An International Journal
computers & mathematics with applications

PERGAMON Computers and Mathematics with Applications 42 (2001) 1331–1352
www.elsevier.nl/locate/camwa

Estimation of Singular Values of Very Large Matrices Using Random Sampling

M. KOBAYASHI, G. DUPRET*,
O. KING** AND H. SAMUKAWA
IBM Japan, Ltd
Tokyo Research Laboratory
1623-14 Shimotsuruma
Yamato-shi, Kanagawa-ken
242-8502, Japan
⟨mei⟩⟨samukawa⟩@trl.ibm.co.jp

(Received November 1999; revised and accepted January 2001)

Abstract—The singular value decomposition (SVD) has enjoyed a long and rich history. Although it was introduced in the 1870s by Beltrami and Jordan for its own intrinsic interest, it has become an invaluable tool in applied mathematics and mathematical modeling. Singular value analysis has been applied in a wide variety of disciplines, most notably for least squares fitting of data. More recently, it is being used in data mining applications and by search engines to rank documents in very large databases, including the Web. Recently, the dimensions of matrices which are used in many mathematical models are becoming so large that classical algorithms for computing the SVD cannot be used.

We present a new method to determine the largest 10%–25% of the singular values of matrices which are so enormous that use of standard algorithms and computational packages will strain computational resources available to the average user. In our method, rows from the matrix are randomly selected, and a smaller matrix is constructed from the selected rows. Next, we compute the singular values of the smaller matrix. This process of random sampling and computing singular values is repeated as many times as necessary (usually a few hundred times) to generate a set of training data for neural net analysis. Our method is a type of randomized algorithm, i.e., algorithms which solve problems using randomly selected samples of data which are too large to be processed by conventional means. These algorithms output correct (or nearly correct) answers *most* of the time as long as the input has certain desirable properties. We list these properties and show that matrices which appear in information retrieval are fairly well suited for processing using randomized algorithms. We note, however, that the probability of arriving at an incorrect answer, however small, is not naught since an unrepresentative sample may be drawn from the data. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords—Artificial neural networks, Randomized algorithms, Singular value decomposition, Random sampling, SVD.

The authors would like to thank the anonymous referees for their thoughtful comments and suggestions and K. Takeda, H. Nomiyama, and T. Sakairi for helpful conversations.

*Graduate student intern from the University of Tsukuba, gedupret@hotmail.com.

**Graduate student intern from U.C. Berkeley.

"One of the most fruitful ideas in the theory of matrices is that of a matrix decomposition or canonical form. The theoretical utility of matrix decompositions has long been appreciated. More recently, they have become the mainstay of numerical linear algebra, where they serve as computational platforms from which a variety of problems can be solved."

G. W. Stewart
for Gene Golub on his 15th birthday[1]

1. INTRODUCTION

The singular value decomposition (SVD), i.e., the factorization of a matrix A into the product

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

of unitary matrices U and V and a diagonal matrix Σ, has a long and rich history, as chronicled in a paper by Stewart [1]. A formal statement of the existence theorem for the SVD and associated definitions can be found in standard texts on linear algebra, such as [2–4]. Although it was introduced in the 1870s by Beltrami and Jordan for its own intrinsic interest, it has become an invaluable tool in applied mathematics and mathematical modeling. Singular value analysis has been applied in a wide variety of disciplines, most notably for least squares fitting of data [5]. More recently, it has been used in data mining applications and by automated search engines, e.g., *Alta Vista*[2], to rank documents in very large databases, including the Web [6–11]. Recently, the dimensions of matrices which are used in many mathematical models have become so large that classical algorithms for computing the SVD cannot be used. We present a new method to determine the largest 10%–25% of the singular values of matrices which are so enormous that use of standard algorithms and computational packages will strain computational resources available to the average user. If the associated singular vectors are desired, they must be computed by another means; we suggest an approach for their computation.

This paper is organized as follows. In the remainder of this section, we give some very brief background information on the topics related to the remainder of the paper. First, we introduce some concepts which serve as the foundation of randomized algorithms, i.e., algorithms which use statistical (random, or randomized) sampling as a means to solve problems which involve data sets which are enormous. Next, we briefly review how some very simple neural networks can be used to predict properties in data. More specifically, we point to how they can be used for analyzing experimental data and curve extrapolation. The deficiencies of alternative methods, e.g., polynomial and spline fitting are noted. Finally, we turn to a discussion on the singular values of a matrix, i.e., how knowledge of the singular values yields valuable information about a matrix, such as its norm, as well as its sensitivity to roundoff errors during computations. We explain how knowledge of the singular values can be valuable for tuning the performance of information retrieval and ranking systems which are based on vector space models. Some standard algorithms for the computation of the SVD are summarized. In the second section, we present our method to determine the top 10%–25% of the singular values of very large matrices. Variations of the method are also presented. In the third section, we present results from implementations of our method. A very large matrix constructed using data from an industrial text mining study and some randomly generated matrices are considered in our experiments. We conclude with a discussion on possible directions for enhancing our method and open theoretical questions.

[1]Gene Golub was born on February 29.
[2]*Alta Vista* homepage: `http://www.altavista.com`

1.1. Randomized Algorithms

Until the late 1980s, with the exception of a few, isolated examples, randomized algorithms were relegated to use in computational number theory (see, e.g., the seminal papers by Solovay and Strassen [12] on primality testing, and Rabin [13] on applications to number theory and computational geometry). Once their power in speeding up computations or making possible previously impossible tasks came to be recognized by the larger scientific community—particularly computer scientists—the development of new associated techniques and hybridization of classical algorithms with statistical sampling soon followed. Randomized algorithms remain an active area of research [14]. In a 1991 ground breaking survey paper, Karp describes them as follows.

> "A *randomized algorithm* is one that receives, in addition to its input data, a stream of random bits that it can use for the purpose of making random choices. Even for a fixed input, different runs of a randomized algorithm may give different results; thus it is inevitable that a description of the properties of a randomized algorithm will involve probabilistic statements. For example, even when the input is fixed, the execution time of a randomized algorithm is a random variable." (See [15, p. 166].)

Randomized algorithms are often used to solve problems using random samples from a set or sets of data which are so enormous that processing by conventional means is not possible or prohibitively slow.

For example, suppose we are given three large and dense $n \times n$ matrices A, B, and C with integer entries, and we want to know if C is the product of A and B, i.e., we would like to determine if the following statement true:

$$A \times B = C.$$

Multiplication of matrices requires $O(n^3)$ floating point operations or *flops* [3]. A more efficient randomized algorithm which *usually* outputs the correct answer is the following.

- Take a random $n \times 1$ bit vector, i.e., a vector of length n, with all but a single entry equal to zero. The single nonzero entry has the value equal to one. We note that the entry which is nonzero is the randomized variable.
- Compute $B \cdot r$.
- Then compute $A(B \cdot r)$.
- Compute $C \cdot r$.
- Finally, check if $A(B \cdot r) = C \cdot r$.
- If "YES", then continue checking with more vectors, until the desired risk of error is below the user specified threshold. If "NO", then terminate and output negative answer.

Steps 2, 3, and 4 each require n^2 flops, and Step 5 requires n operations, so for each bit vector, a total of $(3n^2 + n)$ are required. For this particular type of randomized algorithm, the probability of an error is one-sided, i.e., a negative answer is always correct, but an affirmative answer is always associated with a very small probability of error. The probability of arriving at an incorrect conclusion decreases as the number of iterations are increased to test the matrix product. Fifteen to 20 iterations suffice for most practical purposes, which requires, at most $(60n^2 + 20n)$ operations. To summarize, for large matrices, this randomized algorithm is a highly reliable and efficient means for determining whether a given matrix is the product of two other given matrices [16]. Since it is not 100% fail-safe, it should be only used for appropriate applications.

One of the important principles underlying the successful development of a randomized algorithm such as ours, is the assumption that a random sample from the very large pool of data under consideration is likely to be representative of the complete set of data. Of course, if more samples are taken, or if the population of each sample is increased, there is a higher probability of selecting a more representative sample from the complete set of data. An example of a set of data which is well suited for randomized sampling is an enormous bag filled with over one million integers from the set $\{0, 1, 2, \ldots, 25\}$. Each of the integers appears with the same frequency as all of the others.

An example of a set of data which is not as well suited is an enormous bag filled with over a million integers, all of which are zero, except for a single integer which is unity. Unless many samples are taken, and each sample is fairly large, the probability of selecting the single integer which is unity is very small, and we would be led to believe that all of the integers in the bag are zero. The problem associated with the data in this second example is related to another important underlying principle known as the *abundance of witnesses*, which was introduced by Karp.

> "Randomized algorithms often involve deciding whether the input data to a problem possesses a certain property .... Often, it is possible to establish the property by finding a certain object called a *witness*. While it may be hard to find a witness deterministically, it is often possible to show that witnesses are quite abundant in a certain probability space, and thus one can search efficiently for a witness by repeatedly sampling from the probability space. If the property holds, then a witness is very likely to be found within a few trials; thus the failure of the algorithm to find a witness in a long series of trials gives strong circumstantial evidence, but not absolute proof, that the input does not have the required property." (See [15, p. 166].)

Randomized algorithms, such as the one we present in Section 2, are used to determine certain properties associated with an enormous set of data. In the problem we consider, the properties are the singular values. An analogous example for our problem involving matrices is an enormous $M \times N$ matrix in which all of the row vectors are $(1, 0, 0, \ldots, 0)$ except one row, which is equal to $(0, 1, 0, 0, \ldots, 0)$. We will explain in detail why this matrix would be poor input for our algorithm in Section 2, after we present our algorithm.

We caution the reader that the algorithm we proposed in this paper belongs to class of randomized algorithms which output a correct or nearly correct answer *most* of the time, however, the probability of arriving at an incorrect answer, however small, is not naught since an unrepresentative sample may be drawn from the large pool of data.

1.2. Neural Networks

The development and design of *artifical neural networks* (ANNs) or *neural networks* were motivated by the remarkable ability of the human brain to process massive amounts of sensory data in parallel, to arrive quickly at decisions, to trigger associated reactions, and to learn from experience. The historical development of the field is described in detail in [17,18].

ANNs are (massively parallel) networks of many processors, each of which may have local memory. The processors, or *neurons*, are connected by unidirectional communication channels which carry numeric data. The units only process data in their local memory and which they receive via their connections. Typically, a set of training data is input into a neural network, e.g., a pair of coordinates from a curve $\{(x_i, y_i)\}$. The neural net is "trained" by inputting the x_i. The differences between the predicted y_i and the actual (known) value for y_j are computed, and this difference is used to adjust the weighting parameters in the neural network model. This adjustment can take place as each pair of data points is evaluated for accuracy or it can take place after output from several pairs or the entire set of training data is evaluated. A fairly large number of data points and many iterations are normally needed to train a network to accurately predict output values. Since the computations to complete a single iteration is very light and are independent for each component, the total amount of computation required to refine an input-output model is usually not so large as to be prohibitive for many applications. Furthermore, the localized nature of the computations performed by neural networks lends itself naturally to parallel processing.

A typical neural network for extrapolating a monotone increasing curve, such as the type we consider in subsequent sections, is illustrated in Figure 1. It operates as follows.

- Begin with a neuron with m synapses with m associated synaptic weights w_{k1}, w_{k2}, $\dots, w_{km}$, a summing junction with bias b_k and activation function $\phi(\cdot)$.
- Input a set of training points $\{(x_i, y_i)\}$, i.e., for each input value x_i, the output value is y_i.
- The neuron operates according to the pair of equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j,$$
$$y_k = \phi\left(u_k + b_k\right),$$

  where u_k is the *linear combiner output*, b_k is the *bias*, and $\phi(\cdot)$ is the *activation function*. The activation function is used to limit the amplitude of the output values. The bias increases or lowers the net input to the activation function.
- The differences between the predicted y_i and the actual value for y_j are computed, and is used to adjust the weighting parameters w_{kj}.

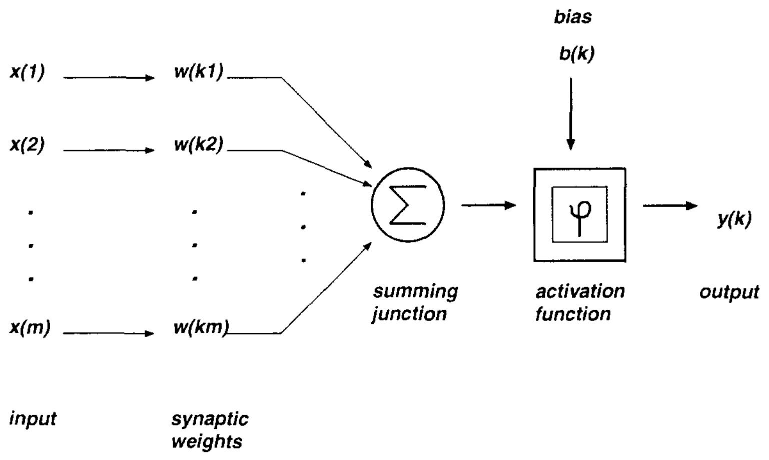

Figure 1. Example of a neuron.

Some examples of activation functions $\phi(\cdot)$ used in many simple implementations of neural networks are the *Heaviside function*

$$\phi(v) = \begin{cases} 1, & v \geq 0, \\ 0, & v < 0, \end{cases}$$

a *piecewise linear function* with a less abrupt transition from naught to unity:

$$\phi(v) = \begin{cases} 1, & v \geq \frac{1}{2}, \\ v, & \frac{1}{2} \geq v > -\frac{1}{2}, \\ 0, & v \leq -\frac{1}{2}, \end{cases}$$

and the *sigmoid function*

$$\phi(v) = \frac{1}{1 + \exp(-av)}.$$

In this paper, we use neural networks to fit experimental data to a monotone, increasing curve. The data we use for training are very small values of x_i and y_i. We predict the singular values of a matrix by extrapolating the curve to very large values of x and y. Simpler curve fitting methods, such as polynomial and spline approximation cannot be used in our application because the values we predict are several times or even an order larger than the values which appear in our training data. Polynomial and spline fitting are appropriate when the data to be estimated lie in the same range as the training data since they oscillate and take on extreme values outside the range of the training data. In our experiments described in later sections, we used a neural net algorithm called *multiple-layer feedforward network* (MLFN) and the program in [19]. *Multilayer* neural nets process data using several layers of neurons before outputting data. *Feedforward* neural nets do not feedback the output data in intermediate layers; the data movement is unidirectional in the forward direction. An illustration of a very simple MLFN with one hidden layer and one output layer is given in Figure 2. Detailed discussions on MLFN algorithms, variations, and enhancements are given in [20]. More sophisticated MLFNs are based on techniques such as the conjugate gradient method, direct line minimization, and stochastic optimization (simulated annealing).

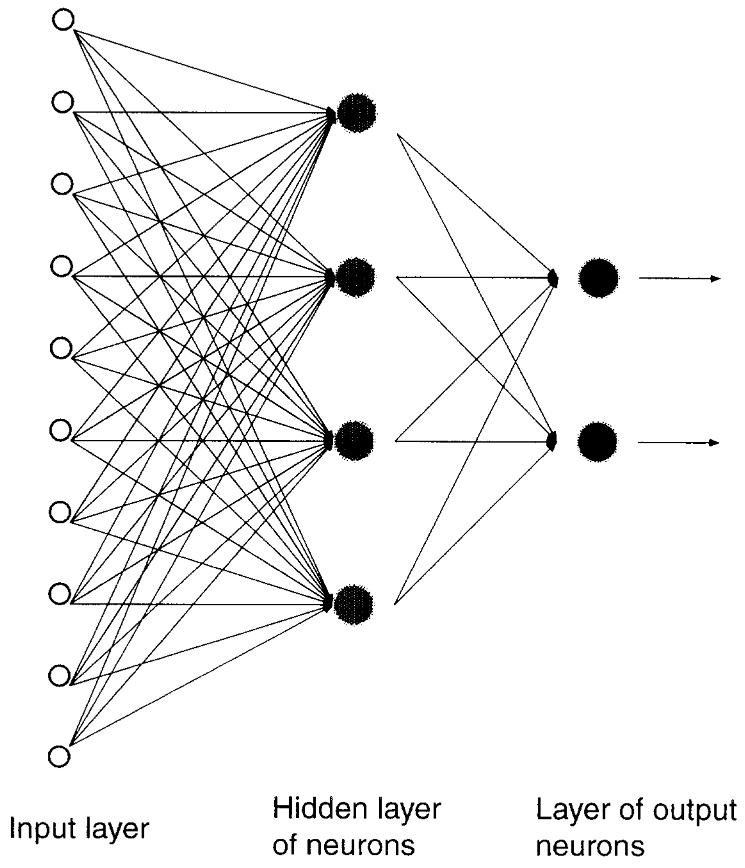

Figure 2. Example of a fully connected feedforward network with one hidden layer and one output layer.

1.3. Singular Values and Properties of Matrices

Accurate estimates of the largest 10%–25% singular values of a matrix are useful for understanding properties of the matrix from a theoretical perspective. For symmetric, positive definite matrices, the singular values are the eigenvalues. For general, rectangular matrices, singular values can be used, among many things, to determine:

- the 2-norm of a matrix;
- the closest distance to any matrix with rank N, whenever the N^{th} singular value can be estimated by our technique; and
- a lower bound for the condition number of a matrix.

We elaborate on these three points. The largest singular value is the 2-norm of a matrix, where the 2-norm of a matrix represents the maximum magnification that can be undergone by any vector when acted on by the matrix. The N^{th} singular value of a matrix can be used to determine the closest distance to any matrix of equivalent dimensions with rank N.

THEOREM. *(See [21].) Let the singular value decomposition of A be given by equation (1) with $r = \mathrm{rank}(A) \le p = \min(m, n)$, and define*

$$A_k = U_k \Sigma_k V_k^{\mathsf{T}} \tag{2}$$

(see Figure 3). Here Σ_k is a diagonal matrix with k nonzero, monotonically decreasing diagonal elements $\sigma_1, \sigma_2, \dots, \sigma_k$, and U_k and V_k are matrices whose columns are the left and right singular vectors of the k largest singular values of A. Unless specified otherwise, the remaining entries of U_k and V_k are zero. Then

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

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. The proof of the theorem is available in many texts, including [2–4].

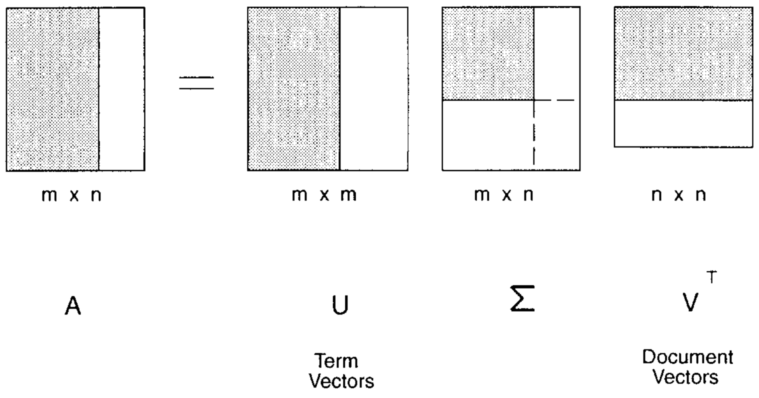

Figure 3. Construction of A_k, the closest rank-k approximation to A, through modification of the singular value decomposition of A. The colored portions of A, U, Σ, and V^{T} remain intact, and entries of the white portions of the matrices are set to zero to construct A_k, U_k, Σ_k, and V_k^{T}.

The *condition number* of a nonsingular matrix A, which we denote by $\kappa(A)$, is one of the simplest and most useful measures of the sensitivity of the linear system associated with the matrix, i.e., $Ax = b$. Although it is defined as the 2-norm of A times the 2-norm of the inverse of A, i.e.,

$$\kappa(A) = \|A\|_2 \cdot \|A^{-1}\|_2,$$

for very large matrices, the computation of the inverse of A and its 2-norm may be too difficult. The condition number is the largest singular value divided by the smallest singular value. The largest singular value for very large matrices can be estimated by our technique or the power method (see Section 1.5). Computation of the smallest singular value of very large matrices is very difficult. Although our technique cannot be always be applied to compute the smallest singular value, if we compute up to the N^{th} singular value, then the quotient $Q = (\sigma_1/\sigma_N)$ will give a lower bound for the condition number of the matrix, i.e., $\kappa(A) \geq Q$. If the matrix A is extremely huge, then a very accurate estimate of σ_N may be costly to compute, however, it is not as expensive to compute a reliable estimate of an upper bound for σ_N using our method (details are given in Section 3). The upper bound for σ_N can be used to compute a lower bound for Q and the condition number $\kappa(A)$. Knowledge of a lower bound for $\kappa(A)$ is useful if it is large, since we know that computations with the matrix A will be very sensitive to roundoff errors. If an estimated lower bound for $\kappa(A)$ is small, we have not gained any new information. We close this subsection with two notes of caution.

(1) Since our estimation method is based on a randomized algorithm, there is a (very small) probability that our estimates are inaccurate.
(2) The bound we obtain is based on empirical observations, not on rigorous mathematical proofs.

1.4. Singular Values and Information Retrieval

As mentioned earlier, the SVD is being used in some automated search and retrieval systems to rank documents in very large databases [22,23] and more recently, the algorithm has been extended to retrieval, ranking, and visualization systems for the Web [6–11]. These systems are based on a preprocessed mathematical model of document-query space. The relationship between possible query terms and documents is represented by an $m \times n$ matrix A, with ij^{th} entry a_{ij}, i.e.,

$$A = [a_{ij}] .$$

The entries a_{ij} consist of information on whether term i occurs in document j, and may also include weighting information to take into account specific properties, such as the length of the document, the importance (or relevance) of the query term in the document, and the frequency of the query term in the document. $A = [a_{ij}]$ is usually a very large, sparse matrix, because the number of keyword terms in any single document is usually a very small fraction of union of the keyword terms in all of the documents.

After creation and preprocessing of the matrix A, the next step is the computation of the singular value decomposition (SVD) of A. Although the computation does not have to take place in real time, it has to be completed quickly enough for very large sparse matrices (at least tens of thousands-by-tens of thousands, and preferably millions-by-millions) to enable frequent updating of the matrix model.

The noise in matrix A is reduced by constructing a modified matrix A_k, from the k largest singular values and their corresponding vectors, i.e.,

$$A_k = U_k \Sigma_k V_k^{\top} .$$

Here we follow the notation used in the theorem by Eckhart and Young given earlier.

Queries are processed in two steps: *query projection* followed by *matching*. In the query projection step, input queries are mapped to *pseudo-documents* in the reduced query-document space by the matrix U_k, then weighted by the corresponding singular values σ_i from the reduced rank, singular matrix Σ_k. The process can be summarized mathematically as

$$q \longrightarrow \hat{q} = q^{\top} U_k \Sigma_k^{-1} ,$$

where q represents the original query vector and $\hat{q}$ the pseudo-document. In the second step, similarities between the pseudo-document $\hat{q}$ and documents in the reduced term document space $V_k^{\top}$ are ranked by measuring the cosine of the angle between the query and the modified document vectors, i.e., by computing the inner product of the normalized vectors.

Although a variety of algorithms based on document vector models for clustering to expedite retrieval and ranking are available [24–26], the type of dimensional reduction scheme described above, known as *latent semantic indexing* (or LSI) [22,23], usually leads to more accurate results since it takes into account *synonymy* and *polysemy*. Synonymy refers to the existence of equivalent or similar terms which can be used to express an idea or object in most languages, and polysemy refers to the fact that some words have multiple, unrelated meanings. Absence of accounting for synonymy will lead to many small, disjoint clusters, some of which should actually be clustered together, while absence of accounting for polysemy can lead to clustering together of unrelated documents.

Information on the spread of the singular values of the document-query matrix, i.e., the relative changes in the singular values when moving from the largest to the smallest can be used to determine an appropriate dimension of a reduced subspace for modeling document-keyword space. Currently, two methods are most commonly used to set the dimension of the subspace.

- Decide *a priori* how many singular values can be computed and set the dimension to be equal to this number, or
- decide on an acceptable range for the dimension, e.g., $d_{\min} \leq \text{dimension} \leq d_{\max}$, and determine a precise value based on whether there is a big relative jump in the distance between two consecutive singular values in the range, i.e., set the dimension to be $i \in [d_{\min}, d_{\max}]$ if $\sigma_i - \sigma_{i-1} \ll \sigma_{i+1} - \sigma_i$.

Before we make a final decision on the dimension of the subspace, we can estimate whether paging will occur, and if so, the extent of paging and associated overhead in time if we use some reliable method to compute the singular vectors, e.g., the Lanczos method.

1.5. Standard Approaches to Computing the SVD

In this section, we review three approaches which are widely used to compute the SVD of matrices.

Householder reflections and Givens rotations

Computation of the SVD of moderate-sized matrices (on the order of a few hundred by a few hundred) is not difficult. If a matrix A is quite small and not necessarily sparse, a reasonable approach is to use *Householder reflections* to bidiagonalize A, i.e., transform A to the form

$$\begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \beta_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & \cdots & 0 & \alpha_n \\ 0 & \cdots & & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & & \cdots & 0 \end{bmatrix}.$$

Next, apply sequences of plane rotators to zero the superdiagonal elements β_i. *Plane rotators* (also called *Givens rotators* and *Givens transformations*) are matrices in which all nondiagonal entries are naught and diagonal entries are unity. Exceptions occur at the four entries, for which

$$A(i,i) = A(j,j) = \cos\theta \qquad \text{and} \qquad A(i,j) = -A(j,i) = -\sin\theta,$$

where θ denotes the angle of rotation (see [3, Chapter 5]). Written out explicitly, *Givens rotators* are matrices which have the form

$$\begin{bmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & \cos\theta & \cdots & -\sin\theta & & & \\ & & & 1 & & & & & \\ & & & \vdots & \ddots & \vdots & & & \\ & & & & 1 & & & & \\ & & & \sin\theta & \cdots & \cos\theta & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{bmatrix}.$$

Note that when the rotator is a 2×2 matrix, it reduces to the standard rotation matrix in a two-dimensional plane.

When Givens rotations are used to zero an entry on the superdiagonal, it normally creates a new nonzero entry on the subdiagonal. For instance, zeroing the (1,2) entry causes recomputation of the (2,1) entry. When another Givens rotation is used to zero the new subdiagonal (2,1) element, a new nonzero entry is normally created in the super-super diagonal (1,3) entry. This process of using a sequence of Givens rotations to eventually remove each superdiagonal entry of a bidiagonal matrix is called "*chasing*" or "*zero chasing*" (see Figure 4) [5]. A sequence of Givens rotations or zero chasing must be performed to zero each $(i, i+1)^{\text{th}}$ element of the bidiagonal matrix, beginning with $i = 1$, then $i = 2, 3, \ldots$.

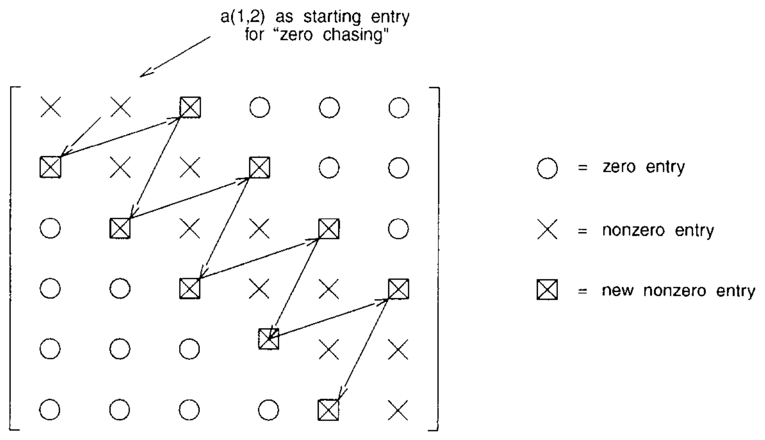

Figure 4. Example of "*zero chasing*" using Givens rotations on a 6×6 bidiagonal matrix.

Use of Householder transformations followed by Givens for computing the SVD will normally:

1. destroy special features of the matrix A (including sparsity);
2. require significant memory; and
3. be computationally slow.

Most of the older, over-the-counter software packages for computing the SVD were designed for solving least squares problems, and they use the Householder plus Givens approach, e.g., [27].

The power method and subspace iteration

If A is very sparse and only a few of the singular values and singular vectors of A are needed in an application (for example, LSI described in Section 1.4), a reasonable approach for computing the SVD of A may be subspace iteration followed by modified Gram-Schmidt. *Subspace iteration* is based on the *power method*—an even simpler algorithm, which is used in many scientific applications to determine the largest eigenvalue and the associated eigenvector of a matrix A. As we shall see, these methods work best when the singular values are distinct and spaced well apart.

In both the power and subspace iteration methods, we consider the matrix products

$$B = A^{\mathsf{T}} \cdot A \qquad \text{and} \qquad B = A \cdot A^{\mathsf{T}},$$

then set the matrix with smaller dimension to be B. The eigenvalues λ_i of B are the square of the singular values σ_i of A, i.e., $\lambda_i = \sigma_i^2$. Eigenvalue determination for this problem is not as difficult as for general matrices. Since B is symmetric, positive semidefinite, its eigenvalues are real, and all of its Jordan boxes are 1-by-1. In general eigenvalue finding programs, a substantial portion of extra code is devoted to tests for determining the (possible) existence of multiple roots and the size of associated Jordan boxes. And it is very difficult to write fail-safe, fast code which processes multiple and very close roots.

In the *power method* [2–4], we begin with an arbitrary vector v of unit length. In most cases, the vector has a nontrivial component in the direction of the eigenvector v_1 associated with the largest eigenvalue λ_1.[3] Then we compute the limit of the *Rayleigh quotient* of the matrix B, defined as

$$\lambda_1 = \lim_{m\to\infty} \frac{v^{\mathsf{T}} B^{m+1} v}{v^{\mathsf{T}} B^m v}.$$

This computation can be reduced to matrix-vector and vector-vector multiplications to avoid explicit matrix-matrix multiplications, i.e., $B = A^{\mathsf{T}} \cdot A$, the $v^{\mathsf{T}} B v$ is computed as follows:

$$v^{\mathsf{T}} B v = v^{\mathsf{T}} \left(A^{\mathsf{T}} (A v) \right).$$

Similarly, multiplication begins from the rightmost vector and matrix when $B = A \cdot A^{\mathsf{T}}$.[4] To determine the second largest eigenvalue, we select a starting vector of unit length with no component in the direction of the eigenvector v_1. Subsequent eigenvalues λ_n can be determined by using a starting vector with no component in the directions of the $(n-1)$ largest eigenvectors $v_1, v_2 \ldots, v_{n-1}$, corresponding to the $(n-1)$ largest eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$.[5]

As many eigenvalues as desired can be computed this way, in theory, so long as the eigenvalues are distinct and are spaced well apart; if two or more eigenvalues are very close in values, it is very difficult to separate them during the iterative process. This "sequential" approach of determining progressively smaller eigenvalues using the power method is not used in practice. The standard practice is to compute the desired number of eigenvalues simultaneously, using subspace or simultaneous iteration followed by modified Gram-Schmidt to ensure orthogonality of the recovered eigenvectors (details can be found in [3]). Use of the modified, rather than classical, Gram-Schmidt is recommended since numerical roundoff often leads to poor results when the classical method is used.

Lanczos algorithms for symmetric, positive semidefinite matrices

A good algorithm for computing some (but not all) of the singular values and the associated singular vectors of a large, sparse matrix A is to apply Lanczos tridiagonalization to the square

[3]Even if the starting vector has essentially no component in the direction of v_1, round-off errors will usually accumulate during the iterative computations to generate a component in the direction.
[4]If we just want $v^{\mathsf{T}} B v$, we would be better off computing the dot product $Av \cdot Av$ (which would reduce the work by one matrix-vector multiplication), however, we would like to know the value of Bv.
[5]During the computation of λ_n, after several iterations, round-off errors usually begin to contribute components in the direction of $v_1, v_2 \ldots v_{n-1}$. Orthogonalization with respect to these vectors needs to performed every several steps to ensure orthogonality with respect to v_n.

matrix $B = A^{\top}A$. Note that B should be computed implicitly to minimize the use of memory. Since B is symmetric, positive definite, Lanczos tridiagonalization will convert it to the form

$$\begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ddots & \vdots \\ 0 & \beta_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & 0 & \beta_{n-1} & \alpha_n \end{bmatrix},$$

without many of the difficulties associated with the Lanczos method for more general matrices. A fast, reliable, and parallelizable, eigenvalue routine, such as the Sturm sequence method can be used to compute the eigenvalues of B. Unfortunately, the associated eigenvectors must be computed separately. Concise references to the algorithms are given in [3,4]. Variations of the algorithm which exploit special properties of the input matrix are given in [28]. The theory is given in the first volume and programming code in an outdated version of FORTRAN in the second volume.

Specialized software packages for computing the SVD of very large matrices using the Lanczos algorithm are the subroutine SSVDC in LINPACK [29,30], LANSO [31], and LAPACK and ScaLAPACK [2,32,33]. SVDPACK and SVDPACKC are two Lanczos software packages which have been used extensively for information retrieval by academic institutions or for noncommercial purposes [32,34].

In our implementations of the Lanczos algorithm for (partial) tridiagonalization of a symmetric matrix [9,10], we followed the algorithm in [4, Section 13.1.1, pp. 288–289]. In straightforward implementations of the algorithm, the computed Lanczos vectors ceased to be mutually orthogonal after some steps, and duplicate copies of eigenvalues were recovered. The virtues of maintaining orthogonality of the Lanczos vectors and several useful techniques for carrying out orthogonalization, including full reorthogonalization, selective orthogonalization, (with and without modifications), and Scott's orthogonalization are given in [4,35,36]. Details of the various versions of our implementations, including a discussion of dynamic data structures are given in [9,10].

2. SAMPLING ALGORITHMS FOR DETERMINING SINGULAR VALUES

In the first half of this section, we present a randomized sampling algorithm to estimate the eigenvalues of a symmetric, positive, semidefinite matrix and a couple of variations. Further possible variations and their extensions are discussed. In the latter half, we discuss types of matrices which are suitable for processing using our algorithm. Most randomized algorithms require that the input data possess some fundamental properties. We outline these properties and discuss why we can expect very large matrices which appear in information retrieval applications to be (or be fairly) suitable for input.

2.1. Sampling Algorithms

In this section, we present a sampling algorithm and two variations, and we elaborate on how further variations can be devised. The best variation to use in a given situation depends on the special properties of the matrix, its size and available computational resources.

ALGORITHM 1. Let A denote a very large $M \times N$ matrix whose singular values σ_i cannot be computed due to the enormity of its size.[6] Construct a smaller matrix $A^{(1)}$ by randomly selecting P ($P < M$) rows from the very large matrix. Compute the singular values $\sigma_i^{(1)}$ of this

[6]If a user has sufficient memory resources, usually, the primary limitation on computations involving very large matrices is paging.

smaller matrix $A^{(1)}$ using any standard method, such as the Lanczos method followed by Sturm sequencing [4]. Repeat this process 50 times (or any number of times which is sufficiently large to allow statistical analysis), i.e., construct matrices $A^{(i)}$, $i = 1, 2, 3, \ldots, 50$, by taking different random samples of rows of A each time. For each $A^{(i)}$, compute the largest singular value $\sigma_1^{(i)}$, the second largest singular value $\sigma_2^{(i)}$, the third largest singular value $\sigma_3^{(i)}$, and so forth until however many singular values are desired. To estimate σ_k, the k^{th} singular value of the original, full matrix A, plot the statistical spread of the $\sigma_k^{(i)}$, i.e., the singular values of the $A^{(i)}$, and compute the mean ${}^P\sigma_k$. Next, vary P, the number of randomly selected rows, and repeat this process. Finally, plot P versus ${}^P\sigma_i$. The graph will be a smooth curve, which can be used to obtain a good estimate of σ_i by estimating the value of σ_i when $P = M$ through extrapolation. Extrapolation can be performed manually, by a human or with a software tool, such as neural nets, see, e.g., algorithms in [19].

Note that if $P \ll M$ and $P \ll N$, a standard random number generator available on a system library can be run to select the rows to generate the small matrix. This allows for the small possibility that the same row may be selected twice. If P is not extremely small compared to M and N, then it is better to run a program after the random number generating program to check that the row has not already been selected. This note also applies to Algorithms 2 and 3 described below. We used the standard deviation as a guide for the estimated error, however, we do not know how inherent errors in our method will affect the accuracy of our estimates.

ALGORITHM 2. Let A denote a very large $M \times N$ matrix whose singular values σ_i cannot be computed due to the enormity of its size. Construct a smaller matrix ${}^PA$ by randomly selecting P $(P < M)$ rows from the very large matrix. Compute the singular values ${}^P\sigma_i$ of this smaller matrix ${}^PA$ using any standard method. Carry out this process for a series of P, e.g., $P = 20, 21, 22, \ldots, 120$. For each σ_i, plot P versus the estimates ${}^P\sigma_i$. Compared with the data from the first algorithm, the graph will be a curve with considerable noise, however, it can be used to obtain a good estimate of σ_i through extrapolation since there are so many sampling points. Extrapolation can be performed manually, by a human, or a software extrapolation tool, such as neural nets. If we use neural nets, there are many more points to be input for training and more noise in the data (since we did not take many samples for each P to compute an average estimate for ${}^P\sigma_i$), so the quality of the results compared with those from the first algorithm is not known. What is certain is that significantly more computation will be needed to train the neural net for the second algorithm.

ALGORITHM 3. A hybrid of Algorithms 1 and 2 can be used to generate a curve for estimating the singular values. Let A denote a very large $M \times N$ matrix whose singular values σ_i cannot be computed due to the enormity of its size. Construct a smaller matrix ${}^PA$ by randomly selecting P $(P < M)$ rows from the very large matrix. Compute the singular values ${}^P\sigma_i$ of this smaller matrix ${}^PA$ using any standard method. Carry out this process for a series of P evenly or unevenly spaced. If we carry out the process more than once for some P, we take the average of the singular values. For each σ_i, plot P versus the estimates ${}^P\sigma_i$. The graph can be used to obtain an estimate of σ_i if we use extrapolation. To obtain a nice estimate, we would like to either have estimates for the singular values for many values of P or many runs for each P or an intermediate value of both. Extrapolation can be performed manually, by a human, or a software extrapolation tool, such as neural nets.

2.2. Properties of Input Matrices

The randomized sampling algorithms we presented in this section are well suited for certain types of matrices and not so well for others. The suitability of a matrix is not fixed. It can be improved by increasing the sizes of the samples, the frequency of repeated sampling for a given size, and the number of data points to be generated for curve fitting. We give an example below to illustrate these features.

A matrix which has the potential of being highly ill-suited for our sampling algorithms is a very large $M \times M$ diagonal matrix A, with diagonal elements

$$A = \mathrm{diag}(\sigma_1, 1, \dots, 1),$$

where $\sigma_1 \gg 1$. Suppose that in our initial sets of experiments, we sample a successively increasing set of rows P_1, P_2, $\dots$, P_k, where $P_k \ll M$ and k is relatively small. For this set of sampling experiments, the probability that the first row will be sampled in any of the runs is very small, and we may be misled to believe that the largest several singular values of the matrix are all unity. This situation can be avoided by modifying the experiments using any one or a combination of three different enhancements.

- Repeat (or increase the number of repetitions of) experiments for a given sample size P_i and average the results, paying special attention to note if a few runs deviate a lot from the average.[7]
- Increase the number of rows to be sampled in each of the runs and especially so that P_k is no longer relatively tiny with respect to the matrix dimension M.
- Increase the number of different size samples, i.e., increase k for P_1, P_2, $\dots$, P_k.

All three of these enhancements are designed to increase the probability that the first row will be sampled, however, the first enhancement is particularly effective and should always be possible from a hardware perspective since the frequency of runs is increased, but not the sizes of the matrices in each of the runs.

We describe why the enhancements can be expected to improve the accuracy of the solution. First, note that whenever the first row is selected to be in the sample, the largest singular value of the $P_i \times M$ submatrix will be equal to σ_1, and all other singular values equal to unity. When we repeat or increase the number of experimental runs for a fixed sample size, we increase the probability that the first row will be selected in one of the runs. Whenever the first row is selected, the largest singular value will be (correctly) identified as σ_1. For runs which do not sample the first row, the largest singular value will be (incorrectly) identified as unity. As long as the first row is sampled at least once in one of the experimental runs, the average of the results will lead to the estimate that the largest singular value is larger than unity. For small samples, i.e., for small P_i, the probability of sampling the first row for each individual run is small, but if the number of runs is increased significantly, it becomes highly probable that the first row will be selected in at least one of the runs. This means that the average estimate for the largest singular value can be expected to be a little larger than unity. For a larger sample, i.e., for $P_j > P_i$, the probability of sampling the first row for each run increases. (Note that the computational and memory cost of each run also increases.) When we average the results from all of the experiments with sample size P_j, the average estimate for the largest singular value is expected to be a little larger than for the averages of the runs in which P_i samples were taken, but smaller than σ_1. Similarly, it follows that we can expect the estimate for the largest singular value to increase as a function of the sample size when experiments are repeated with sufficient frequency. Similar types of arguments for the other two enhancements can be used to show that the accuracy of the estimate for σ_1 is likely to increase if they are implemented. The latter two enhancements are likely to lead to slightly noisier data to be used for curve fitting.

The enhancements we suggested above are sufficiently general that they are liable to catch problems with problematic matrices. By *problematic matrices*, we mean matrices of very large dimension which have one or just a few rows (or columns) which are unrepresentative of the rows (or columns) of the matrix as a whole. These peculiar rows (or columns) violate the assumption that there will be a abundance of witnesses (a principle described earlier in Section 1.1).

[7]When a few runs have very high deviation from the average, and others have relatively little, it should be taken as a warning signal that the number of experimental runs should be increased to improve the accuracy of results.

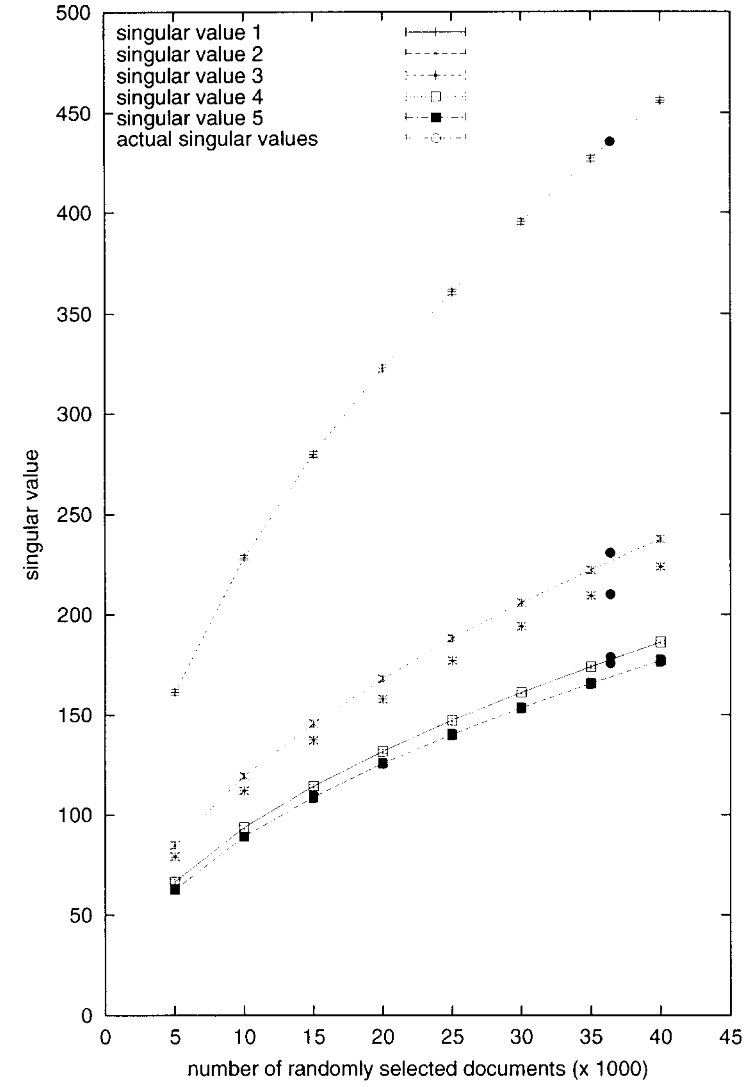

Figure 5. Curves for approximating the largest five singular values of a matrix allowing for the possibility of duplicate sampling.

3. NUMERICAL EXPERIMENTS

We implemented our algorithm using two different types of data:

1. a matrix constructed from industrial text mining data, and
2. randomly generated positive semidefinite, square matrices.

To fit the output from the algorithms to a curve for extrapolation, we used a neural net algorithm called *multiple-layer feedforward network* (MLFN) and the program in [19].

3.1. Text Mining Matrix

In the first set of experiments, we considered a 36403×10000 matrix from a text mining problem. The matrix represents data to be input into an automatic retrieval system based on a variation and enhancement of LSI (described in Section 1.4). It is sufficiently small that we can use a software package we wrote based on the Lanczos algorithm to compute all of the singular values and vectors and compare results with our statistical estimation method.

We took random samples of the rows of the matrix and computed the largest five singular values of the (smaller) matrix constructed from the randomly sampled rows. We repeated the process 100 times and computed the mean and the standard deviation.

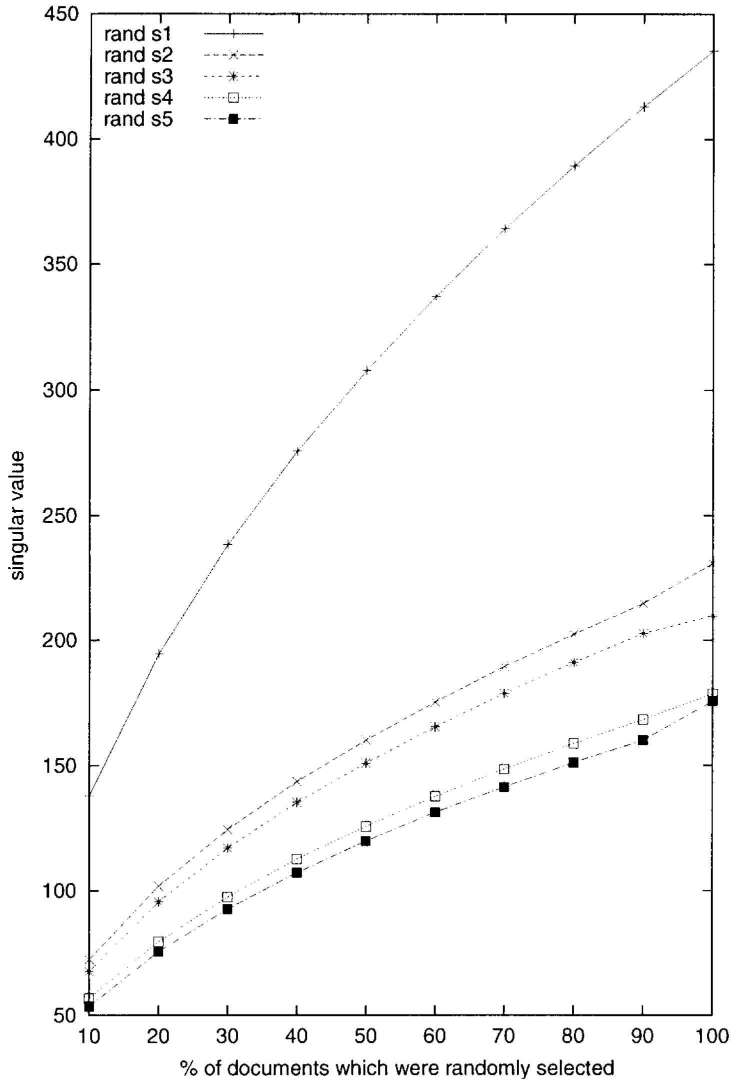

Figure 6. Curves for approximating the largest five singular values of a matrix with no duplicate sampling.

We performed two types of experiments.

1. Rows were allowed to be selected more than once when constructing a small matrix out of randomly sampled rows from the full document-keyword matrix.
2. Rows were not allowed to be selected more than once when constructing a small matrix out of randomly sampled rows from the full document-keyword matrix.

Results from our experiments are given in Figure 5 for the first set of experiments and Figure 6 for the second set. The corresponding numerical data given in Tables 1a–e and Table 2, respectively.

In Experiment 1, since we were allowed to take duplicate copies of rows, we sampled from 15% up to 110% of the rows and plotted the results together with the exact singular values. Note that sampling 110% of the rows means that some rows will be sampled at least twice. Although our primary motivation for using this technique is to reduce the size of the matrix involved in computations, we decided to sample more than the original size matrix out of curiosity, i.e., just to observe what happens. The results from our experiments match very well with the actual singular values; the first and fourth singular values lie on the curve. They are surprisingly good when we consider that we allow rows to be selected twice—which is what occurred when we took 40, 000 rows at random.

Table 1a. σ_1 of a document-query matrix.

No. Docs. Sampled	Estimated Singular Value	Standard Deviation	Std. Dev. as a % of Sing. Val.
5000	161.540	1.48158	0.9172
10000	228.516	1.28316	0.5615
15000	279.560	1.35796	0.4857
20000	322.758	1.58687	0.4917
25000	360.770	1.47101	0.4077
30000	395.405	1.44595	0.3657
35000	426.768	1.34821	0.3159
36403	435.24*	0	0
40000	456.208	1.28724	0.2822

*actual value of σ_1

Table 1b. σ_2 of a document-query matrix.

No. Docs. Sampled	Estimated Singular Value	Standard Deviation	Std. Dev. as a % of Sing. Val.
5000	84.7095	1.69861	2.0052
10000	119.331	1.50119	1.2580
15000	145.622	1.78570	1.2262
20000	168.087	1.58000	0.9447
25000	187.996	1.65415	0.8798
30000	205.775	1.72897	0.8402
35000	222.130	1.67498	0.7541
36403	230.74*	0	0
40000	237.510	1.51121	0.6363

*actual value of σ_2

Table 1c. σ_3 of a document-query matrix.

No. Docs. Sampled	Estimated Singular Value	Standard Deviation	Std. Dev. as a % of Sing. Val.
5000	79.1119	1.85479	2.3445
10000	111.990	1.69140	1.5103
15000	137.308	1.50624	1.0970
20000	157.945	1.71470	1.0856
25000	177.098	1.50329	0.8488
30000	193.911	1.76334	0.9094
35000	209.351	1.65003	0.7882
36403	209.897*	0	0
40000	223.837	1.54668	0.6910

*actual value of σ_3

3.2. Randomly Generated Symmetric, Positive Semidefinite Matrices

In a second set of experiments, we generated random 50×50, 100×100, and 160×160 symmetric, positive, semidefinite matrices and used it to test our method. The matrices were generated by taking the product of a rectangular matrix (for which one dimension was 500) and its transpose, where the entries of the the matrices were generated at random using the rand($\cdot$) function provided in standard *C* libraries.

For experiments for each matrix, we took random samples of the rows of the matrix and computed the largest five singular values. We repeated the process 100 times and computed the

Table 1d. σ_4 of a document-query matrix.

No. Docs. Sampled	Estimated Singular Value	Standard Deviation	Std. Dev. as a % of Sing. Val.
5000	66.5506	1.10516	1.6613
10000	93.5245	0.910262	0.9733
15000	114.257	0.877629	0.7681
20000	131.759	0.974438	0.7396
25000	147.268	0.834012	0.5663
30000	161.239	0.847771	0.5258
35000	174.042	0.828651	0.4761
36403	178.821*	0	0
40000	186.166	0.831076	0.4464

*actual value of σ_4

Table 1e. σ_5 of a document-query matrix.

No. Docs. Sampled	Estimated Singular Value	Standard Deviation	Std. Dev. as a % of Sing. Val.
5000	62.6745	2.18304	3.4831
10000	89.0226	2.29186	2.5747
15000	108.730	2.75832	2.5368
20000	125.721	2.45171	1.9501
25000	140.036	2.61187	1.8651
30000	153.438	2.66805	1.7388
35000	165.746	2.65614	1.6025
36403	175.75*	0	0
40000	177.134	2.85345	1.6108

*actual value of σ_5

Table 2. Estimates for singular values of a document-query matrix.

No. Docs Sampled	% of Docs.	Est. for σ_1	Est. for σ_2	Est. for σ_3	Est. for σ_4	Est. for σ_5
3640	10%	137.885	72.451	67.6975	56.8654	53.4614
7280	20%	194.670	101.748	95.4731	79.6399	75.6355
10920	30%	238.387	124.199	116.824	97.3973	92.5838
14560	40%	275.452	143.469	135.096	112.486	107.040
18200	50%	307.790	160.052	150.843	125.528	119.603
21840	60%	337.201	175.377	165.372	137.502	131.113
25480	70%	364.202	189.472	178.743	148.400	141.214
29120	80%	389.343	202.348	191.060	158.715	151.038
32760	90%	413.009	214.713	202.692	168.301	160.030
36403*	100%	435.240*	230.740*	209.897*	178.821*	175.753*

*actual values

mean and the standard deviation. Results from our experiments using neural networks for curve fitting and extrapolation are given in Tables 3a–f. They show that error for the predicted values are at most 5%, usually at most 3%, and sometime well below 1%. Data from Table 3a are plotted in Figures 7 and 8; data for estimating the first five singular values are shown in Figure 7 and a close-up of the curves for the second to fifth singular values is shown in Figure 8. The plots show fairly typical behavior of the singular values of a matrix, i.e., the largest singular value is usually well separated from the other singular values; and singular values tend to clump together, making estimation of all but the largest singular value more difficult.

Table 3a. Singular values of a randomly generated matrix (433 rows).

No. Docs. Sampled	σ_1	σ_2	σ_3	σ_4	σ_5
20	0.72723	0.44620	0.43252	0.42267	0.413197
30	0.84923	0.46738	0.45406	0.44405	0.435080
40	0.95757	0.48519	0.47275	0.46237	0.453525
50	1.05398	0.50147	0.48858	0.47823	0.470148
60	1.14197	0.51450	0.50179	0.49197	0.483833
100^e	2.91868	0.79740	0.78693	0.76865	0.763555
100^p	2.91829	0.78334	0.78460	0.77399	0.773992
Error	+0.01%	+1.79%	+0.30%	−0.69%	−1.35%

Table 3b. Singular values of a randomly generated matrix (408 rows).

No. Docs. Sampled	σ_1	σ_2	σ_3	σ_4	σ_5
20	5.57267	3.75887	3.66296	3.59168	3.52545
30	6.41632	3.90178	3.80677	3.73416	3.67511
40	7.18334	4.02225	3.93523	3.86842	3.80911
50	7.84493	4.13141	4.04578	3.98116	3.91956
60	8.46411	4.23245	4.14977	4.08335	4.02310
100^e	20.6324	6.17368	5.97194	5.89014	5.83628
100^p	20.7545	6.19614	6.07404	5.98334	5.89627
Error	−0.59%	−0.36%	−1.68%	−1.56%	−1.02%

100^e = exact value

100^p = predicted value

Table 3c. Singular values of a randomly generated matrix (414 rows).

No. Docs. Sampled	σ_1	σ_2	σ_3	σ_4	σ_5
20	5.83888	3.84466	3.73697	3.65872	3.59012
30	6.75588	4.00356	3.90652	3.82936	3.76324
40	7.55936	4.13097	4.03432	3.96119	3.89693
50	8.29249	4.24866	4.15304	4.07951	4.01612
60	8.94493	4.35504	4.25637	4.18587	4.12149
100^e	22.085	6.41339	6.24698	6.19425	6.18721
100^p	21.971	6.36723	6.22210	6.12456	6.03742
Error	+0.52%	+0.72%	+0.40%	+1.14%	+2.48%

Table 3d. Singular values of a randomly generated matrix (50 rows).

No. Docs. Sampled	σ_1	σ_2	σ_3	σ_4	σ_5
10	3.33885	2.45437	2.22538	2.03553	1.86694
15	3.83106	2.70773	2.45462	2.28319	2.09810
20	4.27363	2.90072	2.66843	2.47278	2.31086
25	4.69209	3.09658	2.83547	2.64588	2.46873
30	5.07048	3.26885	3.00324	2.79438	2.61056
100^e	6.33911	3.78967	3.65209	3.40648	3.14792
100^p	6.37887	3.81820	3.53673	3.27863	3.19311
error	−0.62%	−0.75%	+3.26%	+3.90%	−1.41%

100^e = exact value

100^p = predicted value

Table 3e. Singular values of a randomly generated matrix (100 rows).

No. Docs. Sampled	σ_1	σ_2	σ_3	σ_4	σ_5
20	3.6205	2.45983	2.29845	2.19441	2.11359
30	4.23928	2.64988	2.50786	2.40488	2.30662
40	4.76766	2.81614	2.67117	2.57005	2.48070
50	5.24972	2.98001	2.83820	2.71885	2.62267
60	5.69946	3.12268	2.98378	2.86098	2.75399
100^e	7.20429	3.59851	3.45972	3.41632	3.19492
100^p	7.33406	3.76069	3.52718	3.37614	3.25973
Error	−1.77%	−4.31%	−1.91%	+1.19%	−1.99%

Table 3f. Singular values of a randomly generated matrix (150 rows).

No. Docs. Sampled	σ_1	σ_2	σ_3	σ_4	σ_5
20	3.36575	2.28341	2.18359	2.11564	2.05212
30	3.90589	2.41334	2.33068	2.25813	2.19526
40	4.40210	2.56226	2.46582	2.39370	2.32850
50	4.82649	2.65764	2.57050	2.50631	2.44106
60	5.21732	2.76016	2.67073	2.59856	2.54005
100^e	7.95179	3.39929	3.34484	3.33042	3.25039
100^p	7.84338	3.49566	3.34637	3.24582	3.15305
Error	+1.38%	−2.76%	−0.05%	+2.61%	+3.09%

100^e = exact value
100^p = predicted value

3.3. Application of Experimental Results to Information Retrieval

As we noted earlier in Section 1.3, to obtain an estimate for a lower bound for the condition number of a huge matrix (which cannot be easily manipulated due to its size), one could compute an estimate for the largest eigenvalue using the power method and check it with our method. Our numerical experiments show that we can find a reliable estimate for an upper bound for the smallest nonzero singular value $\sigma_{\min}$ using data from our method. Our empirical observations indicate that the estimation curve for any singular value *usually* lies below any line tangent to the curve. Take the curve for the smallest singular value for which we have data and use linear extrapolation, i.e., a tangent line, to find a bound $M \geq \sigma_{\min}$. This method appears to work well when the percentage of sampled documents is less than 20% and works less well as the percentage of sampled documents increases, is close to the total number of documents in the database—particularly when sampling is carried out allowing for replacement (e.g., the fifth singular value in Figure 5). When replacement of sampled data is allowed, it will almost inevitably introduce error into the estimate when large samples are taken, since the probability that at least one of the data will be drawn twice will be very high.

We found that even very crude implementations of our algorithm with very few points allow accurate determination of clustering patterns of the singular values of matrices which have been randomly generated and matrices which appear in information retrieval applications.

The matrices we considered from information retrieval applications were those which modeled the documents in the database as vectors, each coordinate of which represents an attribute. It has been observed that the singular values of these matrices tend to be unevenly distributed. The largest singular value tends to be set apart from the others, then the rest usually cluster together (several or more at a time) to form many clusters. The clustering brings forth both good and bad features for computational scientists. Clustering makes it difficult to accurately distinguish between singular values within the same cluster, however, a good approximation of a singular vector can usually be computed by orthogonalizing only with respect to those singular vectors whose corresponding singular values are in the same cluster [35,36]. Just the knowledge of the clustering patterns of the singular values can be helpful since they allow more accurate estimation of computations which need to be performed to compute singular triplets (i.e., singular values and their associated pairs of singular vectors) of a matrix. A user can decide how many singular triplets he/she will compute based on the available computational resources.

4. FUTURE DIRECTIONS FOR RESEARCH

There are many possible directions for future study associated with the work we presented in Sections 1–3 of this report. In this section, we elaborate on some straightforward tasks and open theoretical questions.

Our experimental results seem to indicate that when two consecutive singular values σ_i and σ_{i+1} are relatively close, our method tends to underestimate the larger singular value and overestimate the smaller singular value, i.e.,

$$\sigma_{i(\text{estimated})} < \sigma_{i(\text{actual})} \qquad \text{and} \qquad \sigma_{i+1(\text{estimated})} > \sigma_{i+1(\text{actual})}.$$

More data needs to be collected to see if mixing of neighboring singular values occurs during our estimation process, and if so, why. A complete explanation for the mixing should include details on what factors (e.g., the spread in singular values and the magnitude of the singular values) influence the extent of mixing.

A second topic for follow-up studies is the choice of the interpolation, i.e., whether neural nets are a good choice or whether a simpler method exists. The choice of the neural net also needs to be studied. We selected MLFN because it is well known and over-the-counter software is readily available, however, we do not know if a better neural net exists; better in terms of ease of use, computational requirements, or results (i.e., reliable and accurate predictions). The optimal format for data to input for training needs to be investigated. For instance, it is not clear how many data points are needed for statistical averaging (for Method 1) or if noisy data but more training data points (for Method 2) is better or if a hybrid of Methods 1 and 2 (i.e., Method 3) is best. If a hybrid looks promising, fine tuning the mix needs to be examined.

A third topic for further study is error analysis. Currently, we do not have a means for computing sharp error bounds for our estimates of singular values. We have taken the standard deviation to be the error in the singular values of the matrices comprised of rows sampled from the original, full matrix, and it appears to yield reasonable error bars for the points in our graphs. Errors from interpolation using MLFN need to be understood.

A major challenge well worth attempting is to develop an accurate and inexpensive method for estimating the singular vectors associated with the singular values computed using our sampling method; we would like to avoid carrying out Lanczos-based computations. One approach may be to compute the singular vectors of the sampled matrices to see if they converge to the singular vectors. Unfortunately, even if this method works, it would require considerable computational work because we would have to perform multidimensional interpolation. Furthermore, since we sample either rows (or columns), we would only be able to estimate just the left (or just the right) singular vectors. To estimate both the left and right singular vectors, we would have to carry out the process twice—first sampling rows and carrying out multidimensional interpolation, then sampling columns for multidimensional interpolation.

REFERENCES

1. G. Stewart, On the early history of the singular value decomposition, *SIAM Review* **35**, 551–566 (December 1993), anonymous ftp: `thales.cs.umd.edu, directory pub/reports`.
2. J. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, (1997).
3. G. Golub and C. Van Loan, *Matrix Computations*, Third Edition, John Hopkins University Press, Baltimore, MD, (1996).

4. B. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, PA, (1998).
5. C. Lawson and R. Hanson, *Solving Least Squares Problems*, Prenctice-Hall, Englewood Cliff, NJ, (1974); Currently available through SIAM, Philadelphia, PA (1995).
6. R. Baeza-Yates and B. Ribeiro-Neto, Editors, *Modern Information Retrieval*, ACM Press and Addison-Wesley, New York, (1999).
7. M. Berry and M. Browne, *Understanding Search Engines*, SIAM, Philadelphia, PA, (1999).
8. M. Berry, S. Dumais and G. O'Brien, Using linear algebra for intelligent information retrieval, *SIAM Review* **37** (4), 573–595 (December 1995).
9. G. Dupret and M. Kobayashi, Information retrieval and ranking on the Web: Benchmarking studies I, *IBM TRL Research Report, RT0300* (March 1999).
10. O. King and M. Kobayashi, Information retrieval and ranking on the Web: Benchmarking studies II, *IBM TRL Research Report, RT0298* (February 1999).
11. M. Kobayashi *et al.*, Multi-perspective retrieval, ranking and visualization of Web data, In *Proc. International Symposium on Digital Libraries (ISDL) '99*, Tsukuba, Japan, September 28–29, 1999, pp. 159–162, (1999).
12. R. Solovay and V. Strassen, A fast Monte-Carlo test for primality, *SIAM Journal of Computation* **6**, 84–85 (1977).
13. M. Rabin, Probabilistic algorithms, In *Algorithms and Complexity*, (Edited by J. Traub), Academic Press, New York, (1976).
14. R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge Univ. Press, Cambridge, UK, (1995).
15. R. Karp, An introduction to randomized algorithms, *Discrete Applied Mathematics* **34**, 165–201 (1991).
16. P. Rangan, Lecture notes from seminar, IBM Tokyo Research Laboratory, June 29, 2000, Japan.
17. T. Kohonen, An introduction to neural computing, *Neural Networks* **1** (1), 3–16.
18. S. Grossberg, Nonlinear neural networks: Principles, mechanisms and architectures, *Neural Networks* **1** (1), 3–16 (1988).
19. T. Masters, *Advanced Algorithms for Neural Networks: A C++ Sourcebook*, John Wiley and Sons, New York, (1993).
20. G. Dupret, Spatiotemporal analysis and forecasting: Identical units artificial neural network, Ph.D. Thesis, University of Tsukuba, Dept. of Policy and Planning Sciences. Preliminary draft available fall 1999, final draft scheduled to be issued March 2000.
21. C. Eckart and G. Young, A principal axis transformation for non-Hermitian matrices, *Bulletin of the American Mathematical Society* **45**, 118–121 (1939).
22. S. Deerwester *et al.*, Indexing by latent semantic analysis, *Journal of the American Society for Information Science* **41** (6), 391–407 (1990).
23. S. Dumais, Improving the retrieval of information from external sources, *Behavior Research Methods, Instruments and Computers* **23** (2), 229–236 (1991).
24. W. Frakes and R. Baeza-Yates, Editors, *Information Retreival*, Prentice-Hall, Englewood Cliffs, NJ, (1992).
25. A. Jain and R. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Englewood Cliffs, NJ, (1988).
26. E. Rasmussen, Clustering algorithms, In *Information Retreival*, (Edited by W. Frakes and R. Baeza-Yates), pp. 419–442, Prentice-Hall, Englewood Cliffs, NJ, (1992).
27. W. Press *et al.*, *Numerical Recipes in C*, Second Edition, Cambridge University Press, New York, (1982).
28. J. Cullum and R. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Volume 1: Theory, Volume 2: Programs*, Birkhäuser, Boston, MA, (1985).
29. T. Coleman and C. Van Loan, *Handbook for Matrix Computations*, SIAM, Philadelphia, PA, (1988).
30. J. Dongarra *et al.*, *LINPACK Users' Guide*, SIAM, Philadelphia, PA, (1979).
31. *LANSO*, Dept. of Computer Science and the Industrial Liason Office, University of California, Berkeley.
32. E. Anderson *et al.*, *LAPACK Users' Guide*, Second Edition, SIAM, Phildelphia, PA, (1995).
33. L. Blackford *et al.*, *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA, (1997).
34. M. Berry *et al.*, *SVDPACKC: Ver. 1.0, Users' Guide*, Technical Report, Department of Computer Science, Univ. of Tennessee, No. CS-93-194, (October 1993).
35. B. Parlett and I. Dhillon, Fernando's solution to Wilkinson's problem: An application of double factorization, *Linear Algebra and its Applications* **267**, 247–279 (1997).
36. B. Parlett and D. Scott, The Lanczos algorithm with selective orthogonalization, *Mathematics of Computation* **33**, 217-238 (1979).