Numerical Linear Algebra with Applications, Vol. 1(1), 1–27 (1996)

Low-Rank Orthogonal Decompositions for Information Retrieval Applications

M.W. Berry and R.D. Fierro

Computer Science Department

CS-95-284 April 1995

1070–5325/96/010001–27$18.50

Received 25 April 1995
Revised 10 January 1996

Numerical Linear Algebra with Applications, Vol. 1(1), 1–27 (1996)

Low-Rank Orthogonal Decompositions for Information Retrieval Applications

Michael W. Berry

Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville TN 37996-1301, berry@cs.utk.edu

and

Ricardo D. Fierro

Department of Mathematics, California State University, San Marcos, CA 92096, fierro@thunder.csusm.edu

Current methods to index and retrieve documents from databases usually depend on a lexical match between query terms and keywords extracted from documents in a database. These methods can produce incomplete or irrelevant results due to the use of synonyms and polysemus words. The association of terms with documents (or implicit semantic structure) can be derived using large sparse *term-by-document* matrices. In fact, both terms and documents can be matched with user queries using representations in k-space (where $100 \leq k \leq 200$) derived from k of the largest approximate singular vectors of these term-by-document matrices. This completely automated approach called Latent Semantic Indexing or LSI, uses subspaces spanned by the approximate singular vectors to encode important associative relationships between terms and documents in k-space. Using LSI, two or more documents may be *close* to each other in k-space (and hence meaning) yet share no common terms. The focus of this work is to demonstrate the computational advantages of exploiting low-rank orthogonal decompositions such as the ULV (or URV) as opposed to the truncated singular value decomposition (SVD) for the construction of initial and updated rank-k subspaces arising from LSI applications.

KEY WORDS information, latent semantic indexing, low-rank, orthogonal, matrices, retrieval, singular value decomposition, sparse, ULV and URV decompositions, updating

1. Introduction

Information is commonly retrieved from documents using a literal match of terms found in documents with those of a user's query. There are two potential drawbacks

1070–5325/96/010001–27$18.50

Received 25 April 1995
Revised 10 January 1996

to these methods. First, there are usually many ways to express a given concept (synonymy), thus relevant documents may be ignored. Second, most words have multiple meanings (polysemy), thus irrelevant documents may be retrieved. These two drawbacks can render lexical matching methods inaccurate when they are used to match a user's query. A more effective approach would perhaps allow a user to retrieve information by conceptual topic or the meaning of a particular document.

Latent Semantic Indexing (LSI) [9] is an attempt to overcome the problems of lexical matching by using statistically derived conceptual indices instead of individual words for retrieval. LSI, which assumes there is some underlying or latent structure in word usage that is obscured by variability in word choice, gives rise to an $m \times n$ sparse *term-by-document* matrix that is generated from text parsing. The singular value decomposition (SVD), cf. [17], of the term-by-document matrix is commonly used to analyze the structure in word usage across documents. Retrieval can be performed using the k-largest singular values and corresponding singular vectors, where $k \ll \min(m,n)$. Performance data [9,10,16] indicates that singular vectors are in fact more robust indicators of meaning than individual terms. A number of software tools have been developed to perform operations such as parsing document texts, creating a term-by-document matrix, computing the truncated SVD of this matrix, creating the LSI database of singular values and vectors for retrieval, matching user queries to documents, and adding new terms or documents to a database [9,19]. However, the bulk of LSI processing time can be spent in computing the truncated SVD of the large sparse term-by-document matrices, especially when several new terms or documents are to be added to the database.

The SVD is the most common example of a *two-sided (or complete) orthogonal decomposition*, which is defined for a matrix as a product of three matrices: an orthogonal matrix, a middle matrix, and another orthogonal matrix. The middle matrix is usually either lower trapezoidal, upper trapezoidal, or diagonal. Although two-sided orthogonal decompositions have been around for some time [18], the *rank-revealing* property for trapezoidal middle matrices is recent [12], [20], [21].

The focus of this work is to demonstrate that alternative two-sided orthogonal decompositions can be used for LSI-based information retrieval at a reduced computational cost compared to the SVD. The main computational advantages of our method over other methods lie mainly in updating. This paper is organized as follows. Section 2 is a review of basic concepts needed to understand LSI. Section 3 is a discussion of the low-rank ULV algorithm with particular focus on computational complexity and ability to produce good approximations to the singular subspaces of sparse rectangular matrices. Section 4 uses a constructive example to illustrate how LSI can use the ULV decomposition to represent terms and documents in the same semantic space, how a query is represented, how additional documents are added (or folded-in), and how ULV-updating represents additional documents. Section 5 is a discussion of L-ULV updating, a procedure based on the L-ULV algorithm (which has not been previously considered in the literature). In particular, we give an algorithm for ULV-updating along with a comparison to the folding-in process with regard to robustness of query matching and computational complexity. Then, L-ULV updating is illustrated using a small example. Section 6 is a brief summary and considerations for future work.

2. Background

The SVD is commonly used in the solution of unconstrained linear least squares problems, matrix rank estimation, and canonical correlation analysis [2]. Although the SVD provides very accurate subspace information, it is computationally demanding and difficult to update for either dense [5] or sparse matrices [1,19]. This can be a drawback for recursive procedures which require simple matrix updates (e.g., appending or deleting a row or column).

Alternatively, rank-revealing QR (RRQR) algorithms such as those by Foster [15], Chan [6], and modifications [4] can be used to obtain subspace information from matrices [7], [8]. RRQR decompositions, however, yield subspaces whose accuracies depend on the gap in the singular values [13] in the sense that a *large* gap is required to produce good approximations to the singular subspaces. In LSI applications there is a *small* gap between the smallest singular value that is retained and the largest singular value that is discarded, hence an RRQR decomposition is not appropriate for LSI.

Recently, *low-rank* revealing ULV and URV algorithms [12] for computing good approximations to the principal singular subspaces associated with dense or sparse matrices have been designed. By low-rank we mean either the numerical rank of the matrix is much smaller than the dimensions of the matrix or a small number of parameters suffice to describe a system or model [2,12], such as in LSI applications. These algorithms provide reliable rank detection while avoiding the loss of orthogonality associated with Lanczos bidiagonalization [17], and may be applied without altering the original matrix A (i.e., quite suitable for sparse matrices).

2.1. Low-Rank Orthogonal Decompositions

Given an $m \times n$ matrix A, where without loss of generality $m \geq n$ and $\text{rank}(A) = r$, the singular value decomposition of A is given by

$$A = \bar{U} \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} \bar{V}^T = \bar{U}_1 \Sigma \bar{V}^T = \sum_{i=1}^{n} \bar{u}_i \sigma_i \bar{v}_i^T \ , \tag{2.1}$$

where $\bar{U}^T\bar{U} = I_m$, $\bar{V}^T\bar{V} = I_n$, and $\Sigma = \text{diag}(\sigma_1, \cdots, \sigma_n), \sigma_i > 0$ for $1 \leq i \leq r, \sigma_j = 0$ for $j \geq r+1$. The first r columns of the orthogonal matrices $\bar{U}$ and $\bar{V}$ define the orthonormal eigenvectors associated with the r nonzero eigenvalues of AA^T and A^TA, respectively. The i-th column of $\bar{U}$, denoted by $\bar{u}_i$, is referred to as the left singular vector corresponding to the i-th largest singular value, σ_i. Similarly, the i-th column of $\bar{V}$, denoted by $\bar{v}_i$, is referred to as the right singular vector corresponding to σ_i. The set $\{\bar{u}_i, \sigma_i, \bar{v}_i\}$ is referred to as the i-th largest singular triplet of matrix A. $\bar{U}_k$ and $\bar{V}_k$ denote submatrices consisting of the first k columns of $\bar{U}$ and $\bar{V}$, respectively.

The ULV decomposition of the matrix A is denoted by

$$A = U \begin{pmatrix} L \\ 0 \end{pmatrix} V^T = U_1 L V^T \ , \ L = \begin{pmatrix} L_k & 0 \\ H & E \end{pmatrix} \ . \tag{2.2}$$

Here, $U^TU = I_m, V^TV = I_n$, and L has the properties: L_k is a $k \times k$ lower triangular

matrix whose singular values approximate $\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, E is an $(m-k) \times (n-k)$ rectangular matrix, and $\|(H\ E)\|_2 = \mathcal{O}(\sigma_{k+1})$. U_k and V_k denote submatrices consisting of the first k columns of U and V, respectively, and the accuracies of the subspaces associated with U_k and V_k certainly depend on $\|H\|_2$, cf. [12]. In LSI applications, k is much less than $\min(m, n)$ and represents the number of important factors needed for information retrieval (see Section 2.2.).

2.2. Latent Semantic Indexing

Latent Semantic Indexing [9,14] is applied to an initial term-by-document matrix. The elements of the term-by-document matrix are the weighted occurrences of each word in a particular document, i.e.,

$$A = [f_{ij} \times L_w(i,j) \times G_w(i)], \tag{2.3}$$

where f_{ij} denotes the frequency in which term i occurs in document j, $L_w(i,j)$ is the local weighting for term i in document j, and $G_w(i)$ is the global weighting for term i. The local and global weightings are applied [11] to increase/decrease the importance of terms within or among documents. Normally every word does not occur in each document so that the matrix A is sparse (i.e., relatively few nonzero elements).

The matrix A is then factored into the product of 3 matrices using the decomposition in Equation (2.1) or Equation (2.2). A model of latent semantic structure can be derived from the ULV decomposition defined by Equation (2.2). Specifically, the orthogonal matrices U and V containing approximate left and right singular vectors of A, respectively, and the triangular matrix, L. Such matrices are used to cast the original term-document relationships as linearly-independent vectors or *factor values*. The use of only k factors or the k-largest approximate singular triplets obtained by the ULV decomposition is achieved by approximating the original term-by-document matrix by

$$A_k = \sum_{i=1}^{k} u_i \cdot l_{ii} \cdot v_i^T \ , \tag{2.4}$$

where l_{ii} denotes the i-th diagonal entry of L from Equation (2.2). This rank-k matrix approximation of A closely resembles the more natural rank-k matrix approximations obtained by truncating the SVD or ULV, but is more efficient to derive when L_k is diagonally dominant and $\|H\|$ is sufficiently small.

In some sense, the ULV (or SVD from Equation (2.1)) is one way to derive a set of uncorrelated indexing variables or factors so that each term and document can be represented by a vector in k-space whose coordinates are defined by the elements of the left or right approximate singular vectors (see Table 1).

It is important for the LSI method that the derived A_k matrix does not reconstruct the original term-by-document matrix A exactly. The ULV decomposition, like the SVD, captures most of the important underlying structure in the association of terms and documents, yet at the same time removes the noise or variability in word usage that plagues word-based retrieval methods. Intuitively, since the number of dimensions, k, is much smaller than the number of unique terms, m,

Table 1. Interpretation of ULV components within LSI.

A_k	= Rank-k approx. to A	m	= Number of terms
U	= Term vectors	n	= Number of documents
diag(L_k)	= Diagonal elements of L_k	k	= Number of factors
V	= Document vectors	r	= Rank of A

minor differences in terminology will be ignored. Terms which occur in similar documents, for example, will be near each other in the k-dimensional factor space even if they never both occur in the same document. This means that some documents which do not share any words with a user's query may none the less be near it in k-space. This derived representation which captures term-term associations is used for retrieval.

Consider the words *doctor*, *physician*, *patient*, and *elephant*. The terms *doctor* and *physician* are synonyms, *patient* is a related concept and *elephant* is unrelated. In most retrieval systems, the query *physicians* is no more likely to retrieve documents about doctors than documents about elephants, if neither used precisely the term *physician* in the documents. It would be preferable if a query about *physicians* also retrieved articles about *doctors* and even articles about patients to a lesser extent. The derived k-dimensional feature space can represent these useful term interrelationships. Roughly speaking, the words *doctor* and *physician* will occur with many of the same words (e.g., *disease, health, hospital, illness, medicine, surgery,* etc.), and they will have similar representations in k-space. The contexts for *patient* will overlap to a lesser extent, and those for *elephant* will be quite dissimilar. The main idea in LSI is to explicitly model the interrelationships among terms (using a two-sided orthogonal decomposition) and to exploit this to improve retrieval.

2.3. Queries

A query, i.e., a set of words, can be considered as just another document which can be represented as a vector. Specifically the $m \times 1$ user query vector q is located at the weighted sum of its component term vectors in k-space. For example, q can be represented as a k-dimensional vector $\hat{q}$ via

$$\hat{q} = q^T U_k \left[\text{diag}(L_k)\right]^{-1}, \tag{2.5}$$

where diag(L_k) denotes the diagonal elements of L_k, the $k \times k$ principal submatrix of L. With this representation, the query vector can then be compared to all existing document vectors, and the documents ranked by their similarity (nearness) to the query. One common measure of similarity is the cosine between the query vector and document vector. Typically, the z closest documents or all documents exceeding some cosine threshold are returned to the user [9].

2.4. Updating

An LSI-generated database already requires that a collection of text objects be parsed, a term-by-document matrix be constructed, and the ULV decomposition

(or SVD) of the term-by-document matrix be determined. If additional terms and documents are to be included, a few alternatives for incorporating them are possible:

- *fold-in* the new terms and documents, or
- update an existing ULV decomposition (or SVD) of the original term-by-document matrix.

Four terms are defined below to avoid confusion when discussing updating. *Updating* is the general process of adding new terms and/or documents to a given LSI-generated database. Updating may refer to folding-in, ULV-updating, or SVD-updating. *SVD-updating* has been considered in [19], and *ULV-updating* is the focus of Section 5. *Folding-in* terms or documents is a much simpler alternative that uses an existing ULV decomposition (or SVD) to represent new information.

Recomputing the ULV decomposition is not a *true* updating method, but rather a way of reconstructing a new LSI-generated database with additional terms and/or documents. Recomputing the ULV decomposition of a larger term-by-document matrix incurs a higher computational cost for large problems, and may also impose excessive memory demands. This procedure allows the new p terms and q documents to directly affect the latent semantic structure by creating a new term-by-document matrix $A^{(m+p)\times(n+q)}$, computing the ULV decomposition of the new term-by-document matrix, and generating a different A_k matrix. Folding-in, on the other hand, is based on the existing latent semantic structure (the current A_k). New terms and documents will have no effect on the representations of any pre-existing terms and documents. Folding-in typically requires less computation time and memory but can produce inaccurate representations of new terms and documents. A potential loss of orthogonality in the columns of an updated U_k matrix for terms and V_k matrix for documents can distort the correct semantic structure (see [3]).

Folding-in documents is basically the process described in Section 2.3. for query representation. New documents are represented in k-space as weighted sums of their component term vectors. New document vectors are then appended to the set of existing document vectors or columns of V_k (see Figure 1). Similarly, new terms, which can be represented as a weighted sum of the vectors for documents they occur in, are appended to the set of existing term vectors or columns of U_k (see Figure 2).

To fold-in a new $m \times 1$ document vector, d, into an existing LSI model, a projection, $\hat{d}$, of d onto the span of the current term vectors (columns of U_k) is computed by

$$\hat{d} = d^T U_k \left[\mathrm{diag}(L_k)\right]^{-1}. \tag{2.6}$$

Similarly, to fold-in a new $1 \times n$ term vector, t, into an existing LSI model, a projection, $\hat{t}$, of t onto the span of the current document vectors (columns of V_k) is determined by

$$\hat{t} = t V_k \left[\mathrm{diag}(L_k)\right]^{-1}. \tag{2.7}$$

3. Obtaining the Semantic Model A_k

Rank-revealing algorithms are usually applied to ill-conditioned matrices to determine the number of *large* singular values or *small* singular values, and therefore the

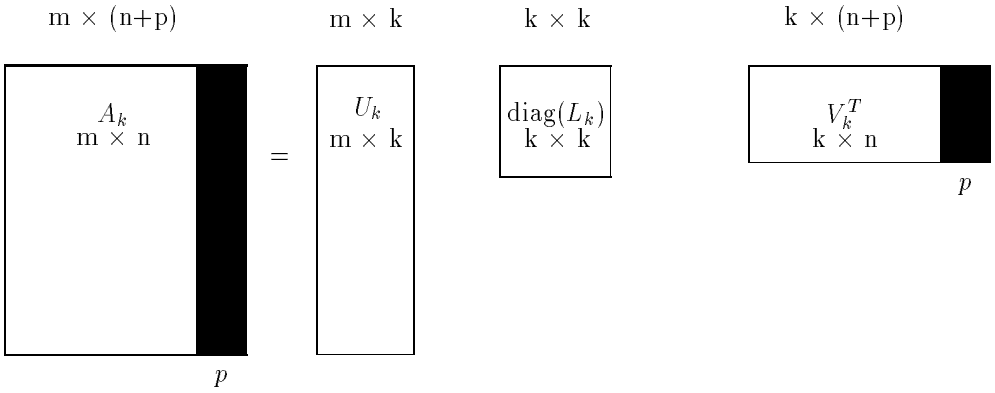


Figure 1. Block matrix representation of folding-in p documents.

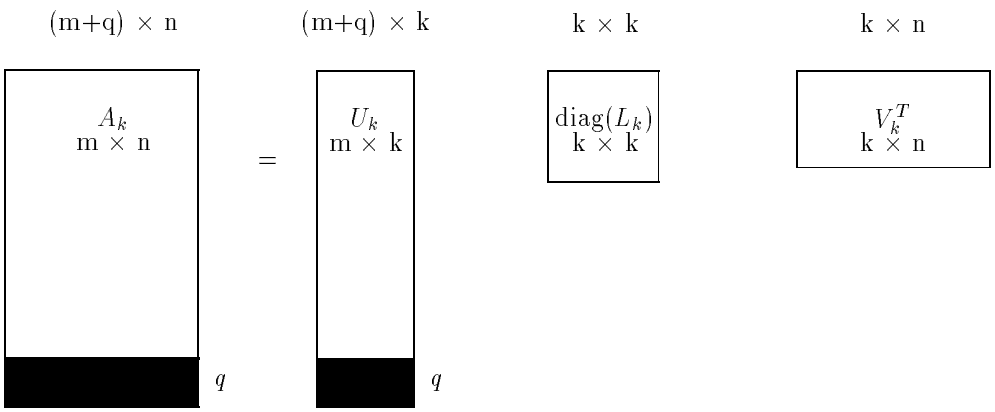


Figure 2. Block matrix representation of folding-in q terms.

numerical rank. This is important for many ill-conditioned least squares problems, for example, where one can then replace the problem with a nearby well-conditioned one to compute a less sensitive solution, cf. [13].

Rank-revealing algorithms applied to term-by-document matrices for LSI require special handling for the following reasons:

- The numerical rank of the matrix is close to $\min(m,n)$, but only approximations of the k-largest singular values and singular vectors are needed for LSI, where $k \ll \min(m,n)$. The matrix cannot be expected to have a large gap between σ_k and σ_{k+1}, thus a partition based on the numerical rank is generally too coarse to extract accurate approximations of the singular subspaces associated with the k-largest singular values.
- Term-by-document matrices are normally large and sparse. Rank-revealing algorithms typically preprocess such matrices using specialized orthogonal triangularization via Givens rotations [12] to obtain the triangular matrix and subsequent rank-revealing form. However, the rank-revealing steps can lead to massive fill-in for the triangular matrix.

Any approach customized for the term-by-document matrix would, at the least, need to compute good estimates of the k-largest singular values and singular vectors, and should preserve the sparsity of the original matrix A. In addition, the algorithm must be amenable to updating.

Algorithms for computing two-sided orthogonal decompositions were presented and analyzed in [12]. One algorithm, called ALGORITHM *L-ULV(A)P*, combines principal singular vector estimation, Householder transformations, and deflation procedures to estimate the column spaces of $\bar{U}_k$ and $\bar{V}_k$. This algorithm preserves the structure or sparsity of A. The main idea behind the algorithm is as follows.

Suppose we have a technique to compute

$$A = \tilde{U} \begin{pmatrix} \sigma_1 & 0 \\ 0 & A_2 \end{pmatrix} \tilde{V}^T,$$

where σ_1 is the largest (principal) singular value of A. Then the second largest singular value of A, i.e., σ_2, is given by $\sigma_2 = \|A_2\|_2$, and we can apply the same technique to A_2 (deflation) to obtain

$$A = \hat{U} \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & A_3 \end{pmatrix} \hat{V}^T.$$

On the other hand, suppose the technique computes

$$A = \tilde{U} \begin{pmatrix} l_{11} & 0 \\ h_1 & A_2 \end{pmatrix} \tilde{V}^T,$$

where $\|h_1\|_2$ is sufficiently small and l_{11} approximates σ_1. We can apply the same

technique to A_2 to obtain

$$A = \hat{U} \begin{pmatrix} l_{11} & 0 & 0 \\ \hat{h}_{21} & l_{22} & 0 \\ \hat{h}_1 & h_2 & A_3 \end{pmatrix} \hat{V}^T,$$

such that $\|h_2\|_2$ is sufficiently small and l_{22} approximates σ_2. Further, the first two columns of $\hat{U}$ and $\hat{V}$ approximate the left and right singular vectors corresponding to σ_1 and σ_2, respectively (cf. [12]).

Our modification of ALGORITHM *L-ULV(A)P* relies on a deflation parameter δ. We monitor $\|h_i\|_2$ and simply delay deflation until either $\|h_i\|_2$ is sufficiently small, i.e., $\|h_i\|_2 < \delta$, or the maximum number of attempts have been made to reduce $\|h_i\|_2$. With a sufficently small δ and $\|h_i\|_2 < \delta$ for $i = 1, \ldots, k$, then L_k is diagonally dominant and the norm of the resulting off-diagonal block satisfies $\|H\|_2 \leq \sqrt{k}\delta$. Thus, a sufficiently small deflation parameter δ guarantees accurate singular subspace approximations needed for LSI.

We define $\prod_{j=s}^{t} B^{(j)} = B^{(s)} B^{(s+1)} \cdots B^{(t)}$ for $s \leq t$ and $\prod_{j=s}^{t} B^{(j)} = I$ for $t < s$, where I is the identity matrix of appropriate dimensions.

ALGORITHM *L-ULV(A)P* for LSI:

Input:
A, an $m \times n$ data matrix
τ, a *rank* tolerance
δ, a *deflation* tolerance
N_δ, maximum number of refinements before deflation
$N_{\max}$, an upper bound for the number of LSI factors k

Output:
orthogonal matrices U and V stored in compact form
integer k such that $\sigma_k > \tau > \sigma_{k+1}$ or $k = N_{\max}$.

1. Initialize $i \leftarrow 1$, count $\leftarrow 0$, $U \in \Re^{m \times 0}$, and $V \in \Re^{n \times 0}$.
2. Compute the estimate $\left\{\sigma_{\text{est}}^{(1)}, u_{\text{est}}^{(1)}, v_{\text{est}}^{(1)}\right\}$ of the principal (largest) singular triplet of A.
3. While $\left(\sigma_{\text{est}}^{(i)} > \tau \text{ and } i \leq N_{\max}\right)$ do
4. Compute the Householder vectors $z_u^{(i)}$ and $z_v^{(i)}$ corresponding to $u_{\text{est}}^{(i)}$ and $v_{\text{est}}^{(i)}$, respectively, and store:
$U \leftarrow \left(U, \begin{pmatrix} 0 \\ z_u^{(i)} \end{pmatrix}\right)$ and $V \leftarrow \left(V, \begin{pmatrix} 0 \\ z_v^{(i)} \end{pmatrix}\right)$.
5. Define the Householder matrices
$P_{(i)} \equiv I_{m-i+1} - \beta_u^{(i)} z_u^{(i)} \left(z_u^{(i)}\right)^T$ and $Q_{(i)} \equiv I_{n-i+1} - \beta_v^{(i)} z_v^{(i)} \left(z_v^{(i)}\right)^T$,
where $\beta_u^{(i)} \equiv 2/\|z_u^{(i)}\|_2^2$ and $\beta_v^{(i)} \equiv 2/\|z_v^{(i)}\|_2^2$, and define the submatrices
$P_2^{(i)} \equiv P_{(i)}(1 : m-i+1, 2 : m-i+1)$ and

$Q_2^{(i)} \equiv Q_{(i)}(1:n-i+1, 2:n-i+1)$.

6. Set count $\leftarrow$ count $+ 1$. Define $h_i \equiv M(2:m-i+1, 1)$, where $M \equiv P_{(i)} \times \left(\prod_{j=1}^{i-1} P_2^{(j)}\right)^T \times A \times \prod_{j=1}^{i-1} Q_2^{(j)} \times Q_{(i)}$.

7. If $\|h_i\|_2 < \delta$ or count $= N_\delta$

   Deflate: set $i \leftarrow i+1$ and count $\leftarrow 0$.

   End If

8. Compute the estimate $\left\{\sigma_{\text{est}}^{(i)}, u_{\text{est}}^{(i)}, v_{\text{est}}^{(i)}\right\}$ of the principal singular triplet of the $(m-i+1) \times (n-i+1)$ matrix $\left(\prod_{j=1}^{i-1} P_2^{(j)}\right)^T \times A \times \prod_{j=1}^{i-1} Q_2^{(j)}$, (see [12]).

End of While loop

9. Set $k \leftarrow i-1$.

End of algorithm.

We note that if deflation does not occur then Step 8 is a *refinement* step in that the needed singular triplet estimate must be recomputed (in iterative algorithms it makes good sense to incorporate the previous estimate as the initial value). When the algorithm terminates, the orthonormal factors U_k and V_k needed in LSI can be recovered by:

$$U_k = \begin{pmatrix} I_k \\ 0_{m-k} \end{pmatrix}$$
for $t = k:-1:1$
$$U_k = \begin{pmatrix} I & 0 \\ 0 & P_{(t)} \end{pmatrix} U_k$$
end

$$V_k = \begin{pmatrix} I_k \\ 0_{n-k} \end{pmatrix}$$
for $t = k:-1:1$
$$V_k = V_k \begin{pmatrix} I & 0 \\ 0 & Q_{(t)} \end{pmatrix}$$
end.

The *backward* accumulation scheme is used, for example, in the skinny QR factorization (see p. 199 of [17]). It can be shown that the first k diagonal elements of L are given by $l_{ii} = \pm\sigma_{\text{est}}^{(i)}$. Here, the singular vector estimate $u_{\text{est}}^{(i)}$ is computed (in Step 8 above) by means of Lanczos (or Power method) iterations, cf. [12], from which $\sigma_{\text{est}}^{(i)}$ and $v_{\text{est}}^{(i)}$ can be obtained. Let s denote a fixed number of iterations the Lanczos (or Power) method used to compute the initial estimate $u_{\text{est}}^{(i)}$. If the resulting $\|h_i\|_2$ is sufficiently small, deflation occurs ($i \leftarrow i+1$). Otherwise, $u_{\text{est}}^{(i)}$ must be improved to reduce $\|h_i\|_2$. Suppose s Lanczos (or Power) iterations are also used for each refinement of the estimate $u_{\text{est}}^{(i)}$, and let I denote the average number of times this step is repeated (usually $0 \leq I < 1$). The flop (floating-point operation) count for computing the needed LSI factors using the modified form of ALGORITHM L-ULV(A)P (applied to a general $m \times n$ matrix A with $m \geq n$) is

given by

$$4(m+n)k^2\left[s+1+I\times s+\frac{1}{2}(I+1)\right]+M(A)\times \\ [2(k+2)(s+I\times s)+k(I+1)]\,, \tag{3.8}$$

where $M(A)$ is the number of flops required to compute a matrix-vector product Ax or y^TA. All lower-order terms have been neglected, including those coming from the fact that the dimensions of M in Step 6 are actually reduced in each step (these terms are negligible as long as $k \ll n$). The flop count in Equation (3.8) clearly illustrates how the overhead depends on both s and I.

For a 285×100 sparse term-by-document matrix (with terms required to globally occur at least 5 times in the first 100 abstracts parsed, see MED collection in [9]), Figure 3 illustrates the number of iterations of the Lanczos methods needed to produce

$$\left\{\sigma_{\text{est}}^{(i)},\, u_{\text{est}}^{(i)},\, v_{\text{est}}^{(i)}\right\},\ i=1,2,\ldots,25,$$

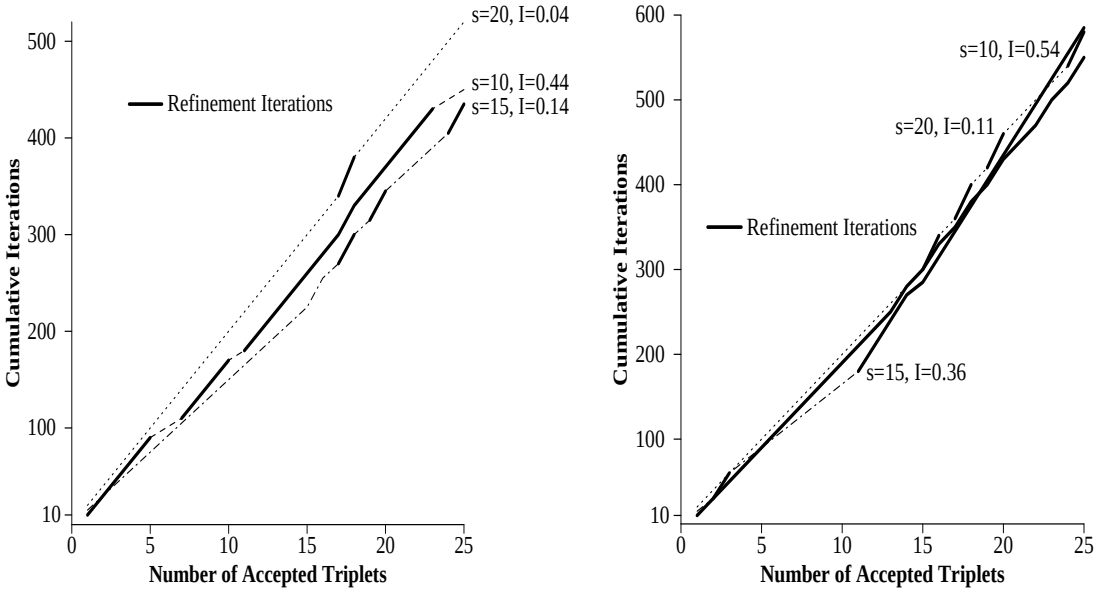
in Steps 2 and 8 of the algorithm. The cumulative number of iterations are shown by dashed or dotted lines and **bold** solid lines indicate when additional refinement iterations were required for $\delta = 10^{-2},\ 10^{-3}$. The maximum number of iterations allowed for the Lanczos method to produce any approximate singular triplet ranged from $s = 10$ to $s = 20$ for these experiments. Residual errors, $\|r_{est}^{(i)}\|_2$, of $\mathcal{O}(\delta)$ were obtained for each approximate singular triplet $\left\{\sigma_{est}^{(i)},\, u_{est}^{(i)},\, v_{est}^{(i)}\right\}$, where

$$\|r_{est}^{(i)}\|_2 \;=\; \left[(\|Av_{est}^{(i)}-\sigma_{est}^{(i)}u_{est}^{(i)}\|_2^2+\|A^Tu_{est}^{(i)}-\sigma_{est}^{(i)}v_{est}^{(i)}\|_2^2)^{\frac{1}{2}}\right] \\ /\left[\|u_{est}^{(i)}\|_2^2+\|v_{est}^{(i)}\|_2^2\right]^{\frac{1}{2}}\ .$$

Figure 3 clearly illustrates that the frequency of refinement (I) for $\left\{\sigma_{\text{est}}^{(i)},\, u_{\text{est}}^{(i)},\, v_{\text{est}}^{(i)}\right\}$ will increase for smaller values of s (the number of Lanczos iterations). However, to maintain a more moderate computation cost (flops) the number of Lanczos iterations (s) should not be too large (e.g., $s = 15$ curve versus $s = 20$ curve in Figure 3(a)). For smaller values of δ, a larger number of Lanczos iterations (s) will ensure the desired accuracy (δ) in the approximate singular triplets [12]. However, as illustrated in Figure 3(b), the cost (in iterations) per triplet will be higher regardless of the number of Lanczos iterations. Although a suitably large value of s will most likely yield a substantial reduction in refinement steps, the overall number of Lanczos iterations (or steps) may be excessive ($s = 20$ curve versus $s = 15$ curve).

4. A Demonstration of Latent Semantic Indexing

In this section, LSI and the folding-in process discussed in Section 2.4. are applied to a small database of medical topics. In Table 2, 18 topics are taken from the testbed of 1033 MEDLINE abstracts mentioned in Section 2.4.. All the underlined words



(a) $\delta = 10^{-2}$ (b) $\delta = 10^{-3}$

Figure 3. Cumulative iteration counts for the Lanczos method within ALGORITHM *L-ULV(A)* when used to approximate the 25-largest singular triplets of the 285×100 MEDLINE test matrix.

in Table 2 denote keywords which are used as referents to the medical topics. The parsing rule used for this sample database required that keywords appear in more than one topic. Of course, alternative parsing strategies can increase or decrease the number of indexing keywords (or terms).

Corresponding to the text in Table 2 is the 18×14 term-by-document matrix shown in Table 3. The elements of this matrix are the frequencies in which a term occurs in a document or medical topic (see Equation (2.3)). For example, in medical topic **M2**, the second column of the term-by-document matrix, *culture*, *discharge*, and *patients* all occur once. For simplicity, term weighting is not used in this example matrix. Now compute the ULV decomposition (with $k = 2$) of the 18×14 matrix in Table 2 to obtain the rank-2 approximation A_2 as defined in Equation (2.4).

Using the first column of U_2 multiplied by l_{11} for the x-coordinates and the second column of U_2 multiplied by l_{22} for the y-coordinates, the terms can be represented on the Cartesian plane. Similarly, the first column of V_2 scaled by l_{11} are the x-coordinates and the second column of V_2 scaled by l_{22} are the y-coordinates for the documents (medical topics). Figure 4 is a two-dimensional plot of the terms and documents for the 18×14 sample term-by-document matrix.

Notice the documents and terms pertaining to *patient behavior or hormone production* are clustered above the x-axis while terms and documents related to *blood disease or fasting* are clustered near the lower y-axis. Such groupings suggest that subsets of medical topics such as {**M2**, **M3**, **M4**} and {**M10**, **M11**, **M12**} each

Table 2. Database of medical topics from *MEDLINE*. Underlined keywords appear in more than one topic.

Label	Medical Topic
M1	study of depressed patients after discharge with regard to age of onset and culture
M2	culture of pleuropneumonia like organisms found in vaginal discharge of patients
M3	study showed oestrogen production is depressed by ovarian irradiation
M4	cortisone rapidly depressed the secondary rise in oestrogen output of patients
M5	boys tend to react to death anxiety by acting out behavior while girls tended to become depressed
M6	changes in children's behavior following hospitalization studied a week after discharge
M7	surgical technique to close ventricular septal defects
M8	chromosomal abnormalities in blood cultures and bone marrow from leukaemic patients
M9	study of christmas disease with respect to generation and culture
M10	insulin not responsible for metabolic abnormalities accompanying a prolonged fast
M11	close relationship between high blood pressure and vascular disease
M12	mouse kidneys show a decline with respect to age in the ability to concentrate the urine during a water fast
M13	fast cell generation in the eye lens epithelium of rats
M14	fast rise of cerebral oxygen pressure in rats

contain topics of *similar* meaning. Although topics **M1** and **M2** share the polysemous terms *culture* and *discharge* they are not represented by nearly identical vectors by LSI. The *meaning* of those terms in topics **M1** and **M2** are clearly different and literal-matching indexing schemes have difficulty resolving such context changes. Following the discussion of query representation in Section 2.3., the next section demonstrates how a particular query is processed for the small MEDLINE collection.

4.1. Queries

Suppose we are interested in the documents that contain information related to the *age of children with blood abnormalities*. Recall that a query vector (q) can be represented as ($\hat{q}$) via $\hat{q} = q^T U_k \, [\mathrm{diag}(L_k)]^{-1}$. Since the words *of*, *children*, and *with* are not indexed terms (i.e., stop words) in the database, they are omitted from the query leaving *age blood abnormalities*. Mathematically, the Cartesian coordinates of the query are determined by Equation (2.5) and the sample query *age blood abnormalities* is shown as the vector labeled QUERY in Figure 5. This query vector is then compared (in the Cartesian plane) to all the documents in the database. All documents whose cosine with the query vector is greater than 0.85 are indicated by the shaded region of Figure 5.

A different cosine threshold, of course, could have been used so that a larger or

Table 3. The 18×14 term-by-document matrix corresponding to the medical topics in Table 2.

Terms	Documents													
	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
abnormalities	0	0	0	0	0	0	0	1	0	1	0	0	0	0
age	1	0	0	0	0	0	0	0	0	0	0	1	0	0
behavior	0	0	0	0	1	1	0	0	0	0	0	0	0	0
blood	0	0	0	0	0	0	0	1	0	0	1	0	0	0
close	0	0	0	0	0	0	1	0	0	0	1	0	0	0
culture	1	1	0	0	0	0	0	1	1	0	0	0	0	0
depressed	1	0	1	1	1	0	0	0	0	0	0	0	0	0
discharge	1	1	0	0	0	1	0	0	0	0	0	0	0	0
disease	0	0	0	0	0	0	0	0	1	0	1	0	0	0
fast	0	0	0	0	0	0	0	0	0	1	0	1	1	1
generation	0	0	0	0	0	0	0	0	1	0	0	0	1	0
oestrogen	0	0	1	1	0	0	0	0	0	0	0	0	0	0
patients	1	1	0	1	0	0	0	1	0	0	0	0	0	0
pressure	0	0	0	0	0	0	0	0	0	0	1	0	0	1
rats	0	0	0	0	0	0	0	0	0	0	0	0	1	1
respect	0	0	0	0	0	0	0	1	0	0	0	1	0	0
rise	0	0	0	1	0	0	0	0	0	0	0	0	0	1
study	1	0	1	0	0	0	0	0	1	0	0	0	0	0

smaller set of documents would be returned. The cosine is merely used to rank-order documents and its numerical value is not always an adequate measure of relevance [19,22].

4.2. Comparison with Lexical Matching

In this example, LSI has been applied using two factors, i.e., A_2 is used to approximate the original 18×14 term-by-document matrix). Using a cosine threshold of .85, three medical topics related to *blood abnormalities and kidney failure* were returned: topics **M8**, **M9**, and **M12**. If the cosine threshold was reduced to just .75, then titles **M7** and **M11** (which are somewhat related) are also returned. With lexical-matching, five medical topics (**M1**, **M8**, **M10**, **M11**, **M12**) would be returned. Clearly, topics **M1** and **M10** are not relevant and topic **M9** would be missed. On the other hand, LSI is able to retrieve the most relevant topic from Table 2 (i.e., **M9**) to the original query *age of children with blood abnormalities* since *christmas disease* is the name associated hemophilia in young children. This ability to retrieve relevant information based on context or meaning rather than literal term usage is the main motivation for using LSI.

Table 4 lists the LSI-ranked documents (medical topics) with different numbers of factors (k). The documents returned in Table 4 satisfy a cosine threshold of .40, i.e., returned documents are within a cosine of .40 of the pseudo-document used to represent the query. As alluded to earlier, the cosine best serves as a measure for rank-ordering only as Table 4 clearly demonstrates that its value associated with returned documents can significantly vary with changes in the number of factors k.

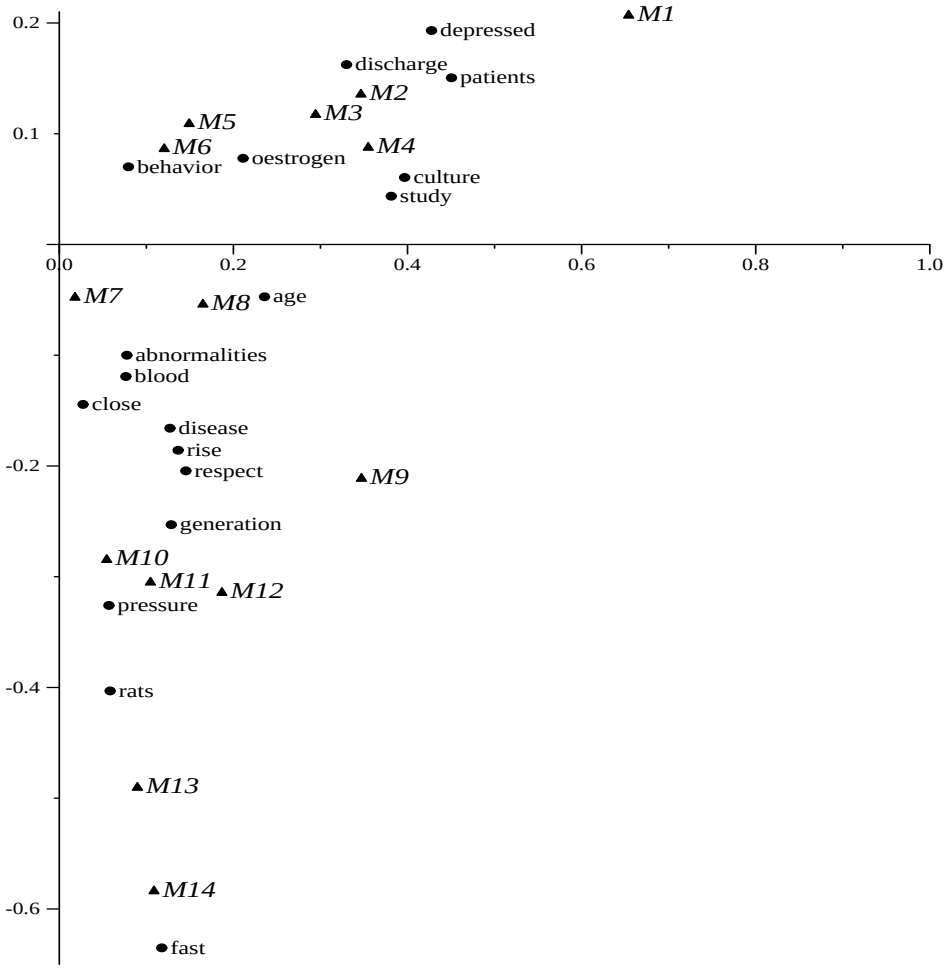


Figure 4. Two-dimensional plot of terms and documents for the 18×14 example.

4.3. Folding-In

Suppose the fictitious topics listed in Table 5 are to be added to the original set of medical topics in Table 2. These new topics (**M15** and **M16**) essentially use the terms as the original topics in Table 2 but in a somewhat different sense or context. Topic **M15** relates a *rise in oestrogen* with the behavior of *rats* rather than *patients*. Topic **M16** uses the term *pressure* in the context of *behavior* rather than *blood*. As with Table 2, all underlined words in Table 5 are considered significant since they appear in more than one title (across all 16 topics from Tables 2 and 5). Folding-in (see Section 2.4.) is one approach for updating the original LSI-generated database with the 2 new medical topics. Figure 6 demonstrates how these topics are folded-into the database based on $k = 2$ LSI factors via Equation (2.6). The new medical topics are denoted on the graph by their document labels (in a different boldface

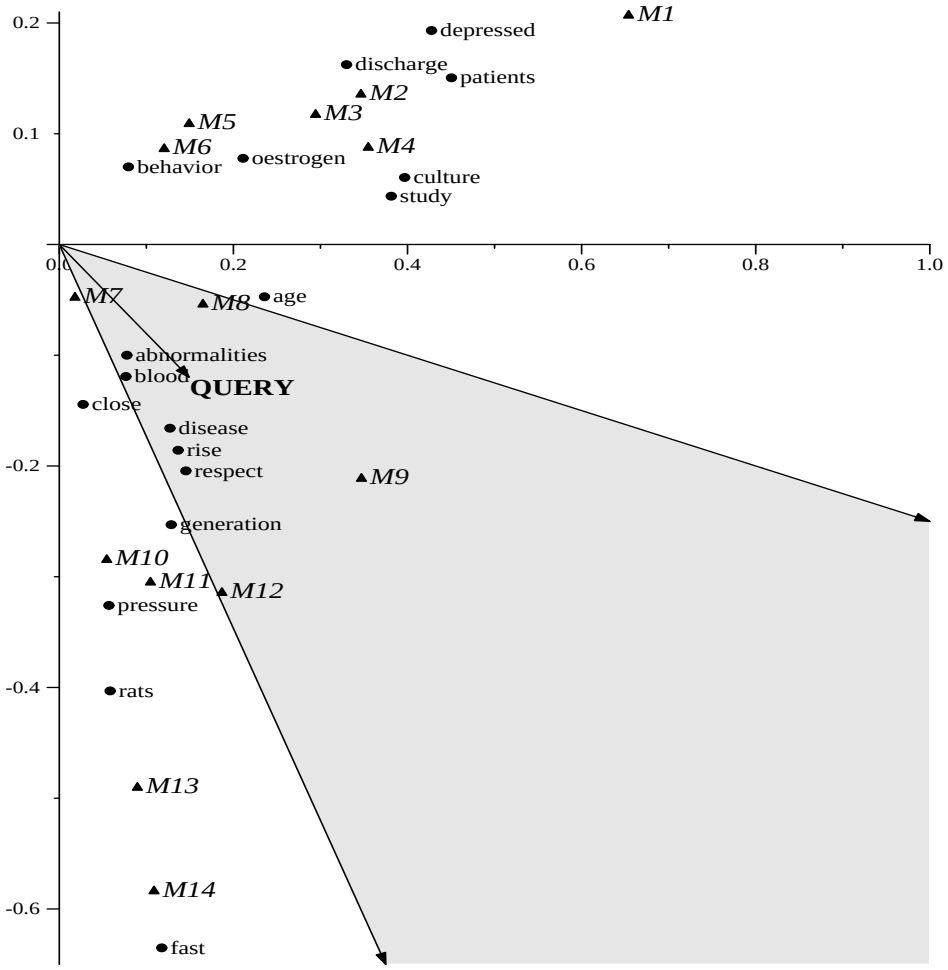


Figure 5. A Two-dimensional plot of terms and documents along with the query **age blood abnormalities**.

font). Notice that the coordinates of the original topics stay fixed, and hence the new data have no effect on the clustering of existing terms or documents.

4.4. Recomputing the ULV

Ideally, the most robust way to produce rank-k approximations (A_k) to a term-by-document matrix which has been updated with new terms and documents is to simply compute the ULV decomposition (see Equation (2.2)) of a reconstructed term-by-document matrix, say $\tilde{A}$.

Suppose the topics from Table 5 are combined with those of Table 2 in order to create a new 18×16 term-by-document matrix $\tilde{A}$. Following Equation (2.4), we

Table 4. Returned documents and corresponding cosine values based on different numbers k of LSI factors for a fixed cosine threshold of 0.40.

Number of Factors					
$k = 2$		$k = 4$		$k = 8$	
M 9	1.00	M 8	0.92	M 8	0.67
M12	0.88	M 9	0.89	M12	0.55
M 8	0.85	M 2	0.64	M10	0.54
M11	0.82	M10	0.48		
M10	0.79	M12	0.46		
M 7	0.74	M11	0.40		
M14	0.72				
M13	0.71				
M 4	0.67				
M 1	0.56				
M 2	0.42				

Table 5. Additional medical topics for updating.

Label	Medical Topic
M15	behavior of rats after detected rise in oestrogen
M16	depressed patients who feel the pressure to fast

then construct the rank-2 approximation to $\tilde{A}$ given by

$$\tilde{A}_2 = \tilde{U}_2 \, \mathrm{diag}(\tilde{L}_2) \, \tilde{V}_2^T. \tag{4.9}$$

Figure 7 is a two-dimensional plot of the 18 terms and 16 documents (medical topics) using the elements of $\tilde{U}_2$ and $\tilde{V}_2$ for term and document coordinates, respectively. Notice the difference in term and document positions between Figures 6 and 7. Clearly, the new medical topics from Table 5 have helped redefine the underlying latent structure when the ULV decomposition of $\tilde{A}$ is computed. That is, one can discuss blood *pressure* and behavioral *pressure* in different contexts. Note that in Figure 7 (unlike Figure 6) the topics (old and new) related to the use of *rats* form a well-defined *cluster* or subset of documents. Folding-in the 2 new medical topics based on the existing rank-2 approximation to A (defined by Table 3) may not accurately reproduce the true LSI representation of the new (or updated) database. In the case of topic **M15**, for example, the existing LSI model did not reflect the association of the term *behavior* with *rats*, and hence the folding-in procedure failed to form the cluster {**M13**, **M14**, **M15**} of related documents shown in Figure 7.

Updating methods which can approximate the ULV (or SVD) of the larger term-by-document matrix $\tilde{A}$ become attractive in the presence of memory or time constraints. In practice, the difference between folding-in and ULV-updating is likely to depend on the number of new documents and terms relative to the number in the original ULV decomposition of A. Thus, we expect ULV-updating to be especially valuable for rapidly changing databases. The accuracy of SVD-updating approaches

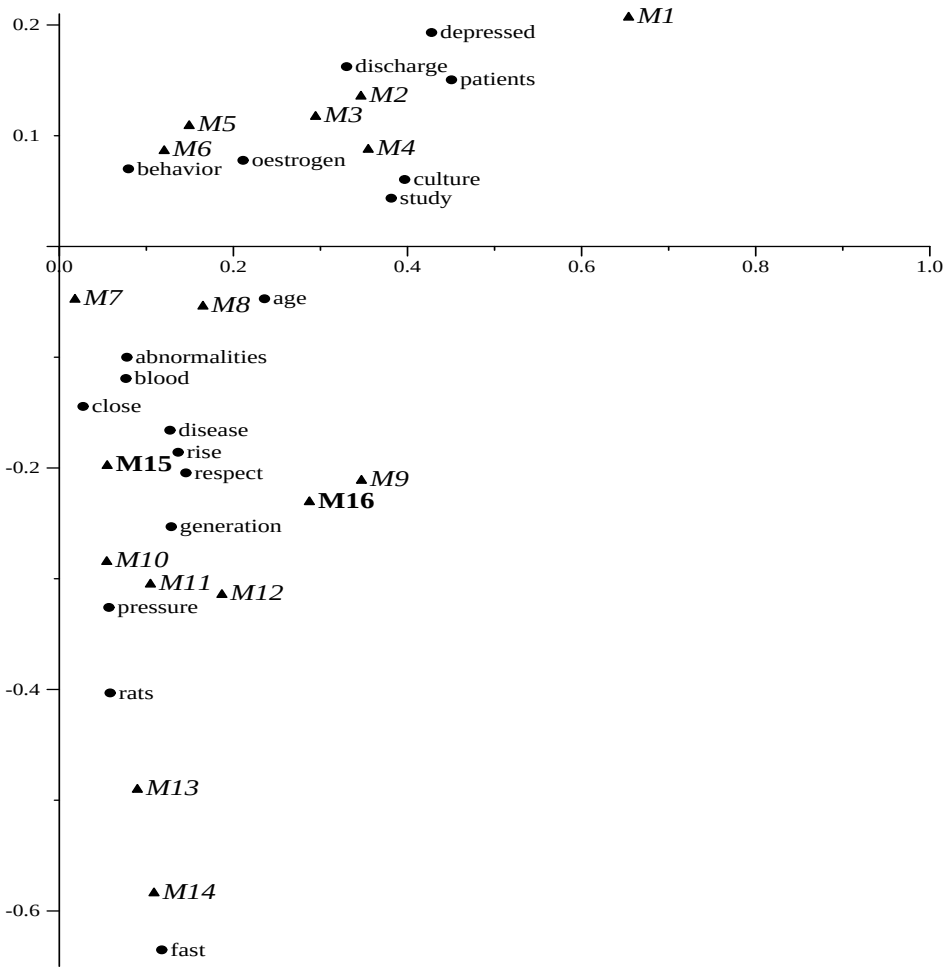


Figure 6. Two-dimensional plot of folded-in medical topics **M15** and **M16**.

can be easily compared to that obtained when the SVD of $\tilde{A}$ is explicitly computed, cf. [1] and [19].

5. ULV-Updating

The process of ULV-updating discussed in Section 2.4. can also be illustrated using titles from Tables 2 and 5. The three steps required to perform a complete ULV-update involve adding new documents, adding new terms, and correction for changes in term weightings. The order of these steps, however, need not follow the ordering presented in this section (see [3,19]).

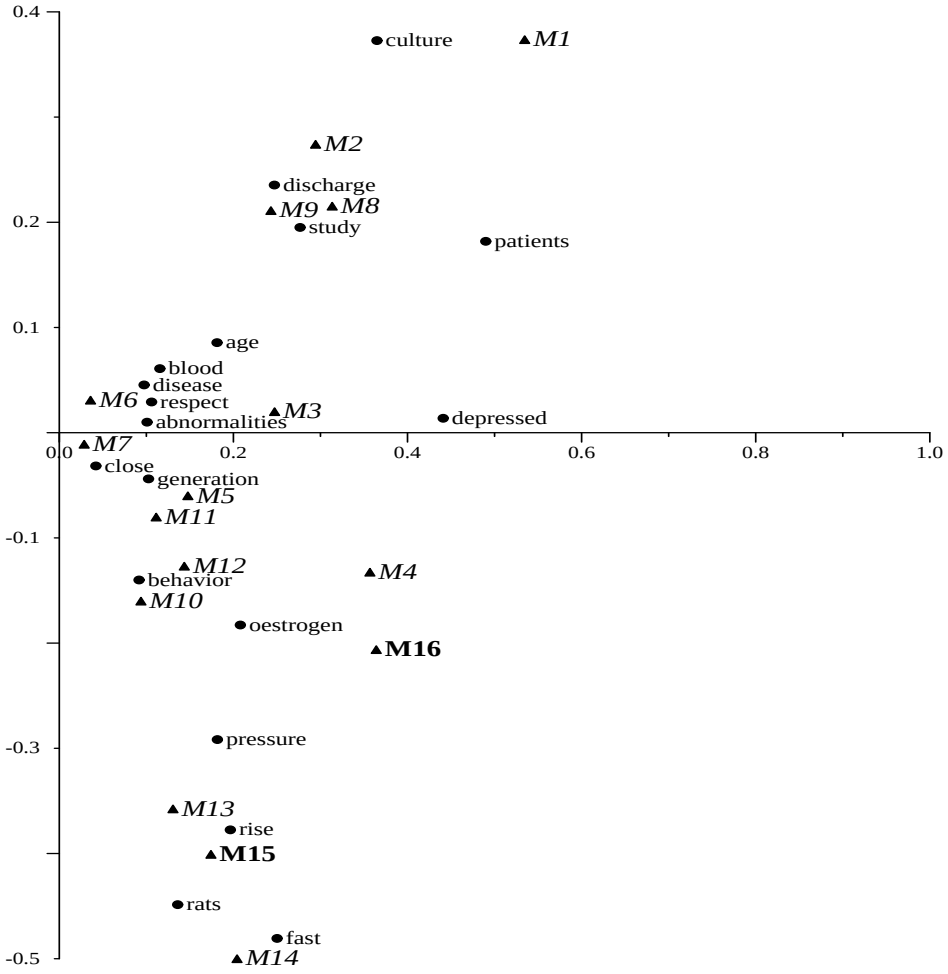


Figure 7. Two-dimensional plot of terms and documents using the ULV decomposition of a reconstructed term-by-document matrix.

5.1. Matrix Representations

Let D denote the p new document vectors to process, then D is an $m \times p$ sparse matrix since most terms (as was the case with the original term-by-document matrix A) do not occur in each document. D is appended to the columns of a rank-k approximation $\hat{A}_k$ of the $m \times n$ matrix A so that the k-largest approximate singular values and corresponding singular vectors of

$$B = (\hat{A}_k \mid D) \tag{5.10}$$

are computed. This is almost the same process as recomputing the ULV, only A is replaced by $\hat{A}_k$. A suitable choice for $\hat{A}_k$ is discussed in Section 5.2..

Let T denote a collection of $q \times 1$ term vectors for ULV-updating. Then T is a

$q \times n$ sparse matrix, since each term rarely occurs in every document. T is then appended to the rows of $\hat{A}_k$ so that the k-largest approximate singular values and corresponding singular vectors of

$$C = \begin{pmatrix} \hat{A}_k \\ \hline T \end{pmatrix} \tag{5.11}$$

are computed.

The correction step for incorporating changes in term weights (see Equation (2.3)) is performed after any terms or documents have been ULV-updated and the term weightings of the original matrix have changed. For a change of weightings in j terms, let Y_j be an $m \times j$ matrix comprised of rows of zeros or rows of the j-th order identity matrix, I_j, and let Z_j be an $n \times j$ matrix whose columns specify the actual differences between old and new weights for each of the j terms (see [19] for examples). Computing the ULV decomposition of the following rank-j update to $\hat{A}_k$ defines the correction step

$$\Phi = \hat{A}_k + Y_j Z_j^T . \tag{5.12}$$

5.2. ULV-Updating Procedures

The mathematical computations required in each phase of the ULV-updating process are detailed in this section. Table 6 contains a list of symbols, dimensions, and variables used to define the ULV-updating phases. ULV-updating incorporates new term or document information into the current semantic model (A_k from Equation (2.4)) using sparse term-by-document matrices (D, T, and $Y_j Z_j^T$) discussed in Section 5.1..

The ULV-updating procedures below exploit the natural rank-k matrix approximation of the original term-by-document matrix A, i.e.,

$$\hat{A}_k = U_k L_k V_k^T ,$$

from which the current semantic model $A_k = U_k \mathrm{diag}(L_k) V_k^T$ (see Equation (2.4)) is derived. Recall that the diagonal elements of the triangular matrix L_k and columns of the orthogonal matrices U_k, V_k serve as approximations to the singular values and singular vectors, respectively, of the original term-by-document matrix A. A brief summary of the required computations for updating the current rank-k semantic model, A_k, using standard linear algebra is given below.

5.2.1. Updating Documents Let $B = (\hat{A}_k \mid D)$ from Equation (5.10) have the ULV decomposition $B = U_B L_B V_B^T$. Then

$$U_k^T B \begin{pmatrix} V_k & O \\ O & I_p \end{pmatrix} = \left(L_k \mid U_k^T D \right) ,$$

where L_k is the lower triangular matrix defined by Equation (2.2), and $U_k^T D$ is a dense $k \times p$ matrix. Since U_k is typically stored as a dense matrix, the nonzero structure of D can be exploited in computing the matrix product $U_k^T D$. If P_k is the appropriate $(k+p) \times (k+p)$ orthogonal matrix (constructed from the accumulation

Table 6. Symbols used in ULV-updating phases.

Symbol	Dimensions	Definition
A	$m \times n$	Original term-by-document matrix
$\hat{A}_k$	$m \times n$	Rank-k ULV decomposition of A
U_k	$m \times k$	First k columns of U in ULV decomposition for A
L_k	$k \times k$	Lower triangular $k \times k$ principal submatrix of L in ULV decomposition for A
V_k	$n \times k$	First k columns of V in ULV decomposition for A
Z_j	$n \times j$	Adjusted term weights
Y_j	$m \times j$	Permutation matrix
D	$m \times p$	New document vectors
T	$q \times n$	New term vectors

of either Householder transformations or Givens rotations for just this step) such that

$$\left(L_k \mid U_k^T D\right) P_k = \left(\tilde{L}_k \mid 0\right),$$

then it follows that

$$U_B = U_k,\ V_B = \begin{pmatrix} V_k & O \\ O & I_p \end{pmatrix} P_k, \quad \text{and} \quad L_B = \tilde{L}_k. \tag{5.13}$$

Hence U_B and V_B are $m \times k$ and $(n+p) \times (k+p)$ dense matrices, respectively.

5.2.2. Updating Terms Let $C = \begin{pmatrix} \hat{A}_k \\ \hline T \end{pmatrix}$ from Equation (5.11) have the ULV decomposition $C = U_C L_C V_C^T$. Then

$$\begin{pmatrix} U_k^T & O \\ O & I_q \end{pmatrix} C V_k = \begin{pmatrix} L_k \\ \hline TV_k \end{pmatrix},$$

where TV_k is a dense $q \times k$ matrix. Here, T is a sparse matrix whose nonzero structure can be exploited for right multiplication by the dense matrix V_k. Let I_{k-1} be the $(k-1)$-th order identity matrix. If Q_k is the appropriate $(q+1) \times (q+1)$ orthogonal matrix (constructed from accumulated Householder transformations or Givens rotations for this step) such that

$$\begin{pmatrix} I_{k-1} & O \\ O & Q_k^T \end{pmatrix} \begin{pmatrix} L_k \\ TV_k \end{pmatrix} = \begin{pmatrix} \tilde{L}_k \\ O \end{pmatrix},$$

then it follows that

$$U_C = \begin{pmatrix} U_k & O \\ O & I_q \end{pmatrix} \begin{pmatrix} I_{k-1} & O \\ O & Q_k \end{pmatrix},\ V_C = V_k, \quad \text{and} \quad L_C = \tilde{L}_k.$$

Hence U_C and V_C are $(m+q) \times (k+q)$ and $n \times k$ dense matrices, respectively. Note that the first $k-1$ rows of the lower triangular matrices L_k and $\tilde{L}_k$ are identical.

5.2.3. Term Weight Corrections Let $\Phi = \hat{A}_k + Y_j Z_j^T$, where Y_j is $m \times j$ and Z_j is $n \times j$ from Equation (5.12) have the ULV decomposition $\Phi = U_\Phi L_\Phi V_\Phi^T$. Then

$$U_k^T \Phi V_k = L_k + X_j,$$

where $X_j = U_k^T Y_j Z_j^T V_k$ is a dense $k \times k$ matrix. The sparsity of Z_j can be exploited for the intermediate matrix product $Z_j^T V_k$ prior to left multiplication by the appropriate columns of U_k defined by $U_k^T Y_j$. If W_k is the appropriate $k \times k$ orthogonal matrix (constructed from either Householder transformations or Givens rotations for this triangularization step) such that

$$W_k^T \, (L_k + X_j) = \tilde{L}_k,$$

where $\tilde{L}_k$ is a $k \times k$ lower triangular matrix, then it follows that

$$U_\Phi = U_k W_k \ , \ V_\Phi = V_k \ \text{ and } \ L_\Phi = \tilde{L}_k.$$

Hence U_Φ and U_Φ are $m \times k$ and $n \times k$ dense matrices, respectively.

5.2.4. Accuracy and Costs With regard to the accuracy of the singular subspace approximations generated from the updated ULV decompositions discussed above, the magnitude of the diagonal elements of $\tilde{L}_k$, i.e., $|\tilde{l}_{ii}|$, in each case can be used as approximations for $\sigma_{\text{est}}^{(i)}$ of the respective matrices B, C, and Φ. The accuracy of the $\tilde{l}_{ii}$'s can be improved through systematic reduction in the norm of the off-diagonal elements of $\tilde{L}_k$, as provided by standard QR iterations (with shifts for optimal convergence). As with ALGORITHM *L-ULV(A)P* of Section 3., deflation occurs once an off-diagonal element of $\tilde{L}_k$ is less than δ in magnitude.

As noted in Section 3., LSI requires the k-largest singular values and corresponding singular vectors, where $k \ll \min(m, n)$ remains fixed (even after applying the updating procedures discussed above). Hence, rank-revealing techniques are not required for the ULV decomposition of matrices B, C, and Φ.

Table 7 contains the complexities (ignoring lower order terms) for folding-in terms and documents, and the three phases of ULV-updating. From these complexities the required number of floating-point operations (or flops) for each method can be compared for varying numbers of added documents or terms. Recall that U_k and V_k are already known (see Table 3.8) so that the only terms involving m and n are those associated with multiplication by the dense matrices U_k and V_k, and the sparse matrices D, T, and Z_j. As shown in [3] and [19] for SVD-updating, the computational complexity in each case of ULV-updating depends the values of the variables listed in Table 6. For example, if the sparsity of the D matrix from Equation (5.10) reflects that of the original $m \times n$ term-by-document matrix A with $m \gg n$, then folding-in will still require considerably fewer flops than ULV-updating when adding p new documents provided $p \ll n$.

Figure 8 illustrates the number of floating-point operations (log-scale) required (using the expressions from Table 7) to update the last $p = 25, 50, 75$ abstracts to the collection of MEDLINE abstracts represented by the $285 \times (100 - p)$ sparse term-by-document matrix previously discussed in Section 2.4.. For as few as $p = 25$ documents, folding-in requires (on average) as much as 1.6 times more operations than ULV-updating with up to $k = 25$ LSI factors with deflation tolerance $\delta = 10^{-2}$

Table 7. Computational complexity of updating methods (based on the use of Householder transformations) without recomputing the ULV decomposition of a new L matrix. The number $M(G)$ is the number of flops required to compute a matrix-vector product Gx or y^TG.

Method	Complexity
ULV-updating documents	$kM(D) + 2k^3/3 + 2k^2p + 2kp^2 + 2k^2n$
ULV-updating terms	$kM(T) + 2k^3/3 - 2k^2q + 2kq^2 + qn$
ULV-updating weight correction	$kM(Z_j) + 2k^3/3 + 2k^2j - k^2 + 2k^2m$
Folding-in documents	$2mkp$
Folding-in terms	$2nkq$

(see Section 2.4.). For $p = 75$ documents (or abstracts), folding-in required (on average) about twice as many operations needed by ULV-updating. Recomputing the ULV decomposition (see Equation (2.2)) for $k = 1$ to $k = 25$ LSI factors can require 75 and 40 times the number of floating-point operations of ULV-updating and folding-in, respectively, when adding $p = 75$ new documents to the collection. When compared to SVD-updating (see [1,3,19]), Figure 9 demonstrates the clear advantage of ULV-updating. For the same document updating depicted in Figure 8, SVD-updating requires about 6 and 14 times more floating-point operations than ULV-updating for adding $p = 75$ and $p = 25$ documents (abstracts), respectively, to the MEDLINE test collection.

5.3. ULV-Updating Example

To illustrate ULV-updating, suppose the two medical topics in Table 5 are to be added to the original set of medical topics in Table 2. In this example, only documents are added and weights are not adjusted, hence only the ULV decomposition of matrix B in Equation (5.10) is computed.

Initially, a 18×2 term-by-document matrix, D, corresponding to the new medical topics in Table 5 is generated and then appended to A_2 to form a 18×16 matrix B of the form given by Equation (5.10). Following Equation 2.4, the rank-2 approximation (B_2) to B is given by

$$B_2 = \hat{U}_2 \, \mathrm{diag}(\hat{L}_2) \, \hat{V}_2^T,$$

where the columns of $\hat{U}_2$ and $\hat{V}_2$ are the left and right approximate singular vectors, respectively, corresponding to the approximate singular values $\hat{l}_{11}$ and $\hat{l}_{22}$ of B.

Figure 10 is a two-dimensional plot of the 18 terms and 16 documents (medical topics) using the elements of $\hat{U}_2$ and $\hat{V}_2$ for term and document coordinates, respectively. Notice the similar clustering of terms and medical topics in Figures 7 (recomputing the ULV) and 10, and the difference in document and term clustering with Figure 6 (folding-in). The slight differences between the plots of Figures 7 and 10 can be attributed to the use of $\hat{A}_k$ rather than A (see Equation (5.10)) in

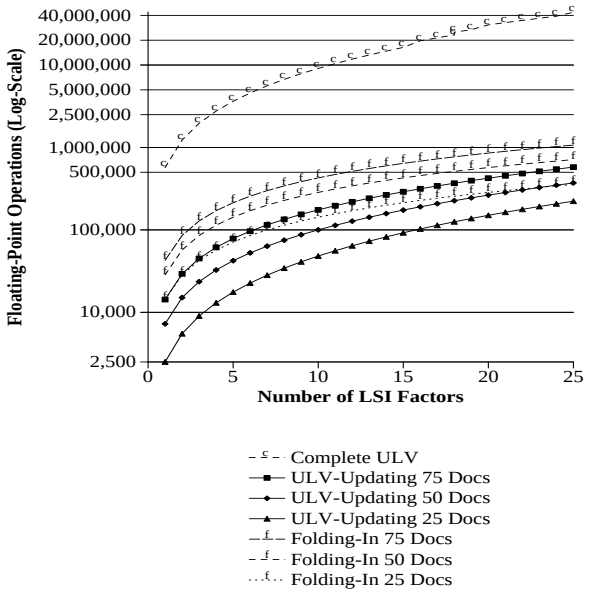


Figure 8. Computational complexity of ULV-updating and folding-in documents from the 285×100 MEDLINE matrix.

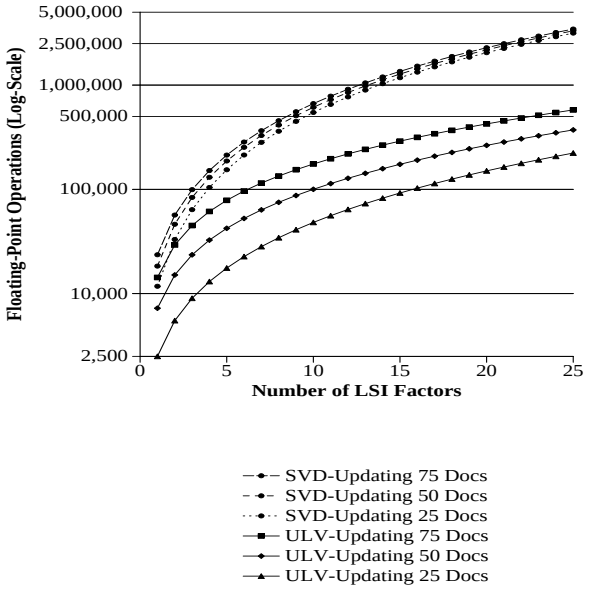


Figure 9. Computational complexity of ULV-updating and SVD-updating documents from the 285×100 MEDLINE matrix.

the ULV-updating process (Figure 10). As discussed in Section 2.2., the intent for using rank-k approximations ($\hat{A}_k$) to original term-by-document matrices (A) is to capture underlying semantic structure.

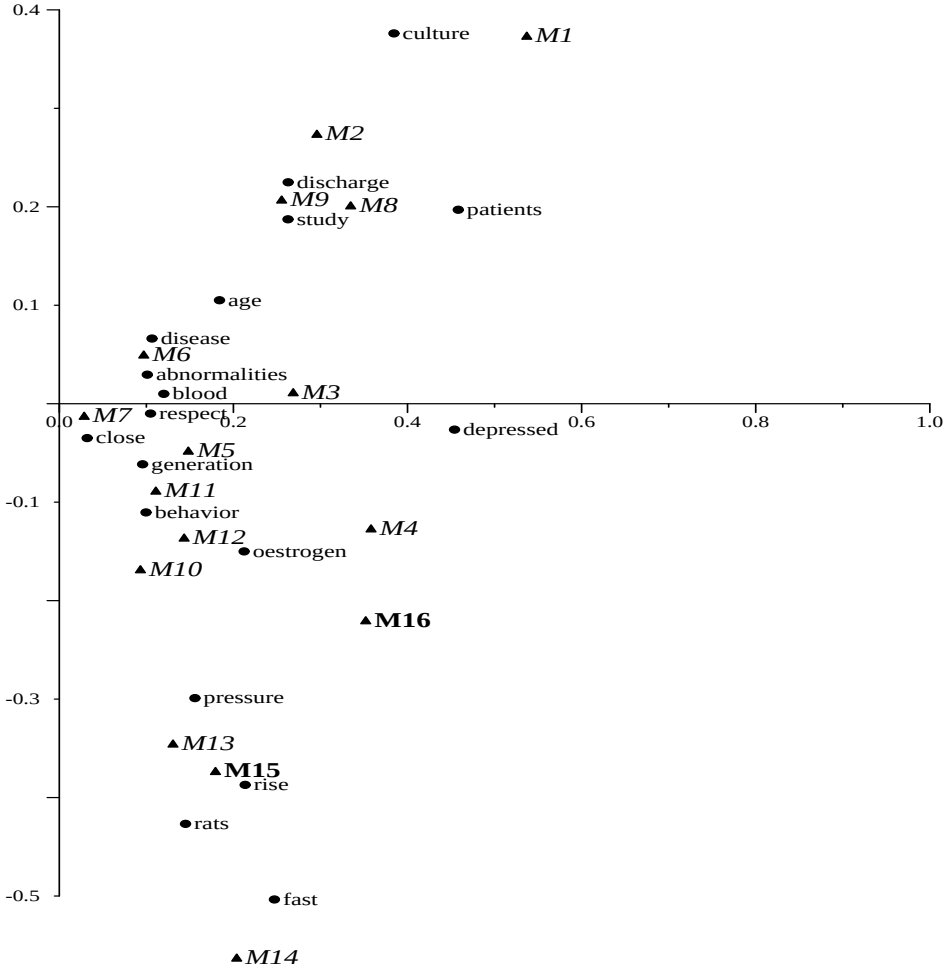


Figure 10. Two-dimensional plot of terms and documents using the ULV-updating process.

6. Summary and Future Work

A slight modification of ALGORITHM *L-ULV(A)P* has been demonstrated to be an efficient and suitably accurate scheme for computing the ULV decomposition of sparse term-by-document matrices arising from information retrieval applications. The algorithm was shown to be especially attractive for its reduced computational complexity in the context of updating documents, terms, or term corrections to

existing term-by-document matrices associated with applications such as Latent Semantic Indexing (LSI).

There are a number of computational improvements that would make LSI even more useful, especially for large collections:

- computing the ULV of extremely large sparse matrices, i.e., much larger than the usual 100,000 by 60,000 term-by-document matrix processed on RISC workstations with less than 500 megabytes of RAM,
- perform ULV-updating (see Section 5.) in real-time for databases that change frequently and/or are distributed across networks, and
- efficiently comparing queries to documents (i.e., finding near neighbors in high-dimension spaces).

Acknowledgements

This research was supported by the National Science Foundation under grant numbers NSF-ASC-92-03004 and NSF-CDA-91-15428. The authors would also like to thank the anonymous referees for their comments and suggestions which greatly improved the presentation of this work.

REFERENCES

1. M. Berry, S. Dumais, and G. O'Brien. The computational complexity of alternative updating approaches for an SVD-encoded indexing scheme. In *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, Philadelphia, 1995. SIAM.
2. M. W. Berry. Large scale singular value computations. *International Journal of Supercomputer Applications*, 6(1):13–49, 1992.
3. M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, December 1995. In press.
4. C. Bischof and P. Hansen. Structure-preserving and rank-revealing qr factorizations. *SIAM Journal on Scientific and Statistical Computing*, 12(1):1332–1350, 1991.
5. J. R. Bunch and N. P. Nielsen. Updating the Singular Value Decomposition. *Numerische Mathematik*, 31:111–129, 1978.
6. T. F. Chan. Rank Revealing QR Factorizations. *Linear Algebra and Its Applications*, 88:67–82, 1987.
7. T. F. Chan and P. C. Hansen. Computing Truncated Singular Value Decomposition Least Squares Solutions by Rank Revealing QR Factorizations. *SIAM Journal on Scientific and Statistical Computing*, 11:519–530, 1990.
8. T. F. Chan and P. C. Hansen. Some Applications of the Rank Revealing QR Factorization. *SIAM Journal on Scientific and Statistical Computing*, 13:727–741, 1992.
9. S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.
10. S. Dumais, G. Furnas, and T. Landauer. Using latent semantic analysis to improve access to textual information. In *Proceedings of Computer Human Interaction '88*, pages 281–285, 1988.
11. S. T. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, & Computers*, 23(2):229–236, 1991.
12. R. D. Fierro and P. C. Hansen. Low-Rank Revealing Two-Sided Orthogonal Decompositions. Technical Report 94-09, Department of Mathematics, California State University, San Marcos, CA, October 1994.

13. R. D. Fierro and P. C. Hansen. Accuracy of TSVD Solutions Computed From Rank Revealing Decompositions. *Numerische Mathematik*, 70:453–471, 1995.
14. P. W. Foltz and S. T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM*, 35(12):51–60, 1992.
15. L. Foster. Rank and Null Space Calculations Using Matrix Decomposition Without Column Interchanges. *Linear Algebra and Its Applications*, 74:47–71, 1986.
16. G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of SIGIR*, pages 465–480, 1988.
17. G. Golub and C. V. Loan. *Matrix Computations*. Johns-Hopkins, Baltimore, second edition, 1989.
18. C. Lawson and R. Hansen. *Solving Least Squares Problems*. Prentice Hall, N.J., first edition, 1974.
19. G. W. O'Brien. Information Management Tools for Updating an SVD-Encoded Indexing Scheme. Master's thesis, The University of Knoxville, Tennessee, Knoxville, TN, 1994.
20. G. W. Stewart. On an Algorithm for Refining a Rank-Revealing URV Decomposition and a Perturbation Theorem for Singular Values. Technical Report CS–TR 2626, University of Maryland, Department of Computer Science, College Park, MD, 1991.
21. G. W. Stewart. An Updating Algorithm for Subspace Tracking. *IEEE Transactions on Signal Processing*, 40:1535–1541, 1992.
22. P. G. Young. Cross-Language Information Retrieval Using Latent Semantic Indexing. Master's thesis, The University of Knoxville, Tennessee, Knoxville, TN, 1994.