Project 2: Rank Deficient Algorithm

Jack Hamel

April 8, 2018

1 Code Overview

My code is written as a number of modules to handle different segments of the computations. The `main.py` file houses the code to run the whole calculation of potentials using both direct methods and rank deficient ones and time the "matrix-vector" product for both. This file draws on classes and functions from the other files. `tree.py` contains the tree class and functions for building, storing, and navigating the tree used to discretize the two dimensional space. `interactions.py` contains the interactions class. This class manages building, storing and navigating the interaction list. It has functions to build G matrices for interactions between any two boxes in the tree. Additionally, it has auxiliary lists for storing UV decompositions of G matrices and holding information to assist in reconstructing the approximated potentials using rank deficient methods. The `source.py` file holds the source class, which defines what it means to be a source in the system. `utilities.py` contains a mix of functions that did not warrant their own class or fit into a preexisting one.

2 Cost Analysis

Figure 1 shows the time elapsed to compute the potentials using rank deficient methods and direct calculations with 30 test points ranging between $N = 1$ and $N = 20000$. This test was done with a 4-level tree with 64-leaves at the lowest level because 4 levels provided a realistically obtainable break-even point on my machine. As the number of tree levels is increased, the break-even point increases relative to less levels. As seen in Figure 1, the fast method becomes faster at approximately $N = 10000$. Also, worth noting, is that in these runs $\epsilon = 0.1$. A lower ϵ improves error, but decreases speed by indirectly using higher rank approximations. The average error between all runs was 2.626×10^{-4}.

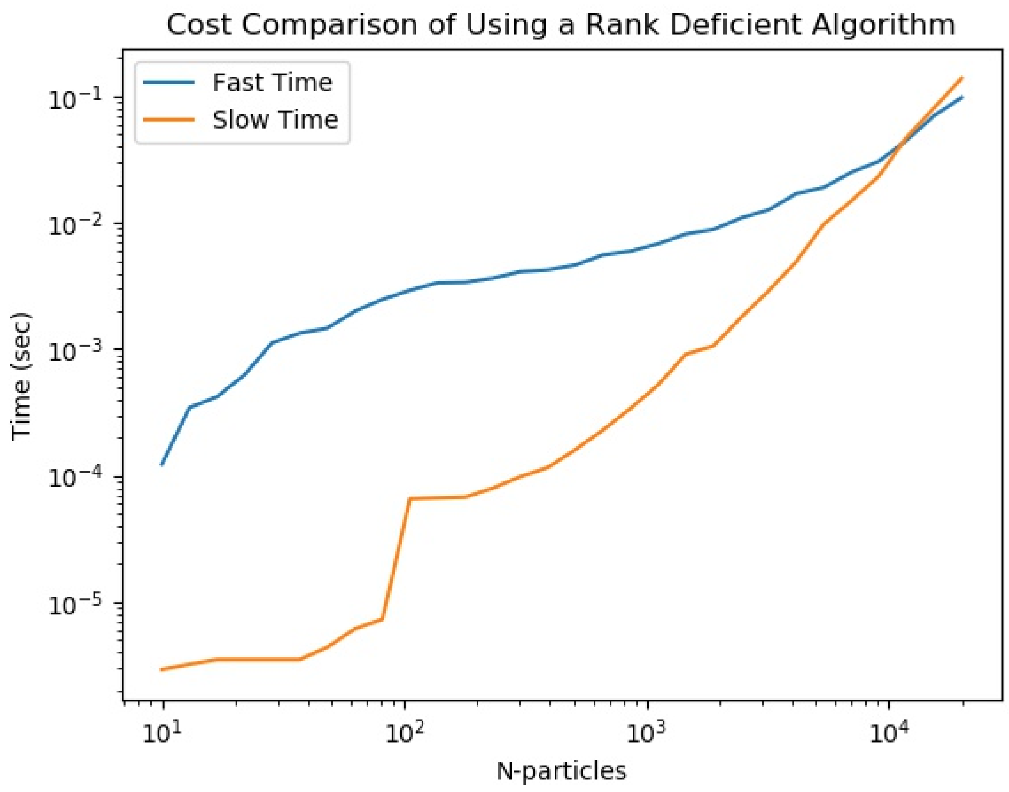

Figure 1: Cost of Calculating Potentials with Rank Deficient Methods

3 Error Control

The effects on error and cost were investigated by varying the tolerance of the rank approximation, ϵ, in the algorithm. These test were all ran with a 4-level tree and $N = 1000$ particles. In Figure 2, the computation time of the rank deficient algorithm is plotted versus ϵ. As the tolerance is increased, the computation time increases because the matrices in the matrix-vector products are smaller. In Figure 3, the effect of ϵ on error is plotted. As expected, when a higher rank approximation is used, the error is lower and in conjunction with Figure 2, the computation time is longer.

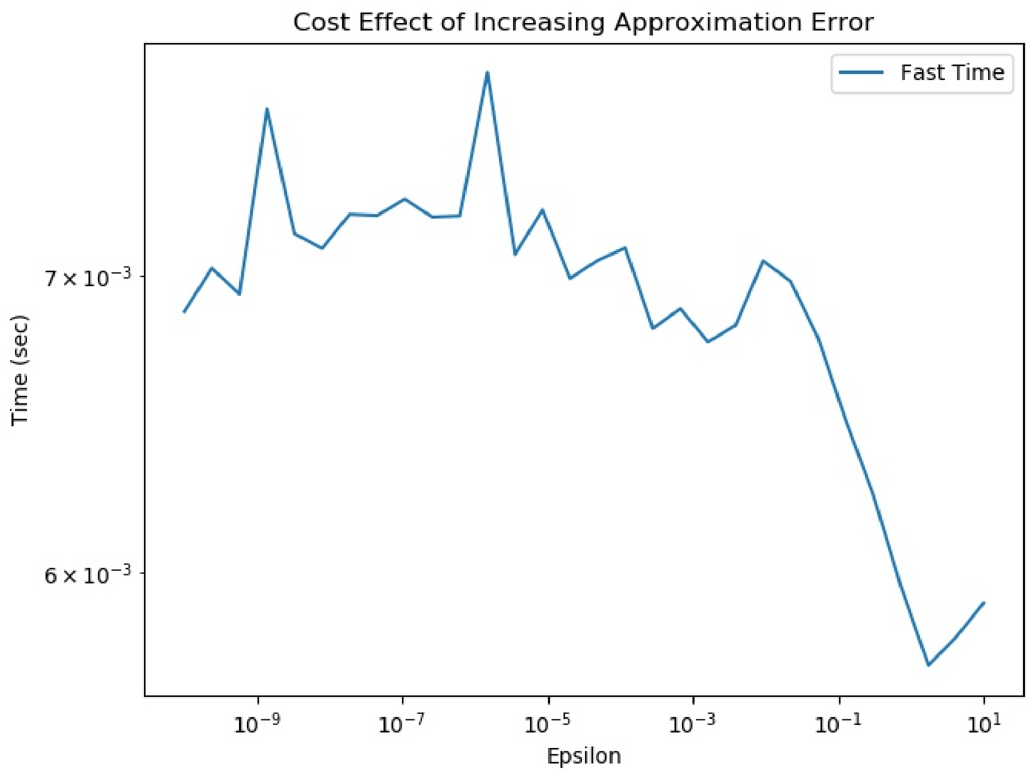

Figure 2: Effect of Approximation Tolerance on Computation Time

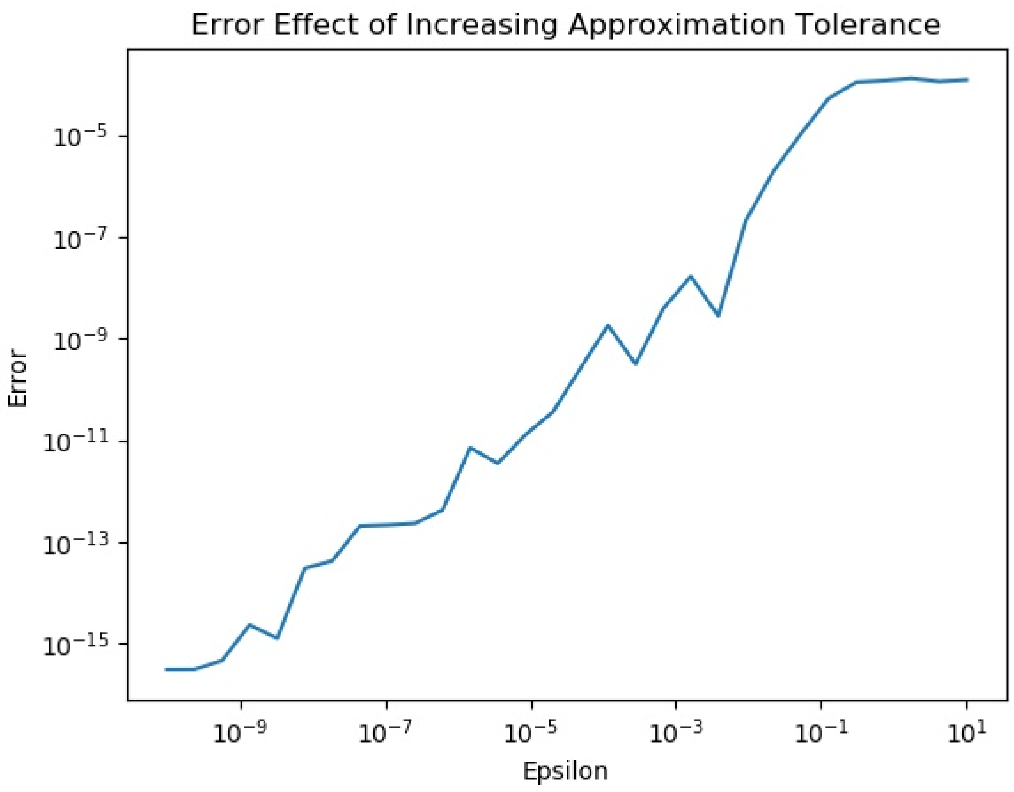

Figure 3: Effect of the Approximation Tolerance on the Accuracy of the Computation