# ACSE 5 Assignment Report

Team Awosile - Xinyu Xiong(xx4518), Yue Wang(yw2619), Tianzong Yu(ty616)

Saturday 1st February, 2020

## 1 Design

We have used altogether four linear solvers, **Gauss-Seidel**, **Cholesky factorisation(LDL)**, **Jacobi** and **Successive over-relaxation(SOR)**for both dense and sparse matrices. Since we built our methods inside the Matrix class and CSMatrix class, the inputs are simply $Matrix < T > \&b$ and $Matrix < double > \&res$ where $b$ is the right hand-side vector and $res$ is the solution for this linear system. We include $res$ as an input is due to the fact that we cannot delete a variable that we need to return. For all functions that we implemented, there is no variable to return. In order to show the result, we deploy the $printMatrix()$ method to print the solution.
Besides the three solvers, we have implemented other functions to help with our solvers:

- `virtual void CSRMatrix<T>::pos_def(int n)` [1]

  `virtual void Matrix<T>::pos_def(int n)` [2]

  These two are responsible for generating random positive definite matrices where $n$ is the rank of the matrix .

- `T MAX(T *a, int n)` [3]
  This is method is put in $Matrix.cpp$, however, this is not in the Matrix class. It is built to compute the infinity norm of a matrix.

- `Matrix<T>: void random_full()` [4]
  This method fills the matrix with random value

### 1.1 Gauss-Seidel

The approach is to get to the correct answer as close as possible by iteration. As long as it converges, the program will stop. Due to the algorithm itself, it requires that **the diagonal of the positive definite matrix cannot contain** $0$.

### 1.2 Cholesky factorisation(LDL)

The original linear system we want to solve is $Ax = b$. We can decompose $A$ into $LDL^T = A$ and we can solve it by using substitution. Let $y = DL^T$ and thus we can solve $Ly = b$. Due to the algorithm itself, it requires that **the diagonal of the positive definite matrix cannot contain** $0$.

### 1.3 Jacobi

This linear solver is quite similar to Gauss-Seidel. The only difference is that Jacobi calculate the current iteration's answer from the previous iteration while Gauss-Seidel uses the most up-to-date values during the iterative procedures.
The other very important notice is that **Jacobi cannot find the answers for all positive definite matrices.** For the matrices that are not diagonal dominant, the solution would diverge and thus can never reach the right answer. In our method, in order to deal with this situation, we created a variable that monitors the difference between result of the $k_{th}$ iteration and the $k - 1_{th}$ generation. If the difference is greater than 10 times, then the programs stops and output a message telling the user that Jacobi is not suitable for this matrix and please user other linear solvers instead. Due to the algorithm itself, it requires that **the diagonal of the positive definite matrix cannot contain** $0$.

### 1.4 Successive over-relaxation(SOR)

This is a modified version of Gauss-Seidel method with a faster convergence. The whole process is almost the same except a relaxation parameter, $\omega \in (0, 2)$. Instead of updating the solution with $\frac{1}{a_{ii}}(b_i - \sigma)$, SOR uses $(1-\omega)x_i + \frac{\omega}{a_{ii}}(b_i - \sigma)$. The choice of $\omega$ is quite important as it directly affects the speed of convergence. In our implementation, we did not give the choice of $\omega$ to the user, instead, we set it as $\omega = 1.4$. Due to the algorithm itself, it requires that **the diagonal of the positive definite matrix cannot contain** $0$.

## 2 User Instructions

All of our code are written in C++ and can be viewed on `https://github.com/acse-2019/acse-5-assignment-awosile`. Detailed user instructions on how to execute it are explained in the README.md file.

## 3 Work BreakDown

- Yue Wang: Cholesky Factorisation(LDL) and Successive over-relaxation(SOR) for both sparse and dense matrices; main method; clean up code
- Tianzong Yu: Gauss-Seidel for both sparse and dense matrices; Report; README.md
- Xinyu Xiong: Jacobi for both sparse and dense matrices; main method; Report; README.md

## 4 Strengths & Weakness

### 4.1 Speed Comparison

In order to compare the efficiency of different solvers, we compute the result with different sizes of input matrices. The graph is shown in Figure 1 and 2. As we can observe from the graphs, Jacobi and Gauss have almost the same speed as we expected. Theoretically, both of them have a time complexity of $O(n^2)$ while LDL is $O(\frac{1}{3}n^3)$. SOR is $O(n^2)$ but theoretically faster than Gauss method. The reason that SOR behaves differently for sparse and dense matrices are due to $\omega$. The optimal value of $\omega$ depends upon the properties of the coefficient matrix, however, we set it to be the same for both test cases. This could explain the pattern observed different from expected. The equation to calculate $\omega$ is

$$1 + \left(\frac{\mu}{1 + \sqrt{1 - \mu^2}}\right)^2 \tag{1}$$

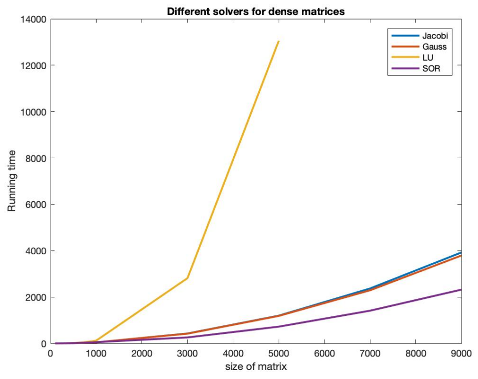

Figure 1: Speed Comparison graph among all solvers we implemented for dense matrices

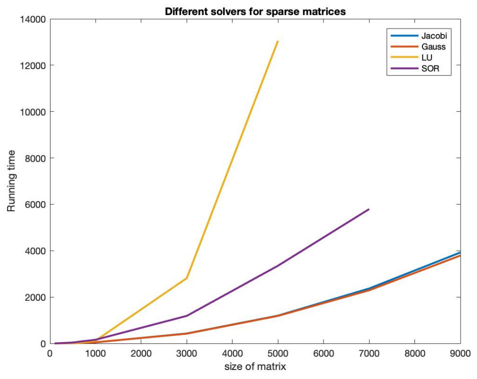

Figure 2: Speed Comparison graph among all solvers we implemented for sparse matrices

where $\mu$ is the spectral radius of $I - D^{-1}A$. It is too expensive to compute, thus we just take the average.

The reason that there is no data for LDL at the rank larger than 3000 is due to our implementation.We need to allocate extra space for L and D matrix. Thus, we need twice the space than usual. However, the space that is allocated to C++ is fixed, the RAM runs out of space if the size of the matrix is too big.

## 4.2 General Comparison

| | Strengths | Weaknesses |
|---|---|---|
| Jacobi | Simple;<br>Less time consumed;<br>Increased convergence rate if preconditioner used. | Not as accurate;<br><br>Spectral radius $< 1$ required. |
| Gauss-Seidel (GS) | Simple;<br>Less memory occupation;<br>Less computation per iteration;<br>Increased convergence rate if preconditioner used. | Positive-definite & Symmetric $A$ required;<br>Slow rate of iteration;<br>No 0 on diagonal required;<br>Spectral radius $< 1$ required. |
| Cholesky Factorisation (LDL) | High stability;<br>High accuracy;<br>High computational efficiency. | Positive-definite & symmetric $A$ required;<br>No 0 on diagonal required;<br>More time taken. |
| Successive-over-Relaxation (SOR) | More Accurate than GS;<br>High accuracy;<br>Faster than GS. | Good selection of $\omega$ relaxation parameter required otherwise unstable;<br>No 0 on diagonal required;<br>Spectral radius $< 1$ required. |