

# **Advanced Programming:**

## **Assessment II Report**

Nourhan Berjawi

- I. Mathematical background and acknowledgments
- II. Library code breakdown
- III. Limitations and future improvements
- IV. `main.cpp`
- V. Final notes

## I. Mathematical Background and Acknowledgements:

The assessment requires us to solve a system  $Ax = b$ .

The iterative methods I have chosen are the **Jacobi Method** and the **Gauss-Seidel Method**. For both iterative methods to converge, the spectral radius of the matrix must be less than 1. But obtaining the spectral radius can be as time consuming as solving the system with a direct method (such as Gaussian elimination.) To avoid having to do that calculation, and for the solution to converge, it is sufficient that  $A$  be strictly diagonally dominant. For the Gauss-Seidel method, it is sufficient for  $A$  to be symmetric and positive definite. We know that if a symmetric matrix is strictly row diagonally dominant and has strictly positive diagonal entries, then it is positive definite. Thus, generating a symmetric matrix and ensuring the diagonal entries are strictly dominant is sufficient for both methods to converge.

To achieve this goal, I created a function in main.cpp that, given a size, generates a random sparse symmetric positive matrix, that is also diagonally dominant.

## II. Library code breakdown:

The library I have created specializes in solving linear systems involving a sparse matrix, using Compressed Row Storage format. A `sparseMatrix<T>` object is a templated class, that inherits from `Matrix<T>`, and that stores a matrix as 3 vectors:

- `vector<T> sparse_vals`: stores the non-zero matrix vals
- `vector<int> cols_ind`: stores the corresponding column for every non-zero
- `vector<int> rows_ptr`: stores the index of where each row begins in `sparse_vals` and `cols_ind`

In order to construct a `sparseMatrix` object, the user must provide the sizes of the matrix as well as a pointer to the array of values. Then, the constructor of `sparseMatrix` calls the constructor of `Matrix`, and gets `sparse_vals`, `cols_ind`, and `rows_ptr` from the array of values of the matrix.

Alternatively, the user can initialize an empty `sparseMatrix` with a specific number of rows and columns, then proceed to fill it later using the `set_matrix_vals` member function.

The types acceptable to replace `T` in `sparseMatrix<T>` are `double` and `int`. In other words, the `sparseMatrix` can be a matrix of doubles or integers. However, the solution returned from solvers will always be a `vector<double>`.

To solve a system, the user can call the public member functions `solve_jacobi` and `solve_gauss`, and given them a vector of integers or doubles.

The Jacobi and Gauss solvers are optimized to only loop over non-zero values, which are stored in the member `sparse_vals`. They also terminate when the error (sum of absolute differences between  $A\bar{x}$  and  $b$ ) becomes less than 0.5 or when the number of iterations exceeds 5000.

The class `sparseMatrix` has additional member functions that perform sparse matrix-vector multiplication and getter functions for the members as well as the size. The class has no destructor because vectors were used ensuring garbage collection would be automatically done.

### III. Limitations and future improvements:

As mentioned before, the solvers I have implemented converge only for a well-conditioned matrix. They do not manipulate the matrix in any way to overcome the condition problem. In the future, I would like to implement additional solvers that can converge for a larger array of matrices.

### IV. `main.cpp`:

For each of the sizes 10, 50, and 100:

- the `main.cpp` program generates a random symmetric tridiagonal strictly diagonally dominant matrix  $A$  and a random vector of doubles  $b$
- solves the system using the Jacobi method and displays the solution
- solves the system using the Gauss-Seidel method and displays the solution

To execute `main.cpp`, download the repository, and in its directory run:

```
g++ Matrix.cpp sparseMatrix.cpp main.cpp -o main  
./main
```

### V. Final notes:

Building my own library that solves linear systems using `c++` was a challenging but mostly fulfilling experience. I was able to refresh my memory about some topics in linear algebra, as well as better familiarize myself with the working of `c++`. The only thing that I would have additionally preferred was having fellow classmates to collaborate with, so we could share knowledge and produce a larger, more robust library.