An Experimental Evaluation of a Monte-Carlo Algorithm for Singular Value Decomposition *

Petros Drineas[1], Eleni Drinea[2], and Patrick S. Huggins[1]

[1] Yale University, Computer Science Department,
New Haven, CT 06520, USA
{drineas,huggins}@cs.yale.edu
[2] Harvard University, Division of Engineering and Applied Sciences,
Cambridge, MA 02138, USA
edrinea@deas.harvard.edu

Abstract. We demonstrate that an algorithm proposed by Drineas *et. al.* in [7] to approximate the singular vectors/values of a matrix A, is not only of theoretical interest but also a fast, viable alternative to traditional algorithms. The algorithm samples a small number of rows (or columns) of the matrix, scales them appropriately to form a small matrix S and computes the singular value decomposition (SVD) of S, which is a good approximation to the SVD of the original matrix. We experimentally evaluate the accuracy and speed of this randomized algorithm using image matrices and three different sampling schemes. Our results show that our approximations of the singular vectors of A span almost the same space as the corresponding exact singular vectors of A.

1 Introduction

In many applications we are given an $m \times n$ matrix A and we want to compute a few of its left (or right) singular vectors. Such applications include data clustering [1], information retrieval [13], property testing of graphs, image processing, etc. Singular vectors are usually computed via the Singular Value Decomposition (SVD) of A (see section 2).

There are many algorithms that either exactly compute the SVD of a matrix in $O(mn^2 + m^2n)$ time (an excellent reference is [11]) or approximate it faster (e.g. Lanczos methods [17]). In [10] and [7] randomized SVD algorithms were proposed: instead of computing the SVD of the entire matrix, pick a subset of its rows or columns (or both), compute the SVD of this smaller matrix and show that it is a good approximation to the SVD of the initial matrix; theoretical error bounds for these Monte-Carlo algorithms were presented. In this paper we experimentally evaluate the performance of the algorithm proposed in [7] (which is better suited for practical applications) by demonstrating its accuracy and speedup over traditional SVD algorithms. Our test set consists of image

* A preliminary version of this work appeared in the 2001 Panhellenic Conference on Informatics [5].

matrices; we explain our choice below. This randomized SVD algorithm returns approximations to the top k right (or left) singular vectors of the image matrix. Our goal is to compare these approximations to the exact singular vectors; we measure the accuracy of our approximations by computing rank k approximations to the matrices of our test set, using both sets of vectors, and comparing the results. We will explore three different schemes for sampling rows (columns) of the input matrix: uniform sampling with replacement, uniform sampling without replacement and weighted sampling; in this latter case, "heavier" rows/columns are included in the sample with higher probability. The most encouraging result is that the experimental performance of the algorithm is *much better* than its theoretical error bounds.

A general family of applications for this algorithm is Principal Component Analysis applications (e.g. eigenfaces [14] or Latent Semantic Indexing in information retrieval), where a database (of documents, images etc.) that exists in a high dimensional space is projected to a lower dimensional space using SVD. Then, answering a query (that is searching for an instance in the database that is close to the query) amounts to projecting the query to the same low-dimensional space and then finding the nearest neighbor. The projections need not be exact for two reasons: the values of the elements of the database are usually determined using inexact methods *and* the exact projection to the lower dimensional space is not necessary, since we are only interested in a nearest neighbor search.

The main reason behind picking image matrices in order to demonstrate the accuracy and speed of our algorithm is that beyond evaluating numerical results (i.e. the relative error of the approximation), we can also estimate the accuracy of the approximation visually. Also, by experimenting with image matrices, we can demonstrate that our methods work well even for relatively small and very dense matrices (up to 1000 rows and columns, density in general close to 1). We will see that for these matrices, uniform sampling performs equally well to our more sophisticated sampling methods. The performance of our algorithm has also been examined in [13] using a matrix from information retrieval datasets, but the matrix there was very large (more than 10^5 rows and columns) and less than 10% dense. Finally, singular vectors of image matrices are quite useful in image processing (e.g. eigenfaces, image compression, image restoration, etc. for details see [19, 15, 2, 20, 4, 3, 12, 16]). With images getting larger and larger, certain applications might not be able to afford the computational time needed for computing the SVD. Next we describe just such an application in medical imaging.

In dynamic Magnetic Resonance Imaging (MRI) a series of time ordered images is obtained by continually updating image data as changes occur (e.g. monitoring of surgical procedures). In [23, 22] Zientara *et. al.* investigated the use of SVD for creating and encoding these images. Their technique approximates the top few left or right singular vectors of an initial image and uses them to define "excitation profiles". These profiles are in turn used to create SVD encoded data for the next image in the series. The authors argue that this process is much faster than fully generating the image using state of the art

MRI equipment. Recreating the image (that is "decoding" SVD) amounts to a multiplication with the computed singular vectors. One major constraint is the time required by the SVD computation, which can now be reduced using our algorithm.

The paper is organized as follows: in section 2 we state some basic Linear Algebra definitions and theorems related to SVD. In section 3 we present the algorithm in a different way than it was presented in [7], more suitable for implementation purposes. In section 4.1 we describe our experimental dataset and in section 4.2 we demonstrate that although the theoretical error bounds are very weak for relatively small matrices (such as image matrices), in practice the algorithm is very efficient and accurate. Finally, in section 4.3 we experimentally compare the speed of our algorithm vs. Lanczos/Arnoldi techniques.

2 Background on SVD

Any $m \times n$ matrix A can be expressed as

$$A = \sum_{t=1}^{r} \sigma_t u^{(t)} {v^{(t)}}^T$$

where r is the rank of A, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$ are its singular values and $u^{(t)} \in \mathcal{R}^m, v^{(t)} \in \mathcal{R}^n, t = 1, \ldots, r$ are its left and right singular vectors respectively. The $u^{(t)}$'s and the $v^{(t)}$'s are orthonormal sets of vectors; namely, ${u^{(i)}}^T \cdot u^{(j)}$ is one if $i = j$ and zero otherwise. We also remind the reader that

$$\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2} \quad \text{and} \quad |A|_2 = \max_{x \in \mathcal{R}^n : |x|_2 = 1} |Ax|_2 = \sigma_1$$

In matrix notation, SVD is defined as $A = U \Sigma V^T$ where U and V are orthonormal matrices, containing the left and right singular vectors of A. $\Sigma = \mathbf{diag}(\sigma_1, \ldots, \sigma_r)$ contains the singular values of A; we remind the reader that $U^T U = I$ and $V^T V = I$.

If we define $A_k = \sum_{t=1}^{k} \sigma_t u^{(t)} {v^{(t)}}^T$, then A_k is the best rank k approximation to A with respect to the 2-norm and the Frobenius norm. Thus, for any matrix D of rank at most k, $|A - A_k|_2 \leq |A - D|_2$ and $\|A - A_k\|_F \leq \|A - D\|_F$. A matrix A has a "good" rank k approximation if $A - A_k$ is small with respect to the 2-norm and the Frobenius norm. It is well known that

$$\|A - A_k\|_F = \sqrt{\sum_{t=k+1}^{r} \sigma_t^2(A)} \quad \text{and} \quad |A - A_k|_2 = \sigma_{k+1}(A)$$

Finally, from basic Linear Algebra, $A_k = U_k \Sigma_k V_k^T = A V_k V_k^T = U_k U_k^T A$, where U_k and V_k are sub-matrices of U, V containing only the top k left or right singular vectors respectively; for a detailed treatment of Singular Value

Decomposition see [11]. In the following, $A_{(i)}$ denotes the i-th row of A as a row vector and $A^{(i)}$ denotes the i-th column of A as a column vector. The length of a column (or row) will be denoted by $|A^{(i)}|$ (or $|A_{(i)}|$) and is equal to the square root of the sum of the squares of its elements.

3 The randomized SVD algorithm

In this section we discuss the SVD algorithm of [7] and, more specifically, an efficient implementation of the algorithm. We also present theoretical error bounds for three sampling schemes (weighted sampling, uniform sampling with replacement, uniform sampling without replacement) and we comment on the quality of the bounds in practice.

3.1 The algorithm

Given an $m \times n$ matrix A we seek to approximate its top k right singular values/vectors. Intuitively, our algorithm picks s rows of A, forms an $s \times n$ matrix S and computes its right singular vectors. Assume that we are given a set of probabilities $p_1, \ldots, p_m$ such that $\sum_{i=1}^{m} p_i = 1$.

FastSVD Algorithm
Input: $m \times n$ matrix A, integers $s \leq m$, $k \leq s$.
Output: $n \times k$ matrix H, $\lambda_1, \ldots, \lambda_k \in \mathcal{R}^+$.

1. **for** $t = 1$ **to** s
   - Pick an integer from $\{1 \ldots m\}$, where $\mathbf{Pr}(\text{pick } i) = p_i$.
   - Include $A_{(i)}/\sqrt{sp_i}$ as a row of S.
2. Compute $S \cdot S^T$ and its singular value decomposition. Say
$$SS^T = \sum_{t=1}^{s} \lambda_t^2 w^{(t)} w^{(t)^T}$$
3. Compute $h^{(t)} = S^T w^{(t)}/|S^T w^{(t)}|, t = 1 \ldots k$. Return H, a matrix whose columns are the $h^{(t)}$ and $\lambda_1, \ldots, \lambda_k$ (our approximations to the top k singular values of A).

In step 2, λ_t^2 are the singular values of SS^T and $w^{(t)}, t = 1 \ldots s$ its left (and right[3]) singular vectors. In step 3, the $h^{(t)}$'s are the right singular vectors of S (and our approximations to the right singular vectors of A). It should be obvious that the SVD of S is $S = \sum_{t=1}^{s} \lambda_t w^{(t)} h^{(t)^T}$. We emphasize here that if we were

[3] We remind the reader that for symmetric matrices the left and right singular vectors are equal.

computing the right singular vectors of S directly after step 1, the running time would be $O(n^2)$, while now it is $O(n)$ (see section 3.3). The algorithm is simple and intuitive; the only part that requires further attention is the sampling process (see section 3.3).

3.2 Theoretical analysis

How does one evaluate how close H is to V_k (the top k right singular vectors of A)? We are usually interested in the space spanned by V_k; this space is *invariant* for A: $\mathbf{span}(AV_k) \subset \mathbf{span}(V_k)$. A consequence of this property is that $A_k = AV_kV_k^T$ is the "best" rank k approximation to A. Thus, to evaluate the quality of H as an approximation to V_k, we will show that $P = AHH^T$ (a rank k matrix) is *almost* as "close" to A as A_k is. The "closeness" will be measured using the standard unitarily invariant norms: the Frobenius norm and the 2-norm (see section 2). More specifically, our analysis guarantees that $\|A - P\|_{F,2}$ is at most $\|A - A_k\|_{F,2}$ plus some additional error, which is inversely proportional to the number of rows that we included in our sample. As the "quality" of H increases, H and V_k span *almost* the same space and P is *almost* the optimal rank k approximation to A.

In the following theorem $\mathbf{E}(X)$ denotes the expectation of X. For detailed proofs of the theorem see [7, 6].

Theorem 1. *If $P = AHH^T$ is a rank (at most) k approximation to A, constructed using the algorithm of section 3, then, for any $s \leq m$,*

1. *If $p_i = |A^{(i)}|^2/\|A\|_F^2$ and sampling is done with replacement,*

$$\mathbf{E}\left(\|A - P\|_F^2\right) \leq \|A - A_k\|_F^2 + 2\sqrt{\frac{k}{s}}\|A\|_F^2$$

$$\mathbf{E}\left(|A - P|_2^2\right) \leq |A - A_k|_2^2 + \frac{2}{\sqrt{s}}\|A\|_F^2$$

   This sampling minimizes the variance for the error of the approximation.

2. *If $p_i = 1/m$ and sampling is done with replacement,*

$$\mathbf{E}\left(\|A - P\|_F^2\right) \leq \|A - A_k\|_F^2 + 2\sqrt{k\frac{m}{s}\sum_{t=1}^{m}|A^{(t)}|^4}$$

$$\mathbf{E}\left(|A - P|_2^2\right) \leq |A - A_k|_2^2 + 2\sqrt{\frac{m}{s}\sum_{t=1}^{m}|A^{(t)}|^4}$$

3. *If $p_i = 1/m$ and sampling is done without replacement,*

$$\mathbf{E}\left(\|A - P\|_F^2\right) \leq \|A - A_k\|_F^2 + 2\sqrt{\left(\frac{km}{m-1}\right)\left(\frac{m}{s} - 1\right)\sum_{t=1}^{m}|A^{(t)}|^4}$$

$$\mathbf{E}\left(|A - P|_2^2\right) \leq |A - A_k|_2^2 + 2\sqrt{\left(\frac{m}{m-1}\right)\left(\frac{m}{s} - 1\right)\sum_{t=1}^{m}|A^{(t)}|^4}$$

The first bound is clearly a relative error bound; the relative error of our approximation is at most the relative error of the optimal rank k approximation plus an additional error, inversely proportional to the number of rows that we include in our sample. The other two bounds imply weaker relative error bounds, depending on how close $\|A\|_F^2$ is to $\sqrt{m\sum_{t=1}^{m}|A^{(t)}|^4}$. Intuitively, these two quantities are close if the Frobenius norm of A is distributed evenly among its rows. Indeed, if $|A_{(i)}|^2 \geq (\gamma/m)\|A\|_F^2$ for some $\gamma \leq 1$, it is easy to show that

$$\sqrt{m\sum_{t=1}^{m}|A^{(t)}|^4} \geq \gamma\|A\|_F^2$$

Comparing the above bounds, we see that the first one is, generally, the tightest. The third one is tighter than the second, since a row can not be included in the sample twice[4] (thus our sample has more useful information). Finally, we note that weighted sampling without replacement is very difficult to analyze; indeed, it is not even clear that we can perform such sampling and compute the scaling factors efficiently.

Theoretically, the above algorithm necessitates the sampling of a significant number of rows of A in order to achieve reasonable error guarantees. As an example, if we seek to approximate the top 50 right singular vectors, in order to achieve 2% expected relative error with respect to the Frobenius norm, we need to pick at least $5 \cdot 10^5$ rows! The situation is even worse if we seek a probabilistic statement[5] instead of the *expected* error. Thus, an experimental evaluation of the algorithm is imperative; indeed, in practice (see section 4.2), the algorithm achieves small relative errors by sampling only a constant fraction of the rows of A.

3.3 Implementation details and running time

An important property of our algorithm is that it can be easily implemented. Its heart is an SVD computation of an $s \times s$ matrix (SS^T). Any fast algorithm computing the top k right singular vectors of such a matrix could be used to speed up our algorithm (e.g. Lanczos methods). One should be cautious though; since s is usually of $O(k)$, we might end up seeking approximations to almost all singular vectors of SS^T. It is well known that in this scenario full SVD of SS^T is much more efficient than approximation techniques. Indeed, in our experiments, we observed that full SVD of SS^T was faster than Lanczos methods.

The other interesting part of the algorithm is the sampling process. Uniform sampling ($p_1 = \ldots = p_m = 1/m$), with or without replacement, is trivial to implement and can be done in constant time. Sampling with respect to row (or column) lengths is more interesting; we describe a way of implementing it when $p_i = |A_{(i)}|^2/\|A\|_F^2, i = 1 \ldots m$.

[4] Observe that if $s = m$, the error in this case is zero, unlike 1 and 2.

[5] We can use martingale arguments to show that $\|A - P\|_{F,2}$ is tightly concentrated around its mean, see [8, 9, 6].

Preprocessing step:

1. **For** $i = 1$ **to** m compute $|A_{(i)}|^2$.
2. Compute $\|A\|_F^2 = \sum_{i=1}^{m} |A_{(i)}|^2$.
3. Compute $z_i = \sum_{j=1}^{i} |A_{(j)}|^2$,$i = 1 \ldots m$.

Sampling step:

1. Pick a real number r,uniformly at random from $(0, \|A\|_F^2]$.
2. Find i such that $z_i < r \leq z_{i+1}$ and include row i in the sample.

It is easy to see that $\mathbf{Pr}(\text{pick } i) = |A_{(i)}|^2/\|A\|_F^2$. We can repeat the sampling step s times to form S. The total running time of the preprocessing step is mn and of the sampling steps $s \log m$; we use binary search to implement the latter.

We emphasize that our sampling can be viewed as a two-pass algorithm: it reads the input matrix once to form the z_i's, decides which rows to include in the sample and, in a second pass, extracts these rows. Observe that once S is formed, A can be discarded. Thus, we only need $O(sn)$ RAM space to store S and not $O(mn)$ RAM space to store A.

Theorem 2. *After the preprocessing step, the algorithm runs in $O(n)$ time.*

Proof: The sampling step needs $s \log m$ operations; the scaling of the rows prior to including them in S needs sn operations. Computing SS^T takes $O(s^2 n)$ time and computing its SVD $O(s^3)$ time. Finally, we need to compute H, which can be done in $O(nsk)$ operations. Thus, the overall running time (excluding the preprocessing step) is $O(s \log m + s^2 n + s^3 + nsk)$. Since s and k are constants, the total running time of the algorithm is $O(n)$. However, the constant hidden in this big-Oh notation is large, since it is proportional to the square of the number of rows that are included in our sample.

$\diamond$

Finally, we note that we could modify the algorithm to pick columns of A instead of rows and compute approximations to the *left* singular vectors. The bounds in Theorem 1 remain essentially the same (rows become columns and m becomes n). P is now equal to $RR^T A$, where R is an $m \times k$ matrix containing our approximations to the top k *left* singular vectors. The analysis of Theorem 2 holds and the running time of the algorithm is $O(m)$.

4 Experiments

4.1 Setup

We implemented our randomized SVD algorithm using weighted sampling with replacement, uniform sampling with replacement and uniform sampling without

replacement. We did not use weighted sampling without replacement, since we have no theoretical bounds for it. These experiments returned approximations to the top k right singular vectors; we also implemented the same experiments sampling columns instead of rows, thus approximating the top k left singular vectors. We ran every experiment 20 times for each image-matrix.

We fixed k for each image so that $\|A-A_k\|_F^2/\|A\|_F^2$ is small ($\leq 1\%$). We varied s (the number of rows or columns that are included in our sample) between k and $k+160$ (in increments of 8). We ran our experiments on all images of the `MatLab` ImDemos [21] directory (more than 40 images of different characteristics). We present a variety of results, both with respect to accuracy and running time.

To measure the error of our approximations to the top k right (or left) singular vectors, we compute the rank k approximation P to A using H (or R, see section 3.3) and the relative error of the approximation, namely $\|A-P\|_{F,2}^2/\|A\|_{F,2}^2$. We also compute (for each image) the relative error of the optimal rank k approximation to A, namely $\|A-A_k\|_{F,2}^2/\|A\|_{F,2}^2$.

The speedup is measured as the ratio of the time spent by the deterministic SVD algorithm to compute the right singular vectors (which are used to compute A_k) over the time needed by our randomized SVD algorithm to compute H or R (which is used to compute P). In section 4.3 we compare the running time of our approach with the running time of Lanczos/Arnoldi techniques.

4.2 Accuracy

General observations Our first goal is to demonstrate the accuracy of our algorithm. In figures 1 through 4 we plot the average loss in accuracy for different values of s (number of rows sampled) over all images. The loss in accuracy for a particular image and a specific s is defined as the relative error of our approximation *minus* the relative error of the optimal rank k approximation, namely

$$\|A-P\|_{F,2}/\|A\|_{F,2}^2 - \|A-A_k\|_{F,2}^2/\|A\|_{F,2}^2 \geq 0$$

We average the above error over all images (for the same value of s) and plot the results. It should be obvious from the plots that as the number of rows included in our sample increases the accuracy increases as well.

The most important observation is that our methods seem to return very reasonable relative error bounds, much better than the ones guaranteed by Theorem 1. The second observation is that uniform sampling without replacement -usually- returns the best results! The reason for this phenomenon is twofold: sampling without replacement guarantees that more rows are included in the sample (while weighted sampling is biased towards including heavier rows, even more than once[6]) *and* the lengths of the rows in our matrices are -more or less- in the same range (see section 3.2). We also observe that uniform sampling with replacement and weighted sampling return almost the same relative error in all

[6] This feature of weighted sampling is very useful when dealing with large, sparse matrices (see i.e. [13])

cases (the distance between the 2 curves is within the standard deviation of the experiments).

We now examine the speedup of our technique, for various values of s, over full SVD. The speedups (assuming uniform sampling w/out replacement) are: 827, 230, 95, 70, 56, 45, 35, 30, 25, 21, 18, 15, 13, 11, 10, 8, 7, 6, 6, where each value corresponds to a sample of size $s = k+8 \cdot i$, $i = 1, \ldots, 19$. The corresponding values for approximating the left singular vectors (column sampling) are: 643, 341, 170, 104, 76, 54, 44, 37, 29, 25, 21, 18, 15, 13, 11, 9, 8, 7, 6 (again using uniform sampling w/out replacement). We should emphasize here that the above comparison is not fair; full SVD returns *all* singular vectors of a matrix, while our technique approximates the top k singular vectors. We present a fair running time comparison in section 4.3; still, we believe that the above numbers illustrate the power of our technique.

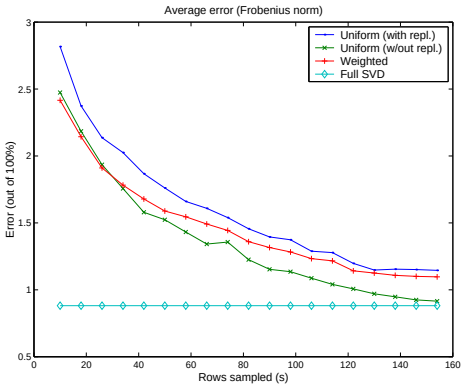

Fig. 1. F-norm error (row sampling)

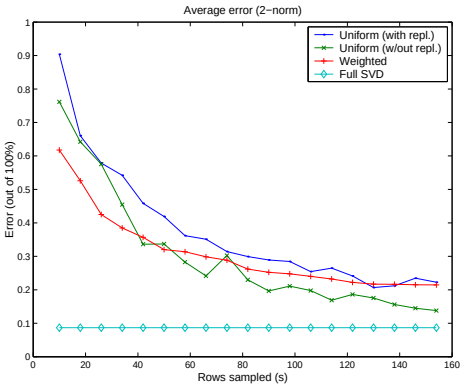

Fig. 2. 2-norm error (row sampling)

Case-studies on 2 images We present results on 2 images: the well-known `baboon` image and a large and more complicated image, the `hand_with_sphere` image[7]. In figures 5 through 8 we plot the error of our approximation with respect to the 2-norm and the Frobenius norm as a function of s. All relative error values are out of 100%. For the `baboon` image we set $k = 73$, thus seeking approximations to the top 73 right singular vectors, while for the `hand_with_sphere` image we set $k = 191$.

Figures 9-12 show the optimal rank k approximation to the image and our randomized rank k approximation to the image. One can easily observe that our approximation is quite accurate, even though most of the original image was discarded while computing the singular vectors!

[7] Hand with Reflecting Sphere image (1935), copyright by M.C. Escher, Cordon Art, Baarn, Netherlands.

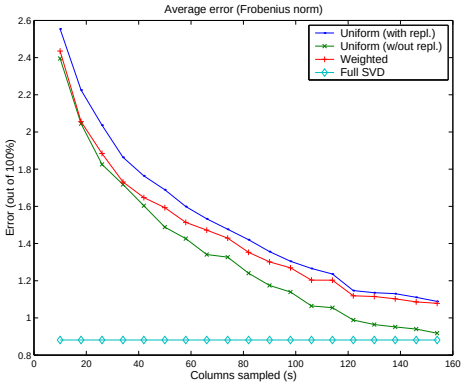

Fig. 3. F-norm error (column sampling)

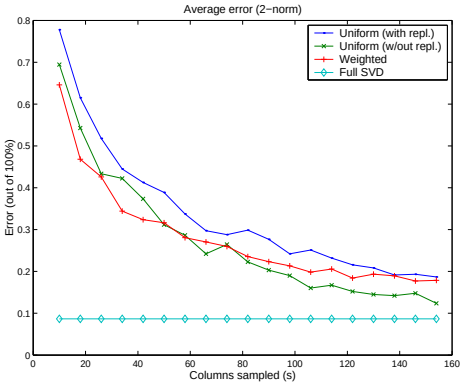

Fig. 4. 2-norm error (column sampling)

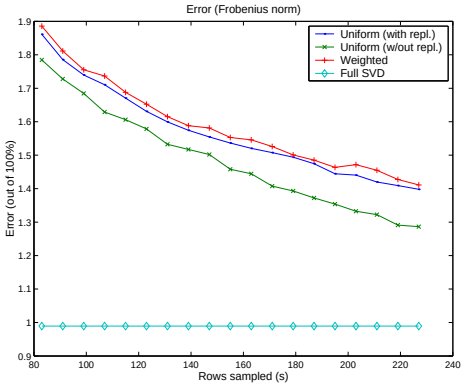

Fig. 5. Baboon (F-norm error, row sampling)

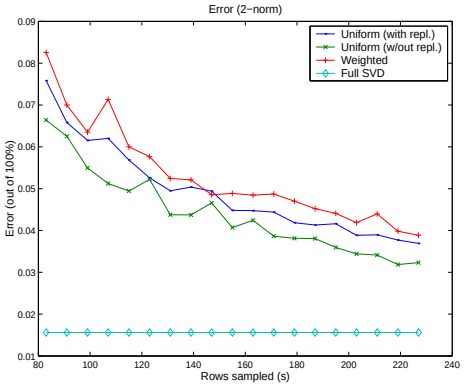

Fig. 6. Baboon (2-norm error, row sampling)

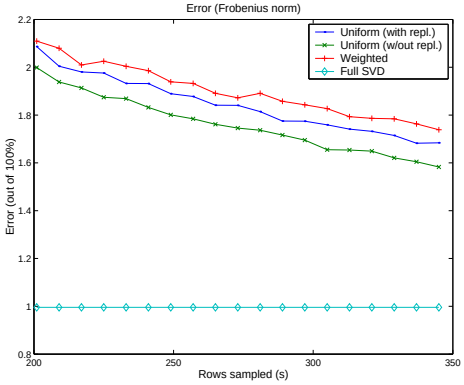

Fig. 7. Hand with sphere (F-norm error, row sampling)

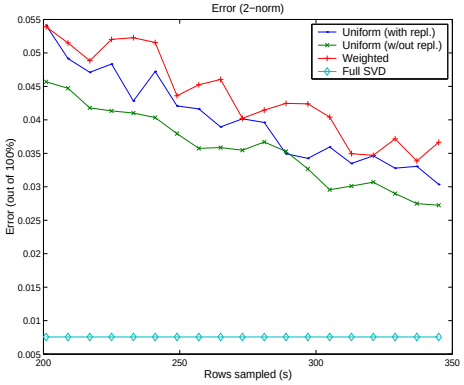

Fig. 8. Hand with sphere (2-norm error, row sampling)

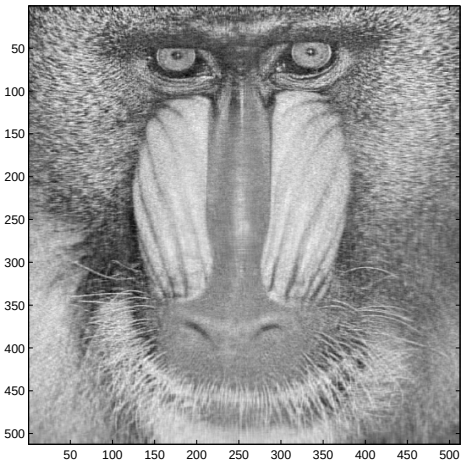

Fig. 9. Baboon (rank 73 approximation, error=0.99%)

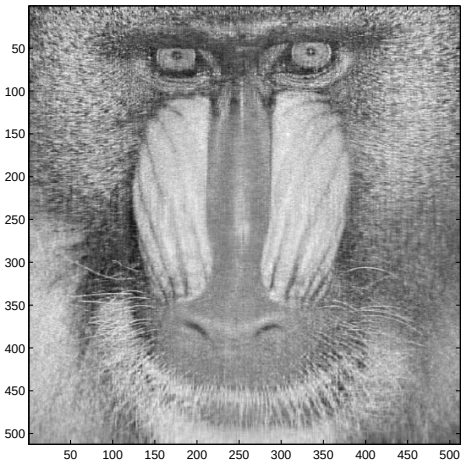

Fig. 10. Baboon (uniform sampling w/out repl. 203 rows, error=1.33%)

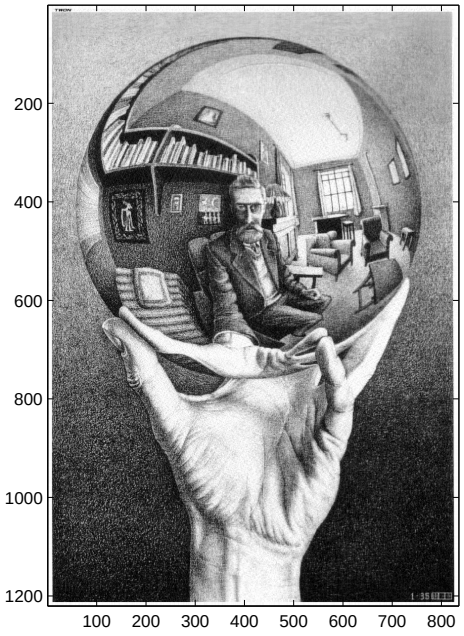

Fig. 11. Hand with sphere (rank 191 approximation, error=1%)

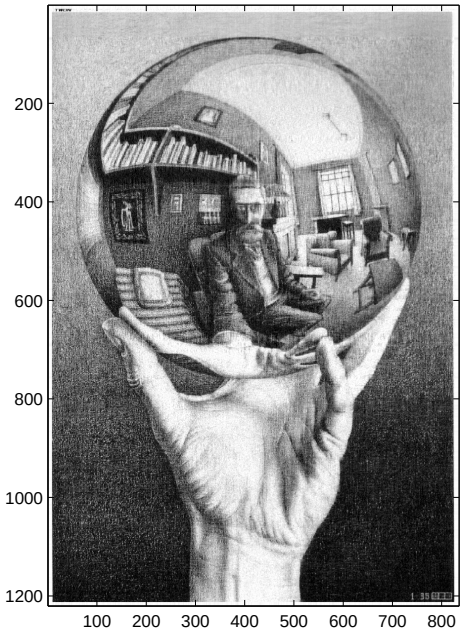

Fig. 12. Hand with sphere (uniform sampling w/out repl. 321 rows, error=1.65%)

4.3 Running time

In this section we compare the running times of our algorithms vs. the running time of the well-known Lanczos/Arnoldi methods. The latter are the dominant techniques used to approximate singular vectors of matrices. There is an extensive literature exploring the power of Lanczos/Arnoldi methods (see e.g. [17]); we give a brief, high-level presentation of Lanczos/Arnoldi methods.

Lanczos/Arnoldi methods Consider a symmetric $n \times n$ matrix A. The heart of these techniques is the ℓ-dimensional Krylov subspace of A, defined as

$$\mathcal{K}_\ell(x) = \textbf{span}\left(x, Ax, A^2x, A^3x, \ldots, A^\ell x\right)$$

where x is a -usually random- vector in $\mathcal{R}^n$. Assume that Q is an $n \times \ell$ orthogonal matrix, such that the columns of Q form an orthogonal basis for $\mathcal{K}_\ell(x)$; then, Q^TAQ is an $\ell \times \ell$ matrix whose singular values are "close" to the singular values of A. Similarly, if $\tilde{v}^{(i)}$ is the i-th right singular vector of Q^TAQ, then $Q\tilde{v}^{(i)}$ approximates the i-th right singular vector of A. We note that Q^TAQ is an $\ell \times \ell$ matrix, thus its SVD can be computed in $O(\ell^3) = O(1)$ time. As ℓ increases, the accuracy of the approximations increases as well; for theoretical error bounds see [17].

In practice, computing Q *after* $\mathcal{K}_\ell(x)$ has been formed is very unstable; there are huge losses in accuracy due to numerical errors. Instead, the following incremental, iterative algorithm is employed.

Basic Lanczos/Arnoldi

1. Pick $x \in \mathcal{R}^n$ such that $|x|_2 = 1$.
2. Set $q^{(1)} = x$.
3. **For** $i = 1$ **to** ℓ
   - Set $t = A \cdot q^{(i)}$.
   - Orthonormalize t with respect to $q^{(1)}, \ldots, q^{(i-1)}$.
   - Set $q^{(i+1)} = t$.

Now, $q^{(1)}, \ldots, q^{(\ell)}$ form an orthogonal basis for $\mathcal{K}_\ell(x)$; in practice, it is easy to modify the above algorithm to compute $H_\ell = Q_\ell^T A Q_\ell$ as well. There are various modifications of the above algorithm to increase its stability, i.e. a second orthogonalization step might be added etc.

It should be obvious that as ℓ increases, the above procedure becomes very expensive; a remedy was suggested by Sorensen [18]: at some point, *restart* the algorithm with a starting vector x which is "rich" in the subspace spanned by the top k eigenvectors. Since we already have some information on the top k eigenvectors of A (from the iterations done thus far) we can compute such a starting vector; Sorensen described an elegant solution to this problem. Indeed, most Lanczos/Arnoldi codes incorporate restarting to reduce their running time/memory requirements.

Finally, even though the above description was focused on symmetric matrices, one should note that given an arbitrary $m \times n$ matrix B, $A = \begin{bmatrix} \mathbf{0} & B^T \\ B & \mathbf{0} \end{bmatrix}$ is an $(m+n) \times (m+n)$ symmetric matrix.

Comparing Lanczos to our technique We remind the reader that, from Theorem 2, the running time of our algorithm using uniform sampling without replacement is $O(n)$. The running time required for *convergence* of Lanczos/Arnoldi techniques to machine precision accuracy is usually $O(n^2k^2)$ for $n \times n$ symmetric matrices (see e.g. [11]). It should be obvious that step 3 of the Lanczos/Arnoldi algorithm described above takes at least $O(n^2)$ time; as more and more vectors are added in $\mathcal{K}_\ell(x)$, the orthogonalization becomes more and more expensive. Implicit restarting can reduce these costs, by guaranteeing that a bounded number of orthogonalizations will be performed; still, in order to create a Krylov subspace of dimension ℓ we need at least $O(\ell n^2)$ time.

By construction, our algorithm seems faster than Lanczos/Arnoldi techniques; for a fair comparison, we are interested in the accuracy vs. running time performance of the two approaches; a theoretical comparison of such performances is difficult. Thus, we resort to an experimental evaluation in the next section.

Finally, we emphasize that our approach works by keeping only a few rows of the original matrix in RAM; in the case of uniform sampling without replacement, one pass through A is all we need. On the other hand, the Lanczos/Arnoldi algorithm requires multiple accesses to A; one could easily see that if A were a large matrix stored in secondary devices the performance penalty would be significant.

Comparing the accuracy experimentally We used a widely available implementation of the Lanczos/Arnoldi algorithm, the `svds` command of MatLab. `svds` is an interface, calling `eigs`, which computes the top few eigenvalues/eigenvectors of a matrix. `eigs` is also an interface, calling routines from `ARPACK` iteratively to perform the computations.

We ran `svds` in our image-matrices; we forced an upper bound on the dimension (denoted by ℓ) of the Krylov subspace computed by `svds`. Indeed, we tried different values for ℓ, starting at $\ell = k$; since we seek to approximate the top k singular vectors, it should be obvious that a Krylov subspace of lesser dimension could not return approximations to the top k singular vectors.

We emphasize that `svds` does implement restarting to achieve maximum speedup and minimize memory usage. `svds` also offers timing results, which essentially bypass the inherent inefficiencies of MatLab; e.g. it does not time `for loops`. Thus, we believe that the running time that we measured for `svds` accurately reflects running time spent in actual computations; we remind that `ARPACK` is built in `C/Fortran`.

Experiments We now present running time vs. accuracy curves on a few images for `svds` and our algorithm. In all cases, our technique was faster, usually significantly so, regardless of the number of vectors (k) that we seek to approximate. In the figures 13–18 we present results on a few images; we chose these images to represent different values of k.

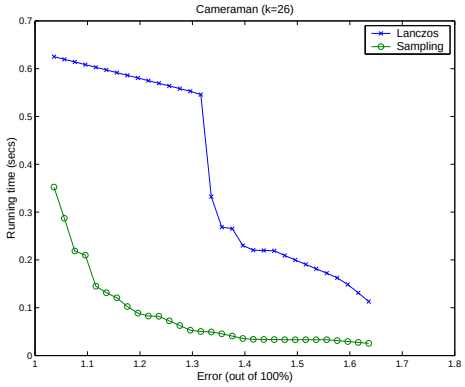

Fig. 13. Cameraman

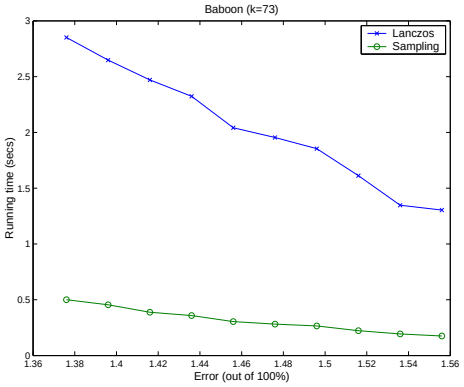

Fig. 14. Baboon

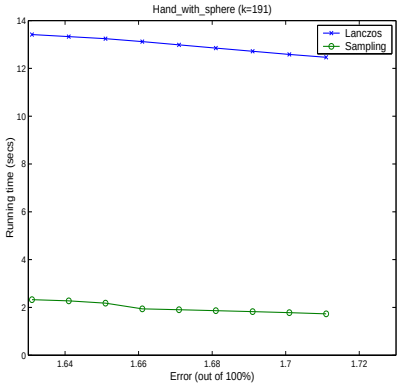

Fig. 15. Hand_with_sphere

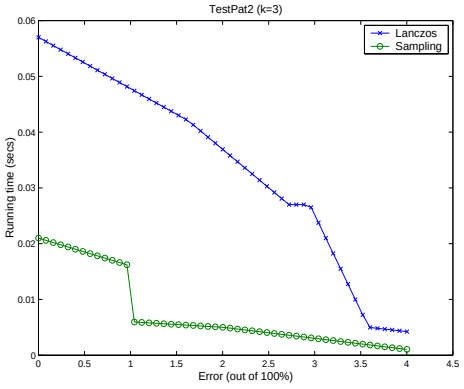

Fig. 16. TestPat2

We present two tables for the `baboon` image (a 512×512 image). Table 1 shows how the accuracy of the approximation increases as the dimension (ℓ) of the Krylov subspace increases. We also give an estimate of the number of operations needed to construct $\mathcal{K}_\ell(x)$ ($512^2 \cdot \ell$) as well as the actual running time of `svds` in every instance. We note that this running time includes the orthogonalization

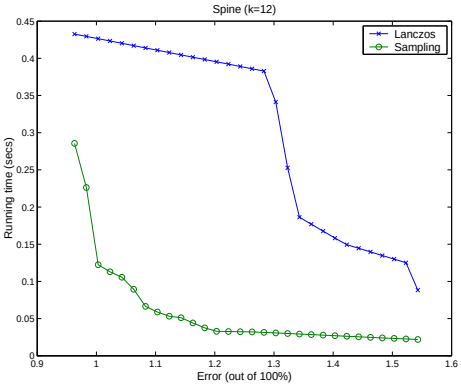

Fig. 17. Spine

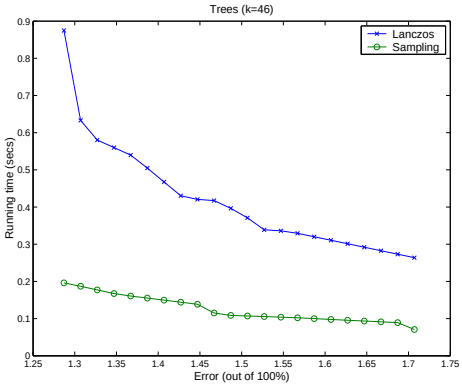

Fig. 18. Trees

steps as well, which are not included in our estimate since restarting makes such a prediction impossible. In table 2, we present similar results for our approach; namely, how the accuracy increases as a function of s, as well as the number of operations required by our approach to compute the small matrix (see section 3) and its SVD ($512 \cdot s^2 + s^3$). Finally, in figures 19 and 20 we show low rank approximations of the `baboon` and `hand_with_sphere` image, by approximating the top few right singular vectors using Lanczos/Arnoldi techniques.

5 Open problems

The most interesting open problem is to combine our sampling technique with Lanczos/Arnoldi methods. It is well-known that the speed of convergence of the Lanczos/Arndoldi algorithm crucially depends on the starting vector (see section 4.3). Our sampling technique could be used to create starting vectors for Lanczos/Arnoldi iterations; more specifically, we could use our randomized SVD algorithm to generate vectors that are rich in the directions of the singular vectors that we seek to approximate. We experimented with the above idea in our test set but we did not achieve significant gains; we were only able to save a few iterations. This essentially means that random starting vectors are actually quite good for our test set. On the other hand, this hybrid scheme might be useful for matrices with poorly separated singular values, where random vectors might be almost orthogonal to the singular vectors that we seek to approximate with high probability.

A second problem is to theoretically analyze and compare the accuracy vs. running time tradeoff of Lanczos/Arnoldi techniques and our algorithm; such analysis seems quite challenging.

Acknowledgments : We wish to thank Ravi Kannan for many helpful discussions. The first author was supported by NSF grant CCR-9820850 and the second author by NSF grants CCR-9983832 and ITR/SY-0121154.

Dimension (ℓ)	**Computing** $\mathcal{K}_\ell(x)$ ($\times 10^6$)	**Running time (in seconds)**	**Error (out of 100%)**
73	19.1365	1.2980	1.5586
77	20.1851	1.3610	1.5299
81	21.2337	1.4580	1.5255
85	22.2822	1.5090	1.5212
89	23.3308	1.5600	1.5198
93	24.3794	1.6350	1.5144
97	25.4280	1.6800	1.5029
101	26.4765	1.7740	1.4990
105	27.5251	1.8520	1.4965
109	28.5737	1.9430	1.4791
113	29.6223	2.0220	1.4570
117	30.6708	2.1260	1.4518
121	31.7194	2.2200	1.4517
125	32.7680	2.3120	1.4381
129	33.8166	2.3940	1.4233
133	34.8652	2.4360	1.4226
137	35.9137	2.5370	1.4036
141	36.9623	2.6300	1.4000
145	38.0109	2.7090	1.3823
149	39.0595	2.8010	1.3764
153	40.1080	2.9040	1.3756

Table 1. Statistics for *svds* for the *baboon* image.

References

1. http://cluster.cs.yale.edu/.
2. P. Anandan and M. Irani. Factorization with uncertainty. *IJCV*, 49(2-3):101–116, September 2002.
3. H.C. Andrews and C.L. Patterson. Singular value decomposition image coding. *IEEE Trans. on Communications*, pages 425–432, April 1976.
4. H.C. Andrews and C.L. Patterson. Singular value decompositions and digital image processing. *IEEE Trans. ASSP*, pages 26–53, February 1976.
5. E. Drinea, P. Drineas, and P. Huggins. A randomized singular value decomposition algorithm for image processing. *Panhellenic Conference on Informatics*, 2001.
6. P. Drineas. *Fast Monte-Carlo algorithms for Approximate matrix operations and applications.* PhD thesis, Yale University, 2002.
7. P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. *Proceedings of the 10th Symposium on Discrete Algorithms*, pages 291–299, 1999.
8. P. Drineas and R. Kannan. Fast Monte-Carlo algorithm for approximate matrix multiplication. *Proceedings of the 10th Annual Symposium on Foundations of Computer Science*, pages 452–459, 2001.
9. P. Drineas and R. Kannan. Fast Monte-Carlo algorithms for approximating the product of matrices. under submission, 2002.

Rows (s)	**Computing** $\texttt{svd}(S \cdot S^T)$ ($\times 10^6$)	**Running time (in seconds)**	**Error (out of 100%)**
81	3.8907	0.0640	1.7846
89	4.7605	0.0770	1.7279
97	5.7301	0.1000	1.6844
105	6.8024	0.1270	1.6289
113	7.9806	0.1390	1.6062
121	9.2678	0.1540	1.5785
129	10.6669	0.1970	1.5325
137	12.1811	0.2200	1.5171
145	13.8134	0.2600	1.5020
153	15.5670	0.2970	1.4580
161	17.4448	0.3460	1.4446
169	19.4500	0.4000	1.4076
177	21.5857	0.4690	1.3931
185	23.8548	0.5070	1.3724
193	26.2605	0.5660	1.3538
201	28.8059	0.6040	1.3326
209	31.4940	0.6880	1.3224
217	34.3279	0.8420	1.2864

Table 2. Statistics for uniform sampling w/out replacement for the *baboon* image.

10. A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low rank approximations. *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 370–378, 1998.
11. G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1989.
12. T. Huang and P. Narendra. Image restoration by singular value decomposition. *Applied Optics*, 14(9):2213–2216, September 1974.
13. F. Jiang, R. Kannan, M. Littman, and S. Vempala. Efficient singular value decomposition via improved document sampling. *Technical Report CS-99-5, Duke University*, February 1999.
14. B. Moghaddam and A. Pentland. Probabilistic visual learning for object representation. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 19(7):696–710, 1997.
15. H. Murase and S.K. Nayar. Visual learning and recognition of 3-d objects from appearance. *IJCV*, 14(1):5–24, January 1995.
16. M. Murphy. Comparison of transform image coding techniques for compression of tactical imagery. *SPIE*, 309:212–219, 1981.
17. B. Parlett. *The Symmetric Eigenvalue Problem*. Classics in Applied Mathematics. SIAM, 1997.
18. D. Sorensen. Implicit application of polynomial filters in a k-step arnoldi method. *SIAM J. Matrix Analysis and Applications*, 13:357–375, 1992.
19. M. Turk and A.P. Pentland. Face recognition using eigenfaces. In *CVPR91*, pages 586–591, 1991.
20. P. Wu, B.S. Manjunath, and H.D. Shin. Dimensionality reduction for image retrieval. In *ICIP00*, page WP07.03, 2000.
21. www.MathWorks.com.

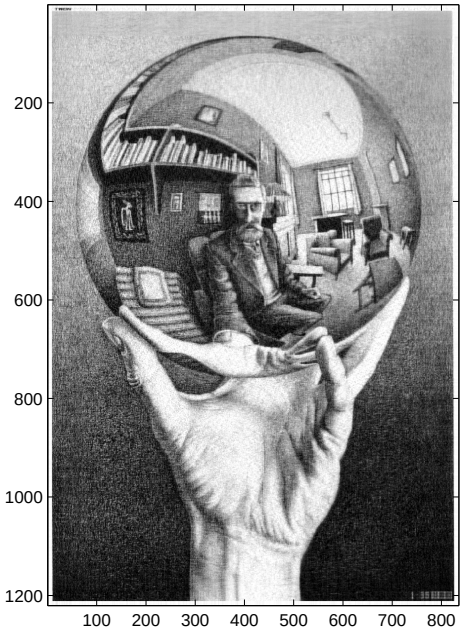

Fig. 19. Hand with sphere. 199 iterations, error = 1.67%.

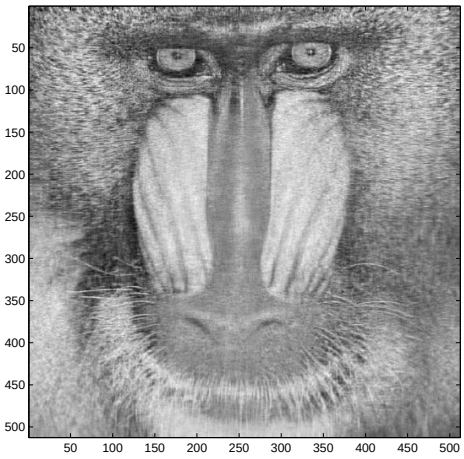

Fig. 20. Baboon. 149 iterations, error = 1.38%.

22. G. Zientara, L. Panych, and F. Jolesz. Dynamic adaptive MRI using multiresolution SVD encoding incorporating optical flow-based predictions. Technical report, National Academy of Sciences Committee on the "Mathematics and Physics of Emerging Dynamic Biomedical Imaging", November 1993.
23. G. Zientara, L. Panych, and F. Jolesz. Dynamically adaptive MRI with encoding by singular value decomposition. *MRM*, 32:268–274, 1994.