PHYM004 Assessment 1: Matrix Operations

Jay Malhotra

October 23, 2021

1 Introduction

The goal of this project was to investigate how the computational costs for matrix operations scale based on the method used. The matrix operation that I chose to examine was the calculation of the determinant.

Two methods were compared. The first one was the typical formula for finding the determinant of a square N by N matrix:

$$\det(A) = a_{11}C_{11} + a_{12}C_{12} + ... + a_{1n}C_{1n}. \tag{1}$$

Here, C_{nm} is the cofactor of element a_{nm}, given by calculating the determinant of the minor matrix of a_{nm}.

The second method used was to perform an LU decomposition of the matrix using Doolittle's method. The determinant, then, is given by

$$\det(A) = \det(L)\det(U). \tag{2}$$

Because of the properties of L and U, this becomes

$$\det(A) = \left(\prod_{i=1}^{n} l_{ii}\right)\left(\prod_{i=1}^{n} u_{ii}\right). \tag{3}$$

2 Testing methodology

The program generates a large square matrix corresponding to the maximum value of N. It fills this with random floating point numbers ranging from -10 to 10. Then, in a loop spanning $2 \leq i \leq N$, it takes the ith leading matrix of the master matrix and performs the two calculations on that. This saves having to regenerate random numbers each time the matrix size is increased.

The elapsed time was measured using `clock.gettime` from `time.h`, which should have precision on the order of nanoseconds.

Curve-fitting analysis was performed using `scipy.optimize.curve_fit`, which uses a non-linear least-squares regression algorithm.

3 Results

Figure 1 is a plot showing how the time of each operation scales with the N – i.e. the number of rows/columns of the square matrix.

3.1 Brute-force

The brute-force method was found to scale very poorly. A 13x13 matrix took almost 15 minutes to invert via the brute-force method, and further data collection was impossible due to time constraints. This awful scaling is due to the recursive nature of the method; as per Equation 1, taking the determinant of an NxN matrix requires taking the determinant of N (N-1)x(N-1) matrices, which itself requires taking the determinant of (N-1) (N-2)x(N-2) matrices, and so on and so forth. So, theoretically speaking, the time taken should scale with $N!$.

I performed curve-fitting analysis to the function $T(N) = aN!$. I found $a = 1.23 \times 10^{-7}$ s, which results in a fairly close fit, as seen in Figure 1.

3.2 LU decomposition

As the LU method does not scale as poorly, and can easily go up to matrices with hundreds of rows and columns without breaking the 1-second barrier, I have included a separate plot, Figure 2, which shows only the LU method.

I performed curve-fitting analysis of the extended LU timing data to the function $T(N) = aN^b$. I found $a = 2.64 \times 10^{-9}$ s and $b = 2.99$. This is again a fairly close fit, as seen in Figure 2, but similarly to the last fit there is divergence at lower sizes. This may be due to small overhead processing costs that are not accounted for by the scaling relation.

4 Conclusion

The final scaling relations found were $T \propto N!$ for the brute-force method, and $T \propto N^3$ for the LU decomposition method. Clearly, the brute-force method is inappropriate for anything except very small matrices, where it may have an advantage over the LU method as some matrices may not accept LU decomposition without row swaps.

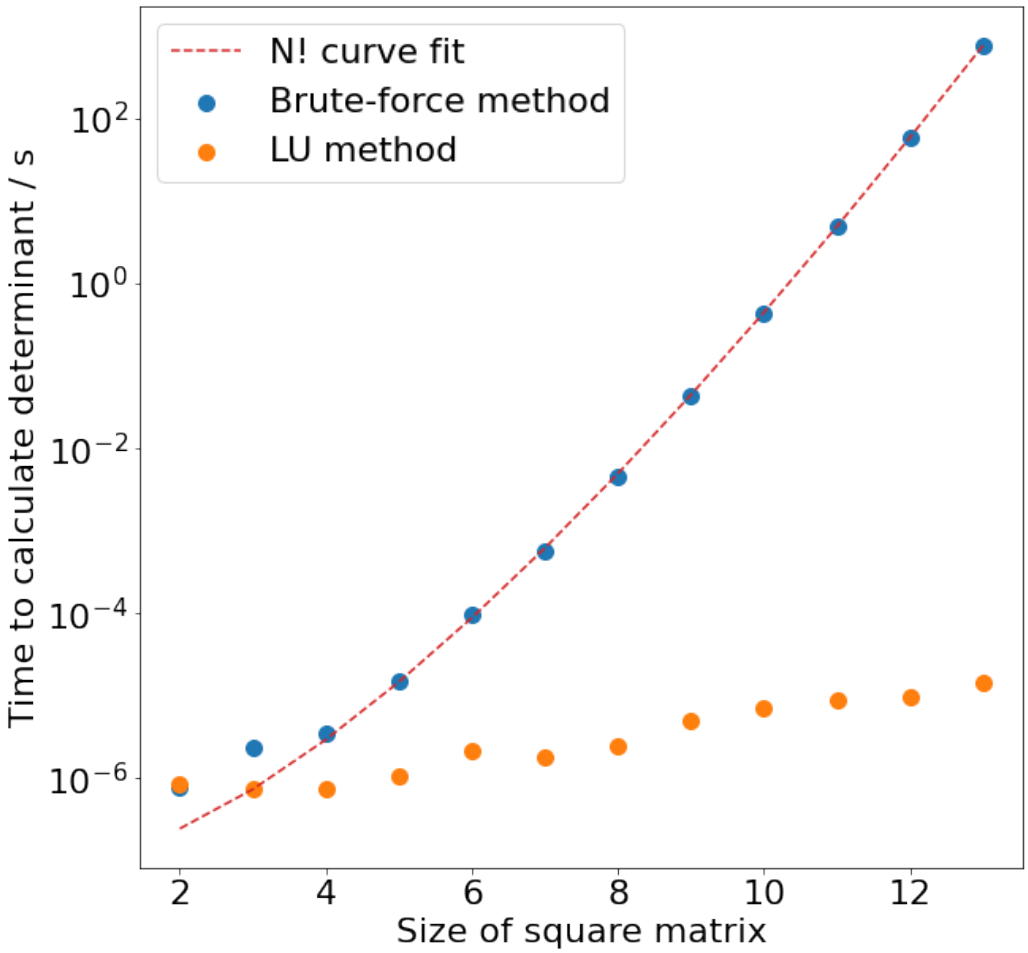

Figure 1: x-log(y) plot showing how the time to calculate the determinant scales with matrix size, depending on the method used. The x-axis is not log scaled due to the small range and number of data-points. The results of the curve-fitting analysis for the brute-force data are also plotted.

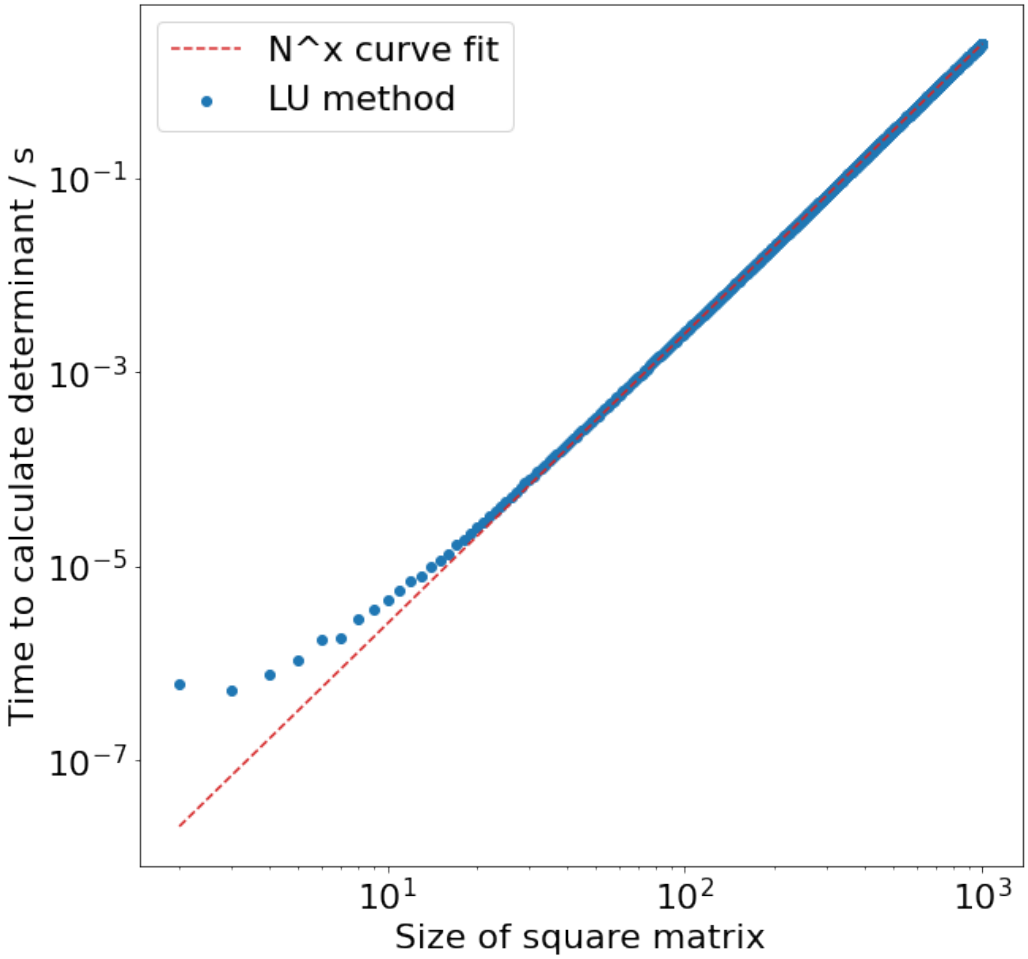

Figure 2: Log-log plot showing how the time to calculate the determinant scales with matrix size for the LU method, with an expanded range of sizes compared to Figure 1. The results of the curve-fitting analysis are plotted against the data.