May 17, 2000
RT0365
Computer Science 26 pages

Research Report

Efficient Estimation of Singular Values for Searching Very Large and Dynamic Web Databases

Mei Kobayashi, Georges Dupret, Oliver King
and Hikaru Samukawa

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Efficient Estimation of Singular Values for Searching Very Large and Dynamic Web Databases

Mei Kobayashi [1], Georges Dupret [2], Oliver King [3], and Hikaru Samukawa [4]

IBM Japan, Ltd., Tokyo Research Laboratory,

1623-14 Shimotsuruma, Yamato-shi, Kanagawa-ken 242-8502 Japan

Abstract

The singular value decomposition (SVD) has enjoyed a long and rich history. Recently, it is being used in data mining applications and by search engines to rank documents in very large databases, including the Web. The dimensions of matrices which appear in these applications are becoming so large that classical algorithms for computing the SVD cannot always 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 most users. 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. We demonstrate the power and accuracy of our method through examples.

keywords: eigenvalues,information retrieval, Lanczos algorithm, Multiple-Layer Feedforward Network, neural network, singular value decomposition, SVD

[1]e-mail: MEI@jp.ibm.com

[2]Graduate Student Intern from the University of Tsukuba, e-mail: gedupret@hotmail.com

[3]Graduate Student Intern from the U.C. Berkeley, e-mail: king@math.berkeley.edu

[4]e-mail: samukawa@trl.ibm.co.jp

1. Introduction

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

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

of unitary matrices U and V and a diagonal matrix Σ, has a long and rich history, as chronicled in a paper by Stewart [19]. A formal statement of the existence theorem for the SVD can be found in standard texts on linear algebra, such as [7],[14]. Although it was introduced in the 1870's 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 applied in a wide variety of disciplines, most notably for least squares fitting of data [11]. Recently it is being used in data mining applications and by some automated search engines, e.g., *Alta Vista* [5], to rank documents in very large databases, including the Web [1],[2],[3],[9],[10],[12]. The dimensions of matrices in mathematical models for these applications are becoming so large that classical algorithms for computing the SVD cannot always 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 discuss 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. Next, we explain how knowledge of the singular values can be valuable for tuning the performance of two types of information retrieval systems which are based on vector space models. Finally, we review some standard algorithms for the computation of the SVD. 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.

[5] *Alta Vista* homepage: http://www.altavista.com

1.1 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 l_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 (Eckhart and Young [5]): *Let the singular value decomposition of A be given by equation (1) with $r = \text{rank}(A) \leq p = \min(m, n)$, and define*

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

(see Figure 1). Here Σ_k is a diagonal matrix with k non-zero, monotonically decreasing diagonal elements $\sigma_1, \sigma_2, \ldots, \sigma_k$, 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

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

The proof of the theorem is available in many texts, including [7],[14].

The *condition number* of a matrix A, which we denote by $\kappa(A)$, is one of the simplest and 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 non-zero singular value divided by the smallest non-zero singular value. The largest singular value for very large matrices can be estimated by our technique or the power method (see Section 1.2). Computation of the smallest singular value of very large matrices is very difficult. Although our technique cannot always be applied to compute the smallest non-zero 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 huge, then a an accurate estimate of σ_N may be costly to compute, however, it is not as expensive to compute a reliable upper bound for σ_N (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 it implies that computations with the matrix A may be very sensitive to roundoff errors. If an estimated lower bound for $\kappa(A)$ is small, we have not gained any new information.

1.2 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 [3], and more recently, the algorithm has been extended to retrieval, ranking and visualization systems for the Web [1],[2],[10]. These systems are based on a vector space model of document-query space [18]. The relationship between documents and their attributes (e.g., keywords, time stamp information, frequency of updates and access) is represented by an m-by-n matrix A, with ij-th entry a_{ij}, i.e., $A = [a_{ij}]$. The entries a_{ij} consist of information on whether attribute 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 relevance of a keyword in the document; and the frequency of the keyword term in the document. Ideally, only those which can help in distinguishing documents are incorporated in the attribute space. A is usually a very large, sparse matrix, because the number of attributes in any single document is usually a very small fraction of union of the attributes in all of the documents. In the simplest retrieval and ranking systems, each query is also modeled by a vector in the same manner as the documents. The ranking of documents with respect to a query is determined by its "*distance*" to the query vector. A frequently used yardstick is the angle defined by a query and document vector. It is impractical for very large databases.

One well-known algorithm known as *latent semantic indexing (LSI)* uses the SVD to reduce the dimension of the document-attribute matrix to expedite the retrieval and ranking process by constructing a modified matrix A_k, from the k largest singular values and their corresponding vectors, as shown in Figure 1.

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

Here we follow the notation use 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 as follows.

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

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^T 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, e.g., [6],[8],[17], LSI 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 may lead to many small, disjoint clusters, some of which should be clustered together, while absence of accounting for polysemy may 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:

1. decide apriori how many singular values can be computed and set the dimension to be equal to this number, or

2. 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 making a final decision on the dimension of the subspace, we can estimate whether breakdown of a program due to memory overflow or paging will occur. It might be possible to make some rough estimates of computational costs and time delays associated with paging.

The singular value decomposition appears in a modified form in another promising ranking and retrieval technique for very large databases described in [12]. The method uses principal component analysis to reduce the dimensionality of the document-attribute space. The retrieval and ranking problem is projected into the subspace spanned by the eigenvectors associated with the largest 5%–20% eigenvalues of the covariance matrix of the document vectors a database. (Note that the covariance matrix is a symmetric, positive semi-definite matrix so its eigenvalues are its singular values.) This algorithm can be implemented to run efficiently without explicit computation of the covariance matrix, with a simple means to update the matrix to take into account changes in the entries of the database. Before the eigenvectors are computed (using, e.g., standard linear algebra algorithms for symmetric matrices, or neural nets), knowledge of the eigenvalues is helpful for deciding on the dimension of the subspace. The dimension is determined based on the distances between eigenvalues, in a manner analogous to singular value analysis in LSI.

1.3 Standard Approaches to Computing the SVD

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

Method 1: 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. 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 non-diagonal 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 Chapter 5 of [7]). Note that when the rotator is a 2-by-2 matrix, it reduces to the standard rotation matrix in a 2-dimensional plane. When Givens rotations are used to zero an entry on the superdiagonal, it normally creates a new non-zero entry on the subdiagonal. For instance, zeroing the (1,2) entry causes re-computation 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 bi-diagonal matrix is called "*chasing*" or "*zero chasing*" (see Figure 2) [11]. A sequence of Givens rotations or zero chasing must be performed to zero each $(i, i+1)^{th}$ element of the bidiagonal matrix, beginning with $i = 1$, then $i = 2, 3, \ldots$. Use of Householder transformations followed by Givens for computing the SVD will normally: destroy special features of the matrix A, including sparsity; require significant memory; and be computationally slow.

Method 2: the Power Method and Subspace Iteration If a matrix A is very sparse and only a few of the singular values and singular vectors of A are needed for an application, a good method for computing the SVD 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. One drawback is 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

$$A^T \cdot A \qquad \text{and} \qquad A \cdot A^T \;,$$

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 our 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. It is very difficult to write fail-safe, fast code which processes multiple and very close roots.

In the power method we begin with an arbitrary vector v of unit length [7], and assume the vector has a non-trivial component in the direction of the eigenvector v_1 associated with the largest eigenvalue λ_1. (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.) Then we compute the limit of the *Rayleigh quotient* of the matrix B, defined as

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

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 $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$. After several iterations during the computation of λ_n, 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.

In principle, as many eigenvalues as desired can be computed this way 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 [7]. Use of the modified, rather than clasical, Gram-Schmidt is recommended since numerical roundoff often leads to poor results when the classical method is used.

Method 3: Lanczos Algorithms for Symmetric, Positive Semi-Definite 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 the symmetric Lanczos applied to $B = A^T A$. Note that B is computed implicitly to minimize the use of memory Since B is symmetric, positive definite, Lanczos tridiagonalization will convert it to a symmetric matrix without many of the difficulties associated with the Lanczos 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 [7],[14]. Unfortunately, the associated eigenvectors must be computed separately.

2. Sampling Algorithm for Determining Singular Values

In this section we present three versions of an algorithm to estimate the eigenvalues of a symmetric, positive, semi-definite matrix and then discuss possible variations. 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-by-N matrix whose singular values σ_i cannot be computed due to the enormity of its size. 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 smaller matrix $A^{(1)}$ using any standard method, such as the Lanczos Method followed by Sturm sequencing [14]. Repeat this process 50 times (or any number of times which is sufficiently large to allow statistical analysis), i.e., construct matrices $A^{(i)} \quad ; i = 1, 2, 3, \ldots, 50$ 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 [13].

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 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-by-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 with 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 will 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-by-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 unevely 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 or generalized linear regression.

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 semi-definite, 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 program in [13]. Details of the MLFN algorithm, variations and enhancements are given in [4]; it is based on the Conjugate Gradient Method, Direct Line Minimization, and Stochastic Optimization (Simulated Annealing).

3.1 Text Mining Matrix

In the first set of experiments, we considered a $36,403$-by-$10,000$ 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. 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 5 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 for 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; and

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 3 for the first set of experiments and Figure 4 for the second set. The corresponding numerical data given in Tables 1a-e and Table 2, respectively.

In experiment 1, 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. Our experimental results 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.

3.2 Randomly Generated Symmetric, Positive Semi-Definite Matrices

In a second set of experiments, we generated random symmetric, positive, semi-definite 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. In our estimation experiments for a matrix, we took random samples of the rows of the matrix and computed the largest 5 singular values. We repeated the process 100 times and computed the mean and the standard deviation. Typical results from our experiments using neural networks for curve fitting and extrapolation are given in Tables 3a-d. 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 5 and 6; data for estimating the first five singular values are shown in Figure 5 and a close-up of the curves for the second to fifth singular values is shown in Figure 6. 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.

3.3 Application of Experimental Results to Information Retrieval

As we noted earlier in section 1.1.1, 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. We can find an upper bound for the smallest nonzero singular value σ_{min} using data from our method. We use the fact that the estimation curve for any singular value always 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 $M \geq \sigma_{min}$.

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 a matrix. It has been observed that singular values of matrices tend to be unevenly distributed; they usually cluster around several values. Furthermore, a good approximation of a singular vectors can usually be computed by orthogonalizing only with respect to those singular vectors whose corresponding singular values are in the same cluster [15],[16]. Knowledge of the clustering patterns of the singular values will 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 to compute based on the availability of memory and computing 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.,

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

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 3) is better or if a hybrid of Methods 1 and 3 (i.e., Method 2) 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 for LSI; 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 multi-dimensional interpolation, then sampling columns for multi-dimensional interpolation.

For retrieval and ranking based on principal component analysis, computations of the eigenvalues of covariance matrices associated with very large databases can be carried out by several algorithms. Requirements on the databases which can be processed by the techniques (e.g., the maximum number of documents and keywords, properties of the eigenvalues of the associated covariance matrix) needs to be investigated.

Acknowledgements: The authors would like to thank Hiroshi Nomiyama, Koichi Takeda and Loïc Malassis for helpful conversations and Tetsuya Nasukawa for allowing us to use his text mining data.

References

[1] R. Baeza-Yates, B. Ribeiro-Neto (eds.), *Modern Information Retrieval*, ACM Press and Addison-Wesley, NY, 1999.

[2] M. Berry, S. Dumais, G. O'Brien, Using linear algebra for intelligent information retrieval, *SIAM Review*, **37**, 4, pp. 573–595, Dec. 1995.

[3] S. Deerwester et al., Indexing by latent semantic analysis, *Journal of the American Society for Information Science*, **41**, 6, pp. 391–407, 1990.

[4] G. Dupret, *Spatiotemporal Analysis and Forecasting: Identical Units Artificial Neural Network*, Ph.D. Thesis, Univ. of Tsukuba, Dept. of Policy and Planning Sciences, March 2000.

[5] C. Eckart and G. Young, A principal axis transformation for non-Hermitian matrices, *Bulletin of the American Mathematical Society*, **45**, pp. 118–121, 1939.

[6] W. Frakes and R. Baeza-Yates (eds.), *Information Retreival*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[7] G. Golub and C. Van Loan, *Matrix Computations*, third ed., John Hopkins University Press, Baltimore, MD, 1996.

[8] A. Jain and R. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[9] M. Kobayashi et al., "Multi-perspective retrieval, ranking and visualization of Web data, *Proceedings of the International Symposium on Digital Libraries (ISDL) '99*, Tsukuba, Japan, 1999, pp. 159–162.

[10] M. Kobayashi and K. Takeda, Information retrieval on the Web, *IBM Research Report*, RT0347, April 2000.

[11] C. Lawson and R. Hanson, *Solving Least Squares Problems*, SIAM, Philadelphia, 1995.

[12] L. Malassis and M. Kobayashi, *IBM Research Report*, in preparation, 2000.

[13] T. Masters, *Advanced Algorithms for Neural Networks: a C++ Sourcebook*, John Wiley and Sons, NY, 1993.

[14] B. Parlett, *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, PA, 1998.

[15] B. Parlett and I. Dhillon, Fernando's solution to Wilkinson's problem: an application of double factorization, *Linear Algebra and its Applications*, **267**, pp. 247–279, 1997.

[16] B. Parlett and D. Scott, The Lanczos algorithm with selective orthogonalization, *Mathematics of Computation*, **33**, pp. 217–238, 1979.

[17] E. Rasmussen, Clustering algorithms pp. 419–442 in [6]. [18] G. Salton, *The SMART Retrieval System - Experiments in Automatic Document Processing*,

Prentice-Hall, Englewood Cliffs, NJ, 1971.

[19] G. Stewart, On the early history of the singular value decomposition, *SIAM Review*, **35**, pp. 551–566, Dec. 1993, anonymous ftp: thales.cs.umd.edu, directory pub/reports

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

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

Table 3a: singular values of a randomly generated matrix (433 rows)

% 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)

% 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)

% 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)

% 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

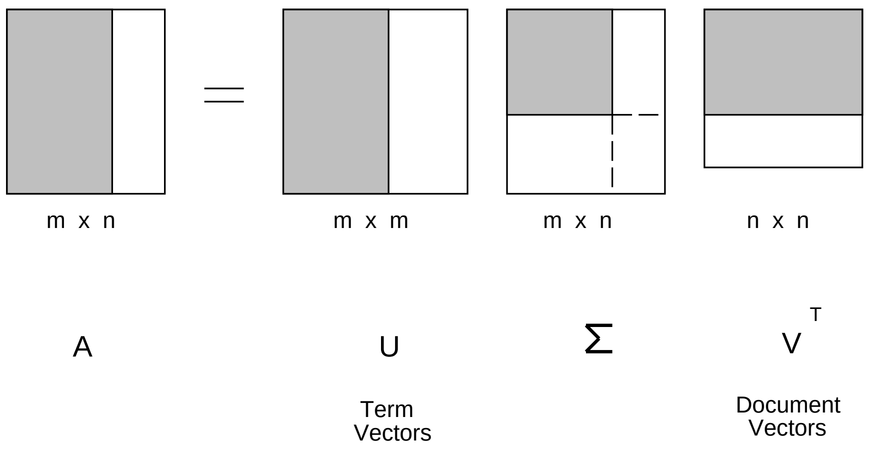

Figure 1: Construction of A_k, the closest rank-k approximation to A, through modification of the singular value decomposition of A. The colored portions of A, U, Σ and V^T remain intact, and entries of the white portions of the matrices are set to zero to construct A_k, U_k, Σ_k and V_k^T.

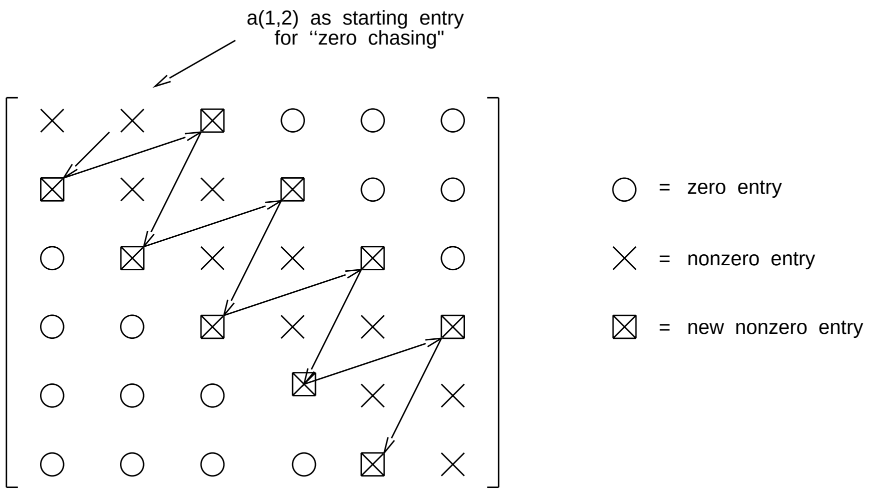

Figure 2: Example of "*zero chasing*" using Givens rotations on a 6-by-6 bidiagonal matrix.

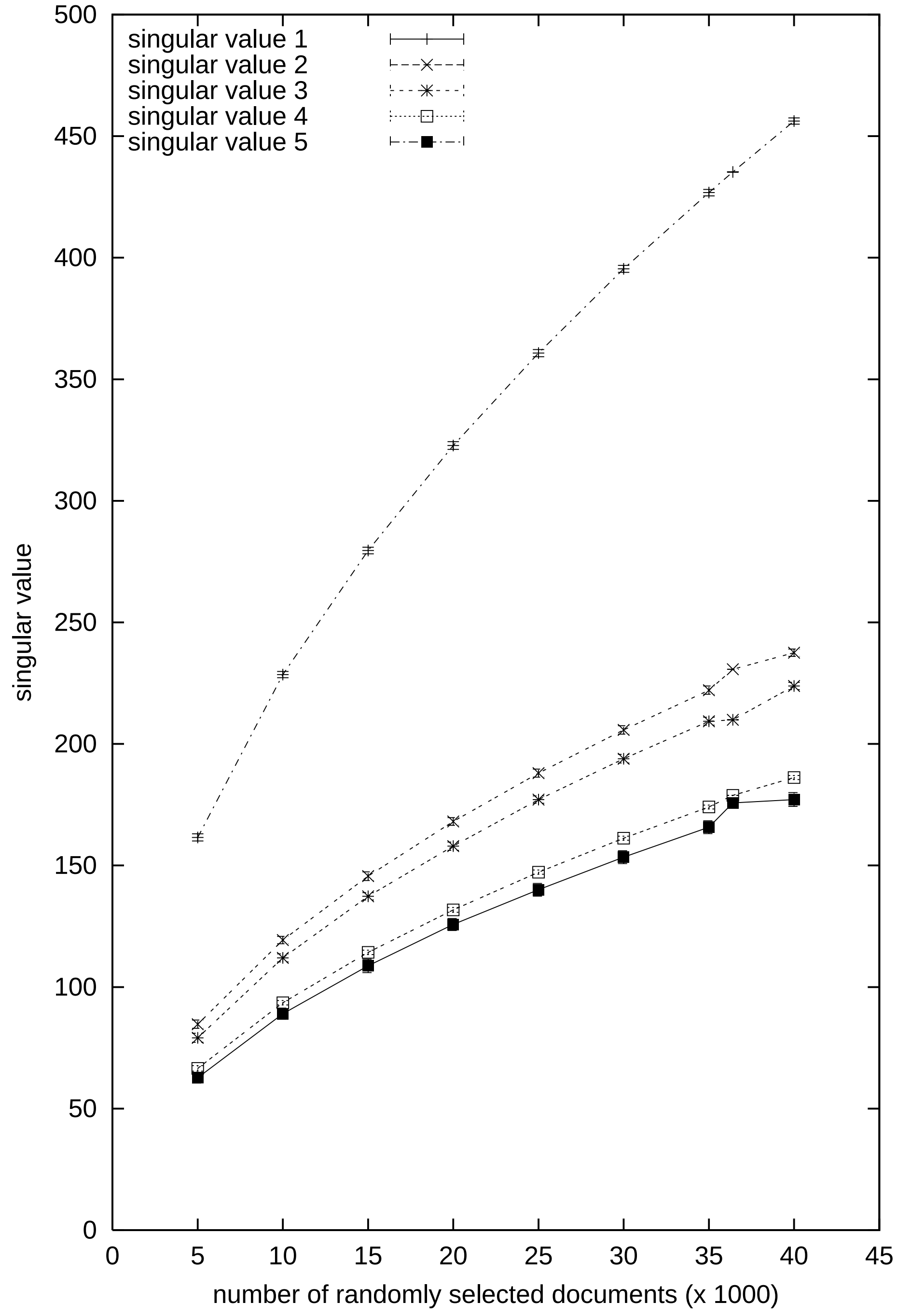

Figure 3: Curves for approximating the largest five singular values of a matrix allowing for the possibility of duplicate sampling.

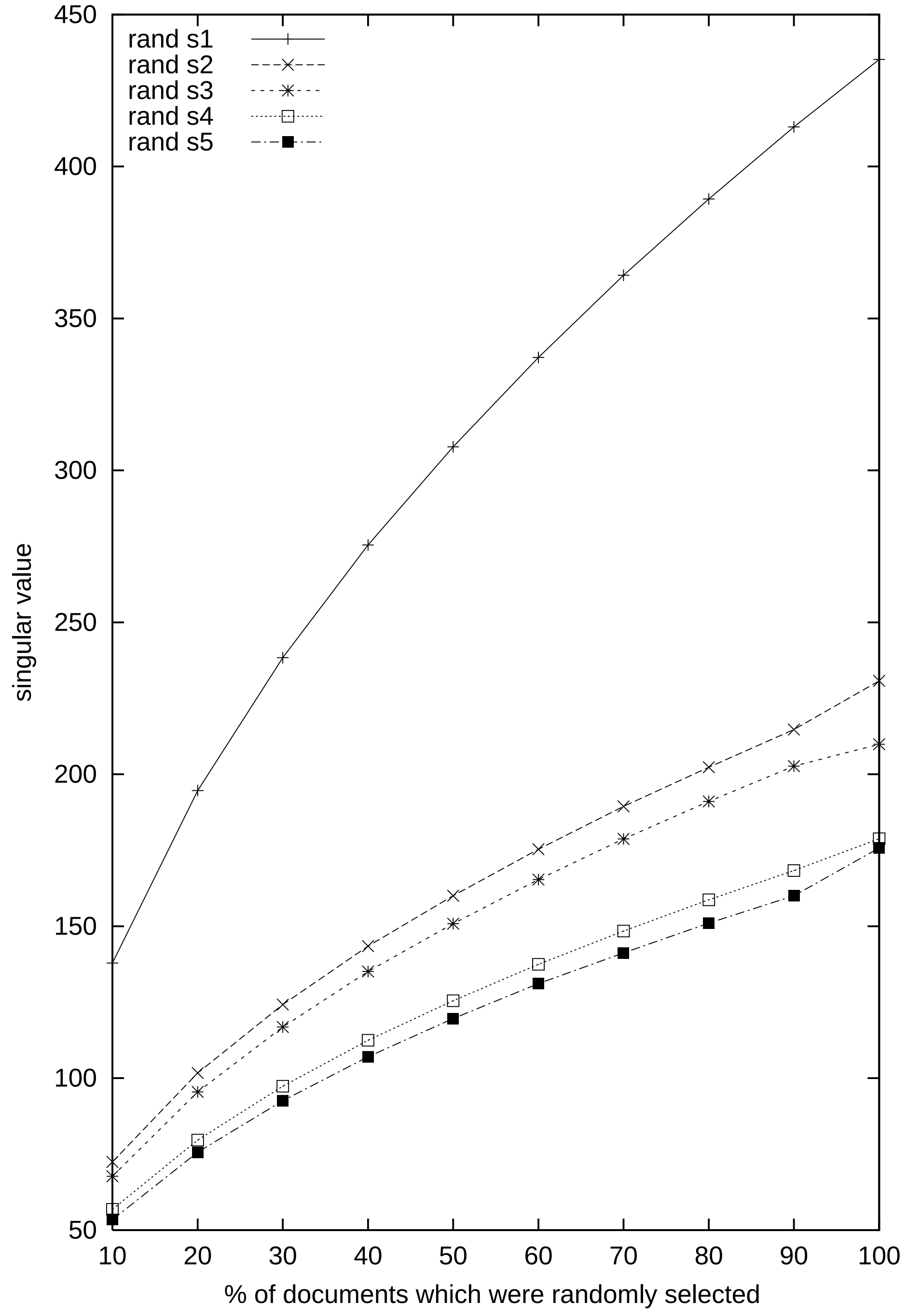

Figure 4: Curves for approximating the largest five singular values of a matrix with no duplicate sampling.

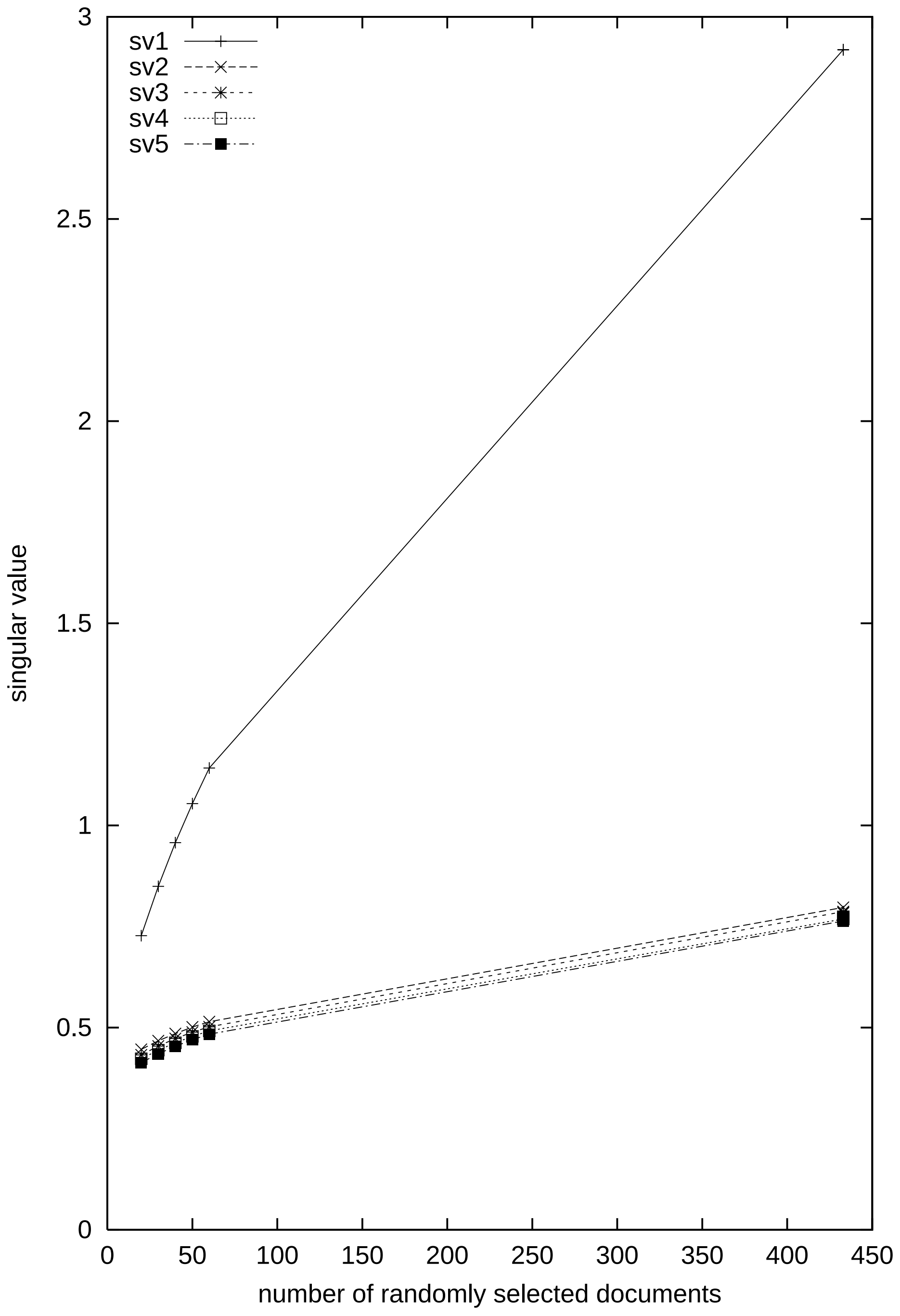

Figure 5: Curves for approximating the largest five singular values of a randomly generated matrix (433 rows); Corresponding numerical data given in Table 3a.

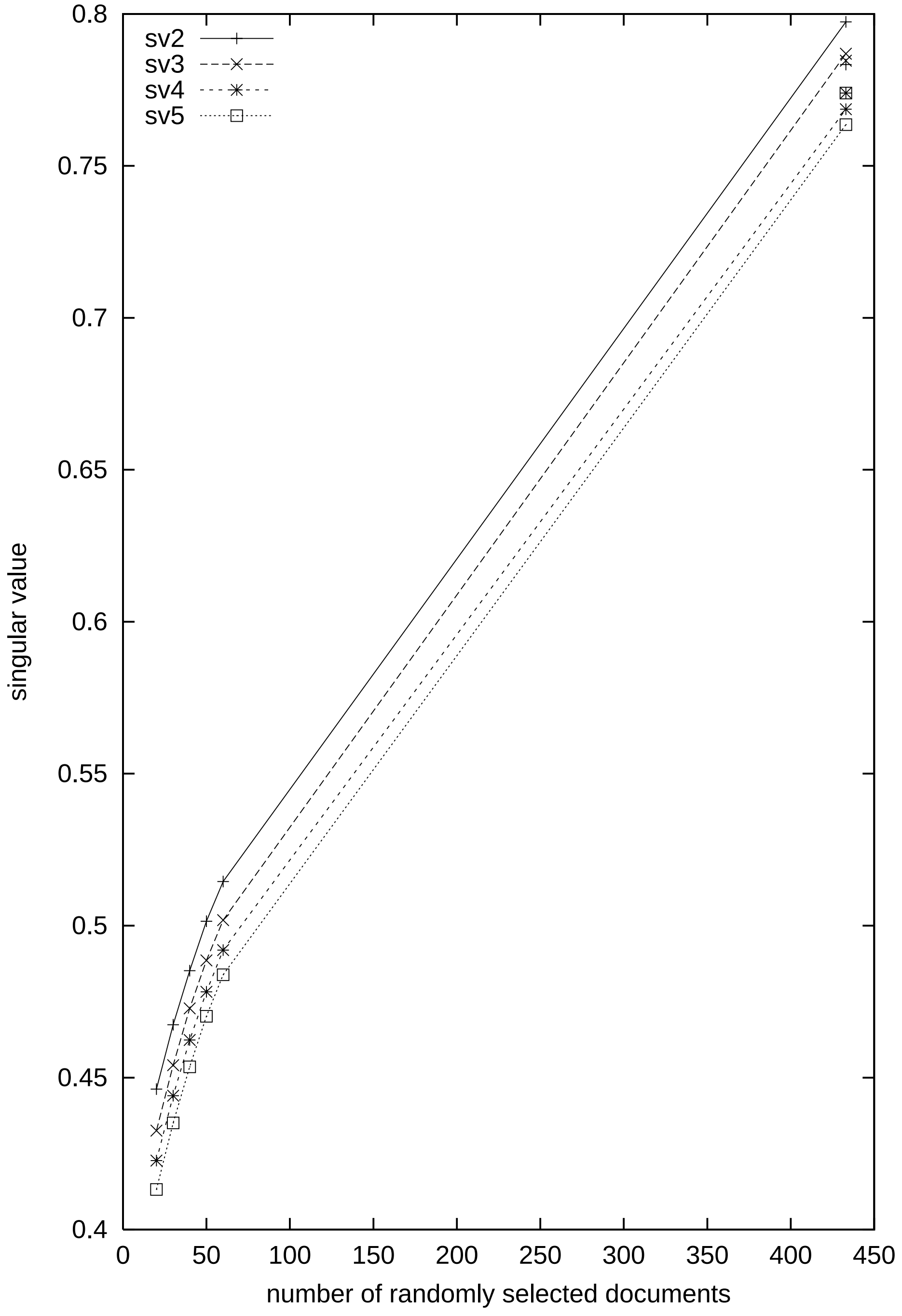

Figure 6: Curves for approximating the second to fifth largest singular values of a randomly generated matrix (433 rows); Corresponding numerical data given in Table 3a.