Linear Solvers Assignment Report

Longyun Liao ll2819@ic.ac.uk Qingyang Li ql919@ic.ac.uk Qing Ma qm319@ic.ac.uk

<https://github.com/acse-2019/acse-5-assignment-3_static_variables>

# Introduction

We implemented algorithms to solve the linear system Ax=b, where A is a positive definite matrix, and x and b. In our software, the Matrix A can be both stored as dense and sparse format. Our solvers can be classified into two groups, one is direct solver group and the other is a group of iterative solves. Jacobi, Gauss Seidel as iterative methods, LU decomposition and Cholesky factorization as direct methods have been applied in both dense and sparse matrix. Convergence of the Gauss-Seidel method can be improved by a technique known as successive over relaxation (SOR) and this technique is implemented in dense matrix.

# Background

* **Backward substitution:**

For an upper triangular matrix **U** whose elements are denoted by with a right-hand vector **y** whose components are denoted by , the solution vector **x** whose components are denoted by , the backward substitution algorithm can be represented as ((Algowiki-project.org, 2020)

* **Forward substitution:**

For a lower triangular matrix **L** (with entries ) and a right-hand side vector **b** (with components ), the solution vector **y** whose components are denoted by , the forward substitution algorithm can be represented as (Algowiki-project.org, 2020)

Note: Backward and forward substitution are extended versions of Gauss Elimination.

* **LU decomposition:**

Thus if ***A = LU*** then: **.** Since we add partial pivoting in our program, the linear system then becomes , .

* **Cholesky factorization:**

For any symmetrical positive definite (SPD) matrix A (in the field of real numbers, positive definite matrix is symmetrical), there is always a decomposition method which takes the forms that: (U is a upper triangle matrix).

To derive the upper triangle matrix, we could take the following steps:1) compute the first entry in the U:; 2) compute the first row in the U: ; 3) compute the elements on the diagonal: Assume we have already known the entries above the th row, then we can get the entries on the th row ;4) compute the elements on the diagonal: .We have to compute the firstly: . Then: . And when : , thus: ; 5) transpose to get the lower triangle matrix ; 6) then the initial problem is converted to , we can use the similar method in the LU solve section to solve the linear problem.(Mathforcollege.com, 2020)

* **Jacobi method:**

For a matrix system like can be rewritten by pulling out the term involving  (i.e. for each row 𝑖 pull out the diagonal from the summation):

, then we can get .

We can use an *iterative method* which can’t directly give us the numerical solution for *implicit* problems. We can start from a guess at the solution , then iterate for 𝑘>0:

Note that for this iteration, for a fixed 𝑘, it does not matter in which order we perform the operations looping over as the right-hand side only contains the entries of ***𝑥*** at the previous iteration.

* **Gauss Seidel method:**

We can make a small improvement to Jacobi’s method using the updated components of the solution vector as soon as they become available.

Note that opposed to Jacobi, we can overwrite the entries of ***x*** as they are updated. as we are using updated knowledge immediately, the Gauss-Seidel algorithm should converge faster than Jacobi, but note that this convergence can only be guaranteed for matrices which are diagonally dominant..

* **Gauss Seidel method with SOR:**

Convergence of the Gauss-Seidel method can be improved by a technique known as successive over relaxation (SOR): . 𝜔 cannot be determined beforehand for an arbitrary system, however, an estimate can be computed during run time. Let be the magnitude of the change in x during the iteration. If 𝑘 is sufficiently large (say 𝑘≥5), it can be shown that an approximation of the optimal value .

# Software Structure

The software contains three major part, CSRMatrix, Matrix and Main. CSRMatrix class is inherited from Matrix. All of solvers for dense Matrix with the needed functions are included in Matrix.cpp file, while solvers for sparse matrix are included in CSRMatrix.cpp file. Matrix have Jacobi, Gauss Seidel, Gauss Seidel with SOR, LU decomposition and Cholesky Factorization solvers. CSRMatrix have Jacobi, Gauss Seidel, LU decomposition and Cholesky Factorization solvers.

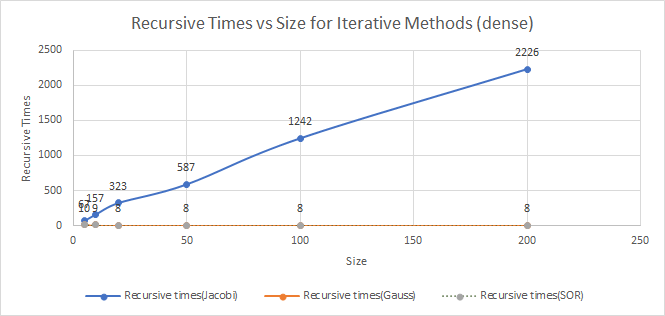
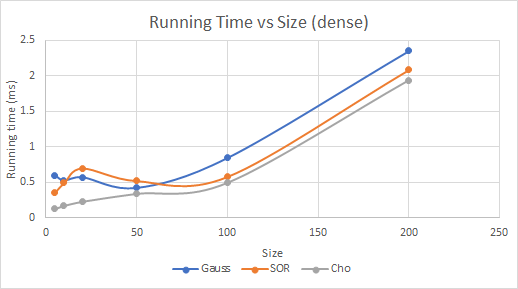
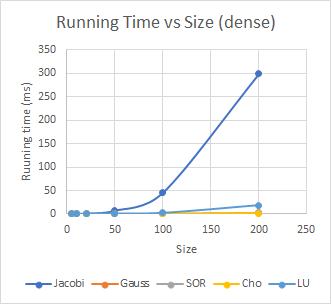
Main file has functions to read user input file and solve x. To demonstrate our process of testing, test\_dense() function is also included in main file.

# Using the Software

Please refer to README.md on our Github page.

<https://github.com/acse-2019/acse-5-assignment-3_static_variables>

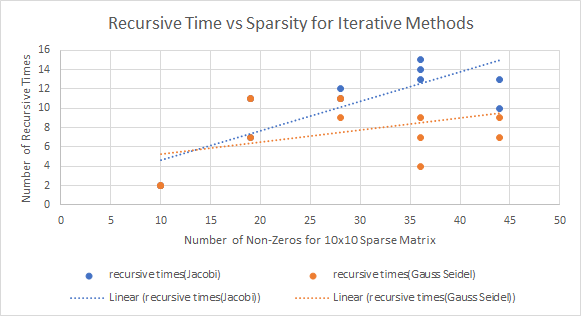
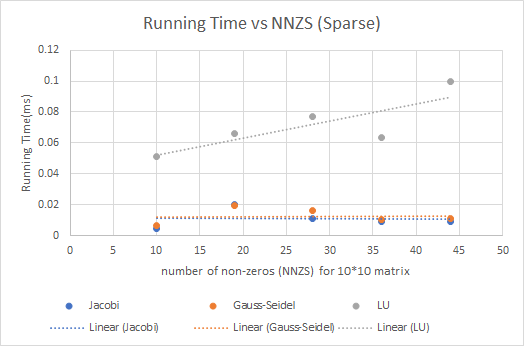
# Performance Analysis



For dense matrix, all of the methods cost more time with larger size. Among these methods, Jacobi grows significantly faster than others, this is probably due to the fact that Jacobi converges slower than Gauss Seidel, which also results an extremely larger recursive times than other two iterative methods.

Among direct solving methods, Cholesky factorization performs better than LU. The reason of the difference is because partial pivoting is included in our methods, which means that rows swapping happens frequently.

In our testing cases, the running time of Gauss Seidel, Gauss Seidel with SOR, and Cholesky factorization are close.

For sparse matrix solvers, the running time of all of the methods are increasing with growth of number of non-zeroes at size 10x10. Among iterative methods, all recursive times are getting larger. In all testing cases, LU decomposition stays as the most time-consuming solver which is probably due to the frequent row swapping operations. Jacobi and Gauss have similar performance. (Note: the accuracy for iteration methods is 0.000001)

# Limitations and Further Work

**For LU decomposition method**, solving requires storing in memory the LU factors and around n^3/3 flops. Besides, it needs (like most) pivoting to avoid the case when diagonal element is zero. According to the above reasons, LU decomposition is not a memory friendly method nor a way to save computation time. Although when implementing LU for sparse matrix, CSRMatrix format reduce memory usage, the computations regarding to rows swap increase significantly. **Cholesky Factorization method,** which is an efficient way to solve symmetrical positive matrix related problems, is also a good way to judge if a matrix is positive definite, especially for sparse matrices. We have used both Cholesky Factorization and determinant to perform the test. Our codes implementing Cholesky Factorization method to solve sparse matrices didn’t perform accurately for large sparse matrices. **For SOR method,** we haven’t tried larger size of matrix or smaller precision, both of which might have more significantly improvement on recursive times.

If we had more time, we could have explored the **BLAS/LAPACK library.** Functions like row swapping may be called directly from those libraries. For now, all of our functions are included in Matrix.cpp and CSRMatrix.cpp, which results that the content of files looks redundant. In future, we could arrange the major part of our **solves in a separate solvers.cpp file and leave the get\_transpose, find\_max\_index etc. in Matrix.cpp and CSRMatrix.cpp.**

# Work Distribution

|  |  |
| --- | --- |
| Longyun Liao | LU Decomposition with backward and forward substitution for both sparse and dense matrix. |
| Qingyang Li | Iterative methods for both sparse (Jacobi, Gauss-Seidel, SOR) and dense matrix (Jacobi and Gauss-Seidel); Read matrix from user. |
| Qing Ma | Cholesky factorization for both sparse and dense matrix; Check Positive Definiteness of the input matrix. |

# References

Algowiki-project.org. (2020). *Forward substitution - Algowiki*. [online] Available at: https://algowiki-project.org/en/Forward\_substitution [Accessed 2 Feb. 2020].

Algowiki-project.org. (2020). *Backward substitution - Algowiki*. [online] Available at: https://algowiki-project.org/en/Backward\_substitution [Accessed 2 Feb. 2020].

Mathforcollege.com. (2020). Mathematics for College Students: Open Courseware. [online] Available at: http://mathforcollege.com [Accessed 2 Feb. 2020].