Implementation of Linear Solvers for Linear Algebra system using C++

Qiuchen Qian Yusen Zhou He Zhu

Feb, 02, 2020

1. Introduction

This report will introduce the linear solver system. The class diagram can be seen below. There are 5 class in total, 4 template class and 1 normal class. CSR Matrix is derived from Matrix, this two both have attribute and method while other Class mainly encapsulated the function method.

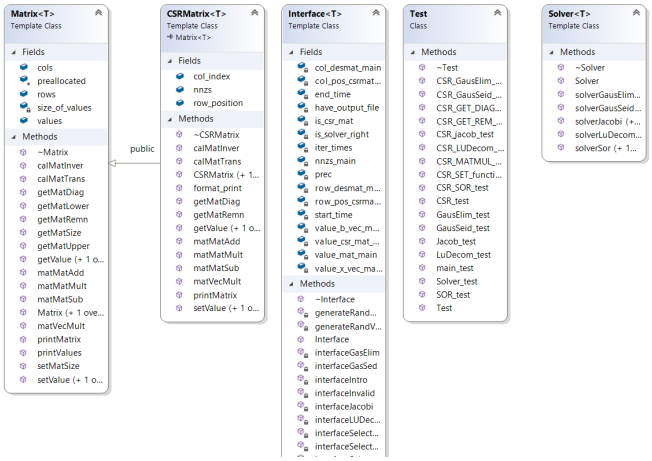

Figura 1: Class diagram

The remainder of this report is organized as follows. In Section 2, the five linear solvers will be explained. In Section 3, the user interface and the corresponding logic flow chart are introduced. In section 4, we illustrate the implementation of dense and CSR matrix. In Section 5, the advantage and weaknesses of the project will be further analyzed from robustness, memory management and algorithm optimization point of view.

2. Linear Solvers

The program proposes five algorithms to solve the linear system $\boldsymbol{Ax} = \boldsymbol{b}$, where the $\boldsymbol{A}$ is a positive definite matrix. Each of these algorithms takes matrix $\boldsymbol{A}$ and RHS vector $\boldsymbol{b}$ as inputs and returns the corresponding vector $\boldsymbol{x}$ as an output.

(a) Gaussian elimination. To perform this method for linear systems when $det(A) \neq 0$, an augmented matrix needs to be formed by adding the RHS vector as an additional final column. After the elimination stage, the square matrix part of the updated augmented system is now an upper-triangular matrix. Then the back-substitution method is performed to obtain the exact solution upper-triangular form of the augmented system.

(b) LU decomposition. This decomposition involves a lower- L and an upper- U triangular matrix such that A can be transformed as:

$$A = LU.$$

In this case the matrix system required to solve for $\mathbf{x}$ becomes

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

The above system then reads

$$L\boldsymbol{y} = \boldsymbol{b},$$

where L is a matrix and $\boldsymbol{y} = U\boldsymbol{x}$ is an unknown. As L is in lower-triangular form, the $\boldsymbol{y}$ can be easily found by using forward substitution. Once identified, the second linear system can be solved

$$U\boldsymbol{x} = \boldsymbol{y},$$

where now the U is upper-triangular. Therefore, the back substitution is used to give the solution $\boldsymbol{x}$.

(c) Jacobi. This is an iterative solver which takes an initial guess. The next state solutions can be calculated using:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{n} a_{ij} x_j^k \right), \quad i = 1, 2, \ldots, n.$$

Note that following GS and SOR have same range of i. Jacobi method separates A with diagonal and remainder matrix, which will reduce the difficulty of

calculating the inverse matrix. Convergence is considered as well, Jacobi and following iterative solvers can converge for strictly diagonally dominant and most symmetric positive definite matrices [1].

(c) Gauss–Seidel. *GS* is similar as *Jacobi* with a small improvement (using updated components of the solution vector once they are accessible). The equation shows as below:

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

(e) Successive over-relaxation. *SOR* can further improve the convergence of the *GS* method.The main innovation point is considering the new x_i as a weighted average between its previous value and the value predicted by conventional *GS* iteration. Therefore. the corresponding formula for the k^{th} iteration of the method and the i^{th} row is:

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

3. User Interface

The general user manual can be found in github (README.md). The logical flow chart is:

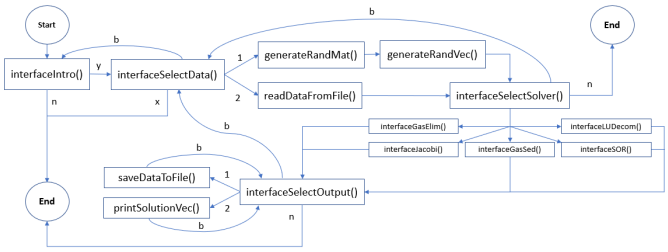

Figura 2: Logic flow chat

As can be seen from the figure, in each selection function, users can choose 'b' or 'x', where 'b' means 'back', 'x' means 'exit'. Other options like '1' or '2' correspond to specified action. In all, users can input data source through a built-in random matrix generator or '.txt' data file. Then users can select one of five solvers to calculate the solution vector. The output can be either printed if there exist solutions or saved to a '.txt' file without considering having right solutions or not. Users can continue select data source if they want.

4. Matrix

Original and Compressed Sparse Row format stored matrix will be adopted according to different scenarios. CSR matrix class derived from Matrix class and rewrite functions like multiplication, value assignment, etc. In the program, the matrix type will be distinguished by the number of non-zero values. According to different types of the matrix, printing method and algorithms will be differ. Especially for a large matrix (like 1000 x 1000), CSR matrix shows better storage and reading performance. The comparison test results can be checked in the interface program.

5. Performance

	Jacob	GS	GE	LU	SOR	Jacob(CSR)	GS(CSR)	GE(CSR)	LU(CSR)	SOR(CSR)
10	0.001	0.001	0	0	0	0.012	0.001	0	0	0
50	0.138	0.011	0.002	0	0.01	2.23	0.035	0.013	0.001	0.035
80	0.501	0.025	0.009	0	0.025	11.88	0.126	0.075	0.002	0.127
100	0.952	0.037	0.017	0.001	0.038	26.812	0.233	0.178	0.002	0.24

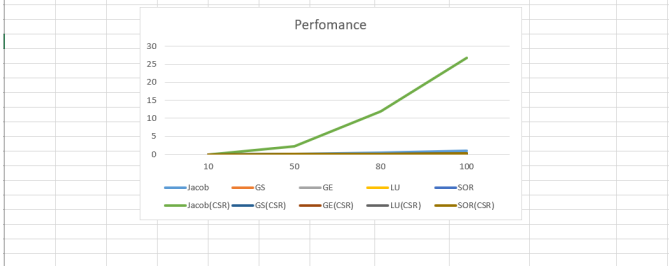

Figura 3: run time/Size

	Jacob	GS	GE	LU	SOR	Jacob(CSR)	GS(CSR)	GE(CSR)	LU(CSR)	SOR(CSR)
5	3.019	0.818	0.018	0.016	0.4	9.32	0.98	0.025	0.019	0.046
7	3.018	0.808	0.017	0.015	0.392	9.77	0.976	0.024	0.017	0.449
9	2.904	0.788	0.0015	0.013	0.341	9.696	0.933	0.023	0.015	0.41

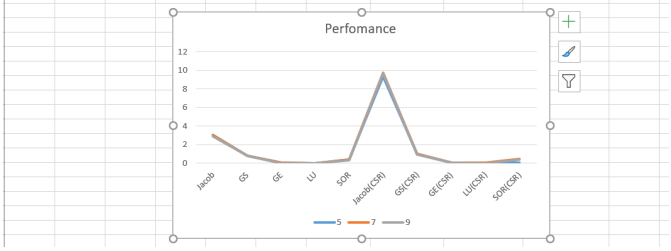

Figura 4: run time / sparsity

In performance test 1, full randomly generated Matrix is uesed. As can be seen from the figure, CSR generally slower than normal matrix, especially for Jacobi method, which has a significant increment. Because full randomly generated Matrix is used and CSR matrix has no advantage on this. Another reason is that a insert value operation will be called in CSR matrix if the value in original is 0. It normally has to copy the full array with size n to a bigger array with size n + 1, this will cost a lot of

time. In performance test2, a 4x4 maitrix with different sparsity were tested. It can be seen from the figure and table that the result of three solutions is very closed, Although CSR matrix is still slower than the normal matrix, the gap become smaller. This is in line with expectations that using sparse matrix can improve performance when the matrix have high sparsity. Finally, the speed of five solvers follow the order LU > GE > SOR > GS > Jacobi in this project. This is due to SOR and GS are improved version of Jacobi and direct method often takes less time (with less precision sometimes).

6. Sustainability

In practical applications, the CSR format is used to speed up calculation. However, in the test stage, it was found that the CSR format does not show an significant performance improvement, comparing to densely stored matrix. The reason for this is when calling setting value function in ÇSRMatrixçlass, if the original value in the position is 0, it is necessary to copy the whole array with size n to a new array with size n + 1. This can be improved by storing the CSR matrix in linked list format. Moreover, algorithm optimization was not considered much. For example, it is difficult to optimize solvers through CSR matrix. But the team considered much about memory management and unnecessary calculation avoidance. For example, the solver will break looping once required convergence meets; Interface member functions will allocate temporary heap memory mostly instead of using stack memory; etc. Besides, building correct architecture in early stage can be extremely important. During developing period, team members often found new requirements and this may lead to extensive modification of the original code. For further projects, this kind of low-efficient developing mode can be avoided through making a clear and normative structure.

7. Reference

[1] J. Kiusalaas. (2014, Jun). *Numerical Methods in Engineering with Python 3* [Online]. Available: $https://www.cambridge.org/core/services/aop-cambridge-core/content/view/7E51D8F9311854CCE5017968C6655457/9781139523899c2p31-103CBO.pdf/systems_of_linear_algebraic_equations.pdf$