RANDOMIZED TECHNIQUES FOR MATRIX DECOMPOSITION AND ESTIMATING THE APPROXIMATE RANK OF A MATRIX

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

SHASHANKA UBARU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

YOUSEF SAAD AND ARYA MAZUMDAR

NOVEMBER, 2014

Acknowledgements

First and foremost, I wish to thank my advisors, Professor Yousef Saad and Professor Arya Mazumdar, for guiding me throughout the work of this thesis. Without their inputs and guidance, this thesis work would never have been possible.

I am extremely grateful to Professor Yousef Saad, for funding me throughout the thesis and I feel extremely fortunate and honoured to be able work and continue working under him even for my PhD. His energy, dedication, work ethics and immense knowledge is a great inspiration for me. I am also grateful to Professor Arya Mazumdar, for his suggestions, guidance and encouragement during this thesis work. I would also like to thank Professor Jarvis Haupt and Professor Daniel Boley who are part of my thesis defense committee for their support, guidance and feedback.

My special thanks to my fellow labmates Thanh T Ngo, Vasileios Kalantzis and Ruipeng Li who have helped me a lot in understanding the subject better, with numerous discussions and helpful suggestions.

Finally, my deepest gratitude goes to my family: my parents and my brother for their unconditional love and constant support.

Dedication

To those who held me up over the years

Abstract

Finding low dimensional matrix approximations of the data is an essential task in data analysis and scientific computing. Recently, several *randomization* schemes have been demonstrated for performing these low-rank matrix approximations. This thesis, reviews some of the randomization techniques for matrix decomposition. It also gives a brief introduction to a similar concept in signal processing called *Compressed Sensing*.

Estimating the approximate (low) rank of the input data matrix is essential to apply these randomized techniques to find the low-rank matrix approximations. There are many other applications in mathematical modeling and rank related problems, where estimating an approximate rank of a matrix is useful. In this thesis, two novel, efficient and computationally inexpensive techniques to find the approximate rank of a matrix are proposed and some applications where these techniques can be used are discussed.

The randomized techniques for matrix decomposition, requires generating and storing a large number of random numbers which could have practical complications, when the input matrix is of very large size. Here in this thesis, we demonstrate how matrices from *Error Control Coding* can be used in place of the random matrices in randomization techniques for matrix decomposition and compressed sensing. These matrices are very easy to generate and either deterministic or partially random in nature.

Contents

List of Figures

Chapter 1

Introduction

The thesis is divided into three major parts, constituting the three main chapters of the thesis. The first part of the thesis is a review of two topics which are highly studied in the recent years. The probabilistic algorithms for matrix decomposition are reviewed. Three techniques based on randomization for matrix decomposition are described and their performance is analyzed. A discussion on the type of matrices (singular value or spectral distribution) for which, these techniques are well suited is given. The second topic reviewed in this part, is the concept of Compressed Sensing(CS). A brief overview of compressed sensing and its framework is given in the last section of the following chapter.

The second part of the thesis introduces two novel, *fast and computationally inexpensive* techniques to find the approximate rank of a matrix. These techniques are fast and inexpensive in the sense that, the computational cost is much lower compared to the size of the matrix, when the matrix is large. This problem is motivated by the fact that, the randomization techniques reviewed in the first part requires us to know the approximate rank of input matrices and there are no simple methods in literature to compute them. Few applications are also discussed, where finding the approximate rank of a matrix using these techniques could be very useful, particularly in the low rank aprroximation problems and other related problems.

The third part of the thesis, introduces some structured matrices from Error Control Coding and describes their applications in the randomization techniques for matrix decomposition and in compressed sensing. These matrices replace, random Gaussian matrix used in randomization techniques and compressed sensing. Performance of these matrices are compared to performance using Gaussian matrices, for these applications. There are some obvious advantages of replacing Gaussian matrices by these structured matrices as disccussed in the chapter.

A brief outline of the thesis is as follows:

- Chapter 2 reviews three techniques that are based on probabilistic algorithms for matrix decomposition. The framework for matrix approximaton is first introduced. Next the three techniques are described along with their cost and error analysis. The concept of compressed sensing is introduced in the last section of the chapter.

- Chapter 3 discusses two novel techniques for finding the approximate rank of a matrix. The first technique, is based on Eigenvalue count method. And the second technique is based on the concept of Density of States. The performance of these techniques are discussed. The last section of the chapter discusses some of the applications of these techniques.

- Chapter 4 discusses applications of some of the matrices from Error Control Coding in randomization techniques for matrix decomposition and compressed sensing. The performance of these matrices are discussed and the results are analyzed.

- Chapter 5 give some important conclusions from the thesis work.

Chapter 2

Randomization Techniques for Matrix Decomposition

2.1 Introduction

In recent years, especially with the advent of "big data", we often encounter in many applications, large matrices that can be well approximated by a low rank basis. A low rank approximation of such matrices suffices in many of the scientific computations and data analysis, like: Principal Component Analysis (PCA), partial Singular Value Decomposition (SVD), Least-squares problem, solving Partial Differential Equations (PDE), etc., Recently, research focussed on developing techniques using randomization as a powerful tool for matrix decomposition of such matrices.

In this chapter we look at (review) some of these inexpensive techniques that have been demonstrated recently, which use randomization and probabilistic algorithms for matrix decomposition of such large approximately low rank matrices. The goal of this chapter is to present the framework for developing these inexpensive randomization techniques for matrix decomposition, analyze their computational cost, theoretical error performance and describe for what type of matrices these techniques work better. That is, the type of singular value or eigenvalue (spectral) distribution for which these methods outperform the traditional techniques.

Matrices which are low rank, have little information compared to their dimensions. That is, they have a far fewer degree of freedom. So, we can approximate such matrices by matrices of lower dimensions. The randomized schemes, find these approximations by using random sampling to identify a subspace that captures most of the action of these matrices. And then, compute the decomposition of the matrix, projected onto this subspace.

A similar concept in signal processing, called *Compressed Sensing* has attracted a lot of attention amongst the research community in recent years. In this chapter, we provide a framework for compressed sensing and show how these two topics are closely related. Compressed sensing also builds on the fact that we can have a sparse (very few non zero coefficents) representation for many signals and non linear optimization can be used to recover such signals with very few measurements. This is similar to the concept in randomization techniques where we know that approximate low rank matrices can be represented (action captured) with very few random samples (subspaces).

In the following sections of the chapter, the low rank matrix approximation framework is introduced, the intuition behind using random sampling for capturing the active subspaces of the matrix is given and then a prototype algorithm based on randomization schemes is described. Also two more techniques that work on similar lines (randomized scheme) for matrix decomposition are reviewed and we show how these techniques improve the performance of the prototype alogrithm. We also give the cost analysis and error analysis for all three techniques. This part of the review is based on the paper by Halko et al.[1]

In the last section of this chapter, the concept of compressed sensing is reviewed, by giving the framework for the technique, introducing an important property called the *Restricted Isometric Property* which is pivotal for compressed sensing and also discuss how the original signals can be recovered from the compressed signals using nonlinear optimization. This part of the review is based on the paper by Davenport et al.[2]

2.2 Matrix approximation framework

Given an $m \times n$ matrix A, we seek to compute a rank- k approximation of the form[3, 4, 5],

$$\underbrace{A}_{m\times n} \approx \underbrace{U}_{m\times k}\underbrace{\Sigma}_{k\times k}\underbrace{V^*}_{k\times n} = \sum_{j=1}^{k} \sigma_j u_j v_j^* \tag{2.1}$$

where
$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k \geq 0$ are the (approximate) singular values of A
$u_1, u_2, \ldots, u_k$ are orthonormal, the (approximate) left singular vectors of A, and
$v_1, v_2, \ldots, v_k$ are orthonormal, the (approximate) right singular vectors of A.

Computing this low rank approximation can be split into two computational stages. The first stage is to find a low-dimensional subspace that captures all the action of the matrix. The second stage is to project the large matrix onto this subspace and then compute the standard factorization (SVD) of the projected matrix.

- **Stage A:** Compute an approximate (orthonormal) basis that spans the range of the input matrix A. That is, a matrix Q having orthonormal columns and

$$A \approx QQ^*A$$

The basis matrix Q contains as few columns as possible, but it needs to be an accurate approximation of the input matrix.

- **Stage B:** Given a matrix Q that satisfies the above condition, use Q to compute a standard factorization (QR, SVD, etc.) of A, like the one in equation (2.1).

In the randomization techniques, Stage A is executed by random sampling and in the following sections of the chapter, we discuss three different methods that uses random vectors (or columns from structured random matrices) to find the approximate basis. Stage B can be executed by using well established deterministic methods.

The problem formulation is as follows. Suppose we have a matrix A and an error tolearance of ϵ, we seek a matrix Q with k orthonormal columns such that,

$$\|A - QQ^*A\| \leq \epsilon \tag{2.2}$$

where the norm is usually the l_2 operator norm. The theoretical minimum that can be achieve with such approximation, in terms of the singular values is given by the Eckart-Young theorem[6] which says,

$$\min_{rank(X)\leq k} \|A - X\| = \sigma_{k+1} \tag{2.3}$$

where $X = QQ^*A$ and the minimizer is where the columns of Q are the k dominant left singular vaulues of A.

For simplicity, k value is assumed to be known, for a given matrix A. In Chapter 3, we discuss some inexpensive methods to find the apporixmate rank k of a matrix. For a known rank k and an oversampling parameter p we try to obtain a matrix Q with $(k + p)$ orthonormal columns such that,

$$\|A - QQ^*A\| \approx \min_{rank(X)\leq k} \|A - X\| \tag{2.4}$$

A small number of additional columns p is chosen to have flexiblity and effectiveness in the computational methods we adopt.

2.2.1 Intuition

The intuition or the motivation to use randomness for fixed rank problems (in Stage A) is as follows. Suppose a random vector ω is generated, and form the product $y = A\omega$. That is, y is random samples from the range of A. And repeat this sampling k times:

$$y^{(i)} = A\omega^{(i)}, i = 1, 2, ..., k \tag{2.5}$$

the set $\{\omega^{(i)} : i = 1, 2, ..., k\}$ of random vectors is likely to be in general linear position. This means, the random vectors form a linearly independent set and no linear combination are likely to be in the null space of A.

The set $\{y^{(i)} : i = 1, 2, ..., k\}$ spans the range of A as the set entries are also linearly independent. Also a larger sample set, $\{y^{(i)} : i = 1, 2, ..., k + p\}$ is used to improve the chances of spanning the entire range of A. To produce an orthonormal basis that spans the range of $A, \mathcal{R}(A)$, we just need to orthonormalize these sample vectors.

Suppose, we consider the input matrix A to be of the form, $A = B + E$ where B is a rank-k matrix containing the information that we are after and E is a small perturbation. Then we have,

$$y^{(i)} = A\omega^{(i)} = B\omega^{(i)} + E\omega^{(i)}, i = 1, 2, ..., k + p \tag{2.6}$$

the enriched set $\{y^{(i)} : i = 1, 2, ..., k+p\}$ of samples has a much better chance of spanning the required subspace of B. Usually in practice, $p = 5$ or $p = 10$ is chosen.

2.3 An algorithm based on randomization

Algorithm 1 Prototype Algorithm

Input: An $m \times n$ matrix A, a target rank k. And an oversampling parameter p.

Output: Rank-k factors U, Σ, and V in an approximate SVD $A \approx U\Sigma V^*$.

1. Draw an $n \times \ell$ Gaussian random matrix Ω. $\ell = (k + p)$
2. Form the $m \times \ell$ sample matrix $Y = A\Omega$.
3. Form an $m \times \ell$ orthonormal matrix Q such that $Y = QR$.
4. Form the $\ell \times n$ matrix $B = Q^*A$.
5. Compute the SVD of the small matrix $B : B = \hat{U}\Sigma V^*$.
6. Form the matrix $U = Q\hat{U}$.

Given the background discussed in the previous sections, a prototype algorithm is developed [7][8]as given above, that constructs a subspace of lower dimension (dimensions close to the approximate rank of the matrix). The subspace is expected to capture most of the action of the matrix (Stage A). For Stage B, the algorithm uses standard

svd to find the decomposition of the matrix in this lower dimensional subspace (Stage B).

The prototype algorithm is similar to the first step in the subspace iteration with a random initial subspace.[9] As the dimension of this initial subspace is slightly higher than the invariant subspace that we are trying to approximate, usually no further iterations are required to get a high quality approximation.

2.3.1 Cost

An important analysis of the performance of any algorithm is the computational cost. Stepwise cost of our prototype algorithm is as follows:

- Cost of Step 1: Generating an $n \times \ell$ random matrix. ($\ell n T_{rand}$).
- Cost of steps 2 and 4: Application of A and A^* to ℓ vectors. So, its ℓ matrix-vector multiplications (ℓT_{mult}).
- Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times \ell$ and $\ell \times n$

The number of flops (T_{basic}), required by the Algorithm is,

$$T_{basic} \sim \ell T_{mult} + \ell n T_{rand} + \ell^2 m \tag{2.7}$$

Stage A dominates the cost in our matrix approximation framework. Within Stage A, the computational bottleneck is usually the matrix-matrix product $A\Omega$ in Step 2 of the prototype algorithm. But based on the structure of the input matrix and the computational archtectures, this matrix multiplication can be evaluated highly efficiently. (See section 2.5).

The cost of this prototype algorithm is very similar to any standard matrix decomposition methods. If we consider the standard method, Lanczos (or Arnoldi for non-symmetric matrices)[10]. The cost T_{Krylov} of these method is approximately given as, satisfy:

$$T_{Krylov} \sim \ell T_{mult} + O(\ell^2(m+n)) \tag{2.8}$$

Cost is dominated by the $2mn\ell$ flops required for steps 2 and 4. The $O(mn\ell)$ flop count is the same as that of standard methods such as Golub-Businger[11]. However, the algorithm above requires no random access to the matrix A - the data is retrieved in *two passes*.

2.3.2 Error analysis

The second parameter for performance analysis of a given algorithm is the error analysis. The error for matrix approximation methods is defined in general as,

$$e_k = \left\| A - A_k^{computed} \right\| \tag{2.9}$$

e_k is a random variable in our case, whose theoretical minimal value according to the Eckart-Young-Mirsky theorem[12] is,

$$\sigma_{k+1} = \min \{ \|A - A_k\| : A_k = rank - \text{“}k\text{”} \} \tag{2.10}$$

It is found that, for the prototype alogirthm, the expectation of $\frac{e_k}{\sigma_{k+1}}$ is large, and has very large variance. So, a small oversampling is used. If p is a small integer (think $p = 5$), then we often can bound e_{k+p} by something close to σ_{k+1}.

An extensive error analysis of the prototype algorithm can be found in [1],[7] and [8]. The Theorem 1.1 in [1] states,

Theorem 2.3.1 *Suppose that A is a real $m \times n$ matrix. Select a target rank $k \geq 2$ and an oversampling parameter of $p \geq 2$, where $k + p \leq \min\{m, n\}$. Execute the prototype algorithm with standard Gaussian test matrix to obtain an $m \times (k + p)$ matrix Q with orthogonal columns. Then*

$$\mathbb{E}\|A - QQ^*A\| \leq \left[1 + \frac{4\sqrt{k+p}}{p-1} \sqrt{\min\{m, n\}} \right] \sigma_{k+1} \tag{2.11}$$

where $\mathbb{E}$ denotes the expectation with respect to the random test matrix and σ_{k+1} is the $(k + 1)$th singular value of A.

In [1], they further simplify this equation and give a precise form as follows,

$$\mathbb{E}\left\|A - A_{k+p}^{computed}\right\| \leq \left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + \frac{\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{n}\sigma_j^2\right)^{1/2}. \tag{2.12}$$

Moreover, if we have a strong concentration of measure, this implies $\Longrightarrow$ Variance of $\left\|A - A_{k+p}^{computed}\right\|$ is small.

A detailed error analysis, a proof for the above theorem and the derivation of the above equation from the theorem can be seen in [1]. Given the above error equation, it is obvious that the performance of randomization techniques are dependent on the behavior of the singular values of the input matrix, i.e., the type of spectral distribution the input matrix has. We can consider two different cases of singular value distribution given below:

Case 1 *- the singular values decay rapidly:*

If (σ_j) decays sufficiently rapidly after σ_k, then we shall have that $\left(\sum_{j=k+1}^{n}\sigma_j^2\right)^{1/2} \approx \sigma_{k+1}$ in (2.12), then we are fine. A minimal amount of over-sampling (say $p = 5$ or $p = 10$) drives the error down close to the theoretically minimal value. Thus, the randomization algorithms perform the best when there is a big gap in the spectrum of the input matrix or the singular values decay rapidly after the first k singular values.

Case 2 *- the singular values do not decay rapidly:*

In the worst case, we have in (2.12),

$$\left(\sum_{j=k+1}^{n}\sigma_j^2\right)^{1/2} \approx \sqrt{n-k}\sigma_{k+1}.$$

In this case, if the size n is very large, and σ_{k+1}/σ_1 is not that small, then we could lose all accuracy and the randomization algorithms will fail completely.

So in general we can conclude that, the randomization algorithms work best when there is a gap in the spectrum (sinuglar value distribution) of our input matrix. The prototype algorithm performs poorly when the singular values of the input matrix decay

slowly. To overcome this behavior and improve the performance of the randomization algorithms for matrices with slowly decaying singular values, [13] proposes the power method which is discussed in the next section.

2.4 Power method

The prototype algorithm work well for matrices whose singular values decay rapidly, but results in poor bases when the input matrices have flat singular spectrum or when the matrices are very large. Also, the error depends on how quickly the singular values decay. The faster the singular values decay — the stronger the relative weight of the dominant modes in the samples. So, an algorithm was proposed by [13] and [14], that combines the proto-algorithm with the power iteration that improves the performance of the randomization algorithms.

Idea: The matrix $B = (AA^*)^q A$ has the same left singular vectors as A, and its singular values are

$$\sigma_j(AA^*)^q A = (\sigma_j(A))^{2q+1} \tag{2.13}$$

Thus, the matrix B has much faster decay in its singluar values than matrix A. So, we modify the sampling of the matrix or apply the randomized sampling scheme as,

$$Y = (AA^*)^q A\Omega \tag{2.14}$$

instead of $Y = A\Omega$.

This is like giving weights to different singular vectors. So, we reduce the weights of singular vectors corresponding to smaller singular values compared to the dominant singular vectros by taking the powers of the matrix to be analyzed. This power method is also known as *Subspace Iteration* in Numerical Linear Algebra. Given below is the algorithm for this power method.

Algorithm 2 Power Method

Input: An $m \times n$ matrix A, a target rank k, oversampling parameter p and a small integer q.

Output: Rank-k factors U, Σ, and V in an approximate SVD $A \approx U\Sigma V^*$.

1. Draw an $n \times \ell$ Gaussian random matrix Ω. $\ell = (k + p)$
2. Form the $m \times \ell$ sample matrix $Y = (AA^*)^q A\Omega$.
3. Form an $m \times \ell$ orthonormal matrix Q such that $Y = QR$.
4. Form the $\ell \times n$ matrix $B = Q^*A$.
5. Compute the SVD of the small matrix $B : B = U\Sigma V^*$.
6. Form the matrix $U = Q\hat{U}$.

This algorithm is more suitable for those matrices whoses singluar values decay slowly. But, it requires $2q+1$ times as many matrix-vector multiplications as Algorithm 1. However, the algorithm is more accurate. So, the computational cost increases to,

$$T_{power} = (2q + 2)\ell T_{mult} + O)\ell^2(m + n)) \tag{2.15}$$

Also, if the basis produced by the prototype algorithm results in an approximate error of C, then this power scheme must produce an approximate error of $C^{1/2q+1}$. A brief discussion of this error analysis is given below.

2.4.1 Error analysis

The expected error for the power method is given by,

$$E\left\|A - U\Sigma_{(k)}V^*\right\| \leq \sigma_{k+1} + \left[1 + 4\sqrt{\frac{2\min\{m,n\}}{k-1}}\right]^{1/2q+1} \sigma_{k+1} \tag{2.16}$$

$E\left\|A - A_{k+p}^{computed}\right\|$ converges exponentially faster to the optimal value of σ_{k+1} as q increases. A detailed analysis and the derivation of this error equation can be see in [1] and [13]. All these error analysis, assumes exact arithmetic. But in real life,

we execute these algorithms in floating point arithmetic and some rounding errors and other complications arise. For example, the top singular values might blow up. These issues can be handled by careful implementation.

The modified scheme obviously comes at a substantial cost; $2q + 1$ passes over the matrix are required instead of 1. However, q can often be chosen quite small in practice, $q = 2$ or $q = 3$, say.

Next, we discuss another method that reduces the cost of matrix-vector multiplications (T_{mult}) by exploiting structured random matrices to find the initial basis.

2.5 Accelerated techniques

In this section, we discuss some techniques that can improve the cost of an approximate $\ell-$rank factorization of a general $m \times n$ matrix from $O(mn\ell)$ to roughly $O(mn\log(\ell))$ by using *structured* random matrix. These algorithms were introduced in [15] and [16] Here $\ell = k + p$ as used earlier. The most expensive step (bottleneck) in Algorithm 1 is computing the matrix product $A\Omega$. The key idea is to replaces the standard Gaussian Matrix by a structured random matrix that requires only $O(mn\log(\ell))$ flops to compute this product. So, the accelerated technique is same as the prototype algorithm except that , we use a structured random matrix instead of Gaussian matrix to sample the input matrix and form the orthonormal basis.

For instance, use of a *subsampled random Fourier Transform (SRFT)* is the most common example of structured random matrices that reduce the product cost. An SRFT is a $n \times \ell$ matrix of the form,

$$\Omega = \sqrt{\frac{n}{l}} DFR \tag{2.17}$$

where,

- D is a diagonal matrix whose entries are i.i.d. random variables drawn from a uniform distribution on the unit circle in $\mathcal{C}$
- F is the discrete Fourier transform, $F_{jk} = \frac{1}{\sqrt{n}} e^{-2\pi(j-1)(k-1)/n}$

- R is a matrix whose entries are all zeros except for a single, randomly placed 1 in each column. (In other words, the action of R is to draw columns at random from DF.)

Then the algorithm for finding the range of the input matrix using this (fast) structured randomized method is given below.

Algorithm 3 Accelerated Randomized Range Finder

Input: An $m \times n$ matrix A, small integer ℓ.

Output: Compute an $m \times \ell$ orthonormal matrix Q whose range approximates the range of A.

1. Draw an $n \times \ell$ SRFT matrix Ω, as defined in (2.17).
2. Form the $m \times \ell$ sample matrix $Y = A\Omega$ using (subsampled) FFT.
3. Construct an $m \times \ell$ orthonormal matrix Q whose columns form basis for thr range of Y, such that $Y = QR$.

For the accelerated randomized technique, the total number T_{struct} of flops required for a rank$-\ell$ factorization of a matrix is,

$$T_{struct} \sim mn \log(\ell) + \ell^2 n \tag{2.18}$$

In this case, ℓ is substantially larger than the numerical rank k of the matrix. It is usually taken to be of the order $\ell = 2k$. Also orthogonalization can be performed with $O(k\ell n)$ flops as the columns are almost linearly dependent. In literature, the DFT matrix in 2.17 are replaced by other structured matrices like Subsampled Hadamard transform, Wavelets, Random chains of Given's rotations and many more have also been suggested. See [17] and its bibliography and [18].

2.6 Compressed sensing

The concept of compressed sensing and randomized matrix approximations come from similar intuitions.

Compressed sensing looks at signals obtained by *transform coding* which results in the basis or frame with *sparse* or *compressible* representations of the signals. That is, a signal of length n, can be represented by $k \ll n$ nonzero coefficients or the signal is well approximated by a signal with only k nonzero coefficients. This concept is called *Sparse Approximation* and has similar intuition as in the *Low-rank Approximations* of matrices described in the previous sections. In compressed sensing, many signals have implicit sparsity or that they can be approximated by some sparse signal representations and can be recovered by (a few number of) random sampling the signal in its sparse representation[19][2]. In low-rank approximations, we can capture most of the action of these matrices with much smaller subspace and by (a few number of) random sampling of the columns. The framework for compressed sensing is given, next.

A given signal x, can be represented as a linear combination of some basis vectors. A set $\{\phi_i\}_{i=1}^n$ is called a basis in $\mathbb{R}^n$ if these vectorsare linearly independent and span $\mathbb{R}^n$. So, for any $x \in \mathbb{R}^n$, we can represent it as,

$$x = \sum\nolimits_{i=1}^{n} s_i \phi_i \quad or \quad x = \Phi s \tag{2.19}$$

where Φ is the $n \times n$ matrix with columns given by ϕ_i and $s = \{s_i\}_{i=1}^n$ are weighted coefficients given by $s_i = \langle x, \phi_i \rangle$ or $s = \Phi^T x$, if the basis is an orthonormal basis, satisfying

$$\langle \phi_i, \phi_j \rangle = \begin{cases} 1, & i = j; \\ 0, & i \neq j; \end{cases}$$

Signals are often approximated by fewer number of basis or dictionary. Such an approximation is called a sparse signal representation. A k-sparse signal has at most k nonzero entries, i.e,. $\|x\|_0 \leq k$. We denote a set of all k-sparse signals by,

$$\Sigma_k = \{x : \|x\|_0 \leq k\}$$

This is similar to representing the set of all matrices with approximate rank of at-most k. But in practice, the signals themselves might not be sparse, but may admit a sparse representation in some basis Φ. Such signals are also referred to as being

$k-$sparse, as we can express such signals x as $x = \Phi c$ with $\|c\|_0 \leq k$ many transformations like Fourier transforms, Wavelet transforms[20] etc,. usually lead to $k-$sparse representations of signals.[21]

Signals that are not truly sparse but can be well-approximated by a sparse singal are called compressible or approximately sparse or relatively sparse signals. These signals are the foundation for *Transform Coding*[22]. In Transform Coding, the full n-sample signal is first acquired; i.e., the complete set of transform coefficients is computed $c = \Phi^T x$. Then the k largest coefficients are located and $n-k$ smallest coefficents are discarded. This method is highly inefficient if the initial number of samples n is very large compared to the number of significant coefficents k. The idea in compressed sensing is to address this shortcoming of transform coding and directly acquire a compressed signal representation without having to acquire n samples.

In the compressed sensing framework, we consider a linear measurement system with m measurements which can we represented as,

$$y = Ax = A\Phi c \tag{2.20}$$

where A is a fixed $m \times n$ matrix and $y \in \mathbb{R}^m$. Here it is a non-adaptive measurement. Also we are considering x to be of finite length n. It is important that our measurement matrix A allows proper reconstruction of the original length n signal x with only $m < n$ measurements.

Reconstruction is only possible if the signal x is $k-$sparse. Otherwise this problem is ill-conditioned. The problem is solvable for $m \geq k$ if the matrix A statisfies a necessary and sufficient condition known as the Restricted Isometry Property (RIP).

2.6.1 Restricted Isometry Property

The Restricted Isometry Property introduced in [23] and [24] is a strong condition that needs to be satisfied for recovery guarentees, particularly when there is *noise* in the signal or the signal is corrupted by errors such as Quantization etc,.

Definition 2.6.1 *A matrix A satifies the Restricted Isometry Property (RIP) of the*

order k if there exists an $\epsilon_k \in (0,1)$ such that

$$(1-\epsilon_k)\|x\|_2^2 \leq \|Ax\|_2^2 \leq (1+\epsilon_k)\|x\|_2^2 \tag{2.21}$$

holds for all $x \in \Sigma_k$

This means that, if a matrix A satifies the RIP of order $2k$, then the matrix approximately presereves the distance between any pair of $k-$sparse vectors. It is interesting that, we can achieve RIP with high probability if we select the measurement matrix A to be a random matrix. Just as in the randomization techniques discussed in the previous sections, a standard iid (idependent and identically distributed) Gaussian Matrix with zero mean and variance $1/n$ satisfies the RIP with high probability if $m \geq ck\log(n/k)$ and also has some useful properties. For example, it is incoherent and universal regardless of the choice of the basis.

Several other matrices like Vandermode Matrices, Bernoulli distributed matrices, sub-Gaussian matrices and the structured random matrices discussed in section2.5 have also been used as measurement matrices. In chapter 4 we show how partially random matrix from error correction coding can be used as measurement matrix for compressed sensing.

2.6.2 Signal Recovery

The reconstruction algorithm will have to take in the m measurements vector y, the measurement matrix A and use the knowledge that the original signal x is sparse or compressible and reconstruct the length$-n$ signal x or the equivalent sparse coefficients c.

The ℓ_0 norm minimization of the form

$$\hat{x} = \arg\min_z \|z\|_0 \; \textit{subject to } z \in \mathcal{B}(y) \tag{2.22}$$

where $\mathcal{B}(y)$ means $\hat{x}$ is consistent with measurements y, seems to be a natural choice to recover sparse signals and in theory it must recover the $k-$sparse signal x. But unfortunately, solving (2.22) is NP complete and numerically unstable.

Surprisingly it is shown that, ℓ_1 norm minimization of the form

$$\hat{x} = \arg\min_z \|z\|_1 \; \textit{subject to } z \in \mathcal{B}(y) \tag{2.23}$$

provided $\mathcal{B}(y)$ is convex, is feasible and can recover the $k-$sparse signal x. Infact, we know that $\mathcal{B}(y) = z : Az = y$, resulting in a problem that can be solved by linear programming. The above ℓ_1 norm minimization is known as *Basis Pursuit*[25]. There are intuitive reasons that show, this ℓ_1- minimization indeed results in sparse solutions[2]. It also recovers compressible signals with high probability. Several ℓ_1- minimization algorithms have been proposed to solve the recovery problem in compressed sensing.

Chapter 3

Estimating the Approximate Rank of a Matrix

3.1 Introduction

In the previous chapter we saw that, the randomized techniques for matrix decomposition assumes that, the rank k of the matrix is known a priori. In many applications, it is required (or useful) to have an approximate estimate of the rank of a matrix. In problems like low rank approximations [26], fixed rank and relaxed rank optimization problems, an estimate of the appproximate rank of input matrix is helpful. There are lot of literature on the approximate rank problem[3], which is motivated by problems from various topics in mathematical modeling and optimization. In numerical methods and applications including the randomization techniques discussed in the previous chapter, it is required to know an approximate rank of a given matrix prior to applying the algorithm. We can see similar motivation for find approximate rank of a matrix in [27] and [28]

In this chapter we discuss two novel techniques to find an approximate rank of the given input matrix. The first method is based on polynomial expansion of trace of an eigen-projector to find an eigenvalue count in the given interval, which in turn can be used to estimate the approximate rank, because rank of a matrix is the count of number of nonzero eigenvalues. This method is based on the eigenvalue count techniques

proposed in [29].

The second method is based on the concept called, Density of States (DoS or Spectral Density). Here, the idea is to integrate the Density of States function to obtain the approximate rank. Many techniques have been proposed in [30] to find the Spectral Density for a given matrix. Two of these techiques are used in this chapter to find the approximate rank.

An important point is that, both these methods for estimating the approximate rank, require the input matrix A, to be Hermitian. When the matrix is either rectangular or non-hermitian, (usually the matrices are thin sand tall) we consider the eigenvalue count of the matrix A^TA instead of matrix A. The rank of A and A^TA are the same. The matrix A^TA is not formed explicitly as this is very expensive. But instead in both these methods, we will see that, we just need additional matrix-vector multiplication to form A^TAv instead of Av (where v is some vector). So, replacing A by A^TA amounts a small additional cost for these additional matrix-vector multiplications. But for simplicity, both the methods are discussed in the following sections, assuming the input matrix is an $n \times n$ Hermitian matrix A.

We develop two different approaches for finding an approximate rank of a matrix in this Chapter.

1. Finding the approximate rank using Eigenvalue count
2. Finding the approximate rank using the concept of Density of States

In the following sections, we first describe the eigenvalue count method to find the approximate rank of a matrix. This method is based on stochastic aprroximation of the trace of the Eigen-projector of the matrix. And we discuss the performace of this method using various examples. We introduce the concept of *Density of States* and decribe two different methods to find the approximate rank of a matrix using the concept of Density of States. It turns out that one of these methods upon simplification, results in the same equation for the approximate rank of a matrix as in the case of the eigenvalue count method. We discuss the performance of this method and also give some applications where finding the approximate rank of a matrix is required (or useful).

3.2 The approximate rank by eigenvalue count method

One of the obvious ways to find the rank of a matrix is to count the number of non zero eigenvalues (or singular values) of the matrix. A large matrix which is structurely low rank will have a number of eigenvalues (or singular values) close to zero. So, counting the number of eigenvalues above a small threshold say $\epsilon > 0$ should give us an approximate estimate of the rank. (Usually, we find the rank of $A^T A$ which is semi-positive definite).

Some eigensolvers based on divide and conquer methods and certain other applications require an estimate of the number of eigenvalues in a given interval[31, 32]. [29] discusses few techniques that give a stochastic estimate of the eigenvalue count in an interval. In this section we describe a method that uses one of these eigenvalue count methods to find an approximate rank of a Hermtian matrix. [29] discusses methods to estimate the eigenvalue count in the interval $[a, b]$ using polynomial and rational approximation filtering combined with stochastic procedure. The idea is that, an approximate rank of a large sparse matrix A with low rank structure must be equal to the eigenvalue count in the interval $[\epsilon, \lambda_{\max}]$, where $\lambda_{\max}$ is the maximum eigenvalue of the matrix A and ϵ is a small threshold above zero, whose value depends on the application. If the matrix is not semipositive definite, then we find the eigenvalue count in the interval $[-\epsilon, \epsilon]$ and subtract it, from the size of the matrix to get an approximate rank.

This section introduces a method that estimates the eigenvalue count $\eta_{[a,b]}$ over the interval $[a, b]$, in a relatively inexpensive way. The method seeks an approximation to the trace of the eigen- projector or a spectral projector P associated with the eigenvalues inside the interval $[a, b]$. The spectral projector is expanded as a polynomial function of A and the trace is computed using stochastic trace estimators [33, 34]. The polynomial function can be treated as a filtering technique based on the Chebyshev polynomials.

This problem of eigenvalue count is closely related to the concept of estimating "Density of States" described in the following sections. The eigenvalue count $\eta_{[a,b]}$ is integral of the spectral distribution or spectral density over the interval $[a, b]$. It can

be noted that the Kernel Polynomial method described in the section 3.4, that used Chebyshev polynomials to compute the DoS and the trace projector method described in this section bear a lot of similarities. In fact, we show that, both these methods result in the same equation for the approximate rank of a matrix.

For a given $n \times n$ Hermitian matrix A, we compute an estimate of the eigenvalue count in the interval $[a, b]$, (in our case $[\epsilon, \lambda_{\max}]$) by finding an approximation to the trace of the eigen-projector:

$$P = \sum_{\lambda_i \in [a,b]} u_i u_i^T \tag{3.1}$$

The trace of P is equal to the number of terms in the sum (3.1), as the eigenvalues of a projector are either ones or zeros. So, the trace of P should yield the number of eigenvalues in $[a, b]$. Therefore the eigenvalue count $\eta_{[a,b]}$ is related to the projector P as,

$$\eta_{[a,b]} = \mathrm{Tr}(P). \tag{3.2}$$

The projector P is not available in practice but can be approximated inexpensively in the form of a polynomial of the matrix A. The projector can be interpreted as a step function of A and can be approximated by a polynomial of the form,

$$P = \psi(A) \quad \text{where} \quad \psi(t) = \begin{cases} 1, & if\ t \in [a, b] \\ 0, & otherwise \end{cases} \tag{3.3}$$

The idea is to expand $\psi(t)$ as a sum of Chebyshev polynomials . In this form, it is possible to estimate the trace of P using the stochastic estimators [33, 34].

A stochastic estimator known as Hutchinson's unbiased estimator that uses matrix-vector products to approximate the trace of any generic matrix A can be used. This method estimates the trace $\mathrm{Tr}(A)$ using randomly generated vectors $v_l, l = 1, \ldots, n_{vec}$ with entries ± 1 with equal probablity and then take the average over the sample $v_l^T A v_l$,

$$\mathrm{Tr}(A) \approx \frac{n}{n_{vec}} \sum_{l=1}^{n_{vec}} v_l^T A v_l \tag{3.4}$$

If we approximate the projector (or the step function) with the chebyshev polynomials, $P \approx \psi(A)$. Then, using the stochastic estimator for trace makes sense because, forming the product $\psi(A)v_l$ is inexpensive than forming the polynomial themselves.

The approximate rank of a matrix A can be estimated as,

$$r \approx \eta_{[\epsilon,\lambda_{max}]} = \frac{n}{n_{vec}} \sum_{l=1}^{n_{vec}} \left((v_l)^T \psi(A) v_l\right)$$

3.2.1 Polynomial expansion

As discussed earlier, the step function $\psi(t)$ can be expanded as a M degree Chebyshev polynomial series:

$$\psi(t) = \sum_{k=0}^{M} \gamma_k T_k(t) \tag{3.5}$$

where T_k are the k-degree Chebyshev polynomials of the first kind, and the coefficients γ_k are the expansion coefficients for the step function over a general interval $[a, b]$, which is given by,

$$\gamma_k = \begin{cases} \frac{1}{\pi}(\cos^{-1}(a) - \cos^{-1}(b)) & : \ k = 0, \\ \frac{2}{\pi}\left(\frac{\sin(k\cos^{-1}(a)) - \sin(k\cos^{-1}(b))}{k}\right) & : \ k > 0 \end{cases}$$

So, the polynomial expansion of the projector P in terms of the matrices $T_k(A)$ is,

$$P \approx \psi_p(A) = \sum_{k=0}^{M} \gamma_k T_k(A) \tag{3.6}$$

We have to make sure that the eigenvalues of A (or $A^T A$ if non-hermitian) lie in the interval $[-1, 1]$ by using a linear transformation discussed later(See eq: (3.10)). Many times when we expand step functions using Chebyshev polynomials, we encounter bad oscillations near the boundary called *Gibbs Oscillations*. To suppress this behavior, damping multipliers like Jackson coefficients are used. So, we can replace (3.5) by,

$$P \approx \psi_p(A) = \sum_{k=0}^{M} g_k^M \gamma_k T_k(A) \tag{3.7}$$

The Jackson coefficients g_k^M are given by the formula (developed in [35]),

$$g_k^M = \frac{\left(1 - \frac{k}{M+2}\right)\sin(\alpha_M)\cos(k\alpha_M) + \frac{1}{M+2}\cos(\alpha_M)\sin(k\alpha_M)}{\sin(\alpha_M)}$$

where $\alpha_M = \frac{\pi}{M+2}$. There is also a shorter form to write these coefficients given by,

$$g_k^M = \frac{\sin(k+1)\alpha_M}{(M+2)\sin(\alpha_M)} + \left(1 - \frac{k+1}{M+2}\right)\cos(k\alpha_M)$$

Thus, the approximate rank of a matrix A using the Eigenvalue Count method is given by,

$$r \approx \eta_{[\epsilon,\lambda_{max}]} = \frac{n}{n_{vec}} \sum_{l=1}^{n_{vec}} \left[\sum_{k=0}^{M} \gamma_k (v_l)^T T_k(A) v_l \right] \tag{3.8}$$

From the above equation we can see that, this approach need only matrix-vector multiplications of the form $w_k = T_k(A)v$ for a given initial vector v, to compute the rank. We have the three term recurrence relation for the Chebyshev polynomials $T_{k+1}(t) = 2tT_k(t) - T_{k-1}(t)$ which results in the iteration,

$$w_{k+1} = 2Aw_k - w_{k-1} \tag{3.9}$$

The Chebyshev polynomials in general are defined in terms of cosines only over the interval $[-1, 1]$. So, the eigenvalues of matrix A needs to be restricted to this interval.

For a general matrix A whose eigenvalues are not in the interval $[-1, 1]$, a linear transformation mapping needs to be applied to A, to restrict the eigenvalues of the matrix within the interval. We apply the following transformation to the matrix,

$$B = \frac{A - cI}{d} \tag{3.10}$$

where

$$c = \frac{\lambda_{\min} + \lambda_{\max}}{2},\ d = \frac{\lambda_{\max} - \lambda_{\min}}{2}$$

The maximum and the minimum eigenvalues are obtained by Ritz values from a standard Lanczos iteration. Also, as Chebyshev polynomials are defined only in the interval $[-1, 1]$, while computing γ_k, we need to transform the interval $[\epsilon, \lambda_{max}]$ to $[\hat{\epsilon}, 1]$, where

$\hat{\epsilon}$ is the linear transformation of ϵ using (3.10).

To find the rank of a rectangular (or non hermtian) matrix in general, we find the rank of A^TA. For which, we just need additional matrix-vector multiplication to apply A^T to the vector Aw_k, in the first term of the recurrence equation (3.9).

The main advantage of this method is that, the polynomial expansion does not require any factorization of A. This makes the method inexpensive when the matrix A is large. It is also seen that, any sequence of random vectors v_k of unit 2-norm will do as long as the mean of their entries is zero [36].

3.2.2 Choosing the threshold ϵ

The approximate rank estimated by the above technique, depends upon the threshold ϵ chosen. The choice of ϵ is application dependent. It depends on the accuracy of the approximation needed and also, some applications might have a range in which the rank r must lie.

It can be seen that, choosing different ϵ and finding the corresponding rank for each ϵ is almost free of cost beacuse, it's only the γ_ks that vary for different choice of ϵ and the product (expensive) term $(v_l)^T T_k(A)v_l$ remains the same. Thus, we can use a range of ϵ and find the corresponding ranks and choose the one, that best fits the application.

We saw in the previous chapter that the theoretical minimum of the error for a rank-k approximation (by Eckart-Young theorem) given in (2.3) is σ_{k+1}. ($k+1$st singular value of the matrix). This is the largest singular value whose corresponding singular vectors were not used to approximate the matrix. So, if we choose to approximate a matrix (say using the techniques defined in the chapter 2) based on the approximate rank found using the above technique, we are approximating the matrix using the singular values (and the corresponding singular vectors) above ϵ. Then, the error must be interms of the largest singular value of the matrix that is less than ϵ. That is, the error will be in terms of $\max(\sigma_i) \leq \epsilon$. (Or we can say the error will in terms of ϵ).

So, if r is the approximate rank estimated for a matrix A using some threshold ϵ,

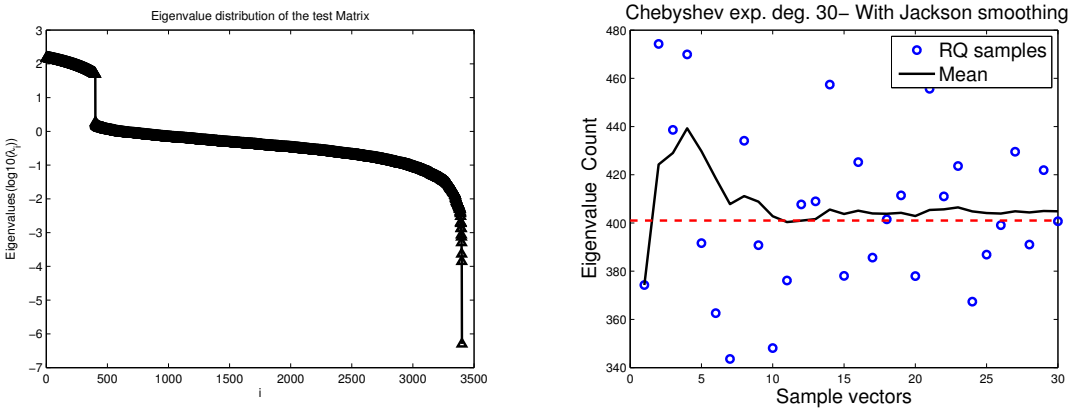
then the theoretical minimum error for a rank-r approximation of A is,

$$\min_{rank(X)\leq r} \|A - X\| = \sigma_{r+1} \leq \epsilon \tag{3.11}$$

Thus, we can choose ϵ based upon the error tolerence desired for the given application.

3.2.3 Results

In the first example to illustrate the eigenvalue count method to find an approximate rank, we generate a 3401×3401 symmetric matrix with the eigenvalue distribution shown in Fig. 3.1 (A). This matrix is approximately low rank. That is, there is a gap in the eigenvalue distribution between the top 401 eigenvalues and the remaining 3000 eigenvalues, in the sense that, the top 401 eigenvalues (linearly decreasing from 150 to 50) are numerically much higher than the remaining 3000 eigenvalues (values are in 10^{-1}). So, even though this matrix is a full rank symmetric matrix, it can be well approximated by just the top 401 eigenvalues and the corresponding eigenvectors. Thus, the approximate rank of this matrix is 401.



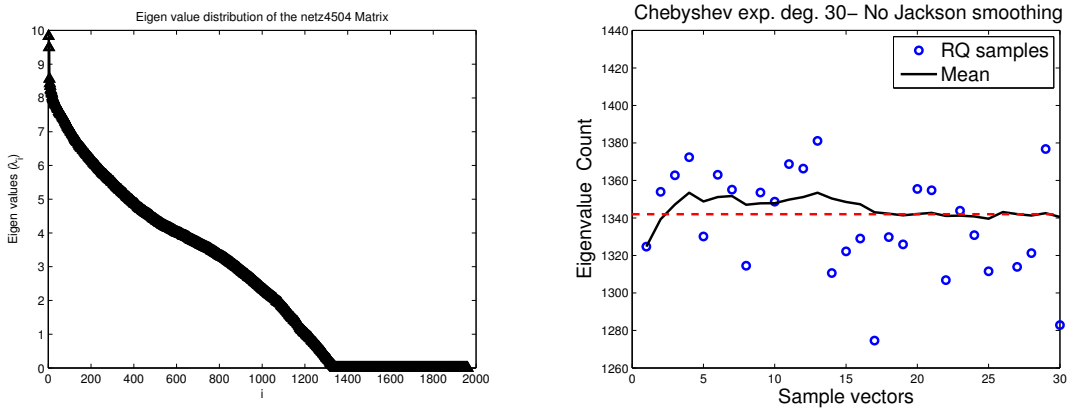
(A) (B)

Figure 3.1: (A) The eigenvalue distribution (logplot) of a 3401×3401 symmetric matrix . (B)Approximate rank (Eigenvalue Count) using Chebyshev Expansion of degree 30.

Figure 3.1(B) shows the estimated approximate rank (Eigenvalue Count) using Chebyshev Expansion of degree 30. 30 sample vectors were used and the interval

$[\epsilon, \lambda_{max}]$ for eigenvalue count was set to $[10, 150]$, that is, $\lambda_{max} = 150$ and $\epsilon = 10$. In the figure, the blue circles indicate the rank value estimated with the kth sample vector and the black line is the running average of these estimated rank values.(Red dotted line is the actual aprroximate rank). The computed average was $r = 404.83$. This experiment shows that, the eigenvalue count method is pretty accurate. In this case, Jackson-Chebyshev (with damping) gives better results than Chebyshev (without damping) because, there is gap in the eigenvalues and overfitting does not matter, in this case.

In the second experiment, we illustrate finding an approximate rank using the eigenvalue count method, with matrix from AG-Monien group called netz4504. The matrix comes from a 2D finite element problem and is available in The University of Florida Sparse Matrix Collection [37]. The matrix is of size $1,961 \times 1,961$ with $nnz = 5,156$, the nonzeros entries. The structural rank of the matrix is $1,344$. (Which is the approximate rank of this matrix). The eigenvalue distribution of the matrix is given fig.3.2(A).



(A) (B)

Figure 3.2: (A) The eigenvalue distribution of a $1,961 \times 1,961$ matrix AG-Monien/netz4504. (B)Approximate rank (Eigenvalue Count) using Chebyshev Expansion of degree 30.

Figure 3.2(B) shows the estimate of the approximate rank (Eigenvalue Count) using Chebyshev Expansion of degree 30. 30 sample vectors were used and the interval $[\epsilon, \lambda_{max}]$ for eigenvalue count was set to $[0.01, 9.828]$, that is, $\lambda_{max} = 9.828$ and $\epsilon = 0.001$. The computed average was $r = 1340.7$. This experiment shows that, the eigenvalue count method is pretty accurate. In this case, Chebyshev (without damping) gives better results than Jackson-Chebyshev (with damping) because, Jackson damping tends to overfit and in this case, we do not have any gap in the eigenvalue distribution.

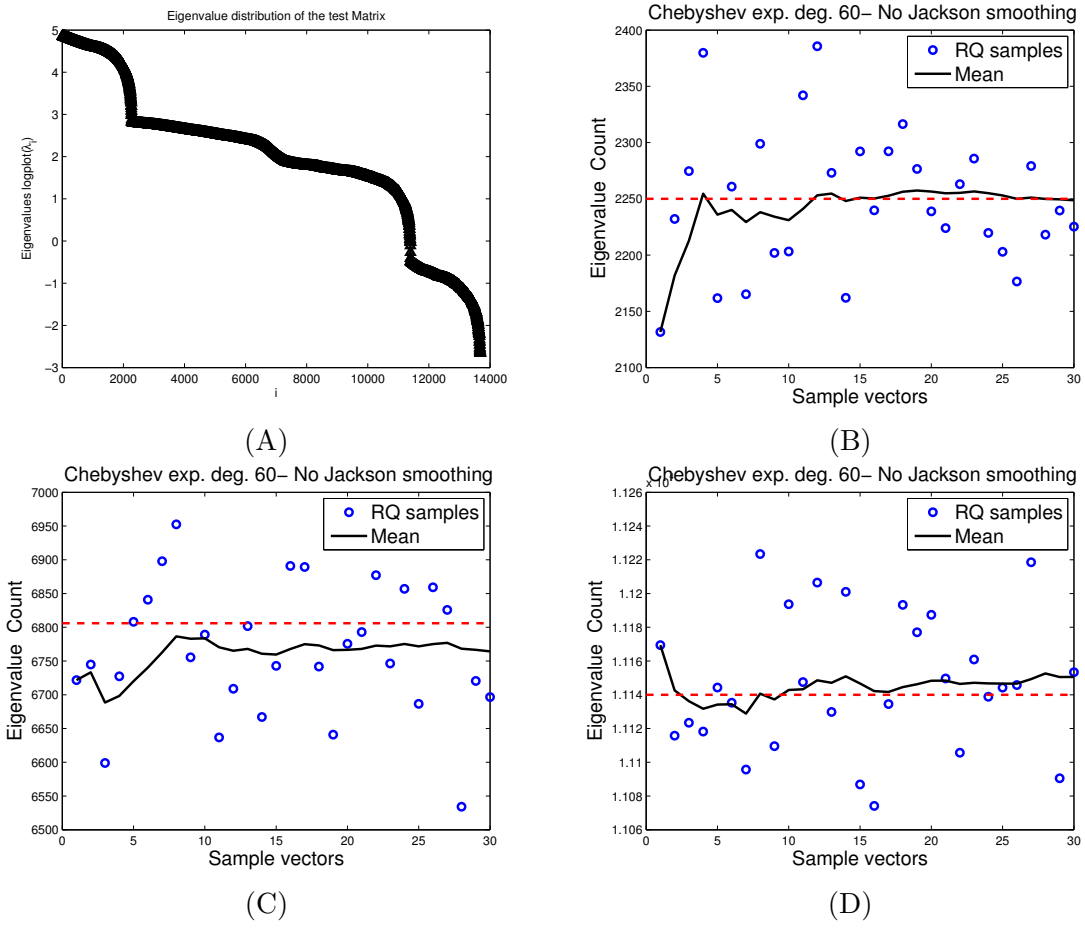


Figure 3.3: The eigenvalue distribution (logplot) of a 13681×13681 matrix TKK/cbuckle (UFL database) and Eigenvalue Counts for different ϵ.

Figure 3.3 illustrates how different choice of ϵ results in different approximate ranks.

The matrix (TKK/cbuckle) is from the UFL database[37] and is from Compressed cylindrical shell buckling. Size $13,681 \times 13,681$, and $nnz = 676,515$. It is structurely full rank but has three gaps (corresponding to eigenvalues $2000, 150$ and 10) in the eigenvalue spectrum. Figures 3.3(B,C,D) show the running average of the rank estimated using $\epsilon = 2000, 150$ and 10. The computed averages were $2250.5, 6754$ and 11156 for the respective ϵ. The actual counts were $2251, 6810$ and 11143. Because the spectrum is wide (Eigenvalues vary from 10^5 to 10^{-3}), a higher degree polynomial ($M = 60$) had to be used. Also Jackson damping overfits for this spectrum.

3.3 Density of States

The concept of *Density of States* (DoS) also known as the *spectral density* of Hermitian matrices are widely used in physics[30]. Density of States is a sort of probability density distribution that gives us a likelihood of finding the eigenvalues of a matrix near some points on the real line.

Given an $n \times n$ Hermitian matrix A, the Density of States (DoS), or spectral density is formally defined as

$$\phi(t) = \frac{1}{n} \sum_{j=1}^{n} \delta(t - \lambda_j) \tag{3.12}$$

where δ is the Dirac -function or Dirac distribution, and the λ_j are the eigenvalues of A, assumed here to be labeled increasingly. There are efficient algorithms for computing the DoS developed for many applications [38, 39]. The number of eigenvalues in an interval [a, b] can be expressed as

$$\eta_{[a,b]} = \int_a^b \sum_j \delta(t - \lambda_j) dt \equiv \int_a^b n\phi(t) dt \tag{3.13}$$

We can view $\phi(t)$ as a probability distribution function which gives the probability of finding eigenvalues of A in a given infinitesimal interval near t.

Thus, we can find an approximate rank of a matrix A by integrating the spectral density function $\phi(t)$ over the interval $[\epsilon, \lambda_{max}]$, as discussed in the previous section. That is,

$$r \approx \eta_{[\epsilon,\lambda_{max}]} = n \int_{\epsilon}^{\lambda_{max}} \phi(t) dt \tag{3.14}$$

The matrix A is assumed to be a large matrix with structurely low rank. This means there are several eigenvalues of A that are almost equal to zero. So, the idea is to choose ϵ to be a small number greater than zero, such that integration take into count only those eigenvalues which are large and contribute to the rank of the matrix. The Dirac $\delta-$function is not a proper function. It is more like a discretized version of a continous function and is difficult to approximate. So, it is replaced by smooth function of the form,

$$\phi_\sigma(t) = \frac{1}{n}\sum_{j=1}^{n} h_\sigma(t-\lambda_j) \tag{3.15}$$

where $h_\sigma(t)$ is any infinitely differentiable function which intergrates in $(-\infty,\infty)$ to one, and is zero or infinitesimally small outside a narrow interval $[-C_\sigma, C_\sigma]$ A good example is the Gaussian function:

$$h_\sigma(t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{t^2}{2\sigma^2}}$$

The idea is to find a smooth approximation to the spectral density function ϕ. This is in a sense, "blurring" the discrete function, or considering a continuous version of the distribution ϕ.

3.4 Kernel Polynomial Method

The Kernel Polynomial Method (KPM) was proposed by [40, 38] to find the DoS of a matrix. This method determines the exact DoS of a matrix as a sum of Dirac δ-functions by using the expansion in Chebyshev polynomials. The idea is to approximate the DoS function $\phi(t)$ by a finite expansion in the Chebyshev polynomials basis of the first kind, which are orthogonal polynomials. We consider the weighted function of the form,

$$\hat{\phi}(t) = \frac{1}{\sqrt{1-t^2}}\sum_{k=0}^{M} \mu_k T_k(t) \tag{3.16}$$

where the expansion coefficients μ_k are

$$\mu_k = \frac{2-\delta_{k0}}{n\pi}\sum_{j=1}^{n} T_k(\lambda_j) \tag{3.17}$$

Here, δ_{ij} is the Kronecker δ so that $2-\delta_{k0}$ is equal to 1 when $k=0$ and 2 otherwise. The coefficients μ_k is the trace of $T_k(A)$ with the scaling factor $(2-\delta_{k0})/(n\pi)$. And usually the problem of trace estimation, $\mathrm{Tr}(T_k(A))$ is solved using the stochastic argument (Hutchinson's unbiased estimator[33]) as we saw in equation (3.4). So, the trace of $T_k(A)$ is estimated as,

$$\mathrm{Tr}(T_k(A)) = \frac{n}{n_{vec}} \sum_{l=1}^{n_{vec}} (v_l)^T T_k(A) v_l \tag{3.18}$$

This method uses a large number of random vectors $v_1, v_2, \ldots, v_{n_{vec}}$. obtained from a normal distribution with mean zero and unit variance. Also these vectors are normalized such that, $\|v_l\| = 1, l = 1, \ldots, n_{vec}$. Then we have the coefficients of the polynomial to be,

$$\mu_k \approx \frac{2-\delta_{k0}}{\pi n_{vec}} \sum_{l=1}^{n_{vec}} (v_l)^T T_k(A) v_l \tag{3.19}$$

We saw in the subsection 3.2.1, that Chebyshev polynomials have the three term recurrence as given in (3.9) and also these polynomials are defined in the interval $[-1, 1]$. So, for a general matrix, we need to use the linear transformation mapping given in the equation (3.10).

The approximate rank of a matrix A is estimated by integrating the DoS function $\hat{\phi}(t)$ over the interval $[\epsilon, \lambda_{max}]$. As the linear mapping of the matrix, restrict the eigenvalues to be in the interval $[-1, 1]$, we need to find the linear mapping for $[\epsilon, \lambda_{max}]$ as $[\hat{\epsilon}, 1]$. Here $\hat{\epsilon}$ is slightly greater than -1 given by,

$$\hat{\epsilon} = \frac{\epsilon - c}{d}$$

where c and d are same as in the equation3.10. So, we find the rank by intergrating $\hat{\phi}(t)$ over this interval.

The approximate rank of a matrix using Kernel Polynomial Method is given by,

$$r \approx n \left(\int_{\hat{\epsilon}}^{1} \hat{\phi}(t) dt \right) = n \sum_{k=0}^{M} \mu_k \left(\int_{\hat{\epsilon}}^{1} \frac{T_k(t)}{\sqrt{1-t^2}} dt \right) \tag{3.20}$$

3.4.1 Integration and relation with the eigenvalue count method

Let us consider the integration term in the rank equation (3.20). This intergral is a function of the summation variable k. Lets call it, $\hat{\gamma}_k$ and for a general eigenvalue count, consider the limits of intergration over a general interval $[a, b]$. By the defination of Chebyshev polynomials, we have

$$\hat{\gamma}_k = \int_a^b \frac{T_k(t)}{\sqrt{1-t^2}} dt = \int_a^b \frac{\cos(k \cos^{-1}(t))}{\sqrt{1-t^2}} dt$$

Let $t = \cos\theta \Rightarrow dt = \sin\theta d\theta$. Then the integral becomes,

$$\begin{aligned} \hat{\gamma}_k &= \int_{a'}^{b'} \frac{\cos k\theta}{\sin\theta} \sin\theta d\theta \\ &= \int_{a'}^{b'} \cos k\theta d\theta \end{aligned}$$

where $a' = \cos^{-1}(a)$ and $b' = \cos^{-1}(b)$.

Case $k = 0$**:** We have $\cos k\theta = 1$ and the integral is equal to,

$$\hat{\gamma}_0 = \cos^{-1}(a) - \cos^{-1}(b)$$

Case $k > 0$**:** We have $\int \cos k\theta = \frac{\sin k\theta}{k}$. So, the integral is equal to,

$$\hat{\gamma}_k = \frac{\sin(k \cos^{-1}(a)) - \sin(k \cos^{-1}(b))}{k}$$

The equation (3.20) for the approximate rank using $\hat{\gamma}_k$ and expansion of the coefficients μ_k becomes,

$$r \approx n \sum_{k=0}^{M} \hat{\gamma}_k \mu_k = \frac{n}{n_{vec}} \sum_{k=0}^{M} \hat{\gamma}_k \frac{(2 - \delta_{k0})}{\pi} \left[\sum_{l=1}^{n_{vec}} (v_l)^T T_k(A) v_l \right] \tag{3.21}$$

So, if we set $\gamma_k = \hat{\gamma}_k \frac{(2-\delta_{k0})}{\pi}$, the approximate rank of a matrix using Kernel Polynomial Method becomes,

$$r \approx \frac{n}{n_{vec}} \sum_{l=1}^{n_{vec}} \left[\sum_{k=0}^{M} \gamma_k (v_l)^T T_k(A) v_l \right] \tag{3.22}$$

where the coefficients γ_k for an integration over the interval $[a, b]$ are given by,

$$\gamma_k = \begin{cases} \frac{1}{\pi}(\cos^{-1}(a) - \cos^{-1}(b)) & : k = 0, \\ \frac{2}{\pi}\left(\frac{\sin(k \cos^{-1}(a)) - \sin(k \cos^{-1}(b))}{k}\right) & : k > 0 \end{cases}$$

The above equation (3.21), is exactly the same equation that we obtained for the approximate rank of a matrix using the eigenvalue count method (eq:(3.8)) and the coefficients γ_k are same as the coefficients for the expansion of a step function in the interval $[a, b]$.

This is an interesting result because here in the KPM, we are integrating an approximation of the spectral density function and in the Eigenvalue Count method, we are evaluating the trace of the approximated Eigen projector and both methods result in the same equation.

This is because in the Eigenvalue count method, we treat the Eigen projector P as a step function and use Chebyshev polynomial approximation of the step function. And in KPM, we are approximating the Density of States function using Chebyshev polynomials which is a summation of delta functions. As we know, intergration of delta functions over an interval $[a, b]$ is equal to a step function over the interval. Thus, intergrating the Chebyshev polynomials that approximates the delta functions results in the same equation as approximating a step function using Chebyshev polynomials. Thus, we have two method which seem to be unrelated to each other, result in the same equation for the aproximate rank of a matrix.

The next section, decribes a second method that uses Lanczos Approximation to find the Spectral Density of a matrix and explain how this method can be used to find an approximate rank of the matrix.

3.5 Lanczos approximation to the spectral density

Since Lanczos algorithm gives good approximation for the extreme eigenvalues, we can use it find the count of extreme eigenvalues below the threshold ϵ. Then, the approximate rank will be n (the size of the matrix) minus this count. It is found that, Lanczos algorithm is a good candidate for computing local spectral densities, particularly at the end of the spectrum. [30] describes a method that combines Lanczos algorithm with multiple randomly generated starting vectors to get a good approximation to the DoS of a matrix. In this section, we use this method of Lanczos approximations to the spectral density to compute the count of extreme eigenvalues below the threshold ϵ.

The M-step Lanczos procedure for a real symmetric matrix A is given by the following equations [10],

$$AV_M = V_M T_M + f e_{M+1}^T \tag{3.23}$$

$$V_M^T V_M = I_M, \qquad V_M^T f = 0, \qquad V_M e_1 = v_0 \tag{3.24}$$

where T_M is an $(M+1) \times (M+1)$ tridiagonal matrix, V_M is an $n \times (M+1)$ orthogonal matrix and I_M is an $(M+1) \times (M+1)$ identity matrix.
The k-th column of V_M can be expressed as

$$V_M e_k = p_{k-1}(A) v_0, \ k = 1, \ldots, M+1$$

where $p_k(t), k = 0, 1, 2, \ldots, M$ is a set of orthogonal polynomials with respect to the weighted spectral distribution $\phi_{v_0}(t)$ given by,

$$\phi_{v_0}(t) = \sum_{j=1}^{n} \beta_j^2 \delta(t - \lambda_j) \tag{3.25}$$

where v_0 is expanded in the basis of the eigenvectors of A. [41] shows that these orthogonal polynomials can be generated by a three term recurrence whose coefficients are defined by the matrix entries of T_M.

Idea is to define the distribution function in terms of eigenpairs of the tridiagonal matrix T_M, obtained by Lanczos procedure. If $(\theta_k, y_k), k = 0, 1, 2, ..., M$ are eigenpairs of the tridiagonal matrix T_M, and τ_k is the first entry of y_k, then the distribution is,

$$\tilde{\phi}(t) = \sum_{k=0}^{M} \tau_k^2 \delta(t - \theta_k) \tag{3.26}$$

This serves as an approximation to the weighted spectral density function $\phi_{v_0}(t)$, in the sense,

$$\sum_{j=1}^{n} \beta_j^2 p_q(\lambda_j) = \sum_{k=0}^{M} \tau_k^2 p_q(\theta_j) \tag{3.27}$$

for all polynomials of degree $0 \leq q \leq M+1$. This equation is known as the moment matching property[42].

Since we are interested in finding the spectral density defined in (3.12), we choose inital vector v_0 such that $\beta_j^2 = 1/n$. But this not possible without knowing the eigenvectors u_j of A in advance. So, if we repeat the Lanczos process with multiple randomly genrated starting vectors $v_0^{(l)}, l = 1, 2, \ldots, n_{vec}$. Then if we consider the average, it indeed is a good approximation to the standard spectral density $\phi(t)$. Taking average over l, we have

$$\tilde{\phi}(t) = \frac{1}{n_{vec}} \sum_{l=1}^{nvec} \left(\frac{1}{n} \sum_{k=0}^{M} (\tau_k^{(l)})^2 \delta(t - \theta_k^{(l)}) \right) \tag{3.28}$$

should yield a good approximation to the standard spectral density. Because we have,

$$\frac{1}{n_{vec}} \sum_{l=1}^{nvec} (\beta_j^{(l)})^2 \approx 1/n$$

A refined method is to approximate cumulative spectral density or cumulative density of states (CDoS). Given by,

$$\tilde{\psi}(t) = \int_{-\infty}^{\infty} H(t-s)\tilde{\phi}(s)ds = \sum_{k=0}^{M} \eta_k^2 \delta(t - \theta_k) \tag{3.29}$$

where $\eta_k^2 = \sum_{i=1}^{k} \tau_i^2$ and θ_k and τ_k are the eigenvalues and the first components of the eigenvectors of the tridiagonal matrix T_M defined in (3.28). The idea for finding the rank is as follows. The cumulative spectral density is a cumulative sum of the probability distribution of the eigenvalues. This is equivalent to the cumulative density function which is the integration of a probablity density function and we know that, the rank of a matrix is integration of the spectral density. The coefficients η_k^2 is sum of τ_i^2's upto the eigenvalue θ_k. And from eq:(3.28) we see that τ_i^2s are the weights of the spectral density. Thus, for a matrix A, summing up η_k^2s corresponding to the θ_ks that lie in a given interval should give us the count of eigenvalues in that interval.

Thus, the approximate rank of a matrix, estimated using Lanczos Approximation method is,

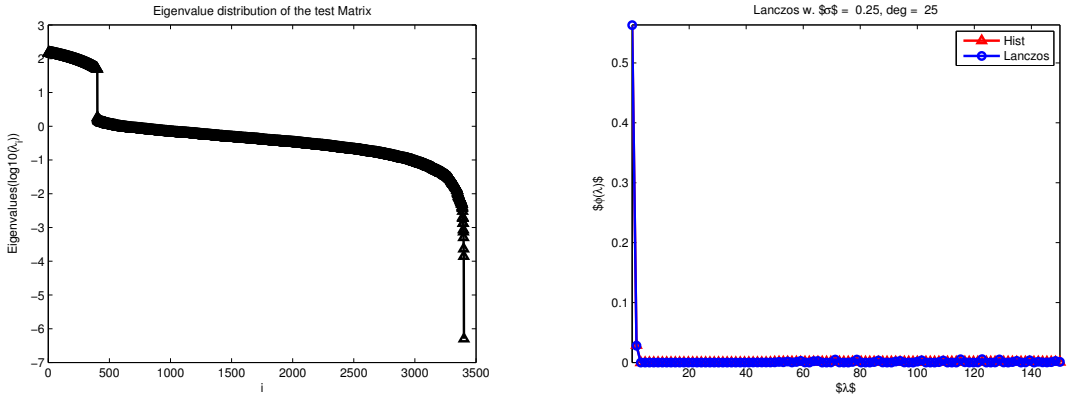
$$r = n - \frac{n}{n_{vec}} \sum_{l=1}^{nvec} \left(\sum_{k} (\eta_k^{(l)})^2 \right) \qquad \forall k : \lambda_{\min} \leq \theta_k < \epsilon \tag{3.30}$$

As discussed earlier, Lanczos yields good approximation for extreme eigenvalues. So, it makes sense to count the eigenvalues between $[\lambda_{\min}, \epsilon]$ and then subtract it from the size n. Also, the η_k^2s cummulatively sum up from $\lambda_{\min}$ to θ_ks. We can also consider η_k^2s $\forall k : \epsilon \leq \theta_k \leq \lambda_{\max}$. This should also yield the same rank.

To find the rank of a rectangular (or non hermtian) matrix in general, we find the rank of $A^T A$. For which, we need additional matrix-vector multiplication to apply A^T to the vector Av_0, in the Lanczos steps.

3.5.1 Results

To evaluate the performance of the Lanczos approximation technique, we use the same three matrices used in the Eigenvalue Count method.



(A) (B)

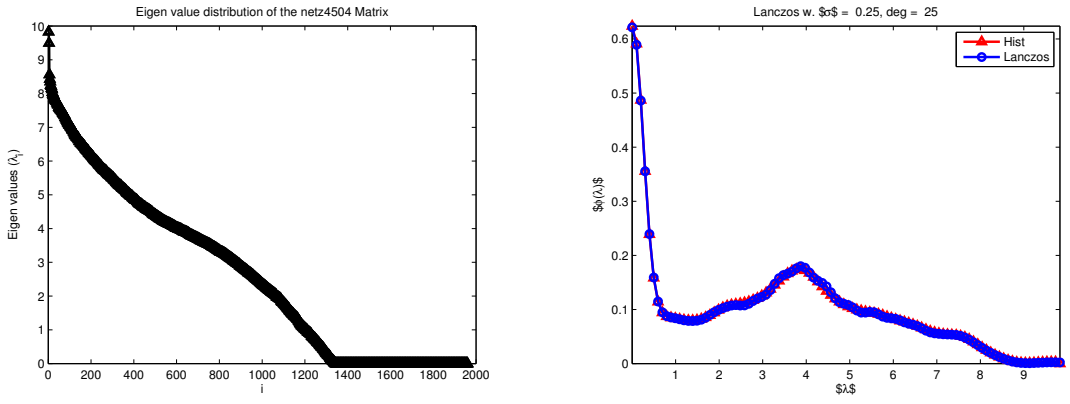
Figure 3.4: (A) The eigenvalue distribution of a 3401 × 3401 symmetric matrix . (B)Approximate DOS with 25 steps of Lanczos method and 20 random starting vectors.

In the first experiment, we used the same 3401 × 3401 symmetric matrix with the eigenvalue distribution shown in Fig. 3.4(A). The approximate rank is $r = 401$. Figure 3.4(B) shows the approximate DoS (in blue) obtained from 25 steps of Lanczos procedure. The red curve is the exact DoS. The 25 step Lanczos was repeat for 20 ($n_{vec} = 20$) random starting vectors and average over them was taken.

The predicted rank was $r = 400.0754$ for $\epsilon = 10$.That is, size minus the average of

sum of all η_k^2s between $[0, 10]$.

In the second example, we use the $1,961 \times 1,961$ matrix from AG-Monien group called netz4504, from the University of Florida matrix collection[37]. (The eigenvalue distribution in Fig3.4(A)). The approximate rank is $r = 1344$. Figure 3.4(B) shows the approximate DoS (in blue) obtained from 25 steps of Lanczos procedure with 20 random starting vectors. The estimated rank was $r = 1.3384e + 03$ for $\epsilon = 0.01$.



(A) (B)

Figure 3.5: (A) The eigenvalue distribution of a $1,961 \times 1,961$ matrix AG-Monien/netz4504. (B)Approximate DOS with 25 steps of Lanczos method and 20 random starting vectors.

To illustrates how different choice of ϵ results in different approximate ranks, we use the $13,681 \times 13,681$ matrix (TKK/cbuckle) is from the UFL databas.e and is from It is structurely full rank but has three gaps (corresponding to eigenvalues $2000, 150$ and 10) in the eigenvalue spectrum 3.3(A). The actual counts for $\epsilon = 2000, 150$ and 10 are $2,251, 6,810$ and 11143, repectively.

The ranks estimated by 40 steps Lanczos approximations are $r = 2.2460e + 03$ for $\epsilon = 2000$, $r = 6.8323e + 03$ for $\epsilon = 150$. and $r = 1.0112e + 04$ for $\epsilon = 10$.

We can see that, in all three examples, Lanczos Approximation method estimates the rank more accurately than Eigenvalue Count method. And also with fewer number of steps (equivalent to the degree of Chebyshev polynomial).

3.6 Applications

In this section, we discuss some potential applications where these approximate rank finding techniques could be useful. Some of the applications are as follows.

- In image processing and data compression, approximate rank of the data helps in reduction of the amount of data to be stored.
- In developing Fast Multipliers to multiply two matrices, an estimate of the approximate rank of matrices will help in faster multiplications.
- In many machine learning applications, knowing the rank of the matrix is important. Most of the fixed rank and relaxed rank optimization problem requires rank to be known in advance.

There are many algorithms that solve the rank minimization problem and a convex relaxation called, nuclear norm minimization problem that require an appproximate estimate of rank. There are many applications for the low rank approximations decribed in Chapter 2. These techniques require knowledge of the rank of a matrix.

3.6.1 Nuclear Norm Minimzation

One of the potential application for these techniques could be in nuclear norm minimzation problem.

The nuclear norm minimization is defined as follows,

$$minimize \qquad f(X) + \|X\|_* \tag{3.31}$$

Where $f(X)$ is a twice differentiable convex function of the unknown matrix X, $\lambda > 0$ is a regularization parameter and $\|X\|_* = \sum_{i=0}^{m} \sigma_i(X)$, is the nuclear norm.

For example, for Matrix Completion problem: Given a partially observed low rank matrix A with observed entries in Ω, we recover A solving,

$$f(X) = \frac{1}{2}\|\Pi_\Omega(X) - \Pi_\Omega(A)\|_F^2$$

where $(\Pi_\Omega(X))_{ij} = X_{ij}$ for all $(i,j) \in \Omega$ and 0 otherwise.

For Multivariate Regression: Given a data matrix A and label matrix B, we compute a model X as,

$$f(X) = \frac{1}{2}\|A(X) - B\|_F^2$$

It is know that, to solve these nuclear norm minimization problem, one can use an iteration of the form given below, which converges to the global minimum, if certain conditions are met by $f(X)$ [43].

$$X_{k+1} = S_\lambda(X_k - \alpha_k \nabla f(X_k))$$

where $S_\lambda(\hat{X}) = U(\Sigma - \lambda I)_+ V^T$, $a_+ = \max\{0, a\}$, for any $\hat{X} = U\Sigma V^T$ is the shrinkage operator and α_k is some step function.

Most algorithms that solve nuclear norm minimization problem [43, 44] find SVD (a large number of singular values) of the matrix, in each iteration and use the shrinkage operator to shrink the spectrum.

The idea is, we can use the approximate rank finding techniques to estimate the approximate number of singular values (say r_k) that lie above the threshold λ (by using $\epsilon = \lambda$) after certain number of iterations. Then instead of evaluating a large number of singular values (and their corresponding singular vectors) and then shrinking the spectrum (throwing away), we have to evaluate only the $r_k + p$ most significant singular values (and their corresponding singular vectors) at each iteration. Here p is a small oversampling. This could improve the computational cost of these algorithms, significantly.

Chapter 4

Matrices From Error Control Coding

4.1 Introduction

In the randomization technique for matrix decomposition which was reviewed in Chapter 2, we compute the matrix product $Y = A\Omega$, where Ω is an $n \times \ell$ random matrix. And compute an aproximate basis that spans the input matrix A[1], using Y. We saw in section 2.5 that, some of the structured random matrices like subsampled random Fourier Transform matrices, subsampled Hadamard transform matrices and others [17] can also be used in place a random Gaussian matrix.
Similarly in compressed sensing, we use a random Gaussian matrix as a measurement matrix to get the basis for sparse representation of the signals. $x = \Phi s$. It has been shown that, DFT matrix, random matrices with bernoulli distributions and some structured matrices like toeplitz structured [45] also perform very well and these matrices do satisfy the Restricted Isometric Property (RIP).

In both these techniques, the input signal (in case of compressed sensing) or the input matrix (in case of matrix decomposition) generally have very large dimensions (in $10^4 - 10^6$). This means the sampling Gaussian matrix, requires generating a large number of ($n \times \ell \approx 10^6 - 10^9$) random numbers which is a serious practical issue. (In terms of time complexity and storage).

In digital communications, we have plethora of codewords which are independent of each other[46]. If we stack up these codewords into a matrix, they will have near orthonormal columns, because each codewords have fixed minimum distance (Hamming Distance) between each other. A good idea would be to use these matrices formed from the codewords as sampling (measurement) matrix to find an approximate basis of the input matrix., or sample the input signal in compressed sensing.

In this chapter, we discuss the use of some of these matrices from Information theory and Error Correcting Codes for matrix decomposition and compressed sensing. Codewords generated using generator matrix (more in the next section) from linear codes (random generator matrices) and dual BCH codes are used in place of random Gaussian matrices. In the following sections we will see that, these matrices from Error Correction codes require generation of far fewer number of random numbers than Gaussian matrices or other structured random matrices. Interestingly, performance of these matrices in both matrix decomposition schemes and compressed sensing, is comparable to that of the random Gaussian matrix .

4.2 Error Control Coding

In communication systems, data are transmitted from a source (transmitter) to a destination (receiver) through physical channels, which are usually noisy. Thus, errors are bound to occur time to time. So, in order to facilitate the receiver the ability to detect and correct these errors, error control coding methods are used [47]. Error control coding aims at developing methods for coding to check the correctness of the bit stream transmitted. The bit stream representation of a symbol is called the codeword of that symbol.

There are different types of error control coding mechanisms like (linear) block codes, cyclic codes, convolution codes etc. A code is linear if two codes are added using modulo-2 arithmetic produces a third codeword in the code. Usually it is represented as a (ℓ, k) linear block code. Here, ℓ represents the codeword length, k is the number of message bit and $\ell - k$ bits are error control bits or parity check bits generated from message

using an appropriate rule. We may therefore represent the codeword as

$$c_i = \begin{cases} b_i, & i = 0, 1, \ldots, \ell - k - 1 \\ m_{i+k-\ell}, & i = \ell - k, \ell - k + 1, \ldots, \ell - 1 \end{cases} \tag{4.1}$$

The $(\ell - k)$ parity bits b_is are linear sums of the k message bits m_is.

$$b_i = p_{0i}m_0 + p_{1i}m_1 + \cdots + p_{k-1,i}m_{k-1} \tag{4.2}$$

where the coefficients are

$$p_{ij} = \begin{cases} 1, & if\ b_i \text{ depends on } m_j \\ 0, & otherwise \end{cases} \tag{4.3}$$

If we represent these in vector form, we have $1 \times k$ message vector $\mathbf{m} = [m_0, m_1, \ldots, m_{k-1}]$, $1 \times (\ell - k)$ parity vector $\mathbf{b} = [b_0, b_1, \ldots, b_{\ell-k-1}]$ and $1 \times \ell$ code vector $\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}]$.

We may thus write simultaneous equations in matrix equation form as

$$\mathbf{b} = \mathbf{mP} \tag{4.4}$$

where P is a $k \times (\ell - k)$ matrix defined by,

$$P = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots & p_{0,\ell-k-1} \\ p_{10} & p_{11} & p_{12} & \cdots & p_{1,\ell-k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{k-1,0} & p_{k-1,1} & p_{k-1,2} & \cdots & p_{k-1,\ell-k-1} \end{bmatrix} \tag{4.5}$$

The codevector c, can be expressed as a partitioned row vector in terms of the vectors m and b as follows

$$c = [b : m] = m[P : I_k]$$

where I_k is a $k \times k$ identity matrix. Now if we define a $k \times \ell$ generator matrix G as,

$$G = [P : I_k] \implies c = mG$$

For different types of error control codes, we have different generator matrices with different properties. We are considering finite field arithmetic, so we use binary multiplication. For certain kind of error control codes, we can generate codes for any

combinations of (ℓ, k) while some error control codes are restricted.

For a chosen k bits, we can have 2^k unique messages. This means we should have 2^k unique codewords. Thus, we can form a sampling matrix of the size $2^k \times \ell$ by stacking up all possible combination of codevectors from a generator matrix of any linear coding scheme. This matrix will have ℓ near orthonormal columns.

$$\underbrace{C}_{2^k \times \ell} = \underbrace{M}_{2^k \times k} \underbrace{G}_{k \times \ell} \tag{4.6}$$

And use this matrix of codewords as the sampling matrix (random subspace) in the randomization techniques for matrix decomposition and in compressed sensing.

In a given application, if the dimension $n < 2^k$, we simply have to throw away $2^k - n$ rows. That is, we use only n out of 2^k possible codewords. Columns will remain near orthonormal as long as $2^{k-1} < n \leq 2^k$ because every codeword is independent of each other and will still maintain a minimum hamming distance between each other.

In the following sections, we describe two coding techniques that can be used to form the codeword matrix C using two different type of generator matrix G and show, how these codeword matrices can be used for matrix decomposition and compressed sensing problem and discuss their performance.

4.3 Random Generator Matrix

In coding theory, a linear code is an error-correcting code for which any linear combination of codewords is also a codeword. Here we use a random generator matrix to generate linear codes. That is, use a $k \times \ell$ random matrix with random 1s and 0s with independent columns as the generator matrix G in the equation (4.6). All 2^k combinations of the message vectors are generated (That is, $2^k \times k$ message matrix M) and multiplied, by the random generator matrix G, using binary multiplication to get 2^k codewords of length ℓ (That is, $2^k \times \ell$ codeword matrix C).

The codeword matrix C has entries 1s and 0s. We need to transform this to a near orthonormal matrix. We do that, by first mapping 1s to -1s and 0s to 1s. This is a

very common practice, when digital signals are needed to be transmitted over analog channels. And then scale the matrix (each entry) by $1/\sqrt{\ell}$, such that the matrix has unit energy. Now, this matrix whose rows are the codewords from this linear coding technique is used as the random matrix for sampling the given input matrix and use it to find the active subspaces of the matrix , in the randomized techniques for matrix decomposition. This linear code matrix (ie,. the partial random matrix) can also be used for compressed sensing.

The advantage of using this codeword matrix is that, we are reducing randomness from $n \times \ell$ to $\log_2(n) \times \ell$. Thus, exponentially reducing the number of random number to be generated. Also, because entries of the codeword matrix C are only 1 and -1, we can treat the codeword matrix as a bernoulli distributed matrix and the corresponding theory for performance analysis, can we applied.

4.4 Dual BCH Code Generator Matrix

In coding theory, the BCH codes form a class of cyclic error-correcting codes that are constructed using finite fields, developed by Bose, Chaudhuri and Hocquenghem[48]. For cyclic codes, any cyclic shift of a codeword in the code is also a codeword. BCH codes are cyclic codes over $GF(q)$ (the channel alphabet) that are defined by a $(d-1)\times\ell$ check matrix over $GF(q^p)$, where GF stands for the *Galois Field* or *Finite Field*, ℓ is the length of the code or blocklength, p is the number of parity bits, and d is an integer with $2 \leq d \leq \ell$.

A primitive BCH code is a BCH code defined using a primitive element α. If α is a primitive element of $GF(q^p)$, then the blocklength is $\ell = q^p - 1$ This is the maximum possible blocklength for decoder alphabet $GF(q^p)$. The parity-check matrix for a t-error-correcting primitive BCH code is of the form,

$$\begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{(\ell-1)} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(\ell-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t} & \alpha^{4t} & \cdots & \alpha^{2t(\ell-1)} \end{bmatrix}$$

where $\ell = q^p - 1$, t must be $\ell - k$ and α is an ℓ-th root of unity in $GF(q^p)$. The generator matrix formed using this parity check matrix as discussed in section 4.2 and the codewords are generated.

The dual of a BCH code is a code [49] with same blocklength ℓ, but the codes are generated using the parity check matrix (defined in section 4.2) of the BCH code as the generator matrix. So, for any blocklength ℓ and t number of errors to be corrected, we can generate 2^t dual BCH codewords of length ℓ using the generator matrix of the dual BCH code. So, we choose t to be k.

The idea is similar to the linear code technique. We use a $k \times \ell$ dual BCH generator matrix G. Generate all 2^k combinations of the message vectors(That is, $2^k \times k$ message matrix M) and multiply, by the dual BCH generator matrix G, using binary multiplication to form 2^k dual BCH codewords of length ℓ (That is, $2^k \times \ell$ codeword matrix C). Then ,this C matrix whose rows are the codewords of dual BCH code is used as the random matrix for sampling the given input matrix to find the active subspaces of the matrix.

An important point now is, this dual BCH code matrix is a deterministic matrix. So, we cannot apply the probabilistic arguments used in randomized algorithms to analyze the performance. So, these matrices are not interesting from the theoretical point of view. But, things get interesting if the input matrix A can be treated as random. For example in recommender systems, the people's ratings can be treated as random. Then, we can apply the similar probabilistic arguments (discussed in Chapter 2) to analyze the performance of dual BCH code matrices for the decomposition of these random input matrices.

In the next section, we discuss the performance of these codeword matrices, when used in randomization techniques for matrix decomposition and compressed sensing.

4.5 Results

4.5.1 Matrix Decomposition

To illustrate the performance of the codeword matrices as sampling matrices in randomized algorithm for matrix decomposition, we use a 4772×4772 matrix named EPA from Pajek network (Matrix is from a directed graph's matrix representation), available in the UFL Sparse Matrix Collection[37]. $nnz = 8,965$ and the structure rank is $r = 951$. There is a nice gap in the singular values where the value falls from 0.76 to $2.113 \times e^{-14}$. Figure 4.1(A) gives the singular value distribution of this matrix.

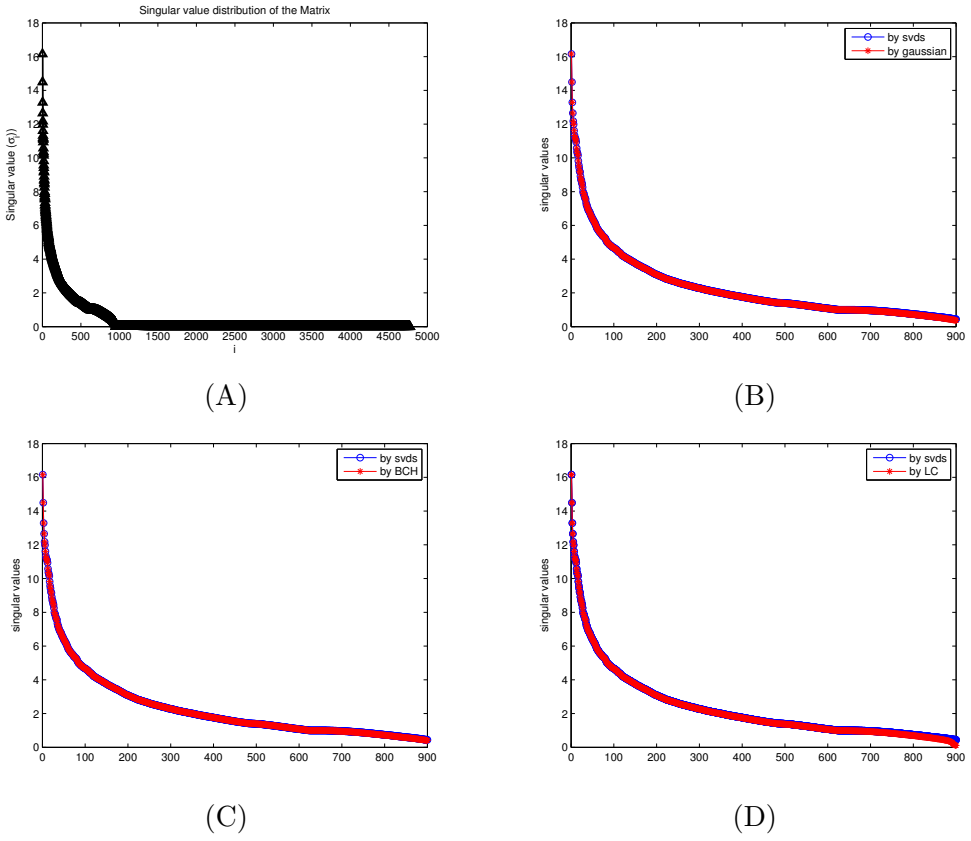
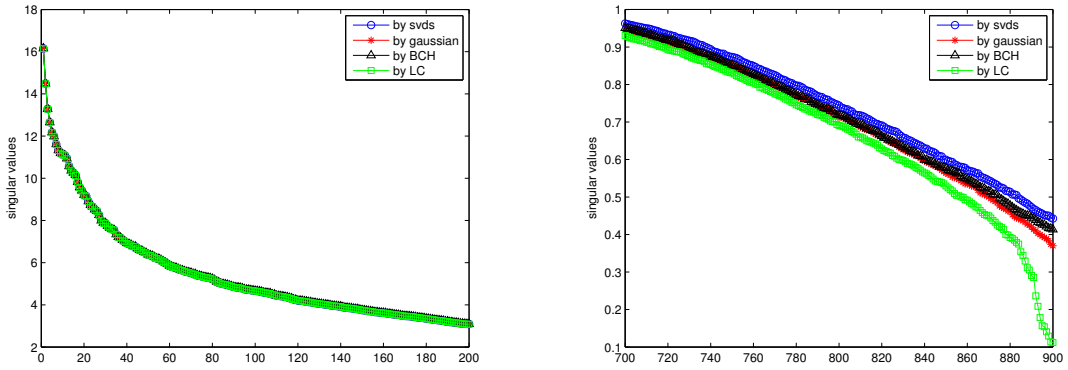


Figure 4.1: (A)The singular value distribution of a 4772×4772 matrix EPA (UFL database). (B),(C)and(D) Top 900 singular values computed by randomized techniques using Gaussian, dual BCH codeword and linear codeword matrices, respectively

The proto-type algorithm (Algorithm 1) described in chapter 2, was used to find the top 900 singular values. $k = 900$ and oversampling $p = 20$. Figure 4.1(B),(C)and(D) show top 900 singular values computed by randomized techniques using Gaussian, dual BCH codeword and linear codeword matrices, respectively. The singular values computed (in red) using each of these matrices are compared against the actual singular value (in blue) computed by using "svds" function in matlab. (These singular values are made available in the UFL database). We can see that, the performance of the codeword matrices is similar to that of Gaussian matrix.



(A) (B)

Figure 4.2: (A) The top 200 singular values (B)The last 200 singular values (out of 900) computed using the three matrices as sampling matrix.

To further analyze the performance, lets look at the top 200 and bottom 200 singular values computed. Figure 4.2(A) shows the top 200 singular values computed using the three sampling matrices, namely Gaussian, dual BCH codeword and linear codeword matrices along wth the actual singular values. All three matrices perform very well. This comes as no surprise because, the top singular values converge much faster than the bottom ones, as k is increase.

Figure 4.2(B) shows the bottom 200 singular values (from 701 to 900) computed using the three sampling matrices along with the actual singular values. Here we see that, the singular values computed using Gaussian and dual BCH codeword matrices

have almost converged to the actual values with just 20 oversamplings. But we see that, last few singular values computed using the linear code matrices are quite away from (the actual values) convergence. If we increase the oversampling to say $p = 50$, these converge too. This is due to the fact that, we are using a random generator matrix and this might not result in all columns of the codeword matrix to be linearly independent of each other. Nevertheless, the performance is comparable to a random Gaussian matrix.

The mean squared error in singular values obtained using Gaussian matrix was, $e_\ell = 0.0129$, using dual BCH codeword matrix $e_\ell = 0.0147$ and using linear codeword matrix $e_\ell = 0.0384$. So, the errors for all three matrices are comparable.

4.5.2 Compressed Sensing

The codeword matrices were used as measurement matrix in compressed sensing to sample the input signal. These codeword matrices are expected to satisfy the Restricted Isometric Property because the collumns of the matrix are near orthonormal, so the coherence of the matrix must be low. The signal was recovered using ℓ_1-norm minimization as given in eq:2.23.

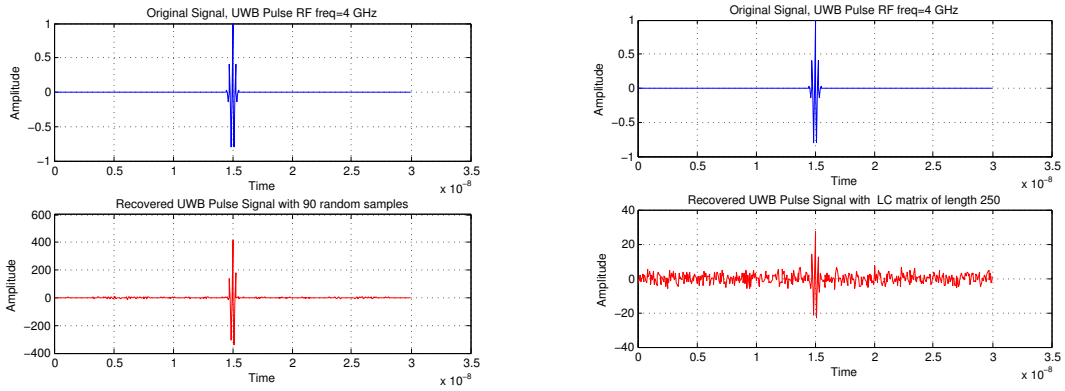
Noiseless Signal:

In the first experiment, for a basic compressed sensing example, we try recover a noise free k-sparse signal of dimension $N = 1500$ with $k = 10$. We compare the performance of the codeword matrices against a Bernoulli distributed random matrix as measurement matrix for compressed sampling of the signal. Since the codeword matrices have only 1s and -1s, it's only fair to compare its performance against Bernoulli distributed random matrix with entries 1s and -1s.

Figure 4.3(A) shows the input signal (on top) and the recovered signal (bottom) using a Bernoulli distributed random Matrix with $m = 90$ samples. That is, using a 90×1500 Bernoulli random matrix. The signal was recovered using ℓ_1-norm minimization. We can see that, the signal recovery is almost exact using just 90 samples.

The Mean Squared Error (MSE) of the recovered signal using this Bernoulli distributed matrix, with $m = 90$ samples was $e_m = 2 \times e^{-10}$. Interestingly, this is better

than the MSE we got while using a Gaussian random matrix. The MSE for Gaussian random matrix, with $m = 90$ samples was $e_m = 4.5 \times e^{-4}$.



(A) (B)

Figure 4.3: Basic compressed sensing example with k-sparse signal of dimension $N = 1500$ with $k = 10$. (A) Actual signal on top and Recovered signal using Bernoulli distributed random Matrix with 90 samples in the bottom (B) Recovery using linear codeword Matrix with 250 samples

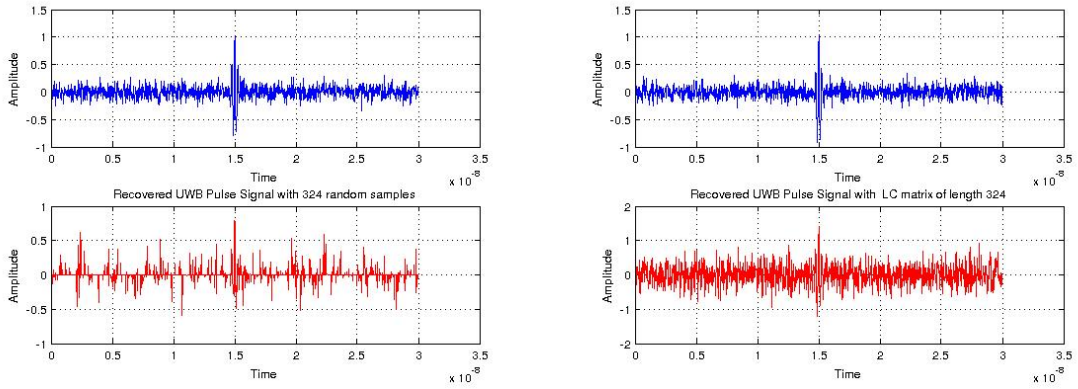
Figure 4.3(B) shows the input signal (on top) and the recovered signal (bottom) using a linear codeword Matrix with $m = 250$ samples. That is, using a 250×1500 linear codeword matrix. The signal was recovered using ℓ_1-norm minimization. We can see that, the signal recovery is not good even with 250 samples, while the random matrices recovered the signal exactly with just 90 samples. Obviously, as we increase the number of samples, the recovery is better. But compared to a completely random matrices, the performance of codeword matrices was poor.

The Mean Squared Error (MSE) of the recovered signal using the linear codeword matrix, with $m = 250$ samples was $e_m = 0.045$. The MSE of the recovered signal using the dual BCH codeword matrix, with the same $m = 250$ samples was $e_m = 0.0367$.

Noisy Signal:

We saw in the previous experiment that, the codeword matrices perform poorly in

recovering a noise free input signal. In the second experiment, we try to recover a noisy input signal by compressed sampling. The input signal is same as in the first experiment, a sparse signal with dimension $N = 1500$ and $k = 10$. A small amount of noise is added to this input signal (Gaussian noise with $\sigma = 0.25$). The original signal is recovered by sampling this noisy signal using Bernoulli distributed matrix and codeword matrices.



(A) (B)

Figure 4.4: Noisy ($\sigma = 0.25$) signal of $N = 1500$ and 10 nonzero (A) Recovery using Bernoulli distributed random Matrix with 324 samples (B) Recovery using linear Code Matrix with 324 samples

Figure 4.4(A) shows the input signal plus Gaussian noise with $\sigma = 0.25$ (on top) and the recovered signal (bottom) using a Bernoulli distributed random Matrix with $m = 324$ samples. That is, using a 324×1500 Bernoulli random matrix. The signal was recovered using ℓ_1-norm minimization. The Mean Squared Error (MSE) of the recovered signal (from a noisy input signal) using this Bernoulli distributed matrix, with $m = 324$ samples was $e_m = 0.013$.

Figure 4.4(B) shows the input noisy signal (on top) and the recovered signal (bottom) using a linear codeword Matrix with $m = 324$ samples. The signal was recovered using ℓ_1-norm minimization. The Mean Squared Error (MSE) of the recovered signal using the linear codeword matrix, with $m = 324$ samples was $e_m = 0.0490$. We can see that, while recovering the noisy signal, the performance of the codeword matrices is comparable to

that of completely random matrix. Also for the codeword matrices, the error rates for the recovered signals are similar for both noiseless and noisy signals. But in both cases, a completely random matrix outperform the codeword matrices in recovering signals using compressed sampling.

Chapter 5

Conclusion and Discussion

In Chapter 2, we reviewed some of the randomization algorithms for matrix decomposition and the concept of Compressed Sensing. An important take away with the review is that, the randomization algorithms generally work well, if there is a gap in the singular values spectrum and the singular values decay rapidly beyond the chosen rank k. If the singular values have slow decay, and the size is very large, the methods lose all the accuracy. There are a few post processing techniques suggested in [1] to help improve the performance. Also, one needs to know the value of the rank k a priori. This was handled in Chapter 3, where we proposed two inexpensive techniques to find an approximate rank of the matrix.

The techniques to find the approximate rank of a matrix disccussed in Chapter 3 are fairly accurate and computationally inpexpensive. Since these methods are based on approximating some function of a spectral projector or Density of States, these are indirect estimate of the spectrum of A. The approximation are based on stochastic sampling and averaging. So, the methods could be biased if the aprroximations are not accurate. An important point is that, the performance is totally dependent on the choice of the threshold ϵ and the degree of approximation M. Higher the degree we use, better is the approximation but more expensive, is the method. And the bias gets exacerbated if there are clusters near the choice of ϵ, as the inaccuracies of the approximation is large near the boundaries.

The Lanczos approximation, is a direct estimate of the spectrum. Stochastic sampling of the starting vector improves the approximation significantly. However, if the number of Lanczos steps used is sufficiently large, even one starting vector may be sufficient to fairly estimate the rank for a given choice of ϵ.

In Chapter 4, we discussed application of matrices from error control coding, in randomized algorithms for matrix decomposition and compressed sensing. The performance of these matrices were similar to that of random Gaussian matrix or other structured random matrices for matrix decomposition. But, the performance as measurement matrix in compressed sensing was not as good. One of the clear advantages of using these matrices is that, we can generate these matrices for any arbitarily large size, inexpensively. Also, these matrices reduce the randomness (number of random numberd to be generated) significantly. But theoretical analysis of their performance (error bounds) for matrix decomposition and proof that they satisfy, the restricted isometric property are still open.

References

[1] N. Halko, P. Martinsson, and J. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011, http://dx.doi.org/10.1137/090771806.

[2] Mark A. Davenport, Marco F. Duarte, YC Eldar, and G Kutyniok. Introduction to compressed sensing. In *Compressed Sensing: Theory and Applications*. Cambridge University Press, 2011.

[3] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[4] Jieping Ye. Generalized low rank approximations of matrices. *Machine Learning*, 61(1-3):167–191, 2005.

[5] Dimitris Achlioptas and Frank Mcsherry. Fast computation of low-rank matrix approximations. *Journal of the ACM (JACM)*, 54(2):9, 2007.

[6] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[7] Per-Gunnar Martinsson, Vladimir Rockhlin, and Mark Tygert. A randomized algorithm for the approximation of matrices. Technical report, DTIC Document, 2006.

[8] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[9] Gene H. Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[10] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[11] Peter Businger and Gene H. Golub. Linear least squares solutions by householder transformations. *Numerische Mathematik*, 7(3):269–276, 1965.

[12] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics*, 11(1):50–59, 1960.

[13] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1100–1124, 2009.

[14] Per-Gunnar Martinsson, Arthur Szlam, and Mark Tygert. Normalized power iterations for the computation of SVD. *Manuscript., Nov*, 2010.

[15] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.

[16] Nir Ailon and Bernard Chazelle. Faster dimension reduction. *Communications of the ACM*, 53(2):97–104, 2010.

[17] Edo Liberty. *Accelerated Dense Random Projections*. PhD thesis, Yale University, 2009.

[18] Vladimir Rokhlin and Mark Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Sciences*, 105(36):13212–13217, 2008.

[19] Emmanuel J Candès. Compressive sampling. In *Proceedings oh the International Congress of Mathematicians: Madrid, August 22-30, 2006: invited lectures*, pages 1433–1452, 2006.

[20] Stéphane Mallat. *A wavelet tour of signal processing*. Academic press, 1999.

[21] David L Donoho and Michael Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via l-1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.

[22] Richard G. Baraniuk. Compressive sensing. 2007.

[23] Emmanuel J Candes and Terence Tao. Decoding by linear programming. *Information Theory, IEEE Transactions on*, 51(12):4203–4215, 2005.

[24] Emmanuel J Candes. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique*, 346(9):589–592, 2008.

[25] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

[26] Ivan Markovsky. Structured low-rank approximation and its applications. *Automatica*, 44(4):891–909, 2008.

[27] Troy Lee and Adi Shraibman. An approximation algorithm for approximation rank. In *Computational Complexity, 2009. CCC'09. 24th Annual IEEE Conference on*, pages 351–357. IEEE, 2009.

[28] Noga Alon, Troy Lee, Adi Shraibman, and Santosh Vempala. The approximate rank of a matrix and its algorithmic applications: approximate rank. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 675–684. ACM, 2013.

[29] Edoardo Di Napoli, Eric Polizzi, and Yousef Saad. Efficient estimation of eigenvalue counts in an interval. *arXiv preprint arXiv:1308.4275*, 2013.

[30] Lin Lin, Yousef Saad, and Chao Yang. Approximating spectral densities of large matrices. *arXiv preprint arXiv:1308.5467*, 2013.

[31] Eric Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Physical Review B*, 79(11):115112, 2009.

[32] Grady Schofield, James R Chelikowsky, and Yousef Saad. A spectrum slicing method for the Kohn–Sham problem. *Computer Physics Communications*, 183(3):497–505, 2012.

[33] Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

[34] Jok M Tang and Yousef Saad. A probing method for computing the diagonal of a matrix inverse. *Numerical Linear Algebra with Applications*, 19(3):485–501, 2012.

[35] Laurent O Jay, Hanchul Kim, Yousef Saad, and James R Chelikowsky. Electronic structure calculations for plane-wave codes without diagonalization.

[36] C Bekas, E Kokiopoulou, and Yousef Saad. An estimator for the diagonal of a matrix. *Applied numerical mathematics*, 57(11):1214–1229, 2007.

[37] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[38] Lin-Wang Wang. Calculating the density of states and optical-absorption spectra of large quantum systems by the plane-wave moments method. *Physical Review B*, 49(15):10154, 1994.

[39] Chen Huang, Arthur F Voter, and Danny Perez. Scalable kernel polynomial method for calculating transition rates. *Physical Review B*, 87(21):214106, 2013.

[40] RN Silver and H Röder. Densities of states of mega-dimensional Hamiltonian matrices. *International Journal of Modern Physics C*, 5(04):735–753, 1994.

[41] Walter Gautschi. Computational aspects of three-term recurrence relations. *SIAM review*, 9(1):24–82, 1967.

[42] Walter Gautschi. A survey of Gauss-Christoffel quadrature formulae. In *EB Christoffel*, pages 72–147. Springer, 1981.

[43] Cho-Jui Hsieh and Peder Olsen. Nuclear norm minimization via active subspace selection. In *Proceedings of The 31st International Conference on Machine Learning*, pages 575–583, 2014.

[44] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research*, 11:2287–2322, 2010.

[45] Waheed U Bajwa, Jarvis D Haupt, Gil M Raz, Stephen J Wright, and Robert D Nowak. Toeplitz-structured compressed sensing matrices. In *Statistical Signal Processing, 2007. SSP'07. IEEE/SP 14th Workshop on*, pages 294–298. IEEE, 2007.

[46] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[47] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[48] Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and control*, 3(1):68–79, 1960.

[49] José Felipe Voloch. On the duals of binary BCH codes.