# Zoom-SVD: Fast and Memory Efficient Method for Extracting Key Patterns in an Arbitrary Time Range

Jun-Gi Jang
Seoul National University
elnino4@snu.ac.kr

Dongjin Choi
Seoul National University
skywalker5@snu.ac.kr

Jinhong Jung
Seoul National University
jinhongjung@snu.ac.kr

U Kang
Seoul National University
ukang@snu.ac.kr

## ABSTRACT

Given multiple time series data, how can we efficiently find latent patterns in an arbitrary time range? Singular value decomposition (SVD) is a crucial tool to discover hidden factors in multiple time series data, and has been used in many data mining applications including dimensionality reduction, principal component analysis, recommender systems, etc. Along with its static version, incremental SVD has been used to deal with multiple semi-infinite time series data and to identify patterns of the data. However, existing SVD methods for the multiple time series data analysis do not provide functionality for detecting patterns of data in an arbitrary time range: standard SVD requires data for all intervals corresponding to a time range query, and incremental SVD does not consider an arbitrary time range.

In this paper, we propose Zoom-SVD, a fast and memory efficient method for finding latent factors of time series data in an arbitrary time range. Zoom-SVD incrementally compresses multiple time series data block by block to reduce the space cost in storage phase, and efficiently computes singular value decomposition (SVD) for a given time range query in query phase by carefully stitching stored SVD results. Through extensive experiments, we demonstrate that Zoom-SVD is up to 15× faster, and requires 15× less space than existing methods. Our case study shows that Zoom-SVD is useful for capturing past time ranges whose patterns are similar to a query time range.

## KEYWORDS

Time range query; singular value decomposition; multiple time series data

**ACM Reference Format:**
Jun-Gi Jang, Dongjin Choi, Jinhong Jung, and U Kang. 2018. Zoom-SVD: Fast and Memory Efficient Method for Extracting Key Patterns in an Arbitrary Time Range . In *2018 ACM Conference on Information and Knowledge Management (CIKM'18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271682

## 1 INTRODUCTION

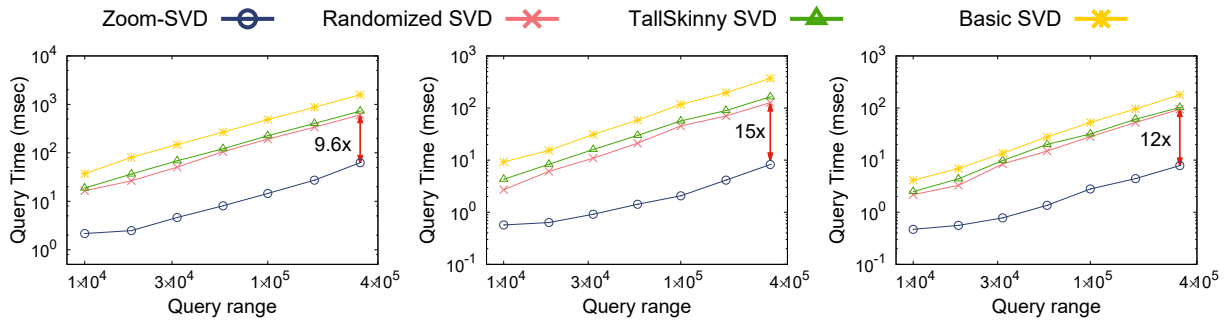
Given multiple time series data (e.g., measurements from multiple sensors) and a time range (e.g., 1:00 am - 3:00 am yesterday), how can we efficiently discover latent factors of the time series in the range? Revealing hidden factors in time series is important for analysis of patterns and tendencies encoded in the time series data. Singular value decomposition (SVD) effectively finds hidden factors in data, and has been extensively utilized in many data mining applications such as dimensionality reduction [29], principal component analysis (PCA) [13, 41], data clustering [23, 36], tensor analysis [10–12, 22, 28, 33], graph mining [14–16, 38] and recommender systems [18, 27]. SVD has been also successfully applied to stream mining tasks [37, 41] in order to analyze time series data. However, methods based on standard SVD [2, 7, 32, 42] are not suitable for finding latent factors in an arbitrary time range since the methods have an expensive computational cost, and they have to store all the raw data. This limitation makes it difficult to investigate patterns of a time range in stream environment even if it is important to analyze a specific past event or find recurring patterns in time series [25]. A naive approach for a time range query on time series is to store all of the arrived data and apply SVD to the data, but this approach is inefficient since it requires huge storage space, and the computational cost of SVD for a long time range query is expensive.

In this paper, we propose Zoom-SVD (Zoomable SVD), an efficient method for revealing hidden factors of multiple time series in an arbitrary time range. With Zoom-SVD, users can zoom-in to find patterns in a specific time range of interest, or zoom-out to extract patterns in a wider time range. Zoom-SVD comprises two phases: storage phase and query phase. Zoom-SVD considers multiple time series as a set of blocks of a fixed length. In the storage phase, Zoom-SVD carefully compresses each block using SVD and low-rank approximation to reduce storage cost and incrementally updates the most recent block of a newly arrived data. In the query phase, Zoom-SVD efficiently computes the SVD results in a given time range based on the compressed blocks. Through extensive experiments with real-world multiple time series data, we demonstrate the effectiveness and the efficiency of Zoom-SVD compared to other methods as shown in Figure 1. The main contributions of this paper are summarized as follows:

- **Algorithm.** We propose Zoom-SVD, an efficient method for extracting key patterns from multiple time series data in an arbitrary time range.
- **Analysis.** We theoretically analyze the time and the space complexities of our proposed method Zoom-SVD.
- **Experiment.** We present experimental results showing that Zoom-SVD computes time range queries up to 15× faster, and requires up to 15× less space than other methods. We also confirm that our proposed method Zoom-SVD provides the best trade-off between efficiency and accuracy.

The codes and datasets for this paper are available at http://datalab.snu.ac.kr/zoomsvd. In the rest of this paper, we describe the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*CIKM '18, October 22–26, 2018, Torino, Italy*
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271682




(b) Query time of the Activity dataset (c) Query time of the Gas dataset (d) Query time of the London dataset

**Figure 1: Query time of Zoom-SVD compared to other SVD methods. The starting point $t_s$ and the ending point $t_e$ are arbitrarily chosen, and we increase time range $t_e - t_s$ from $10^4$ to $3.2 \times 10^5$. (a) The query time of Zoom-SVD is up to $9.6\times$ faster than that of the second best method in Activity dataset. (b) Zoom-SVD is up to $15\times$ faster than the second best method in the query phase of Gas dataset. (c) Zoom-SVD is up to $12\times$ faster than the second best method in the query phase of London dataset.**

**Table 1: Symbol description.**

| Symbol | Description |
|---|---|
| $b$ | Initial block size |
| $\xi$ | Threshold for low-rank approximation |
| $k$ | Number of singular values |
| $k_{(i)}$ | Number of singular values in $i$-th block |
| $[\mathbf{X}\,;\,\mathbf{Y}]$ | Vertical concatenation of two matrices $\mathbf{X}$ and $\mathbf{Y}$ |
| $\mathbf{A}$ | Raw multiple time series data |
| $\mathbf{A}^{(i)}$ | i-th block of $\mathbf{A}$ |
| $\mathbf{U}_{(i)}$ | Left singular vector matrix of $\mathbf{A}^{(i)}$ |
| $\Sigma_{(i)}$ | Singular value matrix of $\mathbf{A}^{(i)}$ |
| $\mathbf{V}_{(i)}$ | Right singular vector matrix of $\mathbf{A}^{(i)}$ |
| $\mathbf{U}_{(S:E)}$ | Left singular vector matrix computed in query phase |
| $\Sigma_{(S:E)}$ | Singular value matrix computed in query phase |
| $\mathbf{V}_{(S:E)}$ | Right singular vector matrix computed in query phase |
| $\mathcal{U}$ | Set of left singular vector matrix $\mathbf{U}_{(i)}$ |
| $\mathcal{S}$ | Set of singular value matrix $\Sigma_{(i)}$ |
| $\mathcal{V}$ | Set of right singular vector matrix $\mathbf{V}_{(i)}$ |
| $[t_s, t_e]$ | Time range query |
| $t_s$ | Starting point of time range query |
| $t_e$ | Ending point of time range query |
| $S$ | Index of block matrix corresponding to $t_s$ |
| $E$ | Index of block matrix corresponding to $t_e$ |

preliminaries and formally define the problem in Section 2, propose our method Zoom-SVD in Section 3, present experimental results in Section 4, demonstrate the case study in Section 5, discuss related works in Section 6, and conclude in Section 7.

## 2 PRELIMINARIES

We describe preliminaries on singular value decomposition (SVD) and incremental SVD (Sections 2.1 and 2.2). We then define the problem handled in this paper (Section 2.3). Table 1 lists the symbols used in this paper.

### 2.1 Singular Value Decomposition (SVD)

SVD is a decomposition method for finding latent factors in a matrix $\mathbf{A} \in \mathbb{R}^{t \times c}$. Suppose the rank of the matrix $\mathbf{A}$ is $r$. Then, SVD of $\mathbf{A}$ is represented as $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$ where $\Sigma$ is an $r \times r$ diagonal matrix whose diagonal entries are singular values. The $i$-th singular value $\sigma_i$ is located in $\Sigma_{i,i}$ where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$. $\mathbf{U} \in \mathbb{R}^{t \times r}$ is called the left singular vector matrix (or a set of left singular vectors) of $\mathbf{A}$; $\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_r]$ is a column orthogonal matrix where $\mathbf{u}_1, \cdots, \mathbf{u}_r$ are the eigenvectors of $\mathbf{A}\mathbf{A}^T$. $\mathbf{V} \in \mathbb{R}^{c \times r}$ is the right singular vector matrix of $\mathbf{A}$; $\mathbf{V} = [\mathbf{v}_1 \cdots \mathbf{v}_r]$ is a column orthogonal matrix where $\mathbf{v}_1, \cdots, \mathbf{v}_r$ are the eigenvectors of $\mathbf{A}^T\mathbf{A}$. Note that the singular vectors in $\mathbf{U}$ and $\mathbf{V}$ are used as hidden factors to analyze the data matrix $\mathbf{A}$.

**Low-rank approximation.** Low-rank approximation effectively approximates the original data matrix based on SVD. The key idea of the low-rank approximation is to keep top-$k$ highest singular values and corresponding singular vectors where $k$ is a number smaller than the rank $r$ of the original matrix. The low-rank approximation of $\mathbf{A}$ is represented as follows:

$$\mathbf{A} \simeq \mathbf{U}_k\Sigma_k\mathbf{V}_k^{\mathbf{T}} \triangleq \hat{\mathbf{A}} \text{ s.t. } k < r$$

where the reconstruction data $\hat{\mathbf{A}}$ is the low-rank approximation of $\mathbf{A}$, $\mathbf{U}_k = [\mathbf{u}_1, \cdots, \mathbf{u}_k]$, $\mathbf{V}_k = [\mathbf{v}_1, \cdots, \mathbf{v}_k]$, and $\Sigma_k = diag(\sigma_1, \cdots, \sigma_k)$. The error of the low-rank approximation is represented as follows:

$$\|\mathbf{A} - \hat{\mathbf{A}}\|_F^2 = \| \sum_{i=k+1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \|_F^2 = \sum_{i=k+1}^{r} \sigma_i^2$$

where $\|\cdot\|_F$ is the Frobenius norm of a matrix, and $r$ is the rank of the original matrix. The parameter $k$ for low-rank approximation is determined by the following equation:

$$k^* = \arg\min_{1 \leq k \leq r} f(k) = \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2}, \text{ s.t. } f(k) \geq \xi \quad (1)$$

where $\xi$ is a threshold between 0 and 1.

### 2.2 Incremental SVD

Incremental SVD dynamically calculates the SVD result of a matrix with newly arrived data rows. Suppose that we have the SVD result $\mathbf{U}_t, \Sigma_t$, and $\mathbf{V}_t^T$ of a data matrix $\mathbf{A}_t \in \mathbb{R}^{n \times c}$ at time $t$. When an $m \times c$ matrix $\mathbf{A}_{t+1}$ arrives at time $t+1$, the purpose of incremental SVD is to efficiently obtain the SVD result of $[\mathbf{A}_t\,;\,\mathbf{A}_{t+1}]$ based on the previous result $\mathbf{U}_t, \Sigma_t$, and $\mathbf{V}_t^T$. Note that $[\mathbf{X}\,;\,\mathbf{Y}]$ denotes the vertical concatenation of two matrices $\mathbf{X}$ and $\mathbf{Y}$. Incremental SVD is used to analyze patterns in time series data [35], and several efficient

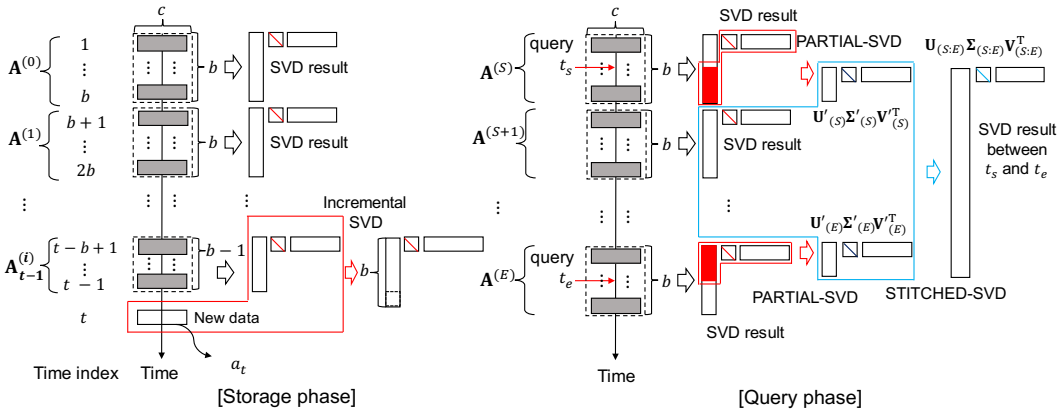



**Figure 2: Overview of Zoom-SVD. In the storage phase, Zoom-SVD stores the SVD results for length-$b$ blocks of multiple time series data, updates the SVD result of the most recent block of a newly arrived data, and discards previously arrived data. In the query phase, Zoom-SVD computes the SVD result in a given time range $[t_s, t_e]$ based on the small SVD results. The query phase exploits Partial-SVD (Section 3.3.1) and Stitched-SVD (Section 3.3.2) modules to process the time range query.**

methods for incremental SVD were proposed [2, 32]. This incremental SVD technique is exploited in our method to incrementally compress and store the data (see Algorithm 1 in Section 3.2).

## 2.3 Problem Definition

We formally define the time range query problem as follows:

Problem 1. (Time Range Query on Multiple Time Series)

- ***Given:*** *a time range $[t_s, t_e]$, and multiple time series data represented by a matrix $\mathbf{A} \in \mathbb{R}^{t \times c}$ where $t$ is the length of the time dimension, and $c$ is the number of the time series,*
- ***Find:*** *the SVD result of the sub-matrix of $\mathbf{A}$ in the time range quickly, without storing all of $\mathbf{A}$. The SVD result includes $\mathbf{U} \in \mathbb{R}^{(t_e - t_s + 1) \times k}$, $\Sigma \in \mathbb{R}^{k \times k}$, and $\mathbf{V} \in \mathbb{R}^{c \times k}$ where $k$ is the rank of the sub-matrix.*

Applying the standard SVD or incremental SVD for the time range query is impractical for the following reasons. Standard SVD needs to extract the sub-matrix corresponding to the time range before performing decomposition. Iwen et al. [8] proposed a hierarchical and distributed approach for computing $\Sigma$ and $\mathbf{V}$ except for $\mathbf{U}$ of a whole matrix $\mathbf{A}$. Zadeh et al. [42] introduce Tall and Skinny SVD which obtains $\Sigma$ and $\mathbf{V}$ by computing eigen-decomposition of $\mathbf{A}^\mathbf{T}\mathbf{A}$, and then computes $\mathbf{U}$ using $\Sigma$, $\mathbf{V}$, and $\mathbf{A}$. Halko et al. [7] propose Randomized SVD which computes SVD of $\mathbf{A}$ using randomized approximation techniques. However, such methods are inefficient because they need to compute SVDs from scratch for multiple overlapping queries. Furthermore, those methods need to keep the entire time series data $\mathbf{A}$, which is practically infeasible in many streaming applications. Incremental SVD considers updates only on newly added data, and thus cannot perform SVD on a specific time range.

To address these limitations, we propose an efficient method for the time range query in Section 3.

# 3 PROPOSED METHOD

We propose Zoom-SVD, a fast and space-efficient method for extracting key patterns from multiple time series data in an arbitrary time range. We first give an overview of Zoom-SVD in Section 3.1. We describe details of Zoom-SVD in Sections 3.2 and 3.3. Finally, we analyze Zoom-SVD's time and space complexities in Section 3.4.

## 3.1 Overview

Zoom-SVD efficiently extracts key patterns from multiple time series data in an arbitrary time range using SVD. The main challenges for the time range query problem (Problem 1) are as follows:

(1) **Minimize the space cost.** The amount of multiple time series data increases over time. How can we reduce the space while supporting time range queries?
(2) **Minimize the time cost.** How can we quickly compute SVD of multiple time series data in an arbitrary time range?

We address the above challenges with the following ideas:

(1) **Compress multiple time series data (Section 3.2).** Zoom-SVD compresses the raw data using incremental SVD, and discards the raw data in the storage phase.
(2) **Avoid reconstructing the original data from the SVD results computed in the storage phase (Sections 3.3.1 and 3.3.2).** We propose Partial-SVD and Stitched-SVD to compute SVD without reconstructing the original data corresponding to the query time range.
(3) **Optimize the computational time of Stitched-SVD (Section 3.3.2).** We optimize the performance of Stitched-SVD by reducing numerical computations using a block matrix structure.

Zoom-SVD comprises two phases: storage phase and query phase. In the storage phase (Algorithm 1), Zoom-SVD stores the SVD results corresponding to length-$b$ blocks in the time series data in order to support time range queries as shown in Figure 2. When a new data arrives, Zoom-SVD incrementally updates the SVD result with the newly arrived data, block by block. In the query phase (Algorithms 2), Zoom-SVD returns the SVD result for a given time range $[t_s, t_e]$. The query phase utilizes our proposed Partial-SVD and Stitched-SVD modules to process the time range query. Partial SVD (Algorithm 3) manipulates the SVD result containing $t_s$ (or $t_e$) to match the query time range as shown in Figure 2. Stitched-SVD (Algorithms 2) efficiently computes the SVD result between $t_s$ and $t_e$ by stitching the SVD results for blocks in the time range.

## 3.2 Storage Phase of Zoom-SVD

Given multiple time series stream $\mathbf{A}$, the objective of the storage phase is to incrementally compress the input data and discard the original input data $\mathbf{A}$ to achieve space efficiency. A naive incremental SVD would update one large SVD result when the data are newly added. However, this approach is impractical because the processing cost for the newly added data increases over time. Also, the naive incremental SVD does not support a time range query quickly in the query phase because it manipulates the large SVD result stored for the total time regardless of the query time range.

The storage phase of Zoom-SVD (Algorithm 1) is designed to efficiently process newly added data and quickly support time range queries. Given multiple time series data $\mathbf{A}$, the storage phase of Zoom-SVD (Algorithm 1) incrementally compresses the input data block by block using incremental SVD, and discards the original input data $\mathbf{A}$ to reduce space cost. Assume the multiple time series data are represented by a matrix $\mathbf{A} \in \mathbb{R}^{t \times c}$ where $t$ is time length, and $c$ is the number of time series (e.g., sensors). We conceptually divide the matrix $\mathbf{A}$ into length-$b$ blocks represented by $\mathbf{A}^{(i)} \in \mathbb{R}^{b \times c}$ as shown in Figure 2. We then store the low-rank approximation result of each block matrix $\mathbf{A}^{(i)}$, where we exploit an incremental SVD method in the process. We formally define the block matrix $\mathbf{A}^{(i)}$ in Definition 1.

Definition 1 (Block matrix $\mathbf{A}^{(i)}$). *Suppose a multivariate time series is* $\mathbf{A} = \left[\mathbf{a}_1; \mathbf{a}_2; \cdots; \mathbf{a}_t\right]$ *where* $\mathbf{a}_j \in \mathbb{R}^{1 \times c}$ *is the $j$-th row vector of* $\mathbf{A}$*, and* $[;]$ *denotes the vertical concatenation of vectors. The $i$-th block matrix* $\mathbf{A}^{(i)}$ *is then represented as follows:*

$$\mathbf{A}^{(i)} \equiv \left[\mathbf{a}_{b \times i+1}; \mathbf{a}_{b \times i+2}; \cdots; \mathbf{a}_{b \times i+b}\right]$$

*where $b$ is a block size. In addition,* $\mathbf{A}^{(i)}_{t-1}$ *denotes the $i$-th block matrix at time $t-1$ where $i$ indicates the index of the most recent block as shown in Figure 2. Note that the number of rows in* $\mathbf{A}^{(i)}_{t-1}$ *is less than or equal to $b$.* □

The computed SVD result $\mathbf{U}_{(i)}$, $\Sigma_{(i)}$, and $\mathbf{V}_{(i)}$ of each block matrix $\mathbf{A}^{(i)}$ are stored as follows.

Definition 2 (Sets of SVD results $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}$). *The sets $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}$ store the SVD results* $\mathbf{U}_{(i)}$, $\Sigma_{(i)}$, *and* $\mathbf{V}_{(i)}$ *for all $i$, respectively.* □

Note that the original time series data are discarded, and we store only the SVD results which occupy less space than the original data. The SVD results for block matrices are used in the query phase (Algorithm 2). Now we are ready to describe the details of the storage phase.

The storage phase (Algorithm 1) compresses the multiple time series data block by block using incremental SVD to support time range queries. When new multiple time series data $\mathbf{a}_t \in \mathbb{R}^{1 \times c}$ are given at time $t$ (line 2), we generate the new SVD result of $\mathbf{a}_t$ for the next block matrix $\mathbf{A}^{(i)}_t$ if the SVD result are stored in $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}$ at time $t-1$ (lines 3 and 4). If not, we have the SVD result $\mathbf{U}_{(i,t-1)}$, $\Sigma_{(i,t-1)}$, and $\mathbf{V}_{(i,t-1)}$ of the most recent block matrix $\mathbf{A}^{(i)}_{t-1}$ which is the $i$-th block matrix at time $t-1$. Assume that we have the SVD result $\mathbf{U}_{(i,t-1)}$, $\Sigma_{(i,t-1)}$, and $\mathbf{V}_{(i,t-1)}$ of $\mathbf{A}^{(i)}_{t-1}$ (i.e., the block matrix from time $t-b+1$ to $t-1$ as seen in Figure 2). We then update the SVD result into $\mathbf{U}_{(i,t)}$, $\Sigma_{(i,t)}$, and $\mathbf{V}_{(i,t)}$ for the new data $\mathbf{a}_t$ using an incremental SVD method (line 6). If the number of rows of $\mathbf{U}_{(i,t)}$ is $b$, we put the SVD result $\mathbf{U}_{(i,t)}$, $\Sigma_{(i,t)}$, and $\mathbf{V}_{(i,t)}$ into $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}$, respectively (lines 8∼11). Equations (2) and (3) represent the details of how to update the SVD result of $\mathbf{A}^{(i)}_{t-1}$ for the new incoming data $\mathbf{a}_t$, when $\mathbf{A}^{(i)}_{t-1}$ contains $b-1$ rows. $\mathbf{A}^{(i)}_t$ is represented by $\mathbf{a}_t$ and the SVD result of $\mathbf{A}^{(i)}_{t-1}$ in Equation (2):

$$\begin{aligned}\mathbf{A}^{(i)}_t = \begin{bmatrix}\mathbf{A}^{(i)}_{t-1}\\ \mathbf{a}_t\end{bmatrix} &\simeq \begin{bmatrix}\mathbf{U}_{(i,t-1)}\Sigma_{(i,t-1)}\mathbf{V}^{\mathrm{T}}_{(i,t-1)}\\ \mathbf{a}_t\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{U}_{(i,t-1)} & \mathbf{O}_{(b-1)\times 1}\\ \mathbf{O}_{1\times k_{t-1}} & \mathbf{I}_{1\times 1}\end{bmatrix}\begin{bmatrix}\Sigma_{(i,t-1)}\mathbf{V}^{\mathrm{T}}_{(i,t-1)}\\ \mathbf{a}_t\end{bmatrix}\end{aligned} \tag{2}$$

where $\mathbf{O}_{x\times y}$ is an $x \times y$ zero matrix, and $\mathbf{I}_{x\times x}$ is an $x \times x$ identity matrix. We then perform SVD to decompose $\begin{bmatrix}\Sigma_{(i,t-1)}\mathbf{V}^{\mathrm{T}}_{(i,t-1)}\\ \mathbf{a}_t\end{bmatrix} \in \mathbb{R}^{(k_{t-1}+1)\times c}$ into $\tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^{\mathrm{T}}$:

$$\begin{aligned}&\begin{bmatrix}\mathbf{U}_{(i,t-1)} & \mathbf{O}_{(b-1)\times 1}\\ \mathbf{O}_{1\times k_{t-1}} & \mathbf{I}_{1\times 1}\end{bmatrix}\begin{bmatrix}\Sigma_{(i,t-1)}\mathbf{V}^{\mathrm{T}}_{(i,t-1)}\\ \mathbf{a}_t\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{U}_{(i,t-1)} & \mathbf{O}_{(b-1)\times 1}\\ \mathbf{O}_{1\times k_{t-1}} & \mathbf{I}_{1\times 1}\end{bmatrix}\tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^{\mathrm{T}} \triangleq \mathbf{U}_{(i,t)}\Sigma_{(i,t)}\mathbf{V}^{\mathrm{T}}_{(i,t)}\end{aligned} \tag{3}$$

where $\mathbf{U}_{(i,t)} = \begin{bmatrix}\mathbf{U}_{(i,t-1)} & \mathbf{O}_{(b-1)\times 1}\\ \mathbf{O}_{1\times k_{t-1}} & \mathbf{I}_{1\times 1}\end{bmatrix}\tilde{\mathbf{U}}$, $\Sigma_{(i,t)} = \tilde{\Sigma}$, and $\mathbf{V}^{\mathrm{T}}_{(i,t)} = \tilde{\mathbf{V}}^{\mathrm{T}}$. Note that $\mathbf{U}_{(i,t)}$ is a column orthogonal matrix since it is the product of two orthogonal matrices. $\mathbf{V}_{(i,t)}$ is also column orthogonal, and $\Sigma_{(i,t)}$ is a diagonal matrix whose diagonal entries are sorted in the descending order. Hence, $\mathbf{U}_{(i,t)}$, $\Sigma_{(i,t)}$, and $\mathbf{V}_{(i,t)}$ are considered as the SVD result of $\mathbf{A}^{(i)}_t$ by the definition of SVD [40]. The time index $t$ can be omitted as in $\mathbf{U}_{(i)}$, $\Sigma_{(i)}$, $\mathbf{V}_{(i)}$, and $\mathbf{A}^{(i)}$, as described in Definitions 1 and 2, if the number of rows of $\mathbf{U}_{(i,t)}$ is $b$.

**Algorithm 1** Storage phase of Zoom-SVD

**Input:** multiple time series data $\mathbf{A}$
**Output:** set $\mathcal{U}$ of left singular vector matrix $\mathbf{U}_{(i)}$, set $\mathcal{S}$ of singular value matrix $\Sigma_{(i)}$, and set $\mathcal{V}$ of right singular vector matrix $\mathbf{V}_{(i)}$
**Parameters:** block size $b$

1: **Initialize:** $\mathbf{U}_{(0,1)}$, $\Sigma_{(0,1)}$, and $\mathbf{V}_{(0,1)} \leftarrow$ SVD result of $\mathbf{a}_1$, block index $i \leftarrow 0$, and time index $t \leftarrow 2$
2: **while** $\mathbf{a}_t$ is newly arriving **do**
3: **if** $\mathbf{U}_{(i,t-1)}$, $\Sigma_{(i,t-1)}$, $\mathbf{V}_{(i,t-1)}$ do not exist **then**
4: $\mathbf{U}_{(i,t)}$, $\Sigma_{(i,t)}$, $\mathbf{V}_{(i,t)} \leftarrow$ SVD result of $\mathbf{a}_t$
5: **else**
6: $\mathbf{U}_{(i,t)}$, $\Sigma_{(i,t)}$, $\mathbf{V}_{(i,t)} \leftarrow$ Incremental-SVD ($\mathbf{U}_{(i,t-1)}$, $\Sigma_{(i,t-1)}$, $\mathbf{V}_{(i,t-1)}$, $\mathbf{a}_t$)
7: **end if**
8: **if** the number of rows of $\mathbf{U}_{(i,t)}$ is equal to $b$ **then**
9: $\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathbf{U}_{(i,t)}\}$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{\Sigma_{(i,t)}\}$, $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{V}_{(i,t)}\}$
10: $i \leftarrow i+1$
11: **end if**
12: $t \leftarrow t+1$
13: **end while**

## 3.3 Query Phase of Zoom-SVD

Given the starting point $t_s$ and the ending point $t_e$ of a time range query, the goal of the query phase of Zoom-SVD is to obtain the SVD result from $t_s$ to $t_e$. A naive approach would reconstruct the time series data from the SVD results of the block matrices ranged between $t_s$ and $t_e$, and perform SVD on the reconstructed data in the range. However, this approach requires heavy computations

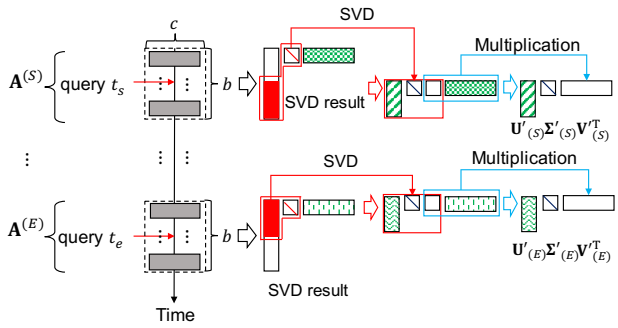



**Figure 3: Example of Partial-SVD. Given a time range query $[t_s, t_e]$, we remove rows of $\mathbf{U}_{(S)}$ and $\mathbf{U}_{(E)}$ which are out of the query time range. Partial-SVD exploits SVD on the filtered matrices for left singular vector matrices and singular value matrices within the query time range (red-colored boxes). Then Partial-SVD computes right singular vector matrices of the output SVD results by multiplying relevant matrices (blue-colored boxes).**

**Algorithm 2** Query phase of Zoom-SVD

**Input:** sets $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}$ of block SVD results, starting point $t_s$, and ending point $t_e$
**Output:** SVD result $\mathbf{U}_{(S:E)}$, $\mathbf{\Sigma}_{(S:E)}$, and $\mathbf{V}_{(S:E)}$ in $[t_s, t_e]$
1: $\mathbf{U'}_{(S)}, \mathbf{\Sigma'}_{(S)}, \mathbf{V'}_{(S)}, \mathbf{U'}_{(E)}, \mathbf{\Sigma'}_{(E)}$, and $\mathbf{V'}_{(E)}$ $\leftarrow$ Partial-SVD $(t_s, t_e, \mathcal{U}, \mathcal{S}, \mathcal{V})$
2: $\mathbf{\Sigma V}^{\mathrm{T}} \leftarrow [\mathbf{\Sigma'}_{(S)}\mathbf{V'}^{\mathrm{T}}_{(S)}; \mathbf{\Sigma}_{(S+1)}\mathbf{V}^{\mathrm{T}}_{(S+1)}; \cdots; \mathbf{\Sigma'}_{(E)}\mathbf{V'}^{\mathrm{T}}_{(E)}]$
3: $\mathbf{U}_r, \mathbf{\Sigma}_r, \mathbf{V}^{\mathrm{T}}_r \leftarrow$ low-rank approximation of $\mathbf{\Sigma V}^{\mathrm{T}}$ using SVD
4: $\mathbf{V}_{(S:E)} \leftarrow \mathbf{V}_r$ and $\mathbf{\Sigma}_{(S:E)} \leftarrow \mathbf{\Sigma}_r$
5: $\mathbf{U}_{(S:E)} \leftarrow [\mathbf{U'}_{(S)}\mathbf{U'}_{r(S)}\ ;\ \mathbf{U}_{(S+1)}\mathbf{U}_{r(S+1)}\ ;\ \cdots\ ;\ \mathbf{U'}_{(E)}\mathbf{U'}_{r(E)}]$
6: **return** $\mathbf{U}_{(S:E)}$, $\mathbf{\Sigma}_{(S:E)}$, and $\mathbf{V}_{(S:E)}$

especially for a long time range query, and thus is not appropriate for serving time range queries quickly.

We propose two sub-modules, Partial-SVD and Stitched-SVD, which are used in the query phase of our proposed method (Algorithm 2) to efficiently process time range queries by avoiding reconstruction of the raw data. Let $S$ be the index of the block matrix including $t_s$, and $E$ be the index of the block matrix including $t_e$. Partial-SVD (Algorithm 3) adjusts the time range of the SVD results for $\mathbf{A}^{(S)}$ and $\mathbf{A}^{(E)}$ as seen in the red-colored boxes of Figure 3 (line 1 of Algorithm 2). Stitched-SVD combines the SVD results of Partial-SVD and those of block matrices from $\mathbf{A}^{(S+1)}$ to $\mathbf{A}^{(E-1)}$ (lines 2 to 5 in Algorithm 2). We describe the details of Partial-SVD and Stitched-SVD in Sections 3.3.1 and 3.3.2, respectively.

*3.3.1* ***Partial-SVD.*** This module manipulates the SVD results of block matrices $\mathbf{A}^{(S)}$ and $\mathbf{A}^{(E)}$ to return the SVD results in a given time range $[t_s, t_e]$. As seen in Figure 2, $\mathbf{A}^{(S)}$ may contain the time range before $t_s$, and $\mathbf{A}^{(E)}$ may include the time range after $t_e$. Note that those time ranges are out of the time range of the given query; thus, our goal for this module is to extract SVD results from $\mathbf{A}^{(S)}$ and $\mathbf{A}^{(E)}$ according to the time range query without reconstructing raw data. Figure 3 depicts the operation of Partial-SVD. For the block matrix $\mathbf{A}^{(S)}$ and its SVD $\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)}\mathbf{V}_{(S)}$, Partial-SVD first eliminates rows of left singular vector matrix $\mathbf{U}_{(S)}$ which are out of the query time range. After that, Partial-SVD multiplies the remaining left singular vector matrix $\mathbf{X}_s\mathbf{U}_{(S)}$ with the singular value

**Algorithm 3** Partial-SVD

**Input:** starting point $t_s$, ending point $t_e$, and sets $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}$ of SVD results
**Output:** SVD results $\mathbf{U'}_{(S)}, \mathbf{\Sigma'}_{(S)}, \mathbf{V'}_{(S)}, \mathbf{U'}_{(E)}, \mathbf{\Sigma'}_{(E)}$, and $\mathbf{V'}_{(E)}$ of $\mathbf{A}^{(S)}$ and $\mathbf{A}^{(E)}$ within the time range
1: $S \leftarrow t_s/b$ and $E \leftarrow t_e/b$
2: $b_S \leftarrow b - t_s\%b$ and $b_E \leftarrow t_e\%b$ (%: the modulus operator)
3: construct $\mathbf{X}_s$ and $\mathbf{X}_e$ based on $b_S$ and $b_E$ as in Equation (4)
4: $\tilde{\mathbf{U}}_{(S)}, \tilde{\mathbf{\Sigma}}_{(S)}, \tilde{\mathbf{V}}^{\mathrm{T}}_{(S)} \leftarrow$ low-rank approximation of $\mathbf{X}_s\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)}$ using SVD
5: $\mathbf{U'}_{(S)} \leftarrow \tilde{\mathbf{U}}_{(S)}$, $\mathbf{\Sigma'}_{(S)} \leftarrow \tilde{\mathbf{\Sigma}}_{(S)}$, and $\mathbf{V'}^{\mathrm{T}}_{(S)} \leftarrow \tilde{\mathbf{V}}^{\mathrm{T}}_{(S)}\mathbf{V}^{\mathrm{T}}_{(S)}$
6: $\tilde{\mathbf{U}}_{(E)}, \tilde{\mathbf{\Sigma}}_{(E)}, \tilde{\mathbf{V}}^{\mathrm{T}}_{(E)} \leftarrow$ low-rank approximation of $\mathbf{X}_e\mathbf{U}_{(E)}\mathbf{\Sigma}_{(E)}$ using SVD
7: $\mathbf{U'}_{(E)} \leftarrow \tilde{\mathbf{U}}_{(E)}$, $\mathbf{\Sigma'}_{(E)} \leftarrow \tilde{\mathbf{\Sigma}}_{(E)}$, and $\mathbf{V'}^{\mathrm{T}}_{(E)} \leftarrow \tilde{\mathbf{V}}^{\mathrm{T}}_{(E)}\mathbf{V}^{\mathrm{T}}_{(E)}$
8: **return** $\mathbf{U'}_{(S)}$, $\mathbf{\Sigma'}_{(S)}$, $\mathbf{V'}_{(S)}$, $\mathbf{U'}_{(E)}$, $\mathbf{\Sigma'}_{(E)}$, and $\mathbf{V'}_{(E)}$

matrix $\mathbf{\Sigma}_{(S)}$, and performs SVD $\tilde{\mathbf{U}}_{(S)}\tilde{\mathbf{\Sigma}}_{(S)}\tilde{\mathbf{V}}^{\mathrm{T}}_{(S)} \leftarrow \mathbf{X}_s\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)}$ of the resulting matrix. The resulting singular vector matrix $\tilde{\mathbf{U}}_{(S)}$ and the singular value matrix $\tilde{\mathbf{\Sigma}}_{(S)}$ constitute the output of Partial-SVD. The remaining right singular vector matrix output of Partial-SVD is computed by multiplying the right singular vector matrix $\tilde{\mathbf{V}}^{\mathrm{T}}_{(S)}$ with $\mathbf{V}^{\mathrm{T}}_{(S)}$. Similar operations are performed for the block matrix $\mathbf{A}^{(E)}$ and its SVD $\mathbf{U}_{(E)}\mathbf{\Sigma}_{(E)}\mathbf{V}_{(E)}$.

Now, we describe the details of this module (Algorithm 3). We first introduce elimination matrices which are used in Partial-SVD to adjust the time range.

Definition 3 (Elimination matrices). *Suppose $r_S$ is the number of rows to be eliminated in $\mathbf{A}^{(S)}$ according to $t_s$. Then $b_S = b - r_S$ is the number of remaining rows in $\mathbf{A}^{(S)}$. Similarly, let $r_E$ be the number of rows to be eliminated in $\mathbf{A}^{(E)}$ according to $t_e$; then $b_E = b - r_E$ is the number of remaining rows in $\mathbf{A}^{(E)}$. The elimination matrices $\mathbf{X}_s$ and $\mathbf{X}_e$ for $\mathbf{A}^{(S)}$ and $\mathbf{A}^{(E)}$ are defined as follows:*

$$\mathbf{X}_s = \begin{bmatrix}\mathbf{O}_{b_S \times r_S} & \mathbf{I}_{b_S \times b_S}\end{bmatrix} \mathbf{X}_e = \begin{bmatrix}\mathbf{I}_{b_E \times b_E} & \mathbf{O}_{b_E \times r_E}\end{bmatrix} \quad (4)$$

□

The matrices $\mathbf{A}^{(S)}$ and $\mathbf{A}^{(E)}$ are multiplied to the elimination matrices, and the time ranges of the resulting matrices $\mathbf{X}_s\mathbf{A}^{(S)}$ and $\mathbf{X}_e\mathbf{A}^{(E)}$ are within the query time range $[t_s, t_e]$. Partial-SVD constructs those elimination matrices based on $b_S$ and $b_E$ (line 3 of Algorithm 3). The filtered block matrix $\mathbf{X}_s\mathbf{A}^{(S)}$ is given by

$$\mathbf{X}_s\mathbf{A}^{(S)} \simeq \mathbf{X}_s(\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)}\mathbf{V}^{\mathrm{T}}_{(S)}) = (\mathbf{X}_s\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)})\mathbf{V}^{\mathrm{T}}_{(S)} \quad (5)$$

where $\mathbf{A}^{(S)} \simeq \mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)}\mathbf{V}^{\mathrm{T}}_{(S)}$ was computed at the storage phase. Partial-SVD decomposes $\mathbf{X}_s\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)}$ into $\tilde{\mathbf{U}}_{(S)}\tilde{\mathbf{\Sigma}}_{(S)}\tilde{\mathbf{V}}^{\mathrm{T}}_{(S)}$ via SVD and low-rank approximation with threshold $\xi$ since $\mathbf{X}_s\mathbf{U}_{(S)}$ is not a column orthogonal matrix, and $\mathbf{X}_s\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)}\mathbf{V}^{\mathrm{T}}_{(S)}$ is not a form of the SVD result; then, Equation (5) is written as follows:

$$(\mathbf{X}_s\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)})\mathbf{V}^{\mathrm{T}}_{(S)} \simeq \tilde{\mathbf{U}}_{(S)}\tilde{\mathbf{\Sigma}}_{(S)}(\tilde{\mathbf{V}}_{(S)}\mathbf{V}_{(S)})^{\mathrm{T}} = \mathbf{U'}_{(S)}\mathbf{\Sigma'}_{(S)}\mathbf{V'}^{\mathrm{T}}_{(S)} \quad (6)$$

where $\mathbf{U'}_{(S)} = \tilde{\mathbf{U}}_{(S)}$, $\mathbf{\Sigma'}_{(S)} = \tilde{\mathbf{\Sigma}}_{(S)}$, and $\mathbf{V'}^{\mathrm{T}}_{(S)} = \tilde{\mathbf{V}}^{\mathrm{T}}_{(S)}\mathbf{V}^{\mathrm{T}}_{(S)}$. In line 4, Partial-SVD performs SVD on $\mathbf{X}_s\mathbf{U}_{(S)}\mathbf{\Sigma}_{(S)}$, and in line 5 it computes $\mathbf{V'}^{\mathrm{T}}_{(S)} = \tilde{\mathbf{V}}^{\mathrm{T}}_{(S)}\mathbf{V}^{\mathrm{T}}_{(S)}$. Partial-SVD similarly computes the SVD result of $\mathbf{X}_e\mathbf{A}^{(E)}$ in lines 6~7 of Algorithm 3.

*3.3.2* ***Stitched-SVD.*** This module combines the Partial-SVD of $\mathbf{A}^{(S)}$ and $\mathbf{A}^{(E)}$, and the stored SVD results of blocks matrices $\mathbf{A}^{(S+1)}, \mathbf{A}^{(S+1)}, \cdots, \mathbf{A}^{(E-1)}$ in the query time range $[t_s, t_e]$ to return the final SVD result corresponding to the query range as shown in Figure 2. A naive approach is to reconstruct the data blocks using the stored SVD results and perform SVD on the reconstructed data of the given query time range. However, this approach cannot provide fast query speed for a long time range due to heavy computations induced by the reconstruction and the following SVD. The goal of Stitched-SVD is to efficiently stitch the SVD results in the query time range by avoiding reconstruction and minimizing the numerical computation of matrix multiplication.

Specifically, Stitched-SVD stitches several consecutive block SVD results together to compute the SVD corresponding to the query time range: is.e., it combines the SVD result $\mathbf{U}_{(i)}, \Sigma_{(i)}$, and $\mathbf{V}_{(i)}$ of the $i$th block matrix $\mathbf{A}^{(i)}$, for $i = S..E$, to compute the SVD $\mathbf{U}_{(S:E)}$, $\Sigma_{(S:E)}$, and $\mathbf{V}_{(S:E)}$. The main idea is 1) to carefully decouple the matrices $\mathbf{U}_{(i)}$ from $\Sigma_{(i)}\mathbf{V}_{(i)}$, 2) construct a stacked matrix containing $\Sigma_{(i)}\mathbf{V}_{(i)}$ for $i = S..E$, 3) perform SVD on the stacked matrix to get the singular value matrix and the right singular vector matrix of the final SVD result, and 4) carefully combine $\mathbf{U}_{(i)}$ with the left singular matrix of SVD of the stacked matrix to get the left singular vector matrix of the final SVD result.

Lines 2 to 5 of Algorithm 2 present how stitched SVD matrices are computed. First, we construct $\Sigma\mathbf{V}^{\mathrm{T}}$ based on the block matrix structure where $\Sigma\mathbf{V}^{\mathrm{T}}$ is equal to $\left[\Sigma'_{(S)}\mathbf{V}'^{\mathrm{T}}_{(S)}\,;\,\Sigma_{(S+1)}\mathbf{V}^{\mathrm{T}}_{(S+1)}\,;\,\cdots\,;\,\Sigma'_{(E)}\mathbf{V}'^{\mathrm{T}}_{(E)}\right]^T$. After organizing the block matrix structure of $\Sigma\mathbf{V}^{\mathrm{T}}$, we define block diagonal matrix $diag(S:E)$ as follows.

Definition 4 (Block diagonal matrix). *Suppose* $\mathbf{U}'_{(S)}$ *and* $\mathbf{U}'_{(E)}$ *are the left singular vector matrices produced by Partial-SVD. Let* $\mathbf{U}_{(S+1)}, \mathbf{U}_{(S+2)}, \cdots, \mathbf{U}_{(E-1)}$ *be the left singular vector matrices in* $\mathcal{U}$. *The block diagonal matrix* $diag(S:E)$ *is defined as follows:*

$$diag(S:E) = \begin{bmatrix} \mathbf{U}'_{(S)} & \mathbf{O} & \cdots & \mathbf{O} \\ \mathbf{O} & \mathbf{U}_{(S+1)} & & \vdots \\ \vdots & & \ddots & \vdots \\ \mathbf{O} & \cdots & \cdots & \mathbf{U}'_{(E)} \end{bmatrix}$$

□

Then, the matrix corresponding to the time range query $[t_s, t_e]$ is represented as follows:

$$\mathbf{A}^{(S:E)} = \begin{bmatrix} \mathbf{X}_s\mathbf{A}^{(S)} \\ \mathbf{A}^{(S+1)} \\ \cdots \\ \mathbf{X}_e\mathbf{A}^{(E)} \end{bmatrix} \simeq \begin{bmatrix} \mathbf{U}'_{(S)}\Sigma'_{(S)}\mathbf{V}'^{\mathrm{T}}_{(S)} \\ \mathbf{U}_{(S+1)}\Sigma_{(S+1)}\mathbf{V}^{\mathrm{T}}_{(S+1)} \\ \cdots \\ \mathbf{U}'_{(E)}\Sigma'_{(E)}\mathbf{V}'^{\mathrm{T}}_{(E)} \end{bmatrix} = diag(S:E)\Sigma\mathbf{V}^{\mathrm{T}} \tag{7}$$

where $\mathbf{X}_s$ and $\mathbf{X}_e$ are elimination matrices of Partial-SVD, and $\Sigma\mathbf{V}^{\mathrm{T}}$ is equal to $\left[\Sigma'_{(S)}\mathbf{V}'^{\mathrm{T}}_{(S)}\,;\,\Sigma_{(S+1)}\mathbf{V}^{\mathrm{T}}_{(S+1)}\,;\,\cdots\,;\,\Sigma'_{(E)}\mathbf{V}'^{\mathrm{T}}_{(E)}\right]^T$. As we apply SVD and low-rank approximation to $\Sigma\mathbf{V}^{\mathrm{T}}$, Equation (7) becomes as follows:

$$\begin{aligned} diag(S:E)\Sigma\mathbf{V}^{\mathrm{T}} &\simeq diag(S:E)\mathbf{U}_r\Sigma_r\mathbf{V}_r^{\mathrm{T}} \\ &= \mathbf{U}_{(S:E)}\Sigma_{(S:E)}\mathbf{V}^{\mathrm{T}}_{(S:E)} \end{aligned} \tag{8}$$

where $\Sigma\mathbf{V}^{\mathrm{T}} \simeq \mathbf{U}_r\Sigma_r\mathbf{V}_r^{\mathrm{T}}$ is computed by low-rank approximation and SVD, $\mathbf{U}_{(S:E)} = diag(S:E)\mathbf{U}_r$, $\Sigma_{(S:E)} = \Sigma_r$, and $\mathbf{V}_{(S:E)} = \mathbf{V}_r$. To avoid matrix multiplication between $\mathbf{U}_r$ and zero sub-matrices of $diag(S:E)$, we split $\mathbf{U}_r$ block by block as follows:

$$\mathbf{U}_r = \left[\mathbf{U}'_{r(S)}\,;\,\mathbf{U}_{r(S+1)}\,;\,\cdots\,;\,\mathbf{U}'_{r(E)}\right]^T$$

where $\mathbf{U}'_{r(S)}$ and $\mathbf{U}'_{r(E)}$ correspond to $\mathbf{U}'_{(S)}$ and $\mathbf{U}'_{(E)}$, respectively, and $\mathbf{U}_{r(i)}$ correspond to $\mathbf{U}_{(i)}$ for $S+1 \le i \le E-1$. Then $\mathbf{U}_{(S:E)} = diag(S:E)\mathbf{U}_r$ of Equation (8) is computed as follows:

$$\begin{aligned} \mathbf{U}_{(S:E)} &= diag(S:E)\mathbf{U}_r \\ &= \left[\mathbf{U}'_{(S)}\mathbf{U}'_{r(S)}\,;\,\mathbf{U}_{(S+1)}\mathbf{U}_{r(S+1)}\,;\,\cdots\,;\,\mathbf{U}'_{(E)}\mathbf{U}'_{r(E)}\right]^T \end{aligned} \tag{9}$$

The column orthogonality of $\mathbf{U}_{(S:E)} = diag(S:E)\mathbf{U}_r$ is established as it is the product of two column orthogonal matrices; also, $\mathbf{V}_{(S:E)} = \mathbf{V}_r$ is column orthogonal. Note that we perform Partial-SVD to satisfy column orthogonal condition before performing Stitched-SVD.

## 3.4 Theoretical Analysis

We theoretically analyze our proposed method Zoom-SVD in terms of time and memory cost. Note that a collection of multiple time series data $\mathbf{A}$ is a dense matrix, and the time complexity to compute SVD of $\mathbf{A} \in \mathbb{R}^{t\times c}$ is $O(min(t^2c, tc^2))$.

**Time Complexity**. We analyze the time complexities of the storage and the query phases in Theorems 1 and 2, respectively.

Theorem 1. *When a vector* $\mathbf{a}_t \in \mathbb{R}^{1\times c}$ *is given at time t, the computation cost of storage phase in Zoom-SVD is* $O(k^2(b+c))$, *where k is the number of singular values.*

Proof. Performing SVD of $\left[\Sigma_{(i,t-1)}\mathbf{V}^{\mathrm{T}}_{(i,t-1)}\,;\,\mathbf{a}_t\right]$ takes $O(min(c^2(1+k_{t-1}), c(1+k_{t-1})^2)$, and multiplication of $\begin{bmatrix}\mathbf{U}_{(i,t-1)} & \mathbf{O}\\ \mathbf{O} & \mathbf{I}\end{bmatrix}$ and $\tilde{\mathbf{U}}$ takes $O(b(k_{t-1}+1)k_t)$ since the row length of $\begin{bmatrix}\mathbf{U}_{(i,t-1)} & \mathbf{O}\\ \mathbf{O} & \mathbf{I}\end{bmatrix}$ is always smaller than or equal to the block size $b$. Assume $k_{t-1}$ and $k_t$ are equal to $k$. The total computational cost of storage phase in Zoom-SVD is $O(min(c^2k, ck^2) + bk^2)$. We simply express the computational cost of storing the incoming data at each time tick as $O(k^2(b+c))$ since the number $c$ of columns is generally greater than $k$. □

In Theorem 1, the computation of storing the incoming data at each time tick takes constant time since $b$ and $c$ are constants and $k$ is smaller than $c$.

Theorem 2. *Given a time range query* $[t_s, t_e]$, *the time cost of query phase (Algorithm 2) is* $O((t_e - t_s)k(k + \frac{c^2}{b}) + k^2(b+c))$.

Proof. It takes $O((b - r_S + c)k'^2_{(S)} + (b - r_E + c)k'^2_{(E)})$ to compute Partial-SVD where $k'_{(S)}$ and $k'_{(E)}$ are the number of singular values computed by Partial-SVD (line 1 in Algorithm 2). The computational time to perform SVD of $\left[\Sigma'_{(S)}\mathbf{V}'^{\mathrm{T}}_{(S)}\,;\,\Sigma_{(S+1)}\mathbf{V}^{\mathrm{T}}_{(S+1)}\,;\,\cdots\,;\,\Sigma'_{(E)}\mathbf{V}'^{\mathrm{T}}_{(E)}\right]$ depends on $O(min(c^2(k'_{(S)} + k'_{(E)} + \sum_{i=S+1}^{E-1} k_{(i)}), c(k'_{(S)} + k'_{(E)} + \sum_{i=S+1}^{E-1} k_{(i)})^2)$ in Stitched-SVD since horizontal and vertical length of the matrix are $c$ and $(k'_{(S)} + k'_{(E)} + \sum_{i=S+1}^{E-1} k_{(i)})$, respectively (lines 2∼ 4 in Algorithm 2). Also, the computational time of block matrix multiplication for $\mathbf{U}_{(S:E)}$ (line 5 in Algorithm 2) takes $O(k_{(S:E)}((b - r_S)k'_{(S)} + (b - r_E)k'_{(E)} + b\sum_{i=S+1}^{E-1} k_{(i)}))$ where $k_{(S:E)}$ is the number of singular values with respect to the SVD result of a given time range $[t_s, t_e]$. Let all $k_{(i)}$'s be $k$ in query phase, $k'_{(S)} + k'_{(E)} +$

**Table 2: Description of real-world multiple time series datasets. Each dataset is a matrix in $\mathbb{R}^{t\times c}$ where $t$ corresponds to total length (time), and $c$ corresponds to the number of time series.**

| Dataset | Total length ($t$) | Attribute ($c$) |
|---|---|---|
| Activity[1] | 382,000 | 41 |
| Gas[2] | 4,107,000 | 16 |
| London[3] | 350,000 | 10 |

$\sum_{i=S+1}^{E-1} k_{(i)}$ be larger than $c$; also, replace $b - r_i$ with block size $b$ since $b$ is always greater than $b - r_i$. Then, the computational time of Partial-SVD and Stitched-SVD takes $O(k^2(b + c))$ and $O((t_e - t_s)k(k + \frac{c^2}{b}))$, respectively. We can simply express the computational cost of Zoom-SVD as $O((t_e - t_s)k(k + \frac{c^2}{b}) + k^2(b + c))$. □

Theorem 2 implies that the computational time of Zoom-SVD in query phase linearly depends on the time range $(t_e - t_s)$.

**Space Complexity.** We analyze the space complexity of the storage phase in Theorem 3.

Theorem 3. *Space complexity of Zoom-SVD for storing data is $O(tk(1 + \frac{k}{b} + \frac{c}{b}))$ where $t$ is the total time length, and $k$ is the number of singular values.*

Proof. At time $t$, we have $\lfloor \frac{t}{b} \rfloor$ SVD results where $\mathbf{U}_{(i)} \in \mathbb{R}^{b_i \times k_{(i)}}$, $\Sigma_{(i)} \in \mathbb{R}^{k_{(i)} \times k_{(i)}}$, and $\mathbf{V}_{(i)} \in \mathbb{R}^{k_{(i)} \times c}$. Therefore, the number of elements in every matrix we have is $O(\sum_{i=1}^{\lfloor \frac{t}{b} \rfloor} k_{(i)}(b + k_{(i)} + c))$. Assuming $k_{(i)}$ is equal to $k$, we briefly express the space cost of Zoom-SVD as $O(tk(1 + \frac{k}{b} + \frac{c}{b}))$. □

The space cost linearly depends on $t$ and $k$, but $k$ is much smaller than the number $c$ of time series. Therefore, Zoom-SVD efficiently compresses incoming data using space linear to time length with low coefficient.

## 4 EXPERIMENT

We aim to answer the following questions to evaluate the performance of our method Zoom-SVD from experiments.

- **Q1. Time cost (Section 4.2).** How quickly does Zoom-SVD process time range queries compared to other methods?
- **Q2. Space cost (Section 4.3).** How much memory space does Zoom-SVD require for compressed blocks?
- **Q3. Time-Space-Accuracy Trade-off (Section 4.4).** What are the tradeoffs between query time, space, and accuracy by Zoom-SVD compared to other baselines?
- **Q4. Parameter (Section 4.5).** How does the block size $b$ in Algorithm 1 affect the performance of Zoom-SVD in terms of time and space?

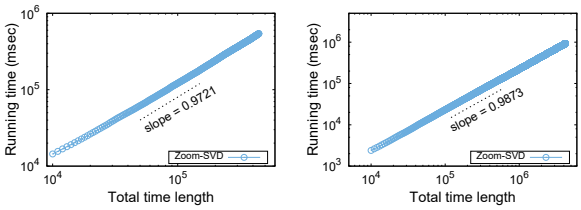
### 4.1 Experiment Settings

**Methods.** We compare Zoom-SVD with Randomized SVD [7], Tall and Skinny SVD [42], and basic SVD of JBLAS library. All these methods are implemented using JAVA, and we use JBLAS, an open source JAVA basic linear algebra package, to support matrix operations.

**Dataset.** We use multiple time series datasets [5, 31] described in Table 2. Activity dataset contains data with timestamps, and 41 measured values such as heartbeat, and inertial measurement units (IMU) data installed in the hands, chest, and ankle. Gas dataset consists of timestamps, and measurement of 16 chemical sensors having 4 different types. Activity and Gas datasets are obtained from UCI repository [4]. London dataset consists of timestamps, and attributes related to London air quality such as nitric oxide, temperature, and so on.

**Parameters**. For all experiments except the ones in Section 4.5, we set the block size $b$ to 1000 in the storage phase. We evaluate the effects of the block size $b$ in Section 4.5. We set the rank $k$ using Equation (1) with $\xi = 0.98$ in the storage phase.

[1]https://archive.ics.uci.edu/ml/datasets/PAMAP2+Physical+Activity+Monitoring
[2]https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+under+dynamic+gas+mixtures
[3]http://www.londonair.org.uk/london/asp/publicdetails.asp




(a) Running time of Zoom-SVD's storage phase in Activity dataset (b) Running time of Zoom-SVD's storage phase in Gas dataset

**Figure 4: Running time of Zoom-SVD's storage phase on Activity and Gas datasets. The results show that the time to update all data is linear to the total time length $t$. The reason is that Zoom-SVD incrementally reads a new input vector, and computes the SVD of the block matrix in a constant time. The pattern on London dataset is similar to the above results.**

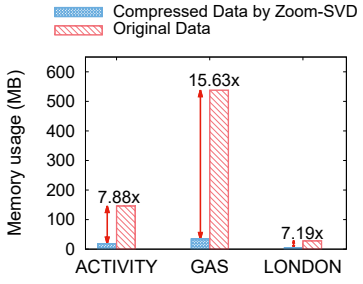



**Figure 5: Space savings by Zoom-SVD. Zoom-SVD requires 7.88×, 15.63×, and 7.19× less space than the original data require for Activity, Gas sensor, and London, respectively.**

### 4.2 Time Cost

We examine the time costs of the storage and the query phases of our proposed method Zoom-SVD.

**Storage phase (Algorithm 1).** We measure the running time of the storage phase for multiple time series data varying the time length. As shown in Figure 4, the running time of Zoom-SVD's storage phase is linearly proportional to the time length over all datasets. The reason is that the storage phase processes the time series data row by row, and the time cost of processing a row is


(a) Query time vs Error in Activity (b) Query time vs Error in Gas (c) Query time vs Error in London

(d) Memory usage vs Error in Activity (e) Memory usage vs Error in Gas (f) Memory usage vs Error in London

**Figure 6: The trade-off between query time, space, and MSE rate on three real-world datasets. The first column shows the results on the Activity dataset. The second and third columns show the results on the Gas and the London datasets. The colors represent the methods, and the shapes distinguish the threshold $\xi$. The bottom-left region indicates the better performance. On all the real-world datasets, Zoom-SVD has the best performance.**

constant as discussed in Theorem 1; thus, the total time cost mainly depends on the time length.

**Query phase (Algorithm 2).** We evaluate the performance of the query phase in terms of running time. We compare Zoom-SVD to other SVD based methods discussed in Section 2, and measure the query time varying the length of the query range. The starting and the ending points of the query are arbitrarily chosen, and we increase the length of the query range from $10^4$ to $3.2 \times 10^5$. As shown in Figure 1, Zoom-SVD is up to 9.6× faster than the second best competitor Randomized SVD on Activity dataset. Also, Zoom-SVD shows up to 15× and 12× faster query speed than Randomized SVD on Gas sensor and London datasets, respectively.

## 4.3 Space Cost

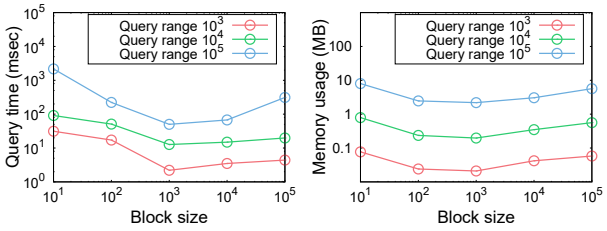
We evaluate the compression performance of Zoom-SVD. In the storage phase, we store the SVD results of multiple time series data with block size $b = 10^3$ using low-rank approximation with threshold $\xi = 0.98$ in Equation (1). We measure the compression ratio for storing the original data and the block compressed data (i.e., the SVD results) by our method. Figure 5 shows the compression performance for storing time series data. Zoom-SVD requires up to 7.88×, 15.63×, and 7.19× less space than the original data require for Activity, Gas, and London, respectively.

## 4.4 Trade-off between Accuracy and Efficiency

We evaluate the trade-off of Zoom-SVD between accuracy, time, and space compared to other methods. We measure the time and the space usage by setting the length of time range query to $1.8 \times 10^5$. The accuracy of each method is measured by the reconstruction error $\frac{\|\mathbf{X}_{(t_s:t_e)} - \hat{\mathbf{X}}_{(t_s:t_e)}\|_F^2}{\|\mathbf{X}_{(t_s:t_e)}\|_F^2}$ where $\hat{\mathbf{X}}_{(t_s:t_e)}$ is the reconstructed data (i.e., $\hat{\mathbf{X}}_{(t_s:t_e)} = \mathbf{U}_{(S,E)}\Sigma_{(S,E)}\mathbf{V}^T_{(S,E)}$), and $\mathbf{X}_{(t_s:t_e)}$ is the original input data in a query time range $[t_s, t_e]$. We use the values 0.95, 0.98, and 0.99 for the threshold $\xi$ in the low-rank approximation of each method to investigate the effect of $\xi$ on the trade-off performance. Figure 6 demonstrates the experimental results on the trade-off of SVD based methods including Zoom-SVD. Figures 6(a), 6(b), and 6(c) present the trade-off of Zoom-SVD between query time and reconstruction error is better than those of other methods over all the datasets. Also, Zoom-SVD shows a better trade-off between space and reconstruction error than its competitors as shown in Figures 6(d), 6(e), and 6(f). These results indicate that Zoom-SVD handles time range queries more efficiently with smaller error and space than other SVD based methods.

## 4.5 Parameter Sensitivity

We examine the effects of the block size $b$ in terms of query time, and space cost. When the block size $b$ grows from 10 to $10^5$, we measure the query time and the memory usage in query phase by setting the query time range $t_e - t_s + 1$ to $10^3$, $10^4$, and $10^5$ in Algorithm 2. In Figure 7, the query time and memory usage show trade-off characteristics. The query time and the space usage for Activity dataset decrease until the block size reaches $10^2 \sim 10^3$, and then increase afterwards when the block size $b$ exceeds $10^3$. Note that Zoom-SVD consists of Partial-SVD and Stitched-SVD modules in the query phase. In Figure 7(a), the query time is dominated by the computation time of the Stitched-SVD when the block size $b$ is relatively small. The reason is that the computation time of Partial-SVD decreases as the block size $b$ decreases, while that of Stitched-SVD increases linearly with the number of blocks which is inversely proportional to the block size. On the other




(a) Query time vs. block size (b) Space cost vs. block size

**Figure 7: The effect of the block size $b$ in terms of the query time and the memory usage in Activity dataset. When the block size $b$ grows from 10 to $10^5$, the query time and the memory usage decrease until the block size is $10^3$, and then increase continuously after block size exceeds $10^3$.**

hand, the query time highly depends on the computation time of Partial-SVD when the block size is relatively large. In Figure 7(b), space cost is high when $b$ is small because the number of stored singular vector matrices for blocks increases. On the other hand, space cost is also high when $b$ is large because the number $k$ of singular values increases as the block size $b$ increases, and then the size of left singular vector matrices $\mathbf{U}_{(i)}$ increases. Therefore, we set the block size $b$ to $10^3$ which is a near-optimum value according to the experimental results. The sensitivities of Zoom-SVD on the block size $b$ in other datasets show similar patterns.

## 5 CASE STUDY

In this section, we present the analysis result of Gas sensor dataset using Zoom-SVD. For a given time range, Zoom-SVD searches past time ranges for similar patterns to that of the given time range.

**Data description.** Gas sensor dataset consists of 16 chemical sensors exposed to gas mixtures which contain ethylene and carbon monoxide (CO) with air. There are four different types of sensors: TGS2600, TGS2602, TGS2610, and TGS2620 (four sensors per one type). The sensors are placed in a 60ml measurement chamber. The electrical conductivities of the sensors are measured at every 0.01 seconds, and the total time length is 12 hours.

**Finding similar time ranges.** Our goal is to find past time ranges whose patterns are similar to that of a given time range query which we call as the 'base'. Suppose a time range $[t_s, t_e]$ is given. First, we compute SVD of data in the range using Zoom-SVD. Next, we continuously compute SVDs of previous time ranges using Zoom-SVD by sliding a window of size $t_e - t_s + 1$. Note that we use the fixed window size for efficiency. We then compare $\mathbf{u_1}$ of the base with $\overline{\mathbf{u}}_\mathbf{1}$ of previous time ranges, where $\mathbf{u_1}$ is the first column of left singular vector matrix $\mathbf{U}$ of the base, and $\overline{\mathbf{u}}_\mathbf{1}$ is the first column of $\overline{\mathbf{U}}$ of a previous time range. Experimental setting is as follows:

- The base time range is [4033294, 4053293] which is randomly chosen, and thus the size of sliding window (time length) is 20,000. The sliding period is 500.
- We compute the cosine similarity $\frac{\mathbf{u_1} \cdot \overline{\mathbf{u}}_\mathbf{1}}{\|\mathbf{u_1}\| \|\overline{\mathbf{u}}_\mathbf{1}\|}$ to compare the patterns, and then find the two most similar time ranges.

Figure 8 shows the first singular vectors of the given time range (denoted by 'Base') and two previous time ranges (denoted by 'Answer1' and 'Answer2') with similar patterns we found, and sensor data corresponding to each time range. In Figure 8(a), Zoom-SVD successfully finds the similar patterns, and each sensor has the same tendency w.r.t. the three time ranges. We also discover a potential anomaly. In Figure 8(c), note an abnormal spike of Base at the time around 17,960, which deviates significantly from the measurements of the two previous time ranges.

## 6 RELATED WORK

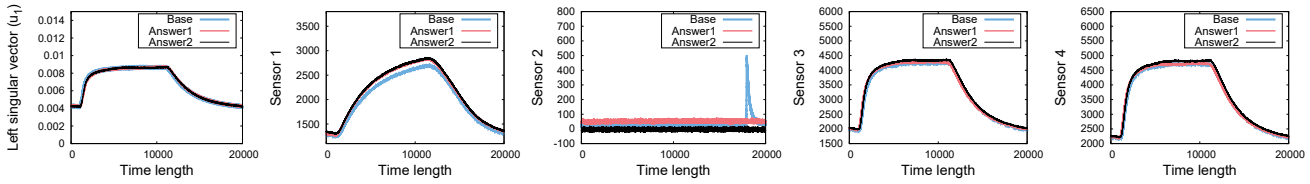
In this section, we discuss related works on incremental time series analysis and incremental SVD.

It is important to analyze on-line time series data efficiently. There are several ideas based on linear modeling, including Kalman Filters (KF), linear dynamical systems (LDS) and the variants [9, 19, 20], time warping [39], and correlation-based methods [17, 21, 34, 43]. Among them, incremental SVD efficiently tracks the SVD of dynamic time series data where new data rows are incrementally attached to the current data. Incremental SVD has been used for updating SVD results when new terms or documents are added to a document-term matrix [3], and has been adopted to build an incremental movie recommender system [2, 35]. Brand [1] developed an incremental SVD method for incomplete data which contain missing values. Papadimitriou et al. [24] proposed SPIRIT which incrementally updates hidden factors in multiple time series, and exploited SPIRIT to predict future signals and interpolate missing values in sensor streams. Ross et al. [32] proposed an incremental PCA for visual tracking applications. Papadimitriou et al. [25] introduced an incremental method to capture optimal recurring patterns indicating the main trends in time series data.

The methods above have limitations in applying to time range query problem. Conventional methods such as Discrete Fourier Transformation [21], data compression [6, 30], and time series clustering [26] require the whole time series data, or additional data to seek hidden patterns for a specific time range, thereby incurring high memory and time complexity. On the other hand, our proposed Zoom-SVD efficiently serves time range queries with small memory and low time complexity.

## 7 CONCLUSIONS

We propose Zoom-SVD, a novel algorithm for finding key patterns in an arbitrary time range from multiple time series data. Zoom-SVD efficiently serves time range queries by compressing multiple time series data, and reducing computation costs by carefully stitching compressed SVD results. Consequently, Zoom-SVD provides fast query time and small memory usage. We provide theoretical analysis on the time and space complexities of Zoom-SVD. Experiments show that Zoom-SVD requires up to 15.66× less space, and runs up to 15× faster than existing methods. Also, we show a real-world case study of Zoom-SVD in searching past time ranges for similar patterns as that of a query time range. Future research includes extending the method for multiple distributed streams.




(a) First left singular vector (b) Sensor 1 (c) Sensor 2 (d) Sensor 3 (e) Sensor 4

**Figure 8: First left singular vectors of the base and two time ranges having the similar patterns to that of the base. Answer1 and Answer2 are the past time ranges we found using Zoom-SVD by sliding a window. The time range of Answer1 is** $[1589294, 1609293]$**, and that of Answer2 is** $[2736794, 2756793]$**. In (a), the three left singular vectors have the same pattern. In (b, c, d, e), each sensor shows similar measurements for the three ranges in general. Note that Zoom-SVD discovers a potential anomaly: (c) shows an abnormal spike of Base around** $17,960$**, which deviates vastly from the measurements of the past ranges.**

## ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2016M3C4A7952587, PF Class Heterogeneous High Performance Computer Development). The Institute of Engineering Research at Seoul National University provided research facilities for this work. The ICT at Seoul National University provides research facilities for this study. U Kang is the corresponding author.

## REFERENCES

[1] Matthew Brand. 2002. Incremental singular value decomposition of uncertain data with missing values. *ECCV* (2002), 707–720.
[2] Matthew Brand. 2003. Fast online svd revisions for lightweight recommender systems. In *SDM*. 37–46.
[3] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391.
[4] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. (2017). http://archive.ics.uci.edu/ml
[5] Jordi Fonollosa, Sadique Sheik, Ramón Huerta, and Santiago Marco. 2015. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical* 215 (2015), 618–629.
[6] Sorabh Gandhi, Suman Nath, Subhash Suri, and Jie Liu. 2009. Gamps: Compressing multi sensor data by grouping and amplitude scaling. In *SIGMOD*. ACM, 771–784.
[7] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.
[8] M. A. Iwen and B. W. Ong. 2016. A Distributed and Incremental SVD Algorithm for Agglomerative Data Analysis on Large Networks. *SIAM J. Matrix Analysis Applications* 37, 4 (2016), 1699–1718.
[9] Ankur Jain, Edward Y Chang, and Yuan-Fang Wang. 2004. Adaptive stream resource management using kalman filters. In *SIGMOD*. ACM, 11–22.
[10] ByungSoo Jeon, Inah Jeon, Lee Sael, and U. Kang. 2016. SCouT: Scalable coupled matrix-tensor factorization - algorithm and discoveries. In *ICDE*. 811–822.
[11] Inah Jeon, Evangelos E. Papalexakis, Christos Faloutsos, Lee Sael, and U. Kang. 2016. Mining billion-scale tensors: algorithms and discoveries. *VLDB J.* 25, 4 (2016), 519–544.
[12] Inah Jeon, Evangelos E. Papalexakis, U. Kang, and Christos Faloutsos. 2015. HaTen2: Billion-scale Tensor Decompositions. In *ICDE*. 1047–1058.
[13] Ian Jolliffe. 2002. *Principal component analysis*. Wiley Online Library.
[14] U. Kang, Brendan Meeder, and Christos Faloutsos. 2011. Spectral Analysis for Billion-Scale Graphs: Discoveries and Implementation. In *PAKDD*. 13–25.
[15] U Kang, B. Meeder, E. Papalexakis, and C. Faloutsos. 2014. HEigen: Spectral Analysis for Billion-Scale Graphs. *Knowledge and Data Engineering, IEEE Transactions on* 26, 2 (February 2014), 350–362. https://doi.org/10.1109/TKDE.2012.244
[16] U. Kang, Hanghang Tong, and Jimeng Sun. 2012. Fast Random Walk Graph Kernel. In *SDM*. 828–838.
[17] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. 2001. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *SIGMOD*. 151–162.
[18] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[19] Lei Li, James McCann, Nancy S Pollard, and Christos Faloutsos. 2009. Dynammo: Mining and summarization of coevolving sequences with missing values. In *KDD*. ACM, 507–516.
[20] Lei Li, B Aditya Prakash, and Christos Faloutsos. 2010. Parsimonious linear fingerprinting for time series. *PVLDB* 3, 1-2 (2010), 385–396.
[21] Abdullah Mueen, Suman Nath, and Jie Liu. 2010. Fast approximate correlation for massive time-series data. In *SIGMOD*. ACM, 171–182.
[22] Sejoon Oh, Namyong Park, Lee Sael, and U. Kang. 2018. Scalable Tucker Factorization for Sparse Tensors - Algorithms and Discoveries. In *ICDE*. 1120–1131.
[23] Stanisław Osiński, Jerzy Stefanowski, and Dawid Weiss. 2004. Lingo: Search results clustering algorithm based on singular value decomposition. In *Intelligent information processing and web mining*. Springer, 359–368.
[24] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. 2005. Streaming Pattern Discovery in Multiple Time-Series. In *VLDB*. 697–708.
[25] Spiros Papadimitriou and Philip S. Yu. 2006. Optimal multi-scale patterns in time series streams. In *SIGMOD*. 647–658.
[26] John Paparrizos and Luis Gravano. 2015. k-shape: Efficient and accurate clustering of time series. In *SIGMOD*. ACM, 1855–1870.
[27] Haekyu Park, Jinhong Jung, and U. Kang. 2017. A comparative study of matrix factorization and random walk with restart in recommender systems. In *BigData*. 756–765.
[28] Namyong Park, ByungSoo Jeon, Jungwoo Lee, and U. Kang. 2016. BIGtensor: Mining Billion-Scale Tensor Made Easy. In *CIKM*. 2457–2460.
[29] KV Ravi Kanth, Divyakant Agrawal, and Ambuj Singh. 1998. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD*, Vol. 27. ACM, 166–176.
[30] Galen Reeves, Jie Liu, Suman Nath, and Feng Zhao. 2009. Managing massive time series streams with multi-scale compressed trickles. *PVLDB* 2, 1 (2009), 97–108.
[31] Attila Reiss and Didier Stricker. 2012. Introducing a New Benchmarked Dataset for Activity Monitoring. In *ISWC*. 108–109.
[32] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. 2008. Incremental learning for robust visual tracking. *International journal of computer vision* 77, 1 (2008), 125–141.
[33] Lee Sael, Inah Jeon, and U Kang. 2015. Scalable Tensor Mining. *Big Data Research* 2, 2 (2015), 82 – 86. https://doi.org/10.1016/j.bdr.2015.01.004 Visions on Big Data.
[34] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. 2005. Braid: Stream mining through group lag correlations. In *SIGMOD*. ACM, 599–610.
[35] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2002. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *ICIS*. 27–28.
[36] Krzysztof Simek, Krzysztof Fujarewicz, Andrzej Świerniak, Marek Kimmel, Barbara Jarzab, Małgorzata Wiench, and Joanna Rzeszowska. 2004. Using SVD and SVM methods for selection, classification, clustering and modeling of DNA microarray data. *Engineering Applications of Artificial Intelligence* 17, 4 (2004), 417–427.
[37] Stephan Spiegel, Julia Gaebler, Andreas Lommatzsch, Ernesto De Luca, and Sahin Albayrak. 2011. Pattern recognition and classification for multivariate time series. In *Sensor-KDD*. ACM, 34–42.
[38] Hanghang Tong, Christos Faloutsos, and Jia-yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *ICDM*. 613–622.
[39] Machiko Toyoda, Yasushi Sakurai, and Yoshiharu Ishikawa. 2013. Pattern discovery in data streams under the time warping distance. *VLDBJ* 22, 3 (2013), 295–318.
[40] Lloyd N Trefethen and David Bau III. 1997. *Numerical linear algebra*. Vol. 50. Siam.
[41] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. 2003. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*. Springer, 91–109.
[42] Reza Bosagh Zadeh, Xiangrui Meng, Alexander Ulanov, Burak Yavuz, Li Pu, Shivaram Venkataraman, Evan R. Sparks, Aaron Staple, and Matei Zaharia. 2016. Matrix Computations and Optimization in Apache Spark. In *KDD*. 31–38.
[43] Yunyue Zhu and Dennis Shasha. 2002. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*. VLDB Endowment, 358–369.