Faster Algorithms for Schatten-p Low Rank Approximation

Praneeth Kacham
Carnegie Mellon University, Pittsburgh, PA, USA
Google Research, New York, USA

David P. Woodruff
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We study algorithms for the Schatten-p Low Rank Approximation (LRA) problem. First, we show that by using fast rectangular matrix multiplication algorithms and different block sizes, we can improve the running time of the algorithms in the recent work of Bakshi, Clarkson and Woodruff (STOC 2022). We then show that by carefully combining our new algorithm with the algorithm of Li and Woodruff (ICML 2020), we can obtain even faster algorithms for Schatten-p LRA.

While the block-based algorithms are fast in the real number model, we do not have a stability analysis which shows that the algorithms work when implemented on a machine with polylogarithmic bits of precision. We show that the LazySVD algorithm of Allen-Zhu and Li (NeurIPS 2016) can be implemented on a floating point machine with only logarithmic, in the input parameters, bits of precision. As far as we are aware, this is the first stability analysis of any algorithm using $O((k/\sqrt{\varepsilon})\,\mathrm{poly}(\log n))$ matrix-vector products with the matrix A to output a $1+\varepsilon$ approximate solution for the rank-k Schatten-p LRA problem.

2012 ACM Subject Classification Theory of computation → Mathematical optimization; Mathematics of computing → Mathematical analysis

Keywords and phrases Low Rank Approximation, Schatten Norm, Rectangular Matrix Multiplication, Stability Analysis

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.55

Category RANDOM

Related Version *Full Version*: `https://arxiv.org/abs/2407.11959`

Acknowledgements We would like to thank Cameron Musco, Christopher Musco, and Aleksandros Sobczyk for helpful discussions. We thank a Simons Investigator Award and NSF CCF-2335411 for partial support.

1 Introduction

Low Rank Approximation (LRA) is an important primitive in large scale data analysis. Given an $m \times n$ matrix A, and a rank parameter k, the task is to find a rank-k matrix B that minimizes $\|A - B\|$ where $\|\cdot\|$ is some matrix norm. Typically, we also require that the algorithms output a factorization $B = XY$ such that $X \in \mathbb{R}^{m\times k}$ and $Y \in \mathbb{R}^{k\times n}$. Such a factorization lets us compute the product Bz with an arbitrary vector z in time $O(k(n+m))$ which can be significantly smaller than the $\mathrm{nnz}(A)$ time required to multiply a vector with the original matrix A. Here $\mathrm{nnz}(A)$ denotes the number of non-zero entries of the matrix A. Thus, replacing A with a low rank approximation can make downstream tasks much faster. Additionally, if the matrix A has a low rank structure but is corrupted by noise, a low rank approximation of A can recover the underlying structure under suitable assumptions on the noise. We note that many low rank approximation algorithms, including ours, compute a rank-k orthonormal matrix W such that $\|A(I - WW^\top)\|$ is small and then define $X = AW$ and $Y = W^\top$.

In this paper, the error metric we consider is given by the Schatten-p norm for $p \geq 1$. Given a matrix M, the Schatten-p norm of M denoted by $\|M\|_{S_p}$ is defined as $(\sum_i \sigma_i(M)^p)^{1/p}$ where $\sigma_i(M)$ denotes the i-th singular value of M. Note that Schatten-2 norm is the same as the Frobenius norm, denoted by $\|M\|_{\mathsf{F}} = (\sum_{i,j} M_{ij}^2)^{1/2}$ and the Schatten-∞ norm is the same as the operator norm, denoted by $\|M\|_2 = \max_{x \neq 0} \|Mx\|_2/\|x\|_2$. In the presence of outliers, the Schatten-1 norm, $\sum_i \sigma_i(M)$, is considered to be more robust since the errors introduced by the outliers are not "squared" as it is done in the case of the Frobenius norm.

The Schatten-p norm low rank approximation problem asks to find a rank-k matrix B that minimizes $\|A - B\|_{S_p}$. As the Schatten-p norms are unitarily invariant, we have from Eckart-Young-Mirsky's theorem that $\|A - A_k\|_{S_p} = \min_{\text{rank-}k\, B} \|A - B\|_{S_p}$ for all $p \geq 1$, where A_k is the matrix obtained by truncating the Singular Value Decomposition (SVD) of A to only the top k singular values. This implies that a single matrix A_k is a *best* rank-k approximation for A for all values of p. However, computing the SVD of an $m \times n$ matrix takes $O(\min(mn^{\omega-1}, nm^{\omega-1}))$ time (see Appendix A), where ω is the matrix multiplication exponent. This time complexity is prohibitive when m and n are large. Thus, we relax the requirements and ask for a rank-k matrix B satisfying $\|A - B\|_{S_p} \leq (1+\varepsilon)\|A - A_k\|_{S_p}$ in the hope of obtaining faster algorithms than the SVD.

While a single matrix A_k is a best low rank approximation for A in all Schatten-p norms, it is not the case for approximate solutions, i.e., if B is a rank-k matrix that satisfies $\|A - B\|_{S_p} \leq (1+\varepsilon)\|A - A_k\|_{S_p}$ for some p, it may not be the case that $\|A - B\|_{S_q} \leq (1+\varepsilon)\|A - A_k\|_{S_q}$ for $q \neq p$. Thus, many approximation algorithms for Schatten-p LRA are tailored to the particular value p. There are two different lines of works for Schatten-p LRA in the literature: (i) Sketching based algorithms of Li and Woodruff [9] and (ii) Iterative algorithms of Bakshi, Clarkson and Woodruff [2]. We summarize the running times of the algorithms in Table 1. The sketch-based algorithms are usually non-adaptive and the iterative algorithms adaptively pick their matrix-vector product queries depending on the results in the previous round which makes them powerful as we can see from the superior running time over sketch-based algorithms when we desire solutions with small ε.

Sketching Algorithms. Li and Woodruff [9] gave (almost) input-sparsity time algorithms for Schatten-p LRA, extending the earlier input-sparsity time algorithms for Frobenius norm LRA from [4]. For $p < 2$, their algorithm runs in $\widetilde{O}(\text{nnz}(A) + \max(m,n) \cdot \text{poly}(k/\varepsilon))$ time and for $p > 2$, their algorithm runs in $\widetilde{O}(\text{nnz}(A) + \max(m,n) \cdot \min(m,n)^{\alpha_p}\, \text{poly}(k/\varepsilon))$ time, where $\alpha_p = (\omega - 1)(1 - 2/p)$. Note that for the current value of $\omega \approx 2.37$, their algorithm runs in $\Omega(mn)$ time for $p \geq 7.4$ and hence is not an "input-sparsity time" algorithm but for all constant p, k, ε, their algorithm runs in $o(\min(mn^{\omega-1}, nm^{\omega-1})$ time and therefore is faster than computing the SVD.

Table 1 Running times for $1+\varepsilon$ rank-k Schatten-p LRA algorithms for $m \times n$ matrices assuming $m \geq n$.

	Time Complexity
Li and Woodruff [9] ($p \in [1,2)$)	$O(\text{nnz}(A)\log n) + \widetilde{O}_p(mk^{2(\omega-1)/p}/\varepsilon^{(4/p-1)(\omega-1)})$ $+\widetilde{O}_p(k^{2\omega/p}/\varepsilon^{(4/p-1)(2\omega+2)})$
Li and Woodruff [9] ($p > 2$)	$O(\text{nnz}(A)\log n) + \widetilde{O}_p(n^{\omega(1-2/p)}k^{2\omega/p}/\varepsilon^{2\omega/(p+2)})$ $+\widetilde{O}_p(mn^{(\omega-1)(1-2/p)}(k/\varepsilon)^{2(\omega-1)/p})$
Bakshi et al. [2]	$O(p^{1/6}\varepsilon^{-1/3}\,\text{nnz}(A)k\log(n/\varepsilon) + mp^{(\omega-1)/6}k^{\omega-1}\varepsilon^{-(\omega-1)/3})$

Iterative Algorithms. Recently, Bakshi, Clarkson, and Woodruff [2] gave an iterative algorithm for Schatten-p LRA. Their algorithm runs the Block Krylov iteration algorithm of Musco and Musco [11] at *two* different block sizes for different number of iterations respectively. They show that the algorithm succeeds in computing a low rank approximation at one of the block sizes and show how to compute which block size succeeds in computing the approximation. For Schatten-p LRA, their algorithm requires $O(kp^{1/6}\operatorname{poly}(\log n)/\varepsilon^{1/3})$ matrix-vector products with the matrix A and hence can be implemented in $\widetilde{O}(\operatorname{nnz}(A)kp^{1/6}/\varepsilon^{1/3})$ time. At a high level, their algorithm runs the Block Krylov iteration algorithm with block size k for $O(p^{1/6}\varepsilon^{-1/3}\operatorname{poly}(\log n))$ iterations and with block size $O(p^{-1/3}\varepsilon^{-1/3}k)$ for $O(\sqrt{p}\operatorname{poly}(\log n))$ iterations. They set these parameters such that the algorithm requires an overall same number of matrix-vector products with A at both block sizes. They argue that for a matrix with a "flat" spectrum, the low rank approximation computed by the block size k algorithm is a $1+\varepsilon$ approximation and for a matrix with a "non-flat" spectrum, the solution computed by block size $O(p^{-1/3}\varepsilon^{-1/3}k)$ algorithm is a $1+\varepsilon$ approximation.

Comparison. As we can see from Table 1, the running times of these algorithms depend in a quite complicated way on the parameters $\operatorname{nnz}(A)$, m, n, ε and p. Throughout the paper, we assume that $m = n$, $\operatorname{nnz}(A) = n^2$ (i.e., the matrix A is dense) and $k \le n^c$ for a small constant c so that $k \ll n$. In some cases, where sparsity in the datasets cannot be well exploited, such as when processing the datasets using GPUs, it is natural to analyze the time complexities of the algorithms and compare the performances assuming that the inputs are dense.

For $p \in [1, 2)$, we have that the time complexity of the algorithm of [9] is $O(n^2\log n + n\operatorname{poly}(k)/\varepsilon^{(4/p-1)(\omega-1)} + \operatorname{poly}(k)/\varepsilon^{(4/p-1)(2\omega+2)})$ and the time complexity of the algorithm of [2] is $O(\varepsilon^{-1/3}n^2k\log(n) + n\operatorname{poly}(k)/\varepsilon^{(\omega-1)/3})$. We see that only when

$$1/\varepsilon > n^{\frac{1}{(4/p-1)(\omega+1)-1/6}},$$

the algorithm of [2] is faster than the sketching based algorithm of [9]. For $\omega \approx 2.371$ and $p = 1$, the above is achieved only when $1/\varepsilon \ge n^{\approx 0.1}$. Hence, in the high accuracy regime, the algorithm of [2] is faster than that of the sketching based algorithm of [9]. For other values of $p \in [1, 2)$, ε has to be even smaller than $1/n^{0.1}$ for the algorithm of [2] to be faster than the algorithm of [9].

For comparing the algorithms in the case $p > 2$, first we pick ε to be a constant and obtain that the running time of the algorithm of [9] is $O(n^2\log n + n^{1+(\omega-1)(1-2/p)}\operatorname{poly}(k))$ and the algorithm of [2] has a running time of $O(p^{1/6}n^2k\log(n))$. Thus, as long as $(\omega-1)(1-2/p) \le 1$, the sketch-based algorithm is faster than the iterative algorithm. We call p such that $(\omega - 1)(1 - 2/p) \le 1$, the *crossover point* from "sketch" to "iterative". For the current value of $\omega \approx 2.371$, the crossover point is ≈ 7.39.

Now consider the case of $\varepsilon = 1/n$ and constant p. The iterative algorithm of [2] has a running time of $O(n^{2+1/3}k\log(n))$ and the sketch based algorithm of [9] has a running time of $O(n^\omega\operatorname{poly}(k))$ and thus offers no improvement over the naïve SVD algorithm. This again shows that in the high precision regime, the small dependence on ε in the running time of the algorithm of [2] is crucial to obtain better than $O(n^\omega)$ time algorithm. Overall, we summarize the comparison between the algorithms in Table 2.

Our Improvements. We first *improve* the time complexity of the iterative algorithm of [2] for *all* parameter regimes. While the focus of their paper was to minimize the number of matrix-vector products required, we observe that by using fast rectangular matrix multiplication algorithms, we can obtain even faster algorithms using their technique of running the block

Table 2 In the case of $m = n$, $\text{nnz}(A) = n^2$ and $k = n^{o(1)}$, the table lists which of the previous works is asymptotically faster for the current value of $\omega \approx 2.371$. **Iterative** algorithm refers to the algorithm of [2] and the **Sketching** algorithm refers to the algorithm of [9]. In the above, crossover ≈ 7.4.

	Small ε ($\approx 1/n$)	Large ε
$p \in [1, 2)$	Iterative	Sketching
$2 < p <$ crossover	Iterative	Sketching
$p >$ crossover	Iterative	Iterative

Krylov iteration algorithm at different block sizes. Fast rectangular matrix multiplication algorithms let us obtain a different block-size vs iteration trade-off giving us faster algorithms. This algorithm directly achieves the fastest running times for small ε since we improve upon [2] in all regimes.

We saw above that for constant ε, the sketch based algorithm takes only $O(n^2 \log n)$ time when $p \lesssim 7.4$ and hence cannot be improved upon over asymptotically by more than polylog(n) factors in that regime. We show that using a combination of our fast iterative algorithm and the algorithm of [9] gives an algorithm that runs in near-linear time[1] for all $p \lesssim 22$ for appropriate ε values extending the values of p for which a Schatten-p LRA can be computed in $O(n^2 \log n)$ time, when the rank parameter $k \le n^c$.

Our combined algorithm works as follows: to solve a sub-problem in the algorithm of [9], we run our improved iterative algorithm for Schatten-p LRA with accuracy parameter $\varepsilon = 1/n$. As our improved iterative algorithm has a better dependence on ε than earlier algorithms, we obtain a faster algorithm for solving the sub-problem and hence obtain an $O(n^2 \log n)$ time algorithm for all $p \lesssim 22$. Thus, improving the performance of iterative algorithms in the small ε regime let us obtain faster algorithms overall in the large ε regime!

Numerically Stable Algorithms

While the algorithm of [2] and our modification give fast algorithms for Schatten-p Low Rank Approximation, it is not known if the Block Krylov iteration algorithm is stable when implemented on a floating point machine with $O(\log(n/\varepsilon))$ bits of precision. It is a major open question in numerical linear algebra to show if the Block Krylov iteration algorithm is stable. Obtaining fast algorithms that provably work on finite precision machines is a tricky problem in general. We note that until the recent work of Banks, Garza-Vargas, Kulkarni and Srivastava [3], it was not clear if an eigendecomposition of a matrix could be computed in $\widetilde{O}(n^\omega)$ time on a finite precision machine. Building on these ideas, another recent work [14] obtains fast and stable algorithms for the generalized eigenvalue problem. The sketch-and-solve methods, such as the algorithm of [9], are usually stable as the operations do not blow up the magnitude of the entries. As we note above, for large p, the algorithms in [9] are not input-sparsity time and hence an important question is if there are any stable input-sparsity time algorithms for large p. We answer this question in affirmative by showing that the LazySVD algorithm of [1] can be stably implemented on a floating point machine with $O(\log m\kappa/\varepsilon)$ bits of precision where $\kappa = \sigma_1(A)/\sigma_{k+1}(A)$. The LazySVD algorithm computes a low rank approximation for all $p \ge 2$.

[1] Note the near-linear here means $\widetilde{O}(n^2)$ as the input-matrix is assumed to have n^2 nonzero entries.

Similar to the Block Krylov iteration algorithm, LazySVD also needs $O(k\,\mathrm{poly}(\log n)/\sqrt{\varepsilon})$ matrix-vector products with A. Additionally, the factorization output by LazySVD is *simultaneously* a $1+\varepsilon$ approximation for all $p \ge 2$. To find a rank-k approximation of A, the LazySVD algorithm first computes a unit vector v which is an approximation to the top eigenvector of $A^\top A$. Then the algorithm deflates $A^\top A$ and forms the matrix $(I - vv^\top)A^\top A(I - vv^\top)$ and proceeds to find an approximation to the top eigenvector of $(I - vv^\top)A^\top A(I - vv^\top)$ and so on for a total of k rounds. The authors show that the span of k vectors found across all the iterations contains a $1 + \varepsilon$ approximation if the eigenvector approximations satisfy an appropriate condition. Thus, to implement the LazySVD algorithm on a floating point machine, we first need a stable routine that can compute approximations to the top eigenvector of a given matrix. We show that such a routine can be implemented stably using the Lanczos algorithm [12]. We additionally modify the LazySVD algorithm and show that the modification allows us to compute matrix-vector products with the deflated matrix to a good enough approximation which lets the Lanczos algorithm compute an approximation to the top eigenvector of the deflated matrix. Our slight modification to LazySVD turns out to be important in making the stability analysis go through.

The novelty of our stability analysis is that instead of showing each of the vectors $\widetilde{v}_1, \ldots, \widetilde{v}_k$ computed by a finite precision algorithm are close to the vectors $v_1, \ldots, v_k$ that would be computed by an algorithm with unbounded precision, we essentially argue that for all i, the projection matrices onto the subspaces spanned by $\widetilde{v}_1, \ldots, \widetilde{v}_i$ and $v_1, \ldots, v_i$ are close using induction. This change makes the stability analysis work with only a polylogarithmic number of bits of precision whereas showing all $\widetilde{v}_i$s are individually close to corresponding v_is would require polynomially many bits of precision.

1.1 Our Results

In the following, α denotes the constant such that an arbitrary $n \times n$ matrix can be multiplied with an arbitrary $n \times n^\alpha$ matrix using $O(n^{2+\eta})$ arithmetic operations for any constant $\eta > 0$. The matrix multiplication exponent ω is the smallest constant such that an arbitrary $n \times n$ matrix can be multiplied with an arbitrary $n \times n$ matrix using $O(n^{\omega+\eta})$ arithmetic operations for any constant $\eta > 0$. For simplicity, we ignore the constant η, and write as if the matrices can be multiplied in $O(n^\omega)$ time. We define $\beta \coloneqq (\omega - 2)/(1 - \alpha)$. Note that $\beta \le 1$.[2]

▶ **Theorem 1** (Informal, Theorem 5). *Given an $n \times n$ matrix A, a rank parameter k and an accuracy parameter ε, there is an algorithm that outputs a rank-k orthonormal matrix W that with probability ≥ 0.9 satisfies, $\|A(I - WW^\top)\|_{S_p} \le (1 + O(\varepsilon))\|A - A_k\|_{S_p}$. If $k \le \varepsilon \cdot n^\alpha$, then the algorithm runs in $\widetilde{O}(\sqrt{p}n^{2+\eta})$ time for any constant $\eta > 0$.*

Combining the algorithm in the above theorem and the algorithm of [9], we obtain the following result:

▶ **Theorem 2** (Informal, Theorem 8). *Given an $n \times n$ matrix A, a rank parameter k independent of n and any constant $\eta > 0$, there is a randomized algorithm that runs in time $\widetilde{O}((n^{1-2/p})^{2+\eta+(1-\alpha)\beta/(1+2\beta)}\,\mathrm{poly}(1/\varepsilon) + n^2)$ and outputs a rank-k projection $\hat{Q}$ that satisfies $\|A(I - \hat{Q})\|_{S_p}^p \le (1 + \varepsilon)\|A - A_k\|_{S_p}^p$, with probability ≥ 0.9*

The above theorem shows that for all p at most a suitable constant, the algorithm runs in $\widetilde{O}(n^2)$ time for $\varepsilon > 1/n^{c_p}$ for a small enough constant c_p and hence is faster than using the algorithm of [9] or the algorithm in Theorem 1.

[2] See Section 2.2.

The following result shows that our modification of LazySVD can be stably implemented on a floating point machine.

▶ **Theorem 3** (Informal, Theorem 11). *Given an $n \times d$ matrix A with condition number $\kappa(A) = \sigma_1(A)/\sigma_{k+1}(A)$, an accuracy parameter ε, a rank parameter k and probability parameter η, if the machine precision $\varepsilon_{\text{mach}} \le \text{poly}(\varepsilon\eta/n\kappa(A))$, then there is an algorithm that outputs a $d \times k$ matrix V_k such that $\kappa(V_k) \le 4$ and with probability $\ge 1-\eta$, for all $p \in [2, \infty]$,*

$$\|A(I - \text{Proj}_{\text{colspace}(V_k)})\|_{S_p} \le (1 + O(\varepsilon))\|A - A_k\|_{S_p},$$

and runs in time $O(\frac{\text{nnz}(A)k}{\sqrt{\varepsilon}}\,\text{poly}(\log(d\kappa(A)/\varepsilon\eta)) + d\,\text{poly}(k, \log(dk/\eta\varepsilon)))$.

In the above theorem, $\text{Proj}_{\text{colspace}(M)}$ denotes the orthogonal projection matrix onto the column space of M.

1.2 Implications to Practice

While the theoretical fast rectangular matrix multiplication algorithms are not practically efficient, the message of this paper is that by optimizing for the number of matrix-vector products as in [2], we are leaving a lot of performance on the table. In modern computing architectures, multiplying an $n \times n$ and an $n \times b$ matrix is, for example, much faster than b times the time required to multiply the $n \times n$ and an $n \times 1$ vector because of data locality and the opportunities for parallelization. Thus, in the algorithm of [2], running the block size k version for fewer iterations while increasing the larger block size b can give faster algorithms in practice than using the parameters that optimize for the number of matrix-vector products. We include a small experiment in the appendix which compares the time required to compute the product of an $n \times n$ matrix with matrices that have different numbers of columns.

LazySVD with our stability analysis uses a similar number of matrix vector products as the widely used Block Krylov iteration algorithm while requiring only polylogarithmic bits of precision. While as mentioned above, block-based algorithms such as Block Krylov iteration can be much faster than single-vector algorithms such as LazySVD and our modification of it, it is only the case when the matrix is directly given to us. When the matrix is implicitly defined in other ways (for e.g., as the Hessian of a neural network where we can efficiently compute Jacobian-Vector products), the difference in performance between block-based algorithms and single-vector algorithms is less pronounced. When guarantees of stability are required, the fastest algorithms in practice for Low Rank Approximation should use some combination of sketching as in [9] to reduce dimension stably and then use our modification of LazySVD algorithm to find the necessary top k subspace.

2 Preliminaries

2.1 Notation

For a positive integer n, we use $[n]$ to denote the set $\{1, \ldots, n\}$. We use the notation $\widetilde{O}(f(n))$ to denote $O(f(n)\,\text{poly}(\log(f(n))))$ and $\widetilde{O}_q(f(n))$ to hide the multiplicative factors that depend only on the parameter q. For a vector x, we use $\|x\|_2 = (\sum_i |x_i|^2)^{1/2}$ to denote the Euclidean norm of x. Given an $m \times n$ matrix A, we use $A_{i,j}$ to denote the entry in the index (i, j) of A. We use A_{i*} to denote the i-th row of A and A_{*j} to denote the j-th column. We identify the multiplication of an $m \times n$ matrix with an $n \times k$ matrix with the notation $[m, n, k]$. For a matrix A, we use $\text{colspace}(A)$ to denote the vector space $\{Ax \mid x \in \mathbb{R}^n\}$. For any vector space $V \in \mathbb{R}^n$, we use Proj_V to denote the linear operator which maps a vector x to the projection of x in the subspace V i.e., the nearest vector to x in V in terms of Euclidean distance. If the columns of X are an orthonormal basis for V, then $\text{Proj}_V = XX^\top$.

Let $A = U\Sigma V^\top$ be the singular value decomposition (SVD) of A and let $\sigma_1 \geq \cdots \geq \sigma_n$ (recall $m \geq n$) denote the singular values of A. For $k \leq n$, let $A_k := \sum_{i=1}^{k} \sigma_i U_{*i}(V^\top)_{i*}$ be the matrix obtained by truncating the SVD of A to the top k singular values.

We use $\|A\|_\mathsf{F}$ to denote the Frobenius norm $(\sum_{i,j} A_{i,j}^2)^{1/2}$ and $\|A\|_2$ to denote the operator norm $\max_{x \neq 0} \|Ax\|_2/\|x\|_2$. For $p \geq 1$, we define $\|A\|_{S_p} = (\sum_{i=1}^{n} \sigma_i^p)^{1/p}$ to be the Schatten-p norm. As $\|\cdot\|_{S_p}$ defines a norm, we have $\|A + B\|_{S_p} \leq \|A\|_{S_p} + \|B\|_{S_p}$ for any two $m \times n$ matrices A and B. Additionally, we have $\|A^\top\|_{S_p} = \|A\|_{S_p}$ and for any unitary matrices U', V', we have $\|U'AV'\|_{S_p} = \|A\|_{S_p}$.

2.2 Fast Rectangular Matrix Multiplication

Let ω denote the best matrix multiplication exponent. The current upper bound on ω is ≈ 2.371 [5] and for $\gamma < 1$, let $\omega(\gamma)$ denote the exponent such that the product of an $n \times n$ with an $n \times n^\gamma$ matrix can be computed using $O(n^{\omega(\gamma)+\eta})$ arithmetic operations for any constant $\eta > 0$. There exists $\alpha > 0.31$ [8, 6] such that for all $\gamma < \alpha$, $\omega(\gamma) = 2$ and for all $\gamma \geq \alpha$,

$$\omega(\gamma) \leq 2 + (\omega - 2)\frac{\gamma - \alpha}{1 - \alpha}.$$

See [7, 10] for the above bound on $\omega(\gamma)$. Recall $\beta := \frac{\omega - 2}{1 - \alpha}$. We now observe that $n^{1-\alpha} n^2 \geq n^\omega$ since a matrix product of the form $[n, n, n]$ can be computed using $n^{1-\alpha}$ matrix products of the form $[n, n, n^\alpha]$. Hence, $1 - \alpha \geq \omega - 2$, which implies $\beta \leq 1$.

3 Schatten-p LRA using Fast Matrix Multiplication

Algorithm 1 Block Krylov Iteration Algorithm [11].

```
  Input: An $n \times n$ matrix $A$, rank parameter $k$, block size $b$ and number $q$ of iterations
  Output: An orthonormal matrix $Z \in \mathbb{R}^{n \times k}$
1 $\Pi \sim \mathcal{N}(0,1)^{n \times b}$
2 $K \leftarrow \begin{bmatrix} A\Pi & (AA^\top)A\Pi & \cdots & (AA^\top)^q A\Pi \end{bmatrix}$    // The Krylov Matrix
3 Orthonormalize columns of $K$ to get an $n \times qb$ matrix $Q$
4 Compute $M := Q^\top AA^\top Q$
5 Set $\overline{U}_k$ to the top $k$ singular vectors of $M$
6 return $Z = Q\overline{U}_k$
```

3.1 Block Krylov Iteration Algorithm

The block Krylov Iteration algorithm of Musco and Musco [11] is stated as Algorithm 1. For any b, let $T(n, b)$ be the time to multiply an $n \times n$ matrix with an $n \times b$ matrix. The Block Krylov iteration algorithm with rank parameter k, block size $b \geq k$ and iteration count q (with $bq \leq n$) runs in time at most $(2q + 1)T(n, b) + n(qb)^{\omega-1} + 3T(n, qb) + (qb)^\omega + T(n, k)$.[3]

[3] Assuming that SVD of the $qb \times qb$ matrix M in Algorithm 1 can be computed in time $O((qb)^\omega)$.

Using the fact that $T(n, qb) \leq qT(n, b)$ and $qb \leq n$, we obtain that the time complexity of the algorithm is $O(qT(n, b) + n(qb)^{\omega-1})$. We now have $T(n, b) \geq (b/n)n^{\omega}$ since otherwise the matrix product of the form $[n, n, n]$ can be computed quicker than in n^{ω} time by computing the n/b products of the form $[n, n, b]$. Hence, $qT(n, b) \geq qbn^{\omega-1} \geq n(qb)^{\omega-1}$ using $qb \leq n$. Thus, we obtain that the time complexity of the Block Krylov Iteration algorithm with parameters k, b, q satisfying $b \geq k$ and $bq \leq n$ is $O(qT(n, b))$. We now state a few properties of the Block Krylov algorithm that we use throughout the paper.

▶ **Theorem 4.** *With a large probability over the Gaussian matrix Π, the following properties hold for the matrix Z computed by Algorithm 1:*

1. *There is a universal constant c such that for all $i \in [k]$,*
$$\sigma_i(Z^{\top}A)^2 \geq \|A^{\top}(Z)_{*i}\|_2^2 \geq \sigma_i^2 - (c\log^2 n/q^2)\sigma_{k+1}^2.$$
This follows from the per-vector error guarantee of Theorem 1 in [11].
2. *If $\mathsf{gap} := (\sigma_k/\sigma_{b+1}) - 1$ and $q \geq C\log(n/\varepsilon)/\sqrt{\min(1, \mathsf{gap})}$ for a large enough constant C, then for all $i \in [k]$, $\sigma_i(Z^{\top}A)^2 \geq \|A^{\top}(Z)_{*i}\|_2^2 \geq \sigma_i^2 - \varepsilon\sigma_{k+1}^2$.*

The second guarantee in the above theorem follows from the gap-dependent error bounds in Theorem 11 in [11]. Note the logarithmic dependence of q on $1/\varepsilon$.

Algorithm 2 Schatten-p Norm Subspace Approximation.

```
Input: An $n \times n$ matrix $A$, rank parameter $k$ and an accuracy parameter $\varepsilon$
Output: Approximate Solution to the Schatten-$p$ Norm Subspace Approximation
        problem
1 $q \leftarrow \begin{cases} \sqrt{p} & k \leq \varepsilon \cdot n^{\alpha} \\ \max(\sqrt{p}, p^{\frac{1}{2(1+2\beta)}}(k/n^{\alpha}\varepsilon)^{\frac{\beta}{1+2\beta}}) & \varepsilon \cdot n^{\alpha} \leq k \leq n^{\alpha} \\ \max(\sqrt{p}, p^{\frac{1}{2(1+2\beta)}}/\varepsilon^{\frac{\beta}{1+2\beta}}) & k \geq n^{\alpha} \end{cases}$
2 $b' \leftarrow \lceil (3/2)\max(1, k/q^2\varepsilon) \rceil$
3 $Z_1 \leftarrow \textsc{BlockKrylov}(A, \text{rank} = k, \text{block size} = k, \text{iterations} = O(q\log(n))$
4 $Z_2 \leftarrow \textsc{BlockKrylov}(A, \text{rank} = k, \text{block size} = b' + k, \text{iterations} = O(\sqrt{p}\log(n/\varepsilon))$
5 $W_1 \leftarrow \text{colspan}(A^{\top}Z_1)$
6 $W_2 \leftarrow \text{colspan}(A^{\top}Z_2)$
7 $W \leftarrow W_2$ if $\hat{\sigma}_k \geq (1 + 1/2p)\hat{\sigma}_{b'+k}$ and $W_1$ otherwise // These approximations to
    $\sigma_k$ and $\sigma_{b'+k}$ can be computed using the $M$ matrix computed in
    Algorithm 1
```

3.2 Main Theorem

▶ **Theorem 5.** *Given an $n \times n$ matrix A, a rank parameter k and an accuracy parameter ε, Algorithm 2 outputs a k dimensional orthonormal matrix W that with probability ≥ 0.9 satisfies, $\|A(I - WW^{\top})\|_{S_p} \leq (1 + O(\varepsilon))\|A - A_k\|_{S_p}$. For any constant $\eta > 0$, the running time of the algorithm is as follows:*

1. *For $k \leq \varepsilon n^{\alpha}$, the algorithm runs in time $\widetilde{O}(\sqrt{p}n^{2+\eta})$.*
2. *For $\varepsilon n^{\alpha} \leq k \leq n^{\alpha}$, the algorithm runs in time $\widetilde{O}(\max(\sqrt{p}n^{2+\eta}, p^{\frac{1}{2(1+2\beta)}}n^{2+\eta}(k/n^{\alpha}\varepsilon)^{\beta/(1+2\beta)}))$.*
3. *For $k \geq n^{\alpha}$, the algorithm runs in time $\widetilde{O}((p^{1/2}\varepsilon^{-\beta})^{1/(1+2\beta)}n^{2+\eta-\alpha\beta}k^{\beta})$.*

Assuming p is a constant independent of ε, the dependence on ε is at least better than $\varepsilon^{-1/3}$ as $\beta \leq 1$ which implies $\beta/(1 + 2\beta) \leq 1/3$. The proof of this theorem is similar to that of [2]. We include the proof in the full version.

4 Comparison with the Algorithm of Li and Woodruff [9]

For $n \times n$ matrices and $p > 2$, the algorithm of [9] for the Schatten-p norm Subspace Approximation problem, shown in Algorithm 3 runs in time

$$O(n^2 \log n) + \widetilde{O}_p\left(\frac{n^{\omega(1-2/p)}k^{2\omega/p}}{\varepsilon^{2\omega/p+2}} + n^{1+(\omega-1)(1-2/p)}(k/\varepsilon)^{2(\omega-1)/p}\right). \quad (1)$$

Let $K = k + \varepsilon/\eta_1 = k + n^{1-2/p}k^{2/p}/\varepsilon^{2/p}$. To obtain the above running time, they use a ridge leverage score sampling algorithm to compute a matrix S with $s = O(\varepsilon^{-2}K \log n)$ rows that satisfies (2) with a large probability. The same guarantee can instead be obtained by using the Sub-sampled Randomized Hadamard Transform (SRHT) [16] with $s = O(\varepsilon^{-2}K \log n)$ rows and the matrix-product SA can be computed in time $O(n^2 \log n)$. To obtain the subspace embedding guarantee for T as required in Algorithm 3, we can let the matrix T again be an SRHT with $r = O(\varepsilon^{-2}s \log n)$ columns and the product SAT can be computed in time $O(ns \log s) = O(\varepsilon^{-2}n(k + n^{1-2/p}k^{2/p}/\varepsilon^{2/p}))$.

The singular value decomposition of the matrix SAT can be computed in $O(rs^{\omega-1}) = O(\varepsilon^{-2\omega}(k + n^{1-2/p}k^{2/p}/\varepsilon^{2/p})^{\omega} \operatorname{polylog}(n))$ time and a basis for the rowspace of $W^\top SA$ can be computed in $O(skn)$ time. Overall, for constant k and ε, the algorithm of [9] runs in time $\widetilde{O}(n^2 + (n^{1-2/p})^{\omega})$. For $p > 2\omega/(\omega-2)$, their algorithm runs in n^{2+c_p} time for a constant $c_p > 0$ that depends on p. For the same parameters, our algorithm runs in $\widetilde{O}(n^2)$ time and hence we have an improvement. For $k \le n^{\alpha}$ and $\varepsilon = 1/n$, their algorithm runs in time $\Omega(n^{\omega})$ which means that computing the SVD of A is already faster whereas our algorithm runs in time $\widetilde{O}(n^{2+\frac{(1-\alpha)\beta}{1+2\beta}}) = o(n^{\omega})$ if $\omega > 2$. Hence, our algorithm improves upon the algorithm of [9] for a wide range of parameters. We note that computing the SVD of SAT turns out to be the most expensive step for large p. In the next subsection, we show that our Algorithm 2 can be used to sidestep the computation of the SVD of SAT, thereby giving an even faster algorithm.

We call $p^* = 2\omega/(\omega-2)$, the *crossover* point. For $p > p^*$, our Algorithm 2 is faster than the algorithm of [9]. For the current value of $\omega \approx 2.37$, $p^* \approx 12.8$. For $p < p^*$, the leading order term in the time complexity of Algorithm 3 is $O(n^2 \log n)$ for $\varepsilon > n^{-c_p}$ for a constant c_p depending on p, and hence is faster than Algorithm 2.

Algorithm 3 Schatten-p Norm Low Rank Approximation for $p > 2$ [9].

```
   Input: A matrix $A \in \mathbb{R}^{m \times n}$ and an accuracy parameter $\varepsilon$
   Output: A rank-$k$ orthonormal projection $Q$ satisfying
           $\|A(I-Q)\|_{S_p} \le (1+\varepsilon)\|A - A_k\|_{S_p}$
1  $\eta_1 \leftarrow O(\varepsilon^{1+2/p}/k^{2/p}n^{1-2/p})$
2  $S$ be a matrix with $s$ rows that satisfies
       $(1-\varepsilon)A^\top A - \eta_1\|A - A_k\|_F^2 \cdot I \preceq A^\top S^\top SA \preceq (1+\varepsilon)A^\top A + \eta_1\|A - A_k\|_F^2 \cdot I.$   (2)
3  $T \leftarrow$ Subspace embedding for $s$-dimensional subspaces with error $O(\varepsilon)$
4  $W \leftarrow$ Top $k$ left singular vectors of $SAT$
5  $Z \leftarrow$ Matrix whose columns are an orthonormal basis for the row space of $W^\top SA$
6  $Q \leftarrow ZZ^\top$
```

4.1 Further Improving the running time of [9] using our algorithm

Given an $n \times n$ matrix A, $p \geq 1$ and $r \leq n$, let $\|A\|_{(p,r)} = (\sum_{i=1}^{r} \sigma_i(A)^p)^{1/p}$. We can show that $\|\cdot\|_{(p,r)}$ is a norm over $n \times n$ matrices. As $\|\cdot\|_{(p,r)}$ is unitarily invariant, we have by Eckart-Young-Mirsky's theorem that $\|A - A_k\|_{(p,r)} = \min_{\text{rank-}k\ B} \|A - B\|_{(p,r)}$. In Lemma 4.2 of [9], they show that for S satisfying (2), if $\hat{Q}$ is a rank-k projection matrix with

$$\|SA(I - \hat{Q})\|_{(p,r)} \leq (1 + \varepsilon) \min_{\substack{\text{rank-}k \\ \text{projections } Q}} \|SA(I - Q)\|_{(p,r)}, \tag{3}$$

then $\|A(I - \hat{Q})\|_{S_p}^p \leq (1 + C_p\varepsilon)\|A - A_k\|_{S_p}^p$, for a constant C_p that only depends on p. They show that the matrix Q returned by Algorithm 3 satisfies (3) and then conclude that the matrix Q is a $1 + O(\varepsilon)$ approximation to the Schatten-p norm low rank approximation problem. We will now argue that there is a faster algorithm for computing a projection that satisfies (3). The algorithm does not require the computation of the SVD of the matrix SAT and hence does not incur the $O_{p,k,\varepsilon}(n^{(1-2/p)\omega})$ term in the running time. We first show that a $1 + \varepsilon$ approximate solution to the Schatten-p norm subspace approximation problem, is a $1 + \varepsilon n/r$ approximation to the (p, r) subspace approximation problem.

▶ **Lemma 6.** *For an arbitrary $m \times n$ matrix A ($m \leq n$), if $\hat{Q}$ is a rank-k projection matrix satisfying $\|A(I - \hat{Q})\|_{S_p}^p \leq (1 + \varepsilon)\|A - A_k\|_{S_p}^p$ and colspan($\hat{Q}$) $\subseteq$ rowspan(A), then for any $r \leq n$,*

$$\|A(I - \hat{Q})\|_{(p,r)}^p \leq (1 + \varepsilon \lceil (m - k)/r \rceil)\|A - A_k\|_{(p,r)}^p.$$

Proof. Let $\hat{Q}$ be a rank-k projection such that

$$\|A(I - \hat{Q})\|_{S_p}^p \leq (1 + \varepsilon) \min_{\text{rank-}k \text{ projections } Q} \|A(I - Q)\|_{S_p}^p = (1 + \varepsilon) \sum_{i=k+1}^{n} \sigma_i(A)^p.$$

Note that $\|A(I - \hat{Q})\|_{S_p}^p = \sum_{i=1}^{m-k} \sigma_i(A(I - \hat{Q}))^p$ since the matrix $A(I - \hat{Q})$ has rank at most $m - k$ from our assumption that colspan(Q) $\subseteq$ rowspan(A). Now, $\|A(I - \hat{Q})\|_{(p,r)}^p = \sum_{i=1}^{r} \sigma_i(A(I - \hat{Q}))^p$ and therefore,

$$\begin{aligned}\|A(I - \hat{Q})\|_{(p,r)}^p &= \|A(I - \hat{Q})\|_{S_p}^p - \sum_{i=r+1}^{m-k} \sigma_i(A(I - \hat{Q}))^p \\ &\leq (1 + \varepsilon) \sum_{i=k+1}^{m} \sigma_i(A)^p - \sum_{i=r+1}^{m-k} \sigma_i(A(I - \hat{Q}))^p.\end{aligned}$$

Since the matrix $A\hat{Q}$ has rank at most k, by Weyl's inequality, $\sigma_i(A(I - \hat{Q})) \geq \sigma_{i+k}(A)$ which implies

$$\begin{aligned}\|A(I - \hat{Q})\|_{(p,r)}^p &\leq \sum_{i=k+1}^{k+r} \sigma_i(A)^p + \varepsilon\|A - A_k\|_{S_p}^p + \left(\sum_{i=k+r+1}^{m} \sigma_i(A)^p - \sum_{i=r+1}^{m-k} \sigma_i(A(I - \hat{Q}))^p\right) \\ &\leq \min_{\text{rank-}k \text{ projections } Q} \|A(I - Q)\|_{(p,r)}^p + \varepsilon\|A - A_k\|_{S_p}^p.\end{aligned}$$

Finally, using the fact that $\|A - A_k\|_{S_p}^p \leq \lceil (m - k)/r \rceil \|A - A_k\|_{(p,r)}^p$, we obtain

$$\|A(I - \hat{Q})\|_{(p,r)}^p \leq (1 + \varepsilon\lceil (m - k)/r \rceil)\|A - A_k\|_{(p,r)}^p. \qquad ◀$$

Finally, we have the following lemma which shows how to find an approximate solution to the (p, r) Low Rank Approximation problem.

▶ **Lemma 7.** *Let $A \in \mathbb{R}^{m \times n}$ be an arbitrary matrix with $m \le n$. Given parameters k, p, r and ε, there is a randomized algorithm to find a rank-k projection $\hat{Q}$, that with probability $\ge 9/10$ satisfies,*

$$\|A(I - \hat{Q})\|_{(p,r)}^p \le (1+\varepsilon)\|A - A_k\|_{(p,r)}^p.$$

For constant p and $k \le m^\alpha$ and any constant $\eta > 0$, the randomized algorithm runs in time $\widetilde{O}(m^{2+\eta+(1-\alpha)\beta/(1+2\beta)}k^{\beta/(1+2\beta)}\operatorname{poly}(1/\varepsilon) + nm + nk^{\omega-1})$ and for $k \ge m^\alpha$, the algorithm runs in $\widetilde{O}(m^{2+\eta-\alpha\beta+\frac{\beta}{1+2\beta}}k^\beta \operatorname{poly}(1/\varepsilon) + nm^{1-\alpha\beta}k^\beta + nk^{\omega-1})$ time.

Proof. First we note that

$$\min_{\text{rank-}k \text{ projections } Q} \|A(I-Q)\|_{(p,r)}^p = \min_{\text{rank-}k \text{ projections } W} \|(I-W)A\|_{(p,r)}^p = \|A - A_k\|_{(p,r)}^p.$$

Let T be an SRHT matrix with $O(\varepsilon^{-2}m \operatorname{polylog}(n))$ rows. With a large probability, T is an ε subspace embedding for the rowspace of matrix A. Then

$$(1-\varepsilon)AA^\top \preceq ATT^\top A^\top \preceq (1+\varepsilon)AA^\top$$

and further for all rank-k projections W,

$$(1-\varepsilon)(I-W)AA^\top(I-W) \preceq (I-W)ATT^\top A^\top(I-W) \preceq (1+\varepsilon)(I-W)AA^\top(I-W).$$

We then have for all i that $\sigma_i((I-W)AT) = (\sqrt{1 \pm \varepsilon})\sigma_i((I-W)A)$. Therefore, $\|(I - W)AT\|_{(p,r)}^p = (1 \pm \varepsilon)^{p/2}\|(I-W)A\|_{(p,r)}^p$ for all rank-k projections W. Let Algorithm 2 be run on the matrix $T^\top A^\top$ with rank parameter k and approximation parameter ε/pm. By Theorem 5, we obtain a rank-k projection $\widehat{W}$ satisfying

$$\|T^\top A^\top(I - \widehat{W})\|_p \le (1 + \varepsilon/pm) \min_{\text{rank-}k \text{ projections } W} \|T^\top A^\top(I-W)\|_p$$

Using, Lemma 6, we obtain that

$$\|T^\top A^\top(I - \widehat{W})\|_{(p,r)}^p \le (1+\varepsilon) \min_{\text{rank-}k \text{ projections } W} \|T^\top A^\top(I-W)\|_{(p,r)}^p.$$

By using the relation between $\|(I-W)AT\|_{(p,r)}^p$ and $\|(I-W)A\|_{(p,r)}^p$ for all projections W, we get

$$\begin{aligned}\|A^\top(I-\widehat{W})\|_{(p,r)}^p &\le \frac{(1+\varepsilon)^{p/2+1}}{(1-\varepsilon)^{p/2}} \min_{\text{rank-}k \text{ projections } W} \|A^\top(I-W)\|_{(p,r)}^p \\ &\le (1+O(\varepsilon p))\|A - A_k\|_{(p,r)}^p.\end{aligned}$$

Now, $\|A - A(\widehat{W}A)^+(\widehat{W}A)\|_{(p,r)}^p \le \|A - \widehat{W}A\|_{(p,r)}^p = \|(I-\widehat{W})A\|_{(p,r)}^p \le (1+O(\varepsilon p))\|A - A_k\|_{(p,r)}^p$. Scaling ε, we obtain the result.

Runtime Analysis. The matrix AT can be computed in time $O(mn \log n)$. For constant p, Algorithm 2 runs on the matrix $T^\top A$ in time $\widetilde{O}(m^{2+\eta+(1-\alpha)\beta/(1+2\beta)}k^{\beta/(1+2\beta)}\operatorname{poly}(1/\varepsilon))$ for $k \le m^\alpha$ and in time $\widetilde{O}(m^{2+\eta-\alpha\beta+\frac{\beta}{1+2\beta}}k^\beta \operatorname{poly}(1/\varepsilon))$ for $k \ge m^\alpha$. Finally, the rowspace of $\widehat{W}^\top A$ can be computed in time $O(nm + nk^{\omega-1})$ for $k \le m^\alpha$ and $O(nm^{1-\alpha\beta}k^\beta + nk^{\omega-1})$ for $k \ge m^\alpha$. ◀

Using the above lemma, we can find a rank-k projection $\hat{Q}$ that satisfies

$$\|SA(I-\hat{Q})\|_{(p,r)}^p \le (1+\varepsilon)\|A-A_k\|_{(p,r)}^p$$

in time $\widetilde{O}((n^{1-2/p})^{2+\eta+(1-\alpha)\beta/(1+2\beta)}\operatorname{poly}(1/\varepsilon)+n^2)$ for constant k improving on the $\widetilde{O}(n^2+(n^{(1-2/p)})^{\omega}\operatorname{poly}(1/\varepsilon))$ running time of [9] for the current value of ω since $2+\frac{(1-\alpha)\beta}{1+2\beta} = 2+\frac{\omega-2}{1+2\beta} < \omega$ if $\beta \ne 0$. We thus have the following theorem.

▶ **Theorem 8.** *Given a dense $n \times n$ matrix A, a constant rank parameter k and any constant $\eta > 0$, there is a randomized algorithm that runs in time $\widetilde{O}((n^{1-2/p})^{2+\eta+(1-\alpha)\beta/(1+2\beta)}\operatorname{poly}(1/\varepsilon)+n^2)$ and outputs a rank-k projection $\hat{Q}$ that, with probability $\ge 9/10$, satisfies $\|A(I-\hat{Q})\|_{S_p}^p \le (1+\varepsilon)\|A-A_k\|_{S_p}^p$.*

For this algorithm, the crossover point is $\widetilde{p} = \frac{4(1+2\beta)}{\omega-2}+2$ i.e., only when $p > \widetilde{p}$, Algorithm 2 is faster than the algorithm in the above theorem for constant k and ε. For current values of ω, α, we have $\widetilde{p} \approx 22$. In particular, for constant k and $\varepsilon > n^{-c_p}$, for $p \lesssim 22$, the algorithm has a time complexity of only $\widetilde{O}(n^2)$.

5 Stability of LazySVD

5.1 Finite Precision Preliminaries

Following the presentation of [12], we say that a floating point machine has precision $\varepsilon_{\text{mach}}$ if it can perform computations to relative error $\varepsilon_{\text{mach}}$. More formally, let $\mathrm{fl}(x \circ y)$ be the result of the computation $x \circ y$ on the floating point machine where $\circ \in \{+,-,\times,\div\}$. We say that the floating point machine has a precision $\varepsilon_{\text{mach}}$ if for all x and y, $\mathrm{fl}(x\circ y) = (1+\delta)(x \circ y)$ where $|\delta| \le \varepsilon_{\text{mach}}$. Additionally, we also require $\mathrm{fl}(\sqrt{x}) = (1+\delta)\sqrt{x}$ for some δ with $|\delta| \le \varepsilon_{\text{mach}}$. Ignoring overflow or underflow, a machine which implements the IEEE floating point standard with $\ge \log_2(1/\varepsilon_{\text{mach}})$ bits of precision satisfies the above requirements (see [12, Section 5]). Given matrices A and B with at most n rows and columns, we can compute a matrix C, on a floating point machine, that satisfies $\|C - A\cdot B\|_2 \le \varepsilon_{\text{mach}}\operatorname{poly}(n)\|A\|_2\|B\|_2$ by directly computing C_{ij} as $\mathrm{fl}(\sum_k A_{ik}B_{kj})$.

5.2 Stability Analysis

Algorithm 4 LazySVD [1].

```
Input: A positive semidefinite matrix $M \in \mathbb{R}^{d\times d}, k \le d, \varepsilon, \varepsilon_{\text{pca}}, \eta$
Output: Vectors $v_1, \ldots, v_k$
1 $M_0 \leftarrow M$ and $V_0 \leftarrow []$
2 for $s = 1, \ldots, k$ do
3     $v'_s \leftarrow \texttt{AppxPCA}_{\varepsilon/2, \varepsilon_{\text{pca}}, \eta/k}(M_{s-1})$
4     $v_s \leftarrow (I - V_{s-1}V_{s-1}^\top)v'_s / \|(I - V_{s-1}V_{s-1}^\top)v'_s\|_2$
5     $V_s \leftarrow [V_{s-1}\, v_s]$
6     $M_s \leftarrow (I - V_sV_s^\top)M(I - V_sV_s^\top)$  // The matrix $M_s$ is not computed as we
        only need matrix vector products with $M_s$
7 return $V_k$
```

The LazySVD algorithm (Algorithm 4) of [1] crucially requires a routine called `AppxPCA` that computes an approximation to the top eigen vector of the given positive semidefinite matrix. While they use a particular `AppxPCA` algorithm in their results, any routine that satisfies the following definition can be plugged into the LazySVD algorithm.

▶ **Definition 9** (AppxPCA). *We say that an algorithm is* ***AppxPCA*** *with parameters $\varepsilon, \varepsilon_{\text{pca}}$ and η if given a positive semidefinite matrix $M \in \mathbb{R}^{d\times d}$ with an orthonormal set of eigenvectors $u_1, \ldots, u_d$ corresponding to eigenvalues $\lambda_1 \geq \cdots \geq \lambda_d \geq 0$, the algorithm outputs a unit vector w such that with probability $\geq 1-\eta$, $\sum_{i\in[d]:\lambda_i \leq (1-\varepsilon)\lambda_1} \langle w, u_i\rangle^2 \leq \varepsilon_{\text{pca}}$.*

We now show that Lanczos algorithm can be used to stably compute a vector that satisfies the AppxPCA guarantee.

▶ **Lemma 10.** *If for any vector x, we can compute a vector y such that*

$$\|y - M_s x\|_2 \leq O(\varepsilon_{\text{mach}}\operatorname{poly}(n)\kappa)\|M_s\|_2\|x\|_2$$

and if $\varepsilon_{\text{mach}} \leq \operatorname{poly}(\varepsilon_{\text{pca}}\eta/n\kappa)$, then we can compute a unit vector v such that with probability $\geq 1-\eta$, $\sum_{i:\lambda_i(M_s)\leq(1-\varepsilon)\lambda_1(M_s)} \langle v, u_i(M_s)\rangle^2 \leq \varepsilon$. The algorithm uses $O(\frac{1}{\sqrt{\varepsilon}}\operatorname{poly}(\log(d/\varepsilon\eta\varepsilon_{\text{pca}})))$ matrix vector products with M_s.

Proof. Let $\mathbf{z}$ be a d dimensional random vector with each coordinate being an independent Gaussian random variable. Let $M_s = \sum_i \lambda_i u_i u_i^\top$ be the eigendecomposition. Let r be the largest index such that $\lambda_r \geq (1-\varepsilon)\lambda_1$. Consider the vector $M_s^q \mathbf{z}$ for a q we choose later. We have

$$\mathbf{y} = M_s^q \mathbf{z} = \sum_{i=1}^{d} \lambda_i^q \langle u_i, \mathbf{z}\rangle u_i.$$

Consider $\langle u_1, \mathbf{z}\rangle$. By 2-stability of Gaussian random variables, $\langle u_1, \mathbf{z}\rangle \sim N(0, \|u_1\|_2^2) = N(0,1)$. Hence with probability $1-\eta$, $|\langle u_1, \mathbf{z}\rangle| \geq \eta$. We also have that with probability $\geq 1-\eta$, for all $i = 1, \ldots, d$ $|\langle u_i, \mathbf{z}\rangle| \leq O(\sqrt{\log d/\eta})$. Condition on these events. Now, $\|\mathbf{y}\|_2^2 = \sum_{i=1}^d \lambda_i^{2q}\langle u_i, \mathbf{z}\rangle^2 \geq \lambda_1^{2q}\langle u_1, \mathbf{z}\rangle^2 \geq \lambda_1^{2q}\eta^2$. Define $\hat{\mathbf{y}} = \mathbf{y}/\|\mathbf{y}\|_2$. Let $i > r$ so that $\lambda_i < (1-\varepsilon)\lambda_1$ by definition of r. We have

$$|\langle u_i, \hat{\mathbf{y}}\rangle| = \frac{|\langle u_i, \mathbf{y}\rangle|}{\|\mathbf{y}\|_2} \leq \frac{\lambda_i^q |\langle u_1, \mathbf{z}\rangle|}{\lambda_1^q \eta} \leq \frac{\lambda_i^q\sqrt{\log d/\eta}}{\lambda_1^q \eta} \leq (1-\varepsilon)^q \frac{C\sqrt{\log d/\eta}}{\eta}.$$

If $q \geq C\varepsilon^{-1}\log(d/\varepsilon_{\text{pca}}\eta)$ for a large enough constant C, we get $|\langle u_i, \hat{\mathbf{y}}\rangle| \leq \operatorname{poly}(\varepsilon_{\text{pca}}/d)$. Thus, $\sum_{i=r+1}^d |\langle u_i, \hat{\mathbf{y}}\rangle|^2 \leq \operatorname{poly}(\varepsilon_{\text{pca}})$.

Now define $f(x) = x^q$ so that $f(M_s)\mathbf{z} = \mathbf{y}$ and define $\rho = \lambda_1/q$. From [13, Chapter 3] there is a polynomial $p(x)$ of degree $\sqrt{2q\log 1/\gamma}$ such that for all $x \in [-\rho, \lambda_1 + \rho]$,

$$|p(x) - x^q| \leq e\gamma\lambda_1^q.$$

As we can compute matrix-vector products with M_s up to an additive error of $O(\varepsilon_{\text{mach}}\operatorname{poly}(n)\kappa)$, using Theorem 1 of [12] as long as $\varepsilon_{\text{mach}} \leq \varepsilon'\rho/(\operatorname{poly}(n)\kappa\|M_s\|_2) \leq \varepsilon'/\operatorname{poly}(n)\kappa$, we can compute a vector $\mathbf{y}'$ on a floating point machine, using $\sqrt{2q\log 1/\gamma}$ iterations such that

$$\begin{aligned}\|\mathbf{y} - \mathbf{y}'\|_2 = \|(M_s)^q\mathbf{z} - \mathbf{y}'\|_2 &\leq ((7e\gamma\sqrt{2q\log 1/\gamma})\lambda_1^q + \varepsilon'\lambda_1^q)\|\mathbf{z}\|_2 \\ &\leq O(\gamma\sqrt{2q\log 1/\gamma} + \varepsilon')\lambda_1^q\sqrt{d}.\end{aligned}$$

where we used that $\|\mathbf{z}\|_2 \leq O(\sqrt{d})$ with high probability. As $\|\mathbf{y}\|_2 \geq \lambda_1^q\eta$, we further obtain that

$$\|\mathbf{y} - \mathbf{y}'\|_2 \leq O(\gamma\sqrt{2q\log 1/\gamma} + \varepsilon')\sqrt{d}\|\mathbf{y}\|_2/\eta.$$

We set $\gamma = \text{poly}(\varepsilon_{\text{pca}}\eta/dq)$ and $\varepsilon' = \text{poly}(\varepsilon_{\text{pca}}\eta/d)$ to obtain that $\|\mathbf{y} - \mathbf{y}'\|_2 \le \text{poly}(\varepsilon_{\text{pca}}/d)\|\mathbf{y}\|_2$. Thus,

$$\|\hat{\mathbf{y}} - \mathbf{y}'/\|\mathbf{y}'\|_2\|_2 \le \|\mathbf{y}/\|\mathbf{y}\|_2 - \mathbf{y}'/\|\mathbf{y}'\|_2\|_2 \le \text{poly}(\varepsilon_{\text{pca}}/d).$$

On a floating point machine, we can normalize the vector $\mathbf{y}'$ to obtain a vector $\hat{\mathbf{y}}'$ such that $\|\hat{\mathbf{y}}'\|_2 = (1 \pm \varepsilon_{\text{mach}} \text{poly}(d))$ and $\|\hat{\mathbf{y}}' - \mathbf{y}'/\|\mathbf{y}'\|_2\|_2 \le \varepsilon_{\text{mach}} \text{poly}(d)$. By triangle inequality, we then obtain $\|\hat{\mathbf{y}} - \hat{\mathbf{y}}'\|_2 \le \text{poly}(\varepsilon/d) + \varepsilon_{\text{mach}} \text{poly}(d)$. Finally, for $i > r$

$$|\langle u_i, \hat{\mathbf{y}}'\rangle| \le |\langle u_i, \hat{\mathbf{y}}\rangle| + \|\hat{\mathbf{y}} - \hat{\mathbf{y}}'\|_2 \le \text{poly}(\varepsilon_{\text{pca}}/d) + \varepsilon_{\text{mach}} \text{poly}(d)$$

which then implies that as long as $\varepsilon_{\text{mach}} \le \text{poly}(\varepsilon_{\text{pca}}/d)$, we get $\sum_{i=r+1}^{d}\langle u_i, \hat{\mathbf{y}}\rangle^2 \le \text{poly}(\varepsilon_{\text{pca}})$.

Thus, we overall obtain that if $\varepsilon_{\text{mach}} \le \text{poly}(\varepsilon_{\text{pca}}\eta/d\kappa)$, we can obtain a vector $\hat{\mathbf{y}}'$ by running the Lanczos method for $O(\frac{1}{\sqrt{\varepsilon}} \text{poly}(\log(d/\varepsilon_{\text{pca}}\eta\varepsilon)))$ iterations such that with probability $\ge 1 - \eta$, $\|\hat{\mathbf{y}}'\|_2 = (1 \pm \varepsilon_{\text{mach}} \text{poly}(d))$ and

$$\sum_{i:\lambda_i(M_s) \le (1-\varepsilon)\lambda_1(M_s)} \langle \hat{\mathbf{y}}', u_i\rangle^2 \le \varepsilon_{\text{pca}}.$$

Overall, the algorithm uses $O(\frac{1}{\sqrt{\varepsilon}} \text{poly}(\log(d/\varepsilon\eta\varepsilon)))$ matrix vector products with M_s and uses an additional $O(\frac{d}{\sqrt{\varepsilon}} \text{poly}(\log(d/\varepsilon_{\text{pca}}\eta\varepsilon)))$ floating point operations. ◀

Finally, we modify the LazySVD algorithm (see Algorithm 5) to make it more stable when implemented on a floating point machine. The modification preserves the semantics of the algorithm in the real number model while allowing the stability analysis to go through. For the matrices that we need to run the routine `AppxPCA` on, we show that we can compute very accurate matrix-vector products so that the Lanczos algorithm can be used to approximate the top eigenvector to obtain the following theorem:

▶ **Theorem 11.** *Given an $n \times d$ matrix A with condition number $\kappa(A) = \sigma_1(A)/\sigma_{k+1}(A)$, an accuracy parameter ε, a rank parameter k and probability parameter η, if $\varepsilon_{\text{mach}} \le \text{poly}(\varepsilon\eta/n\kappa(A))$, there is an algorithm that outputs a $d \times k$ matrix V_k such that $\kappa(V_k) \le 4$ and for all $p \in [2, \infty]$,*

$$\|A(I - \text{Proj}_{\text{colspace}(V_k)})\|_{S_p} \le (1 + O(\varepsilon))\|A - A_k\|_{S_p}$$

and runs in time $O(\frac{\text{nnz}(A)k}{\sqrt{\varepsilon}} \text{poly}(\log(d\kappa(A)/\varepsilon\eta)) + d\,\text{poly}(k, \log(dk/\eta\varepsilon)))$.

Algorithm 5 Modified LazySVD.

```
  Input: A positive semidefinite matrix $M \in \mathbb{R}^{d\times d}, k \le d, \varepsilon, \varepsilon_{\text{pca}}, \eta$
  Output: Vectors $v'_1, \ldots, v'_k$
1 $M_0 \leftarrow M$ and $V_0 \leftarrow []$
2 for $s = 1, \ldots, k$ do
3     $v'_s \leftarrow \texttt{AppxPCA}_{\varepsilon,\varepsilon_{\text{pca}},\eta/k}((I - \text{Proj}_{\text{colspace}(V_{s-1})})M(I - \text{Proj}_{\text{colspace}(V_{s-1})}))$
4     $V_s \leftarrow [V_{s-1}\ v'_s]$
5 return $V_k$
```

For convenience, we denote any algorithm that satisfies Definition 9 as $\texttt{AppxPCA}_{\varepsilon,\varepsilon_{\text{pca}},\eta}$. We abuse notation and say that if a unit vector w satisfies $\sum_{i\in[d]:\lambda_i(M)\le(1-\varepsilon)\lambda_1(M)}\langle w, u_i(M)\rangle^2 \le \varepsilon_{\text{pca}}$, then "$w$ is $\texttt{AppxPCA}_{\varepsilon,\varepsilon_{\text{pca}}}(M)$".

In [1], the authors show that if $\varepsilon_{\text{pca}} = \text{poly}(\varepsilon, 1/d, \lambda_{k+1}/\lambda_1)$, then with probability $\geq 1-\eta$ (union bounding over the success of all k calls to the `AppxPCA` routine), the orthonormal matrix V_k output by Algorithm 4 satisfies

1. $\|(I - V_k V_k^\top)M(I - V_k V_k^\top)\|_2 \leq \frac{\lambda_{k+1}(M)}{(1-\varepsilon)}$,
2. $(1-\varepsilon)\lambda_k(M) \leq v_k^\top M v_k \leq \frac{1}{1-\varepsilon}\lambda_k(M)$ and
3. for every $p \geq 1$, $\|(I - V_k V_k^\top)M(I - V_k V_k^\top)\|_{S_p} \leq (1+O(\varepsilon))(\sum_{i=k+1}^d \lambda_i^p)^{1/p}$.

Since, the modified algorithm (Algorithm 5) has the same semantics as Algorithm 4, the properties 1 and 3 continue to hold for the modified LazySVD algorithm.

The advantage of the modification is that given any vector x, we can compute $(I - \text{Proj}_{\text{colspace}(V_s)})x$ very accurately on a floating point machine using stable algorithms for the least squares problem, thereby obtaining a vector y on a floating point machine that is a very good approximation to $M_s x = (I - \text{Proj}_{\text{colspace}(V_s)})M((I - \text{Proj}_{\text{colspace}(V_s)}))x$ for any given x. Below we have a result that states the stability of solving the Least Squares problem on a floating point machine.

▶ **Theorem 12** (Theorem 19.1 of [15]). *The algorithm for solving the least squares problem $\min_x \|Ax - b\|_2^2$ using Householder triangulation is backwards stable in the sense that the solution $\widetilde{x}$ satisfies*

$$\|(A + \delta A)\widetilde{x} - b\|_2^2 = \min_x \|(A+\delta A)x - b\|_2^2$$

for some matrix δA satisfying $\|\delta A\|_2 \leq O(\varepsilon_{\text{mach}}\|A\|_2)$.

Let $x^* = A^+ b$ and from the above theorem, we have $\widetilde{x} = (A+\delta A)^+ b$. Assuming $\varepsilon_{\text{mach}} \leq 1/2\kappa(A)$, we have $A + \delta A$ is full rank and therefore $(A+\delta A)^+ = ((A+\delta A)^\top(A+\delta A))^{-1}(A+\delta A)^\top$ using which we obtain that $\|A\widetilde{x} - Ax^*\|_2 \leq O(\varepsilon_{\text{mach}}\,\text{poly}(\kappa(A))\|b\|_2)$. Note that $Ax^* = \text{Proj}_{\text{colspace}(A)}b$. Thus, given a matrix A and a vector x, we can compute a vector y on a floating point machine such that $\|y - \text{Proj}_{\text{colspace}(A)}x\|_2 \leq O(\varepsilon_{\text{mach}}\,\text{poly}(\kappa(A), d)\|x\|_2)$.

Finally, we can compute another vector y' satisfying $\|y' - (I - \text{Proj}_{\text{colspace}(A)})x\|_2 \leq O(\varepsilon_{\text{mach}}\,\text{poly}(\kappa(A), d)\|x\|_2)$. Thus, given any vector x, if operations are computed using machine precision $\varepsilon_{\text{mach}}$ and if we assume that for any arbitrary vector x, we can compute a vector y satisfying $\|y - Mx\|_2 \leq \varepsilon_M\|M\|_2\|x\|_2$, then given any vector x, we can compute a vector y on a floating point machine satisfying $\|y - M_s x\|_2 \leq O(\varepsilon_{\text{mach}}\,\text{poly}(\kappa(V_s), d) + \varepsilon_M)\|M\|_2\|x\|_2$.

We now bound $\kappa(V_s)$. Assume that the vector v_s' satisfies $\|v_s'\|_2 = (1 \pm \text{poly}(d)\varepsilon_{\text{mach}})$. If the vector v_s' is $\texttt{AppxPCA}_{\varepsilon,\varepsilon_{\text{pca}}}(M_{s-1})$, then

$$\|\text{Proj}_{\text{colspace}(V_{s-1})}v_s'\|_2^2 \leq \sum_{i\in[d]:\lambda_i(M_{s-1})\leq(1-\varepsilon)\lambda_1(M_{s-1})} \langle v_s', u_i(M_s)\rangle^2 \leq \varepsilon_{\text{pca}}$$

where the first inequality follows from the fact that colspace(V_{s-1}) is spanned by the eigenvectors of M_{s-1} corresponding to zero eigenvalues. Using the above inequality, we can upper bound $\sigma_{\max}(V_s)$ and lower bound $\sigma_{\min}(V_s)$.

▶ **Lemma 13.** *Suppose V_{s-1} is a $d \times (s-1)$ matrix such that $\sigma_{\max}(V_{s-1}) = \alpha_{s-1}$ and $\sigma_{\min}(V_{s-1}) = \beta_{s-1}$. Let v_s' be a vector with $\|v_s'\|_2 = (1 \pm \text{poly}(d)\varepsilon_{\text{mach}})$ and satisfies $\|\text{Proj}_{\text{colspace}(V_{s-1})}v_s'\|_2^2 \leq \varepsilon_{\text{pca}}$. Let $V_s = [V_{s-1}\, v_s']$. Then $\sigma_{\max}(V_s) \leq \max(\sigma_{\max}(V_{s-1}), 1 + \text{poly}(d)\varepsilon_{\text{mach}}) + \sqrt{\varepsilon_{\text{pca}}}$ and*

$$\sigma_{\min}(V_s) \geq \sqrt{\max(0, \min(\sigma_{\min}(V_{s-1})^2, 1 - \text{poly}(d)\varepsilon_{\text{mach}}) - \sigma_{\max}(V_{s-1})\sqrt{\varepsilon_{\text{pca}}})}.$$

Proof. Let Q be an orthonormal basis for the column space of V_{s-1} and let $V_{s-1} = QR$ for a matrix R with $\sigma_{\max}(R) = \sigma_{\max}(V_{s-1}) = \alpha_{s-1}$ and $\sigma_{\min}(R) = \sigma_{\min}(V_{s-1}) = \beta_{s-1}$. We have that $\|Q^\top v'_s\|_2^2 = \|QQ^\top v'_s\|_2^2 = \|\mathrm{Proj}_{\mathrm{colspace}(V_{s-1})} v'_s\|_2^2 \le \varepsilon_{\mathrm{pca}}$. Let $x \in \mathbb{R}^s$ be an arbitrary unit vector. Let $x_1 \in \mathbb{R}^{s-1}$ and $x_2 \in \mathbb{R}$ be such that $x = (x_1, x_2)$. Now,

$$\begin{aligned}\|V_s x\|_2^2 &= \|QRx_1 + v'_s x_2\|_2^2 = \|Rx_1\|_2^2 + x_2^2\|v'_s\|_2^2 + (2x_2)x_1^\top R^\top Q^\top v'_s \\ &\le \alpha_{s-1}^2\|x_1\|_2^2 + (1+\mathrm{poly}(d)\varepsilon_{\mathrm{mach}})x_2^2 + (2|x_2|)\alpha_{s-1}\|x_1\|_2\sqrt{\varepsilon_{\mathrm{pca}}} \\ &\le \max(\alpha_{s-1}^2, 1+\mathrm{poly}(d)\varepsilon_{\mathrm{mach}}) + \alpha_{s-1}\sqrt{\varepsilon_{\mathrm{pca}}}(\|x_1\|_2^2 + |x_2|^2)\end{aligned}$$

which implies that $\|V_s\|_2 \le \max(\alpha_{s-1}, 1+\mathrm{poly}(d)\varepsilon_{\mathrm{mach}}) + \sqrt{\varepsilon_{\mathrm{pca}}}$. Similarly,

$$\begin{aligned}\|V_s x\|_2^2 &= \|QRx_1 + v'_s x_2\|_2^2 = \|Rx_1\|_2^2 + x_2^2\|v'_s\|_2^2 + (2x_2)x_1^\top R^\top Q^\top v'_s \\ &\ge \beta_{s-1}^2\|x_1\|_2^2 + (1-\mathrm{poly}(d)\varepsilon_{\mathrm{mach}})x_2^2 - 2|x_2|\|x_1\|_2\alpha_{s-1}\sqrt{\varepsilon_{\mathrm{pca}}} \\ &\ge \min(\beta_{s-1}^2, 1-\mathrm{poly}(d)\varepsilon_{\mathrm{mach}}) - \alpha_{s-1}\sqrt{\varepsilon_{\mathrm{pca}}}.\end{aligned}$$

Hence, $\sigma_{\min}(V_s) \ge \sqrt{\max(0, \min(\sigma_{\min}(V_{s-1})^2, 1-\mathrm{poly}(d)\varepsilon_{\mathrm{mach}}) - \sigma_{\max}(V_{s-1})\sqrt{\varepsilon_{\mathrm{pca}}})}$. ◀

Conditioned on the event that $\|\mathrm{Proj}_{\mathrm{colspace}(V_{s-1})} v'_s\|_2^2 \le \varepsilon_{\mathrm{pca}}$ and $\|v'_s\|_2^2 = (1 \pm \mathrm{poly}(d)\varepsilon_{\mathrm{mach}})$ for all $s = 1, \ldots, k$, from the above lemma, we obtain that $\|V_s\|_2 \le 1 + \mathrm{poly}(d)\varepsilon_{\mathrm{mach}} + k\sqrt{\varepsilon_{\mathrm{pca}}}$. If $\varepsilon_{\mathrm{pca}} \le 1/\mathrm{poly}(k)$ and $\varepsilon_{\mathrm{mach}} \le 1/\mathrm{poly}(d)$, then $\|V_s\|_2 \le 2$ for all $s = 1, \ldots, k$ which in turn implies that for all $s = 1, \ldots, k$, $\sigma_{\min}(V_s) \ge \sqrt{1 - \mathrm{poly}(d)\varepsilon_{\mathrm{mach}} - 2k\sqrt{\varepsilon_{\mathrm{pca}}}} \ge 1/2$ assuming $\varepsilon_{\mathrm{pca}} \le 1/\mathrm{poly}(k)$ and $\varepsilon_{\mathrm{mach}} \le 1/\mathrm{poly}(d)$.

Hence, $\kappa(V_s) \le 4$ for all $s = 1, \ldots, k$ in Algorithm 5 conditioned on the success of all the calls to `AppxPCA`. Thus, we can assume that given any vector x, we can compute a vector y on a floating point computer with precision $\varepsilon_{\mathrm{mach}} \le 1/\mathrm{poly}(d)$ such that $\|y - M_s x\|_2 \le O(\varepsilon_{\mathrm{mach}}\,\mathrm{poly}(d) + \varepsilon_M)\|M\|_2\|x\|_2$.

Let $A \in \mathbb{R}^{n \times d}$ be an arbitrary matrix with $n \ge d$. Define $M = A^\top A$ to be a $d \times d$ matrix. Given any vector x, we can compute a vector y satisfying $\|A^\top Ax - y\|_2 \le O(\varepsilon_{\mathrm{mach}}\,\mathrm{poly}(n)\|A\|_2^2\|x\|_2)$. As $\|A\|_2^2 = \|M\|_2$, we thus have that for any x, we can compute y satisfying $O(\varepsilon_{\mathrm{machine}}\,\mathrm{poly}(d)\|M\|_2\|x\|_2)$. Thus, ε_M as defined above can be taken as $\varepsilon_{\mathrm{mach}}\,\mathrm{poly}(d)$. Let $\kappa = \lambda_1(M)/\lambda_{k+1}(M)$. By definition of eigenvalues, for any matrix V with at most k columns, we have $\|(I - \mathrm{Proj}_{\mathrm{colspace}(V)})M(I - \mathrm{Proj}_{\mathrm{colspace}(V)})\|_2 \ge \lambda_{k+1}$. Hence for all $s = 1, \ldots, k$, $\|M_s\|_2 \ge \|M\|_2/\kappa$. Thus, given any vector x, we can compute a vector y satisfying $\|y - M_s x\|_2 \le O(\varepsilon_{\mathrm{mach}}\,\mathrm{poly}(n)\kappa)\|M_s\|_2\|x\|_2$.

Finally, we have the main theorem showing the stability of the LazySVD algorithm.

Proof of Theorem 11. Note that the algorithm in Lemma 10 satisfies the $\texttt{AppxPCA}_{\varepsilon,\varepsilon_{\mathrm{pca}},\eta}$ definition. Thus, if $\varepsilon_{\mathrm{pca}} \le \mathrm{poly}(\varepsilon/d\kappa)$, by Theorem 4.1 of [1], Algorithm 5 when run on the $d \times d$ matrix $A^\top A$ outputs a matrix V_k such that with probability $\ge 1 - \eta$, $\|(I - \mathrm{Proj}_{\mathrm{colspace}(V_k)})A^\top A(I - \mathrm{Proj}_{\mathrm{colspace}(V_k)})\|_2 \le \frac{1}{1-\varepsilon}\sigma_{k+1}(A)^2$ and for all $p' \ge 1$,

$$\|(I - \mathrm{Proj}_{\mathrm{colspace}(V_k)})A^\top A(I - \mathrm{Proj}_{\mathrm{colspace}(V_k)})\|_{S_{p'}} \le (1 + O(\varepsilon))(\sum_{i=k+1}^{d} \sigma_i(A)^{2p'})^{1/p'}.$$

Thus, we have $\|A(I - \mathrm{Proj}_{\mathrm{colspace}(V_k)})\|_2 \le (1 + O(\varepsilon))\sigma_{k+1}(A)$ and using the fact that $\|A^\top A\|_{S_p}^p = \|A\|_{S_{2p}}^{2p}$, for all $p \ge 2$, $\|A(I - \mathrm{Proj}_{\mathrm{colspace}(V_k)})\|_{S_p} \le (1 + O(\varepsilon))\|A - A_k\|_p$. We additionally have $\kappa(V_k) \le 4$ from Lemma 13.

Runtime Analysis. In each iteration of Algorithm 5, we require $O(\varepsilon^{-1/2}\operatorname{poly}(\log(d\kappa/\eta\varepsilon)))$ matrix vector products with the matrices A and $A^\top$. For iterations $s = 1, \dots, k$, we solve $O(\varepsilon^{-1/2}\operatorname{poly}(\log(d\kappa/\eta\varepsilon)))$ least squares problems on a fixed $d \times s$ matrix and different label vectors. Thus, the overall time complexity of the algorithm is

$$O\left(\frac{\operatorname{nnz}(A)k}{\sqrt{\varepsilon}}\operatorname{poly}(\log(d\kappa/\eta\varepsilon)) + d\operatorname{poly}(k, \log(d\kappa/\eta\varepsilon))\right). \qquad \blacktriangleleft$$

References

1 Zeyuan Allen-Zhu and Yuanzhi Li. LazySVD: Even faster SVD decomposition yet without agonizing pain. *Advances in neural information processing systems*, 29, 2016. URL: `https://proceedings.neurips.cc/paper/2016/hash/c6e19e830859f2cb9f7c8f8cacb8d2a6-Abstract.html`.

2 Ainesh Bakshi, Kenneth L Clarkson, and David P Woodruff. Low-rank approximation with $1/\varepsilon^{1/3}$ matrix-vector products. *STOC 2022. arXiv:2202.05120*, 2022. `doi:10.48550/arXiv.2202.05120`.

3 Jess Banks, Jorge Garza-Vargas, Archit Kulkarni, and Nikhil Srivastava. Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time. *Foundations of Computational Mathematics*, pages 1–89, 2023. `doi:10.1007/s10208-022-09577-5`.

4 Kenneth L. Clarkson and David P. Woodruff. Low-rank approximation and regression in input sparsity time. *J. ACM*, 63(6):Art. 54, 45, 2017. `doi:10.1145/3019134`.

5 Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. *FOCS 2023. arXiv:2210.10173*, 2022. `doi:10.48550/arXiv.2210.10173`.

6 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018. `doi:10.1137/1.9781611975031.67`.

7 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *2012 IEEE 53rd annual symposium on foundations of computer science*, pages 514–523. IEEE, 2012. `doi:10.1109/FOCS.2012.80`.

8 François Le Gall. Faster rectangular matrix multiplication by combination loss analysis. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3765–3791. SIAM, 2024. `doi:10.1137/1.9781611977912.133`.

9 Yi Li and David Woodruff. Input-sparsity low rank approximation in schatten norm. In *International Conference on Machine Learning*, pages 6001–6009. PMLR, 2020. URL: `http://proceedings.mlr.press/v119/li20q.html`.

10 Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theoretical Computer Science*, 23(2):171–185, 1983. `doi:10.1016/0304-3975(83)90054-3`.

11 Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *Advances in neural information processing systems*, 28, 2015. URL: `https://proceedings.neurips.cc/paper/2015/hash/1efa39bcaec6f3900149160693694536-Abstract.html`.

12 Cameron Musco, Christopher Musco, and Aaron Sidford. Stability of the Lanczos method for matrix function approximation. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1605–1624. SIAM, Philadelphia, PA, 2018. `doi:10.1137/1.9781611975031.105`.

13 Sushant Sachdeva and Nisheeth K Vishnoi. Faster algorithms via approximation theory. *Foundations and Trends® in Theoretical Computer Science*, 9(2):125–210, 2014. `doi:10.1561/0400000065`.

14 Aleksandros Sobczyk, Marko Mladenović, and Mathieu Luisier. Hermitian pseudospectral shattering, cholesky, hermitian eigenvalues, and density functional theory in nearly matrix multiplication time. *arXiv preprint arXiv:2311.10459*, 2023. doi:10.48550/arXiv.2311.10459.

15 Lloyd N. Trefethen and David Bau, III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997. doi:10.1137/1.9780898719574.

16 Joel A Tropp. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(01n02):115–126, 2011. doi:10.1142/S1793536911000787.

A Time Complexity of SVD in the Real RAM model

Consider an $m \times n$ matrix A where $m \le n$. We can compute the matrix $M = A^\top A$ in time $O(nm^{\omega-1})$, where ω is the matrix multiplication exponent. Using the eigendecomposition algorithm of [3], we can then compute a matrix V and a diagonal matrix D satisfying $\|M - VDV^{-1}\|_2 \le \|M\|_2/\operatorname{poly}(n)$ in time $\widetilde{O}(m^\omega)$. Although the matrix M is symmetric, the matrix V output by the algorithm may not be orthonormal. In the real RAM model, we can perform the following changes to their algorithm:

1. The Ginibre perturbation step is replaced with the symmetric Gaussian Orthogonal Ensemble perturbation as mentioned in Remark 6.1 of [3].
2. In step 5 of the algorithm EIG in [3], after computing the orthonormal matrices $\widetilde{Q}_+$ and $\widetilde{Q}_-$, we modify $\widetilde{Q}_-$ to an orthonormal basis of the column space of the matrix $(I - \widetilde{Q}_+\widetilde{Q}_+^\top)\widetilde{Q}_-$. This ensures that $\widetilde{Q}_+^\top\widetilde{Q}_- = 0$, while preserving the properties of $\widetilde{Q}_-$ guaranteed by the algorithm DEFLATE. Note that the matrix $\widetilde{Q}_-$ can be updated in time $\widetilde{O}(n^\omega)$ in the real RAM model.

Thus, the algorithm of [3] can be used to compute an orthonormal matrix V and a diagonal matrix D such that $\|M - VDV^\top\|_2 \le \|M\|_2/\operatorname{poly}(n)$ in $\widetilde{O}(nm^{\omega-1})$ time in the real RAM model.

If we define $U = AV \cdot D^{-1/2}$, then $U, D^{1/2}, V^\top$ is an approximate singular value decomposition of the matrix A, where the matrices U, V are orthonormal up to a $1/\operatorname{poly}(n)$ error. Since the matrix AV can be computed in $\widetilde{O}(nm^{\omega-1})$, we obtain that SVD of a well conditioned matrix can be computed in $\widetilde{O}(nm^{\omega-1})$ time.

B An Experiment

We consider multiplying an $n \times n$ matrix with an $n \times d$ matrix while varying d. We set $n = 10{,}000$ and vary d to take values in the interval $[10, 100]$. If t_d is the median amount of time (over 5 repetitions) to compute the product of an $n \times n$ matrix with an $n \times d$ matrix, we obtain a color map (Figure 1) of the values $(j/i)/(t_j/t_i)$ for $j \ge i$. If $(j/i)/(t_j/t_i)$ is large then t_j is much smaller than $t_i(j/i)$ which is what we would expect if the matrix-multiplication time scales linearly with the dimension.

The experiment was performed using NumPy on a machine with 2 cores. We see that fixing an i, as we increase j, t_j becomes smaller compared to $t_i \cdot (j/i)$. Hence, it is advantageous to run with larger block sizes if it means that it reduces the number of iterations for which the smaller block size is to be run. In the proof of Theorem 5, we see that if we increase the larger block to 4 times the original, then the number of iterations the smaller block size is to be run decreases to 0.5x the original. Based on the characteristics of the machine, we can obtain significant improvements over the parameters obtained by optimizing for matrix-vector products.

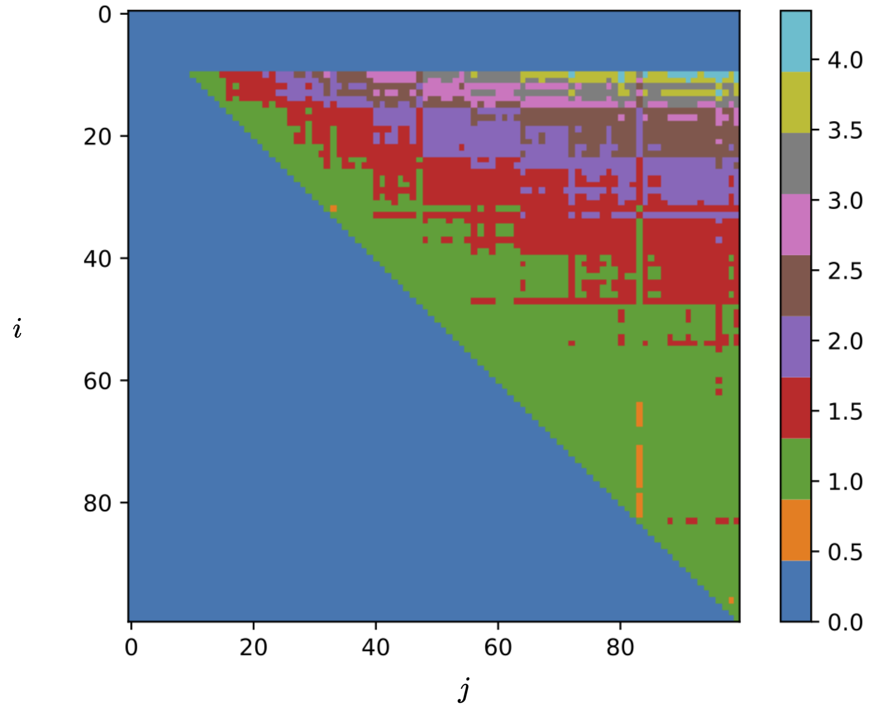

Figure 1 Color Map of $(j/i)/(t_j/t_i)$.