Comparison of Matrix Algebra Computational Performance Between RcppEigen and Base R

Costa Stavrianidis
GitHub Repository - https://github.com/Costa-Stavrianidis/BIOS823_Final

Abstract

In this study, we compare the performance of various matrix algebra computations across functions created using the RcppEigen package and functions in base R. The goal is to quantify and then visualize the efficiency differences between the two for computations that complete the same goal. Square matrices of dimensions 10, 50, 100, 200, and 500 were simulated from random normal distributions, with 100 replications of each computation performed for each size matrix. RcppEigen functions were faster than their base R counterparts for all computations on the largest size matrix besides one, with the increase in efficiency overall becoming more drastic as the dimensions of the matrix increased.

Introduction

Compiled programming languages have their programs compiled into machine-readable instructions before execution. Examples of compiled languages include C, C++, Rust, and Fortran. Interpreted languages have their programs read and executed by an interpreter rather than translating the program into machine-readable instructions. Examples of interpreted languages include Python, R, and JavaScript.

Both types of languages contain their own advantages and disadvantages. The advantages of creating a compiled program are that it is faster than an interpreted program at execution, and you can optimize your code depending on the machine you are using. The main disadvantage is that it may have platform dependence for the machine-readable code it generates. For interpreted languages, the advantages are that the code is platform independent, it allows for dynamic typing, debugging is typically easier, and the program sizes tend to be smaller. The main drawback is that the code executes slower than compiled languages.

Eigen is a high-level C++ library that allows a user to perform matrix and vector operations, among many other things. The RcppEigen package (Bates and Eddelbuettel 2013) integrates this library into R using the Rcpp package (Eddelbuettel and Francois 2011). Users can utilize the RcppEigen package within their R environment to gain access to the various data structures and functions available in the original C++ library, as well as gain the advantage of the speed of using a compiled language such as C++.

Methods

Six functions were written in RcppEigen and then used within R to test their efficiency. An equivalent (in terms of what the computation was doing at a high-level) six base R functions were compared to these functions. First, to make sure the functions were performing the correct calculation, results on simulated data from the equivalent functions across RcppEigen and R were checked to be equivalent. All functions had equivalent results except for one that utilizes the Cholesky decomposition, which requires a positive definite matrix for inversion. Since accuracy was not compared in this study, this result was ok to use.

To compare time efficiency between the RcppEigen and base R computations, the 12 computations were tested on simulated square matrices of sizes 10, 50, 100, 200, and 500, each 100 times. Times to execute the computation were recorded in microseconds, and then compared across RcppEigen functions and R functions.

Results

Table 1: Time in Microseconds of First Set of RcppEigen and Base R Functions by Square Matrix Dimensions

Dimensions	Eigen_matrixmult	R_matrixmult	Eigen_transpose	R_transpose	Eigen_eigenvalues	R_eigenvalues
10	6,275.87	1,483.79	7,377.95	1,308.72	9,992.93	103,775.9
50	12,754.28	42,850.33	4,817.91	4,784.29	130,097.10	377,177.0
100	78,057.85	376,446.42	16,197.46	15,932.19	676,720.99	1,553,491.6
200	582,297.17	2,887,795.64	84,524.78	59,731.67	4,257,808.59	9,007,790.6
500	7,860,758.62	41,385,326.61	606,312.51	580,156.56	67,091,989.59	127,806,036.4

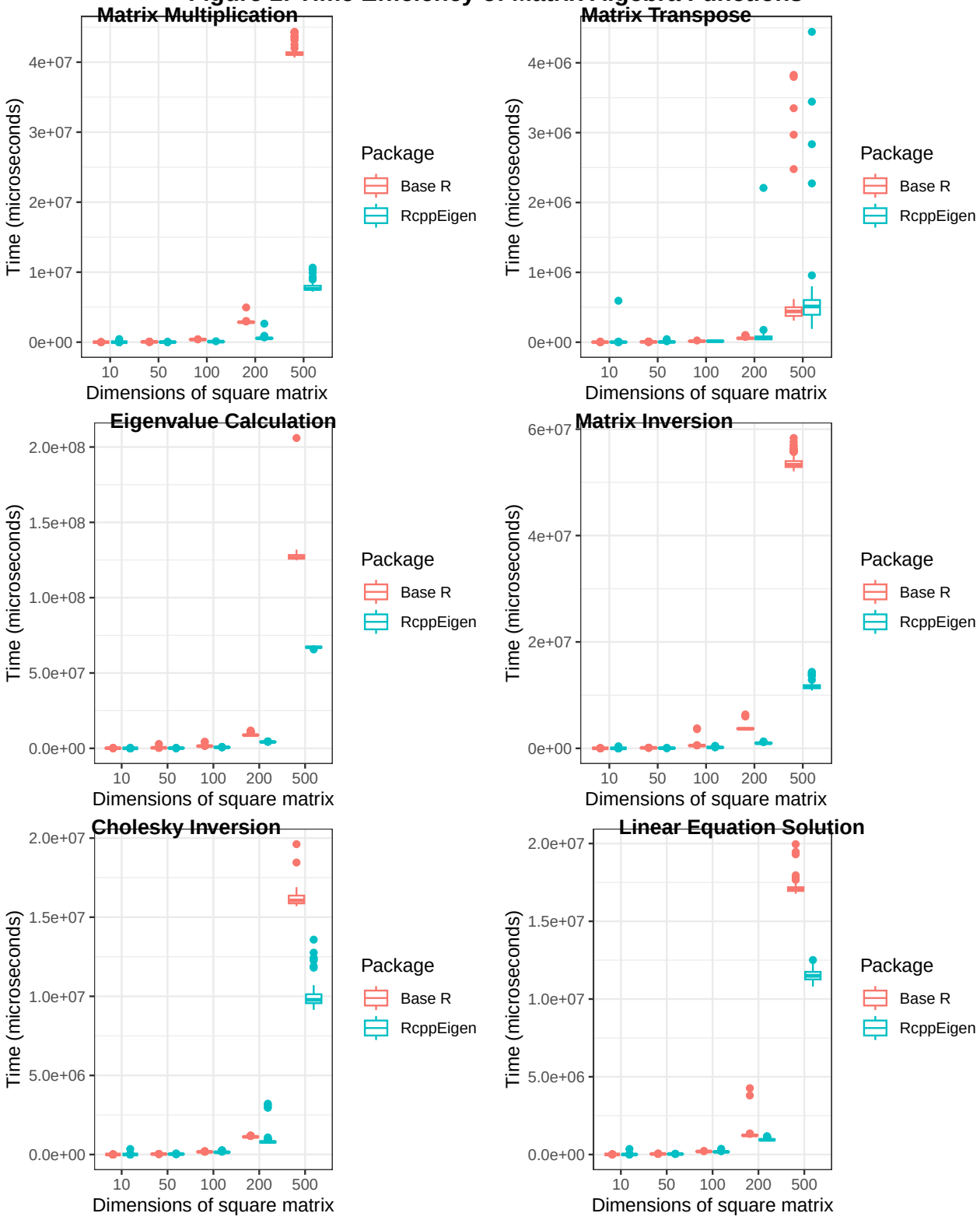
Table 1: Average time in microseconds over 100 replications to execute for various matrix algebra functions using RcppEigen and base R. Dimensions column indicates dimensions of square matrix used for computation. Matrixmult computes product of matrix multiplication between two matrices, transpose transposes a matrix, and eigenvalues computes the eigenvalues of a matrix.

Table 2: Time in Microseconds of Second Set of RcppEigen and Base R Functions by Square Matrix Dimensions

Dimensions	Eigen_eigeninv	R_eigeninv	Eigen_cholinv	R_cholinv	Eigen_linearsolve	R_linearsolve
10	6,749.42	10,399.24	6,277.51	1,829.42	6,469.39	5,954.43
50	33,639.68	89,282.01	28,679.09	29,041.12	33,231.73	37,142.72
100	179,551.30	579,419.79	151,838.17	170,789.19	175,802.26	195,267.01
200	973,603.63	3,743,459.90	894,142.35	1,121,197.48	960,948.16	1,291,058.02
500	11,687,583.82	53,758,201.65	9,987,049.37	16,178,807.46	11,510,142.38	17,158,308.12

Table 2: Average time in microseconds over 100 replications to execute for various matrix algebra functions using RcppEigen and base R. Dimensions column indicates dimensions of square matrix used for computation. Eigeninv inverts the matrix, cholinv inverts the matrix using the Cholesky decomposition, and linearsolve solves the linear equation Ax = b when given A and b by taking the inverse of matrix A and multiplying it by vector b.

Figure 1: Time Efficiency of Matrix Algebra Functions



Discussion

Tables 1 and 2 indicate the average time in microseconds for each computation across RcppEigen and base R functions for different size matrices. For all computations except the transposition, the RcppEigen function was more efficient than the base R equivalent for the largest square matrix with 500 rows and 500 columns.

Figure 1 indicates the time efficiency of the different computations as the dimensions increase, with RcppEigen functions in blue, and base R in red. This allows us to visualize how the differences in efficiency are far more drastic as the size of the matrix increases. We can also see various outliers with the boxplots.

Conclusion

At lower dimensions of the simulated matrices, there does not appear to be a large advantage in speed using the RcppEigen functions, however since the times are in microseconds, the differences may even be seen as negligible. Reasons for this could be due to outliers, or the R function actually being optimized in a certain way (perhaps through use of parallelization). It should also be noted that the RcppEigen functions may not be performing the computations in the most optimized way possible.

The main differences can be seen when the size of the matrix gets large. Here, for the majority of the computations, using the compiled RcppEigen functions have a clear advantage in speed over the base R interpreted functions. To conclude, individuals who may find themselves needing to perform some sort of matrix algebra computation with a very large dataset could find it advantageous to utilize the speed of the RcppEigen functions. The benefits are more obvious as the size of the matrix increases.

References

Douglas Bates and Dirk Eddelbuettel (2013). Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package. Journal of Statistical Software, 52(5), 1-24. URL http://www.jstatsoft.org/v52/i05/.

Dirk Eddelbuettel and Romain Francois (2011). Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1-18, doi:10.18637/jss.v040.i08.