Linear Solver Report

Describing the linear solver built for Matrix- Vector calculation

Developed by: Tianchen Lu, Yujie Ma, Tianhsun Yao

GitHub link: https://github.com/acse-2020/group-assignment-lovecpp.git

Date: 28/Jan/2022

A. Introduction

The aim of this program is to solve the linear system Ax = b, which is usually the case in the real world. The program is built upon the implement of different algorithms dealing with both dense matrix and sparse matrix.

A linear solver is built to solve the linear system A x=b. And a variety of algorithms are implemented for solving linear system involving dense and sparse matrix. As to dense matrix, we implemented 4 algorithms which are Gauss-Seidel, Jacobi, LU factorization and Gaussian elimination, for sparse matrix, we implemented 3 algorithms which are Jacobi iterative method, LU factorization and Conjugate Gradient.

There are test_main.cpp built for user’ s interface and testing the methods. Totally 7 algorithms are built, different sizes of matrices are tested to evaluate the performance.

B. Program Structure

The execute the project, please refer to the detailed guide in ‘ReadMe.md’ .

There are three main parts in our program: LinearSolver.cpp, Matrix.cpp & CSRMatrix.cpp, test_main.cpp.

All the methods are implemented in our Class named LinearSolver. The attributes of the class are shown as below.

```cpp
/*
build our linearsolver class for the problem
*/
#pragma once
#include ...
using namespace std;
template <class T>
class LinearSolver
{
public:
    //linear solver variable
    Matrix <T> *A;
    CSRMatrix <T> *A_CSR;
    vector <double> x;
    vector <T> b;
    int rows;
    int cols;
```

And the member functions including (as shown below):

```cpp
    //constructor
    LinearSolver(Matrix<T>& A, vector<T>& b);

    //randomly construct the Matrix A and vector b
    LinearSolver(int Arow, int Acol);

    //constructor load Matrix A and vector b
    LinearSolver(string MatrixAFile, string VectorBfile);

    //destructor
    ~LinearSolver();

    //Print Linear System
    void PrintLinearSystem();

    //write solver answer, x
    void WriteSolution(string outfilename);

    // LU factorisation
    int LuFactorisation();

    // Gauss Elimination
    int GaussEli();
```

```cpp
    //Jacobi
    int Jacobi(int iterationNum);

    //Gauss-Seidel
    int GaussSeidel(int iterationNum);

    //Jacobi sparse
    int JacobiCSR(int iterationNum);

    //LU sparse
    int LuFactorisationCSR();

    // Conjugate Gradient
    int ConjugateGradient(long double tolerance);

};
```

Main function includes the user interface and error handler. For each method, return 1 if solver works successfully, return error code if corresponding error occurs (for details, please refer to Description of files).

C. Description of files

LinearSolver.h & LinearSolver.cpp	Self-developed, including a class named LinearSolver, including the algorithms (Please check Sec.B for details).
test_main.cpp	Interface, the main function provides a user interaction way to test our Solver it provides two methods of testing which are create Matrix randomly and load Matrix from files. For error_handler, when error occurs in the solver, it returns corresponding error message (integer). Please check the error information shown as appendix. A.
Matrix.h & Maritx.cpp	Referred from lectures, we add the constructor with input Matrix
CSRMatrix.h & CSRMatrix.cpp	Referred from lectures, we add the constructor with input Matrix which will convert a dense matrix to a sparse matrix.

D. Algorithm ideas

Gaussian elimination method	Gaussian elimination method characterizes by step-by-step elimination of the variables. It firstly converts the matrix into upper-triangle format, then back substitute them to get the solutions.
LU factorization method	When doing this algorithm, firstly new two n*n array to store upper triangle matrix, and lower triangle matrix.
Jacobi iterative method	When implementing Jacobi iterative method, we found that every approximation in a particular iteration is based on the approximated values in the previous step but the first iteration itself was based on an initial guess.
Gauss-Seidel method	When implementing Gauss-Seidel method, it' s a modified Jacobi that enables a better speed of convergence. The latest values of the unknown are used at every stage of iteration, thus, it converges faster.
Jacobi iterative & LU factorization method (for sparse matrix)	Implement Jacobi iterative & LU factorization method using sparse matrix, get the value at a certain location using values, row_position, and col_index. It will add more loops in the function, which will cost more time, but the sparse method will save a lot of memory. All the variables newed in the algorithm all apply to the CSR format, which will cost less memory.
Conjugate Gradient	This method is an iterative method suitable for solving sparse systems of linear equations. It is a typical conjugate

	direction method in that each of its search directions are conjugate to each other, and these search directions are simply a combination of the negative gradient directions and the search directions from the previous iteration.

E. Program package

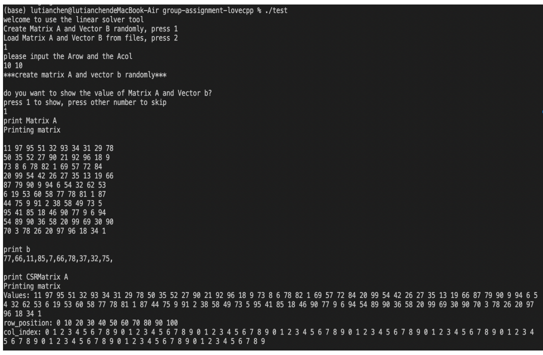
The program has been packaged. With the interface, the data can be input, and the output can be attained.

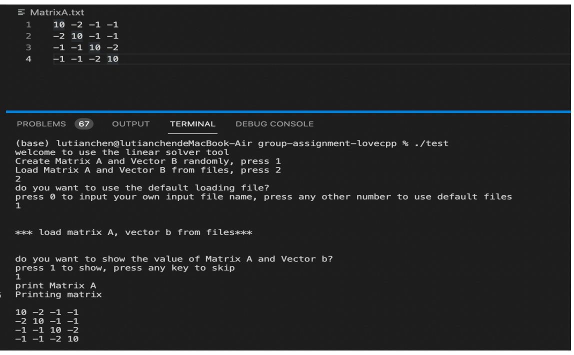
1. <u>Input</u>

The input matrix A and b can be randomly generated or attained from pre-created .txt file.

User can define the dimensions of the matrices themselves, and the randomly created matrices can be printed out to check (shown in below).

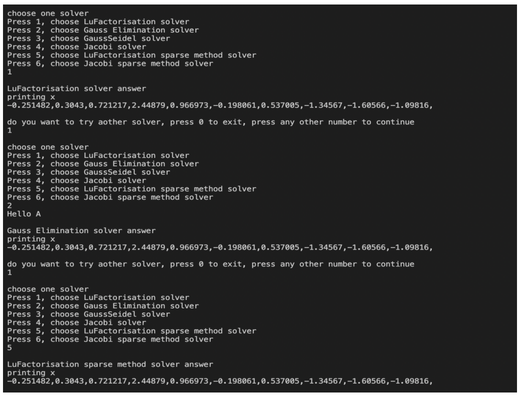
Also, users can create .txt files storing matrix A and b, and the matrices can be passed as an input (as shown below).



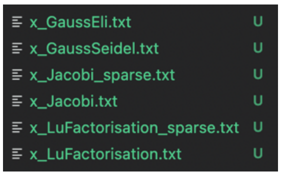


2. <u>Interface</u>

An interface method is built for user’ s convenience. After attaining the input matrices, users can decide which method to apply (as shown below). Besides, users can choose another method to apply after the previous method completed. In this way, they can compare the results.



3. <u>Output</u>



The solution x will initially be printed out in the interface, then it will be stored in the .txt files named by corresponding method name. With that, users can clearly check the results and decide what to do with these solutions next.

F. Results

Due to page limitation, only three methods are evaluated here, involving three different sizes of matrix.

Solutions attained are to be checked correct or not (T for true, F for false). Time.h library is used here.

Method	Size	Solution attained (T/F)	Time consuming
Gauss Elimination	10 * 10	T	0.0068s
	1000 * 1000	T	0.8627s
	100000 * 100000	N/A	N/A
LuFactorisation	10 * 10	T	0.00176s
	1000 * 1000	T	1.19s
	100000 * 100000	N/A	N/A
LuFactorisation_sparse	10 * 10	T	0.0091s
	1000 * 1000	T	3.37s
	100000 * 100000	N/A	N/A

G. Superiotriy

Implement LinearSolver class, well-structured, easy to call the algorithms (implemented as class functions), easy to read and write data, print information.LinearSolver class has three constructors, which can be either constructed by creating Matrix randomly or loading Matrix from files.

Implement interactive user interface, you can test the tool following the instructions (choose to create Matrix randomly or to load Matrix from files), you can continuously test each algorithm and exit at any time.

Implement error handler to make our tool robust. Error message will be shown when corresponding error code returns.

H. Limitations

From the results section, it is obvious that our solver is limited when dealing with the huge matrix size (1e10). This is because that our random creation of matrix takes such a long time, where should be improved in the future work.

Reference

[1]. Wikpedia (2022) Conjugate Gradient. Available at: https://en.wikipedia.org/wiki/ Conjugate Gradient (Accessed: 29 Jan 2022).

[2]. Wikpedia (2022) Gauss-Seidel. Available at: https://en.wikipedia.org/wiki/ Gauss-Seidel (Accessed: 29 Jan 2022).

[3]. Wikpedia (2022) Gaussian elimination. Available at: https://en.wikipedia.org/wiki/ Gaussian elimination (Accessed: 29 Jan 2022).

[4]. Lecture notes

Appendix. A: Error handler

error message number	Error type
-1	Matrix A is not a square, rows not equal cols
-2	Matrix A row number not equal size of the vector b
-3	Jacobi cannot converge
-4	LU decomposition first element is zero
-5	Jacobi sparse method cannot converge

Appendix. B: Task breakdown:

To achieve an excellent group collaboration, the whole task was roughly divided into two main parts: report writing, code implement. The code implement is distributed by the algorithms. And we have discussed about what each member are good at and decided which part should each member take. So that the subtasks are ensured to be fair enough. The task distribution and contribution of each team member can be seen in table below:

Name	Contribution
Tianchen Lu	Design the structure. Implement of main function interface,error handler, LinearSolver class constructor, loading and creating Matrix A, writing answer to the file. Implement algorithm of Jacobi, LU decomposition, Jacobi for sparse matrix.
Yujie Ma	Implement algorithm of Gauss-Seidel, LU decomposition(for sparse matrix). Writing README.md documentation
Tianshun Yao	Implement algorithm of Gauss Elimination, Conjugate Gradient. Report writing. Documentation writing README.md