Faster Streaming Algorithms for Low-Rank Matrix Approximations

by

Timothy Matthew Galvin

S.B., Massachusetts Institute of Technology (2013)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Signature redacted

Author ....

Department of Electrical Engineering and Computer Science

May 23, 2014

Certified by....... Signature redacted ....

Dr. Christopher Yu

Division Leader, Embedded Navigation and Sensor Systems, Draper Laboratory

Thesis Supervisor

Signature redacted

Certified by..

Prof. Piotr Indyk

Professor, MIT

Thesis Supervisor

Signature redacted

Accepted by ....

Prof. Albert R. Meyer

Chairman, Master of Engineering Thesis Committee

Faster Streaming Algorithms for Low-Rank Matrix Approximations

by

Timothy Matthew Galvin

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2014, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Low-rank matrix approximations are used in a significant number of applications. We present new algorithms for generating such approximations in a streaming fashion that expand upon recently discovered matrix sketching techniques. We test our approaches on real and synthetic data to explore runtime and accuracy performance. We apply our algorithms to the technique of Latent Semantic Indexing on a widely studied data set. We find our algorithms provide strong empirical results.

Thesis Supervisor: Dr. Christopher Yu
Title: Division Leader, Embedded Navigation and Sensor Systems, Draper Laboratory

Thesis Supervisor: Prof. Piotr Indyk
Title: Professor, MIT

Acknowledgments

First and foremost, I would like to thank my advisors Dr. Christopher Yu and Professor Piotr Indyk for their advice, guidance, and general mentorship throughout my senior year and graduate year at MIT. Their flexibility and support was invaluable during my studies.

I want to thank Draper Laboratory for generously funding my tuition and providing me a collaborative environment in which to work. I want to thank the MIT VI-A program for connecting me with Draper and for the amazing opportunities that came as a result.

Finally, I want to thank my parents and two brothers for providing me with endless support and motivation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis we present new algorithms for creating low-rank approximations of streaming large data matrices and empirically show their effectiveness on real-world data sets. We present the previous work in the field and demonstrate how recent approaches have led to the creation of our algorithms. Our techniques are compared to state-of-the-art methods as well as a brute-force approach on synthetic and real-world data sets.

1.1 Motivation

There are a variety of applications that utilize the collection and analysis of extremely large data sets. Recommendation systems build matrices that contain each user's preference for each item [28]. Image analysis techniques transform individual frames into a vector representation and append the results to form a matrix capturing a series of images [12]. Text documents are processed using a term-document matrix, in which rows and columns represent terms and documents, respectively [6]. These large data sets, which can easily contain billions of entries, are often processed for model-fitting, clustering, noise reduction, etcetera [7]. Two issues that frequently arise from these matrices and their analysis are the high storage requirements and computation cost. Low-rank approximations mitigate these resource limitations at the cost of precision. The design of approximation methods revolves around the trade-offs that can be made between space, accuracy, and runtime [15]. In this thesis we present matrix approximation algorithms that reduces the computation cost of previous work that targeted space-optimality at the expense of runtime. We believe our approach provides a more well-rounded implementation for data processing.

1.2 Outline of the Thesis

In Chapter 2 we present the previous work related to low-rank matrix approximations. In Chapter 3 we present our streaming algorithms for deterministically generating low-rank matrix approximations. In Chapter 4 we compare the computational efficiency and approximation accuracy of the different algorithms. In Chapter 5 we apply our algorithms to Latent Semantic Indexing and compare our approximation approach to a best rank-k baseline. In Chapter 6 we conclude with a summary of findings and areas of open future research.

Chapter 2

Previous Work

In this chapter, we summarize past research related to creating and maintaining low-rank matrix approximations. In Section 2.1 we provide an overview of the Singular Value Decomposition. In Section 2.2 we highlight the rank-k Singular Value Decomposition and summarize its benefits. In Section 2.3 we discuss the problem of incorporating new data into previously calculated decompositions. In Section 2.4 and Section 2.5 we cover previous work in generating low-rank matrix approximations.

2.1 Singular Value Decomposition

Given a matrix $A \in \mathbb{R}^{n \times m}$ with $n \geq m$, the singular value decomposition (SVD) of A is:

$$A = U\Sigma V^T = \sum_i u_i \sigma_i v_i^T \tag{2.1}$$

where U and V are orthogonal matrices whose columns are the left and right singular vectors of A, respectively, and Σ is a diagonal matrix containing the singular values of A. The corresponding individual elements of those matrices, u_i, σ_i, and v_i, are the singular triplets of A. The matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ also specify the eigenvectors of A^TA and AA^T, respectively. Σ is structured as $\begin{bmatrix}\text{diag}(\sigma_1, & \cdots, & \sigma_m) & \mathbf{0}\end{bmatrix}$ where $\sigma_1 \geq \cdots \geq \sigma_m \geq 0$ and $\sigma_1, \cdots, \sigma_m$ are the nonnegative square roots of the eigenvalues of A^TA and AA^T [16].

Calculating the full singular value decomposition of a matrix $A \in \mathbb{R}^{n \times m}$ is usually computed using one of two methods. The first approach uses the QR decomposition, in which Q is an orthogonal matrix and R is an upper triangular matrix:

$$A = QR = Q(\check{U}\Sigma V^T) = (Q\check{U})\Sigma V^T \doteq U\Sigma V^T \tag{2.2}$$

The second method leverages the relationship between the singular vectors of A and the eigenvectors of A^TA and AA^T:

$$A^TA = V\Sigma^2V^T \quad A = (AV\Sigma^{-1})\Sigma V^T \doteq U\Sigma V^T \tag{2.3}$$

$$AA^T = U\Sigma^2U^T \quad A = U\Sigma(U^TA\Sigma^{-1})^T \doteq U\Sigma V^T \tag{2.4}$$

The singular value decomposition is frequently used in data analysis as it reveals structural information about the matrix. It reorganizes the data into subspaces that have the most variation, making it useful in applications such as image recognition, information retrieval, and signal processing [29].

A major drawback of the SVD is its computation complexity. Given a matrix $A \in \mathbb{R}^{n \times m}$ with $n \geq m$, calculating the SVD(A) takes $\mathrm{O}(nm^2)$ time. As the decomposition scales cubically, it quickly becomes infeasible for large matrices.

2.2 Rank-k Singular Value Decomposition

The singular value decomposition provides another useful property due to the subspace reordering mentioned above. The best rank-k approximation to A is easily obtained from the SVD of A. By throwing away the bottom $n - k$ singular triplets, we are left with a product that yields A_k as seen in Figure 2-1. More explicitly, A_k is the product of the first k columns of U, the top k values of Σ, and the first k rows of V^T:

$$A_k = \sum_i^k \mathbf{u}_i\sigma_i\mathbf{v}_i^T = U_k\Sigma_kV_k^T \tag{2.5}$$

A_k is the best approximation to A under both the Euclidean and Frobenius norm

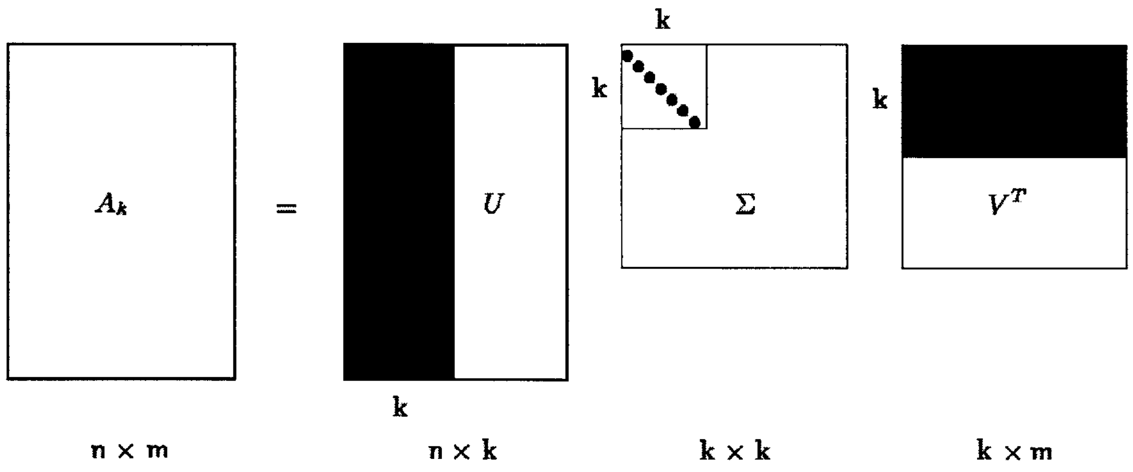

Figure 2-1: A representation of the construction of a rank-k approximation of A, A_k, from [6].

[26]. Therefore, the following two equalities hold:

$$\min_{\text{rank}(B)=k} \|A - B\|_F^2 = \|A - A_k\|_F^2 \tag{2.6}$$

$$\min_{\text{rank}(B)=k} \|A - B\|_2 = \|A - A_k\|_2 \tag{2.7}$$

The rank-k approximation of A provides the following benefits over A:

- **Storage.** Storing A requires $\mathrm{O}(nm)$ space while A_k can be maintained with $\mathrm{O}(k(n+m+1))$. This space difference is achieved by storing U_k, Σ_k, and V_k instead of A_k. Provided that $k \ll m$, we see the rank-k SVD performing as a lossy compression algorithm [9].

- **Denoising.** Removal of the smallest singular values and their corresponding singular vectors often removes noise in the data. This built-in filtering has led to A_k performing better in certain analyses than A itself [6].

There are two primary methods for calculating the rank-k SVD of A. As shown above, we can perform a full decomposition and truncate the result. However, the complexity of this approach is $\mathrm{O}(nm^2)$ which still scales poorly. The other approach involves using iterative solvers, such as orthogonal iteration and Lanczos iteration. These algorithms also become inefficient for large inputs due to superlinear computation time and diminishing convergence rate per singular triplet [2].

2.3 SVD and Streaming Data

The high complexity of the singular value decomposition coupled with applications that involve streaming data, i.e., the matrix A is built incrementally, motivates methods to update a previous singular value decomposition. In a data streaming setting, a new SVD would need to be computed as each new data point arrives. To circumvent this costly calculation, researchers have developed methods with varying robustness to update the U, Σ, and V matrices of a previous SVD instead of performing a new decomposition from scratch [3, 4, 5, 8, 17, 24, 25, 12, 30, 25, 7]. Such algorithms also remove the necessity of storing the matrix A. We will outline two such methods: *folding-in*, a subspace projection method, and the Brand *SVD-update*, an example from a class of low-rank update methods.

2.3.1 Folding-In

Folding-in is a simple procedure requiring only two matrix-vector products. To incorporate a new column of data d to an existing decomposition, $A = U\Sigma V^T$, c must be projected onto U and the result appended as a new row vector on V [6, 5, 28]:

$$\hat{d} = d^T U_k \Sigma_k^{-1} \tag{2.8}$$

$$V = \begin{bmatrix} V \\ \hat{d} \end{bmatrix} \tag{2.9}$$

The same approach but with swapped subspaces allows for a new row of data, f to be folded-in.

$$\hat{f} = f^T V_k \Sigma_k^{-1} \tag{2.10}$$

$$U = \begin{bmatrix} U \\ \hat{f} \end{bmatrix} \tag{2.11}$$

Folding-in can be performed in a block fashion with multiple rows or columns being folded-in simultaneously as seen in Figure 2-2. It is also not limited to full decompositions. Given a rank-k approximation of A, replacing U with U_k and V with V_k in the above formulas would allow for folding-in new data into a low-rank approximation.

The minimal calculations required to perform a folding-in significantly reduce the computation load when compared to a full or partial SVD. However, folding-in does not result in the same accuracy as a new SVD. Appending additional rows onto the left and right singular vectors results in a loss of orthogonality and subsequently a loss in quality. Furthermore, the new data does not affect the representation of the original data in the updated decomposition. This shortcoming causes significant error between the folding-in representation and a recomputed SVD when the new and old data are not closely related as any component not inside the projected subspace is lost [8].

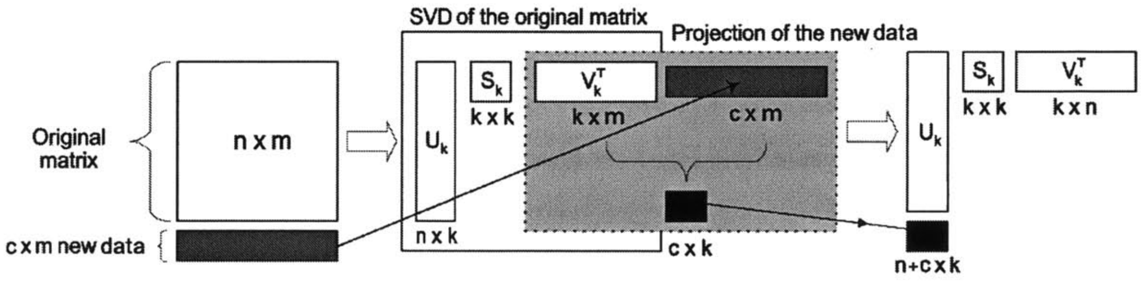

Figure 2-2: A graphical representation of the folding-in process from [28].

2.3.2 SVD-Update

SVD updating techniques aim to not only incorporate new rows and columns but also allow for downdates, edits, and shifts of the matrix while properly reflecting changes in latent structure resulting from said modification of A. Initial work focused on methods to update a full SVD, but high complexity issues led to a shift toward updating a rank-k SVD [11, 10, 4].

While numerous single pass methods for SVD updating have been independently developed, in this thesis we will specifically provide a summary of Brand's approach for both a full and rank-k SVD update [7, 8, 9]. We recommend [4] for a more complete overview of various incremental SVD methods.

Brand generalizes multiple modifications in the following form:

$$A + CD^T = \begin{bmatrix} U & C \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} V & D \end{bmatrix}^T \tag{2.12}$$

The matrices $\begin{bmatrix} U & C \end{bmatrix}$ and $\begin{bmatrix} V & D \end{bmatrix}$ can be decomposed into the product of an orthonormal matrix and another matrix.

$$\begin{bmatrix} U & C \end{bmatrix} = \begin{bmatrix} U & P \end{bmatrix} \begin{bmatrix} I & U^TC \\ 0 & R_C \end{bmatrix} \tag{2.13}$$

$$\begin{bmatrix} V & D \end{bmatrix} = \begin{bmatrix} V & Q \end{bmatrix} \begin{bmatrix} I & V^TD \\ 0 & R_D \end{bmatrix} \tag{2.14}$$

P and Q are the components of C and D that are orthogonal to U and V, respectively. Solving backwards yields $R_C \doteq P^T(I - UU^T)C$ and $R_D \doteq Q^T(I - VV^T)D$. After this orthogonalization, our intermediate result is the product of five matrices, two of which are orthonormal.

$$A + CD^T = \begin{bmatrix} U & P \end{bmatrix} \begin{bmatrix} I & U^TC \\ 0 & R_C \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & V^TD \\ 0 & R_D \end{bmatrix}^T \begin{bmatrix} V & Q \end{bmatrix}^T \tag{2.15}$$

When the product of the middle three matrices is diagonalized using the SVD, an updated SVD of $A + CD^T$ is achieved.

$$\begin{bmatrix} I & U^TC \\ 0 & R_C \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & V^TD \\ 0 & R_D \end{bmatrix}^T = \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} U^TC \\ R_C \end{bmatrix} \begin{bmatrix} V^TD \\ R_D \end{bmatrix}^T = U'\Sigma'V' \tag{2.16}$$

$$A + CD^T = (\begin{bmatrix} U & P \end{bmatrix} U')\Sigma'(\begin{bmatrix} V & Q \end{bmatrix} V')^T \tag{2.17}$$

In [9] Brand notes multiple strategies for reduced computation:

1. When appending a single row or column onto A, the diagonalization step in 2.17 becomes the SVD of a broken arrowhead matrix. This broken arrowhead matrix consists of all zeros except for on the diagonal and on either the last row or last column. As shown in [17], the SVD of a broken arrowhead matrix can be performed in $O(m^2)$ time.

2. If the rank of the updated SVD is fixed, a rank-one update, $\text{rank}(CD^T) = 1$, can have reduced computational complexity by maintaining decompositions of the left and right singular vectors.

   $$A + CD^T = U_{n \times k} U'_{k \times k} S_{k \times k} V'^T_{k \times k} V^T_{m \times k} \tag{2.18}$$

   The matrices U and V maintain the span of the left and right subspaces while $\check{U}$ and $\check{V}$ can be updated to store the rotations U' and V' from 2.17 with less computation.

2.4 Frequent Directions

In Sections 2.1 through 2.3, we reviewed the SVD, the rank-k SVD, and incremental SVD techniques. In this section we introduce a different technique for producing matrix approximations, matrix sketching. Rather than maintaining a decomposition, matrix sketching generates a significantly smaller matrix that approximates the original. Due to the sketch's smaller size, it is inherently low-rank.

In [23] Liberty presents a novel method of performing streaming matrix sketching. In a single pass of the data, this procedure sketches a matrix $A \in \mathbb{R}^{n \times m}$ into a matrix $B \in \mathbb{R}^{\ell \times m}$, $\ell \ll n$, which approximates A well.

Liberty's algorithm, Frequent Directions, builds upon the well-known approach used to estimate item frequency within a stream called Frequent Items or the Misra-

Gries algorithm [13, 19, 27]. Frequent Items maintains the frequency of a stream of n item appearances in a universe of m numbers in $O(\ell)$ space by subtracting equally from each frequency count to always ensure there are fewer than ℓ unique item frequencies being stored. This open space always allow the storage of a new item frequency not currently being tracked while limiting the difference between the calculated and actual item frequency. Frequent Directions is a translation of Frequent Items from single-dimension to multi-dimension data points. Just as Frequent Items tracks individual item frequencies, Frequent Directions is tracking orthogonal vectors. The subtracting process on the orthogonal vectors is done by shrinking the norms of the rows of B [23]. This method can be seen in Algorithm 1.

Algorithm 1 Frequent Directions [23]

```
Input: $\ell, A \in \mathbb{R}^{n \times m}$
 1: $B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times m}$
 2: for $i \in [n]$ do
 3:     Insert $A_i$ into an empty row of $B$
 4:     if $B$ has no empty rows then
 5:         $[U, \Sigma, V] \leftarrow \mathrm{SVD}(B)$
 6:         $\delta \leftarrow \sigma_{l/2}^2$
 7:         $\check{\Sigma} \leftarrow \sqrt{\max(\Sigma^2 - I_l \delta, 0)}$
 8:         $B \leftarrow \check{\Sigma} V^T$
 9:     end if
10: end for
Return: $B$
```

The full update time for Frequent Directions is $O(nm\ell)$, which is an amortized $O(m\ell)$ per row. This running time results from the SVD of B that must be calculated every $\ell/2$ rows that are processed. Liberty proves the following additive bounds for Frequent Directions in [23]:

$$\forall x, \|x\| = 1 \qquad 0 \leq \|Ax\|^2 - \|Bx\|^2 \leq 2\|A\|_F^2/\ell \tag{2.19}$$

$$B^T B \prec A^T A \quad \text{and} \quad \|A^T A - B^T B\| \leq 2\|A\|_F^2/\ell \tag{2.20}$$

In his experiments against other sketching algorithms, Frequent Directions is the most accurate approach apart from a brute force approach using the SVD when measured by $\|A^TA - B^TB\|$. It also performs significantly more accurately than its worse-case bound. However, the empirical running time results reflect the high asymptotic running time and verify that the algorithm is linear in all three variables ℓ, n, and m.

2.5 Frequent Directions II

Ghashami and Phillips present modifications in [15] to improve the accuracy of the Frequent Direction algorithm by Liberty and prove relative bounds for their implementation. Their altered algorithm which we have termed Frequent Directions II is described in Algorithm 2. Instead of performing the shrinking step every $\ell/2$ rows that results in half of the rows being emptied, Frequent Directions II zeros out a single row after incorporating each new row. Frequent Directions II also returns B_k instead of B, $\ell = \lceil k + k/\epsilon \rceil$. B_k is the best rank-k approximation to B and can simply be calculated by taking the top k rows of B.

Algorithm 2 Frequent Directions II [15]

```
Input: $\ell, A \in \mathbb{R}^{n \times m}$
1: $B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times m}$
2: for $i \in [n]$ do
3:     Set $B_+ \leftarrow B^{i-1}$ with the last row replaced with $A_i$
4:     $[U, \Sigma, V] \leftarrow \mathrm{SVD}(B_+)$
5:     $\delta \leftarrow \sigma_l^2$
6:     $\check{\Sigma} \leftarrow \sqrt{\max(\Sigma^2 - I_l\delta, 0)}$
7:     $B^i \leftarrow \check{\Sigma}V^T$
8: end for
Return: $B_k^i$
```

Ghashami and Phillips also prove relative bounds on top of the bounds in [23] for Frequent Directions II:

$$\forall x, \|x\| = 1 \qquad 0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \|A\|_F^2/\ell \tag{2.21}$$

$$\|A - A_k\|_F^2 \leq \|A\|_F^2 - \|B_k\|_F^2 \leq (1+\epsilon)\|A - A_k\|_F^2 \tag{2.22}$$

$$\|A - \pi_{B_k}(A)\|_F^2 \leq (1+\epsilon)\|A - A_k\|_F^2 \tag{2.23}$$

where $\pi_{B_k}(A)$ is the projection of A onto the rowspace of B_k [15].

Chapter 3

New Approximation Algorithms

In this chapter, we present new algorithms for faster streaming low-rank matrix approximations.

3.1 Incremental Frequent Directions II

We first present a modified version of Frequent Directions II that utilizes an incremental SVD update step. The inner loop of the Frequent Directions II, Algorithm 2, can be rephrased as an rank-one SVD update with $C^T = \begin{bmatrix} 0, \cdots, 0, 1 \end{bmatrix}$ and $D^T = A_i$ followed by a full-rank SVD downdate. Returning to 2.15, we begin with the SVD update. A number of simplifications immediately arise in this specific case. First, a lack of need of the left singular vectors in Frequent Directions II allows us to discard the first two matrices that calculate the updated U.

$$A + CD^T \rightarrow \begin{bmatrix} \Sigma & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & V^T D \\ 0 & R_D \end{bmatrix}^T \begin{bmatrix} V & Q \end{bmatrix}^T \tag{3.1}$$

The resulting left matrix-matrix product can further be simplified to:

$$\begin{bmatrix} \Sigma & 0 \\ V^T D & R_D \end{bmatrix} \begin{bmatrix} V & Q \end{bmatrix}^T \tag{3.2}$$

Next we recognize that since V spans $\mathbb{R}^m$ that there will be no component of D orthogonal to V and therefore $R_D \doteq Q^T(I - VV^T)D \doteq 0$.

$$\begin{bmatrix} \Sigma & 0 \\ V^TD & R_D \end{bmatrix} \begin{bmatrix} V & Q \end{bmatrix}^T \doteq \begin{bmatrix} \Sigma \\ V^TD \end{bmatrix} \begin{bmatrix} V & Q \end{bmatrix}^T \tag{3.3}$$

The final step is to incorporate the subspace rotation from the diagonalization into $\begin{bmatrix} V & Q \end{bmatrix}^T$. Note again that we drop $\check{U}$ as the sketch we are producing is only constructed from the singular values and right singular vectors.

$$\begin{bmatrix} \Sigma \\ V^TD \end{bmatrix} \begin{bmatrix} V & Q \end{bmatrix}^T \doteq \check{U}\check{\Sigma}\check{V}^T \begin{bmatrix} V & Q \end{bmatrix}^T \rightarrow \check{\Sigma}(\begin{bmatrix} V & Q \end{bmatrix} \check{V})^T \tag{3.4}$$

Superficially, it seems that our updated V matrix calculated by $(\begin{bmatrix} V & Q \end{bmatrix} \check{V})^T$ grows each iteration. However, since Σ has a zero-valued entry on its diagonal, the last row of $\check{V}$ is empty. Due to this structure, we can forgo appending Q onto V and drop the last row off of Q, simplifying the update to $V\check{V}$.

$$\check{\Sigma}(\begin{bmatrix} V & Q \end{bmatrix} \check{V})^T \rightarrow \check{\Sigma}(V\check{V})^T \tag{3.5}$$

Lines 5-8 of Algorithm 3 capture the entire simplified rank-one SVD update from Frequent Directions II.

It is trivial to see that the Misra-Gries [27] step on line 6 in Algorithm 2 would not only be difficult to frame as a matrix downdate in terms of C and D but also that it would not benefit from any special structures. It follows that introducing an additional diagonalization step (SVD) would be more computationally expensive than directly performing the simple arithmetic reduction of the singular values.

3.1.1 Complexity

We have replaced the full SVD in Algorithm 2 with an SVD update. The SVD update for each row processed consists of a matrix-vector multiplication in $O(m^2)$ time, the

Algorithm 3 Incremental Frequent Directions II

```
Input: $\ell, A \in \mathbb{R}^{n \times m}$
1: $\Sigma \leftarrow$ empty matrix $\in \mathbb{R}^{\ell-1 \times m}$
2: $V \leftarrow I_m$
3: for $i \in [n]$ do
4:     Set $n \leftarrow A_i V$ $\{A_i$ is the $i$th row of $A\}$
5:     Set the last row of $\Sigma$ to $n$
6:     $[\check{U}, \check{\Sigma}, \check{V}] \leftarrow \text{SVD}(\Sigma)$
7:     $V \leftarrow V\check{V}$
8:     $\delta \leftarrow \sigma_l^2$
9:     $\Sigma \leftarrow \sqrt{\max(\check{\Sigma}^2 - I_l\delta, 0)}$
10: end for
Return: $B = \Sigma V^T$
```

SVD of a highly structured $\ell \times m$ matrix in $O(m\ell^2)$ time, and a matrix-matrix multiplication of two $m \times m$ matrices in $O(m^3)$ time. The resulting complexity of Incremental Frequent Directions II is therefore $O(nm^3)$ compared to the $O(nm\ell^2)$ of Frequent Directions II.

3.2 Truncated Incremental Frequent Directions II

The second algorithm we present, named Truncated Incremental Frequent Directions II, is a form of Algorithm 3 built around taking advantage of specific matrix structures. This algorithm is designed to sacrifice precision to reduce the running time complexity of the higher cost steps introduced in Incremental Frequent Directions II: the SVD of the Σ matrix and the matrix-matrix multiplication of $V\check{V}$. We accomplish this my adapting a technique from Brand in [9]: we calculate and maintain a rank-k decomposition of B and also keep the incremental SVD decomposed into three matrices to defer the true subspace rotations.

We first perform a full SVD of $\begin{bmatrix} A_{\ell-1} \\ 0 \end{bmatrix}$. This calculation is the same as that performed during the $\ell-1$th iteration of Incremental Frequent Directions II. However, the algorithms diverge at line 3 in Algorithm 4 with the truncation of the first ℓ columns of V to create V^ℓ. This creates a rank-k decomposition of interest composed of a square matrix with diagonal entries $\sigma_1, \cdots, \sigma_{\ell-1}, 0$ and V^ℓ. The following loop in line 4-10 in Algorithm 4 is similar to the loop in Algorithm 3 barring three subtle differences.

1. Line 4 now involves the product of a vector and two matrices. This change is due to the algorithm maintaining the decomposition of the right singular vectors in V^ℓ and V'.

2. The diagonalization in line 7 is now that of a broken arrowhead matrix of the specific form:

$$\begin{bmatrix} \sigma_1 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n_1 & n_2 & n_3 & \cdots & n_\ell \end{bmatrix} \tag{3.6}$$

3. The subspace rotation in line 8 is matrix-matrix product between two $\ell \times \ell$ matrices.

Algorithm 4 Truncated Incremental Frequent Directions II

```
Input: $\ell, A \in \mathbb{R}^{n \times m}$
1: $V' \leftarrow I_\ell$
2: $[U, \Sigma, V] \leftarrow$ SVD( $\begin{bmatrix} A_{\ell-1} \\ 0 \end{bmatrix}$ ) {where $A_{\ell-1}$ is the first $\ell - 1$ rows of $A$}
3: $V^\ell \leftarrow V \begin{bmatrix} I_\ell \\ 0 \end{bmatrix}$
4: for $i \in [\ell, n]$ do
5:     Set $n \leftarrow A_i V^\ell V'$
6:     Set the last row of $\Sigma$ to $n$
7:     $[\check{U}, \check{\Sigma}, \check{V}] \leftarrow$ SVD($\Sigma$)
8:     $V' \leftarrow V'\check{V}$
9:     $\Sigma \leftarrow \mathrm{diag}(\sqrt{\sigma_1^2 - \sigma_\ell^2}, \sqrt{\sigma_2^2 - \sigma_\ell^2}, \ldots, \sqrt{\sigma_{\ell-1}^2 - \sigma_\ell^2}, 0)$
10: end for
Return: $B = \Sigma(V^\ell V')^T$
```

3.2.1 Complexity

The three changes to the algorithm result in a better computational complexity than both Frequent Directions II and Incremental Frequent Directions II. Grouping the vector-matrix-matrix product in line 5 as two vector-matrix products reduces the calculation to $\mathrm{O}(m\ell)$ time. The SVD on line 7 can be naively calculated in $\mathrm{O}(\ell^3)$ time, however, using Gu and Eisenstat's method for calculating the SVD of a broken arrowhead matrix can reduce it to $\mathrm{O}(\ell^2)$ time. The subspace rotation on line 8 can be performed in $\mathrm{O}(\ell^3)$ time. The total time of the entire algorithm is therefore bounded by $\mathrm{O}(n\ell^3)$.

3.3 Truncated Incremental Frequent Directions

The final algorithm we present applies the concepts from Truncated Incremental Frequent Directions II to Frequent Directions. Instead of performing the truncated SVD update for each row of A, it does a batch computation with $\ell/2$ rows each iteration. It is worth noting that processing multiple rows at once destroys the broken arrowhead structure of Σ from Truncated Incremental Frequent Directions.

Algorithm 5 Truncated Incremental Frequent Directions

```
Input: $\ell, A \in \mathbb{R}^{n\times m}$
 1: $V' \leftarrow I_\ell$
 2: $[U, \Sigma, V] \leftarrow$ SVD( $A_\ell$) {where $A_\ell$ is the first $\ell$ rows of $A$}
 3: $V^\ell \leftarrow V \begin{bmatrix} I_\ell \\ 0 \end{bmatrix}$
 4: for every $\ell/2$ rows $A_{\ell/2} \in A$ {starting with the $(\ell+1)$th row of $A$} do
 5:     Set $n \leftarrow A_{\ell/2} V^\ell V'$
 6:     Set the last $\ell/2$ rows of $\Sigma$ to $N$
 7:     $[\check{U}, \check{\Sigma}, \check{V}] \leftarrow$ SVD($\Sigma$)
 8:     $V' \leftarrow V'\check{V}$
 9:     $\delta \leftarrow \sigma_{l/2}^2$
10:     $\Sigma \leftarrow \sqrt{\max(\check{\Sigma}^2 - I_l\delta, 0)}$
11: end for
Return: $B = \Sigma(V^\ell V')^T$
```

3.3.1 Complexity

By incorporating $\ell/2$ rows into the SVD with each loop iteration, Truncated Incremental Frequent Directions has a single-loop-iteration complexity of $\mathrm{O}(m\ell^2)$. This iteration step occurs for every $\ell/2$ rows in A, resulting in an overall complexity of $\mathrm{O}(nml)$. Truncated Incremental Frequent Directions therefore has the same complexity as Frequent Directions.

Chapter 4

Algorithm Performance

In this chapter we compare the newly proposed algorithms to previous work on numerous data sets under additive and relative error bounds. The algorithms being compared are: a rank-ℓ SVD approximation, Frequent Directions, Frequent Directions II, Truncated Incremental Frequent Directions, and Truncated Incremental Frequent Directions II. The last two algorithms are the new methods developed in this thesis. Due to performance constraints, Incremental Frequent Directions II was not included in our experiments.

Table 4.1 and Table 4.2 review the computational complexity of the algorithms. Truncated Incremental Frequent Directions II enjoys better asymptotic performance than Frequent Directions II, both of which are bounded by the SVD calculated during each loop iteration. While Truncated Incremental Frequent Directions II has the same asymptotic runtime performance as Frequent Directions, the first is limited by a matrix-matrix multiplication while the second is limited by the SVD. The difference in bounding step of the algorithms, while not captured by order of growth, will be evident in the empirical experiments. It is important to note that Table 4.2 reflects the usage of a full SVD to calculate an updated rank-k decomposition. For a full discussion of alternatives to this method, see [4].

Algorithm	Asymptotic Running Time
Singular Value Decomposition	$O(nm^2)$
Frequent Directions	$O(nm\ell)$
Frequent Directions II	$O(nm\ell^2)$
Truncated Incremental Frequent Directions	$O(nm\ell)$
Truncated Incremental Frequent Directions II	$O(n\ell^3)$

Table 4.1: A list of algorithms and their respective asymptotic runtimes.

Algorithm	Asymptotic Update Time
Singular Value Decomposition	$O(nm^2)$
Frequent Directions	$O(m\ell)$
Frequent Directions II	$O(m\ell^2)$
Truncated Incremental Frequent Directions	$O(m\ell)$
Truncated Incremental Frequent Directions II	$O(\ell^3)$

Table 4.2: A list of algorithms and their respective asymptotic updates times.

4.1 Experimental Data

We perform experiments on five different types of data. The first three are synthetic and differ by diminishing rates of the signal singular values. The other two data sets are real-world data sets.

4.1.1 Synthetic Data

Each synthetic data matrix A was generated using the procedure outlined by Liberty in [23]:

$$A = SDU + N/\varsigma \tag{4.1}$$

Matrix S contains the signal coefficients in $\mathbb{R}^{n\times d}$ with $S_{i,j} \sim N(0,1)$ independently and identically distributed. Matrix $D \in \mathbb{R}^{d\times d}$ is a diagonal matrix containing the non-increasing singular values of the signal $d(i)$. Matrix U is a random d dimension subspace in $\mathbb{R}^m$. Matrix N, $N_{i,j} \sim N(0,1)$ i.i.d., adds Gaussian noise to the signal SDU and is scaled by ς to ensure the signal is recoverable.

Three different diminishing rates were used to construct the matrix D: linearly decreasing, inversely decreasing, and exponentially decreasing. We choose to perform tests on data with different singular value trends as the reduction in the Misra-Gries [27] step is dependent on the relative magnitude of the singular values of a matrix. Therefore, observing how the algorithms perform on these data sets better reveals their overall behavior. The diagonal values of D are populated by the associated equation:

$$D_{i,i} = \begin{cases} d(i), & \text{if } i \leq d. \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

$$\begin{aligned} \text{linear} : d(i) &= \frac{1-(i-1)}{d} \\ \text{inverse} : d(i) &= \frac{1}{i} \\ \text{exponential} : d(i) &= \frac{1}{i^x} \end{aligned} \tag{4.3}$$

The variable d captures the signal dimension and ranges from $m/50$ to $m/20$ in our experiments.

4.1.2 Collected Data

The two real data sets are from two common applications of low-rank matrix approximations: image analysis and information retrieval. We used a 980×962 data set from [18] that was created by converting a series of captured images into a bag-of-words representation. The second set was test collection of document abstracts acquired from [1]. These abstracts were then represented by their word presence from a large dictionary, resulting in a matrix with dimensions 7491×11429.

4.2 Accuracy Measurements

To compare the performance of the approximation algorithms, we use the error bounds provided by Liberty in [23] and by Ghashami and Phillips in [15]. Liberty proves in

[23] that $A^TA \approx B^TB$:

$$B^TB \prec A^TA \quad \text{and} \quad \|A^TA - B^TB\| \leq 2\|A\|_F^2/\ell \tag{4.4}$$

Another interpretation is that A and B are close in any direction in $\mathbb{R}^m$:

$$\forall x, \|x\| = 1 \qquad 0 \leq \|Ax\|^2 - \|Bx\|^2 \leq 2\|A\|_F^2/\ell \tag{4.5}$$

Ghasahmi and Phillips show relative error bounds in [15], which aligns with more frequently used means of measuring the accuracy of matrix approximations:

$$\|A - A_k\|_F^2 \leq \|A\|_F^2 - \|B_k\|_F^2 \leq (1+\epsilon)\|A - A_k\|_F^2 \tag{4.6}$$

$$\|A - \pi_{B_k}(A)\|_F^2 \leq (1+\epsilon)\|A - A_k\|_F^2 \tag{4.7}$$

where $\pi_{B_k}(A)$ is the projection of A onto the rowspace of B_k [15].

For the remainder of this thesis, we will refer to error measurement in Equation 4.4 as *Additive Error* and the measurement in Equation 4.6 as *Relative Error*.

4.3 Performance Analysis

We will now analyze the performance of each matrix approximation algorithm in terms of runtime and accuracy under *Additive Error* and *Relative Error* on our five sets of experimental data. For each test, we show log-lin plots of the natural log of runtime, of *Additive Error*, and of *Relative Error* for each algorithm against multiple values for ℓ, the resulting sketch dimension. All tests were performed on a Windows 7 machine with 6GB of RAM using an Intel(R) Core(TM) i7-2630QM CPU @ 2.00 GHz.

We will mainly be comparing Frequent Directions to Truncated Incremental Frequent Directions, and Frequent Directions II to Truncated Incremental Frequent Directions II, while the rank-ℓ SVD provides a solid baseline. Note that the rank-ℓ SVD method computes the full singular value decomposition of A and then truncates to

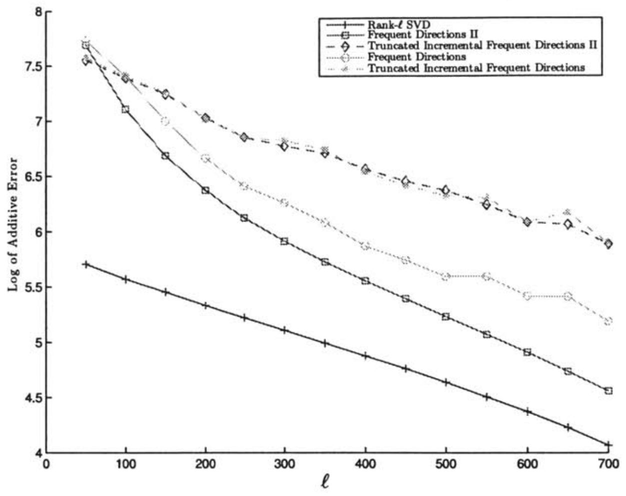

Figure 4-1: Natural log of *Additive Error* over values of ℓ for a 2000×1000 matrix with linearly decreasing singular values.

the top ℓ singular triplets for each value of ℓ, so its runtime is independent of ℓ. Also, given that the rank-ℓ SVD of A is the best rank-ℓ approximation of A, the rank-ℓ SVD will always provide a lower bound for the other algorithms.

4.3.1 Synthetic - Linear

The first experiment is performed using a linearly decreasing singular values, $n = 2000$, $m = 1000$, and $d = 20$.

We notice that in Figure 4-1 both of the proposed algorithms, Truncated Incremental Frequent Directions and Truncated Incremental Frequent Directions II, both perform better than their original counterparts under *Additive Error* for values of $\ell \leq 50$. However, their *Additive Error* only diminishes exponentially versus the faster rate of Frequent Directions and Frequent Directions II. Both new methods maintain lower *Relative Error* than their corresponding previous algorithms over the range of ℓ values in Figure 4-2. Figure 4-3 shows that the runtime difference between Frequent Directions II and Truncated Incremental Frequent Directions II matches what we expect from their asymptotic running times, but the that Truncated Incremental Frequent Directions is faster than Frequent Directions for the range of ℓ values despite their equivalent asymptotic running time. This behavior can be explained by the

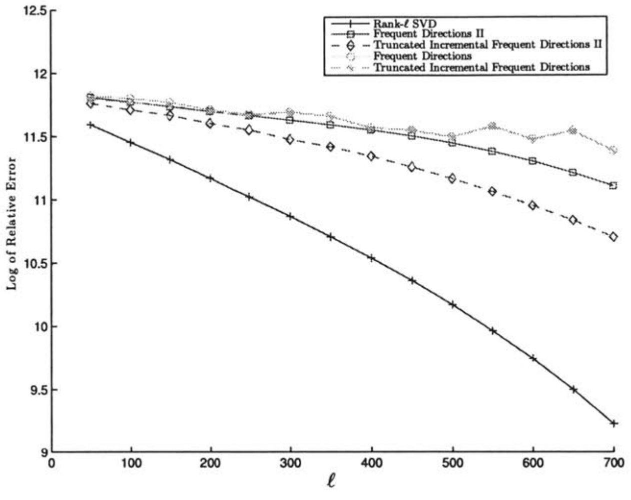

Figure 4-2: Natural log of *Relative Error* over values of ℓ for a 2000×1000 matrix with linearly decreasing singular values.

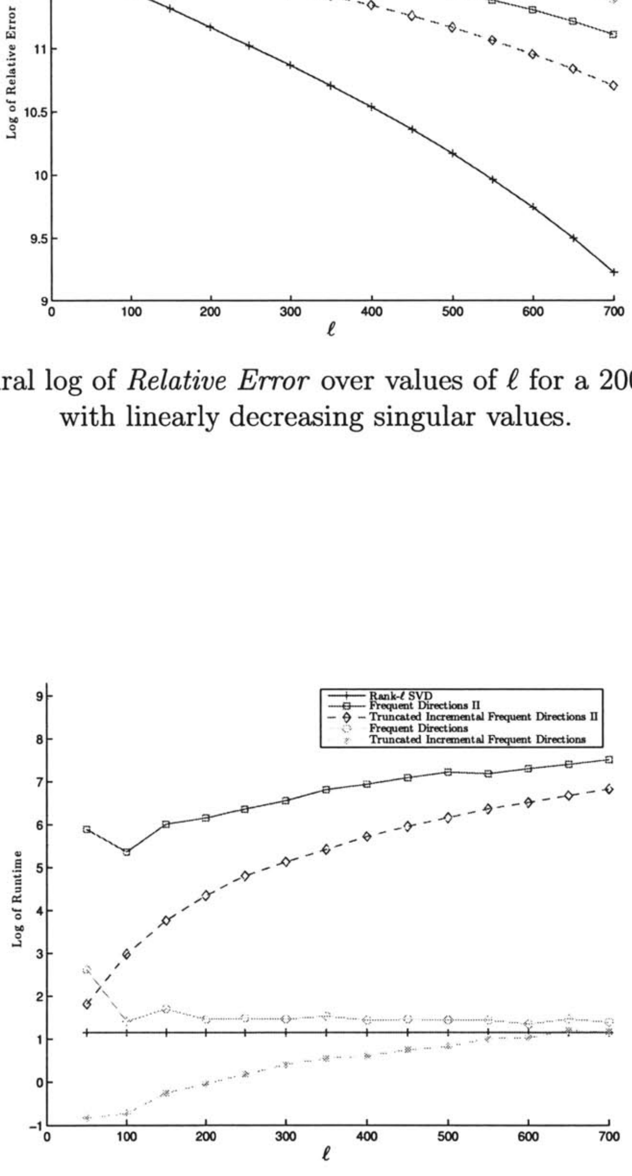

Figure 4-3: Natural log of runtime (in seconds) over values of ℓ for a 2000×1000 matrix with linearly decreasing singular values.

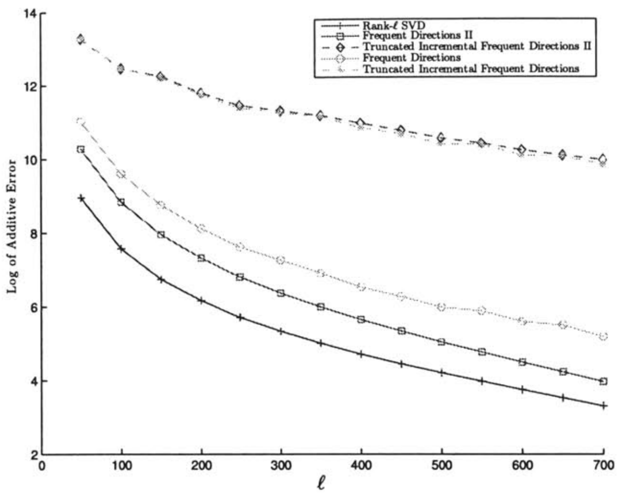

Figure 4-4: Natural log of *Additive Error* over values of ℓ for a 2000×1000 matrix with inversely decreasing singular values.

worst-case update time for Frequent Directions and Truncated Incremental Frequent Directions being bounded by the SVD and a matrix-matrix multiply, respectively. While asymptotically equivalent, the SVD takes significantly more flops to compute.

4.3.2 Synthetic - Inverse

The second experiment is performed using a inversely decreasing singular values, $n = 2000$, $m = 1000$, and $d = 20$. Figure 4-4 shows that neither algorithms proposed in this thesis perform as well as the original algorithms under *Additive Error*, and their error diminishes more slowly as with the previous data set. Again, Truncated Incremental Frequent Directions and Truncated Incremental Frequent Directions II are more accurate under *Relative Error* for the entire range of ℓ values as seen in Figure 4-5. The runtime performances in Figure 4-6 nearly match those in Figure 4-3 as the structure of the data has little effect on the computation being performed.

4.3.3 Synthetic - Exponential

The third experiment is performed using a exponentially decreasing singular values, $n = 2000$, $m = 1000$, $d = 20$, and $x = 2$. Figures 4-7, 4-8, 4-9 exhibit similar behav-

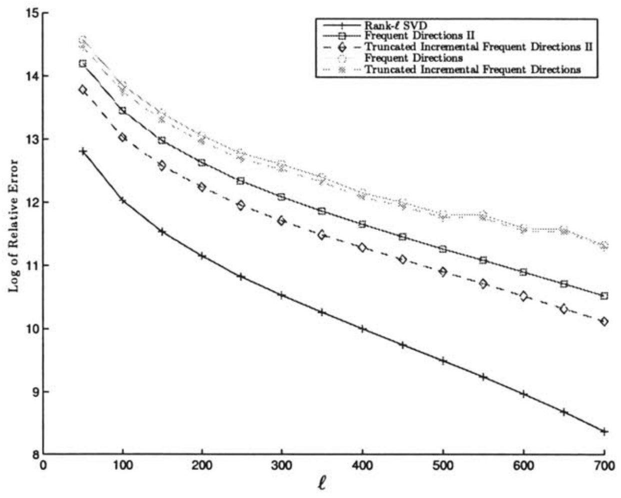

Figure 4-5: Natural log of *Relative Error* over values of ℓ for a 2000×1000 matrix with inversely decreasing singular values.

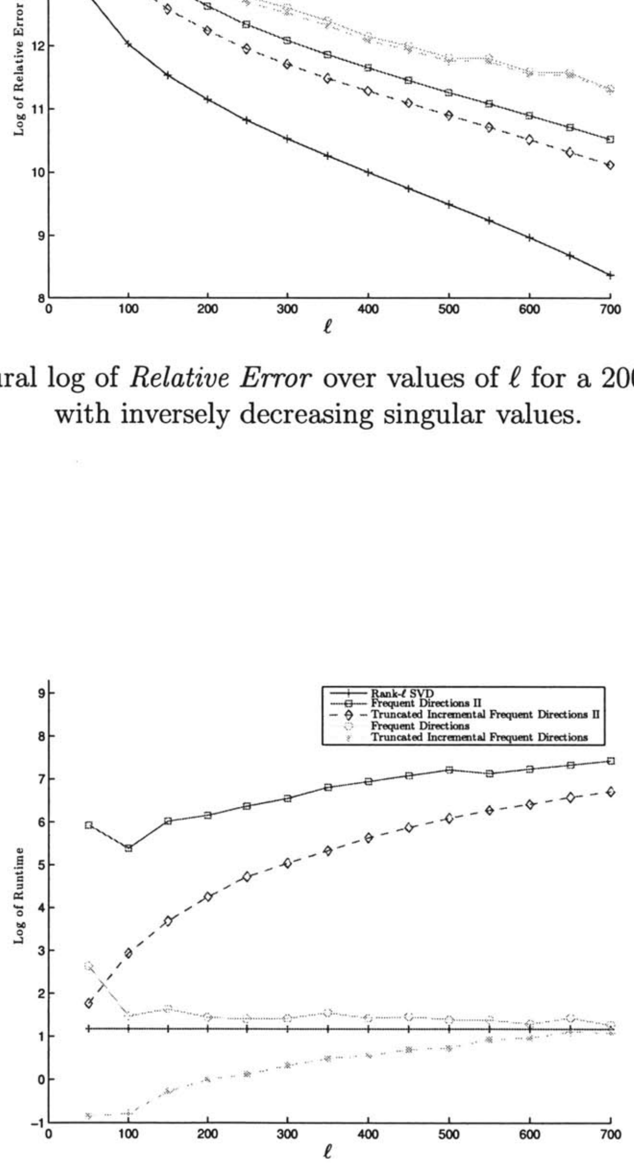

Figure 4-6: Natural log of runtime (in seconds) over values of ℓ for a 2000×1000 matrix with inversely decreasing singular values.

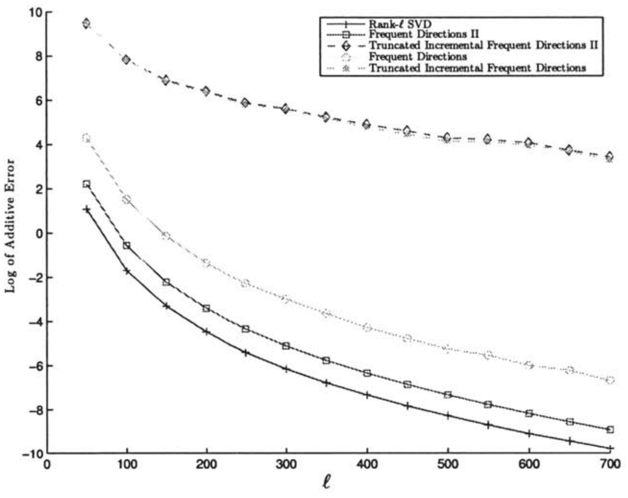

Figure 4-7: Natural log of *Additive Error* over values of ℓ for a 2000×1000 matrix with exponentially decreasing singular values.

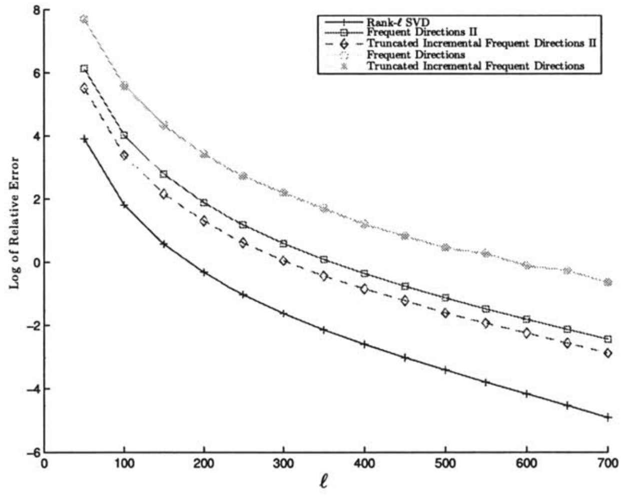

Figure 4-8: Natural log of *Relative Error* over values of ℓ for a 2000×1000 matrix with exponentially decreasing singular values.

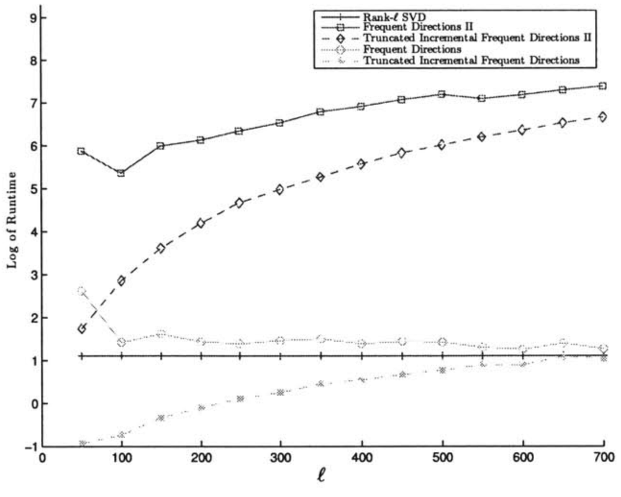

Figure 4-9: Natural log of runtime (in seconds) over values of ℓ for a 2000×1000 matrix with exponentially decreasing singular values.

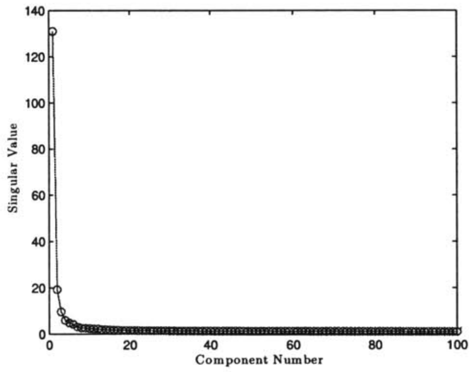

Figure 4-10: 100 largest singular values of the image analysis data set.

iors to the results in Figures 4-4, 4-5, 4-6 suggesting an increased rate of exponential fall-off in the singular values of the data will not change the relative performance of the algorithms.

4.3.4 Image Analysis

The fourth experiment is performed using our image analysis data set from [18] with $n = 980$ and $m = 962$. For comparison, we provide a plot of the 100 largest singular values of the image analysis data set in 4-10.

In Figure 4-11 we see the now familiar trend between the truncated and non-

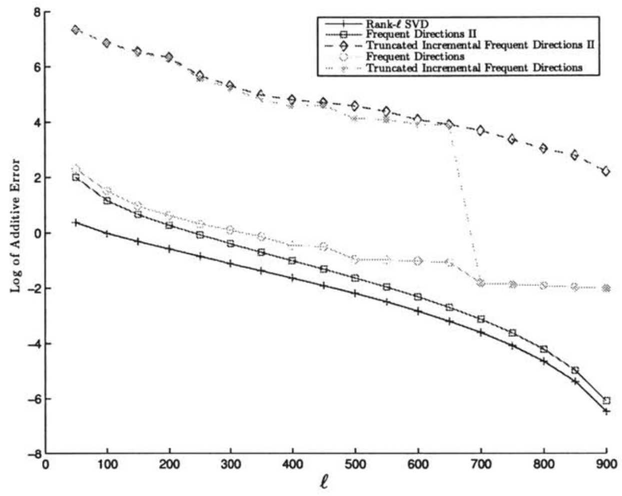

Figure 4-11: Natural log of *Additive Error* over values of ℓ for a 980×962 image analysis matrix.

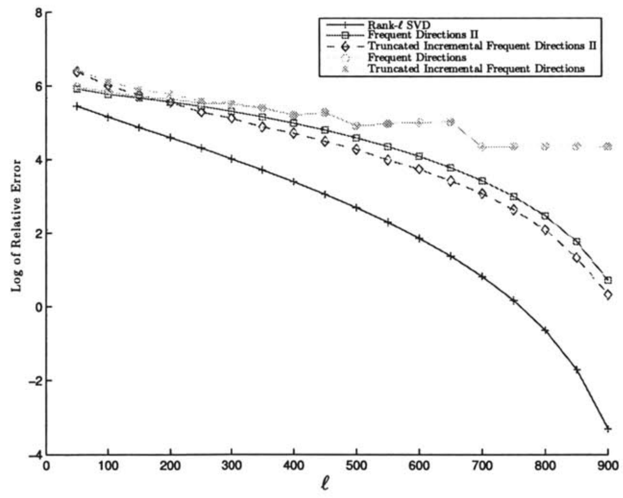

Figure 4-12: Natural log of *Relative Error* over values of ℓ for a 980×962 image analysis matrix

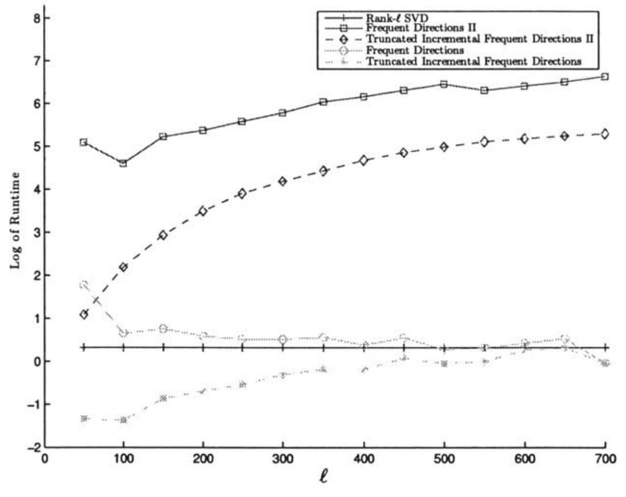

Figure 4-13: Natural log of runtime (in seconds) over values of ℓ for a 980×962 image analysis matrix

truncated algorithms. A new feature exhibited is that when Frequent Directions and Truncated Incremental Frequent Directions only execute their Misra-Gries [27] shrinkage step once ($\ell \geq 700$), they achieve nearly identical results as seen in both Figure 4-11 and Figure 4-12. This behavior causes a significant drop in the *Additive Error* of Truncated Incremental Frequent Directions for $\ell \geq 700$. Frequent Directions II also performs better than Truncated Incremental Frequent Directions II under *Relative Error* for $\ell \leq 150$ in Figure 4-5, which dashes any empirical notion that Truncated Incremental Frequent Directions II might always perform better under *Relative Error.* Once again, we see similar relative runtime performance in Figure 4-13 as in previous experiments due to similar dimensions.

4.3.5 Information Retrieval

The fifth experiment is performed using the information retrieval data set obtained from [1] with $n = 7945$ and $m = 11725$. For comparison, we provide a plot of the 100 largest singular values of the information retrieval data set in Figure 4-14. Due to the size of the matrix in this experiment leading to runtime limitations, Frequent Directions II and Truncated Incremental Frequent Directions II were excluded. Similar to the image analysis data set, this data set exhibits new relative behavior under

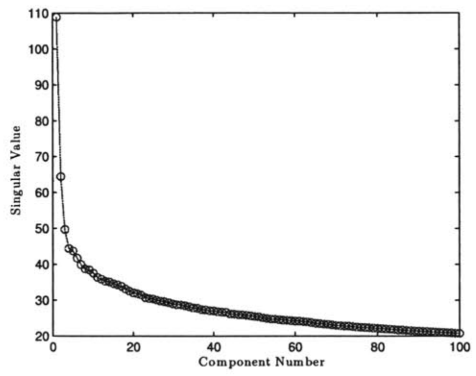

Figure 4-14: 100 largest singular values of the information retrieval data set.

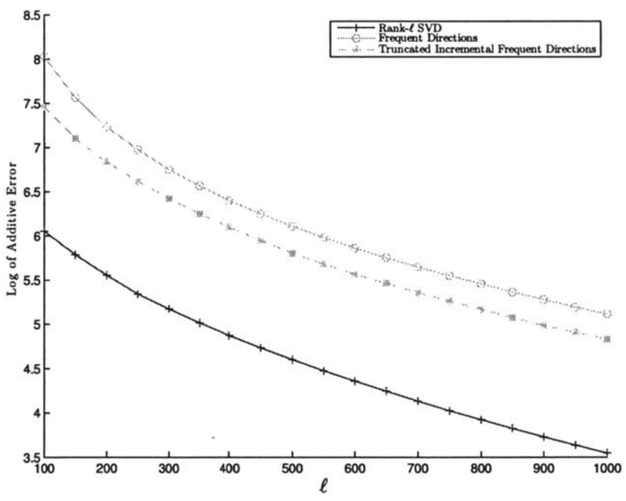

Figure 4-15: Natural log of *Additive Error* over values of ℓ for a 7491×11429 information retrieval matrix.

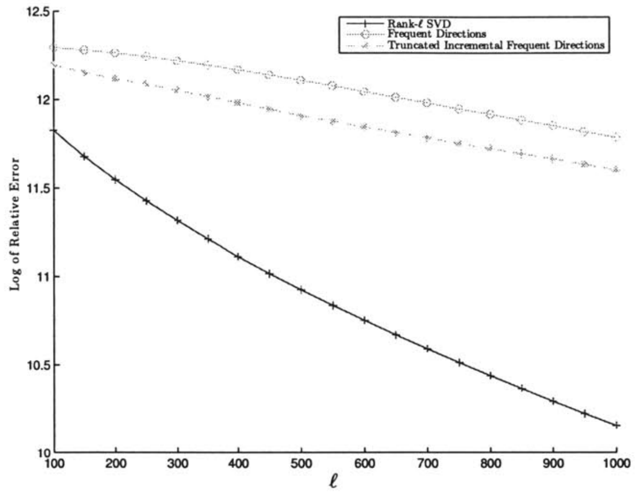

Figure 4-16: Natural log of *Relative Error* over values of ℓ for a 7491 × 11429 information retrieval matrix

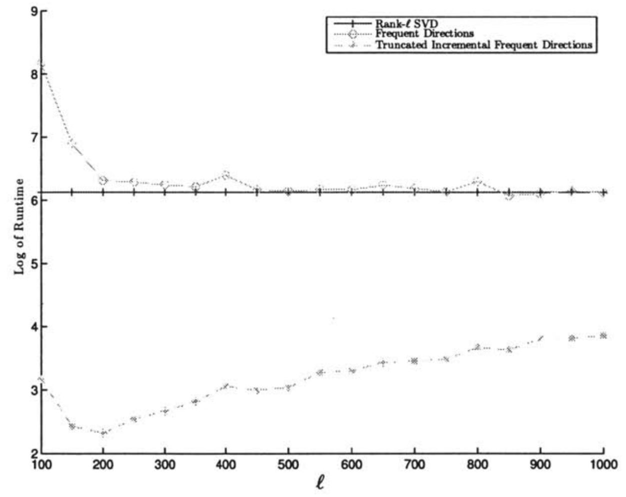

Figure 4-17: Natural log of runtime (in seconds) over values of ℓ for a 7491 × 11429 information retrieval matrix

the approximation algorithms. Truncated Frequent Directions II is more accurate than Frequent Directions II under both *Additive Error* and *Relative Error* as seen in Figures 4-15 and 4-16 while executing in less time for the range of ℓ.

4.4 Additional Observations

Across the experiments there are upward kinks in the error plots for Frequent Directions and Truncated Incremental Frequent Directions. This behavior is a result of the batch processing in both algorithms. As they eliminate and incorporate $\ell/2$ rows in a single iteration, there are times in which the sketch matrices, B, have $(\ell/2)+1$ empty rows while in the previous step there were no empty rows. Some values of ℓ may result in less accurate sketches than other values in the immediate vicinity, though the long-term trend of improved accuracy persists. In a non-online setting, one can minimize empty rows in B by choosing a value for ℓ such that $n = \ell + x((\ell/2)+1)$ for any integer x.

An anomaly in Figure 4-11 reveals an interesting relationship between Frequent Directions and Truncated Incremental Frequent Directions. When $n \geq 3\ell/2$, the Misra-Gries shrinkage step only occurs once. This behavior results in a large drop in *Additive Error* for Truncated Incremental Frequent Directions and near identical approximation error under both error measurements.

The runtime plots for Frequent Directions and Truncated Incremental Frequent Directions do not maintain a consistent upward or downward trend as ℓ increases across all of the data sets. This deviation is due to the trade-off between more expensive but infrequent calculations when ℓ is larger versus less computationally intensive per instance, but more frequent calculations when ℓ is smaller.

Chapter 5

Algorithm Application

In this chapter we apply our proposed approximation algorithms to the application of Latent Semantic Indexing.

5.1 Latent Semantic Indexing

Latent Semantic Indexing (LSI) is a modified approach to standard vector-space information retrieval. In both approaches, a set of m documents is represented by m $n \times 1$ vectors in $A \in \mathbb{R}^{n \times m}$, the term-document matrix. The elements of each vector represent the frequency of a specific word in that document, so $A_{i,j}$ is the frequency of word i in document j. The frequencies in matrix A are often weighted locally, within a document, and/or globally, across all documents to alter the importance of terms within or across documents [6, 5, 14]. Using vector-space retrieval, a query is represented in the same fashion as a document, as a weighted $n \times 1$ vector. The execution of the look-up of a query q is performed by mapping the query onto the row-space of A.

$$w = q^T A \tag{5.1}$$

The vector result w contains the relevancy scores between the query and each document. The index of the highest score in w is the index of the document in A that most closely matches the query, and a full index-tracking sort of w returns the documents in order of relevance to the query as determined by directly matching terms of the query and the documents [22].

Vector-space retrieval has numerous drawbacks. It can often return inaccurate results due to synonymy and polysemy [30]. Synonymy is the issue of concepts being described in different terms, resulting in queries not matching appropriate documents discussing the same concepts due to word choice. Polysemy is the problem of single words having multiple meanings. Such words can lead to documents being returned with high relevancy scores when in fact they share little to no conceptual content with the query [29]. Vector-space retrieval also requires the persistent storage of the matrix A as seen in Equation 5.1. As information retrieval is often performed on extremely large data sets, storing A is often undesirable [31].

Latent Semantic Indexing uses a rank-k approximation of A to try to overcome the issues of synonymy, polysemy, and storage. The concept behind LSI is that the matrix A_k will reveal some latent semantic structure that more accurately matches queries to documents based on shared concepts rather than words [14]. The query matching process in LSI is nearly identically to vector-space retrieval with the replacement of the term-document matrix [6]:

$$\hat{w} = q^T A_k \tag{5.2}$$

$$A_k = U_k \Sigma_k V_k^T \tag{5.3}$$

5.2 Approximate Latent Semantic Indexing

To test our algorithms performance in LSI, we leverage the work of Zhang and Zhu in [32]. When A is large, computing A_k directly using the SVD can be prohibitively expensive. However, A_k can be also computed by projecting A onto the space spanned by its top singular vectors:

$$A_k = U_k U_k^T A \tag{5.4}$$

$$A_k = A V_k V_k^T \tag{5.5}$$

In [32] Zhang and Zhu present an algorithm that uses a column-subset matrix sketching technique to approximate V_k and then applies it to the relationship in Equation 5.5 to approximate A_k. We present their approach in Algorithm 6 which we have generalized for any sketching technique.

Algorithm 6 Approximate Algorithm for LSI

```
Input: $\ell, k, A \in \mathbb{R}^{n \times m}$
1: $B \leftarrow$ sketch($A$) where $B \in \mathbb{R}^{\ell \times m}$
2: $S \leftarrow BB^T$
3: $[U, \Sigma, V] = \text{SVD}(S)$
4: for $i = 1, \dots, k$ do
5:     $\hat{\sigma}_t = \sqrt{\sigma_t}$
6:     $\hat{v}_t = Bu_t / \hat{\sigma}_t$
7: end for
8: $\hat{U}_k = A\hat{V}_k\hat{\Sigma}_k^+$ {$M^+$ is the pseudoinverse of $M$}
Return: $\hat{U}_k, \hat{\Sigma}_k$, and $\hat{V}_k$
```

5.3 NPL Data Set

We perform our LSI experiments on the Vaswani and Cameron (NPL) test collection obtained from [1]. There are $m = 11429$ documents represented by $n = 7491$ terms in this set, resulting in a term-document matrix $A \in \mathbb{R}^{n \times m}$. The data set comes with 93 queries and their corresponding truth relevance assessments [21].

5.4 Performance Analysis

We compare four approaches for information retrieval on the NPL data set: vector-space retrieval, rank-k LSI, approximate rank-k LSI using Frequent Directions, and approximate rank-k LSI using Truncated Incremental Frequent Directions. The latter two are performed using Algorithm 6 and using the corresponding sketching method in line 1. For a given value of k, we set $\ell = \lfloor 5k/4 \rfloor$. We do not test Frequent Directions II or Truncated Incremental Frequent Directions II due to runtime limitations.

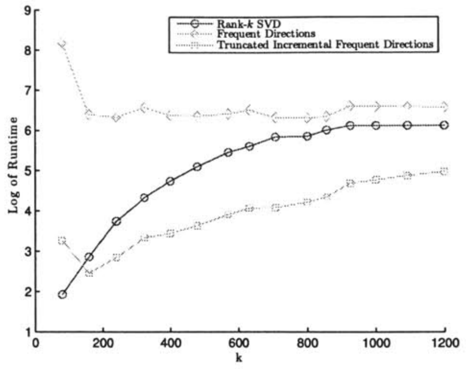

Figure 5-1: Natural Log of runtime (in seconds) of LSI techniques over values of k.

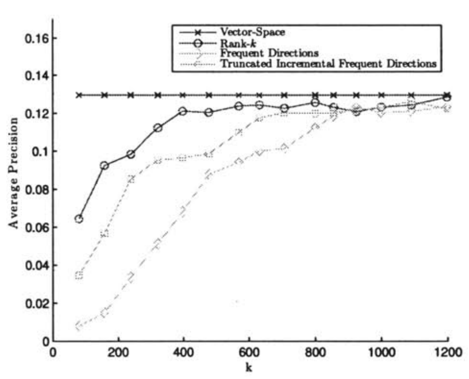

Figure 5-2: Average precision of information retrieval techniques over values of k.

We measure retrieval quality for a single query through precision and recall.

$$\text{precision} = \frac{|\text{correct documents returned}|}{|\text{documents returned}|} \tag{5.6}$$

$$\text{recall} = \frac{|\text{correct documents returned}|}{|\text{correct documents}|} \tag{5.7}$$

To summarize performance across the entire range of queries, we calculate the average precision of all the queries. Precision for a specific query is measured once a threshold percentage of correct documents has been retrieved [20]. In our analysis, we set this threshold to 60%.

Figure 5-1 resembles Figure 4-17 as the runtime-intensive portion of approximate rank-k LSI using matrix sketching is calculating the sketch. The results of the av-

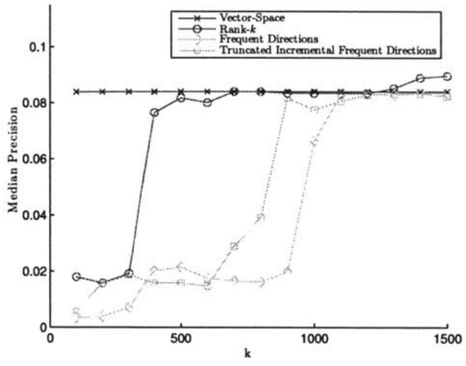

Figure 5-3: Median precision of information retrieval techniques over values of k.

erage precision calculation in Figure 5-2 reveal that Truncated Incremental Frequent Directions is generally more precise than Frequent Directions when used for approximate rank-k LSI. Both sketching methods perform worse under average precision than rank-k LSI and full vector-space retrieval for nearly the entire range of k values. The median precision plot in Figure 5-3 shows that using Truncated Incremental Frequent Directions requires a larger value for k to achieve the same precision as the rank-k approximation around the ideal value of k for this data set, $k = 500$ [20]. The comparative average and median precision between rank-k LSI and Truncated Incremental Frequent Directions LSI seems to indicate that the latter performs poorly for some subset of queries. Through analysis of the individual 93 queries, the offending subset was determined to be those queries with smaller norm values (fewer words). This result is most likely explained by shorter queries containing less conceptual material which makes them more susceptible to error caused by approximating A_k.

Based on these results, Truncated Incremental Frequent Directions seems very promising in LSI. It outperforms its counterpart, Frequent Directions, and is competitive with rank-k LSI for most values of k while enjoying better runtime performance. Each LSI approach performs worse than vector-space retrieval, but this behavior is a known characteristic of the extremely sparse NPL dataset [20]. It is expected that testing with denser data sets will show better LSI average and median precision compared to vector-space retrieval for all three algorithms [20]. Denser term-document matrices will also widen the runtime gap between rank-k LSI and the two matrix sketching approaches as the iterative rank-k SVD will take more flops to compute each singular triplet.

5.5 Additional Observations

Information retrieval collections in the real-world are often dynamic [30]. New documents are added in an streaming fashion. In the case of LSI being used to reduce storage requirements to $\mathrm{O}\big(k(n+m+1)\big)$, incorporating new data involves updating A_k or $\breve{A}_k$. As shown in Chapter 2, rank-k SVD updating techniques are non-trivially expensive. However, when using approximate rank-k LSI, additional $\mathrm{O}\big(\ell m\big)$ space can be used to store the sketch B. This additional constant space allows for the efficient update of B and the calculation of new $\hat{U}_k$, $\hat{\Sigma}_k$, and $\hat{V}_k$ using lines 2 through 8 of Algorithm 6.

Chapter 6

Conclusion

In this thesis we sought to create new algorithms for generating fast low-rank matrix approximations. Current approximations techniques are extremely important as they are used in a wide-range of applications such as principle component analysis, model simplification, k-means clustering, information retrieval, image recognition, and signal processing [6, 4, 23]. When designing our implementations, we consider three measures of performance: accuracy, running time, and space requirements. We combine aspects of multiple current approaches to present new faster algorithms with limited accuracy degradation.

We empirically test our algorithms on multiple synthetic and real data sets. Along with analyzing runtime performance, we measure the accuracy of our algorithms by using error calculations for *Additive Error* and *Relative Error* from previous work [23, 15]. Our experiments show that our new algorithms perform well under the *Relative Error* measurement but not normally under *Additive Error.* We apply our algorithms to Latent Semantic Indexing to see how they perform in an application setting. On the NPL data set, we achieve promising results for average and median precision across a set of 93 queries.

6.1 Future Work

While this thesis performed empirical analysis of the algorithms presented, a natural extension is to provide theoretical bounds for Truncated Incremental Frequent Directions and Truncated Incremental Frequent Directions II. Proper upper bounds for the error of both algorithms would allow for them to be confidently deployed into application settings.

Another avenue of future work is analyzing how these new algorithms perform in conjunction with others in specific applications. This approach follows from the work in [20, 14] that demonstrates that a weighted combination of low-rank approximations tend to achieve the best accuracy.

Appendix A

Implementations

```
function [B] = Incremental_Frequent_Directions(A,L)
        [n,m] = size(A);
        B = zeros(L,m);
        B(1:L-1,:) = A(1:L-1,:);
        [~,S,V] = svd(B);
        S(L,:) = [];
        for i=L:n
            if i == n
                break
            end
            N = A(i,:)*V;
            K = [S;N];
            [~,S,Vk] = svd(K);
            V = V * Vk;
            S = diag(S);
            S = diag((S(1:end-1).^2 - S(L)^2).^.5);
            S = [S zeros(L-1,m-size(S,2))];
        end
        B = S*V';
        B(L,:) = A(i,:);
end
```

```
function [B] = Truncated_Incremental_Frequent_Directions(A,L)
        [n,m] = size(A);
        [~,S,V] = svd(A(1:L,:));
        V_prime = eye(L);
        S = diag(S);
        S = (S(1:L/2-1).^2 - S(L/2)^2).^.5;
        S(L/2:L) = 0;
        S = [diag(S(1:L/2-1)) zeros(L/2-1,L/2+1)];
        V = V(:,1:L);
        row_count = L;
        zero_row_index = L/2;
        while row_count + ceil(L/2) < n
            N = A(1+row_count:1+row_count+(L-zero_row_index)
                ,:) * V * V_prime;
            K = [S;N];
                [~,S,D] = svd(K);
            V_prime = V_prime * D;
            S = diag(S);
            S = (S(1:L/2-1).^2 - S(L/2)^2).^.5;
            S(L/2:L) = 0;
            S = [diag(S(1:L/2-1)) zeros(L/2-1,L/2+1)];
            row_count = row_count + (L - zero_row_index + 1);
            zero_row_index = L/2;
        end
        B = S*V_prime'*V';
        if row_count < n
            B(zero_row_index:zero_row_index+n-row_count-1,:)
```

```
                    = A(row_count+1:n,:);
        end
        if size(B,1) < L
            B(size(B,1)+1:L,:) = 0;
        end
end
```

```
function [B] = Truncated_Incremental_Frequent_Directions_2(A,
    L)
        [n,m] = size(A);
        B = zeros(L,m);
        V = eye(m,L);
        V_prime = eye(L);
        S = zeros(L);
        for i=1:n
            if i == n
                break;
            end
            N = A(i,:) * V * V_prime;
            K = [S;N];
            [~,S,D] = svd(K);
            V_prime = V_prime * D;
            S = diag(S);
            S = diag((S(1:end-1).^2 - S(L)^2).^.5);
            S = [S zeros(L-1,L-size(S,2))];
        end
        B = S(:,1:L)*V_prime'*V';
        B(L,:) = A(i,:);
end
```

Bibliography

[1] Glasgow Test Collections.

[2] Dimitris Achlioptas and F McSherry. Fast computation of low rank matrix approximations. *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, 2001.

[3] CG Baker. A block incremental algorithm for computing dominant singular subspaces. *Vasa*, 2004.

[4] CG Baker, KA Gallivan, and P Van Dooren. Low-rank incremental methods for computing dominant singular subspaces. *Linear Algebra and its Applications*, 436(8):2866–2888, 2012.

[5] MW Berry. Computational methods for intelligent information access. *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*, pages 1–38, 1995.

[6] MW Berry, ST Dumais, and GW O'Brien. Using linear algebra for intelligent information retrieval. *SIAM review*, (December), 1995.

[7] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. *Computer VisionECCV 2002*, 2002.

[8] Matthew Brand. Fast Online SVD Revisions for Lightweight Recommender Systems. *SDM*, 35(TR-2003-14):37–46, 2003.

[9] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, May 2006.

[10] James R Bunch and Christopher P Nielsen. Updating the singular value decomposition. *Numerische Mathematik*, 31(2):111–129, 1978.

[11] P Businger. Updating a singular value decomposition. *BIT*, 10(3):376–385, September 1970.

[12] S. Chandrasekaran and B.S. Manjunath. An eigenspace update algorithm for image analysis. *Graphical Models and Image Processing*, 59(5):321–332, 1997.

[13] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. Frequency estimation of internet packet streams with limited space. In *AlgorithmsESA 2002*, pages 348–360. Springer, 2002.

[14] Andy Garron and April Kontostathis. Applying Latent Semantic Indexing on the TREC 2010 Legal Dataset. *TREC*, 2010.

[15] Mina Ghashami and JM Phillips. Relative Errors for Deterministic Low-Rank Matrix Approximations. *arXiv preprint arXiv:1307.7454*, 2013.

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

[17] Ming Gu and SC Eisenstat. A stable and fast algorithm for updating the singular value decomposition. *Yale University, New Haven, CT*, 1994.

[18] L. Hamilton, P. Lommel, T. Galvin, J. Jacob, P. DeBitetto, and M. Mitchell. Nav-by-Search: Exploiting Geo-referenced Image Databases for Absolute Position Updates. *Proceedings of the 26th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2013)*, pages 529–536, 2013.

[19] Richard M Karp, Scott Shenker, and Christos H Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems (TODS)*, 28(1):51–55, 2003.

[20] April Kontostathis. Essential dimensions of latent semantic indexing (lsi). *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 2007.

[21] April Kontostathis and WM Pottenger. A framework for understanding Latent Semantic Indexing (LSI) performance. *Information Processing & Management*, 19426(June 2004), 2006.

[22] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[23] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM, 2013.

[24] X Ma, D. Schonfeld, and A. Khokhar. Dynamic updating and downdating matrix SVD and tensor HOSVD for adaptive indexing and retrieval of motion trajectories. *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 1129–1132, 2009.

[25] Peter Halla David Marshallb Ralph Martinb. On adding and subtracting eigenspaces with EVD and SVD.

[26] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The Quarterly Journal of Mathematics*, 11(1):50–59, 1960.

[27] Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[28] B Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. *Fifth International Conference on Computer and Information Science*, 2002.

[29] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. *A practical approach to microarray data analysis*, 91, 2003.

[30] Hongyuan Zha and HD Simon. On updating problems in latent semantic indexing. *SIAM Journal on Scientific Computing*, 21(2):782–791, 1999.

[31] Hongyuan Zha and Zhenyue Zhang. Matrices with low-rank-plus-shift structure: Partial SVD and latent semantic indexing. *SIAM Journal on Matrix Analysis and Applications*, 21(2):522–536, 2000.

[32] D Zhang and Z Zhu. A fast approximate algorithm for large-scale latent semantic indexing. *Digital Information Management, 2008. ICDIM 2008. Third International Conference on*, 1(2):626–631, 2008.