Using Linear Algebra for Intelligent Information Retrieval

M.W. Berry, S.T. Dumais & G.W. O'Brien

Computer Science Department

CS-94-270 December 1994

USING LINEAR ALGEBRA FOR INTELLIGENT INFORMATION RETRIEVAL*

MICHAEL W. BERRY†, SUSAN T. DUMAIS‡ AND GAVIN W. O'BRIEN§

Abstract. Currently, most approaches to retrieving textual materials from scientific databases depend on a lexical match between words in users' requests and those in or assigned to documents in a database. Because of the tremendous diversity in the words people use to describe the same document, lexical methods are necessarily incomplete and imprecise. Using the singular value decomposition (SVD), one can take advantage of the implicit higher-order structure in the association of terms with documents by determining the SVD of large sparse term by document matrices. Terms and documents represented by 200-300 of the largest singular vectors are then matched against user queries. We call this retrieval method Latent Semantic Indexing (LSI) because the subspace represents important associative relationships between terms and documents that are not evident in individual documents. LSI is a completely automatic yet intelligent indexing method, widely applicable, and a promising way to improve users' access to many kinds of textual materials, or to documents and services for which textual descriptions are available. A survey of the computational requirements for managing LSI-encoded databases as well as current and future applications of LSI is presented.

Key words. indexing, information, latent, matrices, retrieval, semantic, singular value decomposition, sparse, updating

AMS(MOS) subject classifications. 15A18, 15A48, 65F15, 65F50, 68P20

1. Introduction. Typically, information is retrieved by literally matching terms in documents with those of a query. However, lexical matching methods can be inaccurate when they are used to match a user's query. Since there are usually many ways to express a given concept (synonymy), the literal terms in a user's query may not match those of a relevant document. In addition, most words have multiple meanings (polysemy), so terms in a user's query will literally match terms in irrelevant documents. A better approach would allow users to retrieve information on the basis of a conceptual topic or meaning of a document.

Latent Semantic Indexing (LSI) [4] tries to overcome the problems of lexical matching by using statistically derived conceptual indices instead of individual words for retrieval. LSI assumes that there is some underlying or latent structure in word usage that is partially obscured by variability in word choice. A truncated singular value decomposition (SVD) [14] is used to estimate the structure in word usage across documents. Retrieval is then performed using the database of singular values and vectors obtained from the truncated SVD. Performance data shows that these statistically derived vectors are 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 an existing LSI databases [4, 23]. The bulk of LSI processing time is spent in computing the truncated SVD of the large sparse term by document matrices.

Section 2 is a review of basic concepts needed to understand LSI. Section 3 uses a constructive example to illustrate how LSI represents terms and documents in the same semantic space, how a query is represented, how additional documents are added (or folded-in), and how SVD-updating represents additional documents. In Section 4, an algorithm for SVD-updating is discussed along with a comparison to the folding-in process with regard to robustness of query matching and computational complexity. Section 5 surveys promising applications of LSI along with parameter estimation problems that arise with its use.

* This research was supported by the National Science Foundation under grant Nos. NSF-CDA-9115428 and NSF-ASC-92-03004. Submitted to *SIAM Review*.

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

‡ Information Science Research Group, Bellcore, 445 South Street, Room 2L-371, Morristown, NJ 07962-1910, `std@bellcore.com`.

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

2. Background. The singular value decomposition is commonly used in the solution of unconstrained linear least squares problems, matrix rank estimation, and canonical correlation analysis [2]. 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, denoted by SVD(A), is defined as

$$A = U \Sigma V^T \tag{1}$$

where $U^T U = V^T V = I_n$ 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 U and V define the orthonormal eigenvectors associated with the r nonzero eigenvalues of AA^T and $A^T A$, respectively. The columns of U and V are referred to as the left and right singular vectors, respectively, and the singular values of A are defined as the diagonal elements of Σ which are the nonnegative square roots of the n eigenvalues of AA^T [14].

The following two theorems illustrate how the SVD can reveal important information about the structure of a matrix.

Theorem 2.1. *Let the SVD of A be given by Equation (1) and*

$$\sigma_1 \geq \sigma_2 \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0$$

and let $R(A)$ and $N(A)$ denote the range and null space of A, respectively.
Then,

1. *rank property: rank$(A) = r$, $N(A) \equiv$ span$\{v_{r+1}, \cdots, v_n\}$, and $R(A) \equiv$ span$\{u_1, \cdots, u_r\}$, where $U = [u_1 u_2 \cdots u_m]$ and $V = [v_1 v_2 \cdots v_n]$.*
2. *dyadic decomposition: $A = \sum_{i=1}^{r} u_i \cdot \sigma_i \cdot v_i^T$.*
3. *norms: $\|A\|_F^2 = \sigma_1^2 + \cdots + \sigma_r^2$, and $\|A\|_2^2 = \sigma_1$.*

Proof. See [14]. □

Theorem 2.2. [Eckart and Young] *Let the SVD of A be given by Equation (1) with r = rank(A) $\leq p = \min(m, n)$ and define*

$$A_k = \sum_{i=1}^{k} u_i \cdot \sigma_i \cdot v_i^T \ , \tag{2}$$

then

$$\min_{rank(B)=k} \|A - B\|_F^2 = \|A - A_k\|_F^2 = \sigma_{k+1}^2 + \cdots + \sigma_p^2.$$

Proof. See [15]. □

In other words, A_k, which is constructed from the k-largest singular triplets of A, is the closest rank-k matrix to A [14]. In fact, A_k is the best approximation to A for any unitarily invariant norm [21]. Hence,

$$\min_{rank(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}. \tag{3}$$

2.1. Latent Semantic Indexing. In order to implement Latent Semantic Indexing [4, 11] a matrix of terms by documents must be constructed. The elements of the term-document matrix are the occurrences of each word in a particular document, i.e.,

$$A = [a_{ij}], \tag{4}$$

where a_{ij} denotes the frequency in which term i occurs in document j. Since every word does not normally appear in each document, the matrix A is usually sparse. In practice, local and global weightings are applied [6] to increase/decrease the importance of terms within or among documents. Specifically, we can write

$$a_{ij} = L(i, j) \times G(i), \tag{5}$$

where $L(i,j)$ is the local weighting for term i in document j, and $G(i)$ is the global weighting for term i. The matrix A is factored into the product of 3 matrices (Equation (1)) using the singular value decomposition (SVD). The SVD derives the latent semantic structure model from the orthogonal matrices U and V containing left and right singular vectors of A, respectively, and the diagonal matrix, Σ, of singular values of A. These matrices reflect a breakdown of the original relationships into linearly-independent vectors or *factor values*. The use of k factors or k-largest singular triplets is equivalent to approximating the original (and somewhat unreliable) term-document matrix by A_k in Equation (2). In some sense, the SVD can be viewed as a technique for deriving a set of uncorrelated indexing variables or factors, whereby each term and document is represented by a vector in k-space using elements of the left or right singular vectors (see Table 1).

TABLE 1
Interpretation of SVD components within LSI.

A_k	= Best rank-k approximation to A		
		m	= Number of terms
U	= Term vectors	n	= Number of documents
Σ	= Singular values	k	= Number of factors
V	= Document vectors	r	= Rank of A

Figure 1 is a mathematical representation of the singular value decomposition. U and V are considered the term and document vectors respectively, and Σ represents the singular values. The shaded regions in U and V and the diagonal line in Σ represent A_k from Equation (2).

It is important for the LSI method that the derived A_k matrix not reconstruct the original term document matrix A exactly. The truncated SVD, in one sense, 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, 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 co-occur in the same document. This means that some documents which do not share any words with a users 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 *car*, *automobile*, *driver*, and *elephant*. The terms *car* and *automobile* are synonyms, *driver* is a related concept and *elephant* is unrelated. In most retrieval systems, the query *automobiles* is no more likely to retrieve documents about cars than documents about elephants, if neither used precisely the term *automobile* in the documents. It would be preferable if a query about *automobiles* also retrieved articles about *cars* and even articles about drivers to a lesser extent. The derived k-dimensional feature space can represent these useful term inter-relationships. Roughly speaking, the words *car* and *automobile* will occur with many of the same words (e.g. *motor, model, vehicle, chassis, carmakers, sedan, engine*, etc.), and they will have similar representations in k-space. The contexts for *driver* 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 the truncated SVD) and to exploit this to improve retrieval.

2.2. Queries. For purposes of information retrieval, a user's query must be represented as a vector in k-dimensional space and compared to documents. A query (like a document) is a set of words. For example, the user query can be represented by

$$\hat{q} = q^T U_k \Sigma_k^{-1}, \tag{6}$$

where q is simply the vector of words in the users query, multiplied by the appropriate term weights (see Equation (5)). The sum of these k-dimensional terms vectors is reflected by the $q^T U_k$ term in Equation (6), and the right multiplication by Σ_k^{-1} differentially weights the separate dimensions. Thus, the query vector is located at the weighted sum of its constituent term vectors. 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 [4].

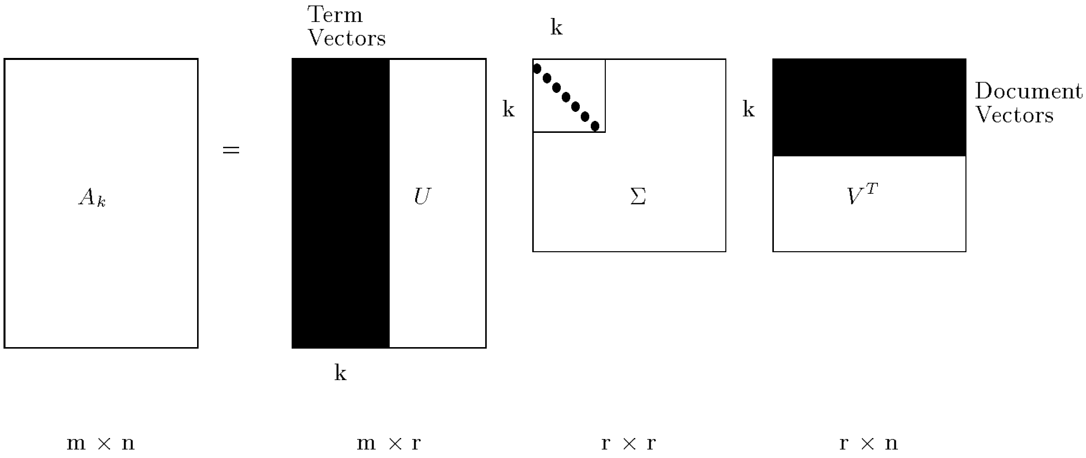

FIG. 1. *Mathematical representation of the matrix A_k.*

2.3. Updating. Suppose an LSI-generated database already exists. That is, a collection of text objects has been parsed, a term-document matrix has been generated, and the SVD of the term-document matrix has been computed. If more terms and documents must be added, two alternatives for incorporating them currently exist: recomputing the SVD of a new term-document matrix or *folding-in* the new terms and documents.

Four terms are defined below to avoid confusion when discussing updating. *Updating* refers to the general process of adding new terms and/or documents to an existing LSI-generated database. Updating can mean either folding-in or SVD-updating. *SVD-updating* is the new method of updating developed in [23]. *Folding-in* terms or documents is a much simpler alternative that uses an existing SVD to represent new information. *Recomputing the SVD* is not an updating method, but a way of creating an LSI-generated database with new terms and/or documents from scratch which can be compared to either updating method.

Recomputing the SVD of a larger term-document matrix requires more computation time and, for large problems, may be impossible due to memory constraints. Recomputing the SVD allows the new p terms and q documents to directly affect the latent semantic structure by creating a new term-document matrix $A^{(m+p)\times(n+q)}$, computing the SVD of the new term-document matrix, and generating a different A_k matrix. In contrast, folding-in is based on the existing latent semantic structure, the current A_k, and hence new terms and documents have no effect on the representation of the pre-existing terms and documents. Folding-in requires less time and memory but can have deteriorating effects on the representation of the new terms and documents.

Folding-in documents is essentially the process described in Section 2.2 for query representation. Each new document is represented as a weighted sum of its component term vectors. Once a new document vector has been computed it is appended to the set of existing document vectors or columns of V_k (see Figure 2). Similarly, new terms can be represented as a weighted sum of the vectors for documents in which they appear. Once the term vector has been computed it is appended to the set of existing term vectors or columns of U_k (see Figure 3).

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

$$\hat{d} = d^T U_k \Sigma_k^{-1}. \tag{7}$$

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

$$\hat{t} = t V_k \Sigma_k^{-1}. \tag{8}$$

3. A Demonstration of Latent Semantic Indexing. In this section, Latent Semantic Indexing (LSI) and the folding-in process discussed in Section 2.3 are applied to a small database of book titles. In Table 2, 17 book titles from book reviews published in the December 1993 issue (volume 54, number 4) of *SIAM Review* are listed. All the underlined words in Table 2 denote keywords which are used as referents to the book titles. The parsing rule used for this sample database required that

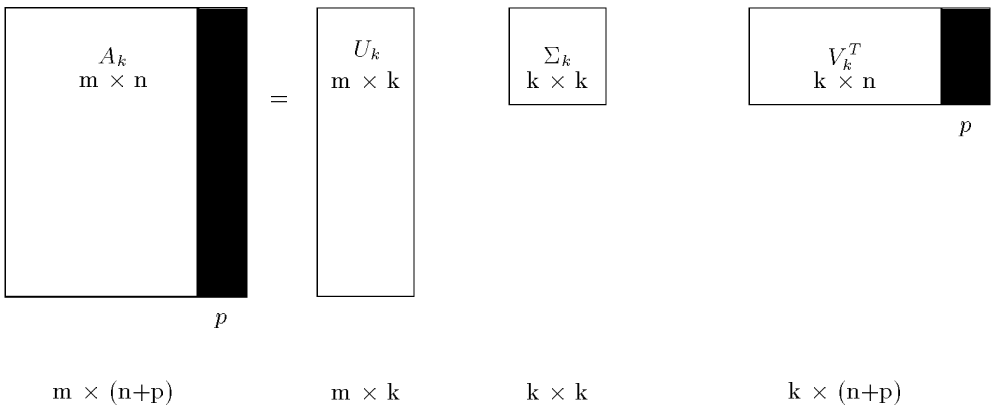

FIG. 2. *Mathematical representation of folding-in p documents.*

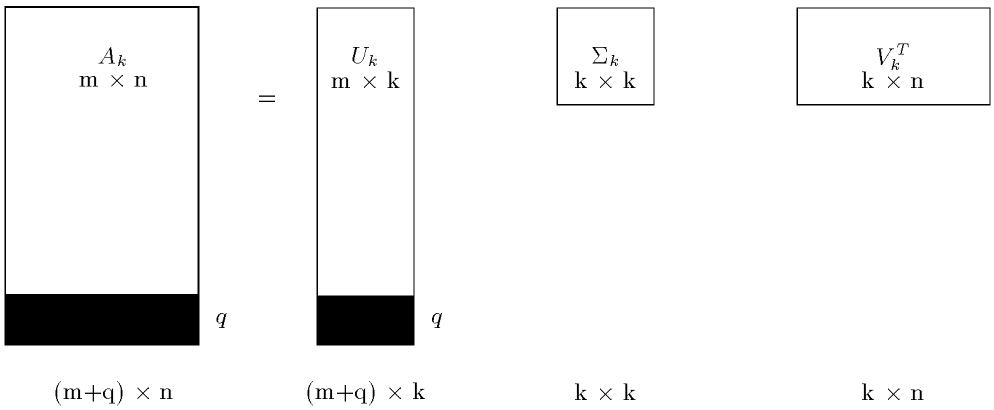

FIG. 3. *Mathematical representation of folding-in q terms.*

keywords appear in more than one book title. Of course, alternative parsing strategies can increase or decrease the number of indexing keywords (or terms).

TABLE 2
Database of titles from books reviewed in SIAM Review. *Underlined keywords appear in more than one book title.*

Label	Titles
B1	A Course on <u>Integral</u> <u>Equations</u>
B2	Attractors for Semigroups and Evolution <u>Equations</u>
B3	Automatic Differentiation of <u>Algorithms</u>: <u>Theory</u>, <u>Implementation</u>, and <u>Application</u>
B4	Geometrical Aspects of <u>Partial</u> <u>Differential</u> <u>Equations</u>
B5	Ideals, Varieties, and <u>Algorithms</u> – An <u>Introduction</u> to Computational Algebraic Geometry and Commutative Algebra
B6	<u>Introduction</u> to Hamiltonian Dynamical <u>Systems</u> and the N-Body <u>Problem</u>
B7	Knapsack <u>Problem</u>s: <u>Algorithms</u> and Computer <u>Implementation</u>s
B8	<u>Methods</u> of Solving Singular <u>Systems</u> of <u>Ordinary</u> <u>Differential</u> <u>Equations</u>
B9	<u>Nonlinear</u> <u>Systems</u>
B10	<u>Ordinary</u> <u>Differential</u> <u>Equations</u>
B11	<u>Oscillation</u> <u>Theory</u> for Neutral <u>Differential</u> <u>Equations</u> with <u>Delay</u>
B12	<u>Oscillation</u> <u>Theory</u> of <u>Delay</u> <u>Differential</u> <u>Equations</u>
B13	Pseudodifferential Operators and <u>Nonlinear</u> <u>Partial</u> <u>Differential</u> <u>Equations</u>
B14	Sinc <u>Methods</u> for Quadrature and <u>Differential</u> <u>Equations</u>
B15	Stability of Stochastic <u>Differential</u> <u>Equations</u> with Respect to Semi-Martingales
B16	The Boundary <u>Integral</u> Approach to Static and Dynamic Contact <u>Problem</u>s
B17	The Double Mellin-Barnes Type <u>Integral</u>s and Their <u>Application</u>s to Convolution <u>Theory</u>

Corresponding to the text in Table 2 is the 16×17 term-document matrix shown in Table 3. The elements of this matrix are the frequencies in which a term occurs in a document or book title (see Section 4). For example, in book title **B3**, the third column of the term-document matrix, *algorithms*, *theory*, *implementation*, and *application* all occur once. For simplicity, term weighting is not used in this example matrix. Now compute the truncated SVD (with $k = 2$) of the 16×17 matrix in Table 2 to obtain the rank-2 approximation A_2 as defined in Figure 1.

Using the first column of U_2 multiplied by the first singular value, σ_1, for the x-coordinates and the second column of U_2 multiplied by the second singular value, σ_2, for the y-coordinates, the terms can be represented on the Cartesian plane. Similarly, the first column of V_2 scaled by σ_1 are the x-coordinates and the second column of V_2 scaled by σ_2 are the y-coordinates for the documents (book titles). Figure 4 is a two-dimensional plot of the terms and documents for the 16×17 sample term-document matrix.

Notice the documents and terms pertaining to *differential equations* are clustered around the x-axis and the more general terms and documents related to *algorithms and applications* are clustered around the y-axis. Such groupings suggest that the subset of book titles {**B2, B4, B8, B9, B10 B13, B14, B15**} contains titles similar in meaning, for example.

TABLE 3
The 16×17 term-document matrix corresponding to the book titles in Table 2.

Terms	Documents																
	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16	B17
algorithms	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
application	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
delay	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
differential	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
equations	1	1	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0
implementation	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
integral	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
introduction	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
methods	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
nonlinear	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
ordinary	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
oscillation	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
partial	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
problem	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
systems	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
theory	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1

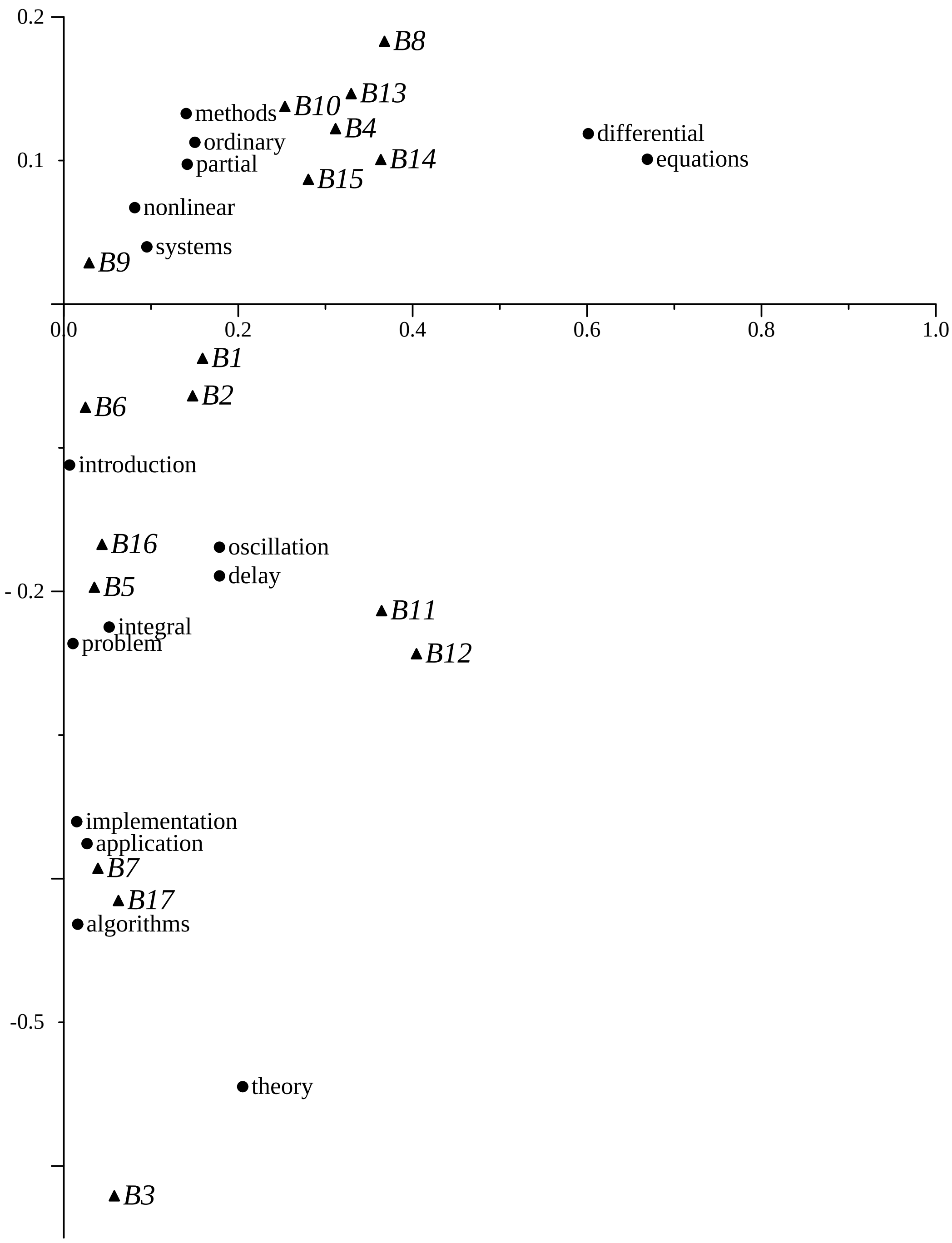

FIG. 4. *Two-dimensional plot of terms and documents for the* 16×17 *example.*

3.1. Queries. Suppose we are interested in the documents that pertain to *application and theory*. Recall that a query vector q is represented as $\hat{q}$ via $\hat{q} = q^T U_k \Sigma_k^{-1}$ (see Equation (6)). Since the word *and* is not an indexed term (i.e., a stop word) in the database, it is omitted from the query leaving *application theory*. Mathematically, the Cartesian coordinates of the query are determined by Equation (6). The coordinates for the sample query *application theory* are computed in Figure 5 and then represented by the point labeled QUERY in Figure 6. 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 0.90 is illustrated in the shaded region of Figure 6.

$$\begin{pmatrix} 0.0511 & -0.3337 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}^T \begin{pmatrix} 0.0159 & -0.4317 \\ 0.0266 & -0.3756 \\ 0.1785 & -0.1692 \\ 0.6014 & 0.1187 \\ 0.6691 & 0.1209 \\ 0.0148 & -0.3603 \\ 0.0520 & -0.2248 \\ 0.0066 & -0.1120 \\ 0.1503 & 0.1127 \\ 0.0813 & 0.0672 \\ 0.1503 & 0.1127 \\ 0.1785 & -0.1692 \\ 0.1415 & 0.0974 \\ 0.0105 & -0.2363 \\ 0.0952 & 0.0399 \\ 0.2051 & -0.5448 \end{pmatrix} \begin{pmatrix} 4.5314 & 0 \\ 0 & 2.7582 \end{pmatrix}^{-1}$$

FIG. 5. *Derived coordinates for the query of* **application theory**.

A different cosine threshold, of course, could have been used so that a larger or smaller set of documents would be returned. The cosine is merely used to rank-order documents and its explicit value is not always an adequate measure of relevance [23, 29].

3.2. Comparison with Lexical Matching. In this example, LSI has been applied using two factors, i.e. A_2 is used to approximate the original 16×17 term-document matrix. Using a cosine threshold of .90, six book titles related to *application and theory* were returned: titles **B3**, **B5**, **B6**, **B7**, **B16**, and **B17**. If the cosine threshold was reduced to .55, then titles **B11** and **B12** (which are somewhat related) are also returned. With lexical-matching, only four book titles (**B3**, **B11**, **B12**, **B17**) are returned. Hence, the LSI approach can extract four additional book titles (**B5**, **B6**, **B7**, **B16**) which are relevant to the query yet share no common terms. This ability to retrieve relevant information based on meaning rather than literal term usage is the main motivation for using LSI.

Table 4 lists the LSI-ranked documents (book titles) with different numbers of factors (k). The documents returned in Table 4 satisfy a cosine threshold of .20, i.e., returned documents are within a cosine of .20 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.

3.3. Folding-In. Suppose the fictitious titles listed in Table 5 are to be added to the original set of titles in Table 2. While some titles in Table 5 use terms related to nonlinear systems or differential equations, notice the different meaning of the specific term *ordinary* in book titles **B19** and **B20** as opposed to book titles **B8** and **B10**. As with Table 2, all underlined words in Table 5 are considered significant since they appear in more than one title (across all 20 titles from Tables 2 and 5). Folding-in (see Section 2.3) is one approach for updating the original LSI-generated database with the 3 new titles. Figure 7 demonstrates how these titles are folded-into the database based on $k = 2$ LSI factors via Equation (7). The new book titles are denoted on the graph by their document labels. Notice that the coordinates of the original titles stay fixed, and hence the new data has no effect on the clustering of existing terms or documents.

3.4. Recomputing the SVD. Ideally, the most robust way to produce the best rank-k approximation (A_k) to a term-document matrix which has been updated with new terms and documents is to simply compute the SVD of a reconstructed term-document matrix, say $\tilde{A}$. Updating methods which can approximate the SVD of the larger term-document matrix $\tilde{A}$ become attractive in the presence of memory or time constraints. As discussed in [23], the the accuracy of SVD-updating approaches can be easily compared to that obtained when the SVD of $\tilde{A}$ is explicitly computed.

Suppose the titles from Table 5 are combined with those of Table 2 in order to create a new 16×20 term-document matrix $\tilde{A}$. Following Figure 1, we then construct the best rank-2 approximation to $\tilde{A}$,

$$\tilde{A}_2 = \tilde{U}_2 \tilde{\Sigma}_2 \tilde{V}_2^T. \tag{9}$$

Figure 8 is a two-dimensional plot of the 16 terms and 20 documents (book titles) 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 7 and 8. Clearly, the the new book titles from Table 5 have helped redefine the underlying latent structure when the SVD of $\tilde{A}$ is computed. That is, one can discuss *ordinary algorithms* and *ordinary differential equations* in different contexts. Folding-in the 3 new book titles 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 practice, the difference between folding-in and SVD-updating is likely to depend on the number of new documents and terms relative to the number in the original SVD of A. Thus, we expect SVD-updating to be especially valuable for rapidly changing databases.

TABLE 4
Returned documents based on different numbers of LSI factors.

Number of Factors					
$k = 2$		$k = 4$		$k = 8$	
B17	.99	B17	.87	B17	.88
B 3	.99	B 3	.82	B 3	.78
B 6	.99	B12	.57	B12	.37
B16	.99	B11	.57	B11	.37
B 5	.98	B16	.38		
B 7	.98	B 7	.38		
B12	.55	B 1	.35		
B11	.55	B 5	.22		
B 1	.38				

TABLE 5
Additional titles for updating.

Label	Titles
B18	Systems of Nonlinear Equations
B19	Ordinary Algorithms for Integral and Differential Equations
B20	Ordinary Applications of Oscillation Theory

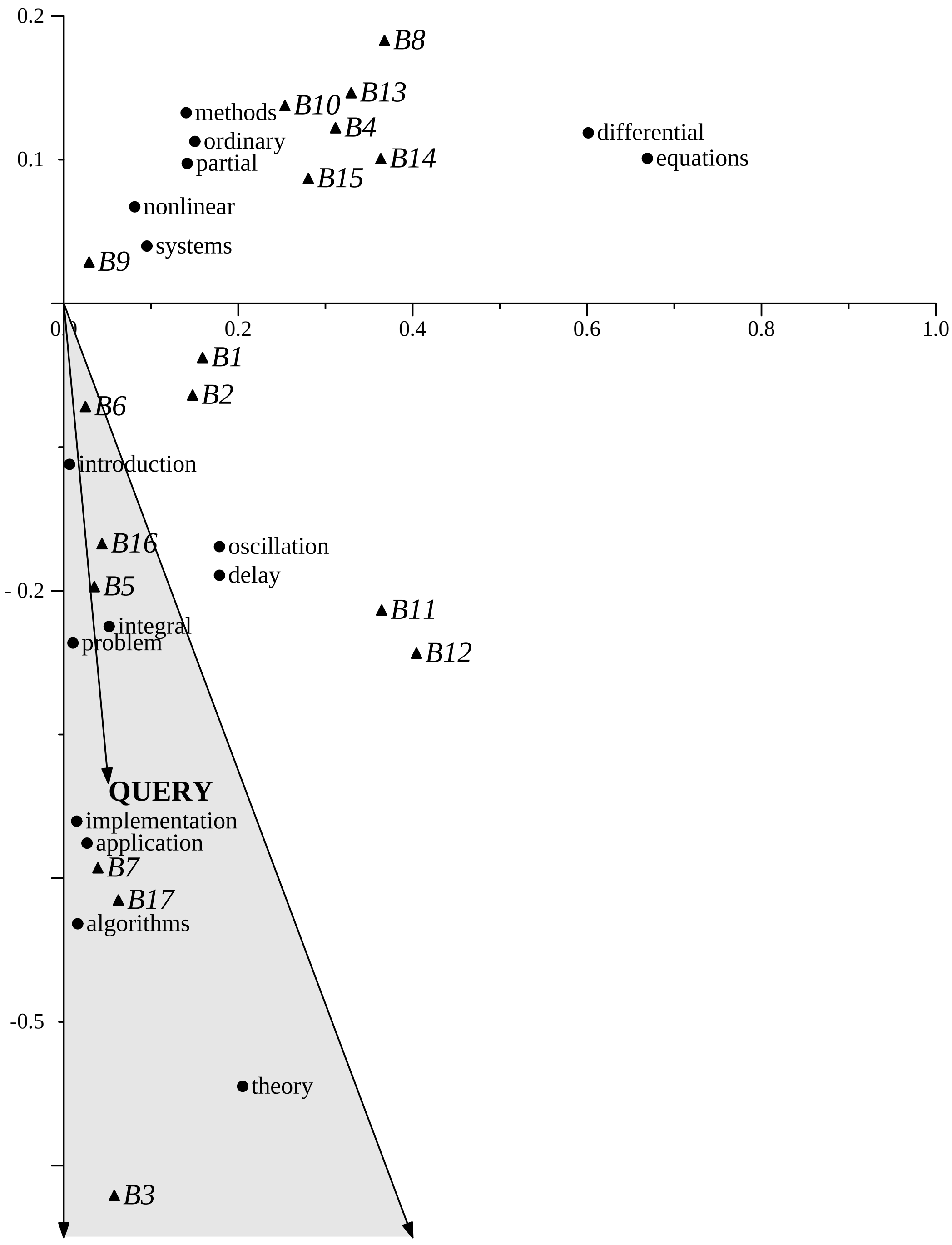

FIG. 6. *A Two-dimensional plot of terms and documents along with the query* **application theory**.

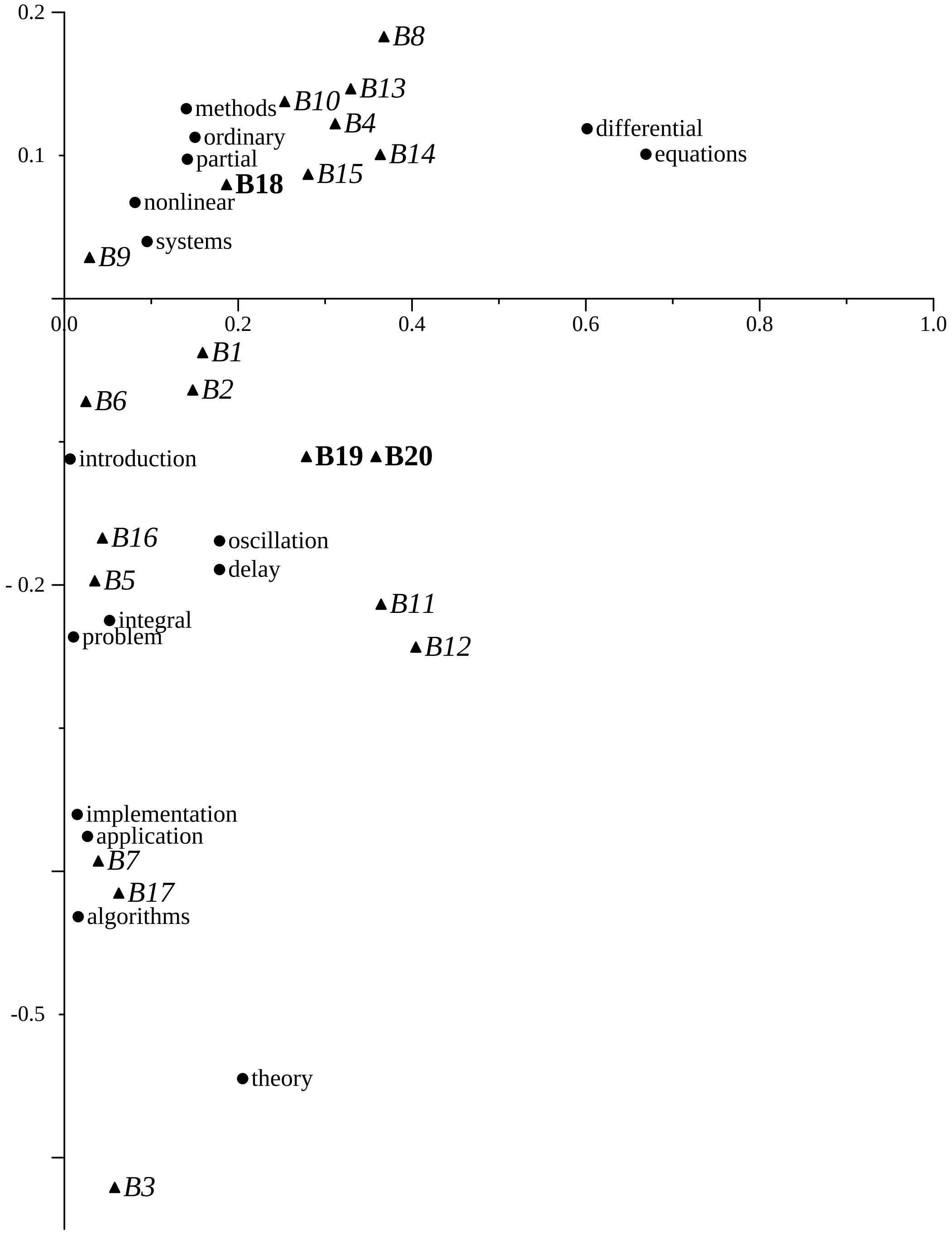

FIG. 7. *Two-dimensional plot of folded-in book titles.*

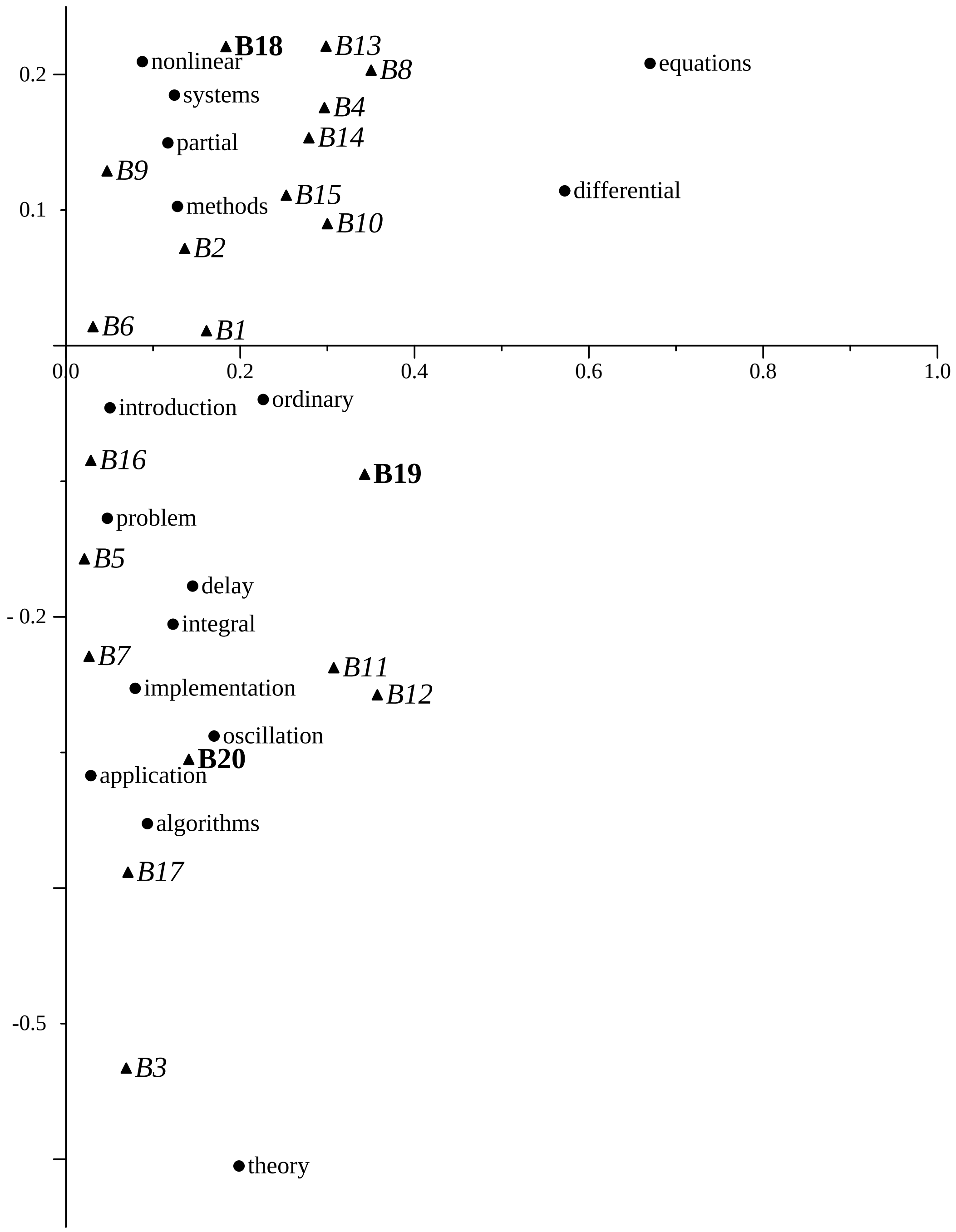

FIG. 8. *Two-dimensional plot of terms and documents using the SVD of a reconstructed term-document matrix.*

4. SVD-Updating. The process of SVD-updating discussed in Section 2.3 can also be illustrated using titles from Tables 2 and 5. The three steps required to perform a complete SVD-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 [23]).

4.1. Overview. 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-document matrix A) do not occur in each document. D is appended to the columns of the rank-k approximation of the $m \times n$ matrix A, i.e., from Equation (2), A_k so that the k-largest singular values and corresponding singular vectors of

$$B = (A_k \mid D) \tag{10}$$

are computed. This is almost the same process as recomputing the SVD, only A is replaced by A_k.

Let T denote a collection of q term vectors for SVD-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 A_k so that the k-largest singular values and corresponding singular vectors of

$$C = \left(\frac{A_k}{T} \right) \tag{11}$$

are computed.

The correction step for incorporating changes in term weights (see Equation (5)) is performed after any terms or documents have been SVD-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 [23] for examples). Computing the SVD of the following rank-j update to A_k defines the correction step.

$$W = A_k + Y_j Z_j^T. \tag{12}$$

4.2. SVD-Updating Procedures. The mathematical computations required in each phase of the SVD-updating process are detailed in this section. SVD-updating incorporates new term or document information into an existing semantic model (A_k from Equation (2)) using sparse term-document matrices (D, T, and $Y_j Z_j^T$) discussed in Section 4.1. SVD-updating exploits the previous singular values and singular vectors of the original term-documents matrix A as an alternative to recomputing the SVD of $\tilde{A}$ in Equation (9). In general, the cost of computing the SVD of a sparse matrix [3] can be generally expressed as

$$I \times \text{ cost } (G^T G x) + trp \times \text{ cost } (Gx),$$

where I is the number of iterations required by a Lanczos-type procedure [2] to approximate the eigensystem of $G^T G$ and trp is the number of accepted singular triplets (i.e., singular values and corresponding left and right singular vectors). The additional multiplication by G is required to extract the left singular vector given approximate singular values and their corresponding right singular vector approximations from a Lanczos procedure. A brief summary of the required computations for updating an existing rank-k approximation A_k using standard linear algebra is given below. Table 6 contains a list of symbols, dimensions, and variables used to define the SVD-updating phases.

TABLE 6
Symbols used in SVD-updating phases.

Symbol	Dimensions	Definition
A	$m \times n$	Original term-document matrix
U_k	$m \times k$	Left singular vectors of A_k
Σ_k	$k \times k$	Singular values of A_k
V_k	$n \times k$	Right singular vectors of A_k
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

Updating Documents. Let $B = (A_k \mid D)$ from Equation (10) and define SVD $(B) = U_B \Sigma_B V_B^T$. Then

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

since $A_k = U_k \Sigma_k V_k^T$. If $F = (\Sigma_k \mid U_k^T D)$ and SVD(F) $= U_F \Sigma_F V_F^T$, then it follows that

$$U_B = U_k U_F, \; V_B = \begin{pmatrix} V_k & O \\ O & I_p \end{pmatrix} V_F, \;\; \text{and} \;\; \Sigma_F = \Sigma_B. \tag{13}$$

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

Updating Terms. Let $C = \left(\dfrac{A_k}{T} \right)$ from Equation (11) and define SVD $(C) = U_C \Sigma_C V_C^T$. Then

$$\begin{pmatrix} U_k^T & O \\ O & I_q \end{pmatrix} C V_k = \left(\frac{\Sigma_k}{T V_k} \right).$$

If $H = \left(\dfrac{\Sigma_k}{T V_k} \right)$ and SVD$(H) = U_H \Sigma_H V_H^T$ then it follows that

$$U_C = \begin{pmatrix} U_k & O \\ O & I_q \end{pmatrix} U_H, \; V_C = V_k V_H, \;\; \text{and} \;\; \Sigma_H = \Sigma_C.$$

Hence U_C and V_C are $(m+q) \times (k+q)$ and $n \times k$ dense matrices, respectively.

Term Weight Corrections. Let $W = A_k + Y_j Z_j^T$, where Y_j is $m \times j$ and Z_j is $n \times j$ from Equation (12), and define SVD$(W) = U_W \Sigma_W V_W^T$. Then

$$U_k^T W V_k = (\Sigma_k + U_k^T Y_j Z_j^T V_k).$$

If $Q = (\Sigma_k + U_k^T Y_j Z_j^T V_k)$ and SVD$(Q) = U_Q \Sigma_Q V_Q^T$, then it follows that

$$U_W = U_k U_Q \;\; \text{and} \;\; V_W = V_k V_Q.$$

Since $(U_Q U_k)^T W V_k V_Q = \Sigma_Q = \Sigma_W$. Hence U_W and V_W are $m \times k$ and $n \times k$ dense matrices, respectively.

Table 7 contains the complexities for folding-in terms and documents, recomputing the SVD, and the three phases of SVD-updating. Using the complexities in Table 7 the required number of floating-point operations (or flops) for each method can be compared for varying numbers of added documents or terms. As shown in [23] for a condensed encyclopedia test case, the computational advantages of one scheme over another depends the values of the variables listed in Table 6. For example, if the sparsity of the D matrix from Equation (10) reflects that of the original $m \times n$ term-document matrix A with $m \gg n$, then folding-in will still require considerably fewer flops than SVD-updating when adding p new documents provided $p \ll n$. The expense in SVD-updating can be attributed to the $\mathcal{O}(2k^2m + 2k^2n)$ flops associated with the dense matrix multiplications involving U_k and V_k in Equation (13).

4.3. Orthogonality. One important distinction between the folding-in (see Section 2.3) and the SVD-updating processes lies in the guarantee of orthogonality in the vectors (or axes) used for term and document coordinates. Recall that an orthogonal matrix Q satisfies $Q^T Q = I_n$, where I_n is the n-th order identity matrix. Let D_p be the collection of all folded-in documents where each column of the $p \times k$ matrix is a document vector of the form $\hat{d}$ from Equation (7). Similarly, let T_q be the collection of all folded-in terms such that each column of the $q \times k$ matrix is a term vector of the form $\hat{t}$ from Equation (8). Then, all term vectors and document vectors associated with folding-in can be represented as $\hat{U}_k = \left(U_k^T \mid T_q^T\right)^T$ and $\hat{V}_k = \left(V_k^T \mid D_p^T\right)^T$, respectively. The folding-in process corrupts the orthogonality of $\hat{U}_k$ and $\hat{V}_k$ by appending non-orthogonal submatrices T_q and D_p to U_k and V_k, respectively. Computing $\hat{U}_k^T \hat{U}_k$ and $\hat{V}_k^T \hat{V}_k$, the loss of orthogonality in $\hat{U}_k$ and $\hat{V}_k$ can be measured by

$$\|\hat{U}_k^T \hat{U}_k - I_k\|_2 \text{ and } \|\hat{V}_k^T \hat{V}_k - I_k\|_2.$$

TABLE 7
Computational complexity of updating methods.

Method	Complexity
SVD-updating documents	$[I \times [4nnz(D) + 4mk + k - 2m - d] +$ $trp \times [2nnz(D) + 2mk - m]]$ $+ [(2k^2 - k)(m + n)]$
SVD-updating terms	$[I \times [4nnz(T) + 4kn + k - 2n - q] +$ $trp \times [2nnz(T) + 2kn + k - 2n - q]]$ $+ [(2k^2 - k)(m + n)].$
SVD-updating correction step	$[I \times [4nnz(Z_j) + 4km + 2mj + 2kn + 3k - 2n - 2j - m]$ $+ trp \times [2nnz(Z_j) + 2km + 2kn + k - j - n]]$ $+ [(2k^2 - k)(m + n)]$
Folding-in documents	$2mkp$
Folding-in terms	$2nkq$
Recomputing the SVD	$I \times [4nnz(A) - (m + q) - (n + p)] +$ $trp \times 2nnz(A) - (m + q)$

Folding-in does not maintain the orthogonality of $\hat{U}_k$ or $\hat{V}_k$ since arbitrary vectors of weighted terms or documents are appended to U_k or V_k, respectively. However, the amount by which the folding-in method perturbs the orthogonality of $\hat{U}_k$ or $\hat{V}_k$ does indicate how much distortion has occurred due to the addition of new terms or documents.

The trade-off in computational complexity and loss of orthogonality in the coordinate axes for updating databases using LSI poses interesting future research. Though the SVD-updating process is considerably more expensive [23] than folding-in, the true lower-rank approximation to the true term-document matrix A defined by Figure 1 is maintained. Significant insights in the future could be gained by monitoring the loss of orthogonality associated with folding-in and correlating it to the number of relevant documents returned within particular cosine thresholds (see Section 3.1).

4.4. SVD-Updating Example. To illustrate SVD-updating, suppose the fictitious titles in Table 5 are to be added to the original set of titles in Table 2. In this example, only documents are added and weights are not adjusted, hence only the SVD of the matrix B in Equation (10) is computed.

Initially, a 16×3 term-document matrix, D, corresponding to the fictitious titles in Table 5 is generated and then appended to A_2 to form a 16×20 matrix B of the form given by Equation (10). Following Figure 1, the best rank-2 approximation (B_2) to B is given by

$$B_2 = \hat{U}_2 \hat{\Sigma}_2 \hat{V}_2^T,$$

where the columns of $\hat{U}_2$ and $\hat{V}_2$ are the left and right singular vectors, respectively, corresponding to the two largest singular values of B.

Figure 9 is a two-dimensional plot of the 12 terms and 16 documents (book titles) using the elements of $\hat{U}_2$ and $\hat{V}_2$ for term and document coordinates, respectively. Notice the similar clustering of terms and book titles in Figures 9 and 8 (recomputing the SVD) and the difference in document and term clustering with Figure 7 (folding-in).

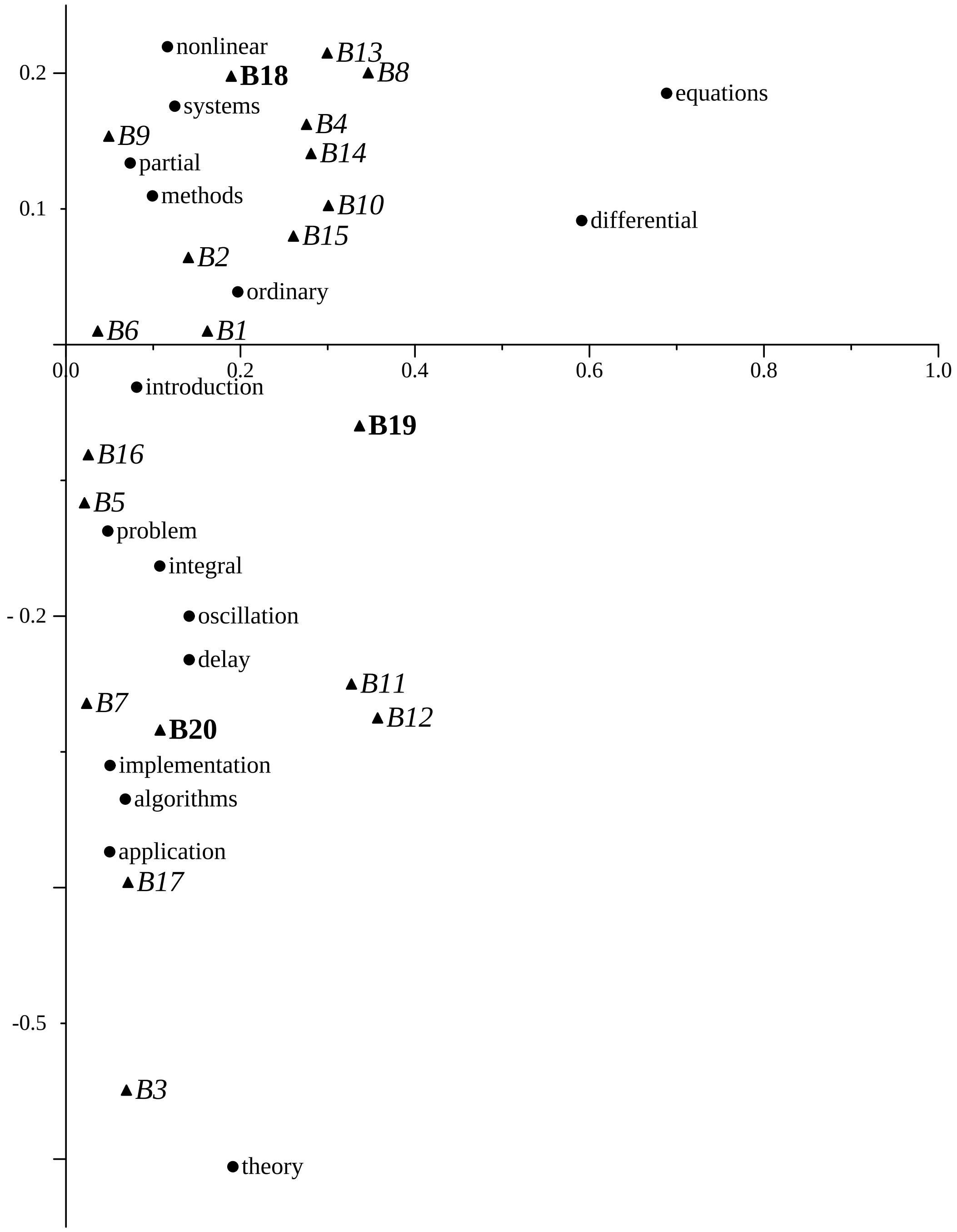

FIG. 9. *Two-dimensional plot of terms and documents using the SVD-updating process.*

5. Applications of Latent Semantic Indexing. In this section, several applications of LSI are discussed ranging from information retrieval and filtering to models of human memory. Some open computational and statistical-based issues related to the practical use of LSI for such applications are also mentioned.

5.1. Information Retrieval. Latent Semantic Indexing was initially developed for information retrieval applications. In these application, a fixed database is indexed and users pose a series of retrieval queries. The effectiveness of retrieval systems is often evaluated using *test collections* developed by the information retrieval community. These collections consist of a set of documents, a set of user queries, and relevance judgements (i.e., for each query every document in the collection has been judged as relevant or not to the query)[1]. This allows one to evaluate the effectiveness of different systems in retrieving relevant documents and at the same time not returning irrelevant documents. Two measures, *precision* and *recall*, are used to summarize retrieval performance. *Recall* is the proportion of all relevant documents in the collection that are retrieved by the system; and *precision* is the proportion of relevant documents in the set returned to the user. Average precision across several levels of recall can then be used as a summary measure of performance.

Results were obtained for LSI and compared against published or computed results for other retrieval techniques, notably the standard keyword vector method in SMART [24]. For several information science test collections, the average precision using LSI ranged from comparable to to 30% better than that obtained using standard keyword vector methods. See [4, 6, 12] for details of these evaluations. The LSI method performs best relative to standard vector methods when the queries and relevant documents do not share many words, and at high levels of recall.

Term Weighting. One of the common and usually effective methods for improving retrieval performance in vector methods is to transform the raw frequency of occurrence of a term in a document (i.e., the value of a cell in the term by document matrix) by some function (see Equation 5). Such transformations normally have two components. Each term is assigned a *global weight*, indicating its overall importance in the collection as an indexing term. The same global weighting is applied to an entire row (term) of the term-document matrix. It is also possible to transform the term's frequency in the document; such a transformation is called a *local weighting*, and is applied to each cell in the matrix.

The performance for several weighting schemes have been compared in [6]. A transformed matrix is automatically computed, the truncated SVD shown in Figure 1 is computed, and performance is evaluated. A *log* transformation of the local cell entries combined with a global *entropy* weight for terms is the most effective term-weighting scheme. Averaged over five test collections, $log \times entropy$ weighting was 40% more effective than raw term weighting.

Relevance Feedback. The idea behind relevance feedback is quite simple. Users are very unlikely to be able to specify their information needs adequately, especially on the first try. In interactive retrieval situations, it is possible to take advantage of user feedback about relevant and non-relevant documents [25]. Systems can use information about which documents are relevant in many ways. Typically the weight given to terms occurring in relevant documents is increased and the weight of terms occurring in non-relevant documents is decreased. Most of the tests using LSI have involved a method in which the initial query is *replaced* with the vector sum of the documents the users has selected as relevant. The use of negative information has not yet been exploited in LSI; for example, by moving the query away from documents which the user has indicated are irrelevant. Replacing the users' query with the first relevant document improves performance by an average of 33% and replacing it with the average of the first three relevant documents improves performance by an average of 67% (see [6] for details). Relevance feedback provides sizable and consistent retrieval advantages. One way of thinking about the success of these methods is that many words (those from relevant documents) augment the initial query which is usually quite impoverished. LSI does some of this kind of query expansion or enhancement even without relevance information, but can be augmented with relevance information.

---

[1] Exhaustive relevance judgements (when all documents are judged for every query) are ideal for system evaluation. In large document collections, however, exhaustive judgements become prohibitively costly. For large collections a *pooling method* is used. Relevance judgements are made on the pooled set of the top-ranked documents returned by several different retrieval systems for the same set of queries. Most of the top-ranked documents for new systems will hopefully be contained in the pool set and thus have relevance judgements associated with them.

5.2. Choosing the Number of Factors. Choosing the number of dimensions (k) for A_k shown in Figure 1 is an interesting problem. While a reduction in k can remove much of the noise, keeping too few dimensions or factors may loose important information. As discussed in [4] using a test database of medical abstracts, LSI performance[2] can improve considerably after 10 or 20 dimensions, peaks between 70 and 100 dimensions, and then begins to diminish slowly. This pattern of performance (initial large increase and slow decrease to word-based performance) is observed with other datasets as well. Eventually performance must approach the level of performance attained by standard vector methods, since with $k = n$ factors A_k will exactly reconstruct the original term by document matrix A in Equation (4). That LSI works well with a relatively small (compared to the number of unique terms) number of dimensions or factors k shows that these dimensions are, in fact, capturing a major portion of the meaningful structure.

5.3. Information Filtering. Information filtering is a problem that is closely related to information retrieval [1]. In information filtering applications, a user has a relatively stable long-term interest or profile, and new documents are constantly received and matched against this standing interest. Selective dissemination of information, information routing, and personalized information delivery are also used to refer to the matching of an ongoing stream of new information to relatively stable user interests.

Applying LSI to information filtering applications is straightforward. An initial sample of documents is analyzed using standard LSI/SVD tools. A users' interest is represented as one (or more) vectors in this reduced-dimension LSI space. Each new document is matched against the vector and if it is similar enough to the interest vector it is recommended to the user. Learning methods like relevance feedback can be used to improve the representation of interest vectors over time.

Foltz [10] compared LSI and keyword vector methods for filtering *Netnews* articles, and found 12%–23% advantages for LSI. Dumais and Foltz in [11] compared several different methods for representing users interests for filtering technical memoranda. The most effective method used vectors derived from known relevant documents (like relevance feedback) combined with LSI matching.

TREC. Recently, LSI has been used for both information filtering and information retrieval in TREC (Text REtrieval Conference), a large-scale retrieval conference conference sponsored by NIST [7, 8]. The TREC collection contains more than $1,000,000$ documents (representing more that 3 gigabytes of ASCII text), 200 queries, and relevance judgements pooled from the return sets of more than 30 systems. The content of the collections varies widely ranging from news sources (AP News Wire, Wall Street Journal, San Jose Mercury News), to journal abstracts (Ziff Davis, DOE abstracts), to the full text of the Federal Register and U.S. Patents. The queries are very long and detailed descriptions, averaging more than 50 words in length. While these queries may be representative of information requests in filtering applications, they are quite unlike the short requests seen in previous IR collections or in interactive retrieval applications (where the average query is only one or two words long). The fact that the TREC queries are quite rich means that smaller advantages would be expected for LSI or any other methods that attempt to enhance users queries.

The big challenge in this collection was to extend the LSI tools to handle collections of this size. The results were quite encouraging. At the time of the TREC conferences it was not reasonable to compute A_k from Figure 1 for the complete collection. Instead, a sample[3] of about $70,000$ documents and $90,000$ terms was used. Such term by document matrices (A) are quite sparse, containing only .001–.002% non-zero entries. Computing A_{200}, i.e. the 200-largest singular values and corresponding singular vectors, by a single-vector Lanczos algorithm [3] required about 18 hours of CPU time on a SUN SPARCstation 10 workstation. Documents not in the original LSI analysis were *folded-in* as previously described in Section 3.3. That is, the vector for a document is located at the weighted vector sum of its constituent term vectors.

Although it is very difficult to compare across systems in any detail because of large pre-processing, representation and matching differences, LSI performance was quite good [8]. For filtering tasks, using information about known relevant documents to create a vector for each query was beneficial. The retrieval advantage of 31% was somewhat smaller than that observed for other filtering tests and is attributable to the good initial queries in TREC. For retrieval tasks, LSI showed 16% improvement when compared with the keyword vector methods. Again the detailed original queries account for the somewhat smaller advantages than previously observed.

[2] Performance is average precision over recall levels of 0.25, 0.50 and 0.75.

[3] Different samples for information retrieval and filtering and for TREC-1 and TREC-2 – see [7, 8] for details.

The computation of A_k for the large sparse TREC matrices A was accomplished without difficulty (numerical or convergence problems) using sophisticated implementations of the Lanczos algorithm from SVDPACKC [3]. However, the computational and memory requirements posed by the TREC collection greatly motivated the development of the SVD-updating procedures discussed in Section 4.

5.4. Novel Applications. Because LSI is a completely automatic method, it is widely applicable to new collections of texts (including to different languages, as described below). The fact that both terms and documents are represented in the same reduced-dimension space adds another dimension of flexibility to the LSI retrieval model. Queries can be either terms (as in most information retrieval applications), documents or combinations of the two (as in relevance feedback). Queries can even be represented as multiple points of interest [17]. Similarly, the objects returned to the user are typically documents, but there is no reason that similar terms could not be returned. Returning nearby terms is useful for some applications like online thesauri (that are automatically constructed by LSI), or for suggesting index terms for documents for publications which require them.

Although term-document matrices have been used for simplicity, the LSI method can be applied to any descriptor-object matrix. We typically use only single terms to describe documents, but phrases or n-grams could also be included as rows in the matrix. Similarly, an entire document is usually the text object of interest, but smaller, more topically coherent units of text (e.g., paragraphs, sections) could be represented as well. For example, LSI has been incorporated as a *fuzzy search* option in NETLIB [5] for retrieving algorithms, code descriptions, and short articles from the NA-Digest electronic newsletter.

Regardless of how the original descriptor-object matrix is derived, a reduced-dimension approximation can be computed. The important idea in LSI is to go beyond the original descriptors to more reliable statistically derived indexing dimensions. The wide applicability of the LSI analysis is further illustrated by describing several applications in more detail.

Cross-Language Retrieval. It is important to note that the LSI analysis makes no use of English syntax or semantics. *Words* are identified by looking for white spaces and punctuation in ASCII text. Further, no stemming is used to collapse words with the same morphology. If words with the same stem are used in similar documents they will have similar vectors in the truncated SVD defined in Figure 1; otherwise, they will not. (For example, in analyzing an encyclopedia, *doctor* is quite near *doctors* but not as similar to *doctoral.*) This means that LSI is applicable to any language. In addition, it can be used for cross-language retrieval – documents are in several languages and user queries (again in several languages) can match documents in any language. What is required for cross-language applications is a common space in which words from many languages are represented.

Landauer and Littman in [20] described one method for creating such an LSI space. The original term-document matrix is formed using a collection of abstracts that have versions in more than one language (French and English, in their experiments). Each abstract is treated as the *combination* of its French English versions. The truncated SVD is computed for this term by combined-abstract matrix A. The resulting space consists of combined-language abstracts, English words and French words. English words and French words which occur in similar combined abstracts will be near each other in the reduced-dimension LSI space. After this analysis, monolingual abstracts can be *folded-in* (see Section 3.3) – a French abstract will simply be located at the vector sum of its constituent words which are already in the LSI space. Queries in either French or English can be matched to French or English abstracts. There is no difficult translation involved in retrieval from the multilingual LSI space. Experiments showed that the completely automatic multilingual space was more effective than single-language spaces. The retrieval of French documents in response to English queries (and vice versa) was as effective as first translating the queries into French and searching a French-only database. The method has shown almost as good results for retrieving English abstracts and Japanese Kanji ideographs, and for multilingual translations (English and Greek) of the Bible [29].

Modeling Human Memory. Landauer and Dumais [19] have recently used LSI spaces to model some of the associative relationships observed in human memory. They were interested in term-term similarities. LSI is often described intuitively as a method for finding synonyms – words which occur in similar patterns of documents will be near each other in the LSI space even if they never co-occur in a single document (e.g., *doctor*, *physician* both occur with many of the same words like *nurse*, *hospital*, *patient*, *treatment*, etc.). Landauer and Dumais tested how well an LSI space would mimic the knowledge needed to pass a synonym test. They used the synonym test from ETS's Test Of English as a Foreign Language (TOEFL). The test consists of 80 multiple choice test items each with a stem word (e.g., *levied*) and four alternatives (e.g., *imposed*, *believer*, *requested*, *correlated*), one of which is the synonym. An LSI analysis was performed on an encyclopedia represented by a 61,000 word by 30,473 article matrix A. For the synonym test they simply computed the similarity of the stem word to each alternative and picked the closest one as the synonym (for the above example *imposed* was chosen – .70 *imposed*, .09 *believed*, .05 *requested*, −.03 *correlated*). Using this method LSI scored 64% correct, compared with 33% correct for word-overlap methods, and 64% correct for the average student taking the test. This is surprisingly good performance given that synonymy relationships are no different than other associative relationships (e.g., *algebra* is quite near words like *algebraic*, *topology*, *theorem*, *Cayley* and *quadratic*, although none are synonyms).

Matching People Instead of Documents. In a couple of applications, LSI has been used to return the best matching *people* instead of documents. In these applications, people were represented by articles they had written. In one application [12], known as the Bellcore Advisor, a system was developed to find local experts relevant to users' queries. A query was matched to the nearest documents and project descriptions and the authors organization was returned as the most relevant internal group. In another application [9], LSI was used to automate the assignment of reviewers to submitted conference papers. Several hundred reviewers were described by means of texts they had written, and this formed the basis of the LSI analysis. Hundreds of submitted papers were represented by their abstracts, and matched to the closest reviewers. These LSI similarities along with additional constraints to insure that each paper was reviewed p times and that each reviewer received no more than r papers to review were used to assign papers to reviewers for a major human-computer interaction conference. Subsequent analyses suggested that these completely automatic assignments (which took less than 1 hour) were as good a those of human experts.

Noisy Input. Because LSI does not depend on literal keyword matching, it is especially useful when the text input is *noisy*, as in OCR (Optical Character Reader), open input, or spelling errors. If there are scanning errors and a word (*Dumais*) is misspelled (as *Duniais*), many of the other words in the document will be spelled correctly. If these correctly spelled context words also occur in documents which contained a correctly spelled version of *Dumais*, then *Dumais* will probably be near *Duniais* in the k-dimensional space determined by A_k (see Equation 2 or Figure 1).

Nielsen et al. in [22] used LSI to index a small collection of abstracts input by a commercially available pen machine in its standard recognizer mode. Even though the error rates were 8.8% at the word level, information retrieval performance using LSI was not disrupted (compared with the same uncorrupted texts). Kukich [18] used LSI for a related problem, spelling correction. In this application, the rows were unigrams and bigrams and the columns were correctly spelled words. An input word (correctly or incorrectly spelled) was broken down into its bigrams and trigrams, the query vector was located at the weighted vector sum of these elements, and the nearest word in LSI space was returned as the suggested correct spelling.

5.5. Summary of LSI Applications. Word matching results in surprisingly poor retrieval. LSI can improve retrieval substantially by replacing individual words with a smaller number of more robust statistically derived indexing concepts. LSI is completely automatic and widely applicable, including to different languages. Furthermore, since both terms and documents are represented in the same space, both queries and returned items can be either words or documents. This flexibility has led to a growing number of novel applications.

5.6. Open Computational/Statistical Issues. There are a number of computational/statistical improvements that would make LSI even more useful, especially for large collections:

- computing the truncated SVD 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 under 500 megabytes of RAM,
- perform SVD-updating (see Section 4) in real-time for databases that change frequently, and
- efficiently comparing queries to documents (i.e., finding near neighbors in high-dimension spaces).

5.7. Related Work. A number of other researchers are using related linear algebra methods for information retrieval and classification work. Schutze [26] and Gallant [13] have used SVD and related dimension reduction ideas for word sense disambiguation and information retrieval work. Hull [16] and Yang and Chute [28] have used LSI/SVD as the first step in conjunction with statistical classification (e.g. discriminant analysis). Using the LSI-derived dimensions effectively reduces the number of predictor variables for classification. Wu et al. in [27] also used LSI/SVD to reduce the training set dimension for a neural network protein classification system used in human genome research.

6. Acknowledgements. The authors would like to thank the referees for their helpful comments and suggestions.

REFERENCES

[1] N. J. Belkin and W. B. Croft, *Information filtering and information retrieval: Two sides of the same coin?*, Communications of the ACM, 35 (1992), pp. 29–38.

[2] M. W. Berry, *Large scale singular value computations*, International Journal of Supercomputer Applications, 6 (1992), pp. 13–49.

[3] M. W. Berry et al., *SVDPACKC: Version 1.0 User's Guide*, Tech. Rep. CS–93–194, University of Tennessee, Knoxville, TN, October 1993.

[4] 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 (1990), pp. 391–407.

[5] J. J. Dongarra and E. Grosse, *Distribution of mathematical software via electronic mail*, Communications of the ACM, 30 (1987), pp. 403–407.

[6] S. T. Dumais, *Improving the retrieval of information from external sources*, Behavior Research Methods, Instruments, & Computers, 23 (1991), pp. 229–236.

[7] ——, *LSI meets TREC: A status report.*, in The First Text REtrieval Conference (TREC1), D. Harman, ed., March 1993, pp. 137–152. National Institute of Standards and Technology Special Publication 500-207.

[8] ——, *Latent Semantic Indexing (LSI) and TREC-2.*, in The Second Text REtrieval Conference (TREC2), D. Harman, ed., March 1994, pp. 105–116. National Institute of Standards and Technology Special Publication 500-215.

[9] S. T. Dumais and J. Nielsen, *Automating the assignment of submitted manuscripts to reviewers*, in SIGIR'92: Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, N. Belkin, P. Ingwersen, and A. M. Pejtersen, eds., Copenhagen, Denmark, June 1992, ACM Press, pp. 233–244.

[10] P. W. Foltz, *Using Latent Semantic Indexing for information filtering*, in Proceedings of the ACM Conference on Office Information Systems (COIS), 1990, pp. 40–47.

[11] P. W. Foltz and S. T. Dumais, *Personalized information delivery: An analysis of information filtering methods*, Communications of the ACM, 35 (1992), pp. 51–60.

[12] 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, 1988, pp. 465–480.

[13] S. I. Gallant, *A practical approach for representing contexts and for performing word sense disambiguation using neural networks*, Neural Computation, 3 (1991), pp. 293–309.

[14] G. Golub and C. V. Loan, *Matrix Computations*, Johns-Hopkins, Baltimore, second ed., 1989.

[15] G. Golub and C. Reinsch, *Handbook for automatic computation II, linear algebra*, Springer-Verlag, New York, 1971.

[16] D. Hull, *Improving text retrieval for the routing problem using Latent Semantic Indexing*, in Proceedings of the Seventeenth Annual International ACM-SIGIR Conference, 1994, pp. 282–291.

[17] Y. Kane-Esrig, L. Streeter, S. T. Dumais, W. Keese, and G. Casella, *The relevance density method for multi-topic queries in information retrieval*, in Proceedings of the 23rd Symposium on the Interface, E. Keramidas, ed., 1991, pp. 407–410.

[18] K. Kukich, *A comparison of some novel and traditional lexical distance metrics for for spelling correction*, in Proceedings of INNC-90-Paris, 1990, pp. 309–313.

[19] T. K. Landauer and S. T. Dumais, *Latent Semantic Analysis and the measurement of knowledge*, in Proceedings of the First Educational Testing Service Conference on Applications of Natural Language Processing in Assessment and Education, 1994. To appear.

[20] T. K. Landauer and M. L. Littman, *Fully automatic cross-language document retrieval using latent semantic indexing*, in Proceedings of the Sixth Annual Conference of the UW Centre for the New Oxford English Dictionary and Text Research, UW Centre for the New OED and Text Research, Waterloo Ontario, 1990, pp. 31–38.

[21] L. Mirsky, *Symmetric gage functions and unitarilly invariant norms*, Q. J. Math, 11 (1960), pp. 50–59.

[22] J. Nielsen, V. L. Phillips, and S. T. Dumais, *Retrieving imperfectly recognized handwritten notes*, Behaviour and Information Technology, (1994). Submitted.

[23] 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.

[24] G. Salton, *Automatic Information Organization and Retrieval*, McGraw Hill, New York, 1968.

[25] G. Salton and C. Buckley, *Improving retrieval performance by relevance feedback*, Journal of the American Society for Information Science, 41 (1990), pp. 288–297.

[26] H. Schutze, *Dimensions of meaning*, in Proceedings of Supercomputing'92, 1992, pp. 787–796.

[27] C. Wu, M. Berry, S. Shivakumar, and J. McLarty, *Neural networks for full-scale protein sequence classification: Sequence encoding with singular value decomposition*, Machine Learning, (1994). To appear.

[28] Y. Yang and C. G. Chute, *An application of least squares fit mapping to text information retrieval*, in Proceedings of the Sixteenth Annual International ACM-SIGIR Conference, 1993, pp. 281–290.

[29] P. G. Young, *Cross-Language Information Retrieval Using Latent Semantic Indexing*, Master's thesis, The University of Knoxville, Tennessee, Knoxville, TN, 1994.