History PCA: A New Algorithm for Streaming PCA

Puyudi Yang, Cho-Jui Hsieh, Jane-Ling Wang
University of California, Davis
pydyang, chohsieh, janelwang@ucdavis.edu

Abstract

In this paper we propose a new algorithm for streaming principal component analysis. With limited memory, small devices cannot store all the samples in the high-dimensional regime. Streaming principal component analysis aims to find the k-dimensional subspace which can explain the most variation of the d-dimensional data points that come into memory sequentially. In order to deal with large d and large N (number of samples), most streaming PCA algorithms update the current model using only the incoming sample and then dump the information right away to save memory. However the information contained in previously streamed data could be useful. Motivated by this idea, we develop a new streaming PCA algorithm called History PCA that achieves this goal. By using $O(Bd)$ memory with $B \approx 10$ being the block size, our algorithm converges much faster than existing streaming PCA algorithms. By changing the number of inner iterations, the memory usage can be further reduced to $O(d)$ while maintaining a comparable convergence speed. We provide theoretical guarantees for the convergence of our algorithm along with the rate of convergence. We also demonstrate on synthetic and real world data sets that our algorithm compares favorably with other state-of-the-art streaming PCA methods in terms of the convergence speed and performance.

Keywords: Streaming PCA, Dimension Reduction

1 Introduction

Principal component analysis (PCA) is one of the most fundamental tools for feature extraction in machine learning problems such as data compression, classification, clustering, image processing and visualization.

Given a data set $X \in \mathbb{R}^{N\times d}$ with N samples and d features, PCA can be easily performed through eigen decomposition of the sample covariance matrix $\frac{1}{N}X^\top X$ or the singular value decomposition of the data matrix X, and the theoretical guarantee is established by matrix Bernstein theory [21, 20]. Throughout the paper we assume that X has mean zero for notation simplicity. For X with large N and d, we may not be able to store matrices of size $N \times d$ or $d \times d$ due to limited memories on small devices such as laptops and mobile phones, and it is often expensive to conduct multiple passes over the data. For these cases, streaming PCA algorithms [15, 6, 17, 16, 7, 12, 5, 4, 14, 13] come into play, with the nice property that they only need to use $O(d)$ or $O(Bd)$ memory to store the current sample or the current block of samples.

While these streaming PCA algorithms resolve the memory issues, they often suffer from slow convergence in practice. The main reason is that most streaming PCA algorithms in the past decades do not fully utilize information in previously streamed data, which came in sequentially and contain valuable information. This motivates us to develop a new algorithm that utilizes past information effectively without taking much memory space.

Specifically, our new algorithm can compute the top-k principal components in the streaming setting with $O(Bd)$ memory requirement, where B is the block size and can be pretty small (e.g., 1 or 10). The convergence speed of the new algorithm is much faster than existing approaches mainly due to effective use of past information, and by setting the number of inner iterations to one, the memory cost of our algorithm can be reduced to $O(d)$. The resulting algorithm is similar to Oja's algorithm but with the ability to compute the top-k principal components altogether without the need to tune the step size as in Oja's algorithm. We provide theoretical justification of our algorithm.

Experimental results show that our algorithm consistently outperforms existing algorithms in the streaming setting. Moreover, we test the ability of our algorithm to compute PCA on massive data sets. Surprisingly it outperforms existing streaming and non-streaming algorithms (including VR-PCA and power method) in terms of both the number of data passes and real run time.

The rest of the paper is outlined as follows. First, we discuss and compare current algorithms on streaming PCA. Then, we propose the History PCA algorithm with theoretical guarantees. Finally, we provide experiments on both simulated streaming data sets and real world data sets.

2 Related Work

There are many existing methods to estimate the top k eigenvectors of the true covariance matrix. It was shown by Adamczak et al. [1] that the sample covariance matrix can guarantee estimation of the true covariance matrix with a fixed error in the operator norm for sub-exponential distributions, if the sample size is $O(d)$. Later, this statement is generalized to distributions with finite fourth moment and the true eigenvectors of the distribution can be recovered with high probability by the singular value decomposition of the sample covariance matrix formed by $O(d)$ samples [21].

2.1 Non-streaming PCA Algorithms

When the full data set is given in advance, the optimal way to estimate principal components is to conduct eigen decomposition on the sample covariance matrix $\frac{1}{N}X^\top X$. However, it is often impossible to form the $d \times d$ sample covariance matrix for large-scale data sets. So we need to compute $\frac{1}{N}(X^\top X)v = \frac{1}{N}X^\top(Xv)$ in the power method without explicitly forming the sample covariance matrix.

Recently, Shamir [18, 19] proposed the VR-PCA method that outperforms power method for PCA computation. The main idea is to reformulate PCA as a stochastic optimization problem and apply the variance reduction techniques [9]. Although VR-PCA looks similar to streaming PCA, it cannot obtain the first update until the second pass. So it is not suitable in the streaming setting.

Since the entire data are often too large to be stored on the hard disk, algorithms that are specifically designed for streaming data have become a research focus in the past few decades.

2.2 Streaming PCA Algorithms

A classical algorithm for streaming PCA is Oja's algorithm [16]. It can be viewed as the traditional stochastic gradient decent method, in which each data point that come into memory can be viewed as a random sample drawn from an underlying distribution. In 2013, Balsubramani et al. [3] modified Oja's algorithm with a special form of step size and provided the statistical analysis of this algorithm. Later on in 2015, Jin et al. [8] proposed a faster algorithm to find the top eigenvector based on shift and invert framework with improved sample complexity. Although this algorithm seem to have good sample complexity, it requires the initialization vector be within certain distance to the true top eigenvector, which is costly to attain and hard to achieve when the dimension d is very high. More recently, Jain et al. [7] proved that Oja's algorithm can achieve the optimal sample complexity for the first eigenvector under certain choices of step size, in the sense that it matches the matrix Bernstein inequality [21, 20]. However, the step sizes depend on some data-related constants that cannot be estimated beforehand. So in practice one still needs to test different step sizes and choose the best one. Under a similar assumption on the step size, Li et al. [11] recently proved that the streaming PCA algorithm matches the minimax information rate, and Allen-Zhu and Li [2] extended such optimality to the k-subspace version and proposed the Oja^{++} algorithm. Recently, Sa et al. [17] showed the global convergence of Alecton algorithm, which is another variant of stochastic gradient descent algorithm applied to a non-convex low-rank factorized problem. In our experiments, we compare our algorithms with Oja's algorithm and Oja^{++} algorithm based on various choices of step sizes and show that our algorithm outperforms all of them.

Mitliagkas et al. [15] introduced the block stochastic power method. This method uses each block of data to construct an estimator of the covariance matrix and applies power method in each iteration to update eigenvectors. Theoretical guarantees are provided under the setting of the spiked data model. More recently, Li et al. [10] extended the work of [15] by using blocks of variable size and progressively increasing the block size as the estimate becomes more accurate. Later on, Hardt and Price [6] gave a robust convergence analysis of a more general version of block stochastic power method for arbitrary distributions.

The aforementioned works are all in the setting of streaming i.i.d. data samples. There is another line of research for an arbitrary sequence of data. Most works [12, 5, 4, 14, 13] use the technique of computing a sketch of matrix to find an estimator of the top eigenvector.

Our algorithm is under the setting of streaming i.i.d. data samples and it differs from other streaming algorithms in the sense that our History PCA algorithm exploits key summaries in the historical data to achieve better performance.

3 Problem Setting and Background

Let $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N$ be a sequence of d-dimensional data that stream into memory. Assume all the x_i's are sampled i.i.d from a distribution with zero-mean and a fixed covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$. Let $\lambda_1 > \lambda_2 \geq \lambda_3 \cdots \geq \lambda_d$ and $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_d$ denote the eigenvalues and corresponding eigenvectors of Σ.

In this paper, our goal is to recover the top k eigenvectors. In the case of $k = 1$, our goal is to find a unit vector that is within the ε-neighborhood of the first true eigenvector v_1. In the general case with $k > 1$, we use the the largest-principal-angle-based distance function as the metric to evaluate our algorithm, which is

$$d(span(U), span(V)) = d(U, V) = \|U_\perp^\top V\|_2 = \|V_\perp^\top U\|_2 \tag{1}$$

for any $U, V \in \mathbb{R}^{d\times k}$. Here $U_\perp$ denotes the $d \times (d-k)$ orthogonal basis of the perpendicular subspace to the subspace spanned by $d \times k$ matrix U.

Our algorithm is motivated by the block stochastic power method [15]. While the classical power method keeps on multiplying the vector with a fixed sample covariance matrix at each iteration ($\boldsymbol{w}_i = \frac{1}{N}X^\top X\boldsymbol{w}_{i-1}$), the block stochastic power method [15] updates the current estimate by

$$\boldsymbol{w}_i \leftarrow \frac{1}{B}X_i^\top X_i \boldsymbol{w}_{i-1} \tag{2}$$

where $\frac{1}{B}X_i^\top X_i$ is a sample covariance matrix formed by the block of B data points that stream into memory at iteration i. Thus, the block stochastic power method can be viewed as a stochastic variant of the power method, and using a larger block size at each iteration can reduce the potentially large variance by averaging out the noises.

The block stochastic power method has several limitations. First, the block size B needs to be very large in order for this algorithm to work. That is mainly because the algorithm requires the sample covariance matrix formed by each block of data points to be very close to the true covariance matrix, which is difficult to achieve if the block size B is not large enough. And this block size B depends highly on the target accuracy ε. Secondly, the block stochastic power method cannot converge to the true eigenvectors of Σ. If the block size B is fixed, it can only converge to the ε-approximated solution, where $\varepsilon = O(1/B)$.

Our algorithm aims at solving the above problems by using more past information. Instead of doing only one matrix-vector multiplication at each step after a block of data stream in, we do matrix-vector multiplication iteratively either until the algorithm converges or for a pre-specified number of times. This strategy gives us good results at very early stage. Moreover, with the use of past information, we could further reduce the variance of the estimates. Since intuitively there is no reason to favor a particular past block, we assign equal weights to all data samples (or blocks) in each iteration. Details are provided below.

4 Proposed Algorithms

Since data come into memory sequentially, we can form blocks of sample, each with size B, as data streaming in, and denote the ith data block by a $B \times d$ matrix $X_i = (\boldsymbol{x}_{i1}, \boldsymbol{x}_{i2}, \ldots, \boldsymbol{x}_{iB})^T$, where x_{it} is a data point of dimension d for $t = 1, \cdots, B$.

4.1 Main algorithm: History PCA (Rank-1)

We first provide the method and intuition of our History PCA in the rank-1 case.

Algorithm 1 History PCA: k = 1

```
Input: $\{X_1, \dots, X_n\}$, block size: $B$.
$\boldsymbol{w}_0 \sim N(0, I_{d\times d})$. $\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_0/\|\boldsymbol{w}_0\|_2$
while $\boldsymbol{w}_1$ not converge do
    $\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_1 + \frac{1}{B}X_1^\top X_1 \boldsymbol{w}_1$
    $\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_1/\|\boldsymbol{w}_1\|_2$
end while
for $\tau = 2, \dots, n$ do
    $\boldsymbol{w}_\tau = \boldsymbol{w}_{\tau-1}$
    while $\boldsymbol{w}_\tau$ not converge do
        $\boldsymbol{w}_\tau \leftarrow \frac{\tau-1}{\tau}\boldsymbol{w}_{\tau-1}\boldsymbol{w}_{\tau-1}^\top \boldsymbol{w}_\tau + \frac{1}{\tau}\frac{1}{B}X_\tau^\top X_\tau \boldsymbol{w}_\tau$
        $\boldsymbol{w}_\tau \leftarrow \boldsymbol{w}_\tau/\|\boldsymbol{w}_\tau\|_2$
    end while
end for
Output $\boldsymbol{w}_n$
Extension: the "while" loops can be run by $m$ iterations.
```

At time 1, we have the first block of data X_1. Since we have no past information, we can make use of the sample covariance matrix formed by the first block of data point, $\frac{1}{B}X_1^\top X_1 = \frac{1}{B}\sum_{t=1}^{B} \boldsymbol{x}_{1t}\boldsymbol{x}_{1t}^\top$, to estimate the true covariance matrix and its eigenvectors. But instead of finding the eigenvectors of $\frac{1}{B}X_1^\top X_1$, we try to find the eigenvectors of $(I + \frac{1}{B}X_1^\top X_1)$ in the first step, which is mainly for our convergence theory to go through. In the rank-1 case, we only need to save the first eigenvector $\boldsymbol{w}_1$ of $(I + \frac{1}{B}X_1^\top X_1)$.

At time 2, we have the second block of data X_2. If we could have both blocks of data in memory, then the best we can do is to form the sample covariance matrix $\frac{1}{2}(\frac{1}{B}X_2^\top X_2 + \frac{1}{B}X_1^\top X_1)$. However, limited memory space prohibits us from storing the entire first block of data. Thus, we need to find an alternative scheme to extract key information from the first block so that the estimator at time 2 could be better than just using the sample covariance matrix $\frac{1}{B}X_2^\top X_2$. For simplicity of illustration we assume $\lambda_1 = 1$. An intuitive idea is to form a rank-1 matrix $\lambda_1 \boldsymbol{w}_1\boldsymbol{w}_1^\top = \boldsymbol{w}_1\boldsymbol{w}_1^\top$ and make use of it in our estimator. Intuitively, we should assign equal weights to the information we get from both blocks of data. Thus, the updated estimator is $\frac{1}{2}(\frac{1}{B}X_2^\top X_2 + \boldsymbol{w}_1\boldsymbol{w}_1^\top)$. Now we only need to save the first eigenvector $\boldsymbol{w}_2$ of this estimator.

At time τ, we have the τth block of data X_τ. Similarly, we could get a new estimator of the covariance matrix by exploiting the history from the past $\tau - 1$ blocks of data. Intuitively, each block of data should have equal weights. Thus, we assign the weight $\frac{1}{\tau}$ and $\frac{\tau-1}{\tau}$, respectively, to the sample covariance matrix formed by X_τ and the rank-1 matrix formed by the past $\tau - 1$ blocks of data. Therefore, the new covariance estimator at time τ is

$$\frac{1}{\tau}\frac{1}{B}X_\tau^\top X_\tau + \frac{\tau-1}{\tau}\boldsymbol{w}_{\tau-1}\boldsymbol{w}_{\tau-1}^\top, \tag{3}$$

where $\boldsymbol{w}_{\tau-1}$ is the eigenvector at time $\tau - 1$.

At the final time n, we have seen in total n blocks of data and we output $\boldsymbol{w}_n$. We summarize this algorithm in Algorithm 1.

4.2 Solving each subproblem approximately

In Algorithm 1, at each iteration τ we run the power method on the current covariance estimator, which is equation (3), to get the first eigenvector. This can be viewed as a subproblem. However, it will in theory require infinite number of iterations to get the exact eigenvector. Simulations show that using exact convergence have similar results as using power method with small m iterations. So instead of solving the sub-problem exactly, we solve it approximately by performing matrix vector multiplication m times in the for loops. Note that the space complexity of our algorithm is $O(Bd)$. However, when we set $m = 1$ (only one power iteration at each step), Algorithm 1 will only require $O(d)$ memory. This gives us the flexibility to reduce the memory size to $O(d)$ if memory is not enough.

4.3 Extending to rank-k History PCA

Our method can be generalized to find the first k eigenvectors, as shown in Algorithm 2. In the rank-k case, we can simply replace the d-dimensional vector $\boldsymbol{w}$ with a $d \times k$ matrix Q and then replace the normalization update by the QR decomposition.

In the rank-k case, we make use of the history information of eigenvalues. So the new covariance estimator at time τ, which we utilize to update our estimate, is

$$\frac{1}{\tau}\frac{1}{B}X_\tau^\top X_\tau + \frac{\tau-1}{\tau}Q_{\tau-1}\Lambda_{\tau-1}Q_{\tau-1}^\top \tag{4}$$

where $Q_{\tau-1}$ is a matrix of k eigenvectors and $\Lambda_{\tau-1}$ is a $k \times k$ diagonal matrix with corresponding eigenvalues as the diagonal elements at time $\tau - 1$.

Similarly, in the generalized History PCA algorithm, we can replace the exact solver by an approximate solver (with 1 or a fixed number of iterations m) to find Q_τ. We can further reduce the memory complexity of our algorithm from $O((k+B)d)$ to $O(kd)$ when we set $m = 1$.

Algorithm 2 History PCA: $k \geq 1$

```
Input: $\{X_1, \dots, X_n\}$, block size: $B$.
$H^i \sim N(0, I_{d\times d}), 1 \leq i \leq k$. $H = Q_1 R_1$ (QR-decomposition)
while $Q_1$ not converge do
    $S_1 \leftarrow Q_1 + \frac{1}{B}X_1^\top X_1 Q_1$
    $S_1 = Q_1 R_1$ (QR-decomposition)
end while
$\lambda_j = \|S_1[:j]\|_2$ for $j = 1, \cdots, k$.
$\Lambda_1 = diag(\lambda_1, \cdots, \lambda_k)$
for $\tau = 2, \dots, n$ do
    $Q_\tau = Q_{\tau-1}$
    while $Q_\tau$ not converge do
        $S_\tau \leftarrow \frac{\tau-1}{\tau}Q_{\tau-1}\Lambda_{\tau-1}Q_{\tau-1}^\top Q_\tau + \frac{1}{\tau}\frac{1}{B}X_\tau^\top X_\tau Q_\tau$
        $S_\tau = Q_\tau R_\tau$ (QR-decomposition)
    end while
    $\lambda_j = \|S_\tau[:j]\|_2$ for $j = 1, \cdots, k$.
    $\Lambda_\tau = diag(\lambda_1, \cdots, \lambda_k)$
end for
Output $Q_\tau$
Extension: the "while" loops can be run by $m$ iterations.
```

4.4 Theoretical Analysis

Now we provide the convergence theorem for the proposed algorithm. Let $A_i = \frac{1}{B}X_i^\top X_i \in \mathbb{R}^{d\times d}$ be the sample covariance matrix formed by i-th block of data, for $i = 1, \cdots, n$, that satisfies the following assumptions:

- $E[A_i] = \Sigma$, where $\Sigma \in \mathbb{R}^{d\times d}$ is a symmetric PSD matrix.
- $\|A_i - \Sigma\|_2 \leq M$ with probability 1.
- $\max\{\|E[(A_i-\Sigma)(A_i-\Sigma)^\top]\|_2, \|E[(A_i-\Sigma)^\top(A_i-\Sigma)]\|_2\} \leq \mathcal{V}$.

Let $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_d$ denote the eigenvectors of Σ and $\lambda_1 > \lambda_2 \geq \lambda_3 \cdots \geq \lambda_d$ denote the corresponding eigenvalues. Our goal is to compute an ε-approximation to $\boldsymbol{v}_1$ which is a unit vector $\boldsymbol{w}$ satisfying $\sin^2(\boldsymbol{w}, \boldsymbol{v}_1) = 1 - (\boldsymbol{w}^\top \boldsymbol{v}_1)^2 \leq \varepsilon$, where $\sin(\boldsymbol{w}, \boldsymbol{v}_1)$ denotes the sin of the angle between $\boldsymbol{w}$ and $\boldsymbol{v}_1$. In the following, we prove that our History PCA algorithm can achieve ε accuracy with

$$\varepsilon = O\left(\frac{\mathcal{V}}{|2(\lambda_1 - \lambda_2) - 1|}\frac{1}{n}\right) \tag{5}$$

after seeing n data blocks.

Jain et al. [7] proved the convergence of their algorithm by directly analyzing the convergence of Oja's algorithm as an operator on the initialized vector rather than analyzing the error reduced after each update. Oja's algorithm applies the matrix

$$B_n = (I + \eta_n A_n)(I + \eta_{n-1} A_{n-1}) \cdots (I + \eta_1 A_1)$$

to the random initial vector, say $\boldsymbol{w}_0$, and then output the normalized result

$$\boldsymbol{w}_n = \frac{B_n \boldsymbol{w}_0}{\|B_n \boldsymbol{w}_0\|_2}.$$

Inspired by their approach, we apply the following trick to prove the theoretical guarantees of our algorithm. If we set $m = 1$, our algorithm can also be viewed as applying B_n on the initialized vector $\boldsymbol{w}_0$. In our algorithm, we have $\eta_1 = 1$ and $\eta_i = \frac{1}{i-1}$ for $i > 1$. We summarize the main convergence theorem for the History PCA algorithm here. The detailed proof can be found in the appendix.

Theorem 1. *Fix any $\delta > 0$. Let n_0 be the smallest integer such that $n_0 > \max(4\mathcal{M}, 18\frac{\mathcal{V}+\lambda_1^2}{\log(1+\frac{\delta}{100})})$. Let $n > n_0$. Under the assumption that B_n is invertible, the History PCA algorithm (rank-1 with $m = 1$) converges to an ε-accurate solution with*

$$\varepsilon_n = O\left(\frac{\mathcal{V}}{|2(\lambda_1 - \lambda_2) - 1|} \frac{1}{n}\right), \tag{6}$$

with probability at least $1 - \delta$.

To prove our main Theorem 1, we need to introduce the following lemma.

Lemma 1. *Assume $B_n \in \mathbb{R}^{d \times d}$ is invertible for any positive integer n. Let n_0 be a non-negative integer. Let $C_{n,n_0} \in \mathbb{R}^{d \times d}$ be such that $B_{n+n_0} = C_{n,n_0} B_{n_0}$ for any positive integer n. Let $\boldsymbol{v} \in \mathbb{R}^d$ be a unit vector, and let $V_\perp$ be a matrix whose columns form an orthonormal basis of the subspace orthogonal to $\boldsymbol{v}$. If $\boldsymbol{w} \in \mathbb{R}^d$ is chosen uniformly at random from the surface of the unit sphere, then with probability at least $1 - \delta$,*

$$\sin^2\left(\boldsymbol{v}, \frac{B_{n+n_0}\boldsymbol{w}}{\|B_{n+n_0}\boldsymbol{w}\|_2}\right) \le \frac{c \log(1/\delta)}{\delta} \frac{Tr(V_\perp^\top C_{n,n_0} C_{n,n_0}^\top V_\perp)}{\boldsymbol{v}^\top C_{n,n_0} C_{n,n_0}^\top \boldsymbol{v}},$$

where c is an absolute constant.

Define the step sizes $\hat{\eta}_t = \eta_{t+n_0} = \frac{1}{n_0+t}$ for all positive integer t. We can see that

$$\begin{aligned} C_{n,n_0} &= (I + \eta_{n_0+n} A_{n_0+n}) \cdots (I + \eta_{n_0+1} A_{n_0+1}) \\ &= (I + \hat{\eta}_n A_{n_0+n}) \cdots (I + \hat{\eta}_1 A_{n_0+1}). \end{aligned}$$

Applying C_{n,n_0} on $\boldsymbol{w}$ gives us

$$\sin^2\left(\boldsymbol{v}, \frac{C_{n,n_0}\boldsymbol{w}}{\|C_{n,n_0}\boldsymbol{w}\|_2}\right) \le \frac{c_1 \log(1/\delta)}{\delta} \frac{Tr(V_\perp^\top C_{n,n_0} C_{n,n_0}^\top V_\perp)}{\boldsymbol{v}^\top C_{n,n_0} C_{n,n_0}^\top \boldsymbol{v}},$$

where c_1 is some absolute constant. Therefore, Lemma 1 can be interpreted as a relationship between the error bound after $n + n_0$ iterations of Oja's algorithm and the error bound of applying the last n iterations on a randomly chosen initial vector if we totally dump the result from the first n_0 iterations. With Lemma 1 and a carefully selected constant integer n_0, we can prove Theorem 2, which implies Theorem 1 directly.

Theorem 2. *Fix any $\delta > 0$. Let n_0 be the smallest integer such that $n_0 > \max(4\mathcal{M}, 18\frac{\mathcal{V}+\lambda_1^2}{\log(1+\frac{\delta}{100})})$. Assume B_{n+n_0} is invertible, the output $\boldsymbol{w}_{n+n_0}$ of the History PCA algorithm (rank-1 with $m = 1$) satisfies:*

$$1 - (\boldsymbol{w}_{n+n_0}^\top v_1)^2 = O\left(\frac{\mathcal{V}}{|2(\lambda_1 - \lambda_2) - 1|} \frac{1}{n + n_0}\right),$$

with probability at least $1 - \delta$.

5 Experimental Results

In this section we conduct numerical experiments to compare our algorithm with previous streaming PCA approaches using synthetic streaming data as well as real world data listed in Table 1.

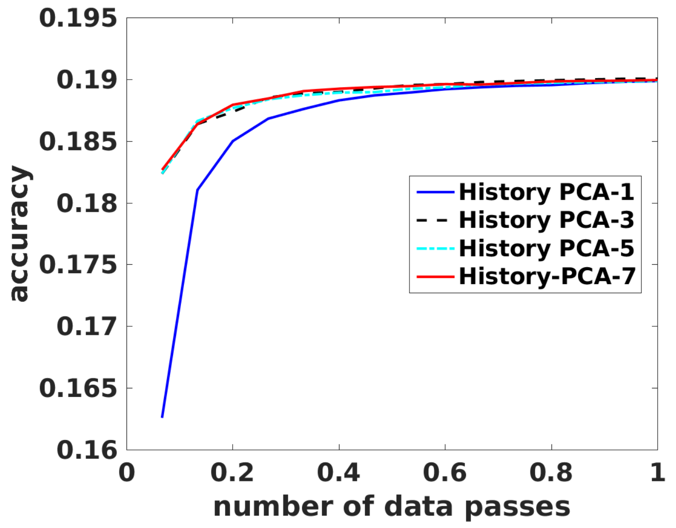


Figure 1: The choice of m in History PCA on NIPS data set

5.1 Choice of m for History PCA

In Figure 1, we compare the performances of our History PCA algorithm with different choices of m (number of inner iterations) on NIPS data set with $k = 1$. Here the accuracy is the explained variance of the algorithm output. We can see that when $m = 3, 5, 7$, History PCA's performances are equally well. And when $m = 1$, its performance is slightly worse. Thus, a small m is already good enough for our algorithm. In the following experiments, we will just set $m = 3$ in our algorithm to compare with other existing streaming PCA methods.

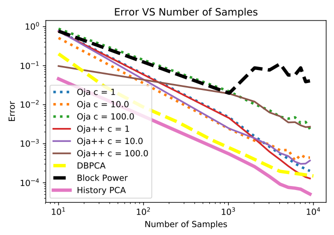
5.2 Simulated Streaming Data

In this section, we compare the proposed History PCA algorithm with state-of-the-art streaming PCA algorithms: block power method, DBPCA method, Oja's algorithm and Oja^{++} algorithm.

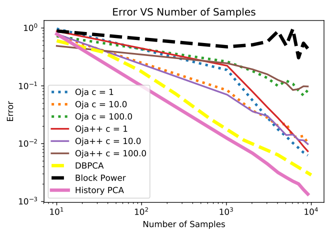
We set Oja's algorithm and Oja^{++} algorithm with step sizes $\frac{c}{t}$, where t is the number of iterations and c is the tuning parameter. We obtain the results for $c = 10^j, j = -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4$, but only include the best three c values in Figure 2 and Figure 3. The same experimental setting in [15] is used in order to compare our methods with theirs. Here we tried $n = 10000$ and varied the noise standard deviation $\sigma = 0.1, 0.5, 0.8$ for the simulated data sets. Thus, $\Sigma = UU^\top + \sigma^2 I$, where U is the $d \times k$ true orthonormal matrix we are trying to recover.

We consider two scenarios: $d = 100$ and $d = 1000$. For each scenario, we try to recover the top $k = 1, 5, 10$ eigenvectors of the data sets respectively. We also vary the block size $B = 10, 100$. The performances of the algorithms are evaluated based on the largest-principal-angle metric, which measures the distances between the estimated principal vectors and the ground truth.

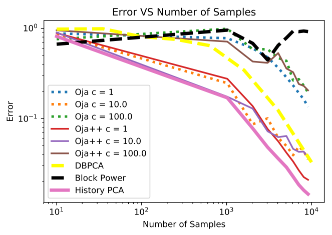
Figure 2 shows the results for $d = 100$ and Figure 3 shows the results for $d = 1000$. Both figures suggest that the History PCA method is the best for the case of recovering the top eigenvector, as well as for cases recovering additional top eigenvectors, regardless of the block size, noise level, and whether d is large or small. What's more, the error from the History PCA method continues to decrease as the sample size of streaming data increases, while some other streaming methods stop improving at a very early stage. From the simulations, we conclude that the performance of the block power method depends highly on the block size B. The performance of the block power method improves as B increases. But the History PCA method does not depend on the block size, which is an advantage over the block power method. DBPCA's performance is sometimes much better than the block power method and outperforms the Oja's algorithm and Oja^{++} algorithm in most scenarios. But it is less stable and less accurate than our History PCA method. The performance of Oja's algorithm and Oja^{++} algorithm depends highly on the tuning of step size. There is no universal good step size for Oja's algorithm and Oja^{++} algorithm. The History PCA method does not need to tune the step size but yields superior performance than those of the best tuned Oja's algorithm and Oja^{++} algorithm in all scenarios.



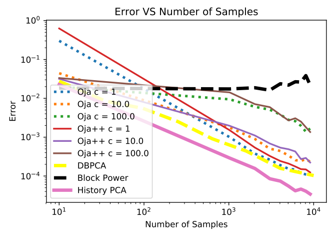
(a) $d = 100, B = 10, k = 1, sig = 0.1$



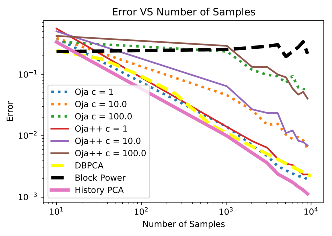
(b) $d = 100, B = 10, k = 1, sig = 0.5$



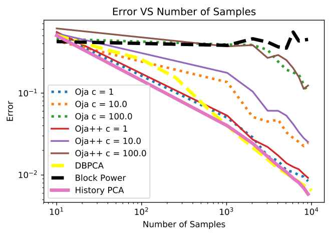
(c) $d = 100, B = 10, k = 1, sig = 0.8$



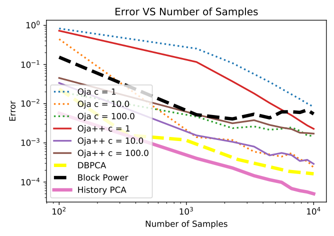
(d) $d = 100, B = 10, k = 5, sig = 0.1$



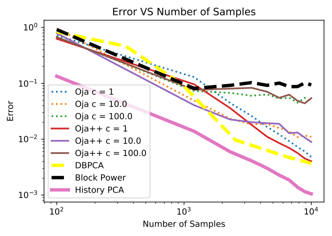
(e) $d = 100, B = 10, k = 5, sig = 0.5$



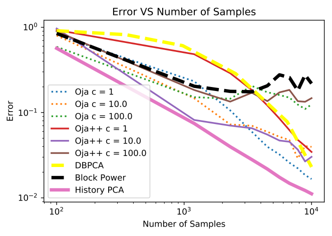
(f) $d = 100, B = 10, k = 5, sig = 0.8$



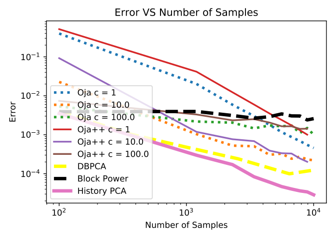
(g) $d = 100, B = 100, k = 1, sig = 0.1$



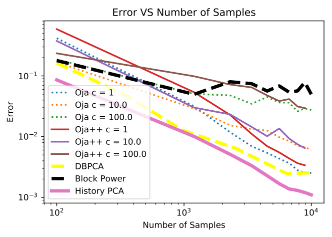
(h) $d = 100, B = 100, k = 1, sig = 0.5$



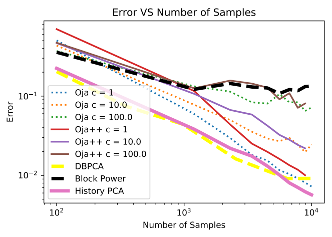
(i) $d = 100, B = 100, k = 1, sig = 0.8$



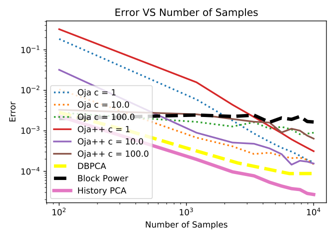
(j) $d = 100, B = 100, k = 5, sig = 0.1$



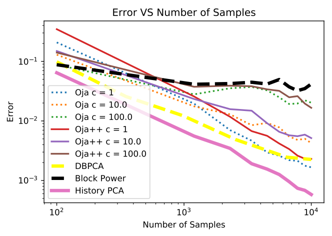
(k) $d = 100, B = 100, k = 5, sig = 0.5$



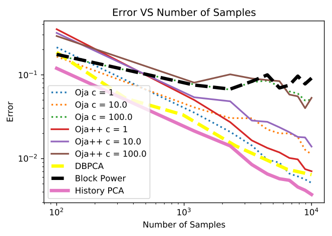
(l) $d = 100, B = 100, k = 5, sig = 0.8$



(m) $d = 100, B = 100, k = 10, sig = 0.1$

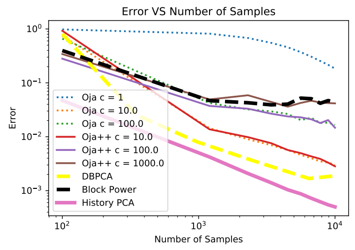


(n) $d = 100, B = 100, k = 10, sig = 0.5$

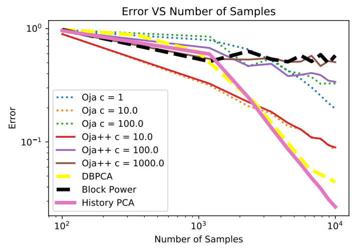


(o) $d = 100, B = 100, k = 10, sig = 0.8$

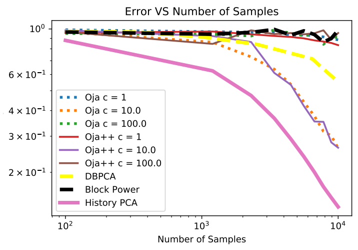
Figure 2: Comparison of streaming PCA algorithms on simulated data sets (d = 100).



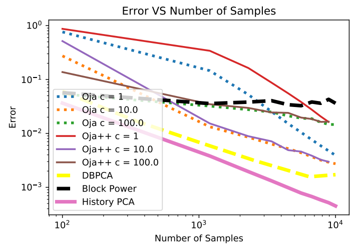
(a) $d = 1000, B = 100, k = 1, sig = 0.1$



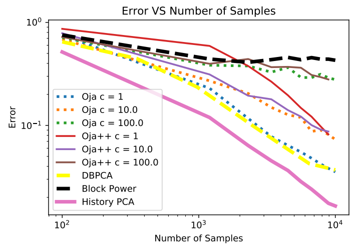
(b) $d = 1000, B = 100, k = 1, sig = 0.5$



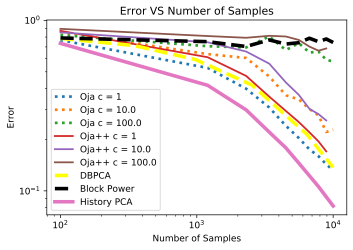
(c) $d = 1000, B = 100, k = 1, sig = 0.8$



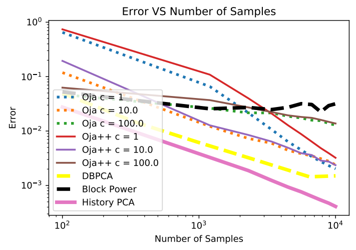
(d) $d = 1000, B = 100, k = 5, sig = 0.1$



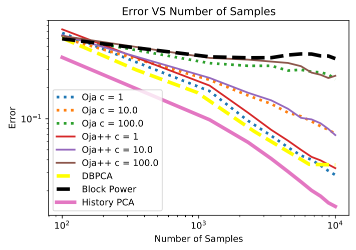
(e) $d = 1000, B = 100, k = 5, sig = 0.5$



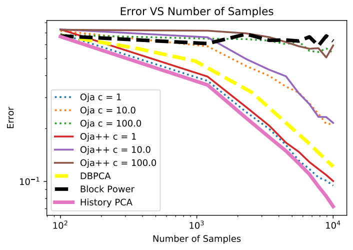
(f) $d = 1000, B = 100, k = 5, sig = 0.8$



(g) $d = 1000, B = 100, k = 10, sig = 0.1$



(h) $d = 1000, B = 100, k = 10, sig = 0.5$



(i) $d = 1000, B = 100, k = 10, sig = 0.8$

Figure 3: Comparison of streaming PCA algorithms on simulated data sets (d = 1000).

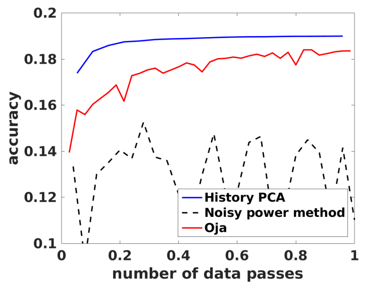
5.3 Large-scale real data

	# samples	# features	# nonzeroes
NIPS	1500	12419	746,316
NYTimes	300,000	102,660	69,679,427
RCV1	677,399	47,236	49,556,258
KDDB	19,264,097	29,890,095	566,345,888

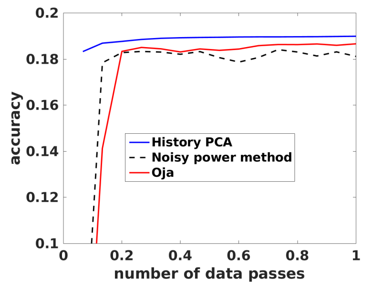
Table 1: Data set statistics

We compare our algorithm with Oja's algorithm, VR-PCA and noisy power method [15, 6] on four real data sets. We do not include the Oja^{++} algorithm as we observe that its performance is almost identical to the Oja's algorithm. NIPS and NYTimes data sets are downloaded from UCI data. RCV1 and KDDB data sets are downloaded from LIBSVM data sets. A summary of our data sets is provided in Table 1.

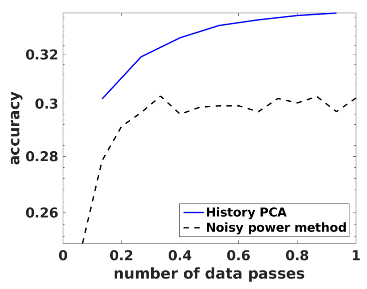
We first consider the streaming setting, where the algorithms can only go through the data once. For a fair comparison, we set Oja's algorithm with step sizes $\frac{c}{t}$, where t is number of iterations and c ranges from 10^{-6} to 10^4 in all the experiments. The best result of the Oja's algorithm is presented in the plots. We also compare the algorithms with different block size B and at different target rank k. In practice we find Oja's algorithm ($k > 1$) often numerically unstable and cannot achieve the comparable performance with other approaches, so we do not



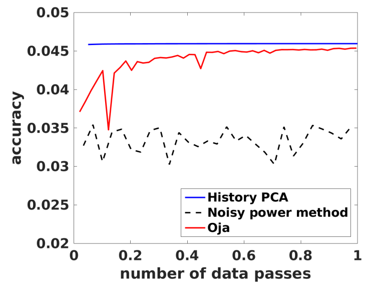
(a) NIPS, $B = 10$, $k = 1$



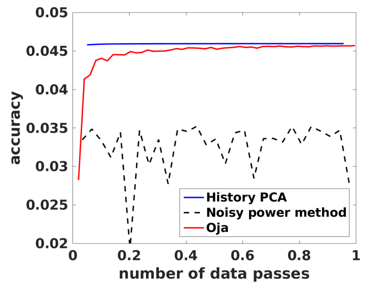
(b) NIPS, $B = 100$, $k = 1$



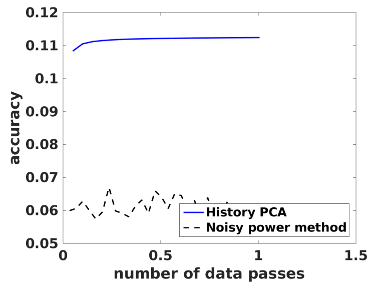
(c) NIPS, $B = 100$, $k = 10$



(d) NYTimes, $B = 10$, $k = 1$



(e) NYTimes, $B = 100$, $k = 1$



(f) NYTimes, $B = 100$, $k = 10$

Figure 4: Comparison of streaming PCA algorithms on real data sets (streaming setting).

include their result for $k > 1$. To evaluate the results, we use the widely used metric "explained variance",

$$\frac{\text{trace}(W^\top X^\top X W)}{\|X\|_F^2},$$

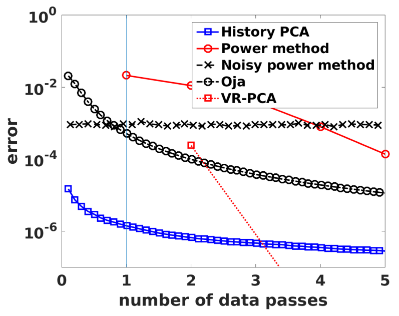
to measure the quality of the solution (where W is the computed solution). Note that the value is always less or equal to 1, and will be maximized when W is the leading eigenvectors of $X^\top X$. In Figure 4, we observe that our algorithm (History PCA) is very stable and consistently better in all settings (big or small B, big or small k).

5.4 Comparing streaming with non-streaming algorithms

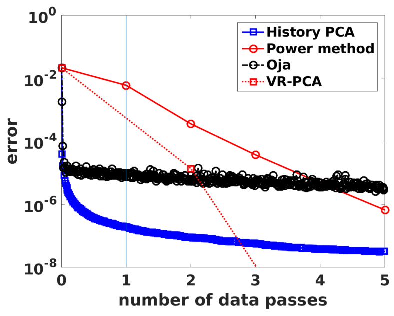
In the final set of experiments, we go beyond the streaming setting and allow several passes of data. We want to test (1) In addition to having fewer number of data access, can our algorithm also have short running time in practice? (2) If we go beyond the first pass of the data, can our algorithm keep on improving the solution by making more passes? (3) In practice, if there is a large data set with millions or billions of data points and we want to compute PCA, should we use our algorithm instead of other batch PCA algorithms?

In order to answer these questions, we test our algorithm on two large data sets: RCV1 and KDDB (with 20 million samples and 30 million features). Also, we add two strong batch algorithms into comparison, including power method and VR-PCA [18], which are the state-of-the-art PCA solvers. In order to clearly show the convergence speed, we compute the "error" by the largest eigenvalue minus the unnormalized explained variance $\boldsymbol{w}^\top X^\top X \boldsymbol{w}$, so ideally the error will converge to 0 (we are doing this in order to show the error in log-scale in the y-axis). The results are presented in Figure 5. In Figure 5a and 5b, we show the number of data passes versus errors, and the vertical line means the end of the first data pass. We observe that even within the first data pass, the error of our algorithm can be bounded by 10^{-6} and after the first pass our algorithm is still able to improve the solution. We also observe that VR-PCA achieves better performance with more passes of data and sufficient time while we argue that an error of order 10^{-6} is accurate enough for practical use.

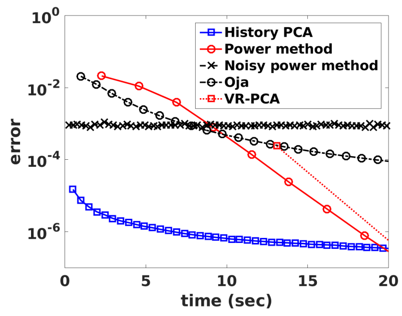
Finally we compare the real run time with all the other algorithms. In Figure 5c, since the data set is not that large, we store the data set in memory and perform in-memory data access, while for KDDB data (Figure 5d), since the data size is larger than memory, we load data block by block and compare the overall running time. The results show that our algorithm is still much better than other methods in terms of run time. The only real competitor,



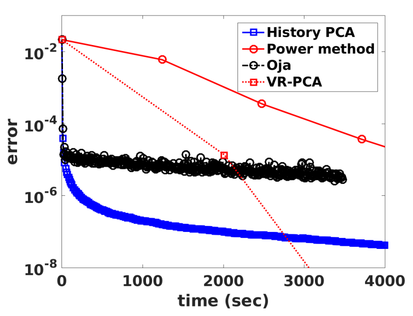
(a) RCV1 $k = 1$, data passes



(b) KDDB $k = 1$, data passes



(c) RCV1 $k = 1$, time



(d) KDDB $k = 1$, time

Figure 5: Comparison of all batch and streaming PCA algorithms on large data sets with respect to number of data access (a, b) and run time (c, d).

again, is VR-PCA, which will catch up our algorithm when reaching an error of order 10^{-7}. In the future, it will be interesting to apply a variance reduction technique to speed up the convergence of our algorithm after the first pass of data.

6 Conclusions

We propose History PCA, a hyperparameter-free streaming PCA algorithm which utilizes past information effectively. We extend our algorithm to solve the rank-k PCA problems with $k > 1$. What's more, we provide the theoretical guarantees and convergence rate for our algorithm, and we show that the convergence speed is faster than existing algorithms on both synthetic and real data sets.

References

[1] R. Adamczak, A. Litvak, A. Pajor, and N. Tomczak-Jaegermann. 2010. Quantitative estimates of the convergence of the empirical covariance matrix in log-concave ensembles. *Journal of the American Mathematical Society* 234 (2010), 535–561.

[2] Z. Allen-Zhu and Y. Li. 2017. First Efficient Convergence for Streaming k-PCA: a Global, Gap-Free, and Near-Optimal Rate. *arXiv:1607.07837* (2017).

[3] A. Balsubramani, S. Dasgupta, and Y. Freund. 2013. The fast convergence of incremental pca. *Advances in Neural Information Processing Systems* (2013), 3174–3182.

[4] Peilin Zhong Christos Boutsidis, David P. Woodruff. 2016. Optimal Principal Component Analysis in Distributed and Streaming Models. *in Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 236–249.

[5] Zohar Karnin Edo Liberty Christos Boutsidis, Dan Garber. 2015. Online Principal Components Analysis. *in Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, 887–901.

[6] M. Hardt and E. Price. 2014. The noisy power method: A meta algorithm with applications. In *NIPS*.

[7] P. Jain, C. Jin, S. Kakade, P. Netrapali, and A. Sidford. 2016. Streaming PCA: Matching Matrix Bernstein and Near-Optimal Finite Sample Guarantees for Oja's Algorithm. In *COLT*.

[8] C. Jin, S. M. Kakade, C. Musco, P. Netrapalli, and A. Sidford. 2015. Robust shift-and-invert preconditioning: Faster and more sample efficient algorithms for eigenvector computation. *arXiv:1510.08896* (2015).

[9] R. Johnson and T. Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. *Conference on Neural Information Processing Systems* (2013).

[10] C. Li, H. Lin, and C. Lu. 2015. Rivalry of Two Families of Algorithms for Memory-Restricted Streaming PCA. *arXiv:1506.01490* (2015).

[11] C. Li, M. Wang, H. Liu, and T. Zhang. 2017. Near-Optimal Stochastic Approximation for Online Principal Component Estimation. *arXiv:1603.05305* (2017).

[12] Edo Liberty. 2013. Simple and Deterministic Matrix Sketching. *in Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 581–588.

[13] Jeff M. Phillips Mina Ghashami. 2014. Relative Errors for Deterministic Low-Rank Matrix Approximations. *in Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (2014), 707–717,.

[14] Jeff M. Phillips David P. Woodruff Mina Ghashami, Edo Liberty. 2015. Frequent Directions: Simple and Deterministic Matrix Sketching. *SIAM J. Comput.* 45, 5 (2015), 1762–1792.

[15] I. Mitliagkas, C. Caramanis, and P. Jain. 2013. Memory Limited, Streaming PCA. In *NIPS*.

[16] E. Oja. 1982. On the distribution of the largest eigenvalue in principal components analysis. *Journal of Mathematical Biology* 15, 3 (1982), 267–273.

[17] Christopher De Sa, Kunle Olukotun, and Christopher Re. 2015. Global convergence of stochastic gradient descent for some nonconvex matrix problems. In *ICML*.

[18] O. Shamir. 2015. A stochastic PCA and SVD algorithm with an exponential convergence rate. In *ICML*.

[19] O. Shamir. 2016. Fast Stochastic Algorithms for SVD and PCA: Convergence Properties and Convexity. In *ICML*.

[20] Joel A Tropp. 2012. User-friendly tail bounds for sums of random matrices. *Foundations of computational mathematics* 12, 4 (2012), 389–434.

[21] Roman Vershynin. 2010. Introduction to the non-asymptotic analysis of random matrices. *arXiv:1011.3027* (2010).

A Proof of Lemma 2

Proof. Since $B_n = (I + \eta_n A_n)\cdots(I + \eta_1 A_1)$ with $B_0 = I$ and $B_{n+n_0} = C_{n,n_0}B_{n_0}$, we know that $C_{n,n_0} = (I + \eta_{n_0+n}A_{n_0+n})\cdots(I + \eta_{n_0+1}A_{n_0+1})$. By Lemma 6 of [7], we know that with probability at least $1-\delta$,

$$\begin{aligned}\sin^2(\boldsymbol{v}, \frac{B_{n+n_0}\boldsymbol{w}}{\|B_{n+n_0}\boldsymbol{w}\|_2}) &= 1 - (\frac{\boldsymbol{v}^\top B_{n+n_0}\boldsymbol{w}}{\|B_{n+n_0}\boldsymbol{w}\|_2})^2 \\ &\le \frac{c_1\log(1/\delta)}{\delta}\frac{Tr(V_\perp^\top B_{n+n_0}B_{n+n_0}^\top V_\perp)}{\boldsymbol{v}^\top B_{n+n_0}B_{n+n_0}^\top \boldsymbol{v}} \\ &\le \frac{c_1\log(1/\delta)}{\delta}\frac{Tr(V_\perp^\top C_{n,n_0}B_{n_0}B_{n_0}^\top C_{n,n_0}^\top V_\perp)}{\boldsymbol{v}^\top C_{n,n_0}B_{n_0}B_{n_0}^\top C_{n,n_0}^\top \boldsymbol{v}},\end{aligned}$$

where c_1 is an absolute constant. Since

$$\begin{aligned}Tr(V_\perp^\top C_{n,n_0}B_{n_0}B_{n_0}^\top C_{n,n_0}^\top V_\perp) &= Tr(B_{n_0}B_{n_0}^\top C_{n,n_0}^\top V_\perp V_\perp^\top C_{n,n_0}) \\ &\le \|B_{n_0}B_{n_0}^\top\|_2 Tr(C_{n,n_0}^\top V_\perp V_\perp^\top C_{n,n_0}),\end{aligned}$$

$$\begin{aligned}\boldsymbol{v}^\top C_{n,n_0}B_{n_0}B_{n_0}^\top C_{n,n_0}^\top \boldsymbol{v} &= Tr(B_{n_0}B_{n_0}^\top C_{n,n_0}^\top \boldsymbol{v}\boldsymbol{v}^\top C_{n,n_0}) \\ &\ge \sigma_n(B_{n_0}B_{n_0}^\top)Tr(C_{n,n_0}^\top \boldsymbol{v}\boldsymbol{v}^\top C_{n,n_0}) \\ &\ge \sigma_n(B_{n_0}B_{n_0}^\top)\boldsymbol{v}^\top C_{n,n_0}C_{n,n_0}^\top \boldsymbol{v},\end{aligned}$$

we have

$$\begin{aligned}\sin^2(\boldsymbol{v}, \frac{B_{n+n_0}\boldsymbol{w}}{\|B_{n+n_0}\boldsymbol{w}\|_2}) &\le \frac{c_1\log(1/\delta)}{\delta}\frac{Tr(V_\perp^\top C_{n,n_0}B_{n_0}B_{n_0}^\top C_{n,n_0}^\top V_\perp)}{\boldsymbol{v}^\top C_{n,n_0}B_{n_0}B_{n_0}^\top C_{n,n_0}^\top \boldsymbol{v}} \\ &\le \frac{c_1\log(1/\delta)}{\delta}\frac{\|B_{n_0}B_{n_0}^\top\|_2 Tr(C_{n,n_0}^\top V_\perp V_\perp^\top C_{n,n_0})}{\sigma_n(B_{n_0}B_{n_0}^\top)\boldsymbol{v}^\top C_{n,n_0}C_{n,n_0}^\top \boldsymbol{v}} \\ &\le \frac{c_2\log(1/\delta)}{\delta}\frac{Tr(C_{n,n_0}^\top V_\perp V_\perp^\top C_{n,n_0})}{\boldsymbol{v}^\top C_{n,n_0}C_{n,n_0}^\top \boldsymbol{v}} \\ &\le \frac{c_2\log(1/\delta)}{\delta}\frac{Tr(V_\perp^\top C_{n,n_0}C_{n,n_0}^\top V_\perp)}{\boldsymbol{v}^\top C_{n,n_0}C_{n,n_0}^\top \boldsymbol{v}},\end{aligned}$$

where c_2 is an absolute constant. □

B Proof of Theorem 3

Proof. Define the step sizes $\hat{\eta}_t = \eta_{t+n_0} = \frac{1}{n_0+t}$ for all positive integer t. Then we have $C_{n,n_0} = (I + \hat{\eta}_n A_{n_0+n})\cdots(I + \hat{\eta}_1 A_{n_0+1})$. With $n_0 > \max(4\mathcal{M}, 18\frac{\nu+\lambda_1^2}{\log(1+\frac{\delta}{100})})$, we have $\hat{\eta}_t = \frac{1}{n_0+t} \le \frac{1}{4\max(\mathcal{M},\lambda_1)}$. Thus, $\hat{\eta}_t$ satisfy the conditions in Theorem 3.1 of [7]. Define $\bar{\mathcal{V}} = \mathcal{V} + \lambda_1^2$. So, we have a bound for the error

$$\begin{aligned}1 - (\boldsymbol{w}_{n+n_0}^T v_1)^2 &\le \frac{c_1\log(1/\delta)}{\delta}\frac{Tr(C_{n,n_0}^\top V_\perp V_\perp^\top C_{n,n_0})}{v^\top C_{n,n_0}C_{n,n_0}^\top v} \\ &\le \frac{1}{Q}\exp(5\bar{\mathcal{V}}\sum_{i=1}^n \hat{\eta}_i^2)(d\exp(-2(\lambda_1-\lambda_2)\sum_{i=1}^n \hat{\eta}_i) \\ &\quad + \mathcal{V}\sum_{i=1}^n \hat{\eta}_i^2 \exp(-\sum_{j=i+1}^n 2\hat{\eta}_j(\lambda_1-\lambda_2))),\end{aligned}$$

where $Q = \frac{\delta^2}{c_1\log(1/\delta)}(1 - \frac{1}{\sqrt{\delta}}\sqrt{\exp(18\bar{\mathcal{V}}\sum_{i=1}^n \hat{\eta}_i^2) - 1})$.

Since $\hat{\eta}_t = \frac{1}{n_0+t}$, we have

$$\sum_{i=1}^{n} \hat{\eta}_t^2 \leq \frac{1}{n_0}$$
$$\sum_{i=1}^{n} \hat{\eta}_t^2 \leq \frac{\log(1+\frac{\delta}{100})}{18\bar{\mathcal{V}}}$$
$$\exp(18\bar{\mathcal{V}}\sum_{i=1}^{n} \hat{\eta}_t^2) - 1 \leq \frac{\delta}{100}$$
$$1 - \frac{1}{\sqrt{\delta}}\sqrt{\exp(18\bar{\mathcal{V}}\sum_{i=1}^{n} \hat{\eta}_i^2) - 1} > \frac{9}{10}.$$

Thus, we can conclude that $Q \geq \frac{9}{10}\frac{\delta^2}{c_1 \log(1/\delta)}$.

Besides, we have $\exp(5\bar{\mathcal{V}}\sum_{i=1}^{n} \hat{\eta}_i^2) \leq 1 + \frac{\delta}{100}$. Therefore,

$$\frac{\exp(5\bar{\mathcal{V}}\sum_{i=1}^{n} \hat{\eta}_i^2)}{Q} \leq (1+\frac{\delta}{100})\frac{10}{9}\frac{c_1\log(1/\delta)}{\delta^2} \leq \frac{c\log(1/\delta)}{\delta^2}.$$

for some constant c. Since $\sum_{t=1}^{n} \hat{\eta}_t \geq \log(1+\frac{n}{n_0})$, we have

$$\exp(-2(\lambda_1-\lambda_2)\sum_{t=1}^{n} \hat{\eta}_t) \leq (\frac{n_0}{n_0+n})^{2(\lambda_1-\lambda_2)}.$$

Thus,

$$\begin{aligned}
&\sum_{i=1}^{n} \hat{\eta}_i^2 \exp(-\sum_{j=i+1}^{n} 2\hat{\eta}_j(\lambda_1-\lambda_2)) \\
&= \sum_{i=1}^{n} \frac{1}{(n_0+i)^2}\exp(-2(\lambda_1-\lambda_2)\sum_{j=i+1}^{n}\hat{\eta}_j) \\
&\leq \sum_{i=1}^{n} \frac{1}{(n_0+i)^2}\exp(2(\lambda_1-\lambda_2)\log\frac{i+n_0+1}{n+n_0+1}) \\
&\leq \sum_{i=1}^{n} \frac{1}{(n_0+i)^2}(\frac{i+n_0+1}{n+n_0+1})^{2(\lambda_1-\lambda_2)} \\
&\leq \sum_{i=1}^{n} \frac{1}{(n_0+i)^2}\frac{(n_0+i+1)^{2(\lambda_1-\lambda_2)}}{(n_0+i)^{2(\lambda_1-\lambda_2)}}(\frac{i+n_0}{n+n_0+1})^{2(\lambda_1-\lambda_2)} \\
&\leq (\frac{n_0+2}{n_0+1})^{2(\lambda_1-\lambda_2)}\sum_{i=1}^{n} \frac{1}{(n_0+i)^2}(\frac{i+n_0}{n+n_0+1})^{2(\lambda_1-\lambda_2)} \\
&\leq (\frac{n_0+2}{n_0+1})^{2(\lambda_1-\lambda_2)}(\frac{1}{n+n_0+1})^{2(\lambda_1-\lambda_2)}\sum_{i=1}^{n}(i+n_0)^{2(\lambda_1-\lambda_2)-2}
\end{aligned}$$

Since $\lambda_1 \neq \lambda_2$, we can always scale up the matrices A_i's so that $2(\lambda_1-\lambda_2) > 1$. Now we have

$$\begin{aligned}
&\sum_{i=1}^{n} \hat{\eta}_i^2 \exp(-\sum_{j=i+1}^{n} 2\hat{\eta}_j(\lambda_1-\lambda_2)) \\
&\leq (\frac{n_0+2}{n_0+1})^{2(\lambda_1-\lambda_2)}(\frac{1}{n+n_0+1})^{2(\lambda_1-\lambda_2)}\frac{(n+n_0)^{2(\lambda_1-\lambda_2)-1}}{2(\lambda_1-\lambda_2)-1} \\
&\leq (\frac{n_0+2}{n_0+1})^{2(\lambda_1-\lambda_2)}\frac{1}{2(\lambda_1-\lambda_2)-1}\frac{1}{n+n_0}.
\end{aligned}$$

In conclusion, we have

$$\begin{aligned}1-(\boldsymbol{w}_{n+n_0}^T v_1)^2 &\leq \frac{c\log(1/\delta)}{\delta^2}(d(\frac{n_0}{n_0+n})^{2(\lambda_1-\lambda_2)}\\&+(\frac{n_0+2}{n_0+1})^{2(\lambda_1-\lambda_2)}\frac{1}{2(\lambda_1-\lambda_2)-1}\frac{1}{n+n_0})\\&=O(\frac{\mathcal{V}}{|2(\lambda_1-\lambda_2)-1|}\frac{1}{n+n_0})\end{aligned}$$

$\square$