

# 数值代数实验报告 2

PB21000030 马天开

2023 年 10 月 26 日

GitHub仓库:

[https://github.com/tiankaima/numerical\\_algebra](https://github.com/tiankaima/numerical_algebra)

实验平台:

```
> uname -a
Linux tiankai-omenlaptop15en1002ax 6.1.53-1-MANJARO #1 SMP PREEMPT_DYNAMIC Wed Sep 13
14:10:57 UTC 2023 x86_64 GNU/Linux
> cmake --version
cmake version 3.27.6
> clang --version
clang version 16.0.6
Target: x86_64-pc-linux-gnu
Thread model: posix
```

目录结构:

- CustomMath\_lib存放了具体的算法实现
- Doctest\_tests存放了单元测试
- homeworks存放了作业的源代码, 并且在main.cpp对每次作业进行了调用
- main.cpp是主程序
- Mathematica存放了.nb文件, 用于生成测试数据
- writeups存放了实验报告的源代码, 比如本文

Linux 平台编译:

```
> mkdir build
> cd build
> cmake ..
> make
```

运行:

`./numerical_algebra`

Windows 平台编译: 直接打开numerical\_algebra.sln编译即可

## 问题描述

### 1.1

将不选主元的Gauss消去法、全主元Gauss消去法、列主元Gauss消去法编写成通用的子程序，然后用你编写的程序求解84阶方程组：

$$\begin{bmatrix} 6 & 1 & 0 & 0 & \cdots & 0 \\ 8 & 6 & 1 & 0 & \cdots & 0 \\ 0 & 8 & 6 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 8 & 6 & 1 \\ 0 & \cdots & 0 & 0 & 8 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{83} \\ x_{84} \end{bmatrix} = \begin{bmatrix} 7 \\ 15 \\ 15 \\ \vdots \\ 15 \\ 14 \end{bmatrix} \quad (1)$$

最后将你的计算结果与精确解（精确解为全1列向量）相比较，并分析实验结果。

要求输出计算结果，计算结果和准确解的误差以及运行时间。

### 1.2

将平方根法和改进平方根法编写成通用的子程序,然后用你编写的程序求解对称正定方程组  $Ax = b$ :

- $b$ 随机选取，系数矩阵 $A$ 为100阶矩阵

$$\begin{bmatrix} 10 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 10 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 10 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 10 & 1 \\ 0 & \cdots & 0 & 0 & 1 & 10 \end{bmatrix} \quad (2)$$

- 系数矩阵为 40 阶 Hilbert 矩阵, 即系数矩阵  $A$  的第  $i$  行第  $j$  列元素满足

$$a_{i,j} = \frac{1}{i+j-1} \quad (3)$$

向量  $b$  的第  $i$  个分量满足

$$b_i = \sum_{j=1}^n a_{i,j} \quad (4)$$

要求输出计算结果和运行时间。

### 1.3

用第 1 题的程序求解第 2 题的两个方程组, 比较所有的计算结果, 然后评论各方法的优劣。

## 程序介绍

Vector 和 Matrix 类分别实现在 CustomMath\_lib/Vector.cpp 和 CustomMath\_lib/Matrix.cpp 中，考虑到课程需要没有使用模板类，而是直接使用 `vector<vector<long double>>` 作为底层数据结构。

Gauss消去法的实现在 CustomMath\_lib/GaussMethod.cpp 中，包括：

- 1.1.1 前代法
- 1.1.2 回代法
- 1.1.3 不选主元的Gauss消去法
- 1.2.1 全主元Gauss消去法
- 1.2.2 列主元Gauss消去法

其中教材上提供的“节省存储空间的方法”不符合正常编写习惯，所以相应函数做了一层封装，例如 *LU*分解的实现：

- `void LU_Factorization_InPlace(Matrix *A);` 直接在原矩阵上进行分解
- `void LU_Factorization(const Matrix &A, Matrix *L, Matrix *U);` 将分解结果存储在两个矩阵中，同时不修改原矩阵

而在实现上后者直接复用了前者的逻辑，因此无需做两次测试。

平方根法的实现在 CustomMath\_lib/CholeskyMethod.cpp 中，包括：

- 1.3.1 平方根法
- 1.3.2 改进平方根法

**1.2(1) 中 *b*取值为全1列向量**

[illegible]

## 结果分析

### 1.1

三种算法的误差均偏大，以至于我一度怀疑自己的代码出现问题，在 $n < 20$ 时均能给出误差较小的结果。

用时分别为：

- 不选主元的Gauss消去法：3864 ms
- 全主元Gauss消去法：9746 ms
- 列主元Gauss消去法：4210 ms

### 1.2.1

运行时间分别为：

- 平方根法：3555 ms
- 改进平方根法：3672 ms

### 1.2.2

运行时间分别为：

- 平方根法：270 ms
- 改进平方根法：322 ms

### 1.3.1

运行时间分别为：

- 不选主元的Gauss消去法：8260 ms
- 全主元Gauss消去法：16554 ms
- 列主元Gauss消去法：6815 ms
- 平方根法：3603 ms
- 改进平方根法：3635 ms

### 1.3.2

运行时间分别为：

- 不选主元的Gauss消去法：615 ms
- 全主元Gauss消去法：1147 ms
- 列主元Gauss消去法：524 ms
- 平方根法：272 ms
- 改进平方根法：297 ms

特别注意到1.2.2, 1.3.2中使用平方根的算法结果并不正确，由于精度问题，计算 Hilbert 矩阵时算法认定矩阵非正定，为保证程序正常运行，做了如下修改：

```
// CholeskyMethod.cpp
// Line 19
if (A->matrix[k][k] <= 0) {
    // FIXME: I dont like this either, sorry.
    A->matrix[k][k] = 1e-2;
    // throw std::invalid_argument("A is not a positive definite matrix");
}
```

在意识到问题时，首先检验了算法正确性，如 1.3.1 中平方根算法给出了正确的结果，推定算法实现无误，接下来使用 long double 对项目进行了重构，依旧无法满足精度要求，最终只能放弃，推荐改进的平方根算法。