Abstract

We study $k$-SVD that is to obtain the first $k$ singular vectors of a matrix $A$ approximately. Recently, a few breakthroughs have been discovered on $k$-SVD: Musco and Musco [18] provided the first gap-free theorem for the block Krylov method, Shamir [20] discovered the first variance-reduction stochastic method, and Bhojanapalli et al. [6] provided the fastest $O(\text{nnz}(A) + \text{poly}(1/\varepsilon))$-type of algorithm using alternating minimization.

In this paper, put forward a new framework for SVD and improve the above breakthroughs. We obtain faster gap-free convergence rate outperforming [18], we obtain the first accelerated and stochastic method outperforming [20]. In the $O(\text{nnz}(A) + \text{poly}(1/\varepsilon))$ running-time regime, we outperform [6] in certain parameter regimes without even using alternating minimization.

1 Introduction

The singular value decomposition (SVD) of a rank-$r$ matrix $A \in \mathbb{R}^{d \times n}$ corresponds to decomposing $A = V \Sigma U^\top$ where $V \in \mathbb{R}^{d \times r}$, $U \in \mathbb{R}^{n \times r}$ are two column orthonormal matrices, and $\Sigma = \text{diag}\{\sigma_1, \ldots, \sigma_r\} \in \mathbb{R}^{r \times r}$ is a non-negative diagonal matrix with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. The columns of $V$ (resp. $U$) are called the left (resp. right) singular vectors of $A$ and the diagonal entries of $\Sigma$ are called the singular values of $A$. SVD is one of the most fundamental tools used in machine learning, computer vision, statistics, and operations research, and is essentially equivalent to principal component analysis (PCA) up to column averaging.

A rank $k$ partial SVD (or $k$-SVD for short) is to find the top $k$ left (resp. right) singular vectors of $A$, or equivalently, the first $k$ columns of $V$ (resp. $U$). Denoting by $V_k^\top \in \mathbb{R}^{d \times k}$ the first $k$ columns of $V$, and $U_k$ the first $k$ columns of $U$, one can define $A_k^+ := V_k V_k^\top A = V_k \Sigma_k U_k^\top$ where $\Sigma_k = \text{diag}\{\sigma_1, \ldots, \sigma_k\}$. Under this notation, $A_k^+$ is the the best rank-$k$ approximation of matrix $A$ in terms of minimizing $\|A - A_k\|$ among all rank $k$ matrices $A_k$. Here, the norm can be any Schatten-$q$ norm for $q \in [1, \infty]$, including spectral norm ($q = \infty$) and Frobenius norm ($q = 2$), therefore making $k$-SVD a very powerful tool in terms of information retrieval, data de-noising, or even data compression.

Traditional algorithms to compute SVD essentially run in time $O(nd \min\{d, n\})$, which is usually very expensive for big-data scenarios. As for $k$-SVD, defining gap := $\left(\sigma_k - \sigma_{k+1}\right)/\sigma_k$ to be the relative $k$-th eigengap of matrix $A$, the famous subspace power method or block Krylov method [13] solves $k$-SVD in time $O(\text{gap}^{-1} \cdot k \cdot \text{nnz}(A) \cdot \log(1/\varepsilon))$ or $O(\text{gap}^{-6.5} \cdot k \cdot \text{nnz}(A) \cdot \log(1/\varepsilon))$ respectively if ignoring lower order terms. Here, $\text{nnz}(A)$ is the number of non-zero elements in matrix $A$, and the more precise running times are stated in Table 1.

Recently, there are breakthroughs to compute $k$-SVD faster, from three incomparable perspectives.

---

*The full version of this paper can be found on https://arxiv.org/abs/1607.03463. This paper is partially supported by a Microsoft Research Grant, no. 0518584.

All GF results above provide $(1 + \varepsilon)\|\Delta\|_2$ spectral and $(1 + \varepsilon)\|\Delta\|_F$ Frobenius guarantees.

Table 1: Performance comparison among direct methods. Define $\text{gap} = (\sigma_k - \sigma_{k+1})/\sigma_k \in [0, 1]$. GF = Gap Free; Stoc = Stochastic; Acc = Accelerated. Stochastic results in this table are assuming $\|a_i\|_2 \leq 1$ following (1.1).

The first breakthrough is the work of Musco and Musco [18] for providing running times that do not depend on properties (i.e., the eigengap) of $A$ for block Krylov method. As highlighted in [18], providing gap-free results was an open question for decades and is a more reliable goal for practical purposes. Specifically, they proved that the same block Krylov method converges in time $O\left(\frac{\text{nnz}(A)}{\varepsilon^{1/2}} + \frac{k^2 d}{\varepsilon^2} + \frac{k^3}{\varepsilon^{3/2}}\right)$, where $\varepsilon$ is the multiplicative approximation error.

The second breakthrough is the work of Shamir [20] for providing a fast stochastic $k$-SVD algorithm. In a stochastic setting, we assume

$$A = \frac{1}{n} \sum_{i=1}^{n} a_i a_i^T \quad \text{and each} \quad a_i \in \mathbb{R}^d \quad \text{has Euclidean norm at most} \ 1 . \quad (1.1)$$

Instead of using the entire matrix $AA^T$ when applying (subspace) power method, one can use a rank-1 copy $a_i a_i^T$ where $i$ is chosen uniformly at random. While this stochastic approach improves the per-iteration running time for obvious reason, one needs to carefully introduce ad-hoc variance-reduction techniques in order to make the algorithm suitable for small $\varepsilon$. We state Shamir’s running time in Table 1. Unfortunately, Shamir’s result is (1) not gap-free; (2) not accelerated (i.e., does not match the gap $^{-0.5}$ dependence comparing to block Krylov); and (3) requires a very accurate warm-start that in principle can take a very long time to compute.

The third breakthrough is to obtain running times of the form $O(\text{nnz}(A) + \text{poly}(k, 1/\varepsilon) \cdot (n + d))$ [6, 7], see Table 2. We call them NNZ running times. To obtain such results, one usually needs sub-sampling on the matrix and thus incurs a poor dependence on $\varepsilon$. For this reason, to make the best use of NNZ type of results, one usually focuses on improving the dependence on $1/\varepsilon$. To this end, Bhosle et al. [6] provide a $1/\varepsilon^2$ result using alternating minimization. Since $1/\varepsilon^2$ also shows up in the sampling complexity, we believe the quadratic dependence on $\varepsilon$ is tight among NNZ types of algorithms.

All the cited results rely on ad-hoc non-convex optimization techniques together with matrix algebra, which make their final proofs complicated. Furthermore, Shamir’s result [20] only works if a very accurate (i.e., $1/\text{poly}(d)$-accurate) warm start is given, and the time needed to find such a warm start remains unclear.

---

1. In this paper, we use $\tilde{O}$ notations to hide possible logarithmic factors on $1/\text{gap}$, $1/\varepsilon$, $n$, $d$, $k$ and potentially also on $\sigma_1/\sigma_{k+1}$.

2. This normalization follows the tradition of stochastic $k$-SVD or 1-SVD literatures [11, 19, 20]. If one removes such an assumption, the results of these papers will be complicated by the introduction of non-uniform sampling and the stable rank notations. We refrain from doing so in this paper.
In this paper, we develop a new algorithmic framework to solve $k$-SVD. It not only improves the aforementioned breakthroughs, but also relies only on simple convex analysis so do not require a warm start.

**Other Related Work.** Some authors focus on the streaming model of 1-SVD [14] or $k$-SVD [2]. These algorithms are slower than off-line methods. We also acknowledge that, unlike $k$-SVD, accelerated stochastic methods were previously known for 1-SVD [11] [12]. The ideas of this paper have later been generalized to canonical component analysis and generalized eigendecomposition by the same authors [1]. If one is only interested in projecting a vector to the top $k$-eigenspace without computing the top $k$ eigenvectors like we do in this paper, this can also be done in an accelerated manner [3].

### 1.1 Overview of Our Result and the Settlement of an Open Question

Our algorithmic framework is extremely simple: instead of computing all $k$ singular vectors together like all recent breakthroughs, we find singular vectors one at a time, for $k$ iterations in total.

A naive implementation of this idea results in a running time that depends on the intermediate eigengaps between all first $k$ singular values [17] [20]. The situation becomes worse if singular values form clusters, and even worse if one wants to obtain a gap-free result. To the best of our knowledge, running time depending on intermediate gaps is the only known result behind this type of algorithms, and even thought necessary by some authors [17]. Furthermore, Musco and Musco [18] explicitly stated it as an open question to design “small-block” or even “single-vector” algorithms like ours to obtain a better running time (i.e., time independent of intermediate gaps).

In this paper, we answer this open question in full. We carefully specify the single-vector computation routine, and provide novel analyses where the convergence neither depends on intermediate eigengaps, nor on the $k$-th eigengap. This yields our gap-free result. We also obtain gap-dependent results for free because all gap-free results imply gap-dependent ones. As for how to find individual singular vectors in each of the $k$ iterations, we use the recent advances of shift-and-inverse preconditioning technique developed in [11] [12], and reduce the problem to convex optimization that can be solved either with accelerated gradient descent (AGD) or accelerated SVRG. The former leads to faster accelerated $k$-SVD algorithm outperforming block Krylov, and the latter leads to faster accelerated and stochastic $k$-SVD algorithm outperforming Shamir. See Table 1 for a detailed comparison.

In terms of NNZ running time, somewhat surprisingly, we show if one sub-samples $A$ and then applies our new $k$-SVD algorithm, the resulting running time becomes $	ilde{O}(\text{nnz}(A) + \text{poly}(k, 1/\varepsilon \cdot d))$ where the polynomial dependence on $\varepsilon$ is quadratic. This improves upon [6] in certain (but sufficiently interesting) parameter regimes, see Table 2 but completely avoids the use of alternating minimization.

Finally, besides the running time advantages above, our algorithm also works when $k$ is not known to the algorithm, as opposed to block power methods or Krylov which need to know $k$ in advance. We call our algorithm LazySVD and discuss its running time formally in the subsequent sections.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Running time</th>
<th>Frobenius norm</th>
<th>Spectral norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7]</td>
<td>$O(\text{nnz}(A)) + O\left(\frac{k^2}{\varepsilon^2}(n + d) + \frac{k^2}{\varepsilon^2}\right)$</td>
<td>$(1 + \varepsilon)|\Delta|_F$</td>
<td>$(1 + \varepsilon)|\Delta|_F$</td>
</tr>
<tr>
<td>[6]</td>
<td>$O(\text{nnz}(A)) + \tilde{O}\left(\frac{(\sigma_1/\sigma_{k+1})^2}{\varepsilon^2} (n + d)\right)$</td>
<td>$(1 + \varepsilon)|\Delta|_F$</td>
<td>$|\Delta|_2 + \varepsilon|\Delta|_F$</td>
</tr>
<tr>
<td>LazySVD</td>
<td>$O(\text{nnz}(A)) + \tilde{O}\left(\frac{(\sigma_1/\sigma_{k+1})^4}{\varepsilon^2} d\right)$</td>
<td>N/A</td>
<td>$|\Delta|_2 + \varepsilon|\Delta|_F$</td>
</tr>
<tr>
<td></td>
<td>$O(\text{nnz}(A)) + \tilde{O}\left(\frac{(\sigma_1/\sigma_{k+1})^2}{\varepsilon^2} (n + d)\right)$</td>
<td>N/A</td>
<td>$|\Delta|_2 + \varepsilon|\Delta|_F$</td>
</tr>
<tr>
<td></td>
<td>$O(\text{nnz}(A)) + \tilde{O}\left(\frac{(\sigma_1/\sigma_{k+1})^4}{\varepsilon^2} d\right)$</td>
<td>$(1 + \varepsilon)|\Delta|_2$</td>
<td>$|\Delta|_2 + \varepsilon|\Delta|_F$</td>
</tr>
</tbody>
</table>

Table 2: Performance comparison among $O(\text{nnz}(A) + \text{poly}(1/\varepsilon))$ type of algorithms. Remark: we have not tried hard to improve the dependency with respect to $k$ or $(\sigma_1/\sigma_{k+1})$. See Remark 5.2.
Algorithm 1 AppxPCA(\(A, M, \delta, \varepsilon, p\))

**Input:** \(A\), an approximate matrix inversion method; \(M \in \mathbb{R}^{d \times d}\), a symmetric matrix satisfying 
\(0 \preceq M \preceq I\); \(\delta, \varepsilon \in (0, 0.5)\), a multiplicative error; \(\varepsilon \in (0, 1)\), a numerical accuracy parameter; and \(p \in (0, 1)\), the confidence parameter. ◦ running time only logarithmically depends on \(1/\varepsilon\) and \(1/p\).

1: \(m_1 \leftarrow \left\lceil 4 \log \left(\frac{28d^3}{p\varepsilon^2}\right)\right\rceil\), \(m_2 \leftarrow \left\lceil \log \left(\frac{36d^3}{p\varepsilon^2}\right)\right\rceil\);

\(\bigcirc\) \(m_1 = T^{PM}(8, 1/32, p)\) and \(m_2 = T^{PM}(2, \varepsilon/4, p)\) using definition in Lemma A.1

2: \(\bar{\varepsilon}_1 \leftarrow \frac{1}{\delta m_1} \left(\delta s^2\right)^{m_1}\) and \(\bar{\varepsilon}_2 \leftarrow \frac{\varepsilon}{\delta m_2} \left(\delta s^2\right)^{m_2}\)

3: \(\hat{w}_0 \leftarrow \text{a random unit vector}; s \leftarrow 0; \lambda(0) \leftarrow 1 + \delta_s;\)

4: **repeat**

5: \(s \leftarrow s + 1;\)

6: **for** \(t = 1\) to \(m_1\) **do**

7: \hspace{1em} Apply \(A\) to find \(\hat{w}_t\) satisfying \(\|\hat{w}_t - (\lambda^{(s-1)}I - M)^{-1}\hat{w}_{t-1}\| \leq \bar{\varepsilon}_1;\)

8: \hspace{1em} \(w \leftarrow \hat{w}_m / \| \hat{w}_m \|;\)

9: **for** \(t = 1\) to \(m_2\) **do**

10: \hspace{1em} \(\Delta^{(s)} \leftarrow \frac{1}{2} \left(\frac{1}{w^\top v^\top \varepsilon^2}\right)\) and \(\lambda^{(s)} \leftarrow \lambda^{(s-1)} - \frac{\Delta^{(s)}}{2};\)

11: until \(\Delta^{(s)} \leq \frac{\delta x \lambda^{(s)}}{3}\)

12: \(f \leftarrow s;\)

13: **for** \(t = 1\) to \(m_2\) **do**

14: \hspace{1em} Apply \(A\) to find \(\hat{w}_t\) satisfying \(\|\hat{w}_t - (\lambda^{(f)}I - M)^{-1}\hat{w}_{t-1}\| \leq \bar{\varepsilon}_2;\)

15: **return** \(w := \hat{w}_m / \| \hat{w}_m \|;\)

2 Preliminaries

Given a matrix \(A\) we denote by \(\|A\|_2\) and \(\|A\|_F\) respectively the spectral and Frobenius norms of \(A\). For \(q \geq 1\), we denote by \(\|A\|_q\) the Schatten \(q\)-norm of \(A\). We write \(A \succeq B\) if \(A, B\) are symmetric and \(A - B\) is positive semi-definite (PSD). We denote by \(\lambda_k(M)\) the \(k\)-th largest eigenvalue of a symmetric matrix \(M\), and \(\sigma_k(A)\) the \(k\)-th largest singular value of a rectangular matrix \(A\). It is clear that \(\lambda_k(\sigma_k(A)^2)\). We often denote by \(\sigma_1 \geq \cdots \sigma_d \geq 0\) the singular values of \(A \in \mathbb{R}^{d \times n}\), by \(\lambda_1 \geq \cdots \lambda_d \geq 0\) the eigenvalues of \(M = AA^\top \in \mathbb{R}^{d \times d}\). (Although \(A\) may have fewer than \(d\) singular values for instance when \(n < d\), if this happens, we append zeros in the end.) We denote by \(A^*_k\) the best rank-\(k\) approximation of \(A\).

We use \(\perp\) to denote the orthogonal complement of a matrix. More specifically, given a column orthonormal matrix \(U \in \mathbb{R}^{d \times k}\), we define \(U^\perp := \{x \in \mathbb{R}^d \mid U^\top x = 0\}\). For notational simplicity, we sometimes also denote \(U^\perp\) as a \(d \times (d - k)\) matrix consisting of some basis of \(U^\perp\).

**Theorem 2.1** (approximate matrix inverse). Given \(d \times d\) matrix \(M \succeq 0\), and constants \(\lambda, \delta > 0\) satisfying \(\lambda I - M \succeq \delta I\), one can minimize the quadratic \(f(x) := x^\top (\lambda I - M)x - b^\top x\) in order to invert \((\lambda I - M)^{-1}b\). Suppose the desired accuracy is \(\|x - (\lambda I - M)^{-1}b\| \leq \varepsilon\).

• Accelerated gradient descent (AGD) produces such an output \(x\) in \(O\left(\frac{\lambda^{1/2} \log \lambda}{\delta \varepsilon}\right)\) iterations, each requiring \(O(d)\) time plus the time needed to multiply \(M\) with a vector.

• If \(M\) is given in the form \(M = \frac{1}{d} \sum_{i=1}^{n} a_i a_i^\top\) and \(\|a_i\|_2 \leq 1\), then accelerated SVRG (see for instance [4]) produces such an output \(x\) in time \(O\left(\max\{nd, \frac{2^{3/4}d \lambda^{1/4}}{\delta \varepsilon}\} \log \frac{\lambda}{\varepsilon}\right)\).

3 Shift-and-Inverse PCA, Revisited

In this section, we define AppxPCA, the (multiplicative-)approximation algorithm for computing the leading eigenvector of a symmetric matrix using the shift-and-inverse routine [11][12]. Our pseudo-code in Algorithm 1 is a modification of Algorithm 5 that appeared in [11]. Since we need a more refined running time statement in this paper in terms of multiplicative error guarantees, and since the stated proof in [11] is anyways only a sketched one, we choose to carefully reprove a similar result of [11] and state the following theorem:
Furthermore, the total number of oracle calls to \( A \) is \( O(\log(1/\delta_x) m_1 + m_2) \), and each time we call \( A \) we have:

\[
\frac{\lambda_{(s)}}{\lambda_{\min}(\lambda^{(s)} I - M)} \leq \frac{12}{\delta_x} \quad \text{and} \quad \frac{1}{\lambda_{\min}(\lambda^{(s)} I - M)} \leq \frac{12}{\delta_x \lambda_1}.
\]

The stated conditions \( \frac{\lambda_{(s)}}{\lambda_{\min}(\lambda^{(s)} I - M)} \leq \frac{12}{\delta_x} \quad \text{and} \quad \frac{1}{\lambda_{\min}(\lambda^{(s)} I - M)} \leq \frac{12}{\delta_x \lambda_1} \) immediately imply the following running time of AppxPCA owing to Theorem 2.1.

**Corollary 3.2.**

- If \( A \) is AGD, the running time of AppxPCA is \( \widetilde{O}\left(\frac{1}{\delta_x^{1/2}}\right) \) multiplied with \( O(d) \) plus the time needed to multiply \( M \) with a vector.
- If \( M = \frac{1}{n} \sum_{i=1}^{n} a_i a_i^\top \) where each \( \|a_i\|_2 \leq 1 \), and \( A \) is accelerated SVRG, then the total running time of AppxPCA is \( \widetilde{O}\left(\max\{nd, \frac{n^{3/4}}{\lambda_1^{1/4} \delta_x^{1/2}}\}\right) \).

## 4 Our Main Algorithm and Theorems

Our algorithm LazySVD is formally stated in Algorithm 2. It simply applies \( k \) times AppxPCA, each time with a multiplicative error factor \( \delta_x/2 \), and projects the matrix \( M \) into the orthogonal space with respect to the obtained leading eigenvector.

**Algorithm 2** LazySVD\((A, M, k, \delta_x, \epsilon_{pca}, p)\)

**Input:** \( A \), an approximate matrix inversion method; \( M \in \mathbb{R}^{d \times d} \), a matrix satisfying \( 0 \preceq M \preceq I \); \( k \in [d] \), the desired rank; \( \delta_x \in (0, 1) \), a multiplicative error; \( \epsilon_{pca} \in (0, 1) \), a numerical accuracy parameter; and \( p \in (0, 1) \), a confidence parameter.

1. \( M_0 \leftarrow M \);\n2. \( V_0 = [] \);
3. **for** \( s = 1 \) **to** \( k \) **do**
4. \( v'_s \leftarrow \text{AppxPCA}(A, M_{s-1}, \delta_x/2, \epsilon_{pca}, p/k) \);
5. \( v_s \leftarrow \left( (I - V_{s-1} V_{s-1}^\top) v'_s \right) / \left\| (I - V_{s-1} V_{s-1}^\top) v'_s \right\| \); \hspace{1cm} \( \text{\textcircled{project}} v'_s \text{ to } V_{s-1}^\perp \)
6. \( V_s \leftarrow [V_{s-1}, v_s] \);
7. \( M_s \leftarrow (I - v_s v_s^\top) M_{s-1} (I - v_s v_s^\top) \); \hspace{1cm} we also have \( M_s = (I - V_s V_s^\top) M (I - V_s V_s^\top) \)
8. **end for**
9. **return** \( V_k \).

### 4.1 Our Main Theorems

We state our main approximation and running time theorems of LazySVD below, and then provide corollaries to translate them into gap-dependent and gap-free results for \( k \)-SVD.

**Theorem 4.1** (approximation). Let \( M \in \mathbb{R}^{d \times d} \) be a symmetric matrix with eigenvalues \( 1 \geq \lambda_1 \geq \cdots \geq \lambda_d \geq 0 \) and corresponding eigenvectors \( u_1, \ldots, u_d \). Let \( k \in [d] \), let \( \delta_x, p \in (0, 1) \), and let \( \epsilon_{pca} \leq \text{poly}(\epsilon, \delta_x, \frac{1}{\sqrt{\delta_x}}, \frac{1}{\lambda_{(k)}}) \). Then, LazySVD outputs a (column) orthonormal matrix \( V_k = (v_1, \ldots, v_k) \in \mathbb{R}^{d \times k} \) which, with probability at least \( 1 - p \), satisfies all of the following properties. (Denote by \( M_k = (I - V_k V_k^\top) M (I - V_k V_k^\top) \).)

(a) Core lemma: \( \| V_k^\top U \|_2 \leq \epsilon \), where \( U = (u_1, \ldots, u_d) \) is the (column) orthonormal matrix and \( j \) is the smallest index satisfying \( \lambda_j \leq 1 - \delta_x \| M_{k-1} \|_2 \).

---

\(^4\) The detailed specifications of \( \epsilon_{pca} \) can be found in the appendix where we restate the theorem more formally. To provide the simplest proof, we have not tightened the polynomial factors in the theoretical upper bound of \( \epsilon_{pca} \) because the running time depends only logarithmic on \( 1/\epsilon_{pca} \).
(b) Spectral norm guarantee: $\lambda_{k+1} \leq \|M_k\|_2 \leq \frac{\lambda_{k+1}}{1-\delta}$.
(c) Rayleigh quotient guarantee: $(1-\delta)\lambda_k \leq \mu_k^T M \mu_k \leq \frac{1}{1-\delta} \lambda_k$.
(d) Schatten-q norm guarantee: for every $q \geq 1$, we have $\|M_k\|_{S_q} \leq \left(\frac{1+\delta}{1-\delta}\right)^q \left(\sum_{i=k+1}^d \lambda_i^q\right)^{1/q}$.

We defer the proof of Theorem 4.1 to the full version, and we also have a section in the full version to highlight the technical ideas behind the proof Below we state the running time of LazySVD.

**Theorem 4.2** (running time). [LazySVD can be implemented to run in time]

- $\tilde{O}\left(\frac{knz(M)+k^2d}{d^2}\right)$ if $A$ is AGD and $M \in \mathbb{R}^{d \times d}$ is given explicitly;
- $\tilde{O}\left(\frac{knz(A)+k^2d}{d^2}\right)$ if $A$ is AGD and $M$ is given as $M = AA^T$ where $A \in \mathbb{R}^{d \times n}$; or
- $\tilde{O}(kn+d + \frac{kn^3/4}{\sigma_k^2})$ if $A$ is accelerated SVRG and $M = \frac{1}{n} \sum_{i=1}^n a_i a_i^T$ where each $\|a_i\|_2 \leq 1$.

Above, the $\tilde{O}$ notation hides logarithmic factors with respect to $k, d, 1/\delta, 1/p, 1/\lambda_1, \lambda_1/\lambda_k$.

**Proof of Theorem 4.2.** We call $k$ times AppxPCA, and each time we can feed $M_{s-1} = (I - V_{s-1} V_{s-1}^T) (I - V_{s-1} V_{s-1}^T)$ implicitly into AppxPCA thus the time needed to multiply $M_{s-1}$ with a $d$-dimensional vector is $O(dk + nzz(M))$ or $O(dk + nzz(A))$. Here, the $O(dk)$ overhead is due to the projection of a vector into $V_{s-1}^T$. This proves the first two running times using Corollary 3.2.

To obtain the third running time, when we compute $M_s$ from $M_{s-1}$, we explicitly project $a_i' \leftarrow (I - v_s v_s^T) a_i$ for each vector $a_i$, and feed the new $a_1', \ldots, a_n'$ into AppxPCA. Now the running time follows from the second part of Corollary 3.2 together with the fact that $\|M_{s-1}\|_2 \geq \|M_{k-1}\|_2 \geq \lambda_k$. □

### 4.2 Our Gap-Dependent and Gap-Free Results

Our main theorems imply the following corollaries (proved in full version of this paper).

**Corollary 4.3** (Gap-dependent $k$-SVD). Let $A \in \mathbb{R}^{d \times n}$ be a matrix with singular values $1 \geq \sigma_1 \geq \cdots \geq \sigma_d \geq 0$ and the corresponding left singular vectors $u_1, \ldots, u_d \in \mathbb{R}^d$. Let $gap = \sigma_k - \sigma_{k+1}$ be the relative gap. For fixed $\varepsilon, p > 0$, consider the output

$$V_k \leftarrow \text{LazySVD} \left(A, AA^T, k, \text{gap}, O\left(\frac{\varepsilon^2 \cdot \text{gap}^2}{k^2 \sigma_k^4 (\sigma_1/\sigma_k)^4}\right), p\right).$$

Then, defining $W = (u_{k+1}, \ldots, u_d)$, we have with probability at least $1 - p$:

$$V_k \text{ is a rank-}k \text{ (column) orthonormal matrix with } \|V_k^T W\|_2 \leq \varepsilon.$$

Our running time is $\tilde{O}\left(\frac{knz(A)+k^2d}{\sqrt{\varepsilon^2 d}}\right)$, or time $\tilde{O}(kn+d + \frac{kn^3/4}{\sigma_k^2})$ in the stochastic setting (1.1).

Above, both running times depend only logarithmically on $1/\varepsilon$ so enjoy linear convergence rates.

**Corollary 4.4** (Gap-free $k$-SVD). Let $A \in \mathbb{R}^{d \times n}$ be a matrix with singular values $1 \geq \sigma_1 \geq \cdots \geq \sigma_d \geq 0$. For fixed $\varepsilon, p > 0$, consider the output

$$\left(v_1, \ldots, v_k\right) = V_k \leftarrow \text{LazySVD} \left(A, AA^T, k, \frac{\varepsilon}{\sigma_1}, O\left(\frac{\varepsilon^6}{kn^2 \sigma_k^4 (\sigma_1/\sigma_k)^6}\right), p\right).$$

Then, defining $A_k = V_k^T A$ which is a rank $k$ matrix, we have with probability at least $1 - p$:

1. Spectral norm guarantee: $\|A - A_k\|_2 \leq (1 + \varepsilon)\|A - A_k^\ast\|_2$;
2. Frobenius norm guarantee: $\|A - A_k\|_F \leq (1 + \varepsilon)\|A - A_k^\ast\|_F$; and
3. Rayleigh quotient guarantee: $\forall i \in [k], \|v_i^T A A^T v_i - \sigma_i^2\|_2 \leq \varepsilon \sigma_i^2$.

Running time is $\tilde{O}\left(\frac{knz(A)+k^2d}{\sqrt{\varepsilon^2 d}}\right)$, or time $\tilde{O}(kn+d + \frac{kn^3/4}{\sigma_k^2})$ in the stochastic setting (1.1).
The total running time depends on (1) whether column or entry sampling is used, (2) which matrix

\[ \|A - A_k\|_{S_q} \leq (1 + \epsilon)\|A - A_k^*\|_{S_q} \] 

where \( \cdot \|_{S_q} \) is the Schatten-q norm. Rayleigh-quotient types of guarantee were introduced by Musco and Musco [18] for a more refined comparison. They showed that the block Krylov method satisfies \( |v_i^*AA^i - A_k^*| \leq \epsilon_2 \), which is slightly stronger than ours. However, these two guarantees are not much different in practice as we evidenced in our experiments.

5 NNZ running time

In this section, we translate our results in the previous section into the \( O(\text{nnz}(A) + \text{poly}(k, 1/\epsilon)(n + d)) \) running-time statements. The idea is surprisingly simple: we sample either random columns of \( A \), or random entries of \( A \), and then apply LazySVD to compute the \( k \)-SVD. Such translation directly gives either \( 1/\epsilon^2 \) results if AGD is used as the convex subroutine and either column or entry sampling is used, or a \( 1/\epsilon^2 \) result if accelerated SVRG and column sampling are used together.

Due to space limitation, we only informally state our theorem and defer all the details to the full paper.

**Theorem 5.1** (informal). Let \( A \in \mathbb{R}^{d \times n} \) be a matrix with singular values \( \sigma_1 \geq \cdots \geq \sigma_d \geq 0 \). For every \( \epsilon \in (0, 1/2) \), one can apply LazySVD with appropriately chosen \( \delta_x \) on a “carefully sampled version” of \( A \). Then, the resulting matrix \( V \in \mathbb{R}^{d \times k} \) can satisfy

- **spectral norm guarantee:** \( \|A - VV^T A\|_2 \leq \|A - A_k^*\|_2 + \epsilon\|A - A_k^*\|_F \)

- **Frobenius norm guarantee:** \( \|A - VV^T A\|_F \leq (1 + \epsilon)\|A - A_k^*\|_F \).

The total running time depends on (1) whether column or entry sampling is used, (2) which matrix inversion routine \( \mathcal{A} \) is used, and (3) whether spectral or Frobenius guarantee is needed. We list our deduced results in Table 2 and the formal statements can be found in the full version of this paper.

**Remark 5.2.** The main purpose of our NNZ results is to demonstrate the strength of LazySVD framework in terms of improving the \( \epsilon \) dependency to \( 1/\epsilon^2 \). Since the \( 1/\epsilon^2 \) rate matches sampling complexity, it is very challenging to match this bound in terms of running time. On one hand, we can use dimension reduction such as [8] to reduce the problem size to \( O(k/\epsilon^2) \); to the best of our knowledge, it is impossible to obtain any rate faster than \( 1/\epsilon^2 \) using solely dimension reduction. On the other hand, obtaining \( 1/\epsilon^2 \) dependency was the main contribution of [6]: they relied on alternating minimization but we have avoided it in our paper.

Finally, we have not tried very hard, and believe it possible, to improve the polynomial dependence with respect to \( k \) or \( (\sigma_1/\sigma_{k+1}) \).

6 Experiment

In this section we demonstrate the practicality of our SVD decomposition framework. We implement an iterative algorithm to compute approximate leading eigenvector \( k \) times, and compare it to block power method or block Krylov method. Notice that in theory, the best worse-case complexity for approximate leading eigenvector computation is obtained by AppxPCA on top of AGD or accelerated SVRG. However, in practice, Lanczos method runs much faster than these shift-and-inverse based methods and therefore we adopt Lanczos method as the method of choice to replace AppxPCA.

**Datasets.** We use datasets SNAP/amazon0302, SNAP/email-enron, and news20 that were also used by Musco and Musco [18], as well as an additional but famous dataset RCV1. The first two can be found on the SNAP website [15] and the last two can be found on the LibSVM website [10]. The four datasets give rise sparse matrices of dimensions 257570×262111, 35600×16507, 11269×53975, and 20242×47236 respectively.

Remark 4.5. The spectral and Frobenius guarantees we adopted are standard. It was observed that the spectral guarantee is more desirable than the Frobenius one in practice [18]. In fact, our algorithm implies for all \( q \geq 1 \), \( \|A - A_k\|_{S_q} \leq (1 + \epsilon)\|A - A_k^*\|_{S_q} \) where \( \cdot \|_{S_q} \) is the Schatten-q norm. Rayleigh-quotient types of guarantee were introduced by Musco and Musco [18] for a more refined comparison. They showed that the block Krylov method satisfies \( |v_i^*AA^i - A_k^*| \leq \epsilon_2 \), which is slightly stronger than ours. However, these two guarantees are not much different in practice as we evidenced in our experiments.

Remark 4.5. The spectral and Frobenius guarantees we adopted are standard. It was observed that the spectral guarantee is more desirable than the Frobenius one in practice [18]. In fact, our algorithm implies for all \( q \geq 1 \), \( \|A - A_k\|_{S_q} \leq (1 + \epsilon)\|A - A_k^*\|_{S_q} \) where \( \cdot \|_{S_q} \) is the Schatten-q norm. Rayleigh-quotient types of guarantee were introduced by Musco and Musco [18] for a more refined comparison. They showed that the block Krylov method satisfies \( |v_i^*AA^i - A_k^*| \leq \epsilon_2 \), which is slightly stronger than ours. However, these two guarantees are not much different in practice as we evidenced in our experiments.

5 NNZ running time

In this section, we translate our results in the previous section into the \( O(\text{nnz}(A) + \text{poly}(k, 1/\epsilon)(n + d)) \) running-time statements. The idea is surprisingly simple: we sample either random columns of \( A \), or random entries of \( A \), and then apply LazySVD to compute the \( k \)-SVD. Such translation directly gives either \( 1/\epsilon^2 \) results if AGD is used as the convex subroutine and either column or entry sampling is used, or a \( 1/\epsilon^2 \) result if accelerated SVRG and column sampling are used together.

Due to space limitation, we only informally state our theorem and defer all the details to the full paper.

**Theorem 5.1** (informal). Let \( A \in \mathbb{R}^{d \times n} \) be a matrix with singular values \( \sigma_1 \geq \cdots \geq \sigma_d \geq 0 \). For every \( \epsilon \in (0, 1/2) \), one can apply LazySVD with appropriately chosen \( \delta_x \) on a “carefully sampled version” of \( A \). Then, the resulting matrix \( V \in \mathbb{R}^{d \times k} \) can satisfy

- **spectral norm guarantee:** \( \|A - VV^T A\|_2 \leq \|A - A_k^*\|_2 + \epsilon\|A - A_k^*\|_F \)

- **Frobenius norm guarantee:** \( \|A - VV^T A\|_F \leq (1 + \epsilon)\|A - A_k^*\|_F \).

The total running time depends on (1) whether column or entry sampling is used, (2) which matrix inversion routine \( \mathcal{A} \) is used, and (3) whether spectral or Frobenius guarantee is needed. We list our deduced results in Table 2 and the formal statements can be found in the full version of this paper.

**Remark 5.2.** The main purpose of our NNZ results is to demonstrate the strength of LazySVD framework in terms of improving the \( \epsilon \) dependency to \( 1/\epsilon^2 \). Since the \( 1/\epsilon^2 \) rate matches sampling complexity, it is very challenging to match this bound in terms of running time. On one hand, we can use dimension reduction such as [8] to reduce the problem size to \( O(k/\epsilon^2) \); to the best of our knowledge, it is impossible to obtain any rate faster than \( 1/\epsilon^2 \) using solely dimension reduction. On the other hand, obtaining \( 1/\epsilon^2 \) dependency was the main contribution of [6]: they relied on alternating minimization but we have avoided it in our paper.

Finally, we have not tried very hard, and believe it possible, to improve the polynomial dependence with respect to \( k \) or \( (\sigma_1/\sigma_{k+1}) \).

6 Experiment

In this section we demonstrate the practicality of our SVD decomposition framework. We implement an iterative algorithm to compute approximate leading eigenvector \( k \) times, and compare it to block power method or block Krylov method. Notice that in theory, the best worse-case complexity for approximate leading eigenvector computation is obtained by AppxPCA on top of AGD or accelerated SVRG. However, in practice, Lanczos method runs much faster than these shift-and-inverse based methods and therefore we adopt Lanczos method as the method of choice to replace AppxPCA.

**Datasets.** We use datasets SNAP/amazon0302, SNAP/email-enron, and news20 that were also used by Musco and Musco [18], as well as an additional but famous dataset RCV1. The first two can be found on the SNAP website [15] and the last two can be found on the LibSVM website [10]. The four datasets give rise sparse matrices of dimensions 257570×262111, 35600×16507, 11269×53975, and 20242×47236 respectively.

\[ \text{Remark 4.5. The spectral and Frobenius guarantees we adopted are standard. It was observed that the spectral guarantee is more desirable than the Frobenius one in practice [18]. In fact, our algorithm implies for all } q \geq 1, \|A - A_k\|_{S_q} \leq (1 + \epsilon)\|A - A_k^*\|_{S_q} \text{ where } \cdot \|_{S_q} \text{ is the Schatten-q norm. Rayleigh-quotient types of guarantee were introduced by Musco and Musco [18] for a more refined comparison. They showed that the block Krylov method satisfies } |v_i^*AA^i - A_k^*| \leq \epsilon_2 \text{, which is slightly stronger than ours. However, these two guarantees are not much different in practice as we evidenced in our experiments.} \]

This is the best known spectral guarantee one can obtain using NNZ running time [6]. It is an open question whether the stricter \( \|A - VV^T A\|_2 \leq (1 + \epsilon)\|A - A_k^*\|_2 \) type of spectral guarantee is possible.

In fact, our framework turns every 1-PCA method satisfying Theorem 3.1 (including Lanczos method) into a rank \( k \)-SVD solver. However, our theoretical results (esp. stochastic and NNZ) rely on shift-and-invert because Lanczos is not a stochastic method.
We make the following observations:

We programmed the four algorithms using the same programming language with the same sparse-matrix implementation. We tested them single-threaded on the same Intel i7-3770 3.40GHz personal computer. As for the final low-dimensional SVD decomposition step at the end of the PM or Krylov method, it is a well-known issue that the Lanczos orthogonalization requires an orthogonalization of each $n \times k$ matrix with respect to all previously obtained $n \times k$ matrices. This greatly improves the numerical stability albeit sacrificing running time. We implement both these algorithms. In sum, we have implemented:

- **PM**: block power method for $T$ iterations.
- **Krylov**: stable block Krylov method for $T$ iterations [18].
- **Krylov(unstable)**: the three-term recurrence implementation of block Krylov for $T$ iterations.
- **LazySVD**: $k$ calls of the vanilla Lanczos method, and each call runs $T$ iterations.

### Implemented Algorithms.

For the block Krylov method, it is a well-known issue that the Lanczos type of three-term recurrence update is numerically unstable. This is why Musco and Musco [18] only used the stable variant of block Krylov which requires an orthogonalization of each $n \times k$ matrix with respect to all previously obtained $n \times k$ matrices. This greatly improves the numerical stability albeit sacrificing running time. We implement both these algorithms. In sum, we have implemented:

- **PM**: block power method for $T$ iterations.
- **Krylov**: stable block Krylov method for $T$ iterations [18].
- **Krylov(unstable)**: the three-term recurrence implementation of block Krylov for $T$ iterations.
- **LazySVD**: $k$ calls of the vanilla Lanczos method, and each call runs $T$ iterations.

### A Fair Running-Time Comparison.

For a fixed integer $T$, the four methods go through the dataset (in terms of multiplying $A$ with column vectors) the same number of times. However, since LazySVD does not need block orthogonalization (as needed in PM and Krylov) and does not need a $(Tk)$-dimensional SVD computation in the end (as needed in Krylov), the running time of LazySVD is clearly much faster for a fixed value $T$. We therefore compare the performances of the four methods in terms of running time rather than $T$.

We programmed the four algorithms using the same programming language with the same sparse-matrix implementation. We tested them single-threaded on the same Intel i7-3770 3.40GHz personal computer. As for the final low-dimensional SVD decomposition step at the end of the PM or Krylov method (which is not needed for our LazySVD), we used a third-party library that is built upon the x64 Intel Math Kernel Library so the time needed for such SVD is maximally reduced.

### Performance Metrics.

We compute four metrics on the output $V = (v_1, \ldots, v_k) \in \mathbb{R}^{n \times k}$:

- **Fnorm**: relative Frobenius norm error: $\frac{\|A - VV^T A\|_F - \|A - A_k^*\|_F}{\|A - A_k^*\|_F}$.
- **spectral**: relative spectral norm error: $\frac{\|A - VV^T A\|_2 - \|A - A_k^*\|_2}{\|A - A_k^*\|_2}$.
- **rayleigh(last)**: Rayleigh quotient error relative to $\sigma_{k+1}$: $\max_{j=1}^k |\sigma_j^2 - v_j^T AA^T v_j|/\sigma_{k+1}^2$.
- **rayleigh**: relative Rayleigh quotient error: $\max_{j=1}^k |\sigma_j^2 - v_j^T AA^T v_j|/\sigma_1^2$.

The first three metrics were also used by Musco and Musco [18]. We added the fourth one because our theory only predicted convergence with respect to the fourth but not the third metric. However, we observe that in practice they are not much different from each other.

### Our Results.

We study four datasets each with $k = 10, 20, 30$ and with the four performance metrics, totaling 48 plots. Due to space limitation, we only select six representative plots out of 48 and include them in Figure 1 (The full plots can be found in Figure 2 [3, 4 and 5 in the appendix.) We make the following observations:

- **LazySVD** outperforms its three competitors almost universally.
- **Krylov(unstable)** outperforms Krylov for small value $T$; however, it is less useful for obtaining accurate solutions due to its instability. (The dotted green curves even go up if $T$ is large.)
- **Subspace power method** performs the slowest unsurprisingly due to its lack of acceleration.

![Figure 1: Selected performance plots. Relative error (y-axis) vs. running time (x-axis).](image-url)
References


