Matrix Linear Solver Report

Submitted short report for Assessment of the ACSE-5 module.
Developed by Chao-Lun Liu, JiaBo Wang and Xun Xie.

Introduction

The aim of this assignment was to implement an algorithm to solve the linear system **Ax=b**, where **A** is a positive definite matrix, and **x** and **b**. We made this program that implements this build upon the Matrix.cpp and Matrix.h libraries that we have constructed in class. We construct a class named Solver which is derived from Matrix and used to store the linear solvers.

The main.cpp is used to load an example **A, x** and **b** and we have already tested any size greater than **10X10**. The slover.cpp is used to write the methods which can take **A** and **b** as inputs and returns **x** as an output.

We chose eight different types of the linear solver. The first one is **the Gaussian elimination method** which is usually understood as a sequence of operations performed on the corresponding matrix of coefficients. The second method is **LU decomposition method** which factors a matrix as the product of a lower triangular matrix and an upper triangular matrix. The third one is **Gauss-Jordan elimination method** which is done by transforming the system's augmented matrix into reduced row-echelon form by means of row operations. The fourth one is **Jacobi method** which is an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations. Each diagonal element is solved, and an approximate value is plugged in. The fifth one is **Gauss-Seidel(GS) algorithm** which should converge faster than Jacobi. However, GS's convergence is decided by our value of tolerance, resulting not faster than Jacobi in every type of matrix. The sixth one Is **GMRES** which is based on Gaussian Elimination (even without partial pivoting, and Krylov subspace method which seeks to minimize the residual (**r = b − AX**), and the Arnoldi iteration is used (for MINRES, Lanczos). For large systems, we may look to an iterative solver as an alternative as for certain classes of the matrix (e.g. sparse), as they can often converge to an acceptable level of precision with far less work.)The seventh one is **Cholesky Decomposition** The idea is the same as LU decomposition, i.e. use the triangular for of the matrix LL. The matrix L and LT are triangular so you solve it directly with forward and backward substitution, (**Ab = x => L LT x = b**) first $Ly=b$, Ly=b and after LT$x=y$, LTx=y. The final one is **Conjugate gradient method** which is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods.

Meanwhile, we have realized multiple **b** (namely Matrix form) for Gauss Elimination, LU decomposition and Gauss Jordan method, which can solve different **b** simultaneously. These new functions are realized by overloading, so the user can call the same name function by their input parameter types.

Software Structure

The solution is a C++ executable, organized into functions held in Matrix.cpp and using three classes: Matrix solver and Interface. Matrix provides the initial matrix (rows, columns, whether there is pre-allocation), and provides different operators such as addition, subtraction, multiplication, equal, transpose between Matrix and other operations between Matrix and vector (in the standard library), etc. Solver provides five different types of linear solver: (1) Gaussian elimination method; (2) LU decomposition method; (3) Gauss-Jordan elimination method; (4) Jacobi method; (5) Gauss-Seidel; (GS) algorithm; (6) GMRES; (7) Conjugate gradient method; The function main() initializes the Interface class and from there the other classes and their respective methods are called in order to produce the final model. It is used to load an example **A, x** and **b** and we have already tested any size greater than **10X10**.

Meanwhile, our class is constructed applying a template, so the user can define their Matrix as float type such as float, double and long double. But users cannot make Matrix-type as an integer.

As for initialization to Matrix, users can store a preallocated array or vector in integer or float type, and users must follow the fixed instructions to initialize the Matrix. In this way, it can ensure the C++ to be safe conversion.

Using the Software (Input / Output)

1. Input:

Test data:

(1) `int test_data_1[100]; OR double test_data_1[100];`

(2) `Matrix<double> A_1(10,10) ;`

```
for (int j = 0; j != A.rows * A.cols; j++)
    A.values[j] = test_data_1[j];
```

(3) `vector<double> b1 = produce_b(A_1, min, max);`

```
PS: min and max represents the range of random values produced for b
OR you can construct a Matrix<double> b to store multiple b to solve different b simultaneously.
Matrix<double> b2(A.rows, size);
// then initialse your b2 by (1) and (2) step;
```

(3) `vector<double> x0_1 = LU_solver(A_1, -100, 100);`

```
OR
Matrix<double> x0_1(A.cols, b.cols);
X0_1 = LU_solver(A_1, b2)
```

Test solver:

`(1)test_GE;(2) test_LU;(3) test_Jacobi;(4) test_gauss_seidel;(5) test_CG;`

2. Output: one dimension vector x => size 10

Performance：

For size 10, 20, 40, 80, 160, 320, 640 Matrix.

These are time consuming for different algorithm (ms unit) and the average increase speed for them.

```
GE time:
62  35  228 1642    12605   97594   790635
Averge increase speed is: 9.3091
LU time:
18  44  254 1780    13164   100408  809228
Averge increase speed is: 8.96574
LU_pp time:
37  107 515 2755    16216   111222  848273
Averge increase speed is: 7.63356
Gauss Jordan time:
13  40  242 1682    12976   99929   742503
Averge increase speed is: 8.9616
Program ended with exit code: 0
```

Algorithm Implementation (Function)

1. Gaussian elimination method which is usually understood as a sequence of operations performed on the corresponding matrix of coefficients.

2. LU decomposition method which factors a matrix as the product of a lower triangular matrix and an upper triangular matrix.

3. Gauss-Jordan elimination method which is done by transforming the system's augmented matrix into reduced row-echelon form by means of row operations.

4. Jacobi method which is an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations. Each diagonal element is solved for, and an approximate value is plugged in.

5. Gauss-Seidel(GS) algorithm which should converge faster than Jacobi. However, GS's convergence is decided by our value of tolerance, resulting not faster than Jacobi in every type of matrix.

6. GMRES which is based on Gaussian Elimination (even without partial pivoting, and Krylov subspace method which seeks to minimize the residual (r = b − AX), and the Arnoldi iteration is used (for MINRES, Lanczos)

7. Conjugate gradient method which is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods.

Model Analysis (Advantage / Weak point)

1. Adv: show the accurate the exact answer
2. Adv: The model can be performed over any field, not just the real numbers. The choice of ordering on the variables is already implicit in the model, manifesting as the choice to work from left to right when selecting pivot positions.
3. Adv: Sometimes even certain conditions are not met, the Jacobian method can still converge.
4. Adv: The model can also be used to solve unconstrained optimization problems such as energy minimization
5. Disadv: using too much time to get convergence
6. Disadv: Optimisation is bad so that theoretically advanced algorithms cannot be faster than Gauss Elimination in computer implementation.
7. Jacobi method has its limitations. The standard convergence condition (for any iterative method) is when the spectral radius of the iteration matrix is less than 1.
8. Our GMRES only can get an exact answer when matrix size is smaller than 5x5
9. Cholesky only can get an exact answer when we input our design matrix A and b.

Limitations and Further Work

1. **Limits:** the xcode too garbage, resulting us that can not debug easily;
2. Limited special matrices used;
3. **Future work:** Implement more difficult algorithm to get better performance
4. Use more special matrices to implement different algorithms
5. Each of these solvers often has a particular niche where they can perform better than the others, so combining all these methods into one bundle might be a reasonable challenge for a C++ programmer. Thinking even further ahead, it is certainly not impossible to conceive of designing a new GUI(combine QT software platform) which would run these codes in a more user-friendly way.

Reference

1. Solvers: (1) https://en.wikipedia.org/wiki/LU_decomposition
   (2) https://en.wikipedia.org/wiki/Jacobi_method
   (3) https://www.mathwords.com/g/gauss-jordan_elimination.htm
   (4) https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method
   (5) https://en.wikipedia.org/wiki/Generalized_minimal_residual_method
   (6) https://en.wikipedia.org/wiki/Conjugate_gradient_method
2. ACSE3 jupyter notebook
3. CSDN

Task Allocation

CHAO-LUN LIU：Design and implementation classes with XUN XIE. Implementing Gauss-Seidel, GMRES, Conjugate gradient method. Writing interface code.

JIABO WANG：Writing report.

XUN XIE：Design and implementation classes with ZHAOLUN LIU. Implementing Gauss Elimination, Jacobi method, Gauss Jordan method, LU decomposition and LU decomposition with partial pivot for single b. Implementing Gauss Elimination, Gauss Jordan method, LU decomposition for multiple b. Writing test code.

Github Liner: https://github.com/acse-2019/acse-5-assignment-c-punisher