Assignment 1 Report

Implementation

Sparse Matrix Class Implementation

Before implementing the Gauss-Seidel algorithm, we created a class called `SparseMatrix` which stored the entries of a matrix as two vectors of vectors. The constructor for this class used 4 parameters: the number of rows, the number of columns, the vector of vectors of unsigned integers which stored the column position of the non-zero entries of the matrix and the vector of vectors of doubles which were the corresponding non-zero entries.

Once we created this constructor, we required certain functions to manipulate sparse matrices. The `add_entry` function, which accepted the column number, the row number and a value as parameters, allowed us to replace the entry of the ith row and jth column of the matrix with the value input. The function first iterates through the ith column vector of the matrix to find whether the position in the matrix is already defined, and depending on whether the entry is already defined, either adds or replaces the entry with the new input.

The `get_entry`, `printV` and `printDense` functions were created to ensure the implementation of the `add_entry` function was correct and allowed us to test the implementation on smaller examples of matrices.

Certain problems arose during the implementation of this class due to incorrect C++ syntax. One particular problem was arose when trying to iterate over empty STL vectors which is needed for certain functions in the class. This particular problem was remedied by including particular `if-else` conditions which gave separate instructions for the case when the column vector was empty and when it was not.

Gauss-Seidel Algorithm

After making the `SparseMatrix` class, we implemented the Gauss-Seidel method of inverting a matrix using the algorithm found in the Wikipedia article.

To implement the algorithm, we first set a constant tolerance to check convergence and then looped until the residual error became less than the tolerance. To find the approximate value of each entry `x[i]`, we first initialised the variables `sum = 0.0` and `a_ii = 0.0` and then iterated through the `column_position_[i]` and `values_[i]` vectors. We noted that this ensured that an error would occur unless `a_ii` was changed. As the matrix must have non-zero diagonal entries for the algorithm to work, the `a_ii` was always changed before the algorithm completed. We note that iterating through both vectors simultaneously allows us to access both the value and the column position of that value easily.

We also noted that convergence is only sure when A is symmetric positive definite or A is strictly or irreducibly diagonally dominant.

Testing Implementation

Tests Varying N

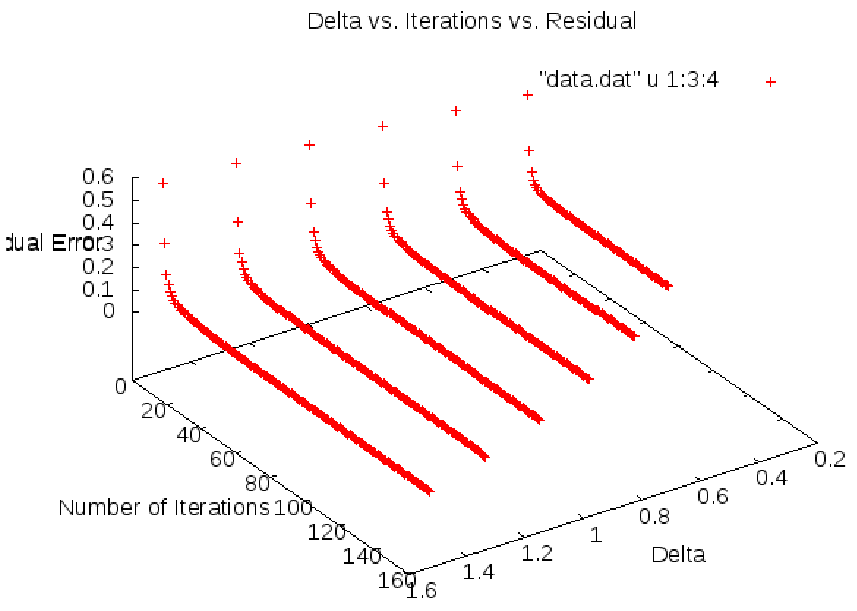
The first set of tests of the Gauss-Seidel algorithm involved fixing `delta` = 1.0, `lambda` = 0.0 and noting the difference between the results when N = 100, 1000 and 10000. In the implementation of the Gauss-Seidel algorithm, the process would stop if either the number of iterations exceeded 10e6 or the residual error became less that 1.0e-6. The follow results were obtained:

N = 100:	Number of iterations: 6664 Error: 0.00103237
N=1000:	Number of iterations: 186647 Error: 0.101512
N=10000:	Number of iterations: 242319 Error: 0.819303

We note that as N was increased, the number of iterations required make the residual error sufficiently small increased and the error (the sup-norm difference between the actual solution and the approximation) increased.

Tests Varying Delta

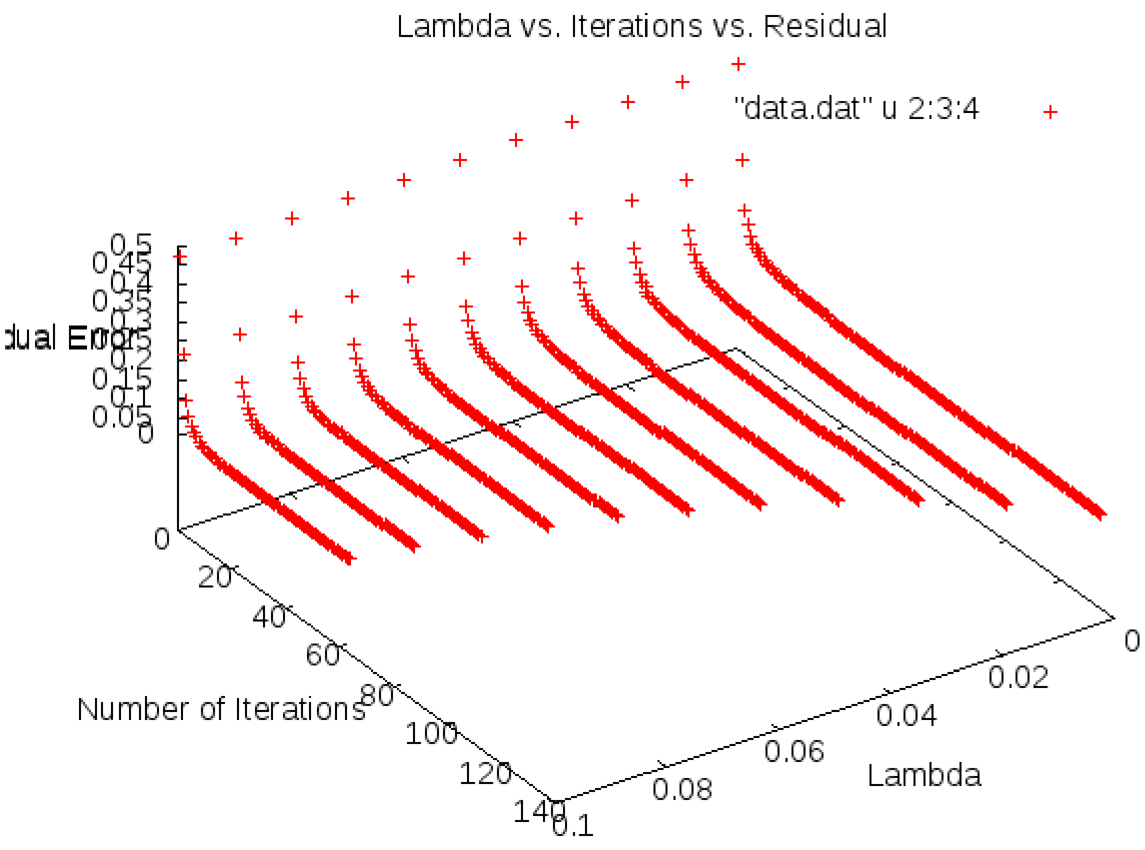
In the second set of tests, we investigated how the rate of convergence of the Gauss-Seidel algorithm was affected by changing delta. We fixed N = 10, and lambda = 0.0 and varied delta between 0.25 and 2.0 in 0.25 increments. From the graph we can see that the number of iterations required for convergence decreased as delta became smaller.



The number of iterations for `delta = 0.25` was 82 while the number of iterations required when `delta = 1.5` was 155.

Tests Varying Lambda

In the final sequence of tests, we again fixed N = 10, `delta` = 1.0 and varied `lambda` between 0.0 and 0.1 in 0.1 increments. After experimenting with larger numbers of `lambda`, we realised that very small increases in the value of `lambda` led to large decreases in the number of iterations required for the residual error to be less than the tolerance. This can be seen in the following graph.



This phenomenon is related to diagonal dominance of the matrix. It is known that one of the criteria for convergence of the Gauss-Seidel method is strict diagonal dominance. By increasing `lambda`, only the values on the diagonal are increased which leads to a faster rate of convergence. This can be seen by the difference in the number of iterations for the case when `lambda` = 0.0, which required 135 iterations while the case when `lambda` = 0.1 required 64 iterations.