Code Design

See README.md for User instruction: https://github.com/acse-2019/acse-5-assignment-idebug

The solvers library consists of three classes: Matrix, CSRMatrix, and Test. All classes are templated to increase flexibility on usage of different data types. The matrix class objects are just normal matrices, which values are stored in row order fashion. This class also includes various methods which allows user to perform a basic matrix operation like matrix multiplication. CSRMatrix is a class derived from Matrix, objects of this class are Compressed Sparse Row matrices. They differ from normal matrices in regard that they only store non-zero elements and their columns and rows indexes, which reduces memory usage and enhance efficiency of operations. Similarly, to Matrix class, various methods to perform basic operations on CSRMatrices were implemented, such as CSRmatrix multiplication and conversion from CSRMatrix to a normal matrix.

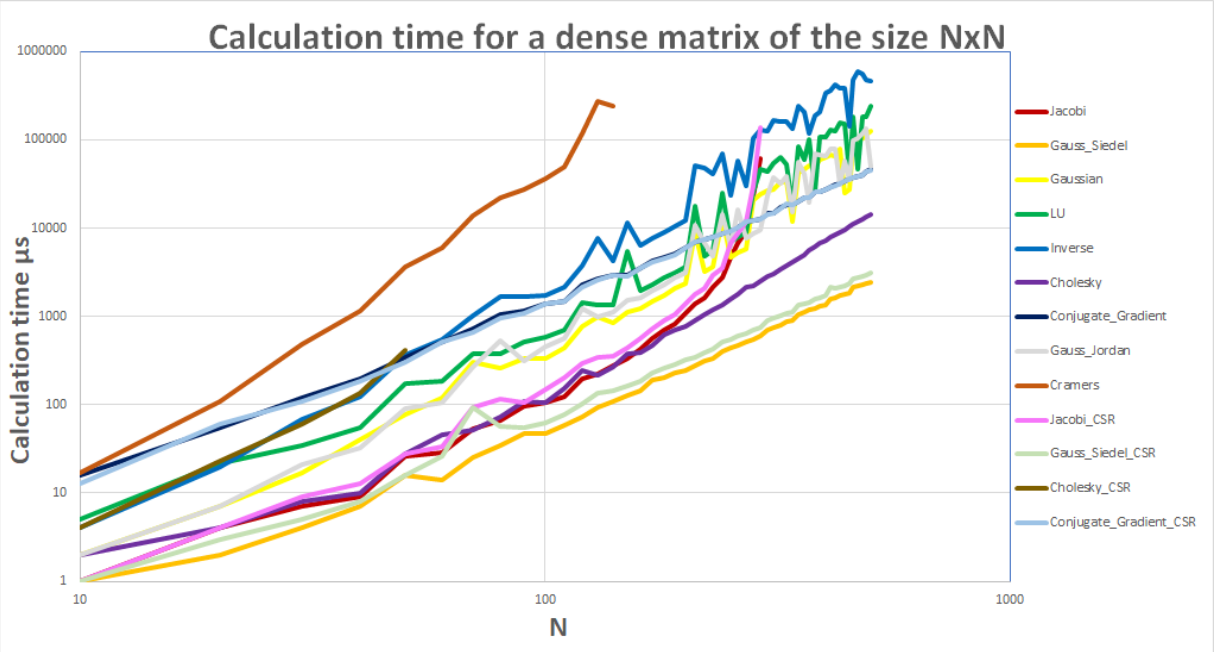
Both classes include a solver method which enable solving linear matrix systems. To solve such a system, user needs to call a solver method on the matrix and as arguments pass the vector b and the vector to which the result will be written. There are 9 dense matrix solvers and 4 sparse matrix solvers implemented. Users decides which one to use by passing a solver name as a third parameter to the solver function. Jacobi, Gauss-Seidel, Conjugate gradient are the three iterative methods we use for solving both dense and sparse matrix. The rest of the methods are direct methods, where Gaussian elimination, LU decomposition, Inverse, Gauss-Jordan elimination and Cramer's Rules are only for solving dense matrix system. Cholesky method is the only direct method we implement for solving both matrix system.

The Test class was used to ensure that all the solvers work properly. It enables the generation of a random matrix of desired size, sparsity, minimal and maximal values on the diagonals and so on. This class also includes various functions which allow user to test if the solver is working correctly, or to estimate how fast and robust it is. The timing tests also allows to write the output to the file.

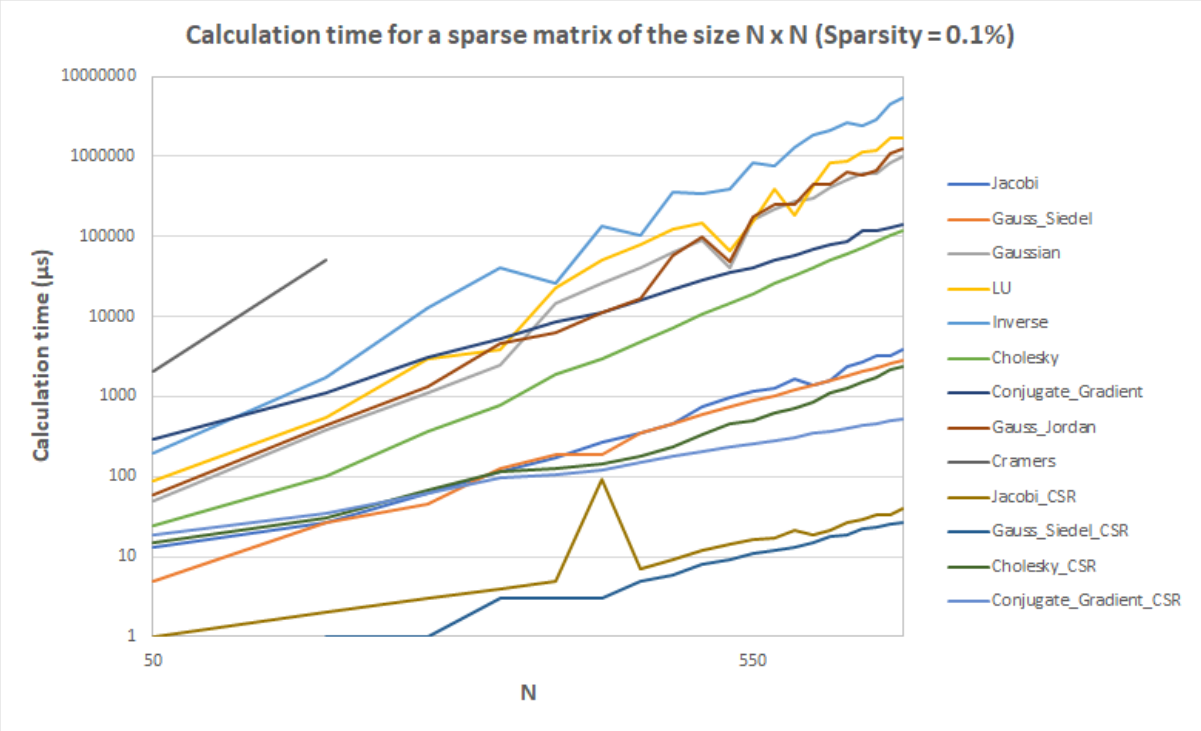
Performance

Timing tests for all the solvers have been performed on the following matrices:

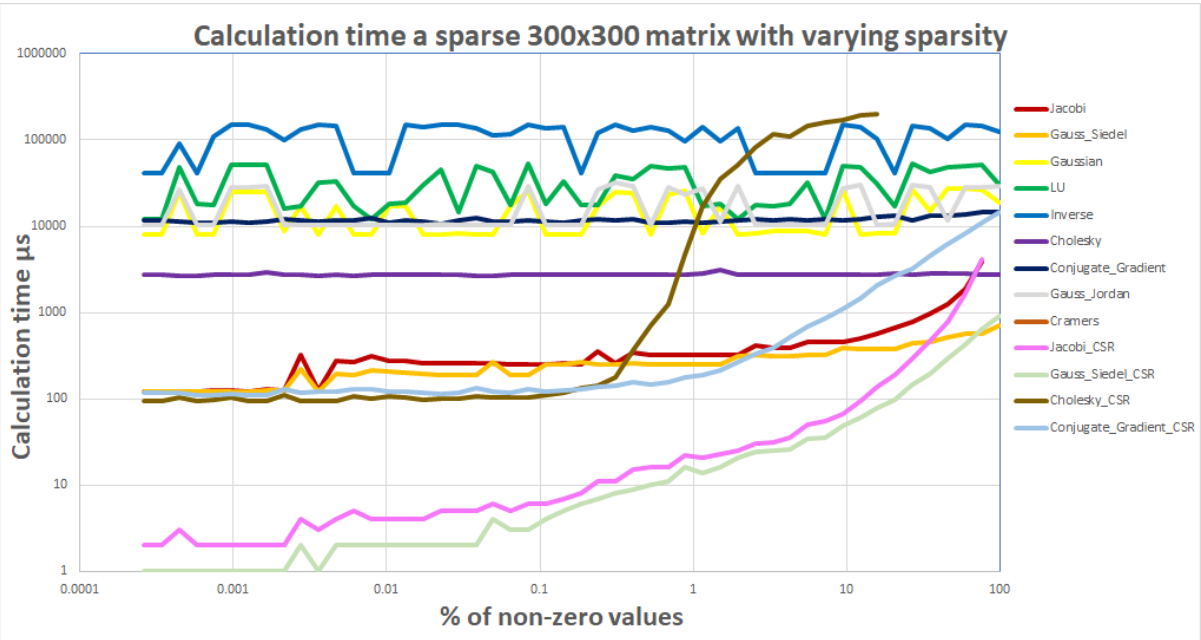
1. Dense matrix with size varied from 10x10 to 500x500. The values on the diagonal were between 100 and 200, all the other values were between 0 and 1.



2. Sparse matrix with size varied from 50x50 to 1000x1000. The matrix sparsity was 0.1%, the values on the diagonal were between 100 and 200,all the other values were between 0 and 1.



3. Matrix of the size 300x300 with sparsity varied from 0.0001% to 100%. The values on the diagonal were between 100 and 200, all the other values were between 0 and 1.



Result Analysis

For both dense and sparse matrices usually the two fastest solvers were the Gauss-Seidel and Jacobi. However, Gauss-Seidel was not only slightly faster but also more robust – unlike the Jacobi, often it was able to solve the problems with matrices which were not diagonally dominant. Most direct methods, like LU decomposition, were usually much slower but more reliable – they were able to solve problems which Jacobi and Gauss-Seidel couldn't. The sparsity of the matrix didn't affect the

calculation time for dense solvers, but it immensely affected the sparse solvers performance. For very sparse matrices (0.1% non-zero values or less), sparse solvers could be even 10^5 times faster than the dense solvers.

Future Improvement

The member functions inside classes would be more concise if the codes that serve the same purpose could be written into a new function. For example, both LU and Gaussian dense matrix solvers look for maximum value in a row multiple times in for loop. Apparently, this could be made to another function called 'getMaxVal()'. Checking if dimension match could also be a separated new function in both Matrix and CSRMatrix classes. In addition, the checkError() in both classes should return the error value instead of telling the user if the error reach tolerance, because some solvers need the error value for iteration purpose.

More effort should have been put on dynamic allocation of objects and containers. For instance, smart pointers could be used when creating a pointer pointing to the matrix /CSR matrix object to avoid possible memory leakage. It would be a good practice to implement smart pointer in member function when we want to return a matrix type.

An additional class called 'Solver' could be implemented. There are 13 member functions for linear solvers, where all of them could be defined in the Solver class instead of defining them in Matrix and CSRMatrix class separately. Once the Solver class is created, it should be derived from Matrix and CSRMatrix class, with a member function named 'Solver' available for calling the type of solver the user wants.

Groupwork Breakdown

Group Name	iDebug		
GitHub link:	https://github.com/acse-2019/acse-5-assignment-idebug		
Date	Jan 31 2020		
Group members	Nurul I. Kamal	Aleksandra Jaroszewska	Ping-Chen Tsai
Dense Matrix Solvers	LU decomposition	Cholesky decomposition	Conjugate gradient
	Inverse method	Gauss-Seidel method	Jacobi method
	Gauss-Jordan elimination		
	Cramer's Rule		
	Gaussian elimination		
Sparse Martrix Solvers		Jacobi method	Conjugate gradient
		Gauss-Seidel method	
		Cholesky method	
Other work		Solvers Accuracy Test	Banded class?
		Solvers Efficiency Test	

Reference Sites

ACSE Lecture 3: Linear Solvers(Numerical Linear Algebra), Prof. Matthew Piggott [IPython Notebooks available at: https://github.com/acse-2019/ACSE-3/tree/master/lectures/L3]

GeeksforGeeks. (2020). *Cholesky Decomposition : Matrix Decomposition - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/cholesky-decomposition-matrix-decomposition/ [Accessed 28 Jan. 2020].