# Lanczos Vectors versus Singular Vectors for Effective Dimension Reduction

Jie Chen and Yousef Saad

***Abstract*— This paper takes an in-depth look at a technique for computing filtered matrix-vector (mat-vec) products which are required in many data analysis applications. In these applications the data matrix is multiplied by a vector and we wish to perform this product accurately in the space spanned by a few of the major singular vectors of the matrix. We examine the use of the Lanczos algorithm for this purpose. The goal of the method is identical with that of the truncated singular value decomposition (SVD), namely to preserve the quality of the resulting mat-vec product in the major singular directions of the matrix. The Lanczos-based approach achieves this goal by using a small number of Lanczos vectors, but it does not explicitly compute singular values/vectors of the matrix. The main advantage of the Lanczos-based technique is its low cost when compared with that of the truncated SVD. This advantage comes without sacrificing accuracy. The effectiveness of this approach is demonstrated on a few sample applications requiring dimension reduction, including information retrieval and face recognition. The proposed technique can be applied as a replacement to the truncated SVD technique whenever the problem can be formulated as a filtered mat-vec multiplication.**

***Index Terms*— Dimension reduction, SVD, Lanczos algorithm, information retrieval, latent semantic indexing, face recognition, PCA, eigenfaces.**

## I. Introduction

In many data analysis problems, such as information retrieval and face recognition, it is desired to compute a matrix-vector (mat-vec) product between the data matrix $A$ and a certain "query" vector $b$. However, in order to remove noise and/or extract latent structures of the data, the optimal rank-$k$ approximation of $A$, denoted $A_k$, is commonly used in place of $A$ itself. Specifically, if we let $A$ have the SVD

$$A = U\Sigma V^T,$$

and define

$$A_k = U_k\Sigma_k V_k^T,$$

where $U_k$ (resp. $V_k$) consists of the first $k$ columns of $U$ (resp. $V$) and $\Sigma_k$ is the $k$-th principal submatrix of $\Sigma$, then the matrix $A_k$ is the best rank-$k$ approximation of $A$ in the 2-norm or Frobenius norm sense [1], [2]. Then the product

$$A_k b \tag{1}$$

is a *filtered* version of the mat-vec product $Ab$. The use of this filtered mat-vec product is illustrated in the following two sample applications.

This work was supported by NSF grants DMS 0510131 and DMS 0528492 and by the Minnesota Supercomputing Institute.

The authors are with the Department of Computer Science and Engineering, University of Minnesota at Twin Cities, MN 55455. Email: {jchen, saad}@cs.umn.edu

*a) Latent Semantic Indexing (LSI):* LSI [3], [4] is an effective information retrieval technique which computes the relevance scores for all the documents in a collection in response to a user query. In LSI, a collection of documents is represented as a term-document matrix $X = [x_{ij}]$ where each column vector represents a document and $x_{ij}$ is the weight of term $i$ in document $j$. A query is represented as a pseudo-document in a similar form—a column vector $q$. By performing dimension reduction, each document $x_j$ ($j$-th column of $X$) becomes $P_k^T x_j$ and the query becomes $P_k^T q$ in the reduced-rank representation, where $P_k$ is the matrix whose column vectors are the $k$ major left singular vectors of $X$. The relevance score between the $j$-th document and the query is then computed as the cosine of the angle between $P_k^T x_j$ and $P_k^T q$:

$$\cos(P_k^T x_j, P_k^T q) = \frac{(P_k^T x_j)^T (P_k^T q)}{\|P_k^T x_j\| \, \|P_k^T q\|} = \frac{(X_k e_j)^T q}{\|X_k e_j\| \, \|P_k^T q\|},$$

where $X_k$ is the best rank-$k$ approximation of $X$, and $e_j$ is the $j$-th column of the identity matrix. Hence, relevance scores for all the documents in the collection are equivalently represented by the column vector

$$X_k^T q \tag{2}$$

modified by scaling each entry with the norm of the corresponding row of $X_k^T$. Thus, LSI reduces to computing the filtered mat-vec product $X_k^T q$.

*b) Eigenfaces:* The eigenfaces methodology [5] applies principal component analysis to face recognition. Similar to LSI, face images in the training set are represented as column vectors to form a matrix $F$. A difference with LSI is that the columns of $F$ are centered by concurrently subtracting their mean. An image $p$ in the test set is also represented as a column vector. Let $Q_k$ be the matrix constituting the $k$ major left singular vectors of $F$. The similarity between the $j$-th training image and the test image is then computed as the cosine of the angle between $Q_k^T f_j$ and $Q_k^T p$, where $f_j$ is the $j$-th column of $F$. Similar to the above derivation in LSI, the similarity scores for all the training images are represented as the column vector

$$F_k^T p \tag{3}$$

modified by scaling each entry with the norm of the corresponding row of $F_k^T$, where $F_k$ is the best rank-$k$ approximation of $F$. The main computation in the eigenfaces technique is the filtered mat-vec product $F_k^T p$.

A notorious difficulty in computing $A_k b$ is that it requires the truncated SVD, which is computationally expensive for large $A$. (See Table I for a summary of the costs.) In addition, frequent changes in the data set require an update of the SVD, and this is not an easy task. Much research has been devoted to the problem of updating the (truncated) SVD [6]–[9]. A drawback of these approaches is that the resulting (truncated) SVD loses accuracy after frequent updates.

To bypass the truncated SVD computation, Kokiopoulou and Saad [10] introduced polynomial filtering techniques, and Erhel *et al.* [11] and Saad [12] proposed algorithms for building good polynomials to use in such techniques. These methods all efficiently compute a sequence of vectors $\phi_i(A)b$ where $\phi_i$ is a polynomial of degree at most $i$ such that $\phi_i(A)b$ is progressively a better approximation to $A_k b$ as $i$ increases.

In this paper, we investigate the Lanczos algorithm as a means to compute a sequence of vectors $\{s_i\}$ (or $\{t_i\}$) which progressively approximate $Ab$. This technique does not use explicit polynomials, but it shares the advantages of polynomial filtering techniques: easy-to-update recurrence formulas and low cost relative to the direct SVD. Both approximating sequences ($\{s_i\}$ and $\{t_i\}$) rapidly converge to $Ab$ in the major singular directions of $A$, achieving the same goal as that of the truncated SVD which preserves the major singular components. Hence when $i = k$, the vector $s_i$ or $t_i$ is a good alternative to the filtered mat-vec product $A_k b$.

More appealingly, the proposed technique can be split into a preprocessing phase and a query response phase, where a query vector $b$ is involved only in the second phase. In contrast, polynomial filtering techniques cannot be naturally separated in two phases. Hence, the proposed technique will certainly outperform polynomial filtering techniques in common practical situations where many queries are to be processed.

Note that the Lanczos algorithm is a commonly used method for large symmetric eigenvalue problems and sparse SVD problems. As it turns out, this approach puts too much emphasis on individual singular vectors and is relatively expensive (see, e.g. [13], and the discussions in Section IV-B). We exploit the Lanczos algorithm to efficiently compute the filtered mat-vec products without explicitly computing the singular components of the matrix. Therefore, when computational efficiency is critical, this technique may be a favorable alternative to the best rank-$k$ approximation.

### A. Related work

The proposed usage of the Lanczos algorithm belongs to a broad class of methods, the so-called *Krylov subspace* methods. Blom and Ruhe [14] suggested the use of the Golub-Kahan bidiagonalization techique [15], which also belongs to this class, for information retrieval. While it is well-known that the Golub-Kahan bidiagonalization and the Lanczos algorithm are closely related, we highlight the differences between these two techniques and note the contributions of this paper in the following:

1) The framework described in this paper could be considered an extension of that of [14]. The method in [14] is equivalent to a special case of one of our approximation schemes (the left projection approximation). In particular the method in [14] used the normalized query as the initial vector. This means that the construction of the Krylov subspace in their method would normally be repeated anew for each different query, which is not economical. In contrast, our method constructs the Krylov subspace only once as the preprocessing phase.
2) The method in [14] generated two sets of vectors that essentially replaced the left and right singular vectors. In contrast, we generate only one set of (Lanczos) vectors. This saving of storage can be critical for large scale problems.
3) We strengthen the observation made in [14] that the difference between $Ab$ and the approximation vector ($s_i$) monotonically decreases, by proving rigorously that this difference decays very rapidly in the major singular directions of $A$. Illustrations are provided to relate the convergence behavior of the method based on the good separation of the largest singular values (eigenvalues) in Section VII.
4) Several practical issues which were not discussed in [14], are explored in detail. These include the choice of the approximation schemes depending on the shape of the matrix, and other practical choices related to memory versus computational cost trade-offs.
5) The technique described in this paper has a broad variety of applications and is not limited to information retrieval. Indeed it can be applied whenever the problem can be formulated as a filtered mat-vec multiplication. This paper presents two such applications.

### B. Organization of the paper

The symmetric Lanczos algorithm is reviewed in Section II, and the two proposed Lanczos approximation schemes are introduced in Section III. Section IV provides a detailed analysis of the convergence of the proposed techniques and their computational complexities. Section V discusses how to incorporate entry scaling into the approximation vectors. Applications of the proposed approximation schemes to information retrieval and face recognition are illustrated in Section VI. Finally, extensive experiments are shown in Section VII, followed by concluding remarks in Section VIII.

## II. The Symmetric Lanczos Algorithm

Given a symmetric matrix $M \in \mathbb{R}^{n\times n}$ and an initial unit vector $q_1$, the Lanczos algorithm builds an orthonormal basis of the Krylov subspace

$$\mathcal{K}_k(M, q_1) = \text{range}\{q_1, Mq_1, M^2q_1, \ldots, M^{k-1}q_1\}.$$

The Lanczos vectors $q_i$, $i = 1, \ldots, k$ computed by the algorithm satisfy the 3-term recurrence

$$\beta_{i+1}q_{i+1} = Mq_i - \alpha_i q_i - \beta_i q_{i-1}$$

with $\beta_1 q_0 \equiv 0$. The coefficients $\alpha_i$ and $\beta_{i+1}$ are computed so as to ensure that $\langle q_{i+1}, q_i\rangle = 0$ and $\|q_{i+1}\| = 1$. In exact arithmetic, it turns out that $q_{i+1}$ is orthogonal to $q_1, \ldots, q_i$ so the vectors $q_i$, $i = 1, \ldots, k$ form an orthonormal basis of the Krylov subspace $\mathcal{K}_k(M, q_1)$. An outline of the procedure is illustrated in Algorithm 1. The time cost of the procedure is $O(kn^2)$. If the matrix is sparse, then the cost reads $O(k(nnz + n))$, where $nnz$ is the number of nonzeros of $M$.

If $Q_k = [q_1, \ldots, q_k] \in \mathbb{R}^{n\times k}$, then an important equality which results from the algorithm is

$$Q_k^T M Q_k = T_k = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \beta_k & \alpha_k \end{bmatrix}. \quad (4)$$

An eigenvalue $\theta$ of $T_k$ is called a *Ritz value*, and if $y$ is an associated eigenvector, $Q_k y$ is the associated *Ritz vector*. As $k$ increases more and more Ritz values and vectors will converge towards eigenvalues and vectors of $M$ [2], [16].

**Algorithm 1** The Lanczos Procedure

**Input:** $M$, $q_1$, $k$
**Output:** $q_1, \ldots, q_{k+1}$, $\alpha_i$'s, $\beta_i$'s
1: Set $\beta_1 \leftarrow 0$, $q_0 \leftarrow 0$.
2: **for** $i \leftarrow 1, \ldots, k$ **do**
3: $\quad w_i \leftarrow M q_i - \beta_i q_{i-1}$
4: $\quad \alpha_i \leftarrow \langle w_i, q_i \rangle$
5: $\quad w_i \leftarrow w_i - \alpha_i q_i$
6: $\quad \beta_{i+1} \leftarrow \|w_i\|_2$
7: $\quad$ **if** $\beta_{i+1} = 0$ **then**
8: $\quad\quad$ stop
9: $\quad$ **end if**
10: $\quad q_{i+1} \leftarrow w_i / \beta_{i+1}$
11: **end for**

### A. Re-orthogonalization

It is known that in practice the theoretical orthogonality of the computed Lanczos vectors $q_i$'s is quickly lost. This loss of orthogonality is triggered by the convergence of one or more eigenvectors [17]. There has been much research devoted to re-instating orthogonality. The simplest technique is to re-orthogonalize each new vector against each of the previous basis vectors. This amounts to adding the following line of pseudocode immediately after line 5 of Algorithm 1:

$$w_i \leftarrow w_i - \textstyle\sum_{j=1}^{i-1} \langle w_i, q_j \rangle \, q_j.$$

This additional *full re-orthogonalization* step increases the computational cost of the Lanczos procedure (by roughly $k^2 n/2$ computations), but the advantages are that all the $q_i$'s are guaranteed to be numerically orthogonal, and any subsequent process relying on the orthogonality is more rigorous.

It is important to add here that there are other more cost-effective re-orthogonalization procedures known. These procedures are not considered in this paper for simplicity, but they should be considered in any realistic implementation of the Lanczos-based technique. The best-known practical alternative to full re-orthogonalization is *partial re-orthogonalization* which consists of taking a re-orthogonalization step only when needed [18]–[20]. Another technique is the *selective re-orthogonalization* approach which exploits the fact that loss of orthogonality is seen only in the direction of the converged eigenvectors [17]. Using one of these approaches should lead to a considerable reduction of the cost of the Lanczos procedure, especially for situations requiring a large number of steps.

## III. Lanczos Approximation

The Lanczos algorithm which we briefly discussed above can be exploited in two ways to efficiently approximate the filtered mat-vec $A_k b$ without resorting to the expensive SVD.

### A. Left projection approximation

The first approach is to apply the Lanczos procedure to the matrix $M = AA^T$, resulting in the equality

$$Q_k^T A A^T Q_k = T_k. \tag{5}$$

We define the sequence

$$s_i := Q_i Q_i^T A b, \tag{6}$$

where $s_i$ is the orthogonal projection of $Ab$ onto the subspace $\mathrm{range}(Q_i)$. By this definition, the vector $s_i$ can be easily updated from the previous vector $s_{i-1}$:

$$s_i = s_{i-1} + q_i q_i^T A b. \tag{7}$$

The sequence $\{s_i\}$ is a progressive approximation to $Ab$ as $i$ increases. We call this the *left projection approximation*, as the projection operator $Q_i Q_i^T$ operates on the left hand side of $A$. (This is to be distinguished from the later definition of the right projection approximation.) In the course of the process to compute the sequence, the matrix $M$ need not be explicitly formed. The only computation involving $M$ is the mat-vec on line 3 of Algorithm 1. This computation can be performed as

$$w_i \leftarrow A(A^T q_i) - \beta_i q_{i-1}.$$

An analysis to be discussed shortly, will show that when $i = k$, the vector $s_k$ is a good approximation to $A_k b$. Algorithm 2 summarizes the steps to compute $s_k$.

**Algorithm 2** Left projection approximation

**Input:** $A$, $b$, $q_1$, $k$
**Output:** $s_k$
1: Apply Algorithm 1 on $M = AA^T$ to obtain Lanczos vectors $q_1, \ldots, q_k$, with line 3 modified to $w_i \leftarrow A(A^T q_i) - \beta_i q_{i-1}$.
2: $\tilde{s} \leftarrow Ab$
3: $s_0 \leftarrow 0$
4: **for** $i \leftarrow 1, \ldots, k$ **do**
5: $\quad s_i \leftarrow s_{i-1} + \langle \tilde{s}, q_i \rangle \, q_i$
6: **end for**

Note that the computation of $s_k$ in Algorithm 2 can equivalently be carried out as a sequence of mat-vec products, i.e., as $Q_k \times (Q_k^T \times (A \times b))$. The algorithm presents the computation of $s_k$ in the update form as this reveals more information on the sequence $\{s_i\}$.

### B. Right projection approximation

A second method consists of applying the Lanczos algorithm to the matrix $\bar{M} = A^T A$, resulting in the relation:

$$\bar{Q}_k^T A^T A \bar{Q}_k = \bar{T}_k. \tag{8}$$

We now define the sequence

$$t_i := A \bar{Q}_i \bar{Q}_i^T b, \tag{9}$$

and call this the *right projection approximation* to $Ab$. The vector $t_i$ is related to $t_{i-1}$ by

$$t_i = t_{i-1} + A \bar{q}_i \bar{q}_i^T b. \tag{10}$$

Similar to the left projection approximation, the matrix $\bar{M}$ does not need to be explicitly formed, and the vector $t_k$ is also a good approximation to $A_k b$ (shown later). Algorithm 3 summarizes the steps to compute $t_k$.

Again, note that lines 2–6 of the algorithm are equivalent to computing $t_k$ directly by $t_k = A \times (\bar{Q}_k \times (\bar{Q}_k^T \times b))$.

**Algorithm 3** Right projection approximation

**Input:** $A$, $b$, $\bar{q}_1$, $k$

**Output:** $t_k$

1: Apply Algorithm 1 on $\bar{M} = A^T A$ to obtain Lanczos vectors $\bar{q}_1, \ldots, \bar{q}_k$, with line 3 modified to $\bar{w}_i \leftarrow A^T(A\bar{q}_i) - \beta_i \bar{q}_{i-1}$.
2: $\tilde{t}_0 \leftarrow 0$
3: **for** $i \leftarrow 1, \ldots, k$ **do**
4: $\quad \tilde{t}_i \leftarrow \tilde{t}_{i-1} + \langle b, \bar{q}_i \rangle \bar{q}_i$
5: **end for**
6: $t_k \leftarrow A\tilde{t}_k$

## IV. Analysis

### A. Convergence

The reason why $s_k$ and $t_k$ are good approximations to $A_k b$ is the fact that the sequences $\{s_i\}$ and $\{t_i\}$ both rapidly converge to $Ab$ in the major left singular directions of $A$. This fact implies that $s_k$ and $t_k$, when projected to the subspace spanned by the major left singular vectors of $A$, are very close to the projection of $Ab$ if $k$ is sufficiently large. Since the goal of performing a filtered mat-vec $A_k b$ in data analysis applications is to preserve the accuracy of $Ab$ in these directions, the vectors $s_k$ and $t_k$ are good alternatives to $A_k b$. The rapid convergence is analyzed in the next.

Let $u_j$ be the $j$-th left singular vector of $A$. The rapid convergence of the sequence $\{s_i\}$ can be shown by establishing the inequality

$$\left|\langle Ab - s_i, u_j \rangle\right| \leq c_j \left\|Ab\right\| T_{i-j}^{-1}(\gamma_j), \tag{11}$$

for $i \geq j$, where $c_j > 0$ and $\gamma_j > 1$ are constants independent of $i$, and $T_l(\cdot)$ is the Chebyshev polynomial of the first kind of degree $l$. In other words, the difference between $Ab$ and $s_i$, when projected on the direction $u_j$, decays at least with the same order as $T_{i-j}^{-1}(\gamma_j)$. The detailed derivation of this inequality is shown in the Appendix. It can be similarly shown that

$$\left|\langle Ab - t_i, u_j \rangle\right| \leq \sigma_j \bar{c}_j \left\|b\right\| T_{i-j}^{-1}(\bar{\gamma}_j) \tag{12}$$

for constants $\sigma_j > 0$, $\bar{c}_j > 0$ and $\bar{\gamma}_j > 1$. Again, this inequality means that $t_i$ converges to $Ab$ in the major left singular directions $u_j$ of $A$ as $i$ increases.

The Chebyshev polynomial $T_l(x)$ for $x > 1$ is defined as

$$T_l(x) := \cosh(l \operatorname{arccosh}(x)) = \frac{1}{2}\left(e^{l\lambda} + e^{-l\lambda}\right)$$

where $\lambda = \operatorname{arccosh}(x)$. When $l$ is large the magnitude of $e^{-l\lambda}$ is negligible, hence

$$T_l(x) \approx \frac{1}{2} e^{l\lambda} = \frac{1}{2}\big(\exp(\operatorname{arccosh}(x))\big)^l.$$

Thus given $x$, the Chebyshev polynomial $T_l(x)$ grows in a manner that is close to an exponential in $l$. The base $\exp(\operatorname{arccosh}(x))$ is strictly larger than one when $x > 1$. In other words, the previous approximation vectors $s_i$ (or $t_i$) converge geometrically, i.e., as $\kappa^l$ with $\kappa < 1$. The convergence factor $\kappa$ can be close to one, but is away from one in situations when there is a good separation of the largest singular values of $A$. In such situations which are often satisfied by practical applications, $s_i$ and $t_i$ can converge very fast to $Ab$ in the major left singular directions of $A$.

### B. Computational complexities

Compared with the truncated SVD, the two Lanczos approximation schemes are far more economical both in memory usage and in time consumption (for the preprocessing phase). This section analyzes the computational complexities of these algorithms and the results are summarized in Table I. We assume that the matrix $A$ is of size $m \times n$ with $nnz$ non-zero elements. The situation when $A$ is non-sparse is not typical, thus we omit the analysis for this case here.[1] Section IV-D has further discussions on the trade-offs between space and time in practical implementations.

In practice, both Algorithms 2 and 3 are split into two phases: preprocessing and query response. The preprocessing phase consists of the computation of the Lanczos vectors, which is exactly line 1 of both algorithms. The rest of the algorithms computes $s_k$ (or $t_k$) by taking in a vector $b$. Multiple query vectors $b$ can be input and different query responses $s_k$ (or $t_k$) can be easily produced provided that the Lanczos vectors have been computed and saved. Hence the rest of the algorithms, excluding line 1, constitutes the query response phase.

Since Algorithms 2 and 3 are very similar, it suffices to show the analysis of Algorithm 2. In the preprocessing phase, this algorithm computes two sparse mat-vec products (once for multiplying $q_i$ by $A^T$, and once for multiplying this result by $A$) and performs a few length-$m$ vector operations during each iteration. Hence with $k$ iterations, the time cost is $O(k(nnz+m))$. The space cost consists of the storage of the matrix $A$ and all the Lanczos vectors, thus it is $O(nnz + km)$.

The bulk of the preprocessing phase for computing $A_k b$ lies in the computation of the truncated SVD of $A$. A typical way of computing this decomposition is to go through the Lanczos process as in (5) with $k' > k$ iterations, and then compute the eigen-elements of $T_{k'}$. Usually the number of Lanczos steps $k'$ is far greater than $k$ in order to guarantee the convergence of the first $k$ Ritz values and vectors. Hence the costs for the truncated SVD, both in time and space, involve a value of $k'$ that is much larger than $k$. Furthermore, the costs have an additional factor for performing frequent eigen-decompositions and convergence tests.

For Algorithm 2, the query response phase involves one sparse mat-vec ($Ab$) and $k$ vector updates. Hence the total storage cost is $O(nnz + km)$, and to be more exact, the time cost is $nnz + 2km$. On the other hand, the filtered mat-vec product $A_k b$ can be computed in three ways: (a) $U_k(U_k^T(Ab))$; (b) $A(V_k(V_k^T b))$; or (c) $U_k(\Sigma_k(V_k^T b))$. Approach (a) requires $O(nnz + km)$ space and $nnz + 2km$ time, approach (b) requires $O(nnz + kn)$ space and $nnz + 2kn$ time, while approach (c) requires $O(k(m+n))$ space and $k(m+n)$ time. Depending on the relative size of $k$ and $nnz/n$ (or $nnz/m$), the best way can be chosen in actual implementations.

To conclude, the two Lanczos approximation schemes consume significantly fewer computing resources than the truncated SVD in the preprocessing phase, and they can be as economical in the query response phase. As a remark, due to the split of the algorithms into these two phases, the storage of the $k$ Lanczos vectors $\{q_i\}$ (or $\{\bar{q}_i\}$) is an issue to consider. These vectors have to be stored permanently after preprocessing, since they are needed to produce the query responses. As will be discussed later

[1] Roughly speaking, if $nnz$ is replaced by $m \cdot n$ in the big-$O$ notations we will obtain the complexities for the non-sparse case. This is true for the algorithms in this paper.

TABLE I
COMPARISONS OF COMPUTATIONAL COSTS.

| | Left approx. | Right approx. | Trunc. SVD[a] |
|---|---|---|---|
| preprocessing space | $O(nnz+km)$ | $O(nnz+kn)$ | $O(nnz+k'm+T_s)$ or $O(nnz+k'n+T_s)$ |
| preprocessing time | $O(k(nnz+m))$ | $O(k(nnz+n))$ | $O(k'(nnz+m)+T_t)$ or $O(k'(nnz+n)+T_t)$ |
| query space | $O(nnz+km)$ | $O(nnz+kn)$ | $O(nnz+km)$ or $O(nnz+kn)$ or $O(k(m+n))$ |
| query time | $nnz+2km$ | $nnz+2kn$ | $nnz+2km$ or $nnz+2kn$ or $k(m+n)$ |

[a] $T_s$ (resp. $T_t$) is the space (resp. time) cost for eigen decompositions of tridiagonal matrices and convergence tests. These two costs are both complicated and cannot be neatly expressed using factors such as $nnz$, $k'$, $m$ and $n$. Also, note that $k' > k$ and empirically $k'$ is a few times larger than $k$.

in Section IV-D, computational time cost can be traded for the storage of these $k$ vectors.

### C. Which approximation to use?

As the above analysis indicates, the choice of which approximation scheme to use largely depends on the relative size of $m$ and $n$, i.e., on the shape of the matrix $A$. It is clear that $s_k$ is a better choice when $m < n$, while $t_k$ is more appropriate when $m \geq n$. The approximation qualities for both schemes are very close; see the experimental results in Section VII-A.

### D. Other implementations

Trade-offs exist between computational efficiency and storage requirement which will lead to different implementations of Algorithms 2 and 3. These choices depend on the practical situation at hand.

*a) Avoiding the storage of the Lanczos vectors:* When $m$ or $n$ is large, the storage requirement of the $k$ Lanczos vectors can not be overlooked. This storage can be avoided by simply regenerating the Lanczos vectors on the fly as efficiently as possible when they are needed. A common practice is that, after preprocessing, only the initial vector $q_1$ and the tridiagonal matrix $T_k$ are stored. At the query response phase, the $q_i$'s are regenerated by applying the Lanczos procedure again, without performing lines 4 and 6 (of Algorithm 1). This way of proceeding requires additional sparse mat-vecs and saxpies but not inner products. The storage is significantly reduced from $k$ vectors to only 3 vectors ($q_1$, and the main- and off-diagonal of $T_k$). Note that since re-orthogonalization is usually performed the storage of the $q_i$'s is needed at least during the preprocessing phase as it is not practical to re-recompute the $q_i$'s for the purpose of re-orthogonalization.

*b) Avoiding the storage of the matrix:* The matrix $A$ occurs throughout the algorithms. The presence of $A$ in the query response phase may be undesirable in some situations for one or more of the following reasons: (a) $A$ is not sparse, so storing a full matrix is very costly; (b) mat-vec operations with a full matrix are expensive; (c) one mat-vec operation may not be as fast as a few vector operations (even in the sparse mode). It is possible to avoid storing $A$ after the preprocessing phase. Specifically, in the left projection approximation, the update formula (7) can be written in the following form:

$$s_i = s_{i-1} + \left\langle b, A^T q_i \right\rangle q_i. \tag{13}$$

Instead of storing $A$, we can store the $k$ intermediate vectors $\{A^T q_i\}$, which are byproducts of the preprocessing phase. This will result in a query response without any mat-vec operations. Similarly, in the right projection approximation, update formula (10) can be rewritten as:

$$t_i = t_{i-1} + \langle b, \bar{q}_i \rangle \, (A\bar{q}_i). \tag{14}$$

The vectors $\{A\bar{q}_i\}$, byproducts of the preprocessing phase, are stored in place of $A$.

In summary, depending on the relative significance of space and time in practical situations, the two approximation schemes can be flexibly implemented in three different ways. From low storage requirement to high storage requirement (or from high tolerance in time to low tolerance in time), they are: (a) Storing only the initial vector $q_1$ (or $\bar{q}_1$) and regenerating the Lanczos vectors in the query response phase; (b) the standard Algorithm 2 (or Algorithm 3); (c) storing two sets of vectors $\{q_i\}$ and $\{A^T q_i\}$ (or $\{\bar{q}_i\}$ and $\{A\bar{q}_i\}$) and discarding the matrix $A$.[2] In all situations these alternatives are far more efficient than those based on the truncated SVD.

## V. ENTRY SCALING

As indicated in the discussions of the two sample applications in Section I, it is usually necessary to scale the entries of the filtered mat-vec product. This amounts to dividing each entry of the vector $A_k b$ by the norm of the corresponding row of $A_k$. In parallel, in the proposed Lanczos approximations, we may need to divide each entry of the approximation vector $s_k = Q_k Q_k^T Ab$ (resp. $t_k = A\bar{Q}_k \bar{Q}_k^T b$) by the norm of the corresponding row of $Q_k Q_k^T A$ (resp. $A\bar{Q}_k \bar{Q}_k^T$). This section illustrates a way to compute these row norms in a minimal cost. Similar to the computation of $s_k$ and $t_k$, the row norms are computed in an iterative update fashion.

For the left projection approximation, define $\eta_j^{(i)}$ to be the norm of the $j$-th row of $Q_i Q_i^T A$, i.e.,

$$\eta_j^{(i)} = \left\| e_j^T Q_i Q_i^T A \right\|. \tag{15}$$

It is easy to see that $\eta_j^{(i)}$ is related to $\eta_j^{(i-1)}$ by

$$\begin{aligned}
\left(\eta_j^{(i)}\right)^2 &= e_j^T Q_i Q_i^T A A^T Q_i Q_i^T e_j \\
&= e_j^T Q_i T_i Q_i^T e_j \\
&= e_j^T (Q_{i-1} T_{i-1} Q_{i-1}^T + \alpha_i q_i q_i^T \\
&\quad + \beta_i q_i q_{i-1}^T + \beta_i q_{i-1}^T q_i) e_j \\
&= \left(\eta_j^{(i-1)}\right)^2 + \alpha_i (q_i)_j^2 + 2\beta_i (q_i)_j (q_{i-1})_j,
\end{aligned} \tag{16}$$

[2] Depending on the relative size of $k$ and $nnz/n$ (or $nnz/m$), method (c) may or may not need more storage than (b). In any case, method (c) should run faster than (b).

where $(v)_j$ means the $j$-th element of vector $v$. Hence the row norms of $Q_k Q_k^T A$, $\eta_j^{(k)}$, $j = 1, \ldots, m$, can be computed using the update formula (16) in the preprocessing phase. Specifically, the following line of pseudocode should be added immediately after line 4 of Algorithm 1:

$$\left(\eta_j^{(i)}\right)^2 \leftarrow \left(\eta_j^{(i-1)}\right)^2 + \alpha_i (q_i)_j^2 + 2\beta_i (q_i)_j (q_{i-1})_j.$$

For the right projection approximation, define $\bar{\eta}_j^{(i)}$ to be the norm of the $j$-th row of $A\bar{Q}_i\bar{Q}_i^T$, i.e.,

$$\bar{\eta}_j^{(i)} = \left\| e_j^T A \bar{Q}_i \bar{Q}_i^T \right\|. \tag{17}$$

It is easy to see that $\bar{\eta}_j^{(i)}$ is related to $\bar{\eta}_j^{(i-1)}$ by

$$\begin{aligned} \left(\bar{\eta}_j^{(i)}\right)^2 &= e_j^T A \bar{Q}_i \bar{Q}_i^T A^T e_j \\ &= e_j^T (A \bar{Q}_{i-1} \bar{Q}_{i-1}^T A^T + A\bar{q}_i \bar{q}_i^T A^T) e_j \\ &= \left(\bar{\eta}_j^{(i-1)}\right)^2 + (A\bar{q}_i)_j^2. \end{aligned} \tag{18}$$

This update formula helps efficiently compute the row norms of $A\bar{Q}_k\bar{Q}_k^T$, that is, $\bar{\eta}_j^{(k)}$, $j = 1, \ldots, m$, in the preprocessing phase. The specific modification to Algorithm 1 is the insertion of the following line of pseudocode immediately after line 3:

$$\left(\bar{\eta}_j^{(i)}\right)^2 \leftarrow \left(\bar{\eta}_j^{(i-1)}\right)^2 + (A\bar{q}_i)_j^2.$$

The computation of row norms $\eta_j^{(k)}$'s (resp. $\bar{\eta}_j^{(k)}$'s) using update formula (16) (resp. (18)) requires minimal resources and will not increase the asymptotic space/time complexities of the preprocessing phase in Algorithm 2 (resp. Algorithm 3).

## VI. Applications

We have briefly discussed two sample applications—LSI and eigenfaces—in Section I. For the sake of completeness, we summarize in this section how the Lanczos approximation with entry scaling can be applied in these two applications as an effective alternative to the traditional approach based on the truncated SVD.

The traditional SVD approach computes the filtered mat-vec product $A_k b$ and divides each entry of this vector by the norm of the corresponding row of $A_k$ to obtain the ranking scores. In LSI, $A_k$ is the transpose of the best rank-$k$ approximation of the term-document matrix, i.e., $X_k^T$, and $b$ is a query $q$. In eigenfaces, $A_k$ is the transpose of the best rank-$k$ approximation of the training set matrix, i.e., $F_k^T$, and $b$ is a test image $p$.

In the proposed Lanczos approximation, we compute $s_k$ or $t_k$ in place of $A_k b$, depending on the relative size of the two dimensions of the matrix $A$. If $m < n$, we compute $s_k$ as an alternative to $A_k b$ using Algorithm 2. Meanwhile we compute the row norms $\eta_j^{(k)}$ for $j = 1, \ldots, m$. Then we divide the $j$-th entry of $s_k$ by $\eta_j^{(k)}$ for all $j$ to obtain the ranking scores. On the other hand, if $m \geq n$, we compute $t_k$ as an alternative to $A_k b$ using Algorithm 3 and also compute the scalars $\bar{\eta}_j^{(k)}$ for $j = 1, \ldots, m$. Then we divide the $j$-th entry of $t_k$ by $\bar{\eta}_j^{(k)}$ for all $j$ hence the similarity scores are obtained.

## VII. Experimental Results

The goal of the extensive experiments to be presented in this section is to show the effectiveness of the Lanczos approximation schemes in comparison to the truncated SVD-based approach. Most of the experiments were performed in Matlab 7 under a Linux workstation with two P4 3.00GHz CPUs and 4GB memory. The only exception was the experiments on the large data set TREC, where a machine with 16GB of memory was used.

### A. Approximation quality

This section shows the convergence behavior of the sequences $\{s_i\}$ and $\{t_i\}$, and the qualities of $s_k$ and $t_k$ in approximating the filtered mat-vec product $A_k b$. The matrices used for experiments were from the data sets MED, CRAN, and NPL, which will be introduced in the next section. We respectively label the three matrices as $A_{\text{MED}}$, $A_{\text{CRAN}}$, and $A_{\text{NPL}}$.

Figure 1 shows the convergence behavior of $\{s_i\}$ and $\{t_i\}$ in the major left singular directions $u_1$, $u_2$ and $u_3$ of the matrix $A_{\text{MED}}$. The vertical axis corresponds to residuals $\left|\langle Ab - s_i, u_j \rangle\right|$ (or $\left|\langle Ab - t_i, u_j \rangle\right|$). As shown by (a) and (c), if no re-orthogonalization was performed, the $\{s_i\}$ and $\{t_i\}$ sequences diverged starting at around the 10-th iteration, which indicated that loss of orthogonality in the Lanczos procedure appeared fairly soon. Hence it is necessary that re-orthogonalization be performed in order to yield accurate results. Plots (b) and (d) show the expected convergence pattern: the residuals rapidly fell towards the level of machine epsilon ($\approx 10^{-16}$).

To show convergence results on more matrices, we plot in Figure 2 the residual curves for $A_{\text{CRAN}}$ and $A_{\text{NPL}}$. For all cases re-orthogonalization was performed. Note that the shapes of the two matrices are different: $A_{\text{CRAN}}$ has more columns than rows while for $A_{\text{NPL}}$ the opposite is true. Figure 2 implies two facts: (a) The approximation sequences converged in all three directions, with $u_1$ being the fastest direction; (b) the approximations did not differ too much in terms of rate of convergence or approximation quality. The first fact can be further investigated by plotting the eigenvalue distribution of the matrices $M = AA^T$ as in Figure 3. The large gap between the first two eigenvalues explains the quick convergence in the $u_1$ direction. The second fact is guaranteed by the similar residual bounds (c.f. inequalities (11) and (12)). It further shows that approximation quality is not a factor in the choice of left or right projection.

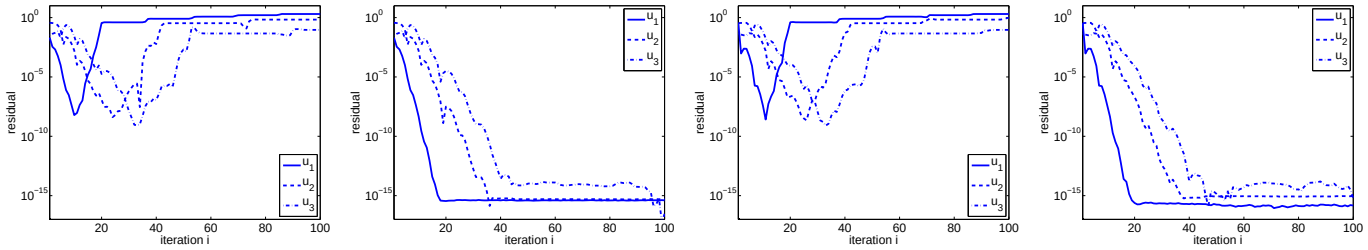
For $k = 300$, we plot in Figure 4 the residuals $\left|\langle A_{300} b - s_{300}, u_j \rangle\right|$ (or $\left|\langle A_{300} b - t_{300}, u_j \rangle\right|$) over the $j$-th left singular directions for $j = 1, \ldots, 300$. All the plots exhibit a similar pattern: $A_{300} b$ and $s_{300}$ (or $t_{300}$) were almost identical when projected onto the first 100 singular directions. This indicates that $s_k$ and $t_k$ can be good alternatives to $A_k b$, since they preserve the quality of $A_k b$ in a large number of the major singular directions.

### B. LSI for information retrieval

*1) Data sets:* Four data sets were used in the experiments. Statistics are shown in Table II.

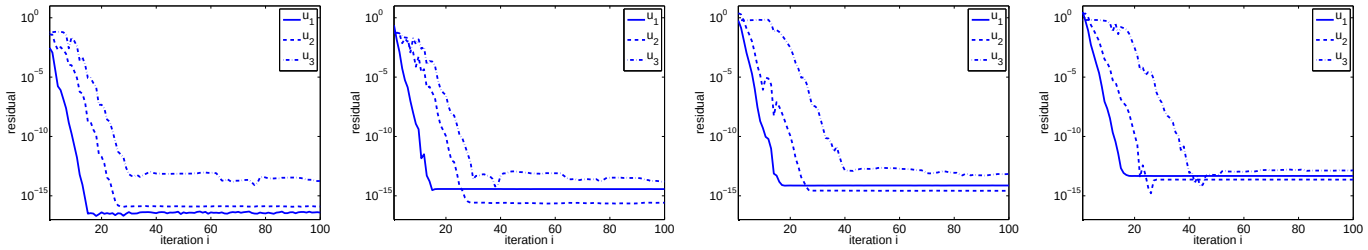
*a) MEDLINE and CRANFIELD*[3]*:* These are two early and well-known benchmark data sets for information retrieval. Their typical statistics is that the number of distinct terms is more than the number of documents, i.e., for the matrix $A$ (or equivalently $X^T$), $m < n$.

[3] `ftp://ftp.cs.cornell.edu/pub/smart/`



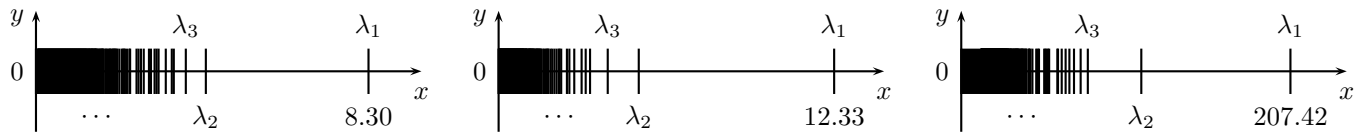

(a) Left proj., w/o re-orth. (b) Left proj., with re-orth. (c) Right proj., w/o re-orth. (d) Right proj., with re-orth.

Fig. 1. Convergence of the $\{s_i\}$ (resp. $\{t_i\}$) sequence to the vector $Ab$ in the major singular directions $u_1$, $u_2$ and $u_3$. Matrix: $A_{\text{MED}}$.



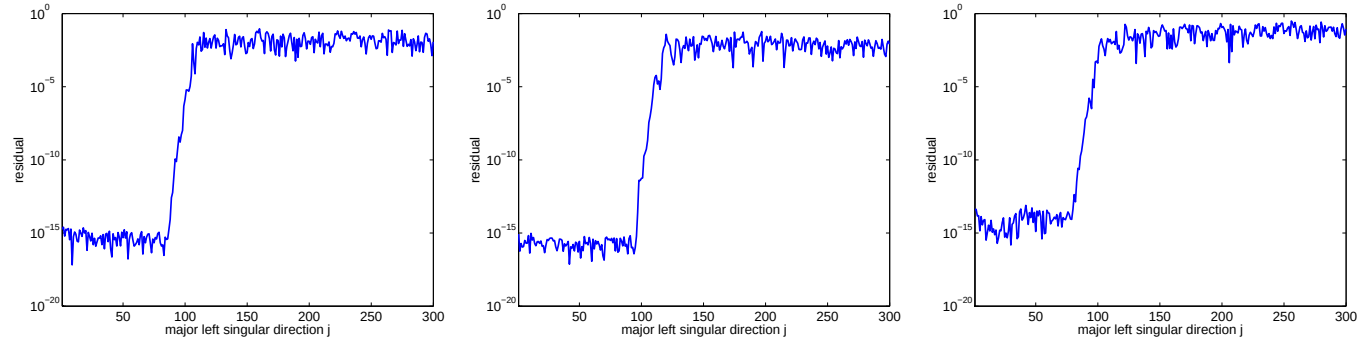

(a) $A_{\text{CRAN}}$: left proj. (b) $A_{\text{CRAN}}$: right proj. (c) $A_{\text{NPL}}$: left proj. (d) $A_{\text{NPL}}$: right proj.

Fig. 2. Convergence of the $\{s_i\}$ (resp. $\{t_i\}$) sequence to the vector $Ab$ in the major singular directions $u_1$, $u_2$ and $u_3$. Matrices: $A_{\text{CRAN}}$, $A_{\text{NPL}}$. All with re-orthogonalization.




(a) $A_{\text{MED}}$. (b) $A_{\text{CRAN}}$. (c) $A_{\text{NPL}}$.

Fig. 3. Eigenvalue distribution of $M = AA^T$ on the real axis, where $A^T$ is the term-document matrix.




(a) $A_{\text{MED}}$: left proj. (b) $A_{\text{CRAN}}$: left proj. (c) $A_{\text{NPL}}$: right proj.

Fig. 4. Differences between $A_{300}b$ and $s_{300}$ (or $t_{300}$) in the major left singular directions. For the first one hundred directions, the differences have decreased to almost zero, while for other less significant directions the differences have not dropped to this level.

*b) NPL*[3]*:* This data set is larger than the previous two, with a property that the number of documents is larger than the number of distinct terms, i.e., $m > n$. Including the above two, these three data sets have the term-document matrices readily available from the provided links, so we did not perform additional processing on the data.

*c) TREC*[4]*:* This is a large data set which is popularly used for experiments in serious text mining applications. Similar to NPL, the term-document matrix extracted from this data set has more documents than distinct terms, i.e., $m > n$. Specifically, the whole data set consists of four document collections (*Financial Times*, *Federal Register*, *Foreign Broadcast Information Service*, and *Los Angeles Times*) from the TREC CDs 4 & 5 (copyrighted). The queries are from the *TREC-8 ad hoc* task[5][6]. We used the software TMG [21] to construct the term-document matrix. The parsing process included stemming, deleting common words according to the stop-list provided by the software, and removing

[4]`http://trec.nist.gov/data/docs_eng.html`

[5]Queries: `http://trec.nist.gov/data/topics_eng/`

[6]Relevance: `http://trec.nist.gov/data/qrels_eng/`

words with no more than 5 occurrences or with appearances in more than 100,000 documents. Also, 125 empty documents were ignored. This resulted in a term-document matrix of size $138232 \times 528030$. For the queries, only the title and description parts were extracted to construct query vectors.

TABLE II
INFORMATION RETRIEVAL DATA SETS.

| | MED | CRAN | NPL | TREC |
|---|---|---|---|---|
| # terms | 7,014 | 3,763 | 7,491 | 138,232 |
| # docs | 1,033 | 1,398 | 11,429 | 528,030 |
| # queries | 30 | 225 | 93 | 50 |
| ave terms/doc | 52 | 53 | 20 | 129 |

*2) Implementation specs:* The weighting scheme of all the term-document matrices was *term frequency-inverse document frequency* (tf-idf). Due to the shapes of the matrices, for MEDLINE and CRANFIELD we used the left projection approximation, and for NPL and TREC we used the right projection approximation. Full re-orthogonalization in the Lanczos process was performed. The filtered mat-vec product $A_k b$ was computed as $U_k(U_k^T(Ab))$ or $A(V_k(V_k^T b))$, depending on the relative sizes of $m$ and $n$. We implemented the partial SVD based on the Lanczos algorithm and Ritz values/vectors [22, Section 3.1]. We did not use the Matlab command `svds` for two reasons. (a) The Matlab function `svds` uses a different algorithm. In order to conform to the computational complexity analysis in Section IV-B, we used the algorithm mentioned in the analysis. (b) While the algorithm used by the Matlab function `svds` is able to accurately compute the smallest singular components, it consumes too many resources in both storage and time. We chose to implement one that was both lightweight and accurate for largest singular components, since the smallest components were not our concern.

*3) Results:* Figure 5 plots the performance of the Lanczos approximation applied to the MEDLINE, CRANFIELD and NPL data sets, compared with that of the standard LSI (using the truncated SVD approach). These plots show that the accuracy obtained from the Lanczos approximation is comparable to that of LSI,[7] while in preprocessing the former runs an order of magnitude faster than the latter. The accuracy is measured using the 11-point interpolated average precision, as shown in (a), (b) and (c). More information is provided by plotting the precision-recall curves. These plots are shown using a specific $k$ for each data set in (d), (e) and (f). They suggest that the retrieval accuracy for the Lanczos approximation is very close to that of LSI, sometimes even better. As shown in (g), (h) and (i), the Lanczos approximation is far more economical than the SVD-based LSI. It can be an order of magnitude faster than the truncated SVD which, it should be recalled, was re-implemented for better efficiency. The average query times for the two approaches, as shown in plots (j), (k) and (l), are (almost) identical, which conforms to the previous theoretical analysis. Note the efficiency of the query responses—in several or several tens of milliseconds. Plot (l) does show small discrepancies in query times between the two approaches. We believe that the differences were caused by the internal Matlab memory management schemes that were unknown to us. If in the experiments the Lanczos vectors ($\bar{Q}_k$) and the singular vectors ($V_k$) had been precomputed and reloaded rather than being computed on the fly, the query times would have been identical for both approaches.

We also performed tests on the large data set TREC. Since we had no access to a fair environment (which requires as much as 16GB working memory and no interruptions from other users) for time comparisons, we tested only the accuracy. Figure 6 shows the precision-recall curves at different $k$'s (400, 600 and 800) for the two approaches. As illustrated by the figure, the Lanczos approximation yielded much higher accuracy than the truncated SVD approach. Meanwhile, according to the rigorous analysis in Section IV-B, the former approach should consume much fewer computing resources and be more efficient than the latter.

### C. Eigenfaces for face recognition

*1) Data sets:* Three data sets were used in the experiments. Statistics are shown in Table III.

*a) ORL*[8]*:* The ORL data set [23] is small but classic. A dark homogeneous background was used at the image acquisition process. We did not perform any image processing task on this data set.

*b) PIE*[9]*:* The PIE data set [24] consists of a large set of images taken under different poses, illumination conditions and face expressions. We downloaded the cropped version from Deng Cai's homepage[10]. This version of the data set contains only five near frontal poses for each subject. The images had been non-uniformly scaled to a size of $32 \times 32$.

*c) ExtYaleB*[11]*:* Extended from a previous database, the extended Yale Face Database B (ExtYaleB) [25] contains images for 38 human subjects. We used the cropped version [26] that can be downloaded from the database homepage. We further uniformly resized the cropped images to half of their sizes. The resizing can be conveniently performed in Matlab by using the command

```
img_small = imresize(img, .5, 'bicubic')
```

without compromising the quality of the images.

TABLE III
FACE RECOGNITION DATA SETS.

| | ORL | PIE | ExtYaleB |
|---|---|---|---|
| # subjects | 40 | 68 | 38 |
| # imgs | 400 | 11,554 | 2,414 |
| img size | $92 \times 112$ | $32 \times 32$ | $84 \times 96$ |

*2) Implementation specs:* In the eigenfaces technique, each image is vectorized to a column, whose elements are pixel values. All the images have a single gray channel, whose values range from 0 to 255. We divided them by 255 such that all the pixel values were no greater than 1. We split a data set into training subset and test subset by randomly choosing a specific number of images for each subject as the training images. Different numbers were experimented with and this will be discussed shortly. As in the previous information retrieval experiments, the choice of left or right projection depended on the shape of the training

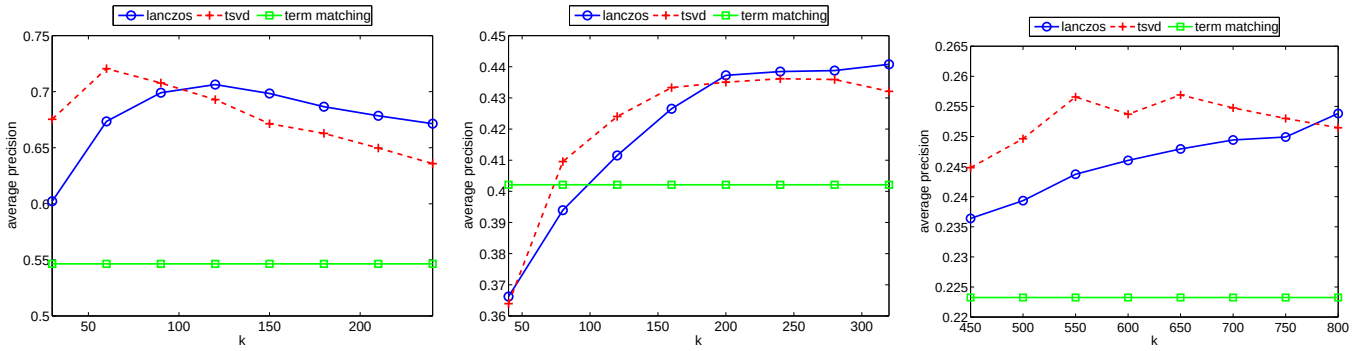
[7]Note that by "comparable", we mean that the accuracies obtained from the two methods do not greatly differ. It is not necessary that the difference between the accuracies is not statistically significant.

[8]http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

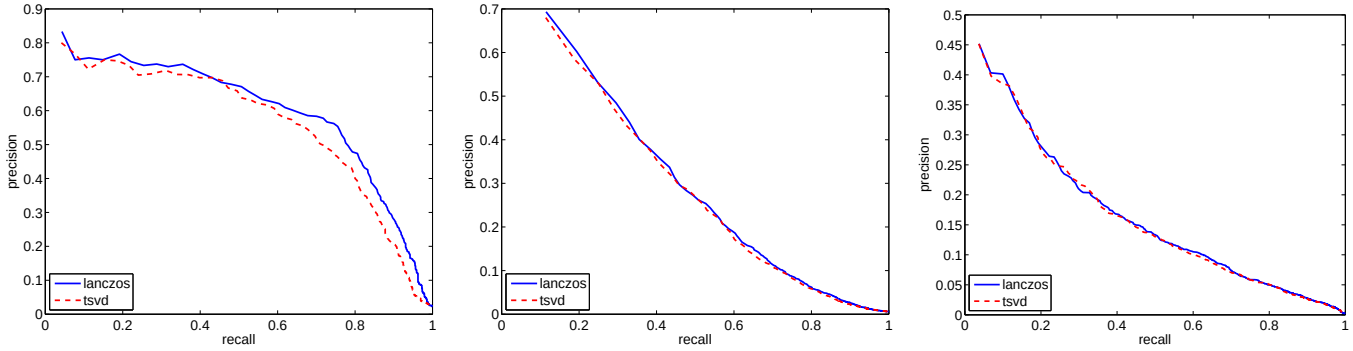
[9]http://www.ri.cmu.edu/projects/project_418.html

[10]http://www.cs.uiuc.edu/homes/dengcai2/Data/FaceData.html

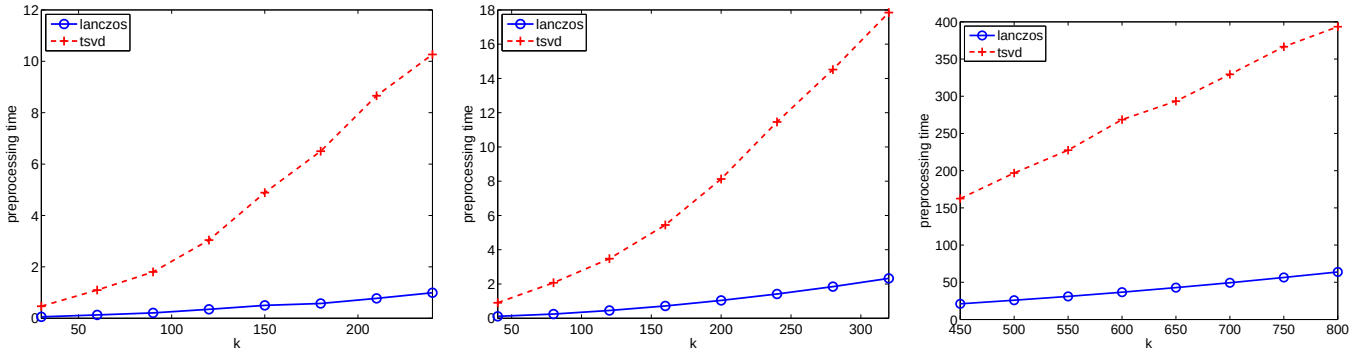
[11]http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html



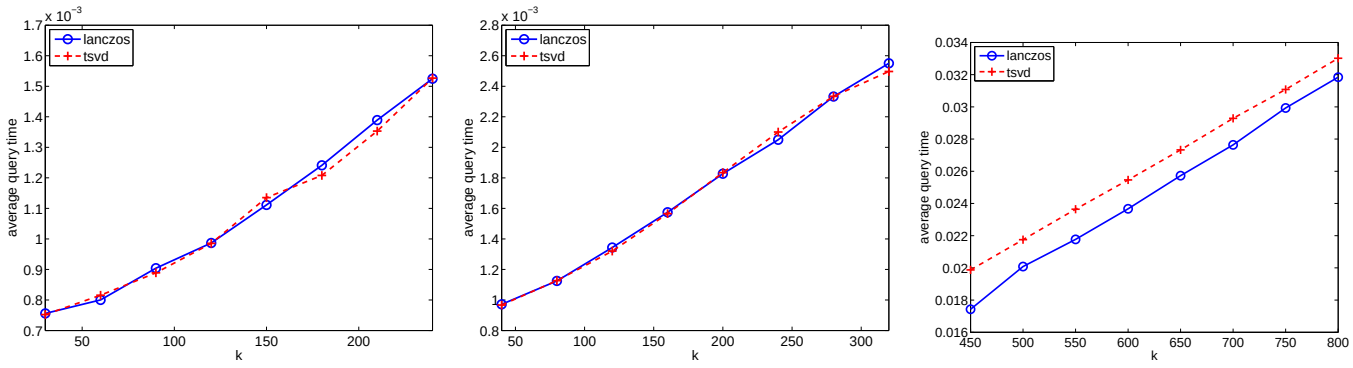

(a) MED: average precision. (b) CRAN: average precision. (c) NPL: average precision.




(d) MED: precision-recall ($k = 240$). (e) CRAN: precision-recall ($k = 320$). (f) NPL: precision-recall ($k = 800$).

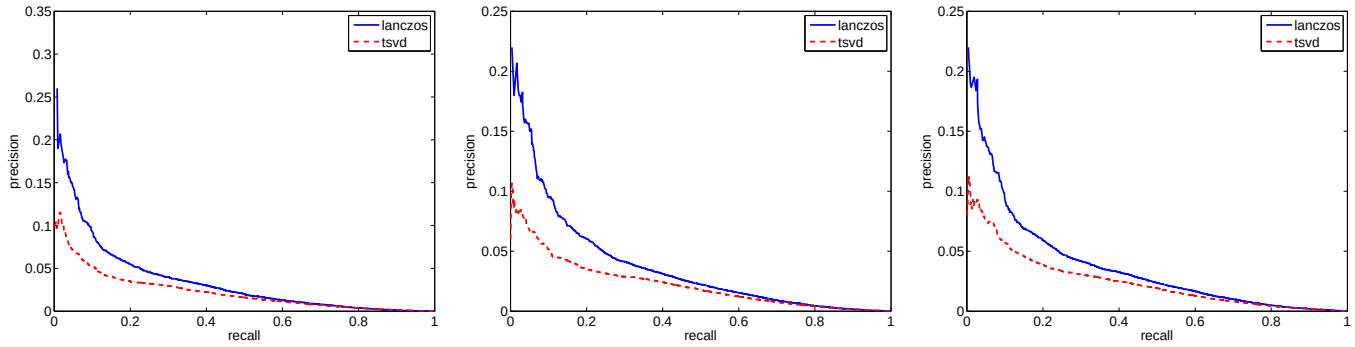



(g) MED: preprocessing time (seconds). (h) CRAN: preprocessing time (seconds). (i) NPL: preprocessing time (seconds).




(j) MED: average query time (seconds). (k) CRAN: average query time (seconds). (l) NPL: average query time (seconds).

Fig. 5. (Information retrieval) Performance tests on MEDLINE, CRANFIELD and NPL.




(a) $k = 400$. (b) $k = 600$. (c) $k = 800$.

Fig. 6. (Information retrieval) Performance tests on TREC: precision-recall curves at different $k$'s.

set matrix, and full re-orthogonalization was performed for the Lanczos approximation approach. In the truncated SVD approach we used the same SVD function as the one implemented for information retrieval.

*3) Results:* Error rates of the two approaches are plotted in Figure 7. For each data set, we used different numbers of images for training. The training sets are roughly $40\%$, $60\%$ and $80\%$ of the whole data set. As shown by the figure, for all the data sets and all the training sizes we experimented with, the Lanczos approximation yielded accuracy that is very close to that of the truncated SVD. These results confirm that the Lanczos approach can be a good alternative to the traditional eigenfaces method.

The advantage of the Lanczos approximation lies in its efficiency. Figure 8 plots the preprocessing times and query times of the two approaches. As can be seen in the figure, in the preprocessing phase the Lanczos approach is faster than the truncated SVD approach by a factor of 3 or more. However this gain in efficiency is not as large as that seen in the information retrieval experiments. This is due to the fact that the matrix is not sparse in this case. The dense mat-vec multiplication occupies a larger fraction of the computing time. Nevertheless, the Lanczos approximation approach always outperforms the traditional truncated SVD approach in time.

## VIII. Conclusions

We have explored the use of the Lanczos algorithm as a systematic means of computing filtered mat-vec products. The goal of the method is to obtain a sequence of vectors which converges rapidly towards the exact product in the major left singular directions of the matrix. The main attraction of the algorithm is its low cost. Extensive experiments show that the proposed method can be an order of magnitude faster than the standard truncated SVD approach. In addition, the quality of the approximation in practical applications, such as information retrieval and face recognition, is comparable to that obtained from the traditional approach. Thus, the proposed technique can be applied as a replacement to the SVD technique in any application where a major computational task is to compute a filtered mat-vec, i.e., the product of a low-rank approximation of a matrix by an arbitrary vector.

An avenue of future research following this work, is to study how the proposed Lanczos approximation, compared with the truncated SVD approach, affects the final quality for a certain application, in a statistical manner. For information retrieval, the quality of a technique could be based on the precision, i.e., how close the actual relevant documents are to the top of the computed rank list. Although we have proposed a technique to effectively approximate the ranking scores, how the final precisions vary away from those computed using the standard LSI approach is inconclusive. Similarly, for face recognition, the accuracy depends on only the top image in the rank list. It is possible to consider for what conditions the top scores obtained from both the Lanczos technique and the truncated SVD technique appear on the same image, thus the two approaches produce exactly the same result.

## Appendix
## Proof of the Convergence of the Approximation Vectors

Before the proof, we introduce a known result on the rate of convergence of the Lanczos algorithm. Saad [27] provided a bound on the angle between the $j$-th eigenvector $\phi_j$ of $M$ and the Krylov subspace $\mathrm{range}(Q_k)$:

$$\frac{\left\|(I - Q_k Q_k^T)\phi_j\right\|}{\left\|Q_k Q_k^T \phi_j\right\|} \leq \frac{K_j}{T_{k-j}(\gamma_j)} \frac{\left\|(I - Q_1 Q_1^T)\phi_j\right\|}{\left\|Q_1 Q_1^T \phi_j\right\|}, \tag{19}$$

where

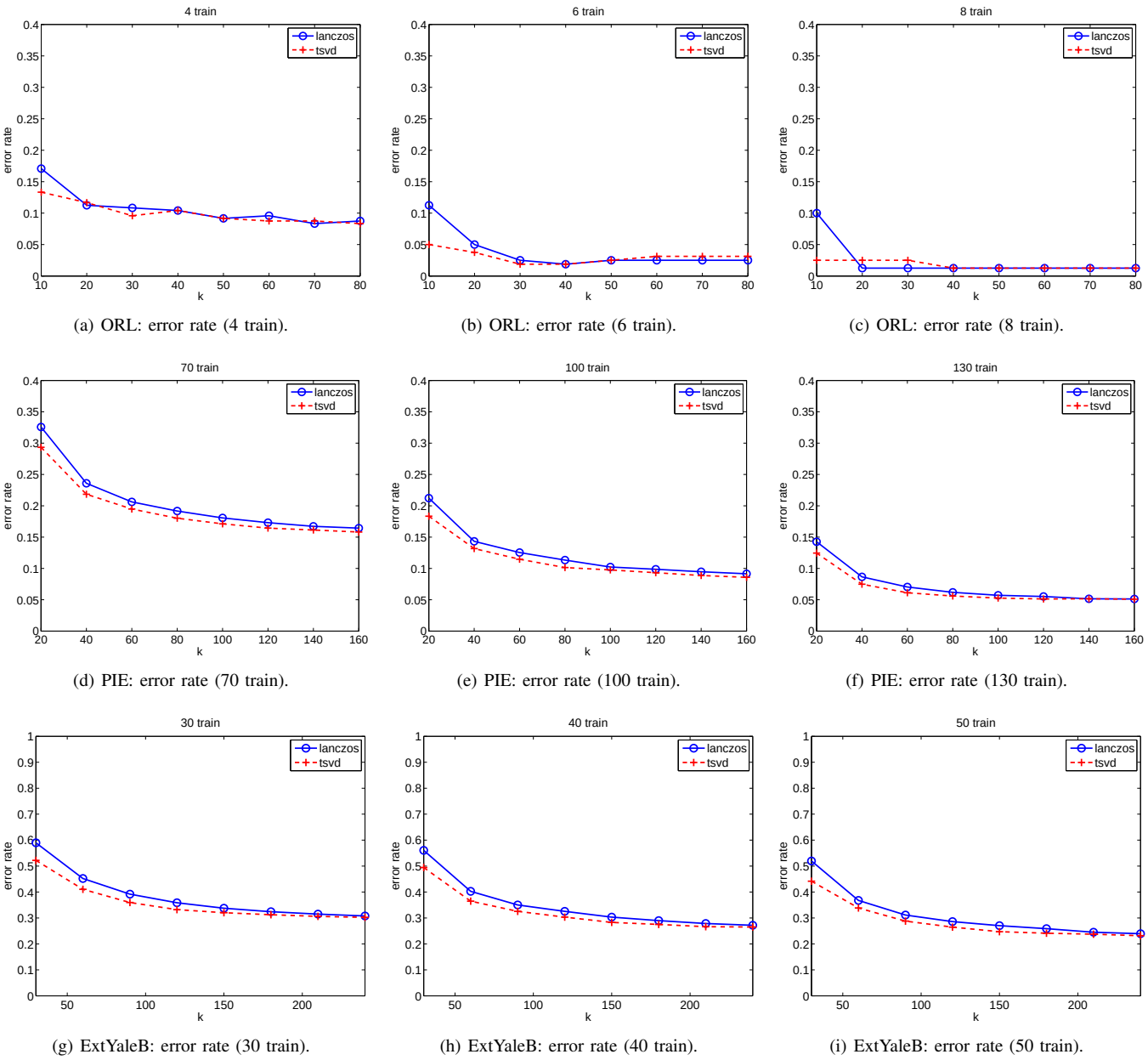
$$\gamma_j = 1 + 2\frac{\lambda_j - \lambda_{j+1}}{\lambda_{j+1} - \lambda_n}, \qquad K_j = \begin{cases} 1 & j = 1 \\ \prod_{i=1}^{j-1} \frac{\lambda_i - \lambda_n}{\lambda_i - \lambda_j} & j \neq 1 \end{cases},$$

$\lambda_j$ is the $j$-th eigenvalue of $M$, and $T_l(x)$ is the Chebyshev polynomial of the first kind of degree $l$. Assuming that $\phi_j$ has unit norm, we simplify the inequality into the following form:

$$\left\|(I - Q_k Q_k^T)\phi_j\right\| \leq c_j T_{k-j}^{-1}(\gamma_j), \tag{20}$$

where $c_j$ is some constant independent of $k$. Inequality (20) indicates that the difference between any unit eigenvector $\phi_j$ and the subspace $\mathrm{span}(Q_i)$ decays at least with the same order as $T_{i-j}^{-1}(\gamma_j)$.

We now analyze the difference between $Ab$ and $s_i$ in the left singular directions of $A$. Recall that $A$ has the SVD $A = U\Sigma V^T$, hence $M = AA^T = U\Sigma\Sigma^T U^T$, which means that the $u_j$'s are




(a) ORL: error rate (4 train). (b) ORL: error rate (6 train). (c) ORL: error rate (8 train).

(d) PIE: error rate (70 train). (e) PIE: error rate (100 train). (f) PIE: error rate (130 train).

(g) ExtYaleB: error rate (30 train). (h) ExtYaleB: error rate (40 train). (i) ExtYaleB: error rate (50 train).

Fig. 7. (Face recognition) Performance tests on ORL, PIE and ExtYaleB: error rate.

eigenvectors of $M$. Then applying (20),

$$\begin{aligned}\left|\left\langle Ab - s_i, u_j\right\rangle\right| &= \left|\left\langle (I - Q_i Q_i^T)Ab, u_j\right\rangle\right| \\ &= \left|\left\langle (I - Q_i Q_i^T)u_j, Ab\right\rangle\right| \\ &\leq \left\|(I - Q_i Q_i^T)u_j\right\| \|Ab\| \\ &\leq c_j \|Ab\| T_{i-j}^{-1}(\gamma_j).\end{aligned}$$

This gives the convergence rate of the approximation vector $s_i$ to the vector $Ab$ along the direction $u_j$.

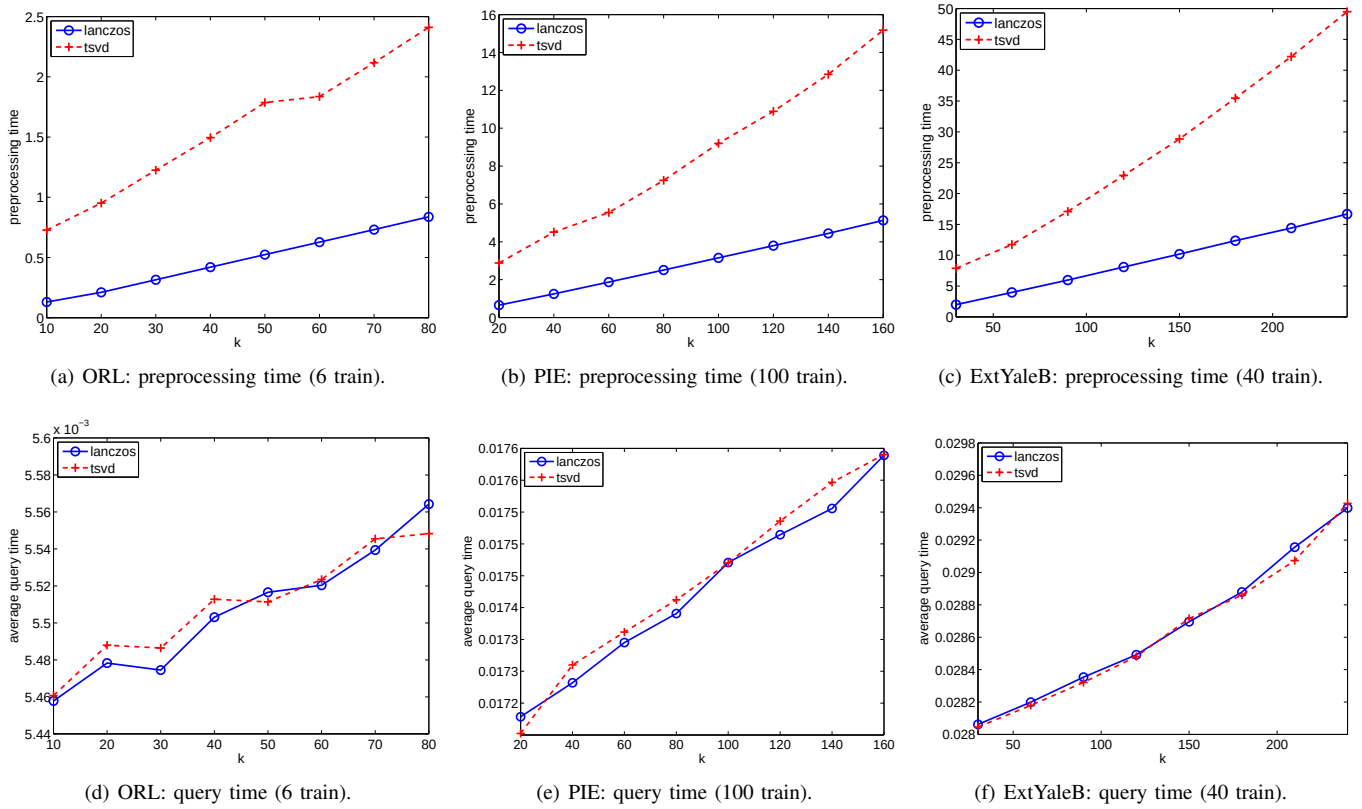
We can similarly give a bound on the difference between $Ab$ and $t_i$ in the direction $u_j$:

$$\begin{aligned}\left|\left\langle Ab - t_i, u_j\right\rangle\right| &= \left|\left\langle A(I - \bar{Q}_i \bar{Q}_i^T)b, u_j\right\rangle\right| \\ &= \left|\left\langle (I - \bar{Q}_i \bar{Q}_i^T)b, A^T u_j\right\rangle\right| \\ &= \sigma_j \left|v_j^T (I - \bar{Q}_i \bar{Q}_i^T)b\right| \\ &\leq \sigma_j \left\|(I - \bar{Q}_i \bar{Q}_i^T)v_j\right\| \|b\|.\end{aligned}$$

Note that $\bar{M} = A^T A = V \Sigma^T \Sigma V^T$, hence $v_j$ is the $j$-th eigenvector of $\bar{M}$. So we can also bound the term $\left\|(I - \bar{Q}_i \bar{Q}_i^T)v_j\right\|$ by Chebyshev polynomial, yielding the result

$$\left|\left\langle Ab - t_i, u_j\right\rangle\right| \leq \sigma_j \bar{c}_j \|b\| T_{i-j}^{-1}(\bar{\gamma}_j).$$

This provides the convergence rate of the approximation vector $t_i$ to the vector $Ab$ in the direction $u_j$.




(a) ORL: preprocessing time (6 train). (b) PIE: preprocessing time (100 train). (c) ExtYaleB: preprocessing time (40 train).

(d) ORL: query time (6 train). (e) PIE: query time (100 train). (f) ExtYaleB: query time (40 train).

Fig. 8. (Face recognition) Performance tests on ORL, PIE and ExtYaleB: time (in seconds).

## REFERENCES


[1] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins, 1996.

[3] S. C. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. A. Harshman, "Indexing by latent semantic analysis," *J. Am. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, 1990.

[4] M. W. Berry and M. Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. SIAM, June 1999.

[5] M. Turk and A. Pentland, "Face recognition using eigenfaces," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1991, pp. 586–591.

[6] D. I. Witter and M. W. Berry, "Downdating the latent semantic indexing model for conceptual information retrieval," *The Computer J.*, vol. 41, no. 8, pp. 589–601, 1998.

[7] H. Zha and H. D. Simon, "On updating problems in latent semantic indexing," *SIAM J. Sci. Comput.*, vol. 21, no. 2, pp. 782–791, 1999.

[8] M. Brand, "Fast low-rank modifications of the thin singular value decomposition," *Linear Algebra Appl.*, vol. 415, no. 1, pp. 20–30, 2006.

[9] J. E. Tougas and R. J. Spiteri, "Updating the partial singular value decomposition in latent semantic indexing," *Comput Statist. Data Anal.*, vol. 52, no. 1, pp. 174–183, 2007.

[10] E. Kokiopoulou and Y. Saad, "Polynomial filtering in latent semantic indexing for information retrieval," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, 2004, pp. 104–111.

[11] J. Erhel, F. Guyomarc, and Y. Saad, "Least-squares polynomial filters for ill-conditioned linear systems," University of Minnesota Supercomputing Institute, Tech. Rep., 2001.

[12] Y. Saad, "Filtered conjugate residual-type algorithms with applications," *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 845–870, August 2006.

[13] M. W. Berry, "Large scale sparse singular value computations," *International Journal of Supercomputer Applications*, vol. 6, no. 1, pp. 13–49, 1992.

[14] K. Blom and A. Ruhe, "A Krylov subspace method for information retrieval," *SIAM J. Matrix Anal. Appl.*, vol. 26, no. 2, pp. 566–582, 2005.

[15] G. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *SIAM J. Numer. Anal.*, vol. 2, no. 2, pp. 205–224, 1965.

[16] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*. Halstead Press, New York, 1992.

[17] B. N. Parlett, *The symmetric eigenvalue problem*. Prientice-Hall, 1998.

[18] R. M. Larsen, "Efficient algorithms for helioseismic inversion," Ph.D. dissertation, Dept. Computer Science, University of Aarhus, Denmark, October 1998.

[19] H. D. Simon, "Analysis of the symmetric Lanczos algorithm with reorthogonalization methods," *Linear Algebra Appl.*, vol. 61, pp. 101–131, 1984.

[20] ——, "The Lanczos algorithm with partial reorthogonalization," *Mathematics of Computation*, vol. 42, no. 165, pp. 115–142, 1984.

[21] D. Zeimpekis and E. Gallopoulos, *TMG: A MATLAB toolbox for generating term document matrices from text collections*. Springer, Berlin, 2006, pp. 187–210.

[22] M. W. Berry, "Large-scale sparse singular value computations," *International Journal of Supercomputer Applications*, vol. 6, no. 1, pp. 13–49, 1992.

[23] F. Samaria and A. Harter, "Parameterisation of a stochastic model for human face identification," in *2nd IEEE Workshop on Applications of Computer Vision*, 1994.

[24] T. Sim, S. Baker, and M. Bsat, "The CMU pose, illumination, and expression database," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 25, no. 12, pp. 1615–1618, 2003.

[25] A. Georghiades, P. Belhumeur, and D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 23, no. 6, pp. 643–660, 2001.

[26] K. Lee, J. Ho, and D. Kriegman, "Acquiring linear subspaces for face recognition under variable lighting," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 27, no. 5, pp. 684–698, 2005.

[27] Y. Saad, "On the rates of convergence of the Lanczos and the block-Lanczos methods," *SIAM J. Numer. Anal.*, vol. 17, no. 5, pp. 687–706, October 1980.