$\tilde{T}_n$是对角占优的，但由于其中元素绝对值较小，笔者不得已使用 double 变量（前文所有实验均使用 float 类型变量）以保证矩阵元素不会虽阶数增长很快低于机器精度。但即便如此，仍然不能得到阶数很大时的数据，且使用 double 类型会导致相对误差大幅降低，不具有可比性。这是本次实验的遗憾，留待以后的实验中改进。

结语

在本次数值实验中，笔者使用列主元 Gauss 消元法、LL^T法和 LDL^T法、Crout 算法、Gauss-Jordan 算法和追赶法，对示例问题进行了求解与分析，锻炼了笔者的实践能力，且极大地加深了笔者对于上述算法的理解。

实验代码

由于篇幅限制，代码不在实验报告中列出，可以在笔者 github 仓库中查看。网址为：https://github.com/lk758tmy/NA2-Codes

参考文献

[1]《数值代数》讲义. 张强

[2]数值计算方法-上册. 林成森. 科学出版社. 2005-1 第二版