Implementing Linear Solvers using C-plus-plus

Team - Group Chat - Qian Chen[a], Jin Yitong[b], and Wang Jiaruijue[c]

[a]cq419@ic.ac.uk; [b]yj319@ic.ac.uk; [c]jw919@ic.ac.uk

This report is aimed to illustrate the work Team Group Chat did in the ACSE-5 Assignment, explain the concepts and techniques of the project. Six algorithms are implemented to solve the linear system Ax=b, where A is a positive definite matrix, and x and b are both vectors using C-plus-plus. Analysis of accuracy and performance is conducted and a user-friendly interface is provided. For codes and other documents, check via GitHub(https://github.com/acse-2019/acse-5-assignment-group-chat).

Linear Solver | CSR Matrix | C-plus-plus

Our Implementation of class Matrix is derived from the Matrix libraries constructed in class, which provides various necessary operations such as matrix multiplication, inversion and so on. Besides, the class CSRMatrix is well designed to inherit the class Matrix and implemented the compressed sparse matrix concept using a singly linked list.

The class Solver is dependent on class Matrix and CSRMatrix. Every available solver is static and able to solve both the Matrix and CSRMatrix, taking advantage of polymorphism of C-plus-plus. Moreover, for most solvers, we also implemented an optimised CSRMatrix version in order to improve the performance.

The report proceeds as follows. In the next section, the description of solvers we implemented are introduced. In section 2, we present details of how we design, store, operate class Matrix and how we implemented solvers. In section 3, tests and evaluations are performed to check the accuracy and performance. In section 4, 5 and 6, user interface, documentation and how we collaborated will be demonstrated.

1. Linear Solvers

Problem Definition. Given an $n \times n$ positive definite matrix A and an $n \times 1$ vector $\boldsymbol{x}$, the product is a new $n \times 1$ vector which we will call $\boldsymbol{b}$.

$$A\boldsymbol{x} = \boldsymbol{b}$$

$$b_i = \sum_{j=1}^{n} a_{ij} x_j \quad \text{for} \quad i = 1, \ldots, n$$

A. Gaussian Elimination Wikipedia (2019a). If $\det(A) \neq 0$, then we can use Gauss Elimination to solve the linear system.

$$[A \,|\, \boldsymbol{b} \,|\, I] \xrightarrow{Gauss-Elimination} [I \,|\, A^{-1}\boldsymbol{b} \,|\, A^{-1}]$$

$$\boldsymbol{x} = A^{-1}\boldsymbol{b}$$

B. Jacobi Wikipedia (2019b). Starting from a guess at the solution $\boldsymbol{x}^{(0)}$, iterate for $k > 0, \quad i = 1, 2, \ldots, n. \quad a_{ii} \neq 0$

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k-1)} \right)$$

The convergence can only be guaranteed for matrices which are diagonally dominant (for every row, the magnitude of value on the main diagonal is greater than the sum of the magnitudes of all the other entries in that row).

C. Gauss-Seidel Wikipedia (2020b). Starting from a guess at the solution $\boldsymbol{x}^{(0)}$, iterate for $k > 0, \quad i = 1, 2, \ldots, n. \quad a_{ii} \neq 0$

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j<i}}^{n} a_{ij} x_j^{(k)} - \sum_{\substack{j=1 \\ j>i}}^{n} a_{ij} x_j^{(k-1)} \right)$$

Note that as opposed to Jacobi, we can overwrite the entries of $\boldsymbol{x}$ as they are updated. The Gauss-Seidel algorithm should converge faster than Jacobi.

The convergence can only be guaranteed for matrices which are diagonally dominant (for every row, the magnitude of value on the main diagonal is greater than the sum of the magnitudes of all the other entries in that row).

D. LU decomposition Wikipedia (2019c) with and without using partial pivoting Wikipedia (2019d). Given a lower-triangular matrix L and an upper-triangular matrix U such that we can write

$$A = LU$$

In the matrix system we need to solve for $\boldsymbol{x}$ becomes

$$A\boldsymbol{x} = \boldsymbol{b} \iff (LU)\boldsymbol{x} = L(U\boldsymbol{x}) = \boldsymbol{b}$$

We can define $\boldsymbol{c} := U\boldsymbol{x}$ and then we use forward substitution to find $\boldsymbol{c}$ where $L\boldsymbol{c} = \boldsymbol{b}$. Once we have found $\boldsymbol{c}$ and then we use back substitution to find $\boldsymbol{x}$ where $U\boldsymbol{x} = \boldsymbol{c}$.

During the process of forward substitution and back substitution, we might use partial pivoting. In partial pivoting, the algorithm selects the entry with largest absolute value from the column of the matrix that is currently being considered as the pivot element.

All the solvers above are introduced in ACSE-3 Lecture 3.

E. Cholesky decomposition Wikipedia (2020a). Given a symmetric positive-definite matrix A, the Cholesky decomposition of matrix A is a decomposition of the form

$$A = LL^T$$

where L is a lower triangular matrix with real and positive diagonal entries. Similarly, we could use same method to solve for $\boldsymbol{x}$ as LU decomposition.

$$A\boldsymbol{x} = \boldsymbol{b} \iff (LL^T)\boldsymbol{x} = L(L^T\boldsymbol{x}) = \boldsymbol{b}$$

We can define $\boldsymbol{c} := L^T\boldsymbol{x}$ and then we use forward substitution to find $\boldsymbol{c}$ where $L\boldsymbol{c} = \boldsymbol{b}$ Once we have found $\boldsymbol{c}$ and then we use back substitution to find $\boldsymbol{x}$ where $L^T\boldsymbol{x} = \boldsymbol{c}$

2. Implementation

Class Matrix. The purpose of class Matrix is to store a dense matrix in a continuous space. It is derived from the libraries constructed in class without using template because the process of solving the linear systems should run with decimals not integer, so that a constructor of an integer array is available as well.

Besides addition, subtraction and multiplication, row and column operations are also implemented. Elimination and swap between two rows or columns with specific length are used for those decomposition methods. Inversion of the matrix can be calculated through calling the function inverse, which uses Gaussian Elimination Method. All the operations mentioned are able and recommended to be overrode when implementing a sub-class.

Class CSRMatrix. is a sub-class of class Matrix, implementing the concept of compressed sparse row with singly linked list. The traditional CSRMatrix consumes too much on inserting or deleting a single value due to the sequential array. The linked list reduces the time of it in a certain degree but sacrifices the convenience of indexing.

Thus, the operations mentioned above must be overrode to reduce the time of indexing by changing the loop sequence and using temporary variables. Especially for multiplication, substituting the second and third loop results in great performance improvement.

Meanwhile, neither the class Matrix nor CSRMatrix has a public attributes so the way of storage is encapsulated with existing operations and is invisible to the outside. Then the caller can set and get value of a Matrix(including CSRMatrix) object through the same functions.

Class Solver. covers six algorithms dependent on class Matrix and CSRMatrix that can solve the linear systems. All the implemented solvers are designed as static for the class Solver only need to do the calculations not to store any value.

Based on the definition of class Matrix, the solver calls those get and set functions to interact with both Matrix object. However, as poor performance in CSR Matrix indexing, solvers designed for dense matrix consumes too much time on CSRMatrix. Optimisation was done as those CSRMatrix operation, and polymorphism leads to convenience of calling solvers. All the solvers has a boolean return value, indicating whether the input linear system can be solved.

As both Jacobi and Gauss-Seidel are iterative methods, we simplify the converge condition by setting a tolerance instead of eigenvalues. If the residual of x between two iterations is less than the threshold, we treat it as convergence. Also, a maximum number of iterations is set to avoid infinite loops, as the solver might not converge.

3. Accuracy and Performance

Test. For testing our linear solver, we write a test function which can check the consistency between the result from our solvers and that from the numpy.linalg.solve() function. And we also have created three txt files in which all of "A" are positive definite matrices.

test_general.txt includes 18 sets of completely random matrices A, varying from 1x1 to 10x10 in shape and each shape have two examples.

Table 1. Accuracy of solvers for test files

	General	Diagonal	Symmetry
Jacobi	/	1e-8	1e-10
Gauss-Seidel	/	1e-8	1e-11
Gauss Elimination	1e-8	1e-8	1e-16
LU	1e-8	1e-8	1e-16
LU-PP	1e-8	1e-8	1e-16
Cholesky	/	/	1e-16

test_diagonal.txt includes 9 sets of diagonallydominant matrices A, varying from 1x1 to 10x10 in shape.

test_symmetry.txt includes 9 sets of symmetry diagonallydominant matrices A, varying from 1x1 to 10x10 in shape.

Above all, our solvers perform quite well in solving linear systems with different shapes, although some solvers may have limitation for using. The GaussElimination, LU and LU_pp solvers are suitable for any linear system with error less than 1e-8 when A is a positive definite matrix. While Jacobi and Gauss-Seidel solvers can only be used for linear systems with a diagonally-dominant matrix A. For Cholesky solver, owing to low accuracy for diagonally-dominant matrix A (0.1 only), it has strict limitation for application and therefore it can only be used for linear systems with a symmetry diagonally-dominant matrix A (1e-16 in accuracy). In addition, comparing with others forms of A, every solvers can lead a better performance in accuracy when A is a symmetry diagonally-dominant matrix.

Evaluation between Solvers. We conducted a performance test on the solvers for linear systems with different size and sparsity rate. The sparsity rate is defined as the proportion of zero value in the matrix. Each solver runs 14x7x10 times on 14 sizes, 7 sparsity rates , 10 times each and 2 modes, dense and sparse.

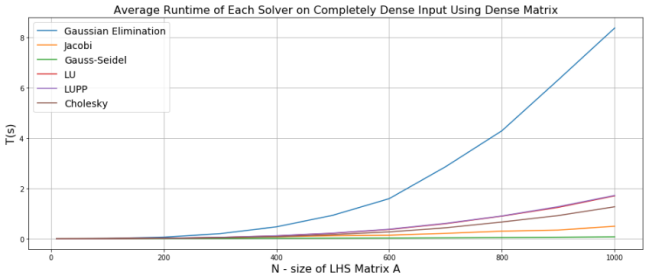

Fig. 1. Average Runtime of Solvers on Completely Dense Input using Dense Matrix

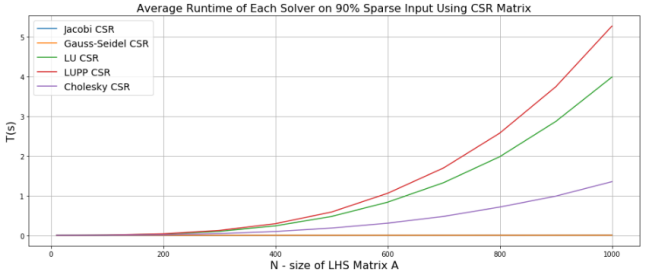

Fig. 2. Average Runtime of Solvers on 90% Sparse Input using Sparse Matrix

As Fig. 1 and 2 shown, no matter which size and sparsity of the matrix is, iterative methods, Jacobi and Gauss-Seidel, are the fastest when the linear is system is diagonally dominant, and Gauss-Seidel always converges faster than Jacobi. Cholesky Decomposition is slower but faster than LU and LU with partial pivoting, while Gaussian Elimination performs worst.

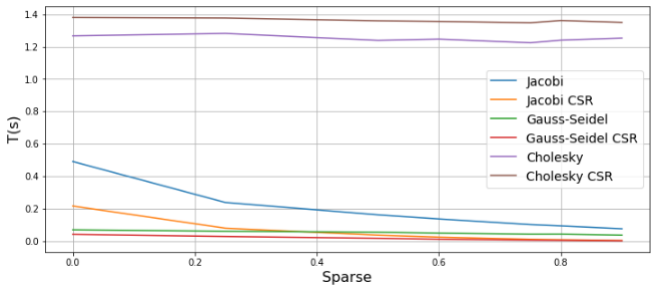

Fig. 3. Average Runtime of Solvers on 1000x1000 90% Sparse Matrix

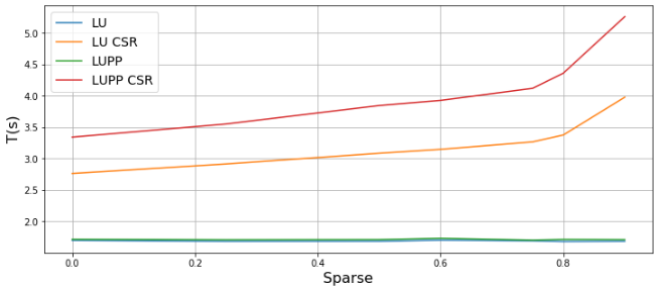

Fig. 4. Average Runtime of Solvers on 1000x1000 90% Sparse Matrix

Evaluation over CSRMatrix. To compare the performance of CSRMatrix, Fig. 3 illustrates the runtime change when changing the sparsity. CSRMatrix performs best on iterative methods as the reduction in time are most obvious, and the reduction on Cholesky are around zero. For these three solvers, we could say they conquer the shortcoming of indexing, thus saving much more space than the dense one.

However, as Fig. 4 shown, the runtime increases almost 4-6 times on LU and LU with partial pivoting, because LU decomposition involves too many row operations, which leads to too many linked list iteration(each slightly slower than integer addition due to the check of null pointer).

4. User Interface

We implemented a terminal application to run the linear solvers. Users can run pre-stored test files to test whether the solvers are running normally. Besides, users can choose different solvers to solve the linear systems in either dense or sparse mode, input through three ways, random generation and standard input. The result will be directly printed to standard output and the application does not support any file stream output for safety concerns.

5. Documentation

As a C-plus-plus program, we found a common documentation tool, doxygen, but it looks ancient and depends on specified environment. Therefore, we use Sphinx to convert a doxygen output to a html-based documents. Download this file and run the index.html on a computer with any browser installed, the documents will be tidy and easy to use.

The documents only include core files such as *Matrix.h*, *CSRMatrix.h* and *Solver.h*.

6. Collaboration

We use GitHub as our development platform, https://github.com/acse-2019/acse-5-assignment-group-chat. There is no direct commits allowed to master branch and a pull request must be created for any change to the master. So we cannot assure every branch is updated with master since they might give wrong outputs.

At the beginning of the project, we designed the all the classes and functions of Matrix and Solver, and we set up rules of how the solvers interact with matrices. Qian and Jin started implementing the solvers and Wang tried to implement the class Matrix.

Accuracy test and oerformance evaluation were conducted in the last week. Jin and Qian were responsible for the accuracy test and fixed bugs occurred while testing. Wang was in charge of optimising the performance of CSRMatrix and solvers in sparse mode.

After test and evaluation, we finished the documentation, interface and this report together.

References

Wikipedia (2019a). Gaussian elimination — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Gaussian_elimination&oldid=932248528. [Online; accessed 1-February-2020].

Wikipedia (2019b). Jacobi method — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Jacobi_method&oldid=931403238. [Online; accessed 1-February-2020].

Wikipedia (2019c). Lu decomposition — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=LU_decomposition&oldid=923660898. [Online; accessed 1-February-2020].

Wikipedia (2019d). Pivot element — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Pivot_element&oldid=902128553. [Online; accessed 1-February-2020].

Wikipedia (2020a). Cholesky decomposition — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Cholesky_decomposition&oldid=937630650. [Online; accessed 1-February-2020].

Wikipedia (2020b). Gauss–seidel method — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Gauss%E2%80%93Seidel_method&oldid=937154112. [Online; accessed 1-February-2020].