Computational Physics Project 7

Jack Messerli-Wallace

March 31, 2023

1 Matrix Operations

1.1 Introduction

In this work, we use both “homemade" matrix operation methods and those from LAPACK to compare computational efficiency. Changes made to the base program `matrix-all-new.f` and computational results will be discussed.

1.2 About the Program

Changes made to modernize the `matrix-all-new.f` program include rewriting the program in Fortran 90 and adjusting all data declaration to be from the `ALLOCATE` function. The LAPACK functions added are `DGETRI` for general matrix inversion and `DSYTRI` for general matrix diagonalization. Execution time is measured using the native `cpu_time()` function. We only time the function call, as opposed to including the memory allocation time or any other process.

1.3 Timing Results

Matrix dimension sizes tested range from 100 to 1000 to understand the scaling of the functions without converting our machine into a space heater.

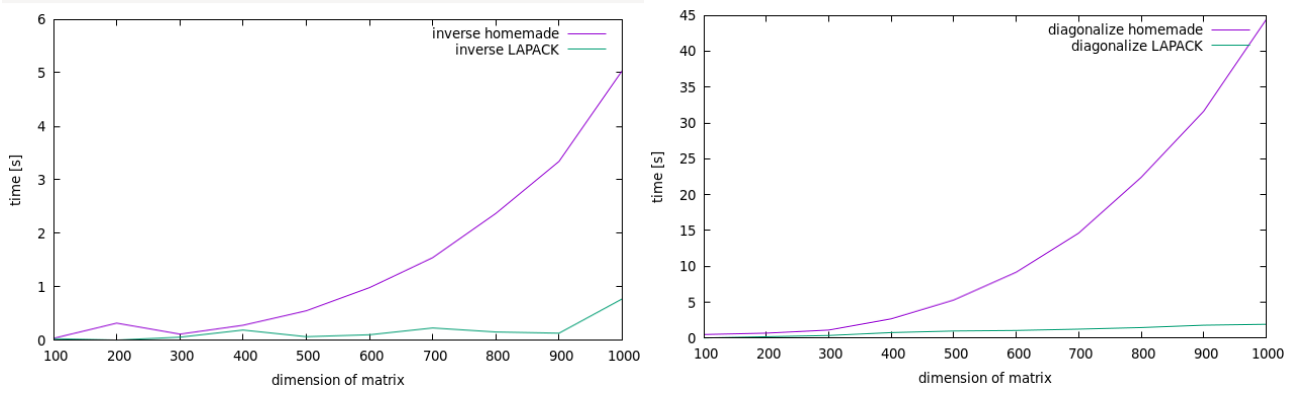

Figure 1: Matrix inversion time (left) and diagonalization time (right)

Note that in figure 1, the LAPACK functions appear to rise linearly with the increase in matrix dimension. We know this to be false, motivating further exploration with just the LAPACK functions.

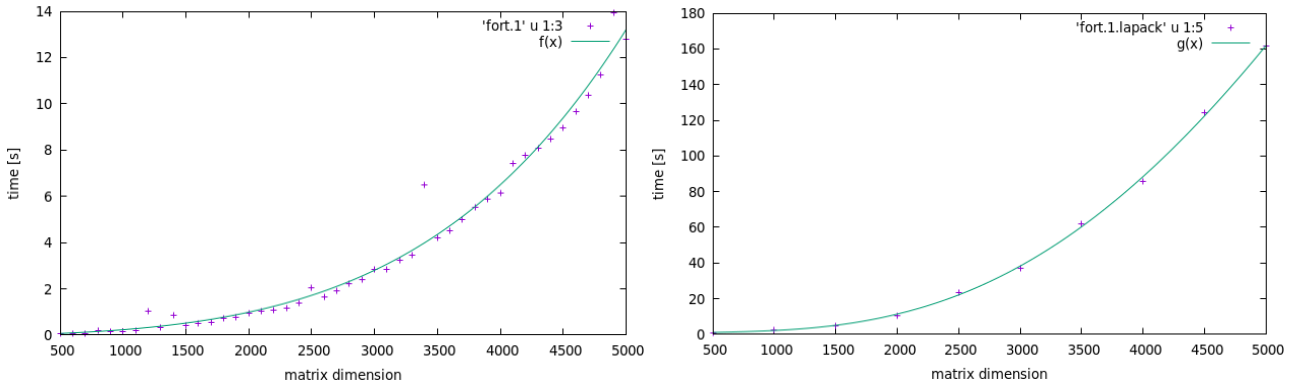

Figure 2: Matrix inversion time (left) and diagonalization time (right) using only LAPACK functions

Note that both curves in figure 2 are the data of the respective function fit the curve $y = ax^4 + bx^3 + cx^2 + dx$, where x is the matrix dimension and $a, b, c,$ and d are the fit constants. Note that the curve in the figure is the data fit the curve $y = ax^4 + bx^3 + cx^2 + dx$, where x is the matrix dimension and $a, b, c,$ and d are the fit constants.

1.4 Conclusion

While in high dimensions both “homemade" and LAPACK functions scale to similar levels of computational complexity, the difference in overall efficiency and raw speed in reasonable dimensions is cause to use LAPACK matrix operations whenever possible.