Assignment 1: High Performance Dense Linear Solver

Christian Wiskott, 01505394.

April 10, 2020

Part 1: Triangular Solve

In this part of the assignment, the goal was to write efficient algorithms for solving a large linear system of the form $Ax = b$, for the vector $x = [1...1]^T$ with A being a non-singular, randomly generated, triangular matrix. Two scenarios were evaluated. First, A being a lower triangular matrix L, where the subsequent system $Lx = b$ was solved by *Forward Substitution*. Secondly, A as a upper triangular matrix U, treated with *Backward Substitution*.

The matrix A was generated using a uniform distribution featuring randomly chosen values within the interval of $[90, 100)$. This interval was chosen in order to obtain a reasonable condition number for L, which proved to be sufficiently small to result in reasonably small forward errors and residuals of the computed solutions. In order to provide a reference value, the condition numbers of L and U at size 1000×1000 showed values of approx. $\kappa = 7400$.

Performance Evaluation

In order to evaluate the performance of the written code, the run-time was measured, such that only the actual computation of the solution vector x was included in the measurement. Hence the two for-loops which scale with $\mathcal{O}(n^2)$ which can be seen in the provided code. Fig. 1 shows the run-time analysis of the *Forward-* and *Backward Substitution* for problem sizes within the range of [100, 20000]. Both have been fitted with a quadratic function, which despite the various outliers, generally show a quadratic increase w.r.t the matrix dimension, which coincides with the expected $\mathcal{O}(n^2)$-scaling. Both algorithms completed the given tasks within the ranges of a few hundred milliseconds, with *Forward Substitution* being generally a bit faster.

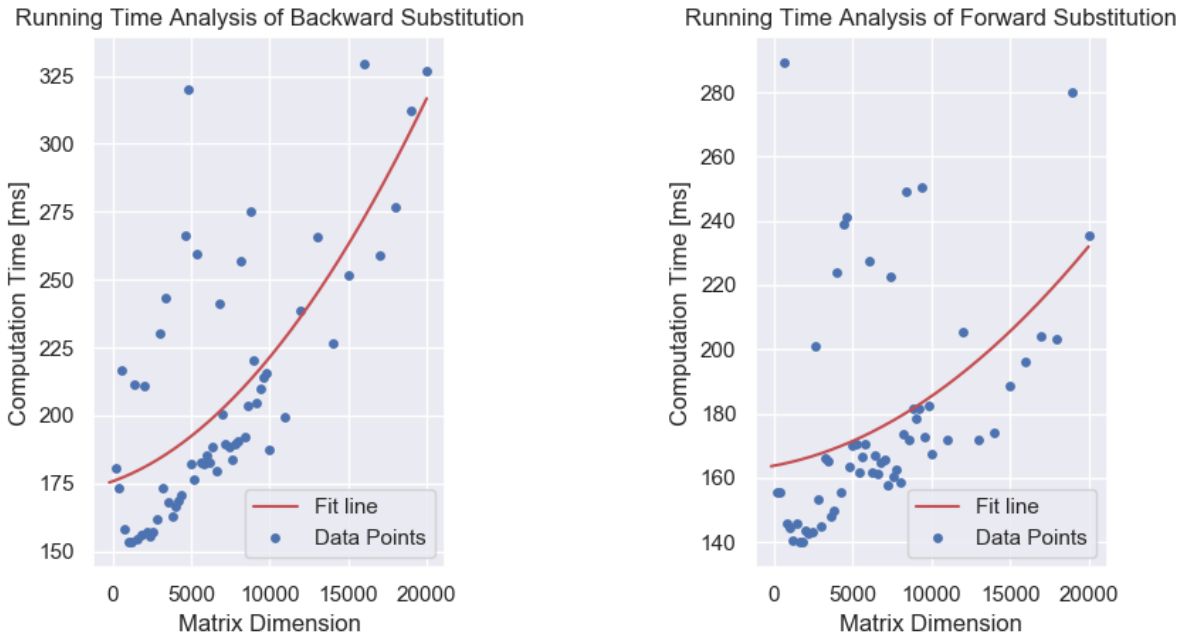

Figure 1: Run-time Analysis

These results are of similar magnitude compared to similar problem sizes computed in *FORTRAN*. This was made possible by using the module $< numba >$, which translates the code into much faster machine code, thus combining the advantages of compiled and interpreted languages. This led to a increased performance by a factor of approx. 100.

Fig. 2 show the relative forward errors and relative residual norms of the computed solutions w.r.t the matrix dimension, which are very similar for both algorithms. At the maximum dimension of 20000, the results still feature relative forward errors of $\sim 10^{-6}$ and relative residuals of $\sim 10^{-15}$.

In order to provide a reference value for the run-time, the code was executed on the server the university provided. Fig. 3 showcases the comparison. The performance is aprrox. 100-times slower because the module $< numba >$ had to be disabled, due to the server not featuring this package and completed the tasks for size $10k \times 10k$ in around 35 seconds.

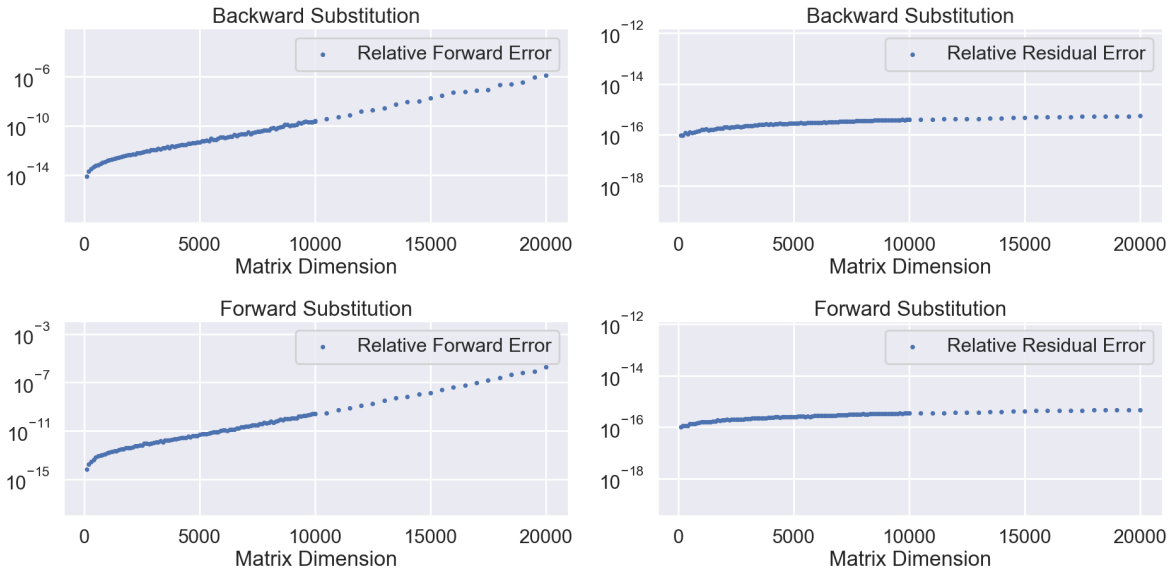

Figure 2: Error Analysis

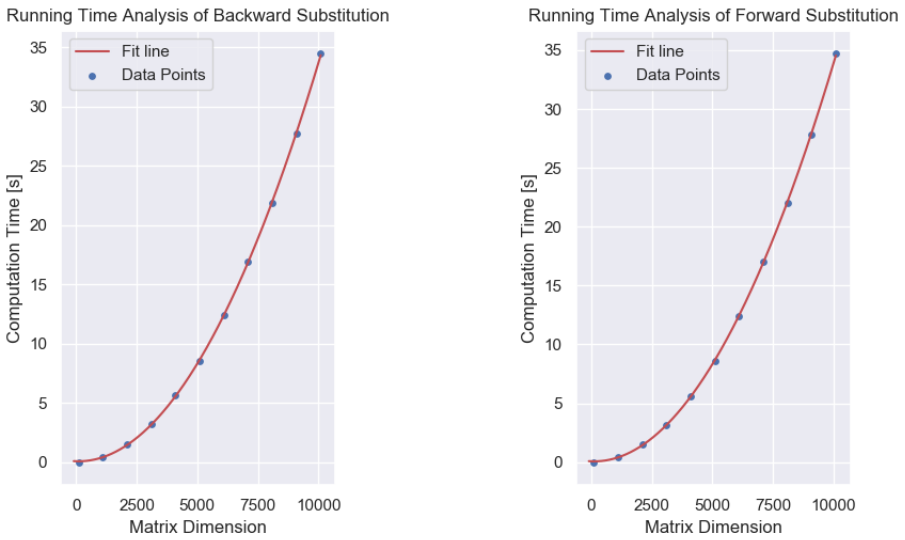

Figure 3: Run-time analysis of both algorithms on the provided university server