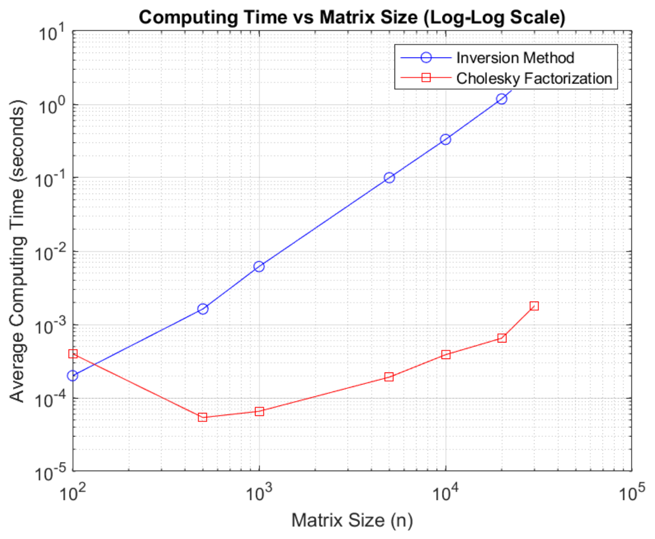
MACM 316 Computing Assignment 2: Methods of Solving Linear Systems - Timing Analysis

This report investigates the computational efficiency of solving linear systems in the form of $A_n x = b_n$. The methods explored here are the Matrix Inversion method and Cholesky Factorization. To achieve efficient results and reduce computer memory usage, we generate random sparse matrices of A_n (size n x n) of various n and corresponding right-hand side vectors, b_n for various values of n, where n = {100, 500, 1000, 5000, 10000, 20000, 30000}.

Part A) The objective is to measure the average computing time required to solve for using the following matrix inversion code:

$$A_{inv} = inv(A_n);\ x = A_{inv} \cdot b_n;$$

To ensure the reliability of our results, 20 executions of the inv() for each matrix size, n, are computed and calculate the average execution time by taking the mean across the number of runs.



The blue line, as seen in the log-log plot, is representative of the relation of the average computing time (y axis) vs the size of the n x n matrix (x axis). As it is plotted on a log-log plot, the power law can be applied to analyze its growth rate using Big O representation.

$$log(Avg\ Compute\ Time) = k \cdot log(n) + constant$$

The slope, k, is representative of the power-law exponent which identifies the nonlinear change between two variables; in this case it is the cost (time) growth of the method. Using the polyfit() in matlab, it is discovered that the slope (k) of sparse matrix growth plot is $k \simeq 1.7903$, meaning the cost (time) of the inversion method grows in comparison to matrix of size n by $O(n^{1.7903})$ or $\simeq O(n^{2})$. In essence, the time taken to solve a linear system by matrix inversion grows polynomially the larger the size of the matrix.

Part B) The objective is to measure the average computing time required to solve a linear system using the following Cholesky Factorization code:

$$R = chol(A\_n,\ 'lower');\ x = R' \setminus (R \setminus b\_n);$$

Again, 20 runs of chol() are executed per n entry and averaged over. Based on the results plot, the red line seen, Cholesky Factorization is significantly more efficient solving with sparse tridiagonal matrices. Using the same method to find the inversion method's big O relation, we can use power law and polyfit to model Cholesky Factorization cost growth. By this, we find the slope, $k \simeq 0.90673$; therefore, an approximate of Cholesky Factorization's Big O can be modeled as $O(n^{0.907})$ $or \simeq O(n)$. Based on the found Big O relation and the data displayed from the plots, the growth of compute time is less steep the larger the size (value of n). This means greater efficiency regardless of matrix size, both time and computer resource wise.

By observing the plot and data obtained, it is determined that Cholesky Factorization vastly outperforms the Inversion Method when comparing the average compute time of the two methods. This is largely attributed to the fact that Cholesky Factorization solves linear systems with triangular matrices, rather than full matrix inversion. Another fact to consider is the use of sparse matrices. The inversion method densifies matrices to use for its calculations which greatly increase compute time; whereas Cholesky's maintains sparse matrices throughout, thus presenting vastly quicker compute time. There is also the note of numerical stability. While Inv() provides a valid solution, it is known to become numerically unstable while computing with large matrices; however, Cholesky Factorization is significantly more stable due to its decomposition approach.

Concerning the standardization of this experiment, there are several variables that can influence the outcome of these results; particularly relating to computer hardware. The nature of solving linear systems of large size matrices is computationally demanding. The reader may note that the first test of the Cholesky Factorization compute time is significantly larger than its second. Given that size n(1) < n(2), this should not be the case. The concept of Just-In-Time (JIT) compilation in addition to memory allocation and CPU processing power can cause small, but noticeable outliers in the data. This report, and the related MATLAB code takes this into account and uses a "warm up" run before each method to mitigate discrepancies related to JIT. It is also important to note that MATLAB is highly optimized at computing with sparse matrices and may contribute to the more optimistic Big O results.