RANDOMIZED NUMERICAL LINEAR ALGEBRA FOR KERNEL MATRIX COMPRESSION

by

Saumya Kandarp Maniar

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Mathematics

Charlotte

2020

ABSTRACT

SAUMYA KANDARP MANIAR. Randomized Numerical Linear Algebra for Kernel Matrix Compression. (Under the direction of DR. DUAN CHEN)

Matrices are used significantly as a medium to store data in many applications like Data Science, Computer Science, Statistics, and Applied Mathematics. Matrix computations like matrix multiplication, matrix inversion, eigenvalue decomposition, singular value decomposition are very substantial in real-world applications. Unfortunately, many of these matrix operations are so time and memory expensive that they are prohibitive when the scale of data is large. Sometimes, when the data has a large amount of meaningless information called noise, machine-precision matrix operations are not necessary, and one can sacrifice a reasonable amount of accuracy for computational efficiency. In addition to the applications mentioned above, in Machine Learning, Linear Algebra, Partial Differential Equations, and Optimization, the data usually boils down to an $m \times n$ matrix A, and often it is very helpful to derive matrix approximations to our original matrix A when our data is seemingly unmanageable. We try to get an approximation matrix A_k which has a particular rank k (considerably smaller than m and n), in other words, *low-rank approximation.* Methods like Singular Value Decomposition and QR Decomposition can be used to achieve such matrix approximations. But, such algorithms usually take a lot of time (superlinear in the number of nonzero elements of the matrix). We want to optimize our algorithms to be used in various applications where data sets are framed by extremely large matrices.

So, in this thesis, we primarily focus on applying a new approach called *Randomized Numerical Linear Algebra.* We will introduce two different methods, the first method known as Random projections and the second known as Random Sampling. Random projections have recently emerged as a powerful method for dimensionality reduction. We will present some experimental results carried out on matrices generated by Kernel functions and work on a specific type of kernel function called *Green's function*. We will try to execute low-rank approximation on them using Randomized Numerical Linear Algebra and show some approximation errors to better understand the nature of the new approach known as RandNLA. We will try to show that using a sparse random matrix gives additional computational savings. By projecting the data onto a random lower-dimensional subspace yields results comparable to conventional dimensionality reduction methods such as SVD and QR: the similarity of data vectors is preserved well under random projection. This approach is computationally significantly less expensive than the conventional approach.

Keywords: Singular value decomposition, Rank, QR decomposition, Spectral norm, Frobenius norm, Complexity, Taylor series approximation, Column selection, Randomized algorithms, Random projections, Random Sampling, Leverage Score Sampling, Randomized SVD.

DEDICATION

I would sincerely like to dedicate this Master's thesis to my dearest grandmother. She was always by my side throughout my childhood and her extraordinary support and encouragement motivated me to pursue my goals. I would like to pay a special tribute to her by writing this thesis.

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Dr. Duan Chen of the Mathematics and Statistics Department at the University of North Carolina at Charlotte. Working on this project gave me a lot more insight over certain complex concepts. Dr. Chen has motivated me to learn something new and provided constant support at every stage of this thesis. He has consistently driven me in the right direction whenever I needed. I would also like to thank the experts who were involved as committee members for this research project: Dr. Shaozhong Deng and Dr. Hae-Soo Oh of the Mathematics and Statistics Department at the University of North Carolina at Charlotte. Without their participation and input, this thesis could not have been conducted.

I would also like to acknowledge the members of the Center for Graduate Life and the Writing Resource Center at the University of North Carolina at Charlotte for creating a structured writing space, accountability, and providing constant guidance for students.

I wish to acknowledge my peer Kshitij Kumbar, a graduate student in the Mechanical Engineering Department at Stanford University for being there as a constant friend and providing me with tips in order to improve my programming skills.

Finally, I would like to express my heartfelt gratitude to my parents for providing me with continuous support and encouragement throughout my years of study. This accomplishment would not have been possible without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

LSI	Latent Semantic Indexing
RandNLA	Randomized Numerical Linear Algebra
SVD	Singular Value Decomposition

CHAPTER 1: INTRODUCTION

1.1 Purpose of research

In the field of Data Science, Computer Science, Statistics, and Applied Mathematics, the heavy volume of aggregating information is becoming a tedious task, and the ability to collect, store, and manage large data sets is challenging. This demands faster and better algorithms in order to extract and interpret important information from different data sets. A lot of times, we express our data in the form of an $m \times n$ matrix A, and it can be burdensome to work on matrix computations if m and n are seemingly large ($> 10^6$).

Often it is very helpful to get matrix approximations to our original matrix A, i.e., we try to get an approximation which has a particular rank k, considerably smaller than m and n, in other words, low-rank approximation. Methods like Singular Value Decomposition and QR Decomposition can be used to achieve such matrix approximations. But such algorithms usually take a lot of time which is superlinear (An algorithm which has time complexity greater than $O(n)$ is said to have a superlinear complexity) in the number of nonzero elements of the matrix ($O(n^3)$ operations are required for $n \times n$ matrices). We want to optimize our algorithms and our goal is to bring the time complexity down to $O(n^2)$, so it can be useful in applications where data sets are framed by extremely large matrices.

So, in this thesis, we primarily focus on a new approach called Randomized Numerical Linear Algebra and we will introduce two methods (1). Random projections and (2). Random Sampling, and present some experimental results carried out on matrices generated by kernel functions. Random projections have recently emerged as a powerful method for dimensionality reduction.

We will discuss kernel functions and how we can approximate them using low-rank approximation. We will show some experimental results and approximation errors to better understand the nature of the new approach commonly known as RandNLA. We will try to show that using a sparse random matrix gives additional computational savings and show that projecting the data onto a random lower-dimensional subspace yields results comparable to conventional dimensionality reduction methods such as SVD and QR: the similarity of data vectors is preserved well under random projection. This approach is computationally significantly less expensive than the conventional approach. We show that experimentally with Random projection.

Big Data applications suffer from performance issues [1]. We will describe a faster way of performing Singular Value Decomposition, by approximating the matrix of interest with a low-rank one, which is computationally easier to deal with. This will be done by using algorithms to sample a set of relevant columns from the total number of columns of the matrix of interest and then constructing the approximation from these sampled columns. We choose some random matrix independent of the data set, multiply with the data set, and get a randomized compression. From this compression, we can still extract enough information to do some sort of reasonable data analysis.

1.2 Low-rank approximation and its applications

Let A be an $m \times n$ matrix. Let $k << \min(m, n)$. If A satisfies any of the following conditions,

1. The columns of A span a subspace of $\mathbb{R}^m$ of dimension k.
2. The rows of A span a subspace of $\mathbb{R}^n$ of dimension k.

   then low-rank approximation of A means

$$\underset{m\times n}{A} \approx \underset{m\times k}{B} \quad \underset{k\times k}{C} \quad \underset{k\times n}{D} \tag{1.1}$$

$$\underset{m\times n}{A} \approx \underset{m\times k}{B} \quad \underset{k\times n}{D'} \tag{1.2}$$

The storage of an $m \times n$ matrix A needs mn locations; however, the reduced matrix A_k, only requires $mk + nk$ locations for storage, which can cause significant reduction when k is small. There are a number of areas where low-rank approximation finds applications. Some of them are listed below.

Applications of Low-rank approximation

Image Compression

An image can be illustrated by an $m \times n$ matrix A whose (i, j)th entry is related to the brightness of the pixel (i, j). The purpose behind Image Compression is to compress the image illustrated by a large matrix to a new matrix which corresponds to a lower-order rank approximation. The quality of the image is tolerable here [2].

Image Restoration

The goal of Image Restoration is to return the original image from a blurry image corrupted by *noises*. These *noises* represent the smaller or unimportant singular values, and their removal can result in the rank-k approximation of A which leads to noise-free images. The need for Image Restoration is seen in practical applications like clinical diagnosis [2].

Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an approach for analyzing a collection of documents that are supposedly connected. Latent refers to "hidden" and so this method tries to extract the latent features from that data set which cannot be directly measured. Suppose we have a raw text data. The Latent Semantic Indexing begins with generating an $m \times n$ document-term matrix, where m is the document each containing n terms, and A_{ij} is the frequency of the jth term in the ith document. The notion behind this matrix is that text documents can be represented as points in the Euclidean space. The second step is to do Singular Value Decomposition to find the set of best k terms that describe the documents relating to the k singular vectors of A and remove the rest of the noise in terms of the extra word usage[3].

1.3 Review of relevant background

Vector norms

Let $x \in \mathbb{R}^n$. Suppose x_i, where $i = 1, 2, ..., n$, is the ith element of x. Then the default norm for vectors is the Euclidean norm, namely the l_2 norm is given by

$$||x|| = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2} \tag{1.3}$$

And in general the l_p norm is given by

$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{1.4}$$

Matrix norms

Let A be an $m \times n$ matrix, where $A(i,j)$ is the entry in the ith row and the jth column, respectively. Then the $Frobenius$ norm of A is defined as

$$||A||_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |A(i,j)|^2 \right)^{1/2} \tag{1.5}$$

and the *spectral* norm is given by

$$||A||_2 = \underset{x \neq 0}{max} \frac{||Ax||_2}{||x||_2} \tag{1.6}$$

Matrix and vector norms provide us with a sense of measure of the matrix and the vector respectively. We want to be as close as possible to this optimal value when we use our algorithms [1].

Singular Value Decomposition

Suppose we have an $m \times n$ matrix A and $r = \text{rank}(A)$. The Singular Value Decomposition (SVD) is a factorization of a matrix into two orthogonal matrices and a diagonal matrix. The Singular Value Decomposition of A is

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T \tag{1.7}$$

where $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{r\times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{r\times r}$ is a diagonal matrix with $r = \text{rank}(A)$. The r diagonal entries of the matrix Σ are denoted by σ_i, for $i = 1, 2, ..., r$, where $r = \min(m, n)$ and σ_i are called the singular values of A. The singular values satisfy the property $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0$ [4].

The best rank-k approximation of A $(U\Sigma_k V^T)$ can be denoted as

$$A_k = \sigma_1 u_1 v_1^T + \cdots + \sigma_k u_k v_k^T \tag{1.8}$$

QR Decomposition

QR decomposition gives out an orthogonal matrix and an upper triangular matrix. Let A be an $m \times n$ matrix with $m \geq n$. The QR decomposition of A is

$$A = Q_A R_A, \quad Q_A \in \mathbb{R}^{m\times n}, R_A \in \mathbb{R}^{n\times n} \tag{1.9}$$

The columns of Q_A are orthonormal, i.e., $Q_A^T Q_A = I$, and the matrix R_A is upper triangular, i.e., for all $i < j, R_{ij} = 0$ [4].

Big-Oh Notation

The Big-Oh notation tells us about the efficiency of an algorithm. It gives us the complexity in terms of time and space. When we talk about Big-Oh notation, we are usually talking about the worst-case scenario. A trend in the speed of algorithms is shown here.

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(2^n) < O(n!)$$

In simple words, Big-Oh notation is used to classify algorithms based on the number of operations, conditions or comparisons[5].

Time and memory complexities

Time complexities: For an $m \times n$ matrix A and an $n \times k$ matrix B, the operation AB leads $O(mnk)$ float operations in general. For QR decomposition and SVD with an $m \times n$ matrix, it will be $O(mn^2)$.

Memory complexities: For an $m \times n$ matrix A, the storage required is O(mn) locations.

CHAPTER 2: METHODOLOGY

2.1 Idea behind Matrix Sketching

Matrix Sketching is a straightforward and systematic method to reduce the columns of a matrix by accepting a small amount of error compared to the original matrix. We will later introduce two methods of matrix sketching: (1). Random projections and (2). Random Sampling.

We choose some random matrix independent of the data set, multiply with the original matrix, and get a randomized compression. From this compression we can still extract enough information to do some sort of reasonable data analysis. Suppose $A \in \mathbb{R}^{m\times n}$ is a matrix.

$$\underset{m\times s}{C} = \underset{m\times n}{A} \quad \underset{n\times s}{S} \tag{2.1}$$

where $S \in \mathbb{R}^{n\times s}$ is a sketching matrix.

Theoretical properties

The sketching matrix is beneficial if it possesses any of the two properties mentioned below:

Subspace Embedding

For a fixed $m \times n$ $(m << n)$ matrix A and any m-dimensional vector y, the inequality given below holds with high probability

$$\frac{1}{\gamma} \leq \frac{||y^T AS||_2^2}{||y^T A||_2^2} \leq \gamma \tag{2.2}$$

where $S \in \mathbb{R}^{n\times s}$ $(s << n)$ is a certain sketching matrix. Intuitively, for all n dimensional vectors x in the row space of A (a rank m subspace within $\mathbb{R}^n$), the length of vector x does not change much after sketching: $||x||_2^2 \approx ||xS||_2^2$ [4].

Low-rank approximation

Let A be any $m \times n$ matrix and k be any positive integer far smaller than m and n. Let $C = AS \in \mathbb{R}^{m\times s}$, where $S \in \mathbb{R}^{n\times s}$ is a certain sketching matrix and $s \geq k$. The Frobenius norm error bound given below holds with high probability for some $\eta \geq 1$

$$||A - CC^{\dagger}A||_F^2 \leq \eta ||A - A_k||_F^2 \tag{2.3}$$

Intuitively, the low-rank approximation property represents that the columns of A_k are almost in the column space of $C = AS$. This property helps us in solving the k-SVD more effectively for $(k \leq s)$ which will be shown later in this thesis [4].

2.1.1 Random projections

Random projections have recently emerged as a powerful method for dimensionality reduction. It is a simple and computationally efficient way to reduce the dimensionality of data by giving a small amount of error for *faster processing times* and a *manageable size* of data.

We will present some experimental results on using Random projections as a dimensionality reduction tool in a number of cases, where the high dimensionality of the data would otherwise lead to burdensome computations. We show that projecting the data onto a random lower-dimensional subspace yields results comparable to conventional dimensionality reduction methods such as SVD. The similarity of data vectors is preserved well under random projections.

The idea behind Random projections comes from the Jhonson-Lindenstrauss lemma. The lemma states that any n points in high dimensional Euclidian space can be *randomly* projected onto ($k = O(\frac{\log n}{\epsilon^2})$)-dimensional Euclidean space, where the pairwise distance between any two points changes only by a factor not more than $(1 \pm \epsilon)$, where $\epsilon \in (0, 1)$. This is a very useful dimensionality reduction tool [6].

The lemma: For any $0 < \epsilon < 1$ and any integer n, let k be a positive integer such that

$$k \geq 4(\epsilon^2/2 - \epsilon^3/3)^{-1} \ln n \tag{2.4}$$

then for any set V of n points in $\mathbb{R}^d$, there is a map $f : \mathbb{R}^d \to \mathbb{R}^d$ such that for all $u, v \in V$,

$$(1-\epsilon)|| \ u - v \ ||^2 \leq || \ f(u) - f(v) \ ||^2 \leq (1+\epsilon)|| \ u - v \ ||^2. \tag{2.5}$$

The reader interested in understanding the proof is advised to refer to [6].

We will introduce the basic MATLAB code to perform Random projection on a matrix and obtain a high-quality sketch of A. This method is called Gaussian projection.

MATLAB code:

```
function [C] = GaussianProjection(A, s)
n = size(A, 2);
S = randn(n, s) / sqrt(s);
C = A * S;
```

The time cost for this code is $O(mns)$. It is easy to implement and gives us a high-quality sketch of A. It is interesting to note that the time complexity can be very high when m and n are comparatively larger. And the C matrix will result in a dense matrix even if A is sparse [4].

Now, we will put up some basic algorithms to further work from the Random projections to obtain a low-rank approximation of the matrix A. Algorithm 1 is proposed by [7] and then we propose two similar but much simpler algorithms.

Algorithm 1:

```
Step - 1: Input: an $m \times n$ matrix $A$ and the target rank $k$.
Step - 2: Draw an $n \times s$ sketching matrix $S$
Step - 3: $C = A * S$;
Step - 4: QR Decomposition: $[Q_C, R_C] = qr(C)$, where $Q_C \in \mathbb{R}^{m\times s}$, $C \in \mathbb{R}^{m\times s}$
Step - 5: $k-$SVD: $[\underset{s\times k}{\bar{U}} \quad \underset{k\times k}{\tilde{\Sigma}} \quad \underset{n\times k}{\tilde{V}}] = svds(\underset{s\times n}{Q_C^T A}, k)$;
Step - 6: $\tilde{U} = Q_C\bar{U} \in \mathbb{R}^{m\times k}$;
Step - 7: return $\tilde{U}\tilde{\Sigma}\tilde{V} \approx A_k$.
```

Algorithm 2:

```
Step - 1: Input: an $m \times n$ matrix $A$ and the target rank $k$.
Step - 2: SVD: $[U\Sigma V^T] = svds(A, k)$;
Step - 3: Draw an $n \times s$ sketching matrix $S$
Step - 4: $C = A * S$;
Step - 5: QR Decomposition: $[Q_C, R_C] = qr(C)$, where $Q_C \in \mathbb{R}^{m\times s}$, $C \in \mathbb{R}^{m\times s}$
Step - 6: $U = Q_C \in \mathbb{R}^{m\times k}$;
Step - 7: return $UU^T A \approx A_k$
```

Algorithm 3:

```
Step - 1: Input: an $m \times n$ matrix $A$ and the target rank $k$.
Step - 2: SVD: $[U\Sigma V^T] = svd(A)$;
Step - 3: Draw an $n \times s$ sketching matrix $S$
Step - 4: $C = A * S$;
Step - 5: QR Decomposition: $[Q_C, R_C] = qr(C)$, where $Q_C \in \mathbb{R}^{m\times s}$, $C \in \mathbb{R}^{m\times s}$
Step - 6: $k-$SVD: $[\underset{m\times k}{U_C} \; \underset{k\times k}{\Sigma} \; \underset{s\times k}{V_C}] = svds(\underset{m\times s}{C}, k)$;
Step - 7: return $U_C U_C^T A \approx A_k$.
```

Notice that the accuracy of the randomized $k-$SVD depends only on the quality of the sketch matrix C. Suppose Q_C is the orthonormal basis of C and the column space of C. The formal derivation of the algorithm can be found in [4].

2.1.2 Random Sampling

Unlike Random projection, Random Sampling selection does not require to look for each and every record of the matrix A, and it preserves the important properties of A. Majorly Randomized Numerical Linear Algebra research pays attention to sketching a matrix by keeping only a few of its rows and/or columns [8]. We are going to talk about a *Random Sampling* technique known as *Leverage Score Sampling.*

This method reveals influential rows and columns of a matrix. By influential we mean, for the rows and columns being picked they give better approximations while working with singular vectors and singular values.

Leverage scores are basically statistic and in the algorithms that we will work, we use them as a sampling probability, so we will only keep scores with high probability, and we will check whether it gives us interesting results [4].

Before talking about the sampling method, we would like to discuss what leverage scores are. Suppose we have an $m \times n$ matrix A, with $r = \text{rank}(A) < n$, and $V \in \mathbb{R}^{n\times r}$ contains the right singular vectors, and let v_i denote the ith row of the matrix V as a row vector. The column leverage scores of A can be defined as a row vector.

$$l_i = ||v_i||_2^2, \quad for \quad i = 1, 2, \cdots, n \tag{2.6}$$

Similarly, the row leverage scores of A will be

$$p_j = ||u_j||_2^2, \quad for \quad j = 1, 2 \cdots, m \tag{2.7}$$

for the left singular vectors $U \in \mathbb{R}^{m\times r}$, u_j denote the jth row of the matrix U as a row vector.

Leverage score sampling is a way to select rows/columns of A with probability proportional to its leverage scores. MATLAB code performing Leverage Score Sampling according to column leverage scores can be seen below. It gives a new matrix C which has significantly reduced columns, therefore occupying less space compared to the matrix A yet maintaining important properties of the original matrix.

MATLAB code[4]:

```
function [C, idx] = LeverageScoreSampling(A, s)
n = size(A, 2);
[ , ,V] = svds(A, k)
leverage scores = sum(V^2, 2);
prob = leveragescores/sum(leveragescores);
idx = randsample(n, s, true, prob);
idx = unique(idx); C = A(:, idx) ;
```

Algorithm: This algorithm is inspired by combining the works of [4] and [3]. Given a matrix $A \in \mathbb{R}^{m \times n}$, our primary goal is to estimate the top k singular values and the complementary singular vectors in a constant number of passes through the data.

Input: $A \in \mathbb{R}^{m \times n}$, $s, k \in \mathbb{Z}^{+}$, such that, $1 \leq s \leq n$, $\{p_i\}_{i=1}^{n}$, such that $p_i \geq 0$ and $\sum_i p_i = 1$

Output: $H_k \in \mathbb{R}^{m \times k}$ and $\sigma_t(C), t = 1, \cdots, k$

1. $[\sim, \sim, V] = svds(A, k)$
2. Compute leverage scores (column): $l_i = ||v_i||_2^2$;
3. Generate probability distribution of the leverage scores: $p_i = \frac{l_i}{\sum_i l_i}$; $\sum_i p_i = 1$
4. Randomly sample the columns according to the probability distribution p_i: $s = randsample(n, s, true, p_i)$
5. Eliminate duplicate columns: $s = unique(s)$
6. $C = A(:, s)$; such that $C \in \mathbb{R}^{m \times s}$

7. Compute C^TC and its SVD; say $C^TC = \sum_{t=1}^{s}(C)y^t{y^t}^T$.

8. Compute $h^t = Cy^t/\sigma_t(C)$; for $t = 1, \cdots, k$.

9. Return H_k, where $H_k^{(t)} = h^t$, and $\sigma_t(C), t = 1, \cdots, k$.

10. Return $H_k^{(t)}H_k^{(t)T}A \approx A_k$.

The additional time and additional space will be $O(s^2m + s^3)$ and $O(ms + s^2)$ respectively. The plan here is to pick s columns of the matrix A by Leverage Score Sampling, and generate a new matrix C. Later on, we will compute the matrix C^TC and its SVD to find the right singular vectors of C, and from that point, we compute the singular values and left singular vectors of C which will be approximations to singular values and left singular vectors to the matrix A.

We then compute the H_k matrix in order to get the rank-k approximation of the original matrix. But Leverage score sampling can be as expensive as SVD, so it is a drawback of this method [4]. This algorithm can be easily implemented in the MATLAB software.

CHAPTER 3: APPLICATIONS AND RESULTS

In this section, we will talk about the kernel functions and give specific reasons why we can use them to address low-rank approximation and show how we generate our data set. In the end we will show some interesting results and figures after having applied the algorithms.

3.1 Kernel functions

Kernel functions show a way to operate the data on the original space while projecting it into a higher dimensional space [9]. It allows us to retrieve important information without computing the coordinates of the data in the original space but by simply computing the inner products between the images of all pairs of data in the feature space. Kernel functions give us a chance to operate computationally effective than the original function for the computation of the coordinates. Examples of some common kernel functions are:

- Polynomials of degree d
$$K(x, y) = (x \cdot y)^d \tag{3.1}$$

- Polynomials of degree d
$$K(x, y) = (x \cdot y + 1)^d \tag{3.2}$$

- Gaussian kernels
$$K(x, y) = \exp(-\frac{||x-y||^2}{2\sigma^2}) \tag{3.3}$$

An example of kernel function is Green's function of the Laplace operator. We will discuss upon that in detail shortly.

3.2 Why kernel functions are compressible?

We will use the Taylor series approximation to understand how we can apply low-rank approximation to kernel matrices. A Taylor series approximation uses a Taylor series to represent an approximation for a function. If we know the function value at some point x_o, $(f(x_o))$ and the value of the derivative at the same point, $(f_x(x_o))$, we can use these to approximate $f(x)$ for other points x. It is important to note here that this approximation is acceptable when x is relatively near x_o. This 1st-degree Taylor Polynomial is also called the linear approximation of $f(x)$ for x near x_o and is given by [10]:

$$f(x) \approx f(x_o) + f_x(x_o)(x - x_o) \tag{3.4}$$

In order to get a better approximation of $f(x)$ for x near x_o we can further approximate our function to a 2nd-degree Taylor polynomial of $f(x)$ at $x = x_o$, which is known as the quadratic approximation:

$$f(x) \approx f(x_o) + f_x(x_o)(x - x_o) + \frac{f_{xx}(x_o)}{2}(x - x_o)^2 \tag{3.5}$$

As one might have noticed, the approximation of our function gets closer to the original function as we go for higher degree polynomials.

Now, suppose we have a two-variable function $f(x, x')$, we can use the Taylor series approximation in two variables to approximate our function near the point (x_o, x'_o).

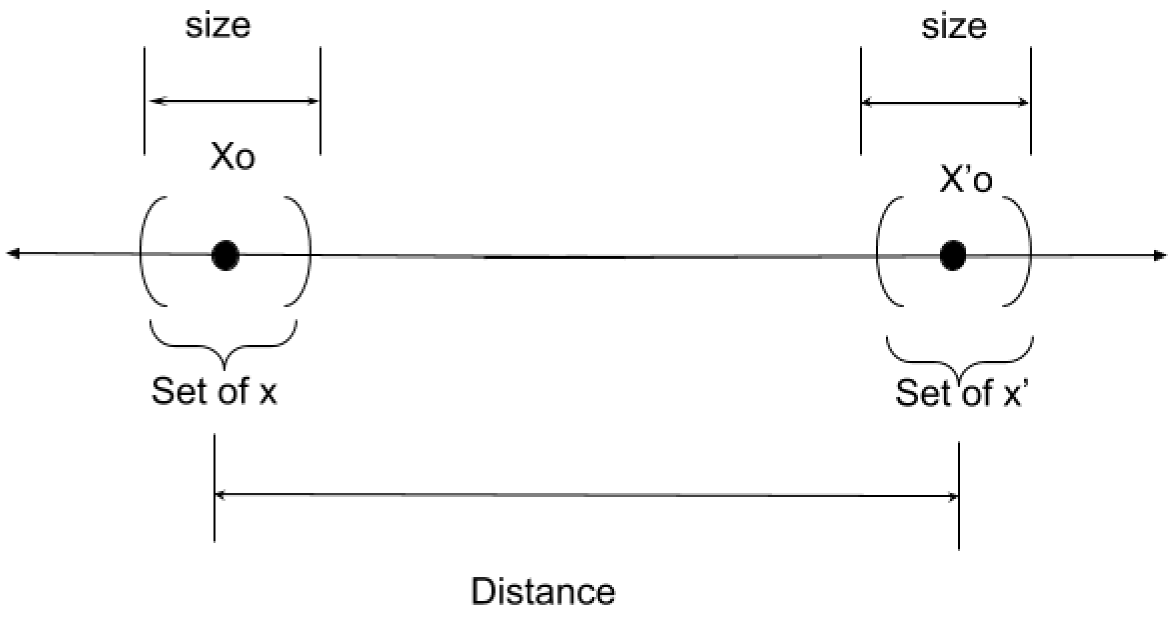

Figure 3.1: To show Size/distance ratio

Let us say we want to show Taylor polynomial in 3rd-degree for our two-variable function approximation. It is represented similarly to the equation above,

$$\begin{aligned} f(x,x') \approx\ & f(x_o,x'_o) + f_x(x_o,x'_o)(x-x_o) + f_{x'}(x_o,x'_o)(x'-x'_o) \\ & + \frac{f_{xx}(x_o,x'_o)}{2}(x-x_o)^2 \\ & + \frac{f_{x'x'}(x_o,x'_o)}{2}(x'-x'_o)^2 + f_{xx'}(x_o,x'_o)(x-x_o)(x'-x'_o) \\ & + \frac{1}{6}f_{xxx}(x_o,x'_o)(x-x_o)^3 + 3f_{xxx'}(x_o,x'_o)(x-x_o)^2(x'-x'_o) \\ & + 3f_{xxx'}(x_o,x'_o)(x-x_o)(x'-x'_o)^2 + \frac{1}{6}f_{x'x'x'}(x_o,x'_o)(x'-x'_o)^3 \end{aligned}$$

where $f_x(x_o,x'_o)$ is the partial derivative of $f(x,x')$ w.r.t x and $f_{x'}(x_o,x'_o)$ is the partial derivative of $f(x,x')$ w.r.t x' and so on.

If we write the above equation in a matrix form (while ignoring the constants);

$$f(x,x') \approx \begin{bmatrix} 1 \\ x-x_o \\ (x-x_o)^2 \end{bmatrix}^T_{3\times 1} \begin{bmatrix} f(x_o,x'_o) & f_{x'}(x_o,x'_o) & f_{x'x'}(x_o,x'_o) \\ f_x(x,x'_o) & f_{xx'}(x_o,x'_o) & f_{xx'x'}(x_o,x'_o) \\ f_{xx}(x_o,x'_o) & f_{xxx'}(x_o,x'_o) & f_{xxx'x'}(x_o,x'_o) \end{bmatrix}_{3\times 3} \begin{bmatrix} 1 \\ x'-x'_o \\ (x'-x'_o)^2 \end{bmatrix}_{3\times 1}$$

Now, we want to generalize this for k terms;

$$f(x,x') \approx \begin{bmatrix} 1 \\ x-x_o \\ (x-x_o)^2 \\ \cdots \\ (x-x_o)^{k-1} \end{bmatrix}^T \begin{bmatrix} f(x_o,x'_o) & \frac{\partial f(x_o,x'_o)}{\partial x'} & \cdots & \frac{\partial^{k-1} f(x_o,x'_o)}{\partial x'^{k-1}} \\ \frac{\partial f(x_o,x'_o)}{\partial x} & \frac{\partial^2 f(x_o,x'_o)}{\partial x \partial x'} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^{k-1} f(x_o,x'_o)}{\partial x^{k-1}} & \cdots & \cdots & \frac{\partial^{2k-2} f(x_o,x'_o)}{\partial x^{k-1}\partial x'^{k-1}} \end{bmatrix} \begin{bmatrix} 1 \\ x'-x'_o \\ (x'-x'_o)^2 \\ \vdots \\ (x'-x'_o)^{k-1} \end{bmatrix}$$

There is a pattern here in the size of the matrices if we pay close attention, this equation is of the form of the low rank matrix approximation as shown in the introduction section.

$$\underset{m\times n}{A} = \underset{m\times k}{B} \quad \underset{k\times k}{C} \quad \underset{k\times n}{D} \tag{3.6}$$

$$\underset{m\times n}{A} = \underset{m\times k}{B} \quad \underset{k\times n}{D'} \tag{3.7}$$

Whenever we talk about making an approximation, it is better to have some idea of the size of the error we introduce. We need to know how adequate our approximation is, which boils down to knowing how much our error is. Error is defined as the difference between the original function and the approximating function.
Suppose $F_n(x)$ is the nth degree Taylor polynomial for a function $f(x)$ centered at the point x_o, the error is,

$$R = f(x) - F_n(x) \tag{3.8}$$

It is interesting to note here that it is difficult to know the error if we do not know the value of the actual function, so we can only estimate the error. According to [10], the error term is

$$|R_n| = \left| \frac{f^{(n+1)}(c)}{(n+1)!}(x-a)^{n+1} \right| \tag{3.9}$$

which gives us a bound on the size of the error. Here we need to get the largest of $f^{(n+1)}(c)$, for all c, where c is in between a and x. This looks similar to the $(n+1)$th term of the Taylor series expansion.

On this basis, we can derive that for our case, the error term can be found as the $(n+1)$th term in the equation above.

$$|R_n| = \left| (x-x_o)^n \frac{\partial^{2n} f(c,d)}{\partial x^n \partial x'^n} (x' - x'_o)^n \right| \tag{3.10}$$

Again it is important to note that the partial derivatives are not at the exact (x_o, x'_o) but at some point (c,d), for all c, where c is in between x_o and x; and for all d, where d is in between x'_o and x'.

According to the works by Dr. Duan Chen in [11] we will try to derive a bound to minimize the error term. Suppose our function is $\dfrac{1}{|x-x'|}$,

$$\left| \frac{1}{(x-x_o)^n} \frac{\partial^{2n} f(c,d)}{\partial x^n \partial x'^n} \frac{1}{(x'-x'_o)^n} \right| \leq \left| \frac{1}{r-r'} \right|^{2n} \tag{3.11}$$

$$\left| \frac{\partial^{2n} f(c,d)}{\partial x^n \partial x'^n} \right| \leq \left| \frac{(x-x_o)^n (x'-x'_o)^n}{(r-r')^{2n}} \right| \tag{3.12}$$

This is nothing but the size/distance ratio which we will use extensively as a parameter in finding the results. When the ratio is smaller than 1 and n is bigger than 1, the error will be small enough, which allows us to do low-rank approximation on our kernel matrix.

3.3 Applications on Green's function

According to Electrostatics, physically, *Green's function* which is a solution to the singular Poisson's equation

$$\nabla^2 G = \delta(x - x')$$

is defined as the potential at the position vector x due to a point charge placed at the position x', where G is the *Green's function*,

$$G(x, x') = \frac{1}{|x - x'|}$$

and $\delta(x - x')$ is the *Dirac Delta function.* A *Green's function* gives the response at point x, due to a point charge that is located at x'. It is important to note here that Green's function does not change if we interchange the coordinates, which means *Green's function* are reciprocal $G(x, x') = G(x', x)$.

So for our case, where we are using 400 different point charges spaced out on the two boxes, we proceed and calculate *Green's function* using the above equation.

The function

$$\kappa(r, r') = \frac{1}{|r - r'|} \tag{3.13}$$

depicts the interaction of the charged particles which we are using to create a distance matrix which is of the size (400 $\times$ 400).

Here is the pictorial representation of our premise. We have point charges on both the boxes which are denoted by '$*$' in the figure above. A point charge is an idealized model of a particle that has an electric charge and is a mathematical point with no dimensions.

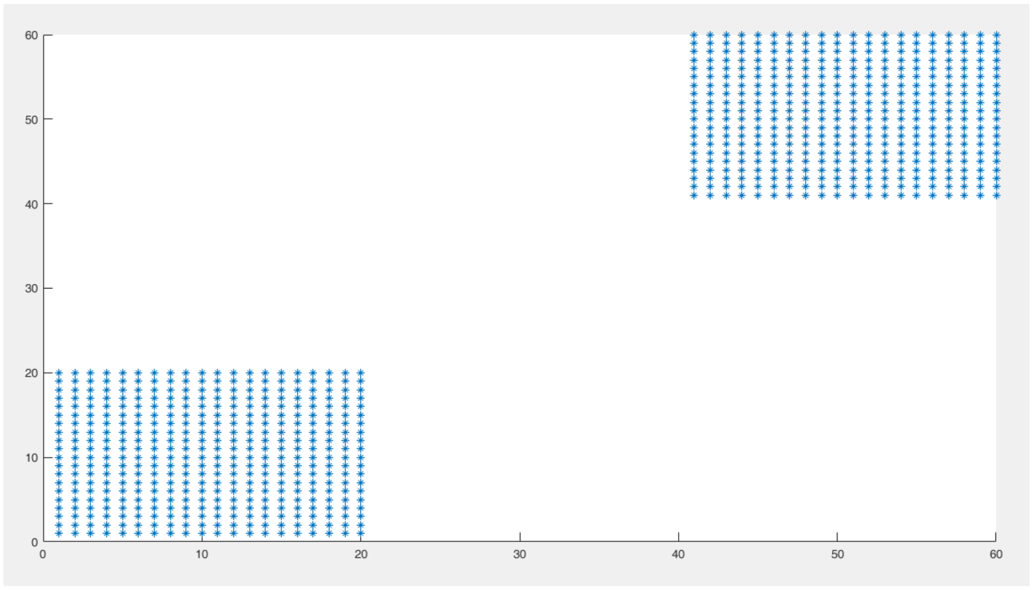

Figure 3.2: Two boxes each having 400 charges on them

3.4 Results

We begin by using two boxes that contain a specific number of electric charges on specific points. We create a matrix using Green's function by the interaction of the charged particles that act as our original dataset matrix (which we call the matrix A throughout this thesis). Here, we fix the number of charges on the two boxes, the size of the two boxes, and only manipulate the distances between them. In the second figure, we have brought the two boxes closer by decreasing the distance between them, which increases the size/distance ratio. To better apply and understand the nature of our algorithms, we will work on this matrix to obtain matrix approximations.

After introducing the idea of Matrix Sketching and Column Selection in the previous section, we find the SVD of $A \in \mathbb{R}^{400\times400}$, and check out the initial trend of the first '20' singular values (given the large size of the matrix, we will ignore the noise).

Given our primary goal for this project, to somehow show that when we apply Matrix Sketching on a certain matrix, we try and preserve the important properties of our original data set, which we will show by extracting relative errors using the Frobenius norms.

The rank of our original dataset matrix is 33, which is found by a simple MATLAB command $rank(A)$. It is important to note here that this rank is according to a certain tolerance, which is again easy to get from a basic MATLAB command $(max(size(A)) * eps(norm(A)))$. And in our case, that tolerance is 3.5527×10^{-13}. The readers can feel free to change the tolerance according to their preference. In the new-age computers, according to the common machine-precision, one can go up to 10^{-16}.

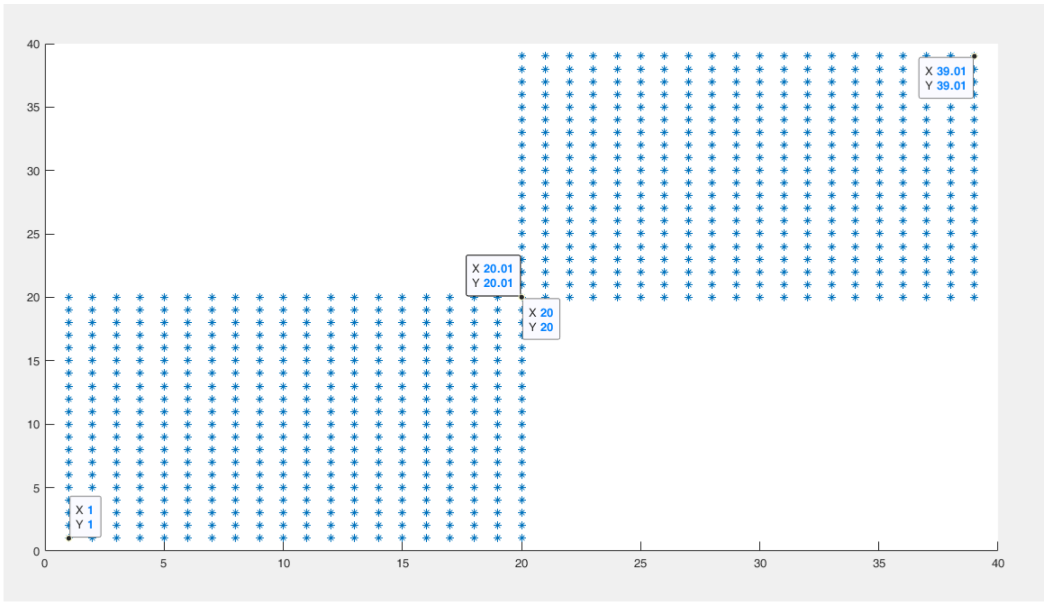

Figure 3.3: Two boxes each having 400 charges on them (close)

Results with Random projections

Having introduced the algorithms in the Methodology section, we put some interesting results here in the form of a table. In this table, we are showing some relative errors between the original matrix and the new rank-k approximating matrix in terms of the Frobenius norms $(||A - A_k||_F/||A||_F)$. We perform the algorithms by generating different data sets with varying the distance between the centers of the two boxes as well as governing the size of the matrix. The results obtained are worked by getting various rank-k approximations over the given size/distance ratios.

Table 3.1: Relative error with Random projections

Target rank (k) of the matrix	Size/distance ratio	Relative error with Random projections
33	0.6438	1.1768×10^{-15}
15	0.6438	1.01496×10^{-15}
5	0.6438	3.684×10^{-16}
33	0.3358	8.07027×10^{-16}
15	0.3358	9.0703×10^{-15}
5	0.3358	1.07027×10^{-10}

When the target rank k is above 15, we find that the approximation error between the two matrices to be of the order 10^{-15}. To back the results shown in the table, we present some figures generated from MATLAB after executing the algorithms. These figures depict the trend in the singular values of the normal SVD compared to the Randomized SVD implemented on the new matrix obtained from after sketching.

The result and the figure shown above are when our boxes are placed far from each other as shown in figure (1). We will further show results changing the size/distance ratio of our boxes containing 400 point charges. When we try to overlap the boxes, it is clear that the original dataset matrix will not generate any results since $|r - r'|$ will be zero for certain entries, which result in $\frac{1}{|r-r'|}$ being undefined, so we will try and maintain a certain distance between our two boxes.

The following three figures are for the size/distance ratio = 0.6438 (as shown in figure 3.1), with the respective rank-k approximations,

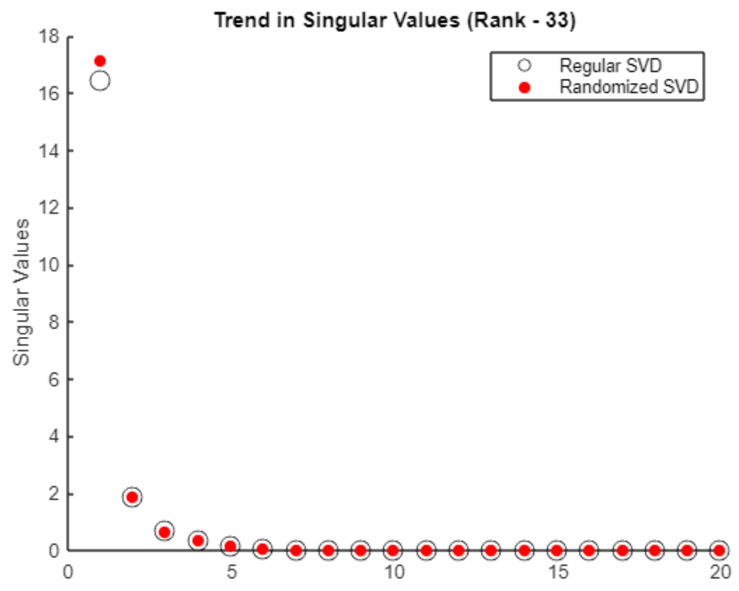

Figure 3.4: Trend in singular values for rank 33

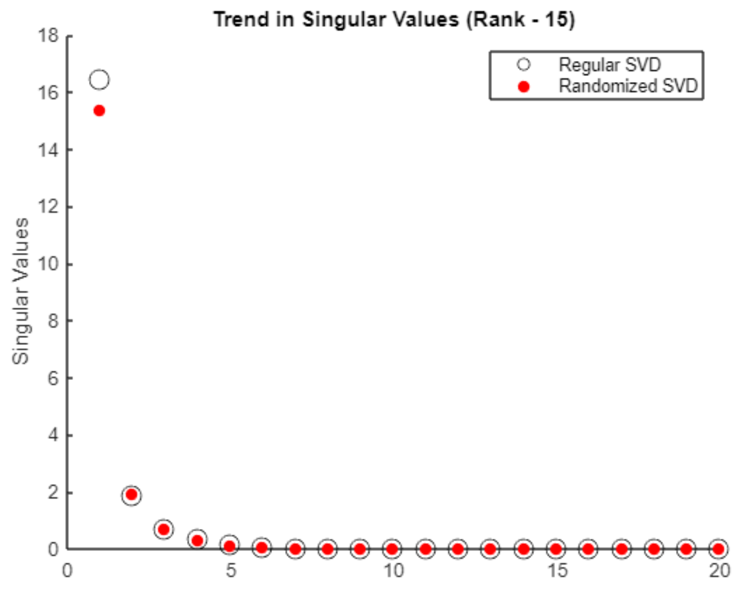

Figure 3.5: Trend in singular values for rank 15

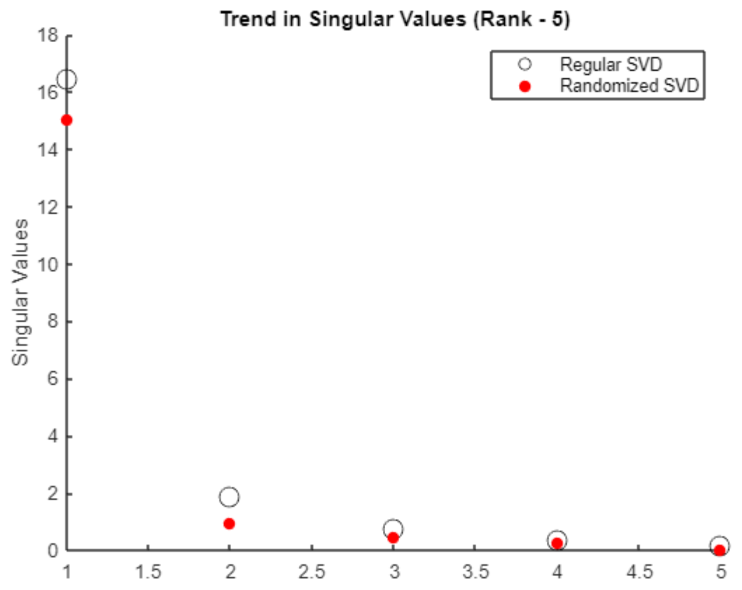

Figure 3.6: Trend in singular values for rank 5

The following three figures after Figure 3.6 are for the size/distance ratio = 0.3358, with the respective rank-k approximations,

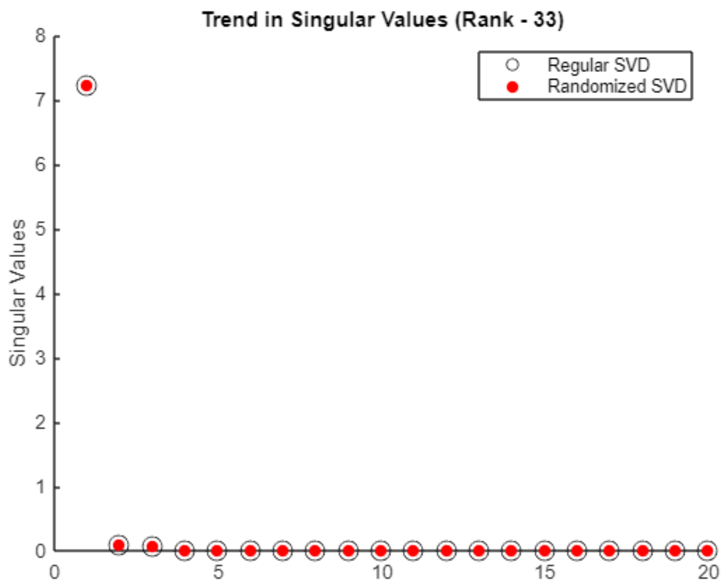

Figure 3.7: Trend in singular values for rank 33

Results with Random Sampling

Table 3.2: Relative error with Random Sampling

Rank (k) of the matrix	Size/distance ratio	Relative error with Random Sampling
33	0.6438	9.11×10^{-4}
15	0.6438	3.04×10^{-2}
5	0.6438	1.646×10^{-1}
33	0.3358	1.785×10^{-5}
15	0.3358	1.26×10^{-2}
5	0.3358	2.7×10^{-2}

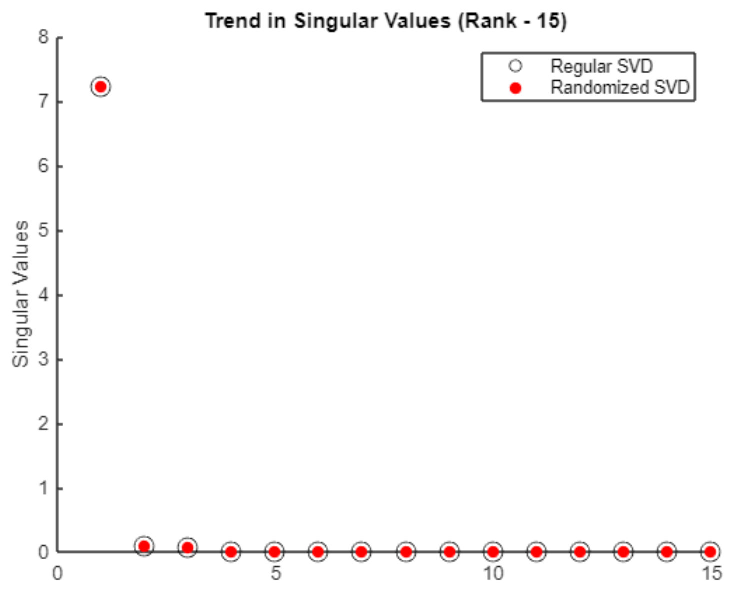

Figure 3.8: Trend in singular values for rank 15

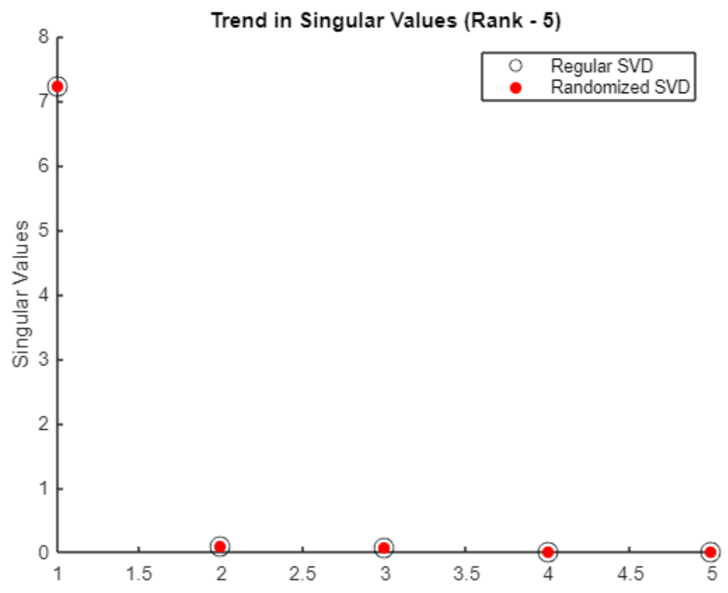

Figure 3.9: Trend in singular values for rank 5

CHAPTER 4: DISCUSSION and CONCLUSION

We began with using kernel functions and we showed why we can apply low-rank approximation on kernel matrices. After having a clear path and reason to work on, we created a general dataset matrix to show some experimental results by working on the interaction between point charges on two boxes. After successfully creating the matrix, we performed certain algorithms based on Random projections and Random Sampling.

As one can notice in the figures generated through the Random projection algorithms, it is apparent that we do not find many differences between the singular values in the Regular SVD and the Randomized SVD. It is also evident that the first singular value starts at nearly the same point for both of our decompositions. We can certainly generate more results just by manipulating our rank, size, and distance parameters, creating new ratios, and eventually finding new relative errors.

It is interesting to note that the results obtained in Random Sampling are comparatively worse than Random projections which is because random sampling only uses partial information from the matrix, while random projection uses the whole matrix. Although the sampling method results in lower accuracy compared to Random projections, it reduces the entry visit of the matrix.

There is potential to increase accuracy in the Random sampling method by reducing the size/distance ratio by creating a hierarchical structure. We selected two different boxes and generated a dataset matrix by spacing them out. To add to that, we can work upon a single box and generate data from the same group of points, in a way creating a hierarchical structure between the same set of points. To explain in terms of our example, we can work on one of the boxes as our primary and we can produce new data just by making the point charges interact with themselves and use low-rank approximation.

For the SVD for a regular $n \times n$ matrix, we know that the time complexity is $O(n^3)$. By implementing Random projections and Random Sampling, we successfully bring down the time complexity to $O(n^2)$. This can result in a significant drop in time and help run algorithms in a faster way for large scale data sets, where information processing is a problem. For the scope of future work, we can further increase the speed of our algorithms and bring this down to $O(n)$ by going a step ahead in Random Sampling and executing the ConstantTime SVD Algorithm. This algorithm helps in sampling the rows of the matrix, after the column sampling step.

Finally, we can conclude that, though having certain errors and scope for future work, the new approach Randomized Numerical Linear Algebra does have good results and it can be used to achieve low-rank approximation in a faster manner than the conventional approach.

REFERENCES

[1] V. S. Burca, “Fast monte carlo algorithms for computing a low-rank approximation to a matrix,” 2014.

[2] B. N. Datta, *Numerical linear algebra and applications*, vol. 116. Siam, 2010.

[3] P. Drineas, R. Kannan, and M. W. Mahoney, “Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix,” *SIAM Journal on computing*, vol. 36, no. 1, pp. 158–183, 2006.

[4] S. Wang, “A practical guide to randomized matrix computations with matlab implementations,” *arXiv preprint arXiv:1505.07570*, 2015.

[5] E. Bingham and H. Mannila, “Random projection in dimensionality reduction: applications to image and text data,” in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 245–250, 2001.

[6] S. Dasgupta and A. Gupta, “An elementary proof of the johnson-lindenstrauss lemma,” *International Computer Science Institute, Technical Report*, vol. 22, no. 1, pp. 1–5, 1999.

[7] N. Halko, P.-G. Martinsson, and J. A. Tropp, “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions,” *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.

[8] P. Drineas and M. W. Mahoney, “Randnla: randomized numerical linear algebra,” *Communications of the ACM*, vol. 59, no. 6, pp. 80–90, 2016.

[9] T. M. Mitchell and A. W. Moore, “Machine learning, 10-701 and 15-781, school of computer science,” 2002.

[10] R. Haas, “Approximating functions by taylor polynomials,” 2011.

[11] D. Chen and W. Cai, “An o (nlogn) hierarchical random compression method for kernel matrices by sampling partial matrix entries,” *Journal of Computational Physics*, vol. 397, p. 108828, 2019.