Joyce Huang

Sh2393

ECE 4960

Programming Assignment 2 Report

**Design**

Using the modular design, I included several layers in my implementation. On the base level, I created a class of row compressed matrix. Originally, I had the row pointer, column indices and values as private members. However, since the given large sparse matrix is already stored in row compressed form, we can exploit that and construct the matrix simply by appending the numbers one by one. Therefore, I exposed those member fields to public. It would be a better design strategy to encapsulate those member field and use add element function to perform the construction operation. However, the current implementation can achieve better performance. Then, on the next level, I wrote the functions that perform matrix operations which are not member functions or properties regarding to that specific matrix but rather the interactions between two matrices such as addition or multiplication. The next layer is the Jacobian solver, which uses those matrix operations. On the top level is the test program and the main program.

**Testing**

The testing strategy we employed is unit testing. We test each function one by one before moving onto the larger operations. Since Jacobian uses many matrix operations, we want these smaller functions to be thoroughly tested. In addition, we use Wilkinson’s principle to check the implementation of CSR matrix by checking values of the smaller matrix implemented using CSR form and the full matrix representation as in the previous homework. This is particularly important since we cannot use full matrix representation on a large matrix, we need to rely on these functions being correctly implemented. The testing for this programming assignment focus more on the functionalities of each of the matrix operations.

Since the program needs to evaluate the computation time of the CSR matrix, I did not include testing in the main driving program since that will add computation cost. Therefore, to perform testing, I included the testing operations in the beginning of the program by constructing sample matrix and pass them through the testing functions. Once the testing functions all passed, the program proceed to perform Jacobian solvers and evaluate the computation time and memory usage.

All of the tests are direct tests and white box tests. To further check for corner cases and perform a more thorough testing, we can include randomized testing in the future.

**Results**

The results are shown in the standard output as in Appendix I. The Jacobian iterative solver was able to converge in 5 iterations for the first two tests and in 7 iterations in the third test. These three test cases differ by the input vectors of b. As seen in the result, different choice of vector b will affect the performance. This is because that the convergence criteria of the iterative solver depends on the residual norm. Therefor, by adjusting b, we can change the number of iterations it takes to converge.

Code Appendix I. Standard Output

Begin Testing

Testing Completed

test 1 begin

Number of iterations: 5

Error = 6.04560606e-11

Computation time: 6.749698 seconds.

test 1 completed

test 2 begin

Number of iterations: 5

Error = 6.90653696e-11

Computation time: 7.497673 seconds.

test 2 completed

test 3 begin

Number of iterations: 7

Error = 8.83959565e-12

Computation time: 6.651056 seconds.

test 3 completed

virtual memory usage: 137233334638.500000