Efficient Singular Value Decomposition Calculation for Information Retrieval

Sarah MacKinnon-Cormier

December 15, 2002

List of Figures

List of Tables

Acknowledgements

I would like to thank everyone who has encouraged and assisted me throughout the completion of this thesis. I would particularly like to thank my supervisor, Dr.Raymond J. Spiteri. Thank you for your patience, your support, and your time. I would also like to thank Dr. Keast for having introduced me to Numerical Linear Algebra, Dr. Blustein for having allowed me to the use his data set, Dr. Milios for having given me access to his computers, and Geoff Johnston for being so helpful. Lastly, I would like to thank NSERC for the financial support needed to realize this thesis.

Abstract

The rapid growth of digital libraries and other electronic databases has caused an increasing need for efficient information retrieval techniques. Latent Semantic Indexing (LSI) is one of several recent information retrieval techniques that is based on a vector space model in which documents in the database and user-specified queries are represented by a sparse term-by-document matrix. The Singular Value Decomposition (SVD) of the term-by-document matrix allows us to take advantage of the higher-order structure in the association of terms and documents by clustering terms and/or documents together, thus reducing the inaccuracies that arise with problems such as polysemy and synonymy. However, the SVD calculation, which breaks down into a bidiagonalization step and an iterative step, is generally very expensive. Although the efficiency of the iterative step cannot be improved, the efficiency of the bidiagonalization can be improved over conventional bidiagonalization methods by means of the Lanczos algorithm because the matrix is sparse. For the examples in this thesis, we find there is an order of magnitude improvement. It is also common for a user to add (update) or remove (downdate) terms or documents to or from a existing query where the SVD of the original term-by-document matrix is already known. In this thesis, we also give one-way and two-way chasing schemes to efficiently remove a document from a search. The chasing schemes modify the previous SVD to

obtain the downdated SVD at a significantly lower cost than recomputing the SVD from scratch. The two-way chasing scheme requires 50 % fewer plane rotations than the one-way chasing scheme.

Chapter 1

Background

1.1 Introduction

Due to the exponential growth of information available in media such as the Internet, information retrieval (IR) has become a topic of growing importance. Databases have become so large that it is essential that one be able to search their contents efficiently. An intuitive way to search a database is to use literal term matching. However, there are a number of issues which can prevent the results of a literal search from being accurate. Two of these issues include *polysemy* and *synonymy*. Polysemous words are characterized by many meanings. For example, *hip* is polysemous because it could mean *fashionable* but it could also mean *the pelvic region from the waist to the thigh*. Polysemy causes terms in the query to be matched with words in irrelevant documents and for the search results to be too broad. Words that are synonynms have equivalent meanings. For example, *spine* and *backbone* are synonyms. Thus, in a literal search, only documents containing the literal query terms will be matched, causing the search results to exclude relevant documents [1].

Latent Semantic Indexing (LSI) is one of numerous IR techniques that are based on the Singular Value Decomposition (SVD). To retrieve information from a database, the data which it contains along with the search query are represented mathematically as vectors in some vector space. Suppose the database in question contains a total of d documents and the query the user wishes to search the database by contains t terms. Then, a term-by-document, $t \times d$, matrix A is formed. Each entry $A(i, j)$, where $1 \leq i \leq t$ and $1 \leq j \leq d$, represents the positive frequency of term i in document j [1]. We now give a simple example to illustrate how a term-by-document matrix is formed.

Example 1:

Suppose our vocabulary consists of 18 keywords:

k_1: knitting	k_7: hat	k_{13}: hook
k_2: needles	k_8: mittens	k_{14}: weaving
k_3: angora	k_9: pattern	k_{15}: buttons
k_4: wool	k_{10}: yarn	k_{16}: ball
k_5: sweater	k_{11}: kit	k_{17}: stitch
k_6: socks	k_{12}: crochet	k_{18}: needlework

Suppose our database consists of the titles of 14 documents:

d_1: Crystal Palace Bamboo Knitting Needles

d_2: Counted Cross, Stitch, Needlework, and Stitchery Book

d_3: Hat, Mittens, and Socks Patterns

d_4: A Simple Ball of Yarn Can Become a Web of Love

d_5: A Child's Weaving Kit

d_6: Donna's Corporation Needlework and Craft Kit Company

d_7: Lessons on How to Knit Cables as Well as Two Cable Patterns

d_8: Sewing Buttons onto Wool Sweaters

d_9: How to Spin Angora Wool Yarn

d_{10}: Yarn Forward - Knitting Needles and Crochet Hooks

d_{11}: Knit Wool Sweaters, Wool Cardigans, and Wool Vests for Christmas

d_{12}: Instructions for Basket Weaving Patterns

d_{13}: Knitting Needles and Gadgets

d_{14}: Tips for Harvesting Angora Wool

A keyword-by-document matrix K is formed as follows:

Keywords	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}
knitting	1	0	0	0	0	0	1	0	0	1	1	0	1	0
needles	1	0	0	0	0	0	0	0	0	1	0	0	1	0
angora	0	0	0	0	0	0	0	0	1	0	0	0	0	1
wool	0	0	0	0	0	0	0	1	1	0	3	0	0	1
sweater	0	0	0	0	0	0	0	1	0	0	1	0	0	0
socks	0	0	1	0	0	0	0	0	0	0	0	0	0	0
hat	0	0	1	0	0	0	0	0	0	0	0	0	0	0
mittens	0	0	1	0	0	0	0	0	0	0	0	0	0	0
pattern	0	0	1	0	0	0	1	0	0	0	0	1	0	0
yarn	0	0	0	1	0	0	0	0	1	1	0	0	0	0
kit	0	0	0	0	1	1	0	0	0	0	0	0	0	0
crochet	0	0	0	0	0	0	0	0	0	1	0	0	0	0
hook	0	0	0	0	0	0	0	0	0	1	0	0	0	0
weaving	0	0	0	0	1	0	0	0	0	0	0	1	0	0
button	0	0	0	0	0	0	0	1	0	0	0	0	0	0
ball	0	0	0	1	0	0	0	0	0	0	0	0	0	0
stitch	0	2	0	0	0	0	0	0	0	0	0	0	0	0
needlework	0	1	0	0	0	1	0	0	0	0	0	0	0	0

Table 1.1: 18×14 Keyword-by-Document Matrix K

For example, since the keyword *wool* appears three times in the title of document 11, $A(4, 11) = 3$. Also, the density of a matrix is defined as the following:

$$density = \frac{number\ of\ non-zero\ entries}{number\ of\ entries} * 100.$$

We note that the density of K is 14%.

Suppose that we wish to narrow our search to only 4 keywords, called terms. Let these terms be

t_1: crochet

t_2: angora

t_3: wool

t_4: sweater.

A 4 $\times$ 14 term-by-document matrix, denoted as A, is formed.

Terms	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}
crochet	0	0	0	0	0	0	0	0	0	1	0	0	0	0
angora	0	0	0	0	0	0	0	0	1	0	0	0	0	1
wool	0	0	0	0	0	0	0	1	1	0	3	0	0	1
sweater	0	0	0	0	0	0	0	1	0	0	1	0	0	0

Table 1.2: 4 $\times$ 14 Term-by-Document Matrix A

In practice, term-by-document matrices are sparse matrices. In other words, if one considers the abundance of documents available, for example, on the Internet, keywords or terms of a query will appear in only a small fraction of the documents, implying that very few of the entries in matrices K and A will be non-zero. The sparsity of term-by-document matrices will depend on the text corpus and the query. For example, TREC (Text REtrieval Conference) uses term-by-document matrices which are 0.001-0.002% dense [3]. The term-by-document matrices used in [2] works are 1% dense. The text corpus which will be introduced in Section 1.3 generates matrices which are 5% dense with our chosen queries.

Calculating the Singular Value Decomposition (SVD) of A is a way to compress the data and elicit the implicit higher-order structure of relevance between terms and documents [2]. It is important to note that calculating the SVD of a matrix is an expensive operation; aspects of how to make the computations more efficient will be explored later on.

1.2 Latent Semantic Indexing

Latent Semantic Indexing is one of many recent information retrieval techniques that is based on a vector space model such as we have described above [1]. In a vector space model, documents are represented as the columns of a matrix, and the terms in the vocabulary are represented as the rows. In this way, a term-by-document matrix is formed. The elements of A are locally weighted to appropriately increase or decrease the relevance of terms within a document, and globally weighted to appropriately increase or decrease the relevance of a term among all documents. Each entry of A is defined as

$$A(i,j) = L(i,j) \times G(i),$$

where $L(i,j)$ is the local weighting of term i in document j, $G(i)$ is the global weighting of term i among all the documents, $1 \leq i \leq t$, and $1 \leq j \leq d$ [7]. For example, a common word such as *and*, which does not say much about the document, would be lightly weighted.

The Singular Value Decomposition (SVD) of a matrix $A \in \Re^{m \times n}$ is as follows:

$$A = U \Sigma V^T$$

where $U \in \Re^{m \times m}$ and $V \in \Re^{n \times n}$ are orthogonal. The diagonal entries of $\Sigma \in \Re^{m \times n}$ are the singular values of matrix A, denoted as σ_i where $i = 1, 2, ..., \min(m, n)$. All other entries are zero. Another way of defining the SVD of A is

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T, \tag{1.1}$$

where u_i is the ith column of U, v_i is the ith column of V, and $r \leq \min(m, n)$ is the rank of A. A more involved definition of SVD will be given later.

A rank-k approximation A_k of A is found by taking the summation of (**??**) from 1 to k. Hence,

$$A_k = U_k \Sigma_k V_k^T.$$

Noise, caused by factors such as polysemy and synonymy, is removed from the vector space by replacing the complete vector space A with the subspace A_k. In this subspace, terms and documents that are closely related are also clustered together [7].

We now give a specific example of a truncated SVD factorization of a term-by-document matrix.

Example 2:

We compute the truncated SVD of A from Example 1 with $k = 2$. Normally, the terms within the term-by-document are weighted; however, for simplicity, this step will be omitted. Let

$$A = U\Sigma V^T \approx A_2 = U_2\Sigma_2 V_2^T,$$

where

$$\begin{aligned} U &= (u_1 \quad u_2 \quad u_3 \quad u_4), u_j \in \Re^{18\times 1} \\ V^T &= (v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6 \quad v_7 \quad v_8 \quad v_9 \quad v_{10} \quad v_{11} \quad v_{12} \quad v_{13} \quad v_{14}), v_j \in \Re^{14\times 1} \end{aligned}$$

and

$$\Sigma = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \sigma_3 & \\ & & & \sigma_4 \end{pmatrix}$$

are the normal SVD factors of A. $U_2 \in \Re^{18\times 2}$, $\Sigma_2 \in \Re^{2\times 2}$, and $V_2 \in \Re^{14\times 2}$.

We wish to compute the truncated SVD factors for U_2, Σ_2, and V_2.

$$U_2 = \begin{pmatrix} u_1 & u_2 \end{pmatrix}, \quad V_2 = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix},$$

and Σ_2 consists of the 2×2 top left-hand block of Σ.

We have carried out these computations in *Matlab*, and report the results to 4 decimal places.

$$U_2 = \begin{pmatrix} -.3384 & -.5850 \\ -.1014 & -.5493 \\ -.1409 & .0591 \\ -.8609 & .3083 \\ -.2946 & .1268 \\ -.0024 & -.0333 \\ -.0024 & -.0333 \\ -.0024 & -.0333 \\ -.0287 & -.1541 \\ -.1268 & -.2836 \\ -.0002 & -.0052 \\ -.0432 & -.2519 \\ -.0432 & -.2519 \\ -.0022 & -.0283 \\ -.0818 & .0656 \\ -.0090 & -.0428 \\ .0000 & -.0006 \\ .0000 & -.0012 \end{pmatrix}, V_2 = \begin{pmatrix} -.1131 & .0000 \\ -.4107 & -.0009 \\ -.0065 & .9361 \\ .0379 & .1293 \\ -.2581 & -.0183 \\ -.0039 & .2330 \\ .1396 & .0768 \\ -.1447 & .1558 \\ .0831 & -.0756 \\ -.3780 & .0000 \\ .2117 & .1071 \\ -.0989 & .0700 \\ -.0750 & .0086 \\ .7071 & .0000 \end{pmatrix},$$

$$\Sigma_2 = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = \begin{pmatrix} 3.8881 & 0 \\ 0 & 2.6720 \end{pmatrix}.$$

To observe clustering of terms and/or documents, the terms and documents have been plotted on a two-dimensional graph. See Figure 1.1.

The x-coordinates and y-coordinates for the terms and documents are calculated in the following way:

$$\begin{aligned} x_{terms} &= \sigma_1 u_1 \\ y_{terms} &= \sigma_2 u_2 \\ x_{documents} &= \sigma_1 v_1 \\ y_{documents} &= \sigma_2 v_2 \end{aligned}$$

We note that some of the labels on the points of the graph have been left out for clarity.

The clustering of the points in Figure 1.1 suggest the terms *knitting* and *needles* are closely related. The third document, *Hat, Mitten, and Sock Patterns*, has little relationship with the word *needles* and therefore the two points by which they are represented in the figure are quite far apart.

Now, suppose we wish to search by only 4 keywords: *crochet*, *angora*, *wool*, and *sweater*. This query, denoted as q, corresponds to the term-by-document matrix A in Example 1. The query itself is represented by

$$q^T = (0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0).$$

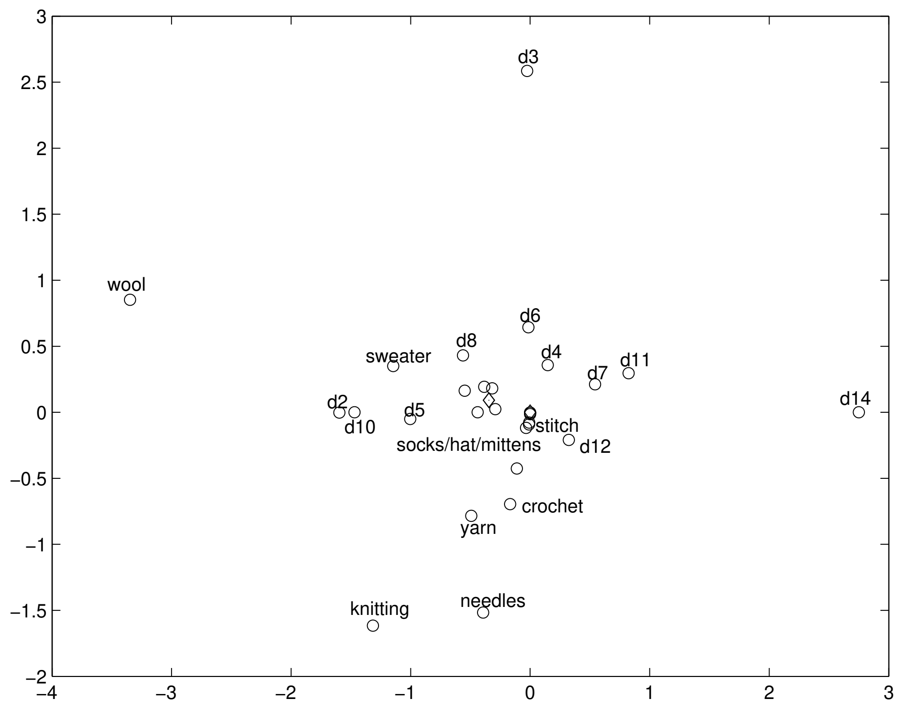

Figure 1.1: Two-Dimensional Plot of 18 Terms and 14 Documents

However, q can be approximated by a 1×2 column vector by the following equation [2]:

$$\hat{q} = q^T U_2 \Sigma_2^{-1}.$$

So,

$$\hat{q} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}^T \begin{pmatrix} -.3384 & -.5850 \\ -.1014 & -.5493 \\ -.1409 & .0591 \\ -.8609 & .3083 \\ -.2946 & .1268 \\ -.0024 & -.0333 \\ -.0024 & -.0333 \\ -.0024 & -.0333 \\ -.0287 & -.1541 \\ -.1268 & -.2836 \\ -.0002 & -.0052 \\ -.0432 & -.2519 \\ -.0432 & -.2519 \\ -.0022 & -.0283 \\ -.0818 & .0656 \\ -.0090 & -.0428 \\ .0000 & -.0006 \\ .0000 & -.0012 \end{pmatrix} \begin{pmatrix} 0.2572 & 0 \\ 0 & 2.6720 \end{pmatrix} = \begin{pmatrix} -.3452 & .0907 \end{pmatrix}.$$

Figure 1.2 shades the documents that would be returned from doing LSI. Using a cosine threshold of 0.85, the documents pertaining to *crochet*,*angora*, *wool*, and *sweater* are returned. Since $\cos^{-1} .85 \approx 31.8^o$, the boundaries of the shaded area are the lines which are 31.8^o from either side of the query vector.

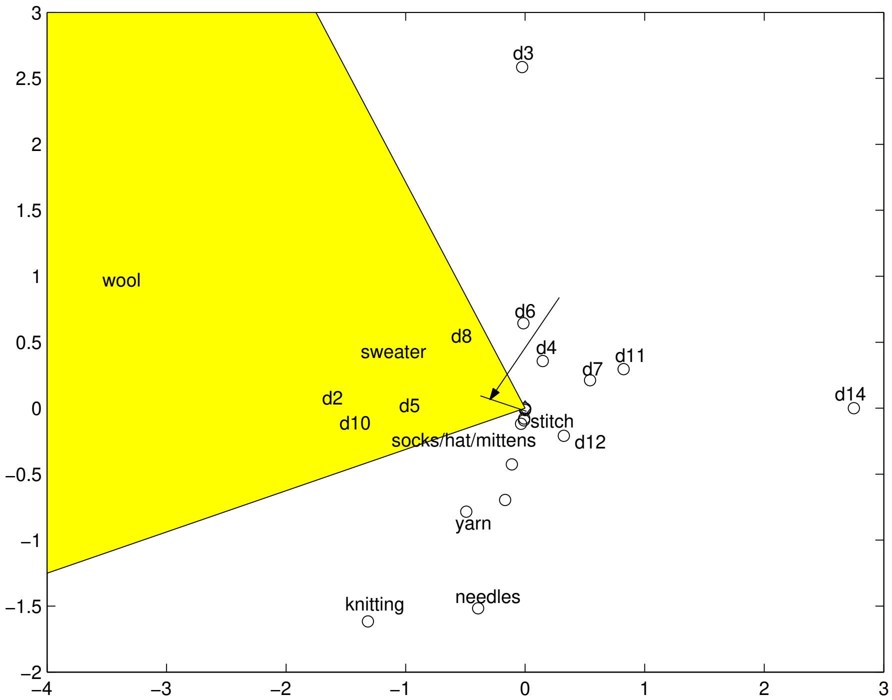

Figure 1.2: Two-Dimensional Plot of 18 Terms and 14 Documents with 4-Term Query

Notice that the documents to be returned as relevant are d_2, d_5, d_8, and d_{10}.

1.3 Text Corpus

The text corpus used throughout this paper is a collection of 1534 documents. The documents are newsgroup postings about computer graphics to Usenet. The text corpus was created by Mark Staveley for his M.Sc. in the faculty of Computer Science, University of Waikato, Hamilton, New Zealand.

1.4 Outline of the Thesis

The remainder of this thesis unfolds as follows. In Chapter 2, we give a detailed description of algorithms to compute the SVD of a matrix. We compare and contrast the computational approaches for sparse and dense matrices. In Chapter 3, we give a detailed description of an algorithm for downdating the SVD. Finally, in Chapter 4, we give some conclusions and direction for future work.

Chapter 2

Computing the SVD of a Matrix

In this chapter, we give a detailed description of an algorithm to compute the SVD of a matrix.

Given a general dense matrix $A \in \Re^{m \times n}$ and a number ϵ representing a multiple of unit round-off, this algorithm returns matrices U, V, and Σ, such that $A = U\Sigma V^T$, where U is an orthogonal $m \times m$ matrix, V is an orthogonal $n \times n$ matrix, and Σ is an $m \times n$ matrix whose "diagonal" entries are the singular values of A [4].

Because it is more efficient, it is common practice to bidiagonalize a matrix before its SVD is calculated [9]. Therefore, it will be assumed the factorization

$$A = UBV^T$$

has been calculated where $B \in \Re^{m \times n}$ is a bidiagonal matrix. For example, we could use one of the following two bidiagonalization procedures to be described below.

As mentioned, in modern software implementations to calculate the SVD of a matrix, the calculations naturally break down into two phases: a bidiagonalization phase and an iterative phase. We now study each of these phases in more detail.

2.1 Matrix Bidiagonalization

We assume the matrix $A \in \Re^{m \times n}$ is an overdetermined matrix; i.e., $m \geq n$. If this is not the case, the matrix should be transposed prior to being passed to one of the following bidiagonalization methods.

If $m > n$, then we have n columns of elements to zero out below the diagonal and $n-2$ rows of elements to zero out above the super-diagonal.

If $m = n$, then we have $n-1$ columns of elements to zero out below the diagonal and $n-2$ rows of elements to zero out above the super-diagonal.

It turns out that the efficiency of the bidiagonalization method depends critically on whether A is sparse. So, there are two cases to consider when bidiagonalizing a matrix:

1. matrix A is dense.

2. matrix A is sparse.

In the first case, the most popular bidiagonalization method involves the use of Householder transformations (see Section 2.1.1). In the second case, many bidiagonalization methods exist. In this thesis, we will consider only the Lanczos algorithm (see Section 2.1.2).

2.1.1 Bidiagonalization using Householder Transformations

When a matrix is dense, the "zeroing out" of columns and rows is done efficiently with Householder transformations.

Householder transformations are matrices of the form [6]

$$H = I - vv^T,$$

where v is a vector such that

$$\|v\|_2 = \sqrt{2}.$$

We now describe an algorithm that will reduce a general matrix to bidiagonal form by means of Householder transformations.

Set the flag *skip* to true. This flag will prevent the zeroing out of columns which are already zero.

Let $num = \min(m-1, n-2)$.

Set $i = 1$.

Until $i = num$

Let x be the part of column i of A from the diagonal entry to the bottom of A, i.e., $x = A(i:m, i)$.

Formulate

$$v = \sqrt{2}\frac{x-y}{\|x-y\|_2},$$

where

$$y = \begin{pmatrix} -sgn(x_1)\|x\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Sequentially check each element of v starting from 1. If an element is found whose absolute value is greater than ϵ, set flag *skip* to false and break from loop. Otherwise, *skip* remains true.

If *skip* is false do

Let

$$A \leftarrow A - vv^T A.$$

This is the left Householder transformation result.

Let

$$U \leftarrow U - vv^T U.$$

end.

Reset *skip* to true, its default value.

Let x be the part of row i of A from the super-diagonal entry to the end of the row.

Formulate

$$v = \sqrt{2}\frac{x - y}{||x - y||_2},$$

where

$$y = \begin{pmatrix} -sgn(x_1)||x||_2 & 0 & \cdots & 0 \end{pmatrix}$$

Individually check each element of v from i to n. If a nonzero element is found, set flag *skip* to false and break from loop. Otherwise, *skip* remains true.

If *skip* is false do

Let

$$A \leftarrow A - Avv^T.$$

This is the right Householder transformation result.

$$V \leftarrow V - Vvv^T.$$

end.

end.

Check if *num* is a positive number. If not, break.

Until $i = n$

Formulate left Householder transformations as above.

Let

$$U \leftarrow U - vv^T U.$$

end.

Let $B =$ the first n rows and n columns of A.

$U \leftarrow U^T$.

Return U, B, and V.

2.1.2 Bidiagonalization using the Lanczos Algorithm

If A is sparse, computations applied to it should take advantage of this property to minimize the amount of computational work. Term-by-document matrices are sparse. Householder transformations destroy sparsity and are therefore very inefficient when bidiagonalizing sparse matrices. The fill-in which results after only one Householder transformation causes the matrix to no longer be considered sparse [9].

The Lanczos algorithm, developed by Cornelius Lanczos (1893-1974), uses the sparsity property of a matrix when bidiagonalizing. Instead of computing B by continually updating A, the Lanczos algorithm computes B directly. It bidiagonalizes sparse matrices while accurately preserving singular values of the largest values. However, the algorithm only approximately bidiagonalizes A because U and V are not exactly orthogonal. Moreover, the smallest singular values are not well approximated [5]. However, since LSI uses only a rank-k approximation of the matrix A, only the largest singular values are important. Therefore, the Lanczos algorithm is a good choice for this application.

The algorithm below can be extracted from the following idea:

Suppose

$$U = \begin{pmatrix} u_1 & u_2 & \cdots & u_m \end{pmatrix},$$

$$V = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix},$$

where $u_1, u_2, \cdots, u_m$ are vectors of size $m \times 1$ and have unit norm, and $v_1, v_2, \ldots, v_n$ are vectors of size $n \times 1$ and have unit norm.

The Lanczos algorithm computes values $\alpha_1, \alpha_2, \cdots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_{n-1}$ of B such that

$$U^T AV = B = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & \alpha_n \end{pmatrix}. \tag{2.1}$$

From (2.1) two equations for α_k and β_k can be seen:

$$\alpha_k = u_k^T A v_k, \qquad k = 1, 2, ..., n$$

and

$$\beta_k = u_k^T A v_{k+1}, \qquad k = 1, 2, ..., n-1$$

The following two manipulations of (2.1) allow us to obtain two further useful relations between the desired quantities α_k and β_k and the left and right singular vectors u_k and v_k. These relations are the basis of the Lanczos algorithm [5].

1. $AV = UB$

$$A \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} = \begin{pmatrix} u_1 & u_2 & \cdots & u_m \end{pmatrix} \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & \alpha_n \end{pmatrix}$$

$$\Longrightarrow \quad Av_k = \beta_{k-1}u_{k-1} + \alpha_k u_k, \qquad k = 1, 2, ..., n.$$

Solving for $\alpha_k u_k$, we get

$$\alpha_k u_k = Av_k - \beta_{k-1}u_{k-1}, \qquad k = 1, 2, ..., n.$$

Taking the norm of both sides, since each column of U and V have unit norm, we get a formula for α_k:

$$\alpha_k = \|Av_k - \beta_{k-1}u_{k-1}\|_2, \qquad k = 1, 2, ..., n.$$

2. $A^T U = VB^T$

$$A^T \begin{pmatrix} u_1 & u_2 & \cdots & u_m \end{pmatrix} = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} \begin{pmatrix} \alpha_1 & & & & \\ \beta_1 & \alpha_2 & & & \\ & \beta_2 & \ddots & & \\ & & \ddots & \alpha_{n-1} & \\ & & & \beta_{n-1} & \alpha_n \end{pmatrix}$$

$$A^T u_k = \alpha_k v_k + \beta_k v_{k+1}, \qquad k = 1, 2, ..., n-1.$$

Solving for $\beta_k v_{k+1}$ we get

$$\beta_k v_{k+1} = A^T u_k - \alpha_k v_k, \qquad k = 1, 2, ..., n-1.$$

Again, since each column of U and V has unit norm, taking the norm of both sides gives us

$$\beta_k = \|A^T u_k - \alpha_k v_k\|_2, \qquad k = 1, 2, ..., n-1.$$

The complete Lanczos algorithm can be described as follows.

Without loss of generality, one can choose the following initializations for the vectors v_1, p_0, and u_0 and for the constant β_0.

Set a *flag* to true.

Let $\mathbf{1}$ be an $n \times 1$ vector of ones.

Let $v_1 = \frac{\mathbf{1}}{\sqrt{n}}$.

Let $p_0 = v_1$.

Let $\beta_0 = 1$.

Let $j = 0$.

Let u_0 be an $m \times 1$ zero vector.

While $j < n$ and *flag* is true

If $\beta_j \neq 0$ then

Let

$$v_{j+1} = \frac{p_j}{\beta_j}.$$

Increment j.

Let

$$\begin{aligned}
r_j &= Av_j - \beta_{j-1}u_{j-1} \\
\alpha_j &= \|r_j\|_2 \\
u_j &= \frac{r_j}{\alpha_j} \\
p_j &= A^T u_j - \alpha_j v_j \\
\beta_j &= \|p_j\|_2.
\end{aligned}$$

Else

Set *flag* to false.

end.

end.

Let $U = I_m$.

Let $V = I_n$.

Let $i = 1$.

Until $i = m + 1$

Column i of $U = u_i$.

Increment i.

end.

Reset i to 1.

Until $i = n + 1$

Column i of $V = v_i$.

Increment i.

end.

Reset i to 1.

Until $i = n$

Let $B(i, i) = \alpha_i$.

Let $B(i, i+1) = \beta_i$.

Increment i.

end.

Let $B(n, n) = \alpha_n$.

Return U, B, and V.

2.1.3 A Comparison of Bidiagonalization Methods

We now give a comparison in terms of CPU times for Householder transformations and the Lanczos algorithm to bidiagonalize a term-by-document matrix generated by the text corpus from Section 1.2.

Example 3:

Suppose we wish to search the graphics newsgroup text corpus mentioned in Section 1.2 for documents relevant to the following 5 terms:

t_1: printer

t_2: colour/color

t_3: laser

t_4: Hewlett

t_5: Packard

A sparse 5×1534 term-by-document matrix is formed with 16% density. The bidiagonalization step of the SVD computation is performed by the Lanczos algorithm and through the use of Householder transformations. The following table displays the fastest CPU times of five different trials. These programs were run on a Pentium 4, with a clock speed 1.6 GHz and 1 GB of RAM. The operating system was Linux Slackware 8.0.

	Lanczos (s)	Householder (s)
Bidiagonalization Step	0.1678	2.0829
Iterative Step	0.0029	0.0026
Total	0.1707	2.0855

Table 2.1: CPU Bidiagonalization Times using the Lanczos Algorithm and Householder Transformations

The iterative step will be defined in Section 2.3.

There is a cost savings of a factor of approximately 12. Since in this case the bidiagonalization step represents the vast majority of the total computation time for SVD, using Lanczos significantly decreases the computing time for SVD calculations. However, we recall that there is a trade-off: the small singular values are not well approximated. This can be seen by the following results from our text corpus.

Exact Singular Values:

$$\begin{pmatrix} 68.0089 \\ 31.0670 \\ 15.9013 \\ 8.1806 \\ 1.6910 \end{pmatrix}$$

Singular Values Calculated by Lanczos:

$$\begin{pmatrix} 68.0089 \\ 31.0670 \\ 15.9012 \\ 8.1783 \\ 0.0670 \end{pmatrix}$$

However, since LSI uses a vector subspace approximation and not the entire vector space, only the dominant singular values are used.

In this example, it seems we could use subspaces of size 1,2,3, or 4 (up to 80% of the vector space) in an LSI context. Although we do not anticipate that such a large fraction of the singular values will always be well approximated by the Lanczos algorithm, we are confident that the limits would be comparable, if not exceed, the typical requirements of LSI [3].

2.2 The Golub-Kahan SVD Step

We now give a detailed description of the iterative phase of the SVD calculation. The main part of this phase is known as the Golub-Kahan SVD step, named after Gene Golub and William Kahan who fist implemented this algorithm in 1965. See also

Section 2.3

The input is a bidiagonal matrix $B \in \Re^{n \times n}$, where n is the number of columns the original matrix $A \in \Re^{m \times n}$ (from the bidiagonalization phase), containing no zeros on the diagonal or super-diagonal. The output is an a bidiagonal matrix, stored in the overwritten original B, and two orthogonal matrices U and V, where $B \leftarrow U^T BV$ [5].

Let $T = B^T B$.

Initialize U and V to be $n \times n$ identity matrices.

If $n > 1$

Calculate the two eigenvalues of the trailing 2×2 submatrix of T.

Let $\mu =$ the eigenvalue whose value is closest to $T(n, n)$.

Let $y = T(1, 1) - \mu$ and $z = T(1, 2)$ (see [5]).

Let $k = 1$.

Until $k = n - 1$

Let

$$c = \frac{y^2}{y^2 + z^2}$$

and

$$s = \frac{-zc}{y}.$$

We are simply finding the values for a Givens rotation, G, which will zero out $B(k, k+1)$. G has the form

$$G = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & c & 0 & \cdots & 0 & s & & & \\ & & & 0 & 1 & & & 0 & & & \\ & & & \vdots & & \ddots & & \vdots & & & \\ & & & 0 & & & 1 & 0 & & & \\ & & & -s & 0 & \cdots & 0 & c & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix}, \qquad (2.2)$$

where c is in positions (k, k) and $(k+1, k+1)$, s is in position $(k, k+1)$, and $-s$ is in position $(k+1, k)$ [6].

Let $B \leftarrow BG$.

Let $V \leftarrow VG$.

Let $y = B(k, k)$ and $z = B(k+1, k)$.

Calculate c and s as before and form G as in (2.2).

Let $B \leftarrow G^T B$.

Let $U \leftarrow G^T B$.

if $k + 1 < n$

  Let $y = B(k, k+1)$ and $z = B(k, k+2)$.

end.

end.

Let $U \leftarrow U^T$.

end.

Return B, U, and V.

2.3 The SVD Algorithm

This algorithm was developed by Golub and Kahan and was presented in [4] in 1965. The basic idea is that the super-diagonal elements will converge to within a predetermined tolerance ϵ. Once they become small enough, they are set to zero and the matrix is re-partitioned as described below.

Initialize q to zero.

Until $q = n$

If $|A(i, i+1)| \leq \epsilon(|A(i,i)| + |A(i+1, i+1)|)$ for $i = 1, 2, ..., n-1$, then let $|A_(i, i+1)| = 0$.

Initialize indices i, p, r, and q to zero.

Decompose B as follows:

$$\begin{pmatrix} B_{11} & & \\ & B_{22} & \\ & & B_{33} \end{pmatrix}$$

.

Find the largest integer value q such that B_{22} is a $q \times q$ diagonal matrix.

Until $i = n - 2$ or $B(n - i - 1, n - i) > \epsilon$

Increment i.

Increment q.

end.

Find the largest integer value r such that B_{33} is a $r \times r$ bidiagonal matrix whose super-diagonal entries are non-zero as shown below.

Until $i = n - 2$ or $B(n - i - 1, n - i) \leq \epsilon$

Increment i.

Increment r.

end.

Let $p = n - q - r$, which also happens to be the number of rows and columns of B_{11}.

If $q < n$

If any diagonal entry $B_{22}(i, i) = 0$, where $1 \leq i < r$, then the super-diagonal entry which lies on the same row must also be zeroed out in the following fashion:

Set a *flag* to false. If B_{22} is changed, it must be recalculated before continuing on in the loop.

Initialize k to 0.

Let $num = \min(r - i, r - 2)$.

Until $k = num$

Let

$$c = \frac{B_{22}(i+1+k, i+1+k)}{\sqrt{B_{22}(i, i+1+k)^2 + B_{22}(i+1+k, i+1+k)^2}}$$

and

$$s = c\frac{B_{22}(i, i+1+k)}{B_{22}(i+1+k, i+1+k)}.$$

We are simply finding the values for a Givens rotation, G, which will zero out $B_{22}(i+k, i+k+1)$. G has the form

$$G = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & c & 0 & \cdots & 0 & -s & & & \\ & & & 0 & 1 & & & 0 & & & \\ & & & \vdots & & \ddots & & \vdots & & & \\ & & & 0 & & & 1 & 0 & & & \\ & & & s & 0 & \cdots & 0 & c & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix},$$

where c is in positions (i, i) and $(i + k + 1, i + k + 1)$, $-s$ is in position $(i, i + 1)$, and s is in position $(i + k + 1, i)$ [6].

Let $B \leftarrow GB$.

Let $U \leftarrow GU$.

Increment k.

end.

Let $U \leftarrow U^T$.

end.

If $B_{22}(r, r) = 0$, then the super-diagonal entry which lies on the same column must be zeroed out in the following fashion:

In this situation a flag must also be set. If B_{22} is changed, it must be recalculated before continuing on in the loop.

Initialize k to 0.

Until $k = r - 1$

Let

$$c = \frac{B_{22}(i+1+k, i+1+k)}{\sqrt{B_{22}(i, i+1+k)^2 + B_{22}(i+1+k, i+1+k)^2}}$$

and

$$s = c\frac{B_{22}(i, i+1+k)}{B_{22}(i+1+k, i+1+k)}.$$

We are simply finding the values for a Givens rotation, G, which will zero out $B_{22}(r, r)$. G has the same form as in the previous case, however, c is in positions $(r-k-1, r-k-1)$ and (r, r), s is in position $(r-k-1, r)$, and $-s$ is in position $(r, r-k-1)$.

$B \leftarrow BG$.

$V \leftarrow VG$.

Increment k.

end.

Let B be the B calculated above stripped of its last row and column which consist only of zeros.

end.

If B_{22} does not contain any zero diagonal entries, i.e., the flag is true, then continue.

Apply the Golub-Kahan SVD step described in section 2.

Let B_{22} in B be replaced by the matrix returned in the previous step.

end.

end.

end.

The iteration process continues until B converges to a diagonal matrix Σ to within tolerance ϵ. The rate of convergence is cubic [5].

Return Σ, U, and V.

Chapter 3

SVD Downdating

Downdating occurs when documents or terms are removed from a database or a query. For example, parental screening of sites and filtering of information can be accomplished through downdating. Also, when a term or a document is removed, the importance of the term or document within the query can be evaluated by the clustering or declustering which occurs. The change in clustering and declustering is an indication of the sensitivity of the database to the term or document [10]. For example, if the clustering changes significantly after the removal of a document, we say that the database is sensitive to that document.

Updating, as the name suggests, occurs when documents or terms are added to a database or query. Adding terms is often performed to refine a given search. Although updating will not be explored any further in this thesis, the process can be derived similarly from the downdating process.

Often, documents in a database become obsolete and are removed. In a vector space model, this corresponds to removing one or more columns from a term-by-document matrix formed by a user-specified query. If the user has previously calculated the SVD of the term-by-document matrix and wishes to search the modified database by the same query, we would like to avoid recalculating the SVD from scratch because of the nontrivial computational cost. The original SVD can be efficiently modified to calculate the SVD of the modified matrix [10]. Park and Van Huffel have developed a one-way chasing scheme and a two-way chasing scheme to bidiagonalize the SVD of a downdated matrix [8]. Once the matrix is bidiagonalized, it is reduced to a diagonal form by the Golub-Kahan iterative step.

Suppose we wish to eliminate one document from our database. By permuting the appropriate columns of A, U, and V, the document in question can be positioned as the first column of A. Let this column be denoted as x. Therefore, without loss of generality, we wish to compute the SVD of $\hat{A}$ where

$$A = \begin{pmatrix} x & \hat{A} \end{pmatrix}.$$

Here, $A \in \Re^{m \times n}$ where $m < n$, $x \in \Re^{m \times 1}$, $\hat{A} \in \Re^{m \times (n-1)}$, and U, Σ, and V are given such that

$$A = U \begin{pmatrix} \Sigma & 0 \end{pmatrix} V^T, \tag{3.1}$$

where $U \in \Re^{m \times m}$, $\Sigma \in \Re^{m \times m}$, and $V \in \Re^{n \times n}$.

3.1 One-way Chasing Scheme

Park and Van Huffel describe a one-way chasing scheme in [8] to efficiently compute the SVD of an over-determined matrix whose first row has been removed. The algorithm assumes $A \in \Re^{m\times n}$ where $m > n$. Although term-by-document matrices are normally underdetermined, they can be transposed and their singular vectors switched before and after the algorithm. For our purposes, we will also assume that the rank of the matrix is n.

Now, suppose $A \in \Re^{m\times n}$ where $m > n$. The equivalent of (3.1) when A is over-determined is the following:

$$A = \begin{pmatrix} x^T \\ \hat{A} \end{pmatrix} = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T = U_n \Sigma_n V^T, \tag{3.2}$$

where $U_n \in \Re^{m\times n}$ and $V \in \Re^{n\times n}$ are composed of the first n columns of U and V respectively. $\Sigma_n \in \Re^{n\times n}$ is composed of the first n rows of Σ. Rearranging (3.2) gives us

$$AV = \begin{pmatrix} x^T V \\ \hat{A} V \end{pmatrix} = U_n \Sigma_n, \tag{3.3}$$

where x is the first row of A, $\hat{A} \in \Re^{(m-1)\times n}$, $U_n \in \Re^{m\times n}$ is orthogonal, and $V \in \Re^{n\times n}$ is also orthogonal.

It is shown in [8] that there exist orthogonal matrices $F \in \Re^{(n+1)\times(n+1)}$ and $G \in \Re^{(n+1)\times(n+1)}$ such that

$$F^T \begin{pmatrix} u_n & \Sigma_n \\ \gamma & 0 \end{pmatrix} G = \begin{pmatrix} 1 & w_n^T \\ 0 & B_n \end{pmatrix} \tag{3.4}$$

and

$$G = \begin{pmatrix} 1 & 0 \\ 0 & \hat{G} \end{pmatrix}, \tag{3.5}$$

where u_n^T is the first row of U_n, $\gamma = \sqrt{1 - u_n^T u_n}$, $w_n \in \Re^{n\times 1}$, and $B_n \in \Re^{n\times n}$ is a bidiagonal matrix.

Combining (3.2) and (3.4) gives us

$$F^T U_n^T \begin{pmatrix} e_1 & A_n V \end{pmatrix} G = \begin{pmatrix} 1 & w^T \\ 0 & B \end{pmatrix}. \tag{3.6}$$

The form of $F^T U_n^T$ is

$$F^T U_n^T = \begin{pmatrix} 1 & 0 \\ 0 & \hat{U}_n^T \end{pmatrix}, \tag{3.7}$$

where $\hat{U} \in \Re^{(n-1)\times(n-1)}$ is an orthogonal matrix.

Equations (3.5), (3.6), and (3.7) yield

$$F^T U_n^T \begin{pmatrix} e_1 & A_n V \end{pmatrix} G = \begin{pmatrix} 1 & 0 \\ 0 & \hat{U}_n^T \end{pmatrix} \begin{pmatrix} e_1 & A_n V \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \hat{G} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & \hat{U}_n^T \end{pmatrix} \begin{pmatrix} 1 & x_n^T V \\ 0 & \hat{A}_n V \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \hat{G} \end{pmatrix} = \begin{pmatrix} 1 & x_n^T V \hat{G} \\ 0 & \hat{U}^T \hat{A} V \hat{G} \end{pmatrix} = \begin{pmatrix} 1 & w_n^T \\ \gamma & B_n \end{pmatrix},$$

where e_i is the ith column of the identity matrix I_n, $i = 1, 2, ..., n$.

Therefore, the following factorization for $\hat{A}^T$ holds:

$$\hat{A} = \hat{U}_n \begin{pmatrix} B_n \\ 0 \end{pmatrix} \hat{V}_n^T,$$

where $\hat{V} = V\hat{G}$.

Hence, we can represent the downdated version of A as D where

$$D \equiv \begin{pmatrix} u_n & \Sigma_n \\ \gamma & 0 \end{pmatrix}.$$

Once D has been formed, it can be bidiagonalized in an efficient way before being sent to the iterative Kahan-Golub step. Givens rotations, named L and R for left and right, are formed to chase away the elements of the first column down the superdiagonal. Fill-in which appears on the diagonal or the first row do not need to be chased away for only the bidiagonal below the first row will be needed.

The plane rotation sequence can be represented as [8]

$$
\begin{array}{llllllll}
L_n & & & & & & & \\
L_{n-1}, & R_n, & L_n & & & & & \\
\ldots & & & & & & & \\
L_2, & R_3, & L_3, & R_4, & L_4, & \ldots & R_n, & L_n \\
L_1 & & & & & & &
\end{array}
$$

The following is the bidiagonalization below the first row of D of dimension 5×5. $\times$ represents potentially non-zero elements. $+$ represents fill-in. Circled elements are to be zeroed out by the next Givens rotations [8].

$$
\begin{array}{c}
\\ \\ \\ \rightarrow \\ \rightarrow
\end{array}
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\times & & & \times & \\
\times & & & & \times \\
\otimes & & & &
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\times & & & \times & \\
\times & & & & \times \\
 & & & & +
\end{pmatrix}
$$

$$
\begin{array}{cc}
\begin{array}{c}
\\ \\ \rightarrow \\ \rightarrow \\ \\ \\
\end{array}
&
\begin{array}{c}
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\times & & & \times & \\
\otimes & & & & \times \\
 & & & & \times
\end{pmatrix}
\\
\begin{array}{ccccc}
 & & & \downarrow & \downarrow
\end{array}
\end{array}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\times & & & \times & + \\
 & & & + & \times \\
 & & & & \times
\end{pmatrix}
$$

$$
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\times & & & \times & \otimes \\
& & & \times & \times \\
& & & & \times
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\times & & & \times & \\
& & & \times & \times \\
& & & + & \times
\end{pmatrix}
$$

$$
\begin{array}{c}
\\ \\ \\ \rightarrow \\ \rightarrow
\end{array}
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\times & & & \times & \\
& & & \times & \times \\
& & & \otimes & \times
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\times & & & \times & \\
& & & \times & \times \\
& & & & \times
\end{pmatrix}
$$

$$
\begin{array}{c}
\\ \rightarrow \\ \rightarrow \\ \\ \\
\end{array}
\begin{array}{c}
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
\otimes & & & \times & \\
& & & \times & \times \\
& & & & \times
\end{pmatrix}
\\
\begin{array}{ccccc}
 & & \downarrow & \downarrow &
\end{array}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & + & \\
& & + & \times & \\
& & & \times & \times \\
& & & & \times
\end{pmatrix}
$$

$$
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & \otimes & \\
& & \times & \times & \\
& & & \times & \times \\
& & & & \times
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & \\
\times & & \times & & \\
& & \times & \times & \\
& & + & \times & \times \\
& & & & \times
\end{pmatrix}
$$

$$\begin{array}{c} \\ \\ \rightarrow \\ \rightarrow \\ \\ \end{array} \begin{pmatrix} \times & \times & & & \\ \times & & \times & & \\ & & \times & \times & \\ & & \otimes & \times & \times \\ & & & & \times \end{pmatrix} \Longrightarrow \begin{pmatrix} \times & \times & & & \\ \times & & \times & & \\ & & \times & \times & + \\ & & & \times & \times \\ & & & & \times \end{pmatrix}$$

$$\begin{array}{ccccc} & & & \downarrow & \downarrow \end{array}$$

$$\begin{pmatrix} \times & \times & & & \\ \times & & \times & & \\ & & \times & \times & \otimes \\ & & & \times & \times \\ & & & & \times \end{pmatrix} \Longrightarrow \begin{pmatrix} \times & \times & & & \\ \times & & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & + & \times \end{pmatrix}$$

$$\begin{array}{c} \\ \\ \\ \rightarrow \\ \rightarrow \end{array} \begin{pmatrix} \times & \times & & & \\ \times & & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & \otimes & \times \end{pmatrix} \Longrightarrow \begin{pmatrix} \times & \times & & & \\ \times & & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix}$$

$$\begin{array}{c} \rightarrow \\ \rightarrow \\ \\ \\ \\ \end{array} \begin{pmatrix} \times & \times & & & \\ \otimes & & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & \times & + & & \\ & + & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix}$$

Only the 4×4 bottom right-hand block of this matrix is sent to the Kahan-Golub iterative step.

3.2 Two-Way Chasing Scheme

We have already examined a one-way chasing scheme which bidiagonalizes a diagonal submatrix bordered on the left by a column of non-zero entries. However, if the column were strategically placed in the "middle" of the diagonal submatrix, the matrix could be reduced to bidiagonal form in half the number of rotations. This is what is known as the two-way chasing scheme.

Again, Park and Van Huffel, [8], show that there exists orthogonal matrices $F \in \Re^{(n+1)\times(n+1)}$ and $G \in \Re^{(n+1)\times(n+1)}$ such that

$$F^T = \begin{pmatrix} u_n & \Sigma_n \\ \gamma & 0 \end{pmatrix} G = \begin{pmatrix} e_k & \begin{pmatrix} B_1 \\ w_n^T \\ B_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 & B_1 \\ 1 & w_n^T \\ 0 & B_2 \end{pmatrix} \tag{3.8}$$

where n,u_n, Σ_n, γ, and w_n^T are defined the same as they were in the one-way chasing scheme, $k = \lceil n/2 \rceil$, $B_1 \in \Re^{k\times n}$, and $B_2 \in \Re^{(n-k)\times k}$.

B_1 and B_2 have the following form:

$$B_1 = \begin{pmatrix} B_{11} & ce_k & O_{k\times(n-k-1)} \end{pmatrix}, \quad B_2 = \begin{pmatrix} O_{(n-k)\times k} & B_{22} \end{pmatrix},$$

where $B_{11} \in \Re^{k\times k}$ is a bidiagonal matrix, $B_{22} \in \Re^{(n-k)\times(n-k)}$ is also a bidiagonal matrix, and c is a constant.

Then,

$$\begin{pmatrix} F^T & 0 \\ 0 & I_{m-n-1} \end{pmatrix} \begin{pmatrix} u_n & \Sigma_n & 0 \\ \gamma & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} G & 0 \end{pmatrix}$$

$$= \begin{pmatrix} F^T & 0 \\ 0 & I_{m-n-1} \end{pmatrix} W^T U^T \begin{pmatrix} e_1 & AV \end{pmatrix} \begin{pmatrix} G & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & B_1 & 0 \\ 1 & w_n^T & 0 \\ 0 & B_2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.9}$$

for some orthogonal matrix W^T that does not operate on the first n rows.

G is similar to the G in the one-way chasing scheme for it does not operate on the first column of the matrix. Therefore,

$$\begin{pmatrix} F^T & 0 \\ 0 & I_{m-n-1} \end{pmatrix} W^T U^T e_1 = e_k.$$

So,

$$\begin{pmatrix} F^T & 0 \\ 0 & I_{m-n-1} \end{pmatrix} W^T U^T = \begin{pmatrix} 0 & \hat{U}_1^T \\ 1 & 0 \\ 0 & \hat{U}_2^T \end{pmatrix} \tag{3.10}$$

for some matrices $\hat{U}_1 \in \Re^{(m-1)\times k}$ and $\hat{U}_2 \in \Re^{(m-1)\times(m-k-1)}$ where

$$\begin{pmatrix} \hat{U}_1^T \\ \hat{U}_2^T \end{pmatrix}$$

is an orthogonal matrix.

From (3.2) and (3.9), we get

$$\begin{pmatrix} 0 & \hat{U}_1^T \\ 1 & 0 \\ 0 & \hat{U}_2^T \end{pmatrix} \begin{pmatrix} 1 & x^T V \\ 0 & \hat{A}V \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \hat{G} & 0 \end{pmatrix} = \begin{pmatrix} 0 & \hat{U}_1^T \hat{A} V \hat{G} & 0 \\ 1 & x^T v_r \hat{G} & 0 \\ 0 & \hat{U}_2^T \hat{A} V \hat{G} & 0 \end{pmatrix} \tag{3.11}$$

where $G \in \Re^{n \times n}$ is also an orthogonal matrix.

Hence,

$$\begin{pmatrix} \hat{U}_1^T \\ \hat{U}_2^T \end{pmatrix} \hat{A} V \begin{pmatrix} \hat{G} & 0 \end{pmatrix} = \begin{pmatrix} B_1 & 0 \\ B_2 & 0 \\ 0 & 0 \end{pmatrix}. \tag{3.12}$$

By permuting the columns of

$$\begin{pmatrix} \hat{u}_n & \Sigma_n \\ \hat{\gamma} & 0 \end{pmatrix}$$

appropriately, where $\hat{u}_n$ is the first row of $\hat{U}_n$, the following holds:

$$\begin{pmatrix} \hat{u}_n & \Sigma_n \\ \hat{\gamma} & 0 \end{pmatrix} \prod = \begin{pmatrix} \Sigma_{n1} & \hat{u}_{n1} & 0 \\ 0 & \hat{u}_{n2} & \Sigma_{n2} \\ 0 & \hat{\gamma} & 0 \end{pmatrix}, \tag{3.13}$$

where $\prod$ is the permutation matrix mentioned above. $\hat{u}_n = (\hat{u}_{n1} \quad \hat{u}_{n2})^T$, $\Sigma_{n1} = \text{diag}(\sigma_1, ..., \sigma_k)$ and $\Sigma_{n2} = \text{diag}(\sigma_{k+1}, ..., \sigma_n)$.

The following figures show how the column $(\hat{u}_{n1}\ \hat{u}_{n2}\ \hat{\gamma})^T$ of dimension 9×1 can be chased away by the two-way chasing scheme [8]:

$$
\begin{array}{c}
\to \\ \to \\ \\ \\ \\ \\ \\ \to \\ \to
\end{array}
\begin{pmatrix}
\times & & & & \otimes & & & & \\
& \times & & & \times & & & & \\
& & \times & & \times & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \times & & \times & & \\
& & & & \times & & & \times & \\
& & & & \times & & & & \times \\
& & & & \otimes & & & &
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
\times & + & & & & & & & \\
+ & \times & & & \times & & & & \\
& & \times & & \times & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \times & & \times & & \\
& & & & \times & & & \times & \\
& & & & \times & & & & \times \\
& & & & & & & & +
\end{pmatrix}
$$

$$
\downarrow \quad \downarrow
$$

$$
\begin{array}{c}
\\ \\ \\ \\ \\ \\ \to \\ \to \\ \\
\end{array}
\begin{pmatrix}
\times & \times & & & & & & & \\
\otimes & \times & & & \times & & & & \\
& & \times & & \times & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \times & & \times & & \\
& & & & \times & & & \times & \\
& & & & \otimes & & & & \times \\
& & & & & & & & \times
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & & & \times & & & & \\
& & \times & & \times & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \times & & \times & & \\
& & & & \times & & & \times & + \\
& & & & & & & + & \times \\
& & & & & & & & \times
\end{pmatrix}
$$

$$
\begin{array}{cc}
 & \begin{matrix} & & & & & & & \downarrow & \downarrow \end{matrix} \\
\begin{matrix} \\ \rightarrow \\ \rightarrow \\ \\ \\ \\ \\ \\ \\ \end{matrix} &
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & & & \otimes & & & & \\
& & \times & & \times & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \times & & \times & & \\
& & & & \times & & & \times & \otimes \\
& & & & & & & \times & \times \\
& & & & & & & & \times
\end{pmatrix}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & + & & & & & & \\
& + & \times & & \times & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \times & & \times & & \\
& & & & \times & & & \times & \\
& & & & & & & \times & \times \\
& & & & & & & + & \times
\end{pmatrix}
$$

$$
\begin{array}{cc}
 & \begin{matrix} & \downarrow & \downarrow & & & & & & \end{matrix} \\
\begin{matrix} \\ \\ \\ \\ \\ \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} &
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & \times & & & & & & \\
& \otimes & \times & & \times & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \times & & \times & & \\
& & & & \otimes & & & \times & \\
& & & & & & & \times & \times \\
& & & & & & & \otimes & \times
\end{pmatrix}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & + & & & & & & \\
& \times & \times & & & & & & \\
& & \times & & \times & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \times & & \times & + & \\
& & & & & & + & \times & \\
& & & & & & & \times & \times \\
& & & & & & & & \times
\end{pmatrix}
$$

$$
\begin{array}{cc}
 & \begin{array}{ccccccccc} & & & & & & \downarrow & \downarrow & \end{array} \\
\begin{array}{c} \rightarrow \\ \rightarrow \\ \\ \\ \\ \\ \\ \\ \\ \end{array} &
\begin{pmatrix}
\times & \times & \otimes & & & & & & \\
 & \times & \times & & & & & & \\
 & & \times & & \times & & & & \\
 & & & \times & \times & & & & \\
 & & & & \times & \times & & & \\
 & & & & \times & & \times & \otimes & \\
 & & & & & & \times & \times & \\
 & & & & & & & \times & \times \\
 & & & & & & & & \times
\end{pmatrix}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & & & & & \\
+ & \times & \times & & & & & & \\
 & & \times & & \times & & & & \\
 & & & \times & \times & & & & \\
 & & & & \times & \times & & & \\
 & & & & \times & & \times & & \\
 & & & & & & \times & \times & \\
 & & & & & & + & \times & \times \\
 & & & & & & & & \times
\end{pmatrix}
$$

$$
\begin{array}{cc}
 & \begin{array}{ccccccccc} \downarrow & \downarrow & & & & & & & \end{array} \\
\begin{array}{c} \\ \\ \\ \\ \\ \\ \rightarrow \\ \rightarrow \\ \\ \end{array} &
\begin{pmatrix}
\times & \times & & & & & & & \\
\otimes & \times & \times & & & & & & \\
 & & \times & & \times & & & & \\
 & & & \times & \times & & & & \\
 & & & & \times & \times & & & \\
 & & & & \times & & \times & & \\
 & & & & & & \times & \times & \\
 & & & & & & \otimes & \times & \times \\
 & & & & & & & & \times
\end{pmatrix}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & & & & & \\
 & \times & \times & & & & & & \\
 & & \times & & \times & & & & \\
 & & & \times & \times & & & & \\
 & & & & \times & \times & & & \\
 & & & & \times & & \times & & \\
 & & & & & & \times & \times & + \\
 & & & & & & & \times & \times \\
 & & & & & & & & \times
\end{pmatrix}
$$

$$
\begin{array}{cc}
 & \begin{array}{ccccccccc} & & & & & & & \downarrow & \downarrow \end{array} \\
\begin{array}{c} \\ \\ \rightarrow \\ \rightarrow \\ \\ \\ \\ \\ \\ \end{array} &
\begin{pmatrix}
\times & \times & & & & & & & \\
 & \times & \times & & & & & & \\
 & & \times & & \otimes & & & & \\
 & & & \times & \times & & & & \\
 & & & & \times & \times & & & \\
 & & & & \times & & \times & & \\
 & & & & & & \times & \times & \otimes \\
 & & & & & & & \times & \times \\
 & & & & & & & & \times
\end{pmatrix}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & & & & & \\
 & \times & \times & & & & & & \\
 & & \times & + & & & & & \\
 & & + & \times & \times & & & & \\
 & & & & \times & \times & & & \\
 & & & & \times & & \times & & \\
 & & & & & & \times & \times & \\
 & & & & & & & \times & \times \\
 & & & & & & & + & \times
\end{pmatrix}
$$

$$
\begin{array}{cc}
 & \begin{array}{ccccccccc} & & \downarrow & \downarrow & & & & & \end{array} \\
\begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \rightarrow \\ \rightarrow \end{array} &
\begin{pmatrix}
\times & \times & & & & & & & \\
 & \times & \times & & & & & & \\
 & & \times & \times & & & & & \\
 & & \otimes & \times & \times & & & & \\
 & & & & \times & \times & & & \\
 & & & & \times & & \times & & \\
 & & & & & & \times & \times & \\
 & & & & & & & \times & \times \\
 & & & & & & & \otimes & \times
\end{pmatrix}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & & & & & \\
 & \times & \times & + & & & & & \\
 & & \times & \times & & & & & \\
 & & & \times & \times & & & & \\
 & & & & \times & \times & & & \\
 & & & & \times & & \times & & \\
 & & & & & & \times & \times & \\
 & & & & & & & \times & \times \\
 & & & & & & & & \times
\end{pmatrix}
$$

$$
\begin{array}{cc}
\begin{array}{c}
\\ \rightarrow \\ \rightarrow \\ \\ \rightarrow \\ \rightarrow \\ \\ \\ \\
\end{array}
&
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & \times & \otimes & & & & & \\
& & \times & \times & & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & \otimes & & \times & & \\
& & & & & & \times & \times & \\
& & & & & & & \times & \times \\
& & & & & & & & \times
\end{pmatrix}
\end{array}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & \times & & & & & & \\
& + & \times & \times & & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & + & & \\
& & & & & + & \times & & \\
& & & & & & \times & \times & \\
& & & & & & & \times & \times \\
& & & & & & & & \times
\end{pmatrix}
$$

$$
\begin{array}{ccccccccc}
& \downarrow & \downarrow & & & \downarrow & \downarrow & &
\end{array}
$$

$$
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & \times & & & & & & \\
& \otimes & \times & \times & & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & \otimes & & \\
& & & & & \times & \times & & \\
& & & & & & \times & \times & \\
& & & & & & & \times & \times \\
& & & & & & & & \times
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
\times & \times & + & & & & & & \\
& \times & \times & & & & & & \\
& & \times & \times & & & & & \\
& & & \times & \times & & & & \\
& & & & \times & \times & & & \\
& & & & & \times & \times & & \\
& & & & & + & \times & \times & \\
& & & & & & & \times & \times \\
& & & & & & & & \times
\end{pmatrix}
$$

$$\begin{array}{cc}\begin{array}{c}\rightarrow\\ \rightarrow\\ \\ \\ \\ \rightarrow\\ \rightarrow\\ \\ \\ \end{array} & \begin{pmatrix} \times & \times & \otimes & & & & & & \\ & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & \times & & & & \\ & & & & \times & \times & & & \\ & & & & & \times & \times & & \\ & & & & & \otimes & \times & \times & \\ & & & & & & & \times & \times \\ & & & & & & & & \times \end{pmatrix}\end{array} \Longrightarrow \begin{pmatrix} \times & \times & & & & & & & \\ + & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & \times & & & & \\ & & & & \times & \times & & & \\ & & & & & \times & \times & + & \\ & & & & & & \times & \times & \\ & & & & & & & \times & \times \\ & & & & & & & & \times \end{pmatrix}$$

$$\begin{array}{c}\begin{array}{ccccccccc}\downarrow & \downarrow & & & & & \downarrow & \downarrow & \end{array}\\ \begin{pmatrix} \times & \times & & & & & & & \\ \otimes & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & \times & & & & \\ & & & & \times & \times & & & \\ & & & & & \times & \times & \otimes & \\ & & & & & & \times & \times & \\ & & & & & & & \times & \times \\ & & & & & & & & \times \end{pmatrix}\end{array} \Longrightarrow \begin{pmatrix} \times & \times & & & & & & & \\ & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & \times & & & & \\ & & & & \times & \times & & & \\ & & & & & \times & \times & & \\ & & & & & & \times & \times & \\ & & & & & & + & \times & \times \\ & & & & & & & & \times \end{pmatrix}$$

$$\begin{array}{c} \\ \\ \\ \rightarrow \\ \rightarrow \\ \\ \rightarrow \\ \rightarrow \\ \\ \end{array}\begin{pmatrix} \times & \times & & & & & & & \\ & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & \otimes & & & & \\ & & & & \times & \times & & & \\ & & & & & \times & \times & & \\ & & & & & & \times & \times & \\ & & & & & & \otimes & \times & \times \\ & & & & & & & & \times \end{pmatrix} \Longrightarrow \begin{pmatrix} \times & \times & & & & & & & \\ & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & & + & & & \\ & & & + & \times & \times & & & \\ & & & & & \times & \times & & \\ & & & & & & \times & \times & + \\ & & & & & & & \times & \times \\ & & & & & & & & \times \end{pmatrix}$$

$$\downarrow \quad \downarrow$$

$$\begin{pmatrix} \times & \times & & & & & & & \\ & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & & \times & & & \\ & & & \times & \times & \times & & & \\ & & & & & \times & \times & & \\ & & & & & & \times & \times & \otimes \\ & & & & & & & \times & \times \\ & & & & & & & & \times \end{pmatrix} \Longrightarrow \begin{pmatrix} \times & \times & & & & & & & \\ & \times & \times & & & & & & \\ & & \times & \times & & & & & \\ & & & \times & & \times & & & \\ & & & \times & \times & \times & & & \\ & & & & & \times & \times & & \\ & & & & & & \times & \times & \\ & & & & & & & \times & \times \\ & & & & & & & + & \times \end{pmatrix}$$

$$
\begin{array}{c}
\\ \\ \\ \\ \\ \\ \\ \rightarrow \\ \rightarrow
\end{array}
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & \times & & & & & & \\
& & \times & \times & & & & & \\
& & & \times & & \times & & & \\
& & & \times & \times & \times & & & \\
& & & & & \times & \times & & \\
& & & & & & \times & \times & \\
& & & & & & & \times & \times \\
& & & & & & & \otimes & \times
\end{pmatrix}
\Longrightarrow
\begin{pmatrix}
\times & \times & & & & & & & \\
& \times & \times & & & & & & \\
& & \times & \times & & & & & \\
& & & \times & & \times & & & \\
& & & \times & \times & \times & & & \\
& & & & & \times & \times & & \\
& & & & & & \times & \times & \\
& & & & & & & \times & \times \\
& & & & & & & & \times
\end{pmatrix}
$$

The fifth row is removed from the above matrix producing an 8×8 bidiagonal matrix. Only the 8×8 matrix is sent to the Kahan-Golub iterative step.

3.3 Comparing the Lanczos Algorithm, the One-Way Chasing Scheme, and the Two-way Chasing Scheme

Example 4:

Example 4 is a continuation of Example 3. Suppose we have already computed the SVD, $A = U\Sigma V^T$, of the term-by-document matrix. Further suppose that now the document 1298, represented by the column vector

$$
\begin{pmatrix} 2 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}
$$

has been removed from the text corpus. Let $\hat{A}$ represent the modified term-by-document matrix.

The table below represents the time, in seconds, it takes to

i) bidiagonalize with Lanczos

ii) bidiagonalize with the one-way chasing scheme

iii) bidiagonalize with the two-way chasing scheme

Lanczos (s)	One-Way (s)	Two-Way (s)
0.1721	0.0097	0.0054

Table 3.1: CPU Times to Bidiagonalize a Downdated Matrix Using the Lanczos Algorithm, the One-Way Chasing Scheme, and Two-Way Chasing Schemes

The above experiments were run on the same computer mentioned in Section 2.1.3. Bidigaonalization of the downdated 5×1533 matrix with the one-way chasing scheme is over 17 times faster than with the Lanczos algorithm. Bidiagonalization of the downdated 5×1533 matrix with the two-way chasing scheme is 1.8 times faster than with the one-way chasing scheme.

Chapter 4

Conclusion

It is important to be able to efficiently retrieve information from a database. The goal of this thesis was to show how the SVD calculations for information retrieval techniques such as LSI can be made more efficient, hence causing information retrieval to be significantly more efficient.

If only the dominant singular values of a sparse matrix are needed, an algorithm that takes advantage of the sparsity such as the Lanczos algorithm should be used instead of Householder transformations for bidiagonalization. In Example 2, where the term-by-document matrix is of dimension 5×1534, the Lanczos algorithm is more than 12 times more efficient than Householder transformations. The bidiagonalization step creates a 5×5 bidiagonal matrix. Since the cost of the iterative step of a 5×5 matrix is negligible compared to the bidiagonalization of a 5×1534 matrix, the complete SVD calculation cost decreases by a factor of 12. In other words, all of the computational savings of the SVD algorithm come from the saving of the bidiagonalization step.

Given $A \in \Re^{m \times n}$, $m < n$ and $rank = n$, we give algorithms for one-way and two-way chasing schemes for downdating A when a document is removed. For the same example as above, the one-way chasing scheme is 17 times more efficient than the one-way chasing scheme and the two-way chasing scheme is 1.8 times faster than the one-way chasing scheme.

Since the SVD calculation time makes up the bulk of the total LSI time, we have shown that it is possible to retrieve information with LSI more efficiently.

All algorithms used throughout this thesis were implemented in *Matlab*. These include bidiagonalization using Householder transformations, the Lanczos algorithm, the Golub-Kahan iterative step, the one-way chasing scheme, and the two-way chasing scheme.

This thesis does not address how to efficiently remove a term, add a term, or add a document to a term-by-document matrix. We hope to further the research started in this thesis and implement algorithms in *Matlab* that will successfully complete the above three tasks. We also have yet to determine a threshold where it becomes less expensive to recompute the SVD from scratch than to continually update or downdate the term-by-document matrix.

Bibliography

[1] M. Berry, Z. Drmac, and E. Jessup, *Matrices, Vector Spaces, and Information Retrieval*, SIAM Review 41:2 (1999), 335–362.

[2] M. Berry, S. Dumais, and T. Letsche, *Computational Methods for Intelligent Information Access*, Presented at the Proceedings of Supercomputing, San Diego, 1995.

[3] M. Berry, S.Dumais, and G. O'Brien, *Using Linear Algebra for Intelligent Information Retrieval*, SIAM Review 37:4 (1995), 573–595.

[4] G. Golub and W. Kahan, *Calculating the Singular Value Decomposition and Pseudoinverse of a Matrix*, SIAM J. Numer. Anal. 2 (1965), 205–224.

[5] G. Golub and C. Van Loan, *Matrix Computations, Second Edition*, The Johns Hopkins University Press, Baltimore, 1989.

[6] W. Hager, *Applied Numerical Linear Algebra*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[7] G. O'Brien, *Information Management Tools for Updating an SVD-Encoded Indexing Scheme*, Masters Thesis, Department of Computer Science, University of Tennessee, Knoxville, 1994.

[8] H. Park and S. Van Huffel, *Two-Way Bidiagonalization Scheme for Downdating the Singular Value Decomposition*, Linear Algebra and its Applications, 222 (1995), 23–39.

[9] S. Watkins, *Fundamentals of Matrix Computations*, John Wiley & Sons, Inc., United States of America, 1991.

[10] D. Witter and M. Berry, *Downdating the Latent Semantic Indexing Modal for Conceptual Information Retrieval*, The Computer Journal 41 (1998), 589-601.