EX 3: Task 2

NSSC1

December, 2024

Group Members:

- Marcus Konrath, 01300688
- Uğur Cem Erdem, 12346958
- Katharina Perl, 01426575

Conjugate Gradients in Eigen

For a brief sketch of the CG algorithm and the types of matrices used for this assignment, see the report for task 1.

This report discusses the differing results of three implementations of the CG algorithm:

1. Our own implementation as presented in task 1 (non-preconditioned).
2. *Eigen's* Conjugate Gradients using DiagonalPreconditioner
3. *Eigen's* Conjugate Gradients using IncompleteCholesky

Given that both of Eigen's methods are preconditioned, we can already expect a better convergence rate for these based on that fact alone.

BCSSTK11.mtx

The matrix to be used for task 2 differs from the one in task 1: we are now using BCSSTK11 from the same online repository "Matrix Market". This matrix measures 1473 x 1473 and contains 34241 non-zero elements, which makes it a sparse matrix with roughly 1.6 % of non-zero entries. Due to the symmetry of the matrix, only the lower triangular part is actually stored in the file BCSSTK11.mtx (which holds 17857 entries).

The structure plot of the matrix is shown in Figure 1.

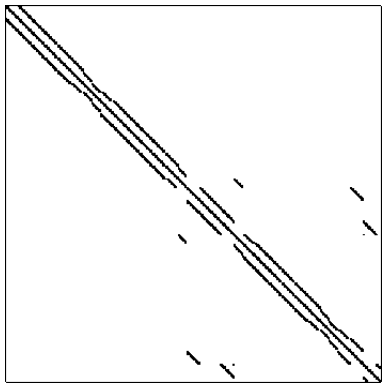

Figure 1: Structure plot of BCSSTK13
https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/bcsstruc1/bcsstk13_lg.html

Comparison of different versions of CG algorithm

The following indicator of the algorithm's performance was calculated:

$$\frac{||r_k||_2}{||r_0||_2} \quad \text{(residual norm)}$$

Figure 2 shows the corresponding plot of the residual norms against the number of iterations for each of the three versions of the CG algorithm. The maximum number of iterations was set to 100.

Firstly, we can see that Eigen's IncompleteCholesky implementation performs best in terms of convergence, followed by Eigen's DiagonalPreconditioner. Our own implementation ranks last, which is likely due to the fact that our version did not use any preconditioner which could have improved convergence rates.

Secondly, we can observe a pattern in the plot's movement similar to that observed in task 1, in that the graph is generally falling. One important difference in plotting, however, is that with task 2 we utilized a logarithmic scale on the y-axis, as otherwise the comparison between the three versions of the algorithm would have been cumbersome (cf. Figure 3).

For the own implementation of the algorithm, the strongest gain in accuracy can be found within the first 1 or 2 iterations. After about the 70^{th} iteration, there is almost no gain in accuracy, next to some oscillations. With Eigen's implementations, the increase in accuracy is strongest within approximately the first 40 iterations. Afterwards, accuracy still increases, but at a slower rate. Overall, just as with task 1, the graph's behavior indicates that the algorithm is converging towards the actual solution of the equation to be solved.

The initial guess for the algorithm was again set to $x^* = [0, \ldots, 0]$, so we can see a similar behavior of the graph's initial value being relatively high. The expected result vector was again $x = [1, \ldots, 1]$.

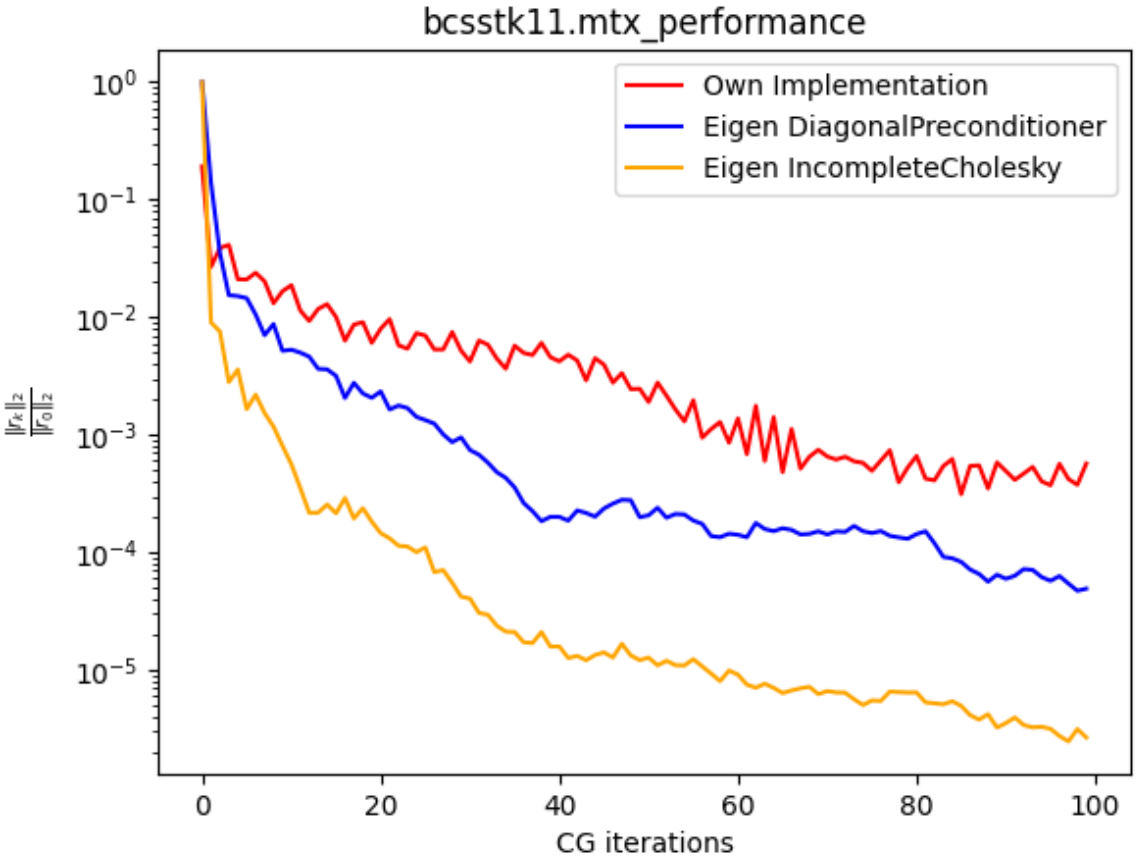

Figure 2: Assessing CG performance using residuals (log scale)

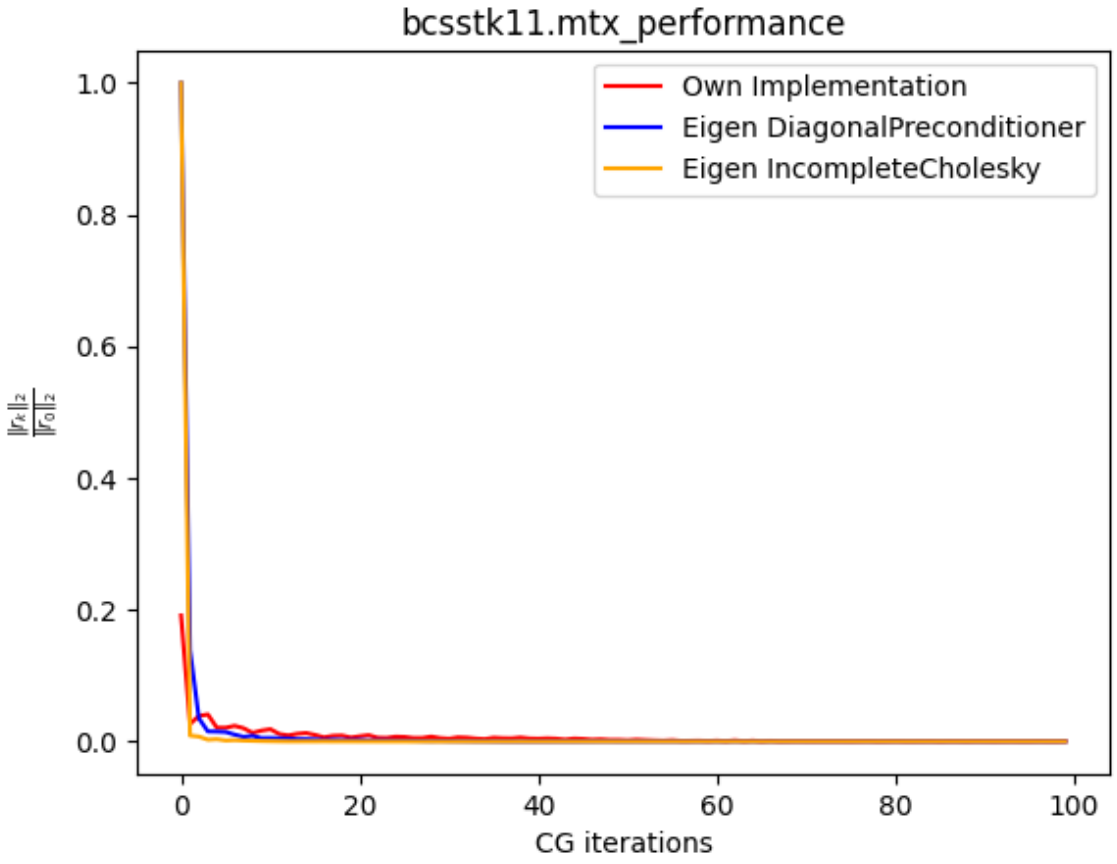

Figure 3: Assessing CG performance using residuals (linear scale)