J. Korean Math. Soc. **57** (2020), No. 2, pp. 401–414
https://doi.org/10.4134/JKMS.j190095
pISSN: 0304-9914 / eISSN: 2234-3008

AN EFFICIENT ALGORITHM FOR SLIDING WINDOW BASED INCREMENTAL PRINCIPAL COMPONENTS ANALYSIS

Geunseop Lee

Abstract. It is computationally expensive to compute principal components from scratch at every update or downdate when new data arrive and existing data are truncated from the data matrix frequently. To overcome this limitations, incremental principal component analysis is considered. Specifically, we present a sliding window based efficient incremental principal component computation from a covariance matrix which comprises of two procedures; simultaneous update and downdate of principal components, followed by the rank-one matrix update. Additionally we track the accurate decomposition error and the adaptive numerical rank. Experiments show that the proposed algorithm enables a faster execution speed and no-meaningful decomposition error differences compared to typical incremental principal component analysis algorithms, thereby maintaining a good approximation for the principal components.

1. Introduction

Principal components analysis (PCA) computes a set of linear uncorrelated variables, *called principal components*, from a set of possibly correlated data by using an orthogonal transformation. This technique aims at extracting the important information from a data, reducing the dimensionality, or analyzing the structure from the observations and the variables [1]. In particular, if we consider a data matrix $\bar{X}_n \in \mathbf{R}^{m\times n}, m \geq n$, then the principal components of $\bar{X}_n$ are computed from the eigenvectors of the covariance matrix, or singular vectors of the zero-mean matrix X_n, where X_n is computed from

$$X_n = \bar{X}_n - \mathbf{m}_n \mathbf{e}^T.$$

Here, the vector $\mathbf{e} \in \mathbf{R}^n$ is a vector of 1s, and the mean vector $\mathbf{m}_n$ of $\bar{X}_n$ is obtained from

$$\mathbf{m}_n = \frac{1}{n}\bar{X}_n \mathbf{e}.$$

Received January 31, 2019; Revised April 19, 2019; Accepted May 31, 2019.
2010 *Mathematics Subject Classification.* Primary 15A18, 15A23.
Key words and phrases. Incremental principal components analysis, sliding window.

Assume that we have the singular value decomposition (SVD) of X_n, such that

$$X_n = U_n \begin{pmatrix} \Sigma_n \\ 0 \end{pmatrix} V_n^T + E_n, \tag{1.1}$$

where $U_n \in \mathbf{R}^{m\times k}$ and $V_n \in \mathbf{R}^{n\times k}$ represent the left and right orthogonal matrices, $\Sigma_n = \mathrm{diag}(\sigma_1, \ldots, \sigma_k), \sigma_1 \geq \cdots \geq \sigma_k$, and σ_i is the i-th largest singular value of X_n, respectively. We denote E_n as the decomposition error matrix due to the truncation, and the truncation level $k \leq n$ as the rank of X_n. Then, the principal components of $\bar{X}_n$ are equivalent to the left singular vectors U_n in (1.1).

In many contexts, a new data $\mathbf{x}_{n+1}$ is continuously added to $\bar{X}_n$, and it generates a new data matrix $\bar{X}_{n+1} = \begin{pmatrix} \bar{X}_n & \mathbf{x}_{n+1} \end{pmatrix}$. Simultaneously in order to maintain the fixed size of the data matrix, $\mathbf{x}_1$ is downdated from $\bar{X}_{n+1}$ to produce $\tilde{X}_n$, where $\bar{X}_{n+1} = \begin{pmatrix} \mathbf{x}_1 & \tilde{X}_{n+1} \end{pmatrix}$. However, because the computation of a typical PCA algorithm from $\tilde{X}_{n+1}$ requires approximately $O(mn^2)$ flops, it is prohibitively expensive to compute the PCA of $\tilde{X}_{n+1}$ from scratch at every update and downdate especially if the size of $\tilde{X}_{n+1}$ is large. To overcome the difficulties, many incremental PCA algorithms, which aim at computing the principal components from their previous PCA matrices, have been studied [2, 4–6, 8, 9]. Unfortunately, those algorithms increase the decomposition error with the number of iterations. Additionally, those algorithms work when a data matrix is updated only, or when PCA is used without computing covariance matrix.

This limitation leads us to focus on developing a new algorithm to compute the approximate principal components of $\tilde{X}_{n+1}$ efficiently and close to those from a typical PCA algorithm, when a data matrix is updated in the sliding windows manner.

The zero-mean matrix X_{n+1} of $\tilde{X}_{n+1}$ comes from

$$\begin{pmatrix} X_n & \mathbf{x}_{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1 - \mathbf{m}_n & X_{n+1} \end{pmatrix} + (\mathbf{m}_n - \mathbf{m}_{n+1})\mathbf{e}^T, \tag{1.2}$$

where the mean vector $\mathbf{m}_{n+1}$ is

$$\mathbf{m}_{n+1} = \frac{1}{n}\tilde{X}_{n+1}\mathbf{e}.$$

Obviously, the procedure of finding the principal components of X_{n+1} in (1.2) comprises of two procedures as follows:

(1) Sliding window computation: This involves updating and downdating the SVD of X_{n+1} with the given SVD of X_n when $\mathbf{x}_{n+1}$ arrives and $\mathbf{x}_1$ is truncated.
(2) The rank-one update: This involves computing the SVD of the form such that
$$\bar{X} = X + \mathbf{s}\mathbf{t}^T = \bar{U}\bar{\Sigma}\bar{V}^T + E,$$
where $\mathbf{s} \in \mathbf{R}^m$ and $\mathbf{t} \in \mathbf{R}^n$ are arbitrary vectors.

The feasible constraints of the SVD on E_n in (1.1) must satisfy

$$\|E_n\|_F \leq \epsilon, \text{ and } E_n V_n = 0, \tag{1.3}$$

which further implies that $E_n = X_n(I - V_n V_n^T)$. Additionally we will adaptively adjust the truncation level k of the SVD based on the value $\|E_n\|_F$ and the predefined upper bound ϵ. In addition, we pursue tracking the value of $\|E_n\|_F$ instead of computing E_n as an easier method to track the decomposition error and save the memory space.

The rest of this paper is organized as follows. In Section 2, we present the new procedure of the sliding window algorithm of the SVD. In Section 3, we review the procedure of updating SVD with a rank-one matrix. In Section 4, we presents some numerical experiments, and finally, in Section 5, we draw conclusions.

2. Sliding window based singluar value decomposition updating algorithm

In this section, we present a new sliding window algorithm to efficiently generate the SVD of $X_{n+1} \in \mathbf{R}^{m\times n}$ from that of X_n in (1.2) while satisfying the constraints (1.3). To begin with, we need to define the function that finds an orthogonal vector from orthogonal matrices U or V. A classical Gram-Schmidt procedure is a potential candidate to produce vector $\mathbf{u} \in \mathbf{R}^m$ orthogonal to given $U \in \mathbf{R}^{m\times k}$, which satisfies

$$\mathbf{x} = \begin{pmatrix} U & \mathbf{u} \end{pmatrix} \begin{pmatrix} \mathbf{d} \\ \alpha \end{pmatrix}, \quad U^T\mathbf{u} = 0, \quad \text{and } \|\mathbf{u}\|_2 = 1, \tag{2.1}$$

where $\mathbf{d} \in \mathbf{R}^k$, and α is a constant. We define this function $[\mathbf{d}, \alpha, \mathbf{u}] = \mathbf{cgs\_orth}(U, \mathbf{x}_n)$ as described in the work of Barlow et al. [3].

Then we call three Gram-Schmidt operations, such that

$$\begin{aligned} [\mathbf{l}, \mathbf{v}_1] &= \mathbf{cgs\_orth}(V_n, \mathbf{e}_1), \\ [\mathbf{d}, \mathbf{u}_{k+1}] &= \mathbf{cgs\_orth}(U_n, X_n\mathbf{v}_1), \\ [\mathbf{h}, \mathbf{u}_1] &= \mathbf{cgs\_orth}(\begin{pmatrix} U_n & \mathbf{u}_{k+1} \end{pmatrix}, \mathbf{x}_{n+1}). \end{aligned} \tag{2.2}$$

By taking

$$\mathbf{h} = \begin{matrix} k \\ 1 \\ 1 \end{matrix} \begin{pmatrix} \mathbf{h}_1 \\ h_2 \\ h_3 \end{pmatrix}, \quad \mathbf{d} = \begin{matrix} k \\ 1 \end{matrix} \begin{pmatrix} \mathbf{d}_1 \\ d_2 \end{pmatrix},$$

we have that

$$\begin{aligned} \begin{pmatrix} \mathbf{x}_1 & X_{n+1} \end{pmatrix} &= \begin{pmatrix} X_n & \mathbf{x}_{n+1} \end{pmatrix} \\ &= \begin{pmatrix} U_n & \mathbf{u}_{k+1} & \mathbf{u}_1 \end{pmatrix} \begin{pmatrix} \Sigma_n & \mathbf{d}_1 & \mathbf{h}_1 \\ 0 & d_2 & h_2 \\ 0 & 0 & h_3 \end{pmatrix} \begin{pmatrix} V_n^T & 0 \\ \mathbf{v}_1^T & 0 \\ 0 & 1 \end{pmatrix} + \tilde{E}_{n+1}, \end{aligned} \tag{2.3}$$

where the new decomposition $\tilde{E}_{n+1}$ is given by

$$\tilde{E}_{n+1} = \begin{pmatrix} \mathbf{x}_1 & X_{n+1} \end{pmatrix} (I - V_{n+1}V_{n+1}^T) = \begin{pmatrix} E_n - X_n\mathbf{v}_1\mathbf{v}_1^T & 0 \end{pmatrix}. \tag{2.4}$$

Proposition 2.1. *Equation* (2.4) *satisfies the second constraint in* (1.3).

Proof. We have that

$$\begin{aligned} \tilde{E}_{n+1}V_{n+1} &= \begin{pmatrix} E_n - X_n\mathbf{v}_1\mathbf{v}_1^T & 0 \end{pmatrix} V_{n+1} \\ &= \begin{pmatrix} (E_n - X_n\mathbf{v}_1\mathbf{v}_1^T)V_n & (E_n - X_n\mathbf{v}_1\mathbf{v}_1^T)\mathbf{v}_1 & 0 \end{pmatrix}. \end{aligned}$$

Since $E_nV_n = 0$, $V_n^T\mathbf{v}_1 = 0$, and $E_n\mathbf{v}_1 = (X_n - U_n\Sigma_nV_n^T)\mathbf{v}_1 = X_n\mathbf{v}_1$, all elements of $\tilde{E}_{n+1}V_{n+1}$ are zero. □

Next, find the orthogonal matrix Q_1, the product of given rotations, which produces the new orthogonal matrix V_{n+1} such that,

$$\begin{pmatrix} V_n & \mathbf{v}_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} Q_1 = \begin{pmatrix} 0 & 1 \\ V_{n+1} & 0 \end{pmatrix},$$

where, Q_1 makes the first row of $\begin{pmatrix} V_n & \mathbf{v}_1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ to $\begin{pmatrix} 0 & 0 & \cdots & 1 \end{pmatrix}$. Since the last element is 0, Q_1 may have a form such that

$$Q_1 = \begin{pmatrix} 0 & \tilde{Q}_1 \\ 1 & 0 \end{pmatrix},$$

where $\tilde{Q}_1^T\tilde{Q}_1 = I$, and the product of the middle matrix in (2.3) has a form

$$\begin{pmatrix} \Sigma_n & \mathbf{d}_1 & \mathbf{h}_1 \\ 0 & d_2 & h_2 \\ 0 & 0 & h_3 \end{pmatrix} Q_1 = \begin{pmatrix} \mathbf{h}_1 & R_1 \\ h_2 & \mathbf{r}_2 \\ h_3 & 0 \end{pmatrix}, \tag{2.5}$$

where R_1, $\mathbf{r}_2$ are parts of intermediate result from the matrix-matrix multiplication. Then, we need an orthogonal matrix P_1, which makes (2.5) triangular form, such that

$$P_1^T \begin{pmatrix} \mathbf{h}_1 & R_1 \\ h_2 & \mathbf{r}_2 \\ h_3 & 0 \end{pmatrix} = \begin{pmatrix} \tilde{R} & \tilde{\mathbf{r}} \\ 0 & \tilde{r} \end{pmatrix}.$$

Thus, by taking a new U_{n+1} from

$$\begin{pmatrix} U_{n+1} & \tilde{\mathbf{u}}_1 \end{pmatrix} = \begin{pmatrix} U_n & \mathbf{u}_{k+1} & \mathbf{u}_1 \end{pmatrix} P_1,$$

we have a new decomposition

$$\begin{pmatrix} \mathbf{x}_1 & X_{n+1} \end{pmatrix} = \begin{pmatrix} U_{n+1} & \tilde{\mathbf{u}}_1 \end{pmatrix} \begin{pmatrix} \tilde{R} & \tilde{\mathbf{r}} \\ 0 & \tilde{h_3} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ V_{n+1} & 0 \end{pmatrix}^T + \tilde{E}_{n+1},$$

and its donwdated form yields

$$X_{n+1} = U_{n+1}\tilde{R}V_{n+1}^T + \tilde{E}_{n+1}(:, 2:n). \tag{2.6}$$

Since $\tilde{R}$ is a triangular matrix, by computing the SVD of the middle matrix in (2.6) like $\tilde{R} = P_2^T \Sigma_{n+1} Q_2$, we have a SVD form of $X_{n+1} = U_{n+1}\Sigma_{n+1}V_{n+1}^T + \tilde{E}_{n+1}(:, 2:n)$ such that

$$\begin{aligned} U_{n+1} &= U_{n+1}P_2, \\ \Sigma_{n+1} &= P_2 \tilde{R} Q_2^T, \\ V_{n+1} &= V_{n+1}Q_2. \end{aligned} \tag{2.7}$$

Proposition 2.2. *Let* $\tilde{E}_{n+1} = \begin{pmatrix} 0 & E_{n+1} \end{pmatrix}$. *Then* E_{n+1} *satisfies the constraint* (1.3).

Proof.

$$\begin{aligned} \tilde{E}_{n+1}\mathbf{e}_1 &= \begin{pmatrix} E_n - X_n\mathbf{v}_1\mathbf{v}_1^T & 0 \end{pmatrix}\mathbf{e}_1 \\ &= \begin{pmatrix} (X_n - U_nR_nV_n^T)\mathbf{e}_1 - X_n\mathbf{v}_1\mathbf{v}_1^T\mathbf{e}_1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{x}_1 - U_nR_nV_n^T\mathbf{e}_1 - X_n\mathbf{v}_1\mathbf{v}_1^T\mathbf{e}_1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{x}_1 - (X_n - E_n)\mathbf{e}_1 - X_n\mathbf{v}_1\mathbf{v}_1^T\mathbf{e}_1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} E_n\mathbf{e}_1 - X_n\mathbf{v}_1\mathbf{v}_1^T\mathbf{e}_1 & 0 \end{pmatrix}. \end{aligned}$$

Since in (2.2), we have that

$$\begin{pmatrix} V_n & \mathbf{v}_1 \end{pmatrix}\mathbf{l} = \mathbf{e}_1,$$

thus

$$\begin{aligned} \tilde{E}_{n+1}\mathbf{e}_1 &= \begin{pmatrix} (E_n - X_n\mathbf{v}_1\mathbf{v}_1^T)\begin{pmatrix} V_n & \mathbf{v}_1 \end{pmatrix}\mathbf{l} & 0 \end{pmatrix} \\ &= \begin{pmatrix} E_n\mathbf{v}_1 - X_n\mathbf{v}_1 & 0 \end{pmatrix} \\ &= 0. \end{aligned}$$

Since, $\tilde{E}_{n+1}\mathbf{e}_1 = \tilde{E}_{n+1}(:,1) = 0$, and

$$\tilde{E}_{n+1}\begin{pmatrix} 0 & 1 \\ V_{n+1} & 0 \end{pmatrix} = \begin{pmatrix} 0 & E_{n+1} \end{pmatrix}\begin{pmatrix} 0 & 1 \\ V_{n+1} & 0 \end{pmatrix} = 0,$$

we have that $E_{n+1}V_{n+1} = 0$. □

Here, we pursue not saving whole elements of E_{n+1}, but rather tracking the value $\|E_{n+1}\|_F$ which is achieved by computing

$$\|E_{n+1}\|_F = \sqrt{\|\tilde{E}_{n+1}\|_F^2 - \|X_n\mathbf{v}_1\|_2^2}.$$

The procedures described above for the sliding window algorithm are summarized in Algorithm 1. Note that the values $\|E\|_F$ and $\|\bar{E}\|_F$ are considered as variables in Algorithm 1.

2.1. Computational complexity

For the data matrix $X \in \mathbf{R}^{m\times n}$, a computational complexity of a PCA algorithm requires $O(mn^2 + n^3)$ flops, thus the cost increases dramatically as the size n increases. Typically, the most expensive operations in Algorithm 1 are simple matrix-matrix multiplications for updating U_n in steps 5, which costs $O(mk^2)$ flops approximately, and updating the decomposition error in step 1 which costs $O(mn)$. Therefore, the overall computational complexity of Algorithm 1 takes around $O(mn)$ flops only, and is much cheaper than that of a typical PCA algorithm.

Algorithm 1 $[\bar{U}, \bar{\Sigma}, \bar{V}, \|\bar{E}\|_F] = \mathbf{SlidingWindowSVD}(U, \Sigma, V, X, \mathbf{x}, \|E\|_F, \epsilon)$

1. Make the calls
   $[\mathbf{l}, \mathbf{v}_1] = \mathbf{cgs\_orth}(V, \mathbf{e}_1)$,
   $[\mathbf{d}, \mathbf{u}_{k+1}] = \mathbf{cgs\_orth}(U, X\mathbf{v}_1)$,
   $[\mathbf{h}, \mathbf{u}_1] = \mathbf{cgs\_orth}(\begin{pmatrix} U_n & \mathbf{u}_{k+1} \end{pmatrix}, \mathbf{x})$.
2. Find the orthogonal matrix Q_1 which satisfies
   $$\begin{pmatrix} V & \mathbf{v}_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} Q_1 = \begin{pmatrix} 0 & 1 \\ \hat{V} & 0 \end{pmatrix},$$
   and find P_1 which satisfies, $P_1^T \begin{pmatrix} R & \mathbf{d}_1 & \mathbf{h}_1 \\ 0 & d_2 & h_2 \\ 0 & 0 & h_3 \end{pmatrix} Q_1 = \begin{pmatrix} \tilde{R} & \tilde{\mathbf{r}} \\ 0 & \tilde{h}_3 \end{pmatrix}$,
   where $\mathbf{d} = \begin{pmatrix} \mathbf{d}_1 & d_2 \end{pmatrix}$, and $\mathbf{h} = \begin{pmatrix} \mathbf{h}_1 & h_2 & h_3 \end{pmatrix}$.
3. Update the decomposition error as
   $\|\bar{E}\|_F = \sqrt{\|E\|_F^2 - \|X\mathbf{v}_1\|_2^2}$.
4. Compute the SVD of the middle matrix such that
   $P_2^T \tilde{R} Q_2 = \hat{\Sigma}$.
5. Set
   $\begin{pmatrix} \hat{U} & \bar{\mathbf{u}} \end{pmatrix} = \begin{pmatrix} U & \mathbf{u}_{k+1} & \mathbf{u}_1 \end{pmatrix} P_1$, $\bar{U} = \hat{U} P_2$, $\bar{U} = \bar{U}(:, 1:k)$
   $\bar{V} = \hat{V} Q_2$, $\bar{V} = \bar{V}(:, 1:k)$,
   return $\bar{U}$, $\bar{R}$, $\bar{V}$ and $\|\bar{E}\|_F$.

3. Rank-one update algorithm

Since the typical rank-one update to the matrix will destroy the structure of the SVD, we need a careful procedure to preserve the SVD form while updating. Rewrite the equation (1.2) with a simplified form, such that

$$X_{n+1} = X_n + \mathbf{s}\mathbf{t}^T = U_n \Sigma_n V_n^T + E_n + \mathbf{s}\mathbf{t}^T. \tag{3.1}$$

Here for (1.2), the vector $\mathbf{s}$ corresponds to $(\mathbf{m}_n - \mathbf{m}_{n+1})$, and the vector $\mathbf{t}$ corresponds to $\mathbf{e}$ in our applications.

First, we use two Gram-Schmidt orthogonalization to decompose $\mathbf{s}$ and $\mathbf{t}$ such that

$$\begin{aligned}[\hat{\mathbf{s}}, \mathbf{u}_{k+1}] &= \mathbf{cgs\_orth}(U_n, \mathbf{s}),\\ [\hat{\mathbf{t}}, \mathbf{v}_{k+1}] &= \mathbf{cgs\_orth}(V_n, \mathbf{t}),\end{aligned} \tag{3.2}$$

and we use the other Gram-Schmidt operation to decompose $X_n\mathbf{v}_{k+1}$ such that

$$[\mathbf{g}, \mathbf{u}_{k+2}] = \mathbf{cgs\_orth}(\begin{pmatrix} U_n & \mathbf{u}_{k+1} \end{pmatrix}, X_n\mathbf{v}_{k+1}). \tag{3.3}$$

Let $\mathbf{g}^T = \begin{pmatrix} \mathbf{g}_1^T & g_2 & g_3 \end{pmatrix}^T$, where $\mathbf{g}_1 \in \mathbf{R}^k$, then we have a SVD form of X_{n+1} in (3.1), such that

(3.4)

$$X_{n+1} = \begin{pmatrix} U_n & \mathbf{u}_{k+1} & \mathbf{u}_{k+2} \end{pmatrix} \left[\begin{pmatrix} \Sigma_n & \mathbf{g}_1 \\ 0 & g_2 \\ 0 & g_3 \end{pmatrix} + \begin{pmatrix} \hat{\mathbf{s}} \\ 0 \end{pmatrix} \hat{\mathbf{t}}^T \right] \begin{pmatrix} \mathbf{V}_n^T \\ \mathbf{v}_{k+1}^T \end{pmatrix} + E_{n+1},$$

where E_{n+1} is a new decomposition error that satisfies the constraint (1.3).

Proposition 3.1. *The decomposition error E_{n+1} in* (3.4) *satisfies the constraint $E_{n+1}V_{n+1} = 0$.*

Proof. The equation (3.4) is derived from

$$X_{n+1} = U_n\Sigma_n V_n^T + E_{n+1} + X_n\mathbf{v}_{k+1}\mathbf{v}_{k+1}^T + \mathbf{s}\mathbf{t}^T.$$

Thus

$$E_{n+1} = E_n - X_n\mathbf{v}_{k+1}\mathbf{v}_{k+1}^T.$$

Then

$$\begin{aligned} E_{n+1}V_{n+1} &= (E_n - X_n\mathbf{v}_{k+1}\mathbf{v}_{k+1}^T) \begin{pmatrix} V_n & \mathbf{v}_{k+1} \end{pmatrix} \\ &= \begin{pmatrix} E_nV_n - X_n\mathbf{v}_{k+1}\mathbf{v}_{k+1}^TV_n & E_n\mathbf{v}_{k+1} - X_n\mathbf{v}_{k+1} \end{pmatrix}. \end{aligned}$$

Since $E_nV_n = 0$ and $V_n \perp \mathbf{v}_{k+1}$, the first element of $E_{n+1}V_{n+1}$ is 0, and the second element is 0 as well, because

$$E_n\mathbf{v}_{k+1} - X_n\mathbf{v}_{k+1} = E_n\mathbf{v}_{k+1} - (U_n\Sigma_nV_n^T + E_n)\mathbf{v}_{k+1} = 0.$$

Therefore, the equation (3.4) satisfies the constraint (1.3). □

The next step is to find the orthogonal matrices P_3 and Q_3 that make the middle matrix in (3.4) diagonal as follows:

$$P_3^T \left[\begin{pmatrix} \Sigma_n & \mathbf{g}_1 \\ 0 & g_2 \\ 0 & g_3 \end{pmatrix} + \begin{pmatrix} \hat{\mathbf{s}} \\ 0 \end{pmatrix} \hat{\mathbf{t}}^T \right] Q_3 = \begin{pmatrix} \Sigma_{n+1} \\ 0 \end{pmatrix}, \tag{3.5}$$

where $\Sigma_{n+1} \in \mathbf{R}^{(k+1)\times(k+1)}$ is a diagonal matrix. Similar to the idea in Section 2, we will not obtain E_{n+1}, and instead compute the value $\|E_{n+1}\|_F$ which is easily calculated by

$$\|E_{n+1}\|_F = \sqrt{\|E_n\|_F^2 - \|X_n\mathbf{v}_{k+1}\|_2^2}.$$

The final SVD of $X_{n+1} = U_{n+1}\Sigma_{n+1}V_{n+1} + E_{n+1}$ after updating is given as follows:

$$U_{n+1} = \begin{pmatrix} U_n & \mathbf{u}_{k+1} & \mathbf{u}_{k+2} \end{pmatrix} P_3,$$
$$V_{n+1} = \begin{pmatrix} V_n & \mathbf{v}_{k+1} \end{pmatrix} Q_3.$$

Since we know that the constraint $\|E_{n+1}\|_F \leq \epsilon$ must hold, we will further check a possible truncation of the small singular triplets from X_{n+1} in (3.5). Assume that $\Sigma_{n+1} = \mathbf{diag}(\sigma_1, \ldots, \sigma_j)$, where j is a numerical rank of X_{n+1}, and possibly among $k-1, k, k+1$ or even $k+2$. By checking the condition $\sqrt{\sigma_j^2 + \|E_{n+1}\|_F^2} \leq \epsilon$ iteratively, we can truncate Σ_{n+1} and its corresponding singular vectors $U_{n+1}(:, j)$ and $V_{n+1}(:, j)$ until satisfying (1.3). Therefore, the final SVD form of X_{n+1} is given by

$$U_{n+1} = U_{n+1}(:, 1:j),$$
$$V_{n+1} = V_{n+1}(:, 1:j),$$
$$\Sigma_{n+1} = \Sigma_{n+1}(1:j, 1:j),$$

and

$$\|E_{n+1}\|_F^2 = \sqrt{\|E_{n+1}\|_F^2 + \sum_{i=k-1}^{j} \sigma_i^2}.$$

The procedure for rank-one update is summarized in Algorithm 2.

Similar to the case of sliding window based SVD update in Section 2.1, the most time-consuming operation in Algorithm 2 is the computation of $X_n\mathbf{v}_{k+1}$ in step 1, which takes around $O(mn)$ flops, and is still much cheaper than that of a typical PCA algorithm.

4. Numerical experiments

This section presents experimental results that measure the performance of the proposed algorithm. The experiments were run on an Intel Core i9 computer with 3.10GHz, and 32 GB memory. The test code were performed using MATLAB version 9.3.0.713579 (R2017b).

We generate random matrices as a reference in order to verify the performance of our algorithm. Specifically, because the data matrix is ill-conditioned under many contexts, we created a random data matrix $X \in \mathbf{R}^{m\times n}$ to be ill-conditioned where its singular values range as

$$\sigma_i = e^{\frac{-i}{n}\log(d)},\ 1 \leq i \leq n,$$

where the parameter d denotes the degree of the ill-posedness in X. Using the data matrix X, we compared the execution time and decomposition error with the following example cases.

Example 1. $X \in \mathbf{R}^{5000\times 500}$ is generated based on the procedure above. We set d as $1.0e+5$ to adjust the ill-posedness in X, and $\epsilon = 0.1\|X\|_F$. We chose

Algorithm 2 $[\hat{U}, \hat{\Sigma}, \hat{V}, \|\hat{E}\|_F] = \mathbf{rankupdateSVD}(U, \Sigma, V, X, \mathbf{s}, \mathbf{t}, \|E\|_F, \epsilon)$

1. Make the call
   $[\hat{\mathbf{t}}, \mathbf{v}_{k+1}] = \mathbf{cgs\_orth}(V, \mathbf{t})$,
   $[\hat{\mathbf{s}}, \mathbf{u}_{k+1}] = \mathbf{cgs\_orth}(U, \mathbf{s})$,
   $[\mathbf{g}, \mathbf{u}_{k+2}] = \mathbf{cgs\_orth}(\begin{pmatrix} U & \mathbf{u}_{k+1} \end{pmatrix}, X\mathbf{v}_{k+1})$.
2. Find the orthogonal matrices P_3 and Q_3 such that
   $$P_3^T \left[\begin{pmatrix} \Sigma & \mathbf{g}_1 \\ 0 & g_2 \\ 0 & g_3 \end{pmatrix} + \begin{pmatrix} \hat{\mathbf{s}} \\ 0 \end{pmatrix} \hat{\mathbf{t}}^T \right] Q_3 = \begin{pmatrix} \hat{\Sigma} \\ 0 \end{pmatrix},$$
   where $\hat{\Sigma}$ is a diagonal matrix.
3. Update $\|\hat{E}\|_F = \sqrt{\|E\|_F^2 - \|X\mathbf{v}_{k+1}\|_2^2}$.
4. Update $\hat{U} = \begin{pmatrix} U & \mathbf{u}_{k+1} & \mathbf{u}_{k+2} \end{pmatrix} P_3$, and $\hat{V} = \begin{pmatrix} V & \mathbf{v}_{k+1} \end{pmatrix} Q_3$.
5. **For** $j = k+2 : -1 : k-1$
   If $\sqrt{\sigma_{k+1}^2 + \|\hat{E}\|_F^2} > \epsilon$
   then break,
   else Update $\|\hat{E}\|_F = \sqrt{\sigma_j^2 + \|\hat{E}\|_F^2}$
   end
6. Update $\hat{U} = \hat{U}(:, 1:j)$,
   $\hat{V} = \hat{V}(:, 1:j)$,
   and $\hat{\Sigma} = \hat{\Sigma}(1:j, 1:j)$,
   return $\hat{U}$, $\hat{\Sigma}$, $\hat{V}$ and $\|\hat{E}\|_F$.

Algorithm 3 $[\hat{U}, \hat{\Sigma}, \hat{V}, \|\hat{E}\|_F] = \mathbf{swIPCA}(U, \Sigma, V, X, \mathbf{x}, \mathbf{s}, \mathbf{t}, \|E\|_F, \epsilon)$

1. Make the call
   $[\bar{U}, \bar{\Sigma}, \bar{V}, \|\bar{E}\|_F] = \mathbf{SlidingWindowSVD}(U, \Sigma, V, X, \mathbf{x}, \|E\|_F, \epsilon)$.
2. Make the call
   $[\hat{U}, \hat{\Sigma}, \hat{V}, \|\hat{E}\|_F] = \mathbf{rankupdateSVD}(\bar{U}, \bar{\Sigma}, \bar{V}, X, \mathbf{s}, \mathbf{t}, \|\bar{E}\|_F, \epsilon)$,
   return $\hat{U}, \hat{\Sigma}, \hat{V}, \|\hat{E}\|_F$.

100 vectors from $X(:, 401:500)$ to make a new arrived data $\mathbf{x}$. To obtain the initial SVD, we executed MATLAB's "svd" command to $X(:, 1:400)$.

Example 2. $X \in \mathbf{R}^{5000\times1000}$ is generated based on the procedure above. We set d as $1.0e+7$ to adjust the ill-posedness in X, and $\epsilon = 0.1\|X\|_F$. We chose 100 vectors from $X(:, 901:1000)$ to make a new arrived data $\mathbf{x}$. To obtain the initial SVD, we executed MATLAB's "svd" command to $X(:, 1:900)$.

Example 3. $X \in \mathbf{R}^{10000\times500}$ is generated based on the procedure above. We set d as $1.0e+7$ to adjust the ill-posedness in X, and $\epsilon = 0.1\|X\|_F$. We chose

100 vectors from $X(:, 401:500)$ to make a new arrived data $\mathbf{x}$. To obtain the initial SVD, we executed MATLAB's "svd" command to $X(:, 1:400)$.

After generating the data matrices, we split these as

$$X = Y_1 \Sigma_1 Z_1^T + Y_2 \Sigma_2 Z_2^T,$$

where $\Sigma_1 = \text{diag}(\sigma_1, \ldots, \sigma_k)$, and $\Sigma_2 = \text{diag}(\sigma_{k+1}, \ldots, \sigma_n)$, and we truncated the unnecessary part $Y_2 \Sigma_2 Z_2^T$ from X. Note that we chose the initial truncation level k satisfying

$$\mathbf{arg\_min}_k(0.9\epsilon - \|X - Y_1 \Sigma_1 Z_1^T\|_F).$$

We set the initial decomposition error as $\|E_1\|_F = \|\Sigma_2\|_F$. To evaluate the performance of the proposed algorithm, we execute Matlab's 'svd' function, the well-known IPCA algorithm proposed by Weng et al. [8] (CCIPCA henceforth), and Algorithm 3 (swIPCA, henceforth). However, since CCIPCA only considers covariance-free matrices, we first experiment the data matrix in Examples 1-3 without considering the zero-mean cases. Specifically, we compare the the execution time and the subspace differences between the result from a typical PCA algorithm and the other algorithms [7], which is defined as

$$\text{relative error} = \sum_{i=1}^{k} \sum_{j=1}^{k} (\bar{\mathbf{u}}_i^T \bar{\mathbf{u}}_j - \mathbf{u}_i^T \mathbf{u}_j)^2,$$

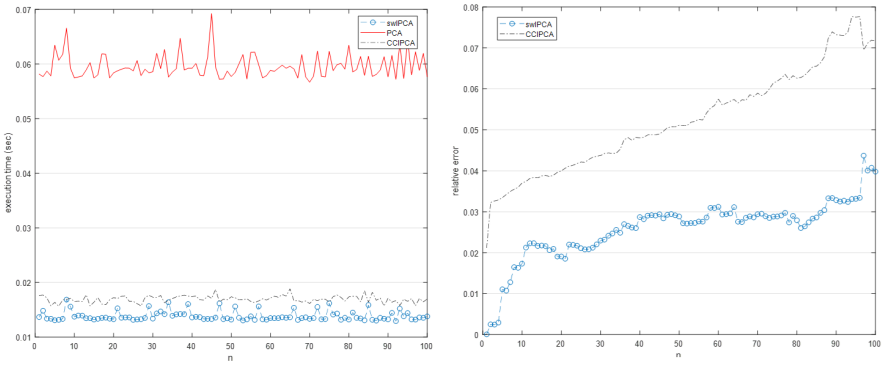
where $\bar{\mathbf{u}}$ represent the eigenvectors generated from a typical PCA algorithm, and $\mathbf{u}$ is calculated from swIPCA or CCIPCA.

Figure 1 depicts the relative errors and execution time from typical PCA algorithm, CCIPCA, and swIPCA. Here, when we compute the update of the covariance-free matrices in Examples 1-3 with swIPCA, we only use Algorithm 1. Figure 1 shows that swIPCA produces the fastest output among three algorithms. Additionally, the relative errors from swIPCA are better than those from CCIPCA, which means the eigenvectors computed from swIPCA are closer to those from a typical PCA.

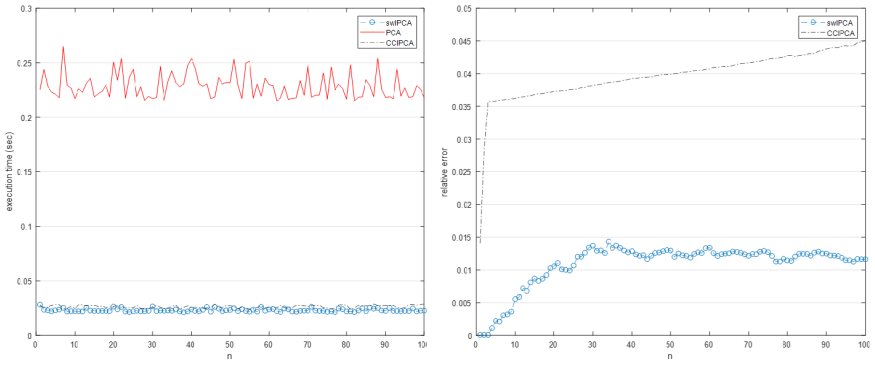
Figure 2, shows the comparison of relative errors and execution speed when zero-mean matrices from Examples 1-3 are considered in each sliding window step. In this experiment, we use Matlab's 'princomp' function and swIPCA. The relative errors to compare the accuracy of the results, are defined as

$$\text{relative error} = \|X - U_n \Sigma_n V_n^T\|_F.$$

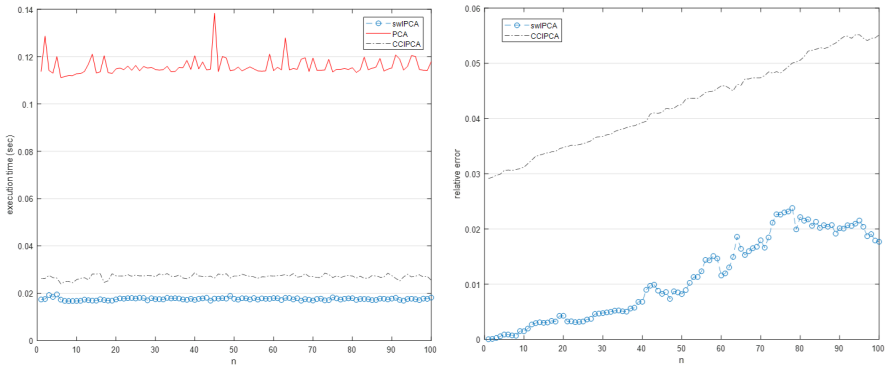
Throughout the iterations, we can see that swIPCA achieved much faster execution speed than Matlab's 'princomp' function. The differences of relative error are not significant. Figure 3 depicts the variation of the numerical rank, and Figure 4 depicts the difference $|\|X - U_n \Sigma_n V_n^T\|_F - \|E_n\|_F|$ with $\log_{10}$ scale to measure how tracking the value of $\|E_n\|_F$ is accurate as iterations proceed in Example 1.



(a) Example 1

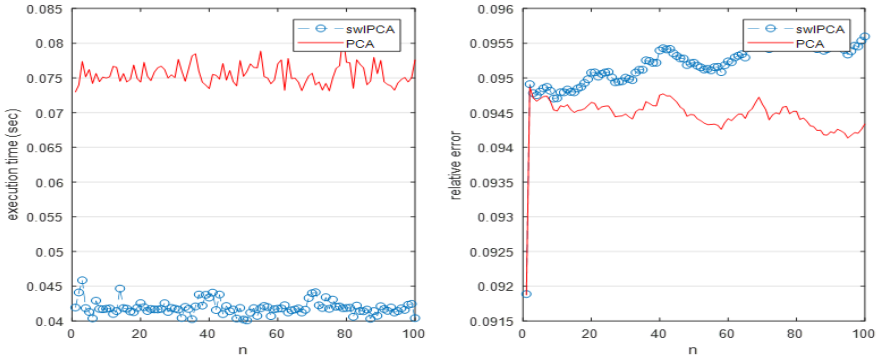


(b) Example 2

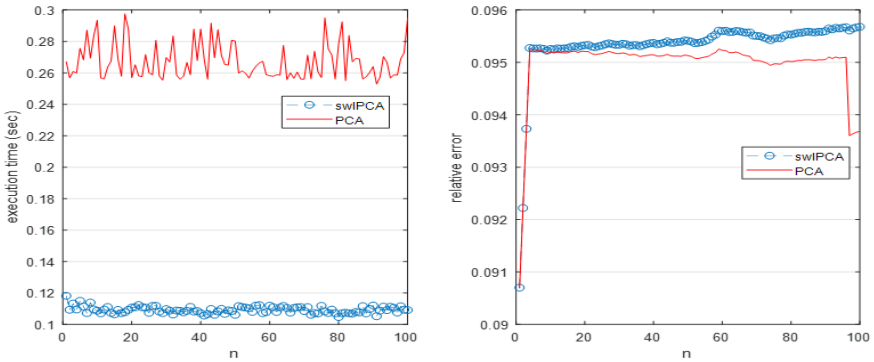


(c) Example 3

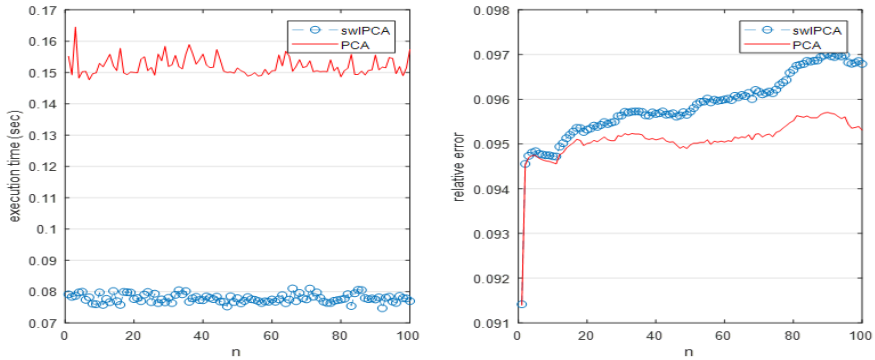
FIGURE 1. Comparison of relative errors and execution time among a typical PCA algorithm, CCIPCA, and swIPCA when covariance-free data matrices are used.



(a) Example 1



(b) Example 2



(c) Example 3

FIGURE 2. Comparison of relative errors and execution time between a typical PCA algorithm and swIPCA when zero-mean matrices are used.

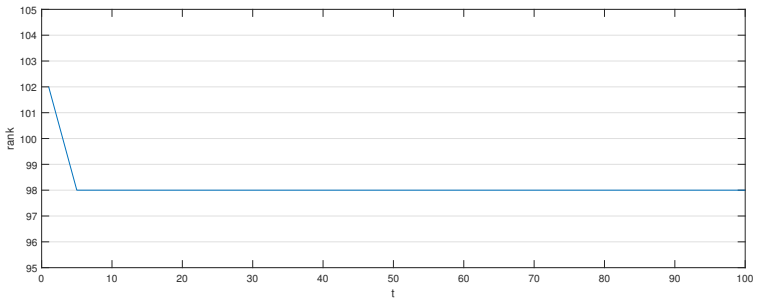


FIGURE 3. Variation of numerical rank when swIPCA is applied to the data matrix in Example 1.

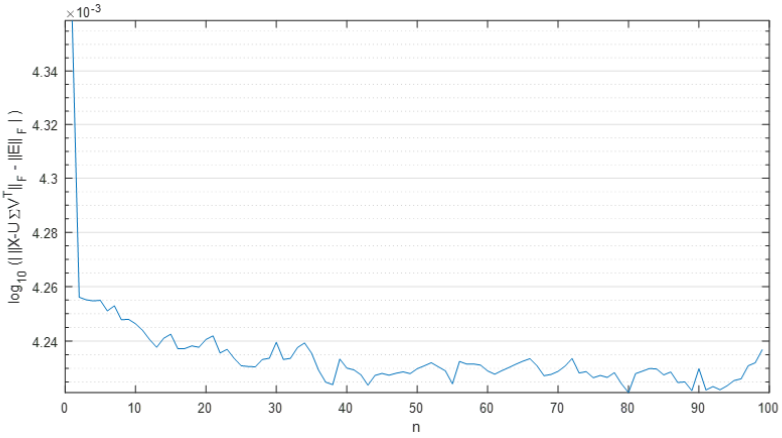


FIGURE 4. Accuracy of $\|E\|_F$ in each sliding window step when swIPCA is applied to the data matrix in Example 1.

5. Conclusion

The computation of principal components from scratch at every iteration is prohibitively expensive when the data matrix is frequently updated and downdated. To overcome the difficulties, we proposed the sliding window based incremental principal component algorithm as a tool to determine approximate principal component. Specifically, the proposed method comprises two procedure: the efficient updating and downdating algorithm of the singular value decomposition, followed by the rank-one matrix update algorithm. The

experimental results show that the proposed algorithm achieves faster execution speed with no-significant decomposition error compared to the results for typical principal component algorithms.

Acknowledgments. This work was supported by Hankuk University of Foreign Studies Research Fund and National Research Foundation of Korea (NRF) grant funded by the Korean government (2018R1C1B5085022).

References

[1] H. Abdi and L. J. Williams, *Principal component analysis*, Wiley Interdiscip. Rev. Comput. Stat. 2–4 (2010), pp. 433–459.

[2] R. Badeau, G. Richard, and B. David, *Sliding window adaptive SVD algorithms*, IEEE Trans. Signal Process. **52** (2004), no. 1, 1–10. https://doi.org/10.1109/TSP.2003.820069

[3] J. L. Barlow, A. Smoktunowicz, and H. Erbay, *Improved Gram-Schmidt type downdating methods*, BIT **45** (2005), no. 2, 259–285. https://doi.org/10.1007/s10543-005-0015-2

[4] M. Brand, *Fast low-rank modifications of the thin singular value decomposition*, Linear Algebra Appl. **415** (2006), no. 1, 20–30. https://doi.org/10.1016/j.laa.2005.07.021

[5] J. R. Bunch and C. P. Nielsen, *Updating the singular value decomposition*, Numer. Math. **31** (1978/79), no. 2, 111–129. https://doi.org/10.1007/BF01397471

[6] Y. Li, L. Xu, J. Morphett, and R. Jacobs, *An integreted algorithm of incremental and robust PCA*, Proc. Int. Conf. on Image Processing (2003), 245–248.

[7] C. Pehlevan, T. Hu, and D. B. Chklovskii, *A Hebbian/anti-Hebbian neural network for linear subspace learning: a derivation from multidimensional scaling of streaming data*, Neural Comput. **27** (2015), no. 7, 1461–1495. https://doi.org/10.1162/neco_a_00745

[8] J. Weng, Y. Zhang, and W. S. Hwang, *Candid covariance-free incremental principal component analysis*, IEEE Trans. Pattern Anal. Mach. Intell, 25–8 (2003), pp. 1034–1040.

[9] H. Zhao, P. C. Yuen, and J. T. Kwok, *A novel incremental principal component analysis and its application for face recognition*, IEEE Trans. on Sys. Man, and Cybernetics, 36–4 (2006), pp. 873–886.

Geunseop Lee
Division of Global Business and Technology
Hankuk University of Foreign Studies
Yongin 17035, Korea
Email address: geunseop.lee@hufs.ac.kr