Article

Patterns

Tracking online low-rank approximations of higher-order incomplete streaming tensors

Highlights

- Online low-rank approximations are modeled under Tucker and CP formats
- Two tensor trackers were proposed with provable convergence guarantee
- Randomized techniques were exploited to facilitate the tracking process
- The proposed algorithms were shown to be among the fastest tensor decompositions

Authors

Le Trung Thanh, Karim Abed-Meraim, Nguyen Linh Trung, Adel Hafiane

Correspondence

linhtrung@vnu.edu.vn

In brief

This article introduces two provable adaptive algorithms for tracking online low-rank approximations of streaming tensors under the Tucker and CP models. Both algorithms converge quickly, have low memory storage, and perform well on synthetic and real data.

Thanh et al., 2023, Patterns *4*, 100759
June 9, 2023
https://doi.org/10.1016/j.patter.2023.100759

CellPress

Patterns

CellPress
OPEN ACCESS

Article

Tracking online low-rank approximations of higher-order incomplete streaming tensors

Le Trung Thanh,[1,2] Karim Abed-Meraim,[1,3] Nguyen Linh Trung,[2,4,*] and Adel Hafiane[1]
[1]PRISME Laboratory, University of Orléans, INSA CVL, 12 Rue de Blois, 45100 Orléans, France
[2]AVITECH Institute, VNU University of Engineering and Technology, 144 Xuan Thuy, Cau Giay 10000, Hanoi, Vietnam
[3]Academic Institute of France, 1 Rue Descartes, 75005 Paris, France
[4]Lead contact
*Correspondence: linhtrung@vnu.edu.vn
https://doi.org/10.1016/j.patter.2023.100759

THE BIGGER PICTURE Low-rank approximation methods are a class of mathematical techniques commonly used to help process large datasets, especially in signal processing and machine learning applications. These methods allow multidimensional data to be represented by a set of low-dimensional components and can be powerful tools for discovering valuable information or deriving new insights from complex data. Applying these methods to streaming data, which are being continuously generated and must be analyzed in real time, remains challenging due to the increasing size and complexity of these datasets over time. These challenges are particularly acute for datasets with missing values. This paper proposes a novel adaptive method for tracking online low-rank approximations of multidimensional streaming data, resulting in two efficient tensor trackers that accurately estimate the underlying components of noisy, incomplete, and high-dimensional observations. The effectiveness of the proposed approach is demonstrated in various experiments, including the analysis of EEG data and video sequences.

Development/pre-production

SUMMARY

In this paper, we propose two new provable algorithms for tracking online low-rank approximations of high-order streaming tensors with missing data. The first algorithm, dubbed adaptive Tucker decomposition (ATD), minimizes a weighted recursive least-squares cost function to obtain the tensor factors and the core tensor in an efficient way, thanks to an alternating minimization framework and a randomized sketching technique. Under the canonical polyadic (CP) model, the second algorithm, called ACP, is developed as a variant of ATD when the core tensor is imposed to be identity. Both algorithms are low-complexity tensor trackers that have fast convergence and low memory storage requirements. A unified convergence analysis is presented for ATD and ACP to justify their performance. Experiments indicate that the two proposed algorithms are capable of streaming tensor decomposition with competitive performance with respect to estimation accuracy and runtime on both synthetic and real data.

INTRODUCTION

The era of "big data," which deals with massive datasets, has brought new analysis techniques for discovering new valuable information hidden in data.[1] Among these techniques is multilinear low-rank approximation (LRA) of matrices and tensors, which has recently attracted considerable attention from engineers and researchers in the signal processing and machine learning communities.[2] A tensor is a multidimensional array and provides a natural representation of high-dimensional data. Low-rank approximation of tensors (t-LRA) can be considered as a multiway extension of LRA of matrices (which are two-way) to higher dimensions.[3] Generally, t-LRA is referred to as tensor decomposition that factorizes a tensor into a sequence of basic components.[3] As a result, t-LRA provides a useful tool for dealing with several large-scale multidimensional problems in modern data analysis that would otherwise be intractable by classical methods.

Two widely used approaches for t-LRA are Tucker decomposition[4] and canonical polyadic (CP) decomposition[5] (there exist some other names for the CP decomposition in the literature, such as PARAFAC [parallel factors], CPD [canonical polyadic decomposition], and CANDECOMP or CAND [canonical decomposition]). Under the CP format, a tensor can be represented as a sum of rank 1 tensors; each rank 1 tensor is formulated as the outer product of vectors. Under the Tucker format, a tensor is factorized into a sequence of factor matrices acting on a reduced-size core tensor. "Workhorse" algorithms are based on the method of alternating least squares (ALS). Readers are referred to the work of Kolda and Bader[3] for a good review.

The characteristics of big data are often associated with the following three "Vs": volume, velocity, and veracity.[1] Velocity and veracity are the focus of this paper. Velocity requires (near) real-time processing of data streams, while veracity demands robust algorithms to better deal with missing, noisy, and inconsistent data. In online applications, data acquisition is often a time-varying process in which data are serially collected or changing with time. In addition, missing data are ubiquitous and more and more common in high-dimensional problems in which collecting all attributes of the data is too expensive or even impossible. However, well-known t-LRA algorithms either face high complexity or operate in batch mode and, thus, may not be suitable for such problems. This has led to defining a variant of t-LRA, namely online (adaptive) t-LRA.

In the literature, there are several studies related to the problem of tracking online t-LRA in the missing data context; the tensors are said to be both *streaming* and *incomplete*. Under the CP format, the very first adaptive tensor algorithms were proposed by Nion and Sidiropoulos[6] more than 10 years ago. Since then, several adaptive CP decomposition methods have been introduced. We refer the reader to Thanh et al.[7] for a good survey. Of these methods, Mardani et al. proposed TeCPSGD,[8] which is a first-order algorithm and uses the method of stochastic gradient descent (SGD) to track the CP decomposition of third-order streaming tensors with missing data. Leveraging the framework of alternating minimization, TeCPSGD can estimate directly all factors except the one corresponding to the dimension growing over time in an efficient way. Because of SGD, TeCPSGD is a low-complexity tensor tracker but with a slow convergence rate. Therefore, it is not really suitable for fast time-varying scenarios in which a class of methods with a fast rate of convergence is preferable. Kasai[9] developed OLSTEC, which is an efficient second-order algorithm and exploits the recursive least-squares technique. OLSTEC provides competitive performance in terms of estimation accuracy, but its computational complexity is much higher than that of TeCPSGD. In parallel, Minh-Chinh et al. proposed to first track the low-dimensional tensor subspace and then derive the loading factors from its Khatri-Rao structure.[10] Its performance, however, is sensitive to initialization (see Figure 6 for an illustration).

All the adaptive CP decomposition algorithms above are specifically designed for factorizing third-order streaming tensors (i.e., the temporal tensor slices, a.k.a. data observations, are matrices), which could limit their applications in practice where the underlying tensor is of higher order (i.e., greater than or equal to 4). One possible way is to reshape the underlying higher-order streaming tensor into a third-order one and then apply the above-mentioned algorithms for tracking. However, the high-dimensional structure of the original tensor might not be fully preserved in its reshaped variant, resulting in low estimation accuracy. Dealing with Nth-order streaming tensors ($N>3$) is non-trivial due to several issues. Some mathematical tools, transformations, and operations applied for third-order streaming tensors are not straightforward for higher-order ones, such as the low-rank regularization, the tensor subspace/dictionary (used in Mardani et al.[8] and Kasai[9]), and the bi-iteration SVD procedure (used in Nion[6] and Sidiropoulos and Minh-Chinh et al.[10]). Particularly in Mardani et al.[8] and Kasai,[9] the nuclear norm of a matrix $\mathbf{Z}$ can be derived from $\min_{\mathbf{Z}=\mathbf{U}\mathbf{V}^\top}\frac{1}{2}(\|\mathbf{U}\|_F^2+\|\mathbf{V}\|_F^2)$, thanks to Lemma 5.1 in Recht et al.[11] This property is widely used by several matrix factorization methods for low-rank regularization.[12] Accordingly, the sum of squared Frobenius norms of two non-temporal factors of third-order streaming tensors can be used as a regularization promoting the LRA of data streams. However, this property does not generally hold for higher-order tensors due to the presence of more than two factors and their multilinear connection. In Nion[6] and Sidiropoulos and Minh-Chinh et al.,[10] the subspace-based algorithms track the underlying low-dimensional tensor subspace matrix $\mathbf{H}_t = \mathbf{U}_t^{(1)}\odot\mathbf{U}_t^{(2)}\odot\cdots\odot\mathbf{U}_t^{(N-1)}$, where N is the tensor order, and then estimate the tensor factors $\{\mathbf{U}_t^{(n)}\}_{n=1}^{N-1}$ by exploiting its Khatri-Rao structure. When $N=3$, subspace tracking algorithms and bi-iteration SVD are specifically applied to estimate $\mathbf{H}_t$ and $\{\mathbf{U}_t^{(1)},\mathbf{U}_t^{(2)}\}$. However, when $N>3$, it becomes more complicated due to two main issues: (1) tracking the matrix $\mathbf{H}_t$, having a "hierarchical" Khatri-Rao structure, over time is non-trivial, especially in noisy and time-varying environments, and (2) even when $\mathbf{H}_t$ is assumed to be estimated correctly at each time t, the estimation of $\{\mathbf{U}_t^{(n)}\}_{n=1}^{N-1}$ might cost a high computational complexity; e.g., if bi-iteration SVD is used, we have to repeat bi-iteration SVD recursively $N-1$ times, which is very expensive for streaming processing. These characteristics prevent us from extending their methods for tracking higher-order streaming tensors efficiently and effectively. Accordingly, designing adaptive methods capable of directly tracking tensors of higher order is of great importance, and it is our main concern in this study. Some adaptive methods have been developed for handling higher-order streaming tensors in the literature. For example, Ahn et al.[13] introduced another online CP algorithm called STF, which is capable of dealing with higher-order streaming tensors. The authors imposed a temporal regularization on the loading factors and used the SGD method to update them over time. Zhang and Hawkins[14] developed a Bayesian-based streaming method called BRST robust to outliers. To track and separate the low-rank and sparsity components of the underlying tensor, a Bayesian statistical model was applied. The computational complexity of BRST is, however, very high, and thus, the method becomes inefficient when handling high-dimensional and fast-arriving data streams. Lee and Shin[15] proposed another robust streaming CP algorithm called SOFIA, which has the potential to handle real-world data streams with missing values and sparse outliers. Specifically, SOFIA exploits a well-known time-series forecasting model, namely, Holt-Winters, for detecting

outliers and temporal patterns and hence factorizing the underlying tensor. In our past work,[16] we developed a robust adaptive CP decomposition (RACP) with missing data and outliers. Thanks to the recursive least-squares technique in adaptive filtering and the alternating direction method of multipliers (ADMM) method, RACP is capable of detecting sparse outliers online and tracking successfully the underlying CP model of data streams over time. In parallel, some adaptive CP decomposition algorithms, such as Zhou et al.,[17] Smith et al.,[18] Thanh et al.,[19] Zeng and Ng,[20] and Lyu et al.,[21] are capable of handling higher-order tensors. However, they do not handle incomplete datasets.

Under the Tucker format, there are many adaptive methods capable of factorizing streaming tensors in online settings.[7] Particularly, several Tucker trackers were proposed to decompose streaming tensors having one mode/dimension evolving with time. Some of them work under the assumption that temporal slices of the streaming tensor interact with the same core tensor of fixed size, for example, RPTucker,[22] BASS-Tucker,[23] RT-NTD,[24] BK-NTD,[24] and D-TuckerO.[25] Some others, on the other hand, assume that the core tensor has one temporal mode and that its temporal slices associate with data streams, for example, STA,[26] OTL,[27] ORLTM,[28] D-L1-Tucker,[29] and ROLTD,[30] to name a few. Among them, a few Tucker trackers can deal with data corruption. RPTucker[22] is specifically designed for dynamic tensor completion, but its ability is limited to third-order tensors. ORLTM,[28] D-L1-Tucker,[29] and ROLTD[30] are robust to sparse outliers. However, their design is not suitable for handling incomplete observations.

Some studies have been conducted to design efficient t-SVD algorithms for higher-order tensors, for example.[31–34] Most of them were designed for batch computation and, thus, are not suitable for dynamic models. Only TOUCAN[31] has the ability to track t-SVD over time. However, it is useful only for third-order streaming tensors. In parallel, it is well known that block-term decomposition (BTD) can be considered as a combination of CP and Tucker decompositions.[35] In the tensor literature, there are two adaptive BTD algorithms that are able to factorize streaming tensors, namely OnlineBTD[36] and O-BTD-RLS.[37] They are, however, sensitive to data corruption. With respect to tensor-train decomposition, we proposed TT-FOA,[38] which is an adaptive tensor-train (TT) model for streaming tensors. Although TT-FOA and its stochastic version are capable of tracking the online low-rank TT representation of large-scale and higher-order tensors, they are not designed to handle missing data. ROBOT was recently proposed by Thanh et al.[39] to overcome this drawback. Very recently, Yu et al.[40] proposed, for the first time, an online tensor completion based on a tensor-ring format. Its convergence, however, has not yet been mathematically proven.

In the multiaspect streaming perspective of tensor analysis, Song et al. proposed an effective multiaspect streaming tensor framework (MAST),[41] used for dynamic tensor completion. MAST can successfully track the multilinear LRA of incomplete tensors with dynamic growth in more than one tensor mode. A robust version of MAST for handling outliers, called outlier-robust multiaspect streaming tensor completion and factorization (OR-MSTC), was proposed in Najafi et al.[42] Thanks to ADMM, OR-MSTC can estimate the low-rank component from measurements corrupted by outliers. A new inductive framework, called SIITA, has been proposed to incorporate side information into incremental tensor analysis.[43] SIITA can be seen as a counterpart of MAST for multiaspect streaming Tucker decomposition. Although all these approaches provide good frameworks for the problem of dynamic tensor completion, they either are either useful for third-order tensors only or are of high complexity and, hence, relatively inefficient in online applications with data streams. In addition, convergence analysis of these algorithms is not available.

This study considers the problem of tracking t-LRA of higher-order incomplete tensors using randomized sketching techniques. It is mainly motivated by the fact that randomized algorithms reduce the computational complexity and memory storage of their conventional counterparts.[44] As a result, they have recently attracted a great deal of attention and achieved success in large-scale data analysis, in general, and in tensor decomposition in particular. For example, Wang et al. applied a sketching technique to develop a fast algorithm for orthogonal tensor decomposition.[45] Under mild conditions, the tensor sketch can be obtained without accessing all the data.[46] Battaglino et al. proposed a practical randomized CP decomposition,.[47] Their work aimed to speed up the traditional ALS algorithm via randomized least-squares regressions. With respect to Tucker decomposition, Malik and Becker proposed two randomized algorithms using TensorSketch for low-rank tensor decomposition.[48] Che and Wei designed an effective randomized algorithm for computing the LRA of tensors under the sequentially truncated HOSVD (ST-HOSVD) model.[49] Recently, they provided an improved version of ST-HOSVD with a lower computational complexity and analyzed its probabilistic error bound.[50] In parallel, two other randomized versions of HOSVD and ST-HOSVD were introduced by Minster et al.[51] However, these algorithms perform only batch computation, so they are not appropriate for online processing. We refer the reader to Ahmadi-Asl et al.[52] for a good survey on randomized algorithms for tensor decomposition. This shortcoming motivates us to develop a new efficient randomized algorithm for the problem of tracking t-LRA.

The main contributions of this paper are 2-fold. First, under the Tucker format, we propose a novel adaptive Tucker decomposition (ATD) algorithm for tracking the online t-LRA of higher-order incomplete streaming tensors. ATD is a low-complexity tensor tracker, and its convergence is fast, thanks to alternating minimization and randomized sketching. It can handle incomplete tensors derived from infinite data streams because it performs Tucker decomposition with constant time and space complexity that is independent of time index t. A convergence analysis is then provided to establish performance guarantees. Second, under the CP format, we derive a second algorithm, namely adaptive CP decomposition (ACP), for the problem of online t-LRA. ACP is faster than ATD, as the cost of both computation and memory storage is lower. ACP exhibits a competitive performance in terms of estimation accuracy and running time. To the best of our knowledge, ATD and ACP are the first of their kind capable of dealing with streaming tensors of higher orders with a "provable" convergence guarantee.

The rest of this paper is structured as follows. The background presents a brief review of tensor operators and the t-LRA

Table 1. Notational conventions

Notation	Conventions
$x, \boldsymbol{x}, \boldsymbol{X}, \boldsymbol{\mathcal{X}}, \mathbb{X}$	scalar, vector, matrix, tensor, and set/subset/support
$x_{i_1,i_2,\ldots,i_N}$	$(i_1,i_2,\ldots,i_N)$-th entry of $\boldsymbol{\mathcal{X}}$
$\boldsymbol{x} = \text{vec}(\boldsymbol{X})$	vectorization of $\boldsymbol{X}$
$\boldsymbol{X} = \text{diag}(\boldsymbol{x})$	diagonal matrix $\boldsymbol{X}$ with $\boldsymbol{x}$ on the main diagonal
$\text{tr}(\boldsymbol{X})$	trace of $\boldsymbol{X}$
$\boldsymbol{X}(i,:), \boldsymbol{X}(:,j)$	i-th row and j-th column of $\boldsymbol{X}$
$\boldsymbol{X}^\top, \boldsymbol{X}^{-1}, \boldsymbol{X}^\#$	transpose, inverse, and pseudo-inverse of $\boldsymbol{X}$
$\underline{\boldsymbol{X}}^{(n)}$	mode-n unfolding of $\boldsymbol{\mathcal{X}}$
$\circ, \odot, \otimes, \circledast$	outer, Khatri-Rao, Kronecker, Hadamard product
$\boldsymbol{\mathcal{X}} \boxplus \boldsymbol{\mathcal{Y}}$	concatenation of $\boldsymbol{\mathcal{X}}$ with $\boldsymbol{\mathcal{Y}}$
$\boldsymbol{\mathcal{X}} \times_n \boldsymbol{U}$	n-mode product of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{U}$
$\boldsymbol{\mathcal{X}} \prod_{n=1}^{N} \times_n \boldsymbol{U}^{(n)}$	$\boldsymbol{\mathcal{X}} \times_1 \boldsymbol{U}^{(1)} \times_2 \ldots \times_N \boldsymbol{U}^{(n)}$
$\bigodot_{n=1}^{N} \boldsymbol{U}^{(n)}$	$\boldsymbol{U}^{(N)} \odot \boldsymbol{U}^{(N-1)} \odot \ldots \odot \boldsymbol{U}^{(1)}$
$\bigotimes_{n=1}^{N} \boldsymbol{U}^{(n)}$	$\boldsymbol{U}^{(N)} \otimes \boldsymbol{U}^{(N-1)} \otimes \ldots \otimes \boldsymbol{U}^{(1)}$
$\|.\|_F$	Frobenius norm
$\lfloor \rceil$	operator for finding the nearest integer
randomsample(n,k)	operator for selecting k integers randomly from [1, n]
r_{CP}	CP rank
$\boldsymbol{r}_{TD}$	Tucker rank

problem. The problem statement formulates the problem of tracking t-LRA for incomplete and streaming tensors. Proposed methods describes in detail the proposed method for tracking t-LRA and its convergence analysis. The section on experimental procedures describes extensive experiments to demonstrate the effectiveness and efficiency of our algorithms in comparison with state-of-the-art algorithms. The last section is the conclusion.

BACKGROUND

Notations and definitions

In this paper, we use the following notational conventions. Scalars and vectors are denoted by lowercase letters (e.g., x) and boldface lowercase letters (e.g., $\mathbf{x}$), respectively. Boldface capital and bold calligraphic letters denote matrices (e.g., $\mathbf{X}$) and tensors (e.g., $\boldsymbol{\mathcal{X}}$). For index notation, the $(i_1,i_2,\ldots,i_N)$-th entry of $\boldsymbol{\mathcal{X}}$ is indicated by $x_{i_1,i_2,\ldots,i_N}$. The symbols $\circ$, $\odot$, $\otimes$, and $\circledast$ are used to denote the outer, Khatri-Rao, Kronecker, and Hadamard products, respectively. The symbol denotes the operator for rounding to the nearest integer. We use $\mathbf{X}^\top$, $\mathbf{X}^{-1}$, and $\mathbf{X}^\#$ to represent the transpose, inverse, and pseudo-inverse of $\mathbf{X}$. Also, $\|.\|_F$ denotes the Frobenius norm of a vector, matrix, and tensor. The operator randomsample(n,k) returns k integers sampled uniformly at random from the range $[1,n]$. In addition, we outline here some algebraic operators on matrices and tensors that are frequently used throughout this paper.

Considering an N-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, the mode-$n$ fibers of $\boldsymbol{\mathcal{X}}$ are I_n-dimensional vectors derived from fixing all but the i_n-th index. The mode-n unfolding of $\boldsymbol{\mathcal{X}}$, written as $\underline{\mathbf{X}}^{(n)}$, is a matrix whose columns are the mode-n fibers of $\boldsymbol{\mathcal{X}}$. We also use $\text{unfold}_n(\boldsymbol{\mathcal{X}})$ to denote the unfolding operation along the n-th mode.

The n-mode product of $\boldsymbol{\mathcal{X}}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$, written as $\boldsymbol{\mathcal{X}} \times_n \mathbf{U}$, yields a new tensor, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_{(n-1)} \times J \times I_{(n+1)} \times \cdots \times I_N}$, such that $\underline{\mathbf{Y}}^{(n)} = \mathbf{U}\underline{\mathbf{X}}^{(n)}$. The product of $\boldsymbol{\mathcal{X}}$ with N matrices $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$ along all N modes is denoted by:

$$\boldsymbol{\mathcal{X}} \prod_{n=1}^{N} \times_n \mathbf{U}^{(n)} = \boldsymbol{\mathcal{X}} \times_1 \mathbf{U}^{(1)} \times_2 \cdots \times_N \mathbf{U}^{(N)}. \quad \text{(Equation 1)}$$

The concatenation of $\boldsymbol{\mathcal{X}}$ with a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N-1}}$, written as $\boldsymbol{\mathcal{X}} \boxplus \boldsymbol{\mathcal{Y}}$, yields a new tensor, $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N + 1}$, such that:

$$z_{i_1,i_2,\ldots,i_N} = \begin{cases} x_{i_1,i_2,\ldots,i_N}, & \text{if } i_N \leq I_N, \\ y_{i_1,i_2,\ldots,i_{N-1}}, & \text{if } i_N = I_N + 1. \end{cases} \quad \text{(Equation 2)}$$

The Khatri-Rao and Kronecker products of a sequence of matrices in a reverse order are denoted by:

$$\bigodot_{n=1}^{N} \mathbf{U}^{(n)} = \mathbf{U}^{(N)} \odot \mathbf{U}^{(N-1)} \odot \cdots \odot \mathbf{U}^{(1)}, \quad \text{(Equation 3)}$$

$$\bigotimes_{n=1}^{N} \mathbf{U}^{(n)} = \mathbf{U}^{(N)} \otimes \mathbf{U}^{(N-1)} \otimes \cdots \otimes \mathbf{U}^{(1)}. \quad \text{(Equation 4)}$$

For clarity, the frequently used acronyms and notational conventions are summarized in Table 1.

Low-rank approximations of tensors

Consider an N-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$; the t-LRA of $\boldsymbol{\mathcal{X}}$ can be achieved by solving the following minimization:

$$\underset{\mathcal{Y}}{\text{argmin}} \; \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}\|_F^2 \;\text{subject to}\; \boldsymbol{\mathcal{Y}} = \mathcal{G} \prod_{n=1}^{N} \times_n \mathbf{U}^{(n)}, \quad \text{(Equation 5)}$$

where $\mathbf{r} = [r_1, r_2, \ldots, r_N]$ is the desired low multilinear rank, $\mathcal{G}$ is the core tensor of size $r_1 \times r_2 \times \cdots \times r_N$, and $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$ with $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times r_n}$ are called loading factors.[53] The two most well-known and widely used approaches for the t-LRA are based on CP decomposition[5] and Tucker decomposition.[4]

Tucker decomposition can be considered as a generalization of SVD for tensors, where the loading factors are orthogonal. Generally, this decomposition is not unique, in the sense that we can rotate the columns of $\mathbf{U}^{(n)}$ by an orthogonal matrix $\mathbf{Q}^{(n)} \in \mathbb{R}^{r_n \times r_n}$ while still retaining the Tucker representation. Fortunately, the column space covering the factor $\mathbf{U}^{(n)}$ is unique; thus, we can estimate subspaces of the loading factors instead.[2,3,54]

CP decomposition allows us to represent the tensor $\boldsymbol{\mathcal{X}}$ by a sequence of factors having the same number of columns, i.e.:

$$\boldsymbol{\mathcal{X}} = \sum_{i=1}^{r} \alpha_i \mathbf{u}_i^{(1)} \circ \mathbf{u}_i^{(2)} \circ \cdots \circ \mathbf{u}_i^{(N)}, \quad \text{(Equation 6)}$$

where r is the tensor rank and $\mathbf{u}_i^{(n)}$ is the i-th column of the n-th factor $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times r}$. This decomposition is more complex than Tucker decomposition in terms of representation and computation, but essentially unique under mild conditions.[2,3] Note that parameters $\{\alpha_i\}_{i=1}^{r}$ can be absorbed in the loading factors,

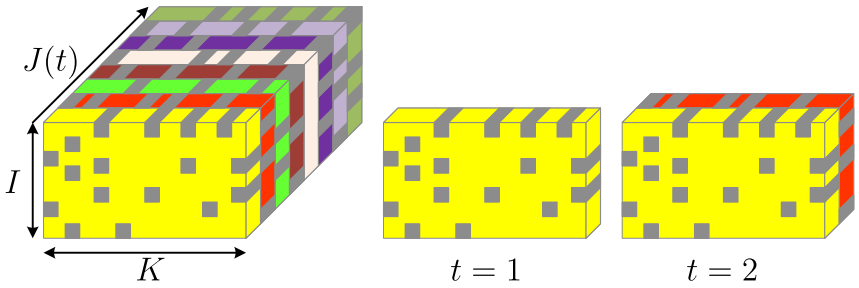

Figure 1. Incomplete streaming tensor
The gray boxes represent missing data. At each time t, the underlying streaming tensor is obtained by appending the new data (i.e., temporal tensor slice) to the old observations along the time dimension. Particularly, the dimension $J(t)$ is increasing with time, while the two dimensions I and K are fixed.

and hence, the main interest of the CP decomposition is in finding $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$.

PROBLEM STATEMENT

In this study, we investigate the problem of tracking t-LRA of an incomplete streaming tensor $\mathcal{X}[t] \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{(N+1)}[t]}$, fixing all but the last dimension, $I_{(N+1)}[t]$ (see illustration in Figure 1 where the gray boxes represent missing data). Specifically, the t-th tensor slice $\mathcal{X}_t \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ of $\mathcal{X}[t]$ is assumed to be generated under the following model:

$$\mathcal{P}_t \circledast \mathcal{X}_t = \mathcal{P}_t \circledast (\mathcal{Y}_t + \mathcal{N}_t), \qquad \text{(Equation 7)}$$

where $\mathcal{P}_t$ is a binary observation mask, $\mathcal{N}_t$ is a Gaussian noise tensor of the same size as $\mathcal{X}_t$, and $\mathcal{Y}_t$ is the multilinear low-rank component. The mask $\mathcal{P}_t$ shows whether the $(i_1, i_2, \ldots, i_N)$-th entry of $\mathcal{X}_t$ is missing, i.e., $p_{i_1,i_2,\ldots,i_N} = 1$ if $x_{i_1,i_2,\ldots,i_N}$ is observed and $p_{i_1,i_2,\ldots,i_N} = 0$ otherwise. The low-rank component $\mathcal{Y}_t$ is given by:

$$\mathcal{Y}_t = \left(\mathcal{G} \prod_{n=1}^{N} \times_n \mathbf{U}^{(n)}\right) \times_{N+1} \mathbf{u}_t^{\top}, \qquad \text{(Equation 8)}$$

where $\mathbf{r} = [r_1, r_2, \ldots, r_{(N+1)}]$ is the desired low multilinear rank, $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_{(N+1)}}$ is the core tensor, $\mathcal{U} = \{\mathbf{U}^{(n)}\}_{n=1}^{N}$ with $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times r_n}$ contains the first N loading factors, and $\mathbf{u}_t \in \mathbb{R}^{r_{(N+1)}}$ is the weight vector. In online settings, the tensor core $\mathcal{G}$ and loading factors $\{\mathbf{U}^{(n)}\}$ may be slowly time-varying, i.e., $\mathcal{G} = \mathcal{G}_t$ and $\mathbf{U}^{(n)} = \mathbf{U}_t^{(n)}, n = 1, 2, \ldots, N$. The weight vector $\mathbf{u}_t$ in Equation 8 is indeed the t-th row of the last loading factor $\mathbf{U}^{(N+1)} \in \mathbb{R}^{I_{(N+1)}[t] \times r_{(N+1)}}$ of $\mathcal{X}[t]$. The tensor $\mathcal{X}[t]$ is derived by appending the new slice $\mathcal{X}_t$ to the previous $\mathcal{X}[t-1]$ along the time dimension $\mathcal{X}[t] = \mathcal{X}[t-1] \boxplus \mathcal{X}_t$, where $I_{(N+1)}[t] = I_{(N+1)}[t-1] + 1$, as shown in Figure 1.

The problem of tracking t-LRA of the incomplete streaming tensor $\mathcal{X}[t]$ can be stated as follows:

Tracking t-LRA. At each time t, we observe a streaming tensor slice $\mathcal{X}_t$ under the model of Equation 7. We aim to estimate $\mathcal{G}_t$ and $\mathcal{U}_t$, which will provide a good multilinear LRA for $\mathcal{X}[t]$ in time.

Applying batch methods to $\mathcal{X}[t]$ is possible, but these turn out to be inefficient for online (adaptive) settings. Our goal is to develop efficient one-pass algorithms, in both computational complexity and memory storage, for tracking the online t-LRA of $\mathcal{X}[t]$ from past estimations at each time t. In particular, in an adaptive scheme, we propose to minimize the following exponentially weighted cost function:

$$\{\mathcal{G}_t, \mathcal{U}_t\} = \underset{\mathcal{G},\mathcal{U}}{\operatorname{argmin}} \left[f_t(\mathcal{G},\mathcal{U}) = \frac{1}{t} \sum_{k=1}^{t} \lambda^{t-k} l(\mathcal{G},\mathcal{U},\mathcal{P}_k,\mathcal{X}_k) \right], \qquad \text{(Equation 9)}$$

where the loss function $l(\cdot)$ with respect to the k-th slice $\mathcal{X}_k$ is given by:

$$l(\mathcal{G},\mathcal{U},\mathcal{P}_k,\mathcal{X}_k) \triangleq \min_{\mathbf{u}_k \in \mathbb{R}^{r_{(N+1)}}} \| \mathcal{P}_k \circledast \left(\mathcal{X}_k - \mathcal{G} \prod_{n=1}^{N} \times_n \mathbf{U}^{(n)} \times_{N+1} \mathbf{u}_k^{\top} \right) \|_F^2, \qquad \text{(Equation 10)}$$

and $\lambda \in (0, 1]$ is the forgetting parameter. Here, all observations (i.e., tensor slices) in the time interval $[1, t]$ are taken into consideration in the estimation of the underlying low-rank component at each time t. The loss $l(.)$ presents the residual for each observation, which measures the difference between the observed value and the estimated value of the tensor slice. As there are many choices of low-rank estimation for a given data stream $\mathcal{X}_k$, $l(.)$ in Equation 10 is defined as the minimum loss over all possible choices, and hence, it results in the best low-rank tensor approximation to $\mathcal{X}_k$. On the arrival of a new data $\mathcal{X}_t$ at each time t, thanks to Equation 10, we can obtain a good estimation of the coefficient vector $\mathbf{u}_t$, which is necessary to establish the first-order surrogate of the cost function $f_t(.)$, to be detailed later under proposed methods. λ is used to discount the effect of past observations exponentially and to ensure that observations in the distant past are substantially downweighted in the cost function relative to the latest ones. Accordingly, when $\lambda < 1$, this can facilitate the tracking ability of estimators, especially in time-varying and non-stationary environments. The effective window length for $\lambda < 1$ is $(1-\lambda)^{-1}$ when t is large. When $\lambda = 1$, Equation 9 boils down to its counterpart of Equation 5 in batch setting.

In the next two sections, we describe the two proposed algorithms for solving Equation 9 under CP and Tucker decompositions. We make the following four assumptions for the convenience of deploying our algorithms as well as analyzing their performance:

(A1) Observed tensor slices $\{\mathcal{X}_t\}_{t \geq 1}$ are independent and identically distributed from a data-generating distribution, which is the underlying distribution of the dataset, having a compact set $\mathcal{V}$. This assumption is very common for convergence analysis in online settings, in general, and adaptive tensor decomposition in particular,

e.g., Mardani et al.,[8] Kasai,[9] Mairal et al.,[55] and Thanh et al.[56,57] (A1) is a strong assumption in our analysis, but it can be relaxed as follows: observed tensor slices $\{\mathcal{X}_t\}_{t\geq 1}$ are Frobenius-norm bounded, i.e., $\|\mathcal{X}_t\|_F < M < \infty$. Low-rank components $\{\mathcal{Y}_t\}_{t\geq 1}$ of the observed tensor slices $\{\mathcal{X}_t\}_{t\geq 1}$ are assumed to be deterministic and bounded. Noise tensors $\{\mathcal{N}_t\}_{t\geq 1}$ are i.i.d. from a distribution having a compact support.

(A2) Tensor slices $\{\mathcal{X}_t\}_{t\geq 1}$ follow the data model (Equation 7), where the true underlying loading factors $\{\mathbf{U}_t^{(n)}\}_{t\geq 1}$ are bounded, i.e., $\|\mathbf{U}_t^{(n)}\|_F \leq \kappa < \infty$. It prevents arbitrarily large values in $\mathbf{U}_t^{(n)}$ and ill-conditioned computation. Furthermore, we assume that $\mathbf{U}_t^{(n)}$ is full-column rank for every n and t. This constraint is useful to establish nice propositions in convergence analysis (e.g., boundedness of solutions and Lipschitz continuity of the objective function) as well as to improve the well-posedness of the tensor tracking problem. When (A1) holds, (A2) naturally holds.

(A3) Observation mask tensors $\{\mathcal{P}_t\}_{t\geq 1}$ are independent of $\{\mathcal{X}_t\}_{t\geq 1}$, and their entries obey the uniform distribution. With respect to the imputation of missing values and recovery of low-rank components, the uniform randomness allows the sequence of binary masks $\{\mathcal{P}_t\}_{t\geq 1}$ to admit stable recovery, which is defined as follows: *Definition 1 (stable recovery[58]). We say that the sequence of binary masks $\{\mathcal{P}_t\}_{t\geq 1}$ admits stable recovery if it satisfies the following property: assume two sequences, $\{\mathcal{A}_t\}_{t\geq 1}$, $\{\mathcal{B}_t\}_{t\geq 1}$, where $\mathcal{A}_t$ and $\mathcal{B}_t$ share the same size as $\mathcal{P}_t$ and the rank and the maximum value of $\mathcal{A}_t$ and $\mathcal{B}_t$ are bounded for every t. For any $\varepsilon > 0$, there exists $\delta > 0$, depending only on ε, such that if $\text{limsup}_{t\to\infty} \|\mathcal{P}_t \circledast (\mathcal{A}_t - \mathcal{B}_t)\|_{\bar{F}} \leq \delta$, then $\text{limsup}_{t\to\infty} \|\mathcal{A}_t - \mathcal{B}_t\|_{\bar{F}} \leq \varepsilon$, where $\|.\|_{\bar{F}}$ is the averaged Frobenius norm defined as $\|\mathcal{A}\|_{\bar{F}} = \|\mathcal{A}\|_F / \sqrt{I_1 I_2 \ldots I_N}$.* Moreover, the number of observed entries in $\mathcal{X}_t$ is assumed to be larger than the lower bound $\mathcal{O}(rL\log(L))$, where $L = \sqrt{I_1 I_2 \ldots I_N}$ and $r = \max(r_1, r_2, \ldots, r_N)$, and every row of $\underline{\mathbf{X}}_t^{(n)}$ is observed at least r entries for all n. It is indicated in Candes and Tao[59] that no completion method can recover missing data of $\mathbf{M} \in \mathbb{R}^{n\times n}$ with rank $r = \mathcal{O}(1)$ unless the number of observed entries in $\mathbf{M}$ satisfies $m \geq cn\log n$ for some positive constant $c > 0$, and this lower bound is the information theoretical limit. The constraints are fundamental conditions to prevent the problem of completion/imputation from being underdetermined where available observations may be insufficient to cover missing entries.

(A4) The low multilinear-rank model is either static or slowly time-varying, i.e., the core tensor and loading factors may vary slowly between two consecutive times $t-1$ and t: $\mathcal{G}_t \simeq \mathcal{G}_{t-1}$ and $\mathbf{U}_t^{(n)} \simeq \mathbf{U}_{t-1}^{(n)}$. The tensor rank is assumed to be known.

PROPOSED METHODS

In this section, we first propose a fast adaptive Tucker algorithm called ATD for tracking the online t-LRA of incomplete streaming tensors. Then, a novel variant of ATD is presented based on the CP format, namely ACP. Next, we provide a performance analysis in terms of complexity and convergence to demonstrate their effectiveness and efficiency.

Proposed ATD algorithm

Leveraging past estimations of the loading factors and the core tensor, we propose to minimize the surrogate $g_t(\mathcal{G},\mathcal{U})$ of $f_t(\mathcal{G},\mathcal{U})$ instead, which is defined, for a given value of $\{\mathbf{u}_k\}_{1\leq k\leq t}$, by:

$$g_t(\mathcal{G},\mathcal{U}) = \frac{1}{t}\sum_{k=1}^{t}\lambda^{t-k}\|\mathcal{P}_k \circledast \left(\mathcal{X}_k - \mathcal{G}\prod_{n=1}^{N}\times_n \mathbf{U}^{(n)} \times_{N+1}\mathbf{u}_k^\top\right)\|_F^2. \quad \text{(Equation 11)}$$

The main motivation here stems from the following observations: first, it is easy to verify that $g_t(\mathcal{G},\mathcal{U})$ provides an upper bound on $f_t(\mathcal{G},\mathcal{U})$ (i.e., $f_t(\mathcal{G},\mathcal{U}) \leq g_t(\mathcal{G},\mathcal{U})$ for all $\mathcal{G}$ and $\mathcal{U}$, and a fixed set of $\{\mathbf{u}_k\}_{1\leq k\leq t}$). Also, the error function $e_t(\mathcal{G},\mathcal{U}) = g_t(\mathcal{G},\mathcal{U}) - f_t(\mathcal{G},\mathcal{U})$ is L-smooth for some constant $L > 0$, i.e., it is differentiable, and $\nabla e_t(\mathcal{G},\mathcal{U})$ is L-Lipschitz continuous. As a result, $g_t(\mathcal{G},\mathcal{U})$ is a *first-order surrogate* function of $f_t(\mathcal{G},\mathcal{U})$,[60] and hence, its theoretical convergence results can be achieved without making any strong assumptions on $f_t(\mathcal{G},\mathcal{U})$. In particular, the sequence of surrogate values $\{g_t(\mathcal{G}_t,\mathcal{U}_t)\}_{t=1}^{\infty}$ is a quasi-martingale and converges almost surely. Accordingly, under a simple assumption that the directional derivative of f_t exists in any direction at any $\mathcal{G}$ and $\mathcal{U}$, $\{g_t(\mathcal{G}_t,\mathcal{U}_t)\}_{t=1}^{\infty}$ and $\{f_t(\mathcal{G}_t,\mathcal{U}_t)\}_{t=1}^{\infty}$ converge to the same limit. Indeed, the solution $\{\mathcal{G}_t,\mathcal{U}_t\}$ derived from minimizing $g_t(\mathcal{G},\mathcal{U})$ converges to a stationary point of $f_t(\mathcal{G},\mathcal{U})$ when t approaches infinity. Furthermore, $g_t(\mathcal{G},\mathcal{U})$ can be effectively minimized with a convergence rate of $\mathcal{O}(1/t)$, and it is much simpler than minimizing $f_t(\mathcal{G},\mathcal{U})$.

To obtain a low-complexity estimator, we exploit the fact that Equation 11 can be efficiently solved using the alternating minimization framework, whose iteration step coincides with the tensor slice's acquisition in time. In particular, it can be divided into three main stages: (1) estimate $\mathbf{u}_t$ first, given the old estimation of $\mathcal{G}_{t-1}$ and $\mathcal{U}_{t-1}$; (2) update the loading factor $\mathbf{U}_t^{(n)}$, given $\mathbf{u}_t$, $\mathcal{G}_{t-1}$, and the remaining factors; and (3) estimate the core tensor $\mathcal{G}_t$. The proposed ATD algorithm is summarized in Algorithm 1. In the following, we will describe the key steps of our algorithm for minimizing Equation 11.

Estimation of $\mathbf{u}_t$

Under the assumption that the loading factors and the core tensor might be static or slowly time-varying (i.e., $\mathcal{U}_t \simeq \mathcal{U}_{t-1}$ and $\mathcal{G}_t \simeq \mathcal{G}_{t-1}$), the weight vector $\mathbf{u}_t$ can be derived from the loss function $l(.)$ in Equation 10 at time t by:

$$\mathbf{u}_t = \underset{\mathbf{u}\in\mathbb{R}^{r(N+1)}}{\text{argmin}} \|\mathcal{P}_t \circledast (\mathcal{X}_t - \mathcal{H}_t \times_{N+1}\mathbf{u}^\top)\|_2^2, \quad \text{(Equation 12)}$$

where $\mathcal{H}_t = \mathcal{G}_{t-1}\prod_{n=1}^{N}\times_n \mathbf{U}_{t-1}^{(n)}$. Equation 12 can be readily converted into the standard form of:

$$\mathbf{u}_t = \underset{\mathbf{u}\in\mathbb{R}^{r(N+1)}}{\text{argmin}} \|\mathbf{P}_t(\mathbf{x}_t - \mathbf{H}_t\mathbf{u})\|_2^2, \quad \text{(Equation 13)}$$

where $\mathbf{P}_t = \text{diag}(\text{vec}(\mathcal{P}_t))$, $\mathbf{x}_t = \text{vec}(\mathcal{X}_t)$, and $\mathbf{H}_t$ is the unfolding matrix of the tensor $\mathcal{H}_t$. For the sake of convenience, let Ω_t

Algorithm 1. Adaptive Tucker decomposition

Input:

-Incomplete slices $\{\mathcal{P}_t \circledast \mathcal{X}_t\}_{t=1}^{\infty}, \mathcal{X}_t \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$

- Tucker rank $\boldsymbol{r}_{TD} = [r_1, r_2, \ldots, r_{(N+1)}]$, forgetting factor $\lambda \in (0, 1]$
- Parameters: $\alpha > 0, \delta > 0$, and $m > 0$

Output: Loading factors $\{\mathbf{U}_t^{(n)}\}_{n=1}^{N}$ and the core tensor $\mathcal{G}_t$

Initialization: $\{\mathbf{U}_0^{(n)}\}_{n=1}^{N}$ and $\mathcal{G}_0$ are initialized randomly and $\{\mathbf{S}_t^{(n)}\}_{n=1}^{N} = \delta \boldsymbol{I}_{r_n}$

Main Program:

for $t = 1, 2, \ldots$ **do**

$\mathcal{X}_{\Omega_t} = \mathcal{P}_t \circledast \mathcal{X}_t$

//Stage 1: Estimation of $\boldsymbol{u}_t$

$\mathcal{S}_t = \text{randomsample}(|\Omega_t|, \lfloor m\, r_{(N+1)} \log r_{(N+1)} \rceil)$

$\mathcal{H}_t = \mathcal{G}_{t-1} \prod_{n=1}^{N} \times_n \boldsymbol{U}_{t-1}^{(n)}$

$\boldsymbol{u}_t = (\boldsymbol{H}_{\mathcal{S}_t}^{\top} \boldsymbol{H}_{\mathcal{S}_t} + \alpha \boldsymbol{I})^{-1} \boldsymbol{H}_{\mathcal{S}_t}^{\top} \boldsymbol{x}_{\mathcal{S}_t} \boldsymbol{u}_t$

$\Delta \mathcal{X}_t = \mathcal{P}_t \circledast (\mathcal{X}_t - \mathcal{H}_t \times_{N+1})$

//Stage 2: Estimation of $\{\mathbf{U}_t^{(n)}\}_{n=1}^{N}$

for $n = 1, 2, \ldots, N$ **do**

$\boldsymbol{W}_t^{(n)} = (\boldsymbol{U}_{t-1}^{(n)})^{\#} \underline{\boldsymbol{X}}_{\Omega_t}^{(n)}$

$\boldsymbol{S}_t^{(n)} = \lambda \boldsymbol{S}_{t-1}^{(n)} + \boldsymbol{W}_t^{(n)} (\boldsymbol{W}_t^{(n)})^{\top}$

$\boldsymbol{V}_t^{(n)} = (\boldsymbol{S}_t^{(n)})^{-1} \boldsymbol{W}_t^{(n)}$

$\boldsymbol{U}_t^{(n)} = \boldsymbol{U}_{t-1}^{(n)} + \Delta \underline{\boldsymbol{X}}_t^{(n)} (\boldsymbol{V}_t^{(n)})^{\top}$

end for

//Stage 3: Estimation of $\mathcal{G}_t$

$\boldsymbol{Z}_t = \boldsymbol{u}_t \otimes (\otimes_{n=2}^{N-1} \boldsymbol{U}_t^{(n)})$

$\Delta \boldsymbol{G}_t = (\boldsymbol{U}_t^{(1)})^{\#} \Delta \underline{\boldsymbol{X}}_t^{(1)} \boldsymbol{Z}_t^{\#}$

$\Delta \mathcal{G}_t = \text{reshape}(\Delta \boldsymbol{G}_t, \boldsymbol{r}_{TD})$

$\mathcal{G}_t = \mathcal{G}_{t-1} + \Delta \mathcal{G}_t$

end for

and $\mathbf{x}_{\Omega_t}$ be the set and vector containing the observed entries of $\mathcal{X}_t$, while $\mathbf{H}_{\Omega_t}$ is the submatrix of $\mathbf{H}_t$ obtained by selecting the rows corresponding to $\mathbf{x}_{\Omega_t}$.

Generally, Equation 13 is an overdetermined least-squares (LS) regression and requires $\mathcal{O}(|\Omega_t| r^2)$ with respect to (w.r.t.) computational complexity to compute the exact LS solution.[61] Thus, it costs time and effort when handling high-dimensional and high-order tensors. We propose to solve a regularized LS sketch of Equation 13 instead, i.e.,

$$\mathbf{u}_t = \underset{\mathbf{u} \in \mathbb{R}^{r_{(N+1)}}}{\text{argmin}} \parallel \mathcal{L}(\mathbf{x}_{\Omega_t} - \mathbf{H}_{\Omega_t} \mathbf{u}) \parallel_2^2 + \alpha \parallel \mathbf{u} \parallel_2^2, \qquad \text{(Equation 14)}$$

where α is a small positive parameter for regularization, and $\mathcal{L}(.)$ is a sketching map that helps reduce the sample size and hence speed up the calculations. Accordingly, the updated rule for $\mathbf{u}_t$ is given by:

$$\mathbf{u}_t = \left(\mathbf{H}_{\mathcal{S}_t}^{\top} \mathbf{H}_{\mathcal{S}_t} + \alpha \mathbf{I}\right)^{-1} \mathbf{H}_{\mathcal{S}_t}^{\top} \mathbf{x}_{\mathcal{S}_t}, \qquad \text{(Equation 15)}$$

where $\mathbf{H}_{\mathcal{S}_t}$ and $\mathbf{x}_{\mathcal{S}_t}$ are transformed versions of $\mathbf{H}_{\Omega_t}$ and $\mathbf{x}_{\Omega_t}$ under the sketching $\mathcal{L}(.)$, respectively. Here, the introduction of $\alpha \parallel \mathbf{u} \parallel_2^2$ is made to avoid the singular/ill-posed computation, multicollinearity, and other pathological phenomena in practice. For example, in some cases, the matrix $\mathbf{H}_{\mathcal{S}_t}$ in Equation 18 may be rank deficient or near singular. The computation of its inverse is prone to large numerical errors, and hence, the ordinary LS solution is no longer well defined. The presence of $\alpha \parallel \mathbf{u} \parallel_2^2$ with $\alpha > 0$ results in an additional positive term, $\alpha \mathbf{I}$, on the diagonal of the moment matrix $\mathbf{H}_{\mathcal{S}_t}^{\top} \mathbf{H}_{\mathcal{S}_t}$, while it does not change the eigenvectors of $\mathbf{H}_{\mathcal{S}_t}^{\top} \mathbf{H}_{\mathcal{S}_t}$. Accordingly, it can ensure that all of the eigenvalues are strictly greater than 0. In other words, the introduction of the regularization term can prevent singularity and ill-posed problems. With respect to the interpretability aspect, the inclusion of $\alpha \parallel \mathbf{u} \parallel_2^2$ effectively eliminates multicollinearity—a phenomenon where two or more variables are highly correlated—that affects the interpretability of regression models, including the LS estimation.[62,63] It stems from the fact that adding $\alpha \parallel \mathbf{u} \parallel_2^2$ introduces bias into an unbiased LS estimation, but it reduces the variance. Thereby, the regularized LS solver can result in a more precise and hence more interpretable estimation in the case in which the multicollinearity problem exists in data. Typically, the value of α can be chosen by cross validation in a batch setting. However, it turns out to be inefficient for stream processing due to its high complexity. In this work, the optimal value of the coefficient vector $\mathbf{u}_t$ is perfect, but a good estimation of $\mathbf{u}_t$ is sufficient for tensor tracking. A small α close to the noise level is enough to avoid singular/ill-posed problems during the tracking process. Therefore, we can choose its value in the range $[10^{-3}, 1]$ for reasonable performance in practice.

Thanks to the tensor structure of $\mathcal{H}_t$, the uniform row sampling is effective in many cases, especially when we deal with a high-order streaming tensor (N is large) and/or with some incoherent tensor factors (see Appendix A for details). In the presence of highly coherent factors, a preconditioning (mixing) step is necessary to guarantee the incoherence. For instance, the subsampled randomized Hadamard transform (SRHT) is a good candidate that can produce a transformed matrix whose rows have (almost) uniform leverage scores.[64] In this context, we here emphasize that well-known randomized LS algorithms can help save much computational complexity while obtaining reasonable estimations of $\mathbf{u}_t$, especially for large-scale low-rank tensors. It is also worth noting that the update of $\mathbf{u}_t$ costs the most computation time of every tensor tracker, as it requires all loading factors $\{\mathbf{U}_{t-1}^{(n)}\}_{n=1}^{N}$ to form $\mathcal{H}_t$, and hence solves the LS problem. Therefore, the proposed randomized technique here plays an important role in reducing the overall complexity.

Estimation of $\mathbf{U}_t^{(n)}$

The loading factor $\mathbf{U}_t^{(n)}$ can be updated by minimizing $g_t(\cdot)$ w.r.t. $\mathbf{U}^{(n)}$, as:

$$\mathbf{U}_t^{(n)} = \underset{\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times r_n}}{\text{argmin}} \left[\frac{1}{t} \sum_{k=1}^{t} \lambda^{t-k} \parallel \underline{\mathbf{P}}_k^{(n)} \circledast \left(\underline{\mathbf{X}}_k^{(n)} - \mathbf{U}^{(n)} \mathbf{W}_k^{(n)} \right) \parallel_F^2 \right], \qquad \text{(Equation 16)}$$

where $\underline{\mathbf{X}}_k^{(n)}$ (resp. $\underline{\mathbf{P}}_k^{(n)}$) is the mode-$n$ unfolding of $\mathcal{X}_k$ (resp. $\mathcal{P}_k$) and the coefficient matrix $\mathbf{W}_k^{(n)}$ is the mode-n unfolding of the tensor $\mathcal{W}_k$, which is defined by $\mathcal{W}_k = (\mathcal{G}_{t-1} \prod_{i=1, i \neq n}^{N} \times_i \mathbf{U}_{t-1}^{(i)}) \times_{N+1} \mathbf{u}_k^{\top}$.

Interestingly, we exploit the fact that minimization (Equation 16) can boil down to the problem of subspace tracking in the presence of missing data.[65] Particularly, the solution of Equation 16 can be obtained by minimizing subproblems for each row $\mathbf{u}_m^{(n)}$ of $\mathbf{U}^{(n)}$, $m = 1, 2, \ldots, I_n$, as:

$$\mathbf{u}_{t,m}^{(n)} = \underset{\mathbf{u}_m^{(n)} \in \mathbb{R}^r}{\operatorname{argmin}} \left[\frac{1}{t} \sum_{k=1}^{t} \lambda^{t-k} \| \underline{\mathbf{P}}_{k,m}^{(n)} \left(\left(\underline{\mathbf{x}}_{k,m}^{(n)} \right)^\top - \mathbf{W}_k^{(n)} \left(\mathbf{u}_m^{(n)} \right)^\top \right) \|_F^2 \right], \quad \text{(Equation 17)}$$

where $\underline{\mathbf{x}}_{k,m}^{(n)}$ is the m-th row of $\underline{\mathbf{X}}_k^{(n)}$ and $\underline{\mathbf{P}}_{k,m}^{(n)} = \operatorname{diag}\left(\underline{\mathbf{P}}_k^{(n)}(m,:) \right)$. Thanks to the parallel scheme of the well-known PETRELS algorithm for subspace tracking,[66] we derive an efficient estimator for minimizing the exponentially weighted LS cost function (Equation 16). Particularly, we first define two auxiliary matrices, $\mathbf{S}_t^{(n)}$ and $\mathbf{V}_t^{(n)}$, as follows:

$$\mathbf{S}_t^{(n)} = \lambda \mathbf{S}_{t-1}^{(n)} + \left(\mathbf{W}_t^{(n)} \right)^\top \mathbf{W}_t^{(n)} \text{ and } \mathbf{V}_t^{(n)} = \left(\mathbf{S}_t^{(n)} \right)^{-1} \left(\mathbf{W}_t^{(n)} \right)^\top. \quad \text{(Equation 18)}$$

The loading factor $\mathbf{U}_t^{(n)}$ is then updated recursively by:

$$\mathbf{U}_t^{(n)} = \mathbf{U}_{t-1}^{(n)} + \Delta \underline{\mathbf{X}}_t^{(n)} \left(\mathbf{V}_t^{(n)} \right)^\top, \quad \text{(Equation 19)}$$

where the matrix $\Delta \underline{\mathbf{X}}_t^{(n)}$ is derived from the mode-n unfolding of the residual error tensor $\Delta \mathcal{X}_t = \mathcal{P}_t \circledast (\mathcal{X}_t - \mathcal{H}_t \times_{N+1} \mathbf{u}_t^\top)$. To enable the recursive updating rule, the matrix $\mathbf{S}_0^{(n)}$ is initialized by a scaled identity matrix, $\mathbf{S}_0^{(n)} = \delta_n \mathbf{I}_{r_n}$ with $\delta_n > 0$. This is not PETRELS, but a modified version. Here, we can utilize the already updated $\mathbf{U}_t^{(n)}$ to track the remaining factors, which can improve the rate of convergence. Also, we can estimate all the N factors in a parallel scheme, which further reduces the overall cost when several computational units are available.

Estimation of $\mathcal{G}_t$

For the estimation of $\mathcal{G}_t$, given the latest updated loading factors, Equation 11 is reformulated as:

$$\mathcal{G}_t = \underset{\mathcal{G}}{\operatorname{argmin}} \left[\frac{1}{t} \sum_{k=1}^{t} \lambda^{t-k} \| \underline{\mathbf{P}}_k^{(1)} \circledast \left(\underline{\mathbf{X}}_k^{(1)} - \mathbf{U}_t^{(1)} \underline{\mathbf{G}}^{(1)} \mathbf{Z}_k \right) \|_F^2 \right], \quad \text{(Equation 20)}$$

where the variable $\underline{\mathbf{G}}^{(1)}$ is the mode-1 unfolding of $\mathcal{G}$ and the matrix $\mathbf{Z}_k$ is given by $\mathbf{Z}_k = \mathbf{u}_k \otimes (\otimes_{n=2}^{N} \mathbf{U}_t^{(n)})$.

When handling a streaming tensor with a huge number of slices (i.e., t is large) and a large number of unknown parameters in $\mathcal{G}$ (i.e., $\prod_{n=1}^{N+1} r_n$ is large), applying batch gradient methods for Equation 20 may be time consuming despite the effect of the forgetting factor λ. Stochastic approximation is introduced as a good alternative.[67] In particular, we minimize the following function:

$$\mathcal{G}_t = \underset{\mathcal{G}}{\operatorname{argmin}} \| \underline{\mathbf{P}}_t^{(1)} \circledast \left(\underline{\mathbf{X}}_t^{(1)} - \mathbf{U}_t^{(1)} \underline{\mathbf{G}}^{(1)} \mathbf{Z}_t \right) \|_F^2. \quad \text{(Equation 21)}$$

Given the estimation of $\mathcal{U}_t$, the residual error between the newcoming tensor slice and the recovered one is given by $\Delta \underline{\mathbf{X}}_t^{(1)} = \underline{\mathbf{P}}_t^{(1)} \circledast \left(\underline{\mathbf{X}}_t^{(1)} - \mathbf{U}_t^{(1)} \underline{\mathbf{G}}_{t-1}^{(1)} \mathbf{Z}_t \right)$. Accordingly, we can derive the variation of $\mathcal{G}$ at time t from:

$$\Delta \underline{\mathbf{X}}_t^{(1)} = \underline{\mathbf{P}}_t^{(1)} \circledast \left(\mathbf{U}_t^{(1)} \Delta \underline{\mathbf{G}}_t^{(1)} \mathbf{Z}_t \right), \quad \text{(Equation 22)}$$

where $\Delta \underline{\mathbf{G}}_t^{(1)} = \underline{\mathbf{G}}_t^{(1)} - \underline{\mathbf{G}}_{t-1}^{(1)}$. In particular, $\Delta \underline{\mathbf{G}}_t$ is computed as:

$$\Delta \underline{\mathbf{G}}_t^{(1)} = \left(\mathbf{U}_t^{(1)} \right)^{\#} \Delta \underline{\mathbf{X}}_t^{(1)} \mathbf{Z}_t^{\#}. \quad \text{(Equation 23)}$$

As $\mathbf{Z}_t$ is of the Kronecker structure, we can obtain the pseudo-inverse of $\mathbf{Z}_t$ efficiently by using the following nice property[68]: $(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_n)^{\#} = \mathbf{A}_1^{\#} \otimes \mathbf{A}_2^{\#} \otimes \cdots \otimes \mathbf{A}_n^{\#}$. After that, $\Delta \underline{\mathbf{G}}_t^{(1)}$ will be reshaped into a tensor $\Delta \mathcal{G}_t$ of size $r_1 \times r_2 \times \cdots \times r_{(N+1)}$. To sum up, we obtain the simple rule for updating $\mathcal{G}_t$ as follows:

$$\mathcal{G}_t = \mathcal{G}_{t-1} + \Delta \mathcal{G}_t. \quad \text{(Equation 24)}$$

We note that, for overdetermined cases, the rule for updating $\mathcal{G}_t$ can be sped up by using the following "vector trick": $\operatorname{vec}(\mathbf{ABC}^\top) = (\mathbf{C} \otimes \mathbf{A}) \operatorname{vec}(\mathbf{B})$, $\forall \mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ of suitable size. In particular, Equation 22 can be cast into the standard LS format as follows:

$$\boldsymbol{\delta}\mathbf{x}_t = \mathbf{P}_t \left(\mathbf{u}_t \otimes \left(\overset{N}{\underset{n=1}{\otimes}} \mathbf{U}_t^{(n)} \right) \right) \boldsymbol{\delta}\mathbf{g}_t, \quad \text{(Equation 25)}$$

where $\boldsymbol{\delta}\mathbf{x}_t = \operatorname{vec}\left(\Delta \underline{\mathbf{X}}_t^{(1)} \right)$, $\boldsymbol{\delta}\mathbf{g}_t = \operatorname{vec}\left(\Delta \underline{\mathbf{G}}_t^{(1)} \right)$ and $\mathbf{P}_t = \operatorname{diag}\left(\operatorname{vec}\left(\underline{\mathbf{P}}_t^{(1)} \right) \right)$. Interestingly, Equation 25 has a Kronecker structure; thus, $\boldsymbol{\delta}\mathbf{g}_t$ can be efficiently computed by applying randomized sketching techniques with a much lower complexity, e.g., the uniform sampling or the Kronecker product regression in Diao et al.[69]

Variants of ATD

Orthogonal ATD

In the cases where orthogonality constraints are imposed on the loading factors, we add an orthogonalization step of $\mathbf{U}^{(n)}$ at each time t as follows:

$$\mathbf{U}_t^{(n)} = \mathbf{U}_t^{(n)} \left[\left(\mathbf{U}_t^{(n)} \right)^\top \mathbf{U}_t^{(n)} \right]^{-1/2}, \quad \text{(Equation 26)}$$

where $(.)^{-1/2}$ represents the inverse square root, or simply take the QR decomposition of $\mathbf{U}_t^{(n)}$. Accordingly, the update of $\Delta \underline{\mathbf{G}}_t$ in Equation 23 can be sped up by replacing the pseudo-inverse with the transpose operator:

$$\Delta \underline{\mathbf{G}}_t = \left(\mathbf{U}_t^{(1)} \right)^\top \Delta \underline{\mathbf{X}}_t^{(1)} \mathbf{Z}_t^\top. \quad \text{(Equation 27)}$$

We refer to this variant of ATD as ATD-O.

Adaptive CP decomposition

It is well known that CP decomposition is viewed as a special case of Tucker decomposition when the core tensor is an identity tensor, $\mathcal{I}$, of size $r \times r \times \cdots \times r$, thanks to the following relation:

$$\mathcal{I} \prod_{n=1}^{N} \times_n \mathbf{U}^{(n)} = \sum_{i=1}^{r} \mathbf{u}_i^{(1)} \circ \mathbf{u}_i^{(2)} \circ \cdots \circ \mathbf{u}_i^{(N)}. \quad \text{(Equation 28)}$$

Therefore, we can derive a new ACP from ATD. Particularly in step 1 and step 2 of ATD, we recast the design matrix $\mathbf{H}_t$ in Equation 13 into $\mathbf{H}_t = \odot_{n=1}^{N} \mathbf{U}_{t-1}^{(n)}$ and the coefficient matrix $\mathbf{W}_k^{(n)}$ in Equation 16 into $\mathbf{W}_k^{(n)} = (\odot_{i=1, i \neq n}^{N} \mathbf{U}_{t-1}^{(i)}) \odot \mathbf{u}_k^\top$. Meanwhile, step

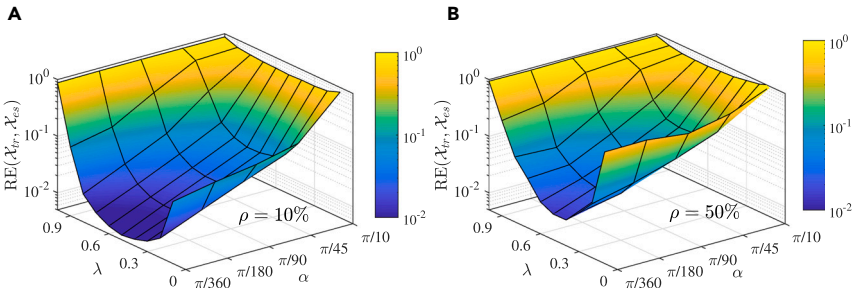

Figure 2. Effect of the forgetting factor λ on the performance of ACP versus the rotation angle α

Its performance is evaluated on a synthetic fourth-order streaming tensor of size 20 × 20 × 20 × 500 and rank 5, while the noise level σ is fixed at 10^{-3}. (A) and (B) respectively illustrate the estimation accuracy of ACP with 10% and 50% missing data.

3 of ATD is no longer required as we set the core tensor $\mathcal{G}_t$ to $\mathcal{I}$. This modification of ATD is referred to as ACP, which stands for adaptive CP decomposition.

Performance analysis

Memory storage and computational complexity

We assume that the fixed dimensions of the streaming tensor are equal to I and the desired Tucker rank is $\mathbf{r}_{\mathrm{TD}} = [r, r, \ldots, r]$. In terms of memory storage, ATD requires $\mathcal{O}(r^{N+1})$ and $\mathcal{O}(NIr)$ words of memory to save the core tensor $\mathcal{G}$ and N tensor factors $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$, respectively. In addition, the cost of saving N matrices $\mathbf{S}_t^{(n)}$ is $\mathcal{O}(Nr^2)$ words of memory. In total, ATD requires $\mathcal{O}(Nr(I+r)+r^{N+1})$ words of memory at each time t. Meanwhile, ACP costs a lower memory storage of $\mathcal{O}(Nr(I+r))$, as it does not need to save the core tensor at each time t.

In terms of computational complexity, the computation of ATD comes from three main estimations: (1) the weight vector $\mathbf{u}_t$, (2) the tensor factors $\{\mathbf{U}^{(n)}\}_{n=1}^{N}$, and (3) the core tensor $\mathcal{G}$. The two former estimations require a cost of $\mathcal{O}(|\Omega_t|r + (I^{N-1} + |\mathcal{S}_1|)r^2)$ flops in a parallel scheme where $\mathcal{S}_1$ denotes the size of the sampling set of Equation 13. The latter estimation costs $\mathcal{O}(|\Omega_t|r + I^{N-1}r^{2(N+1)})$ flops for computing $\Delta\mathcal{X}$ and $\Delta\mathcal{G}$. If using the randomize technique in this stage, the complexity is reduced to $\mathcal{O}(|\Omega_t|r + |\mathcal{S}_2|r^{2(N+1)})$ flops, where $\mathcal{S}_2$ is the set of samples selected from Equation 25. Therefore, the overall computational complexity of ATD is $\mathcal{O}(|\Omega_t|r + (I^{N-1} + |\mathcal{S}_1|)r^2 + |\mathcal{S}_2|r^{2(N+1)})$ flops in a parallel scheme. For ACP, the overall computational complexity is $\mathcal{O}(|\Omega_t|r + (NIr^{N-1} + |\mathcal{S}_1|)r^2)$ flops and reduces to $\mathcal{O}(|\Omega_t|r + (I^{N-1} + |\mathcal{S}_1|)r^2)$ flops in a parallel scheme. Note that when a preconditioning step (e.g., SRHT) is needed to guarantee the incoherence of $\mathbf{H}_{\Omega_t}$, ATD and ACP require an additional cost of $\mathcal{O}(|\Omega_t|r \log r^2)$ flops.

Convergence guarantee

Our main theoretical result is stated in the following lemma.

Lemma 1. Given assumptions (A1)–(A4), $\lambda = 1$, and the true $\mathcal{G}$ and $\mathcal{U}$ are fixed, the solutions $\{\mathcal{G}_t, \mathcal{U}_t\}_{t=1}^{\infty}$ generated by ATD converge to a stationary point of f_t when $t \to +\infty$, i.e., $\nabla f_t(\mathcal{G}_t, \mathcal{U}_t) \to 0$ as $t \to +\infty$.

Proof of Lemma 1 can be obtained by applying the same framework to derive the asymptotic convergence of adaptive algorithms for problems of online matrix and tensor factorization.[8,9,16,55,56,70] In particular, the analysis consists of the following three main stages: (S1) the surrogate function $g_t(\mathcal{G}, \mathcal{U})$ is strongly biconvex in the sense that $\mathcal{G}$ and $\mathcal{U}$ are seen as multivariate variables. Solutions $\{\mathcal{G}_t, \mathcal{U}_t\}_{t=1}^{\infty}$ generated by ATD are bounded and their variations between two successive time instances satisfy $\|\mathbf{U}_{t+1}^{(n)} - \mathbf{U}_t^{(n)}\|_F \to \mathcal{O}(1/t)$ a.s. (S2) The non-negative sequence $\{g_t(\mathcal{G}_t, \mathcal{U}_t)\}_{t=1}^{\infty}$ is a quasi-martingale and hence convergent almost surely. Furthermore, $g_t(\mathcal{G}_t, \mathcal{U}_t) - f_t(\mathcal{G}_t, \mathcal{U}_t) \to 0$ a.s. (S3) The empirical cost function $f_t(\mathcal{G}, \mathcal{U})$ is continuously differentiable and Lipschitz. The sequence of solutions $\{\mathcal{G}_t, \mathcal{U}_t\}_{t=1}^{\infty}$ converges to a stationary point of $f_t(\mathcal{G}, \mathcal{U})$, i.e., when $t \to \infty$, the gradient $\nabla f_t(\mathcal{G}_t, \mathcal{U}_t) \to 0$ a.s. We refer the readers to our companion work on robust tensor tracking[16] for details on this proof framework.

RESULTS

In this subsection, experiments are conducted to evaluate the performance of the two proposed algorithms (ACP and ATD) on both synthetic and real data. We also compare them with several state-of-the-art algorithms to provide practical evidence of their effectiveness and efficiency. All experiments are implemented on a Windows computer with an Intel Core i5-8300H and 16 GB of RAM.

Performance of ACP

We assess the performance of ACP w.r.t. the following aspects: (1) impact of algorithm parameters on its tracking ability; (2) performance of ACP in non-stationary and time-varying environments; and (3) effectiveness and efficiency of ACP compared with other adaptive CP decomposition algorithms.

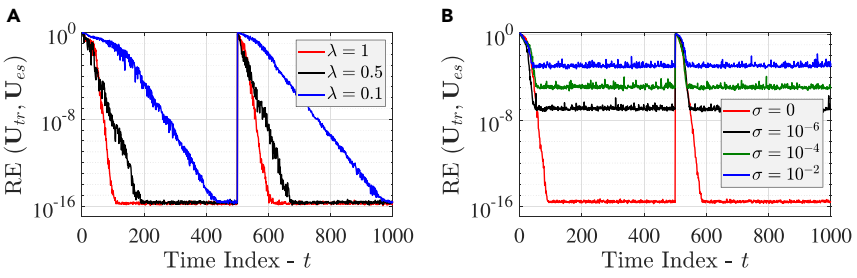

Figure 3. Performance of ACP in stationary environments

The underlying fourth-order streaming tensor is of size 20 × 20 × 20 × 1,000 and rank 5. An abrupt change is created at $t = 500$. (A) shows the tracking ability of ACP with different values of the forgetting factor λ in noise-free and stationary environments (i.e., the noise level σ = 0 and the rotation angle α = 0). (B) demonstrates the effect of noise on the performance of ACP with λ = 1.

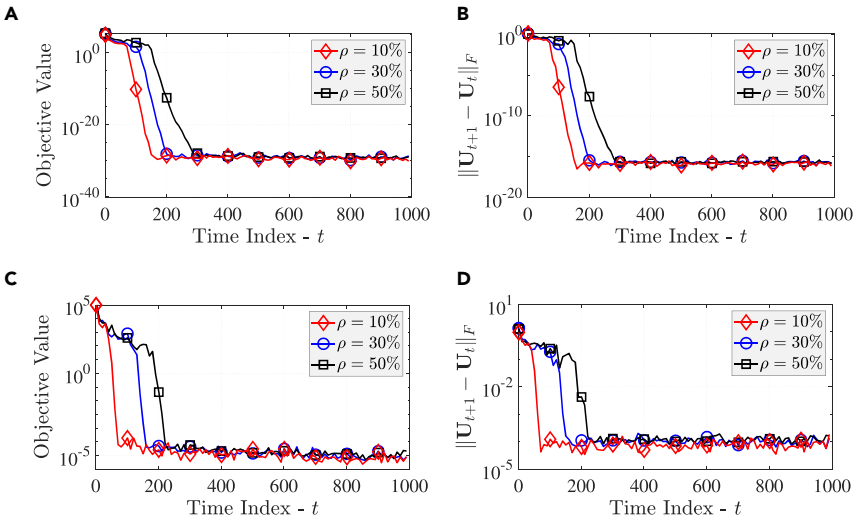

Figure 4. Convergence behavior of ACP

The underlying fourth-order streaming tensor is of size 20 × 20 × 20 × 1,000 and rank 5. Two noise levels ($\sigma = 0$ and $\sigma = 10^{-3}$) and three missing densities (ρ = 10%, 30%, and 50%) are considered. (A) and (C) respectively plot the objective value $f_t(\mathbf{U}_t)$ with time for $\sigma = 0$ and $\sigma = 10^{-3}$. (B) and (D) respectively represent the time variation $\|\mathbf{U}_{t+1} - \mathbf{U}_t\|_F$ for $\sigma = 0$ and $\sigma = 10^{-3}$.

Experimental setup

According to the setup of OLSTEC,[9] a time-varying model for streaming tensors is constructed as follows. At $t = 0$, the loading factor $\mathbf{U}_t^{(n)}$ is generated at random whose entries are i.i.d. drawn from the Gaussian distribution $\mathcal{N}(0,1)$. At time $t > 0$, $\mathbf{U}_t^{(n)} \in \mathbb{R}^{I_n \times r}$ is varied under the model $\mathbf{U}_t^{(n)} = \mathbf{U}_{t-1}^{(n)}\mathbf{Q}_t$, where $\mathbf{Q}_t \in \mathbb{R}^{r \times r}$ is a rotation matrix to control the variation of $\mathbf{U}^{(n)}$ between t and $t-1$, which is defined by:

$$\mathbf{Q}_t = \begin{bmatrix} \mathbf{I}_{p_t-1} & 0 & 0 & 0 \\ 0 & \cos(\alpha_t) & -\sin(\alpha_t) & 0 \\ 0 & \sin(\alpha_t) & \cos(\alpha_t) & 0 \\ 0 & 0 & 0 & \mathbf{I}_{r-p_t-1} \end{bmatrix}, \quad \text{(Equation 29)}$$

where $p_t = \mathrm{mod}(t + r - 2, r - 1) + 1$ and α_t is the rotation angle with $0 \le \alpha_t \le \pi/2$. Specifically, the higher the value of α_t is, the faster the loading factor $\mathbf{U}^{(n)}$ changes. The t-th slice $\mathcal{X}_t$ with missing entries is derived from the model:

$$\mathcal{X}_t = \mathcal{P}_t \circledast \left(\mathcal{I} \prod_{n=1}^{N} \times_n \mathbf{U}_t^{(n)} \times_{N+1} \mathbf{u}_t^\top + \sigma \mathcal{N}_t \right), \quad \text{(Equation 30)}$$

where $\mathcal{P}_t$ is a binary mask tensor whose entries are generated randomly using the Bernoulli model with the probability ρ, i.e., ρ represents the missing density in the measurement; $\mathcal{N}_t$ is a Gaussian noise tensor (with zero-mean, unit power entries) of the same size as $\mathcal{X}_t$, and the non-negative factor σ is to control the noise level; the weight vector $\mathbf{u}_t$ is a Gaussian random vector living on $\mathbb{R}^r$ space. To assess the adaptability of tensor algorithms to unforeseen events and unexpected corruption, we also create abrupt (significant) changes at some time instances during the tracking process by setting the rotation angle at this time instant to $\pi/2$.

To evaluate estimation accuracy, we measure the relative error (RE) metric defined by:

$$\mathrm{RE}(\mathbf{A}_{tr}, \mathbf{A}_{es}) = \frac{\|\mathbf{A}_{tr} - \mathbf{A}_{es}\|_F}{\|\mathbf{A}_{tr}\|_F}, \quad \text{(Equation 31)}$$

where $\mathbf{A}_{tr}$ (resp. $\mathbf{A}_{es}$) refers to the ground truth (resp. estimation). Due to the permutation and scaling indeterminacy of the CP decomposition, we can find $\mathbf{U}_{es}$, which is matched with $\mathbf{U}_{tr}$ from $\mathbf{U}_t$, as follows: $\mathbf{U}_{es} = \mathbf{U}_t\mathbf{P}^\top\mathbf{D}^{-1}$, where the permutation matrix $\mathbf{P} \in \mathbb{R}^{r \times r}$ and the diagonal matrix $\mathbf{D} \in \mathbb{R}^{r \times r}$ are derived by minimizing the optimization $\mathrm{argmin}_{\mathbf{D},\mathbf{P}} \|\mathbf{U}_t - \mathbf{U}_{tr}\mathbf{D}\mathbf{P}\|_F^2$.

Effect of forgetting factor λ

The choice of λ plays a central role in how effective and efficient ACP can be in non-stationary environments. To investigate the effect of the forgetting factor, we vary the value of λ from 0 to 1 and measure the estimation accuracy of ACP in different tests

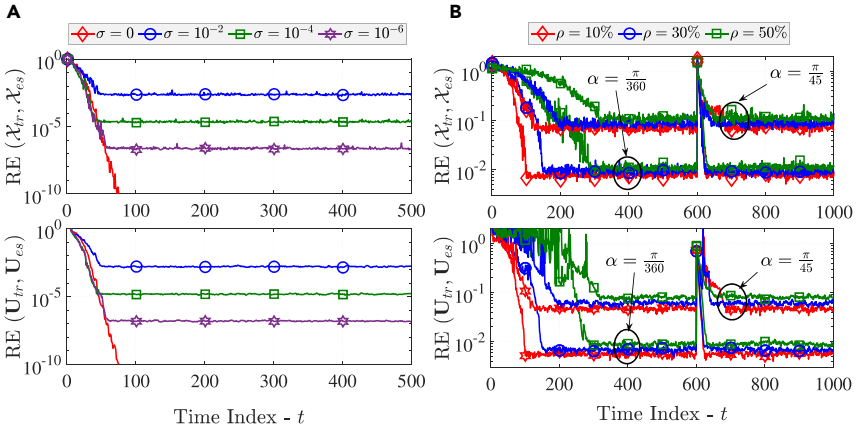

Figure 5. Performance of ACP in noisy and time-varying environments

The underlying fourth-order streaming tensor is of size 20 × 20 × 20 × 1,000 and rank 5. (A) and (B) respectively illustrate the effect of the noise level σ and the rotation angle α on the tracking ability of ACP with time. We fix the value of the rotation angle α at 0 in (A) and the noise level σ at 10^{-3} in (B). Also, an abrupt change is created at $t = 600$ in (B).

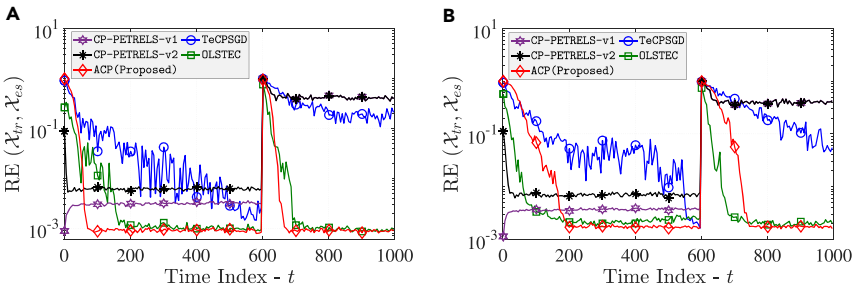

Figure 6. Tracking performance of adaptive CP algorithms

The underlying third-order streaming tensor is of size 20 × 20 × 1,000 and rank $r = 5$. The noise level and the rotation angle are set to $\sigma = 10^{-3}$ and $\alpha = \pi/360$. (A) and (B) illustrate the performance comparison between ACP and other adaptive CP algorithms in the presence of 10% and 50% missing data, respectively.

with regard to the rotational angle α. Figure 2 illustrates the experimental results of applying ACP to a synthetic fourth-order streaming tensor whose size is 20×20×20×500 and its rank $r = 5$. The noise level σ is set at 10^{-3}, while the sketching parameter m is fixed at 10. It is clear that the optimal value of λ depends not only on the rotation angle α, but also on the missing density ρ. When λ increases from 0 to 1, the performance of ACP first increases and then drops. Particularly when λ is close to 1, most of the observations are taken into the estimation of the underlying LRA of streaming tensors. However, the properties of old data may be very different from those of the latest observations, especially in fast time-varying environments. Therefore, the performance of ACP degrades significantly in such a case. When the value of λ is small, ACP discounts exponentially the influence of old observations, including the very recent ones. As a result, its convergence rate is slow in stationary or slowly time-varying environments. Accordingly, the forgetting factor λ should be neither too small nor too large. As can be seen in Figure 2, the value of λ should be around 0.5 for reasonable performance. Thus, we fix $\lambda = 0.5$ in the next experiments. It is worth noting that in stationary environments, we can set the value of $\lambda = 1$ to achieve the best performance (please see Figure 3 for an illustration).

Asymptotic convergence behavior

We next illustrate the convergence behavior of ACP in terms of the variation $\| \mathbf{U}_{t+1} - \mathbf{U}_t \|_F$ and the objective value $f_t(\mathcal{U}_t)$. We use the same fourth-order tensor as above, but with 1,000 tensor slices. Two noise levels are considered (including $\sigma = 0$ and $\sigma = 10^{-3}$), while the missing density ρ is chosen among $\{10\%, 30\%, 50\%\}$. The experimental results are shown in Figures 4 and S1 (in the supplemental information). We can see that the convergence results agree with those stated in Lemma 1.

Noisy and dynamic environments

First, the robustness of ACP is investigated against the noise variance. We test ACP's tracking ability on the same static fourth-order tensor as above with different values of the noise level σ. Figure 5A shows that the value of σ does not affect the convergence rate of ACP, but only its estimation error. Specifically, when we increase the noise level σ, the RE between the ground truth and the estimation gradually increases, but toward an error bound.

Next, we use the same tensor, but the number of slices is double to illustrate the robustness of ACP against time-varying environments. In particular, the proposed algorithm is evaluated in two scenarios, including a slow time-varying case (i.e., $\alpha = \pi/360$) and a fast time-varying case (i.e., $\alpha = \pi/45$). Also, at time $t = 600$, we make an abrupt change in these models. In addition, the missing density ρ is chosen among $\{10\%, 30\%, 50\%\}$. Experimental results indicate that ACP is capable of tracking t-LRA in dynamic environments, as shown in Figure 5B. In both scenarios, the RE between the ground truth and the estimation always converges toward a steady-state error bound.

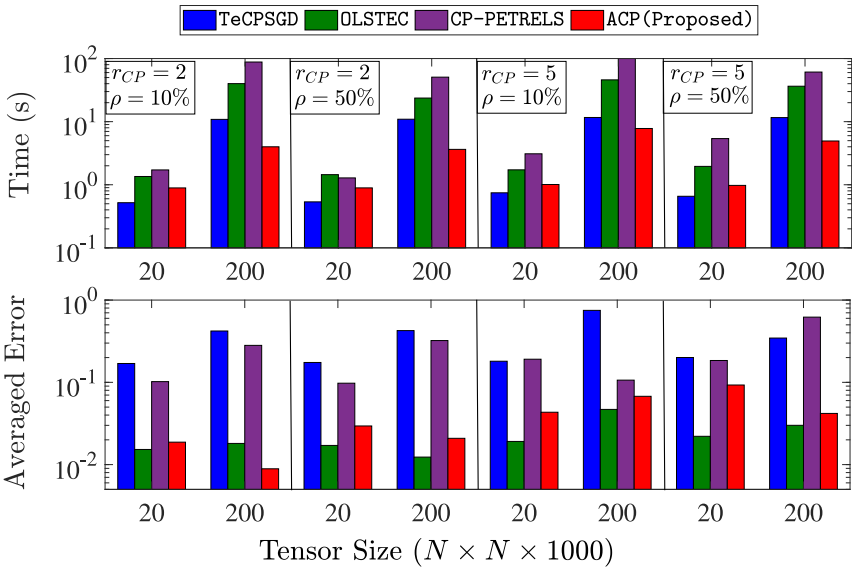

Figure 7. Running time and averaged error of adaptive CP algorithms

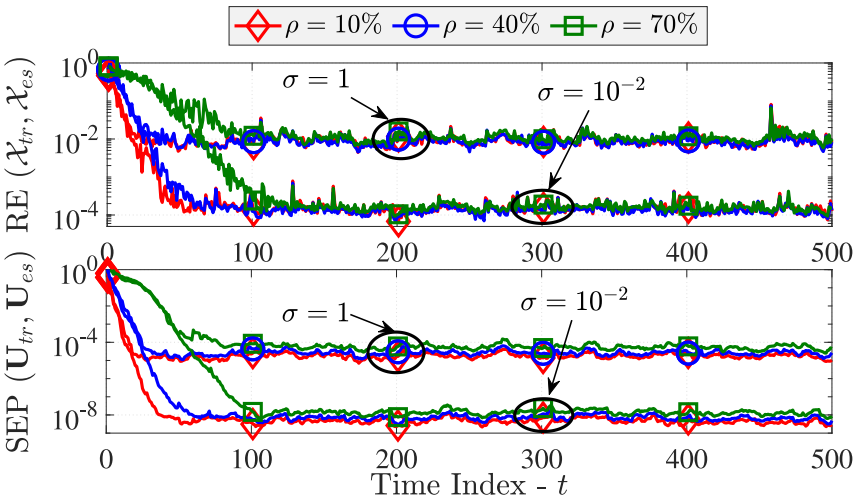

Figure 8. Performance of ATD versus the missing density ρ and the noise level σ

The underlying fourth-order streaming tensor is of size 20 × 20 × 20 × 500 and Tucker rank $\mathbf{r}_{TD}$ = [3, 3, 3, 3]. Two noise levels (σ = 1 and σ = 10^{-2}) and three missing densities (ρ = 10%, 40%, and 70%) are studied.

The missing density ρ has an influence only on the convergence rate of ACP. Specifically, the lower the missing density ρ is, the faster ACP converges.

Evaluation of effectiveness and efficiency

To demonstrate the effectiveness and efficiency of our algorithm, we compare the performance of ACP in terms of estimation accuracy and running time with the state-of-the-art ACP algorithms for incomplete tensors, including OLSTEC,[9] CP-PETRELS,[10] and TeCPSGD.[8] For a fair comparison, the parameters of these algorithms were carefully fine-tuned to achieve good performance. Particularly, the forgetting factor λ is set at 0.7 and 0.98, respectively, for OLSTEC and CP-PETRELS. Moreover, OLSTEC and TeCPSGD are also dependent on a penalty parameter, which is set at 10^{-3} in both cases. As CP-PETRELS is sensitive to initialization, we use a set of L training data samples and apply the batch CP method to obtain a good starting point for it. Here, we consider two versions of CP-PETRELS with L = 100 and L = 20, denoted by CP-PETRELS-v1 and CP-PETRELS-v2, respectively. We use random initialization for ACP, OLSTEC, and TeCPSGD.

Since these algorithms are capable of tracking third-order tensors only, we use synthetic streaming tensors of size $N \times N \times 1,000$ in this task. The noise level is fixed at $\sigma = 10^{-3}$, while we set the rotation angle at $\alpha = \pi/360$. The performance of these algorithms is evaluated on a small tensor 20× 20× 1,000 and a big tensor 200 × 200 × 1,000. Results are shown in Figures 6 and 7. We can see that ACP and OLSTEC provide the best tracking performance in terms of estimation accuracy. When the number of missing data is small (e.g., ρ = 10%), ACP converges faster than OLSTEC. TeCPSGD's convergence rate is much slower than those of ACP and OLSTEC, and its estimation accuracy is also worse. Thanks to the second-order estimation, CP-PETRELS has the fastest convergence rate, but its REs are higher than those of the others. With respect to the running time, ACP is several times faster than OLSTEC, especially in big tensor tests. TeCPSGD is also a fast adaptive algorithm, thanks to SGD, while the running time of CP-PETRELS is high. For additional performance comparison results, we refer the reader to the supplemental information (i.e., Figures S2–S4).

Performance of ATD

Experimental setup

Follow the setup above, the incomplete tensor slice $\mathcal{X}_t$ at time t is generated randomly using the following model:

$$\mathcal{X}_t = \mathcal{P}_t \circledast \left(\mathcal{G}_t \prod_{n=1}^{N} \times_n \mathbf{U}_t^{(n)} \times \mathbf{u}_t^{\top} + \sigma \mathcal{N}_t \right), \qquad \text{(Equation 32)}$$

where the loading factor $\mathbf{U}_t^{(n)}$ and the core tensor $\mathcal{G}_t$ are updated by the following rules:

$$\mathbf{U}_t^{(n)} = \mathbf{U}_{t-1}^{(n)} + \varepsilon \mathbf{N}_t^{(n)} \text{ and } \mathcal{G}_t = \mathcal{G}_{t-1} + \varepsilon \mathcal{V}_t, \qquad \text{(Equation 33)}$$

where $\mathbf{U}_0^{(n)}, \mathbf{N}_t^{(n)} \in \mathbb{R}^{I_n \times r_n}$ and $\mathcal{V}_t \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_{(N+1)}}$ are the Gaussian noises whose entries are distributed i.i.d. from $\mathcal{N}(0,1)$, and the time-varying factor ε is to control their variation.

Table 2. Performance of Tucker algorithms on a static fourth-order tensor of size 20×20×20×500 and the noise level $\sigma = 10^{-2}$

Missing	ρ = 50 %			ρ = 70 %			
metric	RE($\mathcal{X}$)	SEP($\boldsymbol{U}$)	Time (s)	RE($\mathcal{X}$)	SEP($\boldsymbol{U}$)	Time (s)	Rank
iHOOI	3.0e−4	4.2e−8	88.1	8.1e−4	4.7e−7	345.3	[3, 3, 3, 3]
ALSaS	3.1e−4	4.3e−8	109.9	7.8e−4	4.9e−7	539.5	
WTucker	2.1e−4	2.4e−8	209.1	3.5e−4	1.3e−7	597.4	
ATD	6.4e−5	7.6e−9	2.5	1.8e−4	1.4e−8	5.7	
iHOOI	9.1e−2	5.1e−4	192.9	3.5e−1	1.3e−2	571.5	[10, 10, 10, 10]
ALSaS	1.0e−4	4.2e−9	719.1	8.3e−4	3.4e−8	3754.6	
WTucker	3.7e−5	2.8e−10	241.2	5.0e−5	3.3e−10	631.7	
ATD	1.7e−5	6.8e−11	21.7	3.2e−5	2.5e−8	58.2	

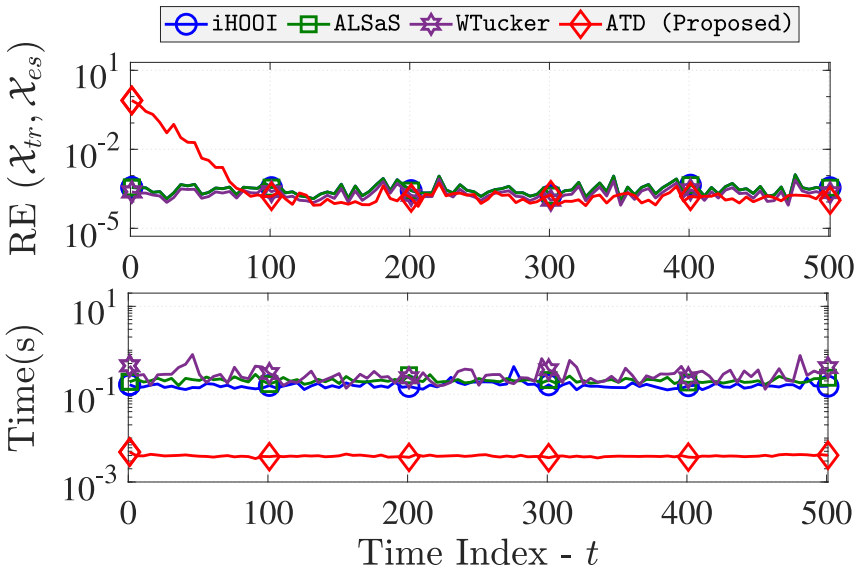

Figure 9. Performance of Tucker algorithms

The underlying fourth-order streaming tensor is of size 20 × 20 × 20 × 500 and Tucker rank $\mathbf{r}_{TD}$ = [3, 3, 3, 3]. The 50% data are supposed to be missing at random and the noise level is fixed at $\sigma = 10^{-2}$. ATD is shown to be the fastest Tucker algorithm, much faster than iHOOI, ALSaS, and WTucker. See Figure S5 in the supplemental information for further performance comparison among these Tucker algorithms.

In addition to the RE metric, we also use the subspace estimation performance (SEP)[71] metric to evaluate the subspace estimation accuracy, which is defined by:

$$\mathrm{SEP}(\mathbf{U}_{tr}, \mathbf{U}_{es}) = \frac{\mathrm{tr}(\mathbf{U}_{es}^{\#}(\mathbf{I} - \mathbf{U}_{tr}\mathbf{U}_{tr}^{\#})\mathbf{U}_{es})}{\mathrm{tr}(\mathbf{U}_{es}^{\#}(\mathbf{U}_{tr}\mathbf{U}_{tr}^{\#})\mathbf{U}_{es})}, \qquad \text{(Equation 34)}$$

where $\mathbf{U}_{tr}$ (resp. $\mathbf{U}_{es}$) refers to the true loading factor (resp. estimated factor). The lower the value of SEP is, the more accurate the algorithm is.

Robustness of ATD

To demonstrate the robustness of ATD against data corruption, we change the number of missing entries in the measurement by varying the value of ρ and evaluate its performance on different noise levels. We also compare ATD with three well-known batch Tucker algorithms for tensor completion, including iHOOI,[72] ALSaS,[72] and WTucker.[73] These algorithms are iterative, so their procedure will be stopped when the accuracy tolerance *tol* or the maximum iteration $\mathrm{ITER}_{\max}$ is achieved. Particularly, the tolerance *tol* is used to bound the fitting/estimation error of iterative algorithms, which is defined as $\|\mathcal{P}\circledast(\mathcal{X} - \mathcal{G}\prod_{n=1}^{N}\times_n\mathbf{U}^{(n)})\|_F / \|\mathcal{P}\circledast\mathcal{X}\|_F$. As a convergence guarantee, we fix the value of *tol* at 10^{-4}, while the value of $\mathrm{ITER}_{\max}$ is set at 500, 500, and 100, respectively, for iHOOI, ALSaS, and WTucker. For ATD, the forgetting factor λ is fixed at 0.7 in the following experiments.

In this task, we use a static tensor of size 20×20×20×500 (i.e., the time-varying factor $\varepsilon = 0$), whose core is generated at random from a Gaussian distribution of zero-mean and unit variance. Under the Tucker model with $\mathbf{r}_{TD} = [3, 3, 3, 3]$, the performance of ATD on the tensor is illustrated in Figure 8. Results show that ATD can successfully track the multilinear low-rank model in all test cases. Similar to ACP, the missing density ρ has influence only on the convergence rate of ATD, i.e., the higher the value of ρ is, the more slowly ATD converges. The performance comparison among Tucker algorithms is reported in Table 2 and shown in Figures 9 and S5 (in the supplemental information). We can see that ATD achieves an estimation

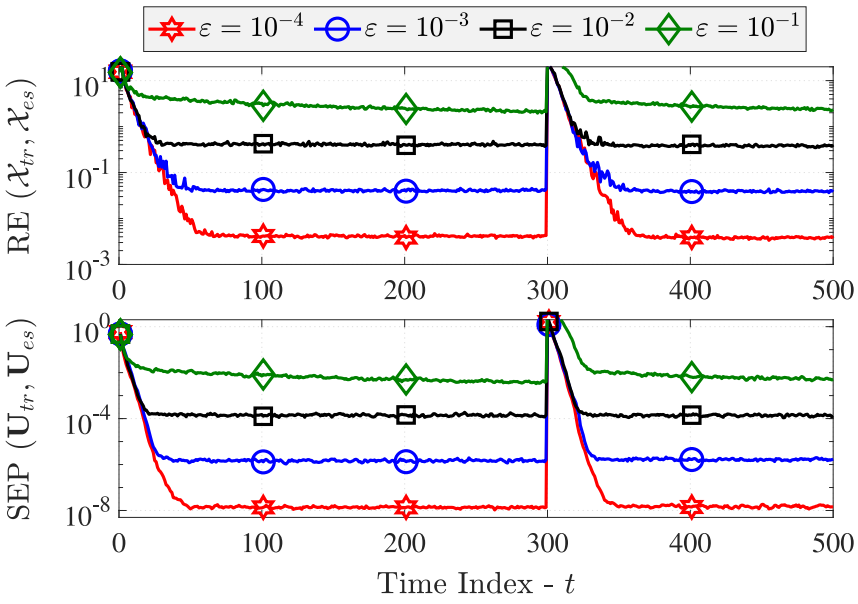

Figure 10. Effect of the time-varying factor ε on the performance of ATD

The underlying fourth-order streaming tensor is of size 20 × 20 × 20 × 500 and Tucker rank [3, 3, 3, 3]. The 90% entries are observed, the noise is $\sigma = 10^{-2}$, and an abrupt change is created at t = 300. The time-varying factor is varied among the range $[10^{-4}, 10^{-1}]$.

Table 3. Performance of adaptive tensor decompositions on video datasets

Methods			TeCPSGD		OLSTEC		CP-PETRELS		ACP		ATD	
Dataset	Size	Missing	RE($\mathcal{X}$)	Time (s)	RE($\mathcal{X}$)	Time (s)	RE($\mathcal{X}$)	Time (s)	RE($\mathcal{X}$)	Time (s)	RE($\mathcal{X}$)	Time (s)
Highway	320 × 240 × 1,700	10%	0.2057	36.582	0.1693	132.02	0.9250	451.41	0.2178	14.437	0.1484	36.587
		50%	0.2111	35.252	0.1709	95.188	0.9346	273.98	0.2251	13.295	0.1526	33.269
		90%	0.2256	27.103	0.1849	54.246	0.9224	107.79	0.2725	13.017	0.1964	26.996
Hall	320 × 240 × 1,700	10%	0.1456	15.060	0.1247	83.789	0.9819	339.10	0.1457	11.852	0.1006	36.293
		50%	0.1450	14.916	0.1260	74.869	0.9269	188.15	0.1602	11.808	0.1045	31.576
		90%	0.1614	12.532	0.1497	47.235	0.9281	71.576	0.2341	11.897	0.1426	25.047
Lobby	320 × 240 × 1,700	10%	0.1324	5.672	0.1213	29.490	0.9161	107.44	0.1258	4.613	0.0868	14.590
		50%	0.1452	4.920	0.1228	21.940	0.8596	61.051	0.1881	4.711	0.0884	10.630
		90%	0.1733	4.022	0.1530	14.701	0.9736	22.150	0.2602	3.811	0.1333	9.245
Park	320 × 240 × 1,700	10%	0.1057	10.303	0.0905	49.213	0.9945	186.28	0.1270	6.458	0.0686	16.157
		50%	0.1246	9.940	0.0916	33.660	0.9892	127.30	0.1441	5.825	0.0759	14.052
		90%	0.1369	8.497	0.1006	22.031	0.9627	50.435	0.2001	5.179	0.1122	10.966

accuracy similar to those of iHOOI, ALSaS, and WTucker. As ATD starts from a random point, it needs a certain number of observations to converge during the early stage of tracking. By contrast, iHOOI, ALSaS, and WTucker process data samples in a group or batch. Therefore, their REs often remain almost the same when the time-varying factor and the noise level are fixed over time. Experimental results also indicate that ATD is the fastest algorithm, much faster than the state-of-the-art Tucker algorithms, thanks to the adaptive scheme and the randomized technique. For example, when dealing with the case of 50% missing observations and $\mathbf{r}_{\text{TD}} = [3, 3, 3, 3]$, the running time of ATD is only 2.51 s, which is 35 times faster than the second-fastest algorithm, iHOOI.

Tracking ability in dynamic environments

We continue to investigate the tracking ability of ATD in non-stationary and time-varying environments by changing the time-varying factor ε in the range $[10^{-4}, 10^{-1}]$. We use the same tensor dimensions as in the previous task. We also create a significant subspace change at time $t = 300$ to see how fast ATD can converge. Figure 10 shows the convergence behavior of ATD versus the time-varying factor ε. We can see that the convergence rate of ATD is not affected by ε but only its estimation error. The performance of the orthogonal ATD (ATD-O) is illustrated in Figure S6 (in the supplemental information). Both ATD and ATD-O converge toward the same steady-state error bound. However, the convergence rate of ATD-O is slightly better than that of ATD.

Real data

To demonstrate the effectiveness of our algorithms on real datasets, we consider two related applications: video completion and multichannel electroencephalography (EEG) analysis.

Video completion

In this task, four real video surveillance sequences are used, including Highway, Hall, Lobby, and Park (video sequences: http://jacarini.dinf.usherbrooke.ca/). Specifically, Highway contains 1,700 frames of size 320×240 pixels showing the two-lane traffic surveillance of vehicles on a highway. Hall, which has 3,584 frames of size 174×144 pixels, shows an airport hall with many people coming in and out. Lobby consists of 1,546 frames of size 128×160 pixels captured in an office lobby, where the background was changed by switching lights on and off. Park includes 600 frames of size 288×352 pixels shot by a thermal camera. To create missing pixels in videos, we use a set of binary masks of the same size as the video frames, and their entries are derived from a Bernoulli model with probability ρ (i.e., ρ indicates the missing density). Then, we apply the Hadamard product to multiply these masks with all video frames (each mask corresponds to each frame) to create zero entries, as in the previous tasks on synthetic data.

We compare the proposed algorithms with OLSTEC,[9] TeCPSGD,[8] and CP-PETRELS.[10] We set the value of λ at 0.7 and 0.999 respectively for OLSTEC and CP-PETRELS. OLSTEC and TeCPSGD also depend on the regularization parameter μ, whose value is fixed at 0.1. The performance of these algorithms is shown statistically in Table 3 and graphically in Figure 11. We can see that ATD outperforms adaptive CP algorithms in almost all tests. ACP also provides reasonable estimation accuracy on these data compared with others. CP-PETRELS seems to fail to track the video background and thus fails to recover missing

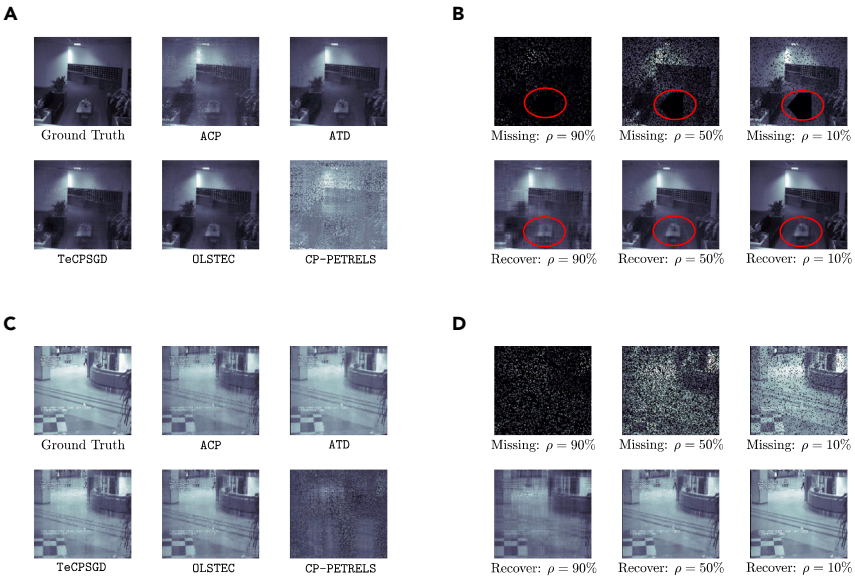

Figure 11. Performance of adaptive tensor completion algorithms on video datasets

(A) and (C) respectively illustrate the completion performance of adaptive tensor algorithms on Lobby and Hall video data in the presence of 50% missing pixels. (B) and (D) respectively illustrate the performance of ATD on Lobby and Hall video data with different missing densities.

data. With respect to the running time, experimental results indicate that ACP is the fastest adaptive CP decomposition. We refer the reader to the supplemental information for more experimental results (see Figures S7–S10).

Multichannel EEG analysis

We follow the experimental framework in Acar et al.[74] and Linh-Trung et al.[75] to illustrate the use of ACP for analyzing multichannel EEG signals. In this task, we use a public EEG dataset collected on 14 subjects during the stimulation of hands (EEG data: http://www.erpwavelab.org/index.htm). The EEG signals are recorded using a system of 64 channels (electrodes) and we have 28 measurements per subject in total.

We construct a third-order EEG tensor of measurement × channel × time − frequency by taking a continuous wavelet transform of each EEG channel, as in our past works.[16,76] Note that the resulting time-frequency representations are reshaped into vectors of length 4,392 and are hence streamed. In a nutshell, we have the EEG tensor whose size is $28 \times 64 \times 4,392$ and its rank is set at 3 as provided in Acar et al.[74] and Linh-Trung et al.[75]. At each time, data of 20 (and 40) channels are assumed to be discarded randomly for our missing observation purpose.

We evaluate the performance of ACP by comparing it with the adaptive NL-PETRELS algorithm in Linh-Trung et al.[75] and the batch CP-WOPT algorithm in Acar et al.[74] To have a good initialization for NL-PETRELS, the first 1,500 slices are used to construct the training tensor. Also, the forgetting factor λ is set at 0.999. By contrast, ACP is not as sensitive to initialization conditions, so it is initialized at random. We consider results obtained by using the batch algorithm as our ground truth.

Under the CP model with $r_{CP} = 3$, taking the tensor decomposition of the EEG tensor results in three loading factors, $\mathbf{A} \in \mathbb{R}^{28\times3}$, $\mathbf{B} \in \mathbb{R}^{64\times3}$, and $\mathbf{C} \in \mathbb{R}^{4392\times3}$, corresponding to, respectively, the measurement, channel, and time-frequency modes. Figures 12 and 13 illustrate the estimation of **A**, **B,** and **C** using CP-WOPT, NL-PETRELS, and ACP. In particular, we use bar plots and 3D head plots to represent the column vectors of **A** and **B**, while the time-frequency representations corresponding to the columns **C** are plotted as matrices. We can see from Figure 12 that both adaptive algorithms are capable

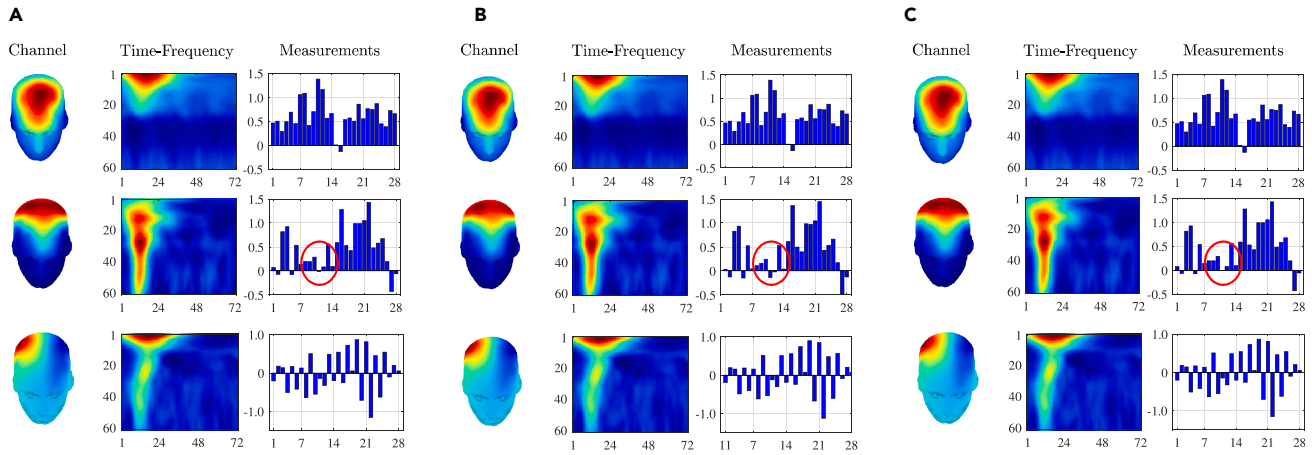

Figure 12. Waveform-preserving character of ACP on the EEG tensor with 20 missing channels

(A), (B), and (C) show the estimation performance of CP-WOPT (batch), NL-PETRELS (adaptive), and ACP (adaptive), respectively. Since the ground truth is not available, the estimation of CP-WOPT is regarded as benchmark. It is clear that ACP provides results similar to those of CP-WOPT and a slightly better estimation than NL-PETRELS. Red circles are to highlight the difference among the three results.

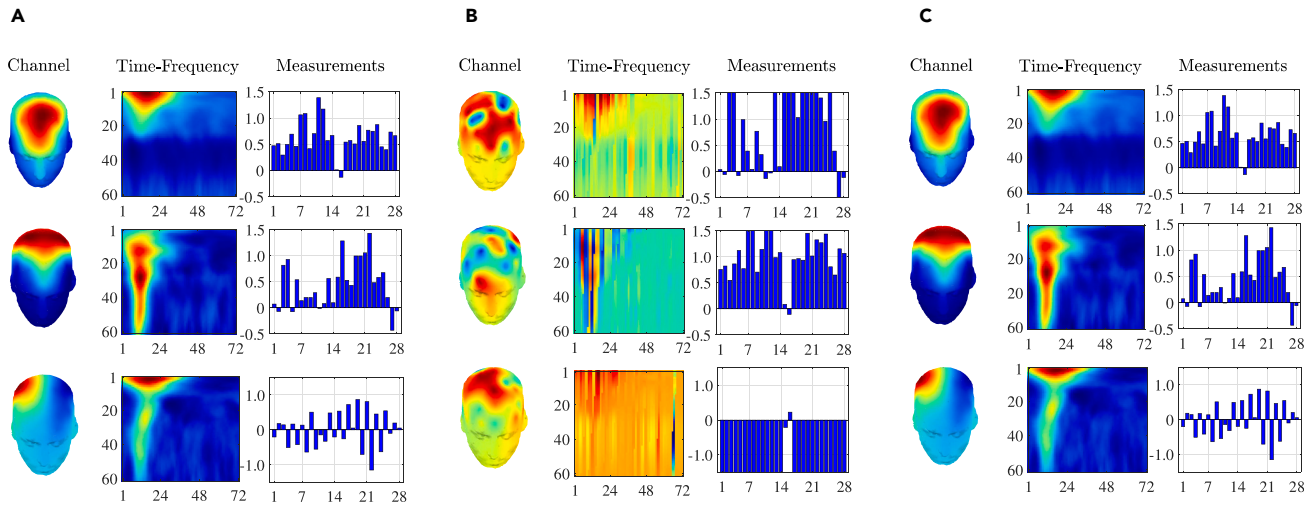

Figure 13. Waveform-preserving character of ACP on the EEG tensor with 40 missing channels

(A), (B), and (C) illustrate the estimation performance of CP-WOPT (batch), NL-PETRELS (adaptive), and ACP (adaptive), respectively. NL-PETRELS fails to recover the EEG loading factors, while ACP still works well compared with CP-WOPT.

of tracking three EEG loading factors. Our ACP provides a slightly better estimation compared with that of CP-WOPT. In the presence of highly incomplete observations (e.g., 40 channels are missing), NL-PETRELS fails to estimate the EEG loading factors, while our ACP algorithm still works well, as shown in Figure 13. The running time of the three algorithms is reported in Table 4, which indicates that ACP is much faster than NL-PETRELS and CP-WOPT.

DISCUSSION

In this paper, we have proposed two new low-complexity algorithms, namely ACP and ATD, for the adaptive decomposition of higher-order incomplete and streaming tensors. Developed based on CP decomposition, ACP estimates a multilinear LRA of streaming tensors from noisy and high-dimensional data with high accuracy, even when the decomposition model is slowly time-varying. In parallel, developed based on Tucker decomposition, ATD is a fast randomized tracker, able to recover missing entries from highly incomplete observations. Due to the advantages of stochastic approximation and uniform sampling, ATD is one of the fastest Tucker algorithms, much faster than batch algorithms, while providing good estimation accuracy. Our experimental results indicate that ATD outperforms state-of-the-art adaptive CP algorithms, including our ACP algorithm, in video completion. Thanks to the Tucker format, ATD is more stable than ACP, but with more parameters, and is thus slower than ACP.

Table 4. Multichannel EEG analysis: Running times of tensor algorithms

Missing data	20 channels	40 channels	60 channels
CP-WOPT	87.51 (s)	90.15 (s)	95.29 (s)
NL-PETRELS	59.72 (s)	39.83 (s)	37.12 (s)
ACP	1.84 (s)	1.75 (s)	1.42 (s)

EXPERIMENTAL PROCEDURES

Resource availability

Lead contact

Further information and requests for resources should be directed to and will be fulfilled by the lead contact, Dr. Nguyen Linh Trung (linhtrung@vnu.edu.vn).

Materials availability

No new materials were generated in this study.

Data and code availability

All original code has been deposited at https://github.com/thanhtbt/tensor_tracking and archived in Figshare under https://doi.org/10.6084/m9.figshare.22276105. Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.patter.2023.100759.

ACKNOWLEDGMENTS

This research was conducted under research project QG.22.62 "Multidimensional data analysis and application to Alzheimer's disease diagnosis" of Vietnam National University, Hanoi.

AUTHOR CONTRIBUTIONS

Conceptualization, L.T.T.; methodology, L.T.T. and K.A.-M.; formal analysis, L.T.T.; validation, K.A.-M., N.L.T., and A.H.; software, L.T.T.; writing – original draft, L.T.T.; writing – review & editing, L.T.T., K.A.-M., N.L.T., and A.H.; visualization, L.T.T.; supervision, K.A.-M., N.L.T., and A.H.; funding acquisition, N.L.T.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: October 5, 2022
Revised: December 28, 2022
Accepted: May 2, 2023
Published: June 9, 2023

REFERENCES

1. Chen, M.H., Li, C.T., Lin, W.C., Wei, H.T., Chang, W.H., Chen, T.J., Pan, T.L., Su, T.P., and Bai, Y.M. (2014). Big data: a survey. Schizophr. Res. *159*, 171–175. https://doi.org/10.1007/s11036-013-0489-0.
2. Sidiropoulos, N.D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E.E., and Faloutsos, C. (2017). Tensor decomposition for signal processing and machine learning. IEEE Trans. Signal Process. *65*, 3551–3582. https://doi.org/10.1109/TSP.2017.2690524.
3. Kolda, T.G., and Bader, B.W. (2009). Tensor decompositions and applications. SIAM Rev. *51*, 455–500. https://doi.org/10.1137/07070111X.
4. De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000). A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. *21*, 1253–1278. https://doi.org/10.1137/S0895479896305696.
5. Harshman, R.A. (1970). Foundations of the PARAFAC procedure: models and conditions for an explanatory multimodal factor analysis. UCLA Work. Pap. Phonetics *16*, 1–84.
6. Nion, D., and Sidiropoulos, N.D. (2009). Adaptive algorithms to track the PARAFAC decomposition of a third-order tensor. IEEE Trans. Signal Process. *57*, 2299–2310. https://doi.org/10.1109/TSP.2009.2016885.
7. Thanh, L.T., Abed-Meraim, K., Trung, N.L., and Hafiane, A. (2022). A contemporary and comprehensive survey on streaming tensor decomposition. IEEE Trans. Knowl. Data Eng. 1–20. https://doi.org/10.1109/TKDE.2022.3230874.
8. Mardani, M., Mateos, G., and Giannakis, G.B. (2015). Subspace learning and imputation for streaming matrices and tensors. IEEE Trans. Signal Process. *63*, 2663–2677. https://doi.org/10.1109/TSP.2015.2417491.
9. Kasai, H. (2019). Fast online low-rank tensor subspace tracking by CP decomposition using recursive least squares from incomplete observations. Neurocomputing *347*, 177–190. https://doi.org/10.1016/j.neucom.2018.11.030.
10. Minh-Chinh, T., Nguyen, V.D., Linh-Trung, N., and Abed-Meraim, K. (2016). Adaptive PARAFAC decomposition for third-order tensor completion. IEEE Int. Conf. Consum. Electron. 297–301. https://doi.org/10.1109/CCE.2016.7562652.
11. Recht, B., Fazel, M., and Parrilo, P.A. (2010). Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. SIAM Rev. *52*, 471–501. https://doi.org/10.1137/070697835.
12. Hu, Z., Nie, F., Wang, R., and Li, X. (2021). Low rank regularization: a review. Neural Network. *136*, 218–232. https://doi.org/10.1016/j.neunet.2020.09.021.
13. Ahn, D., Kim, S., and Kang, U. (2021). Accurate online tensor factorization for temporal tensor streams with missing values. ACM Int. Conf. Inf. Knowl. Manag. 2822–2826. https://doi.org/10.1145/3459637.3482048.
14. Zhang, Z., and Hawkins, C. (2018). Variational Bayesian inference for robust streaming tensor factorization and completion. IEEE Int. Conf. Data Min. 1446–1451. https://doi.org/10.1109/ICDM.2018.00200.
15. Lee, D., and Shin, K. (2021). Robust factorization of real-world tensor streams with patterns, missing values, and outliers. IEEE Int. Conf. Data Eng. 840–851. https://doi.org/10.1109/ICDE51399.2021.00078.
16. Thanh, L.T., Abed-Meraim, K., Trung, N.L., and Hafiane, A. (2022). Robust tensor tracking with missing data and outliers: novel adaptive CP decomposition and convergence analysis. IEEE Trans. Signal Process. *70*, 4305–4320. https://doi.org/10.1109/TSP.2022.3201640.
17. Zhou, S., Vinh, N.X., Bailey, J., Jia, Y., and Davidson, I. (2016). Accelerating online CP decompositions for higher order tensors. In ACM Int. Conf. Knowl. Discover. Data Min., pp. 1375–1384. https://doi.org/10.1145/2939672.2939763.
18. Smith, S.W., Huang, K., Sidiropoulos, N.D., and Karypis, G. (2018). Streaming tensor factorization for infinite data sources. Proceedings of SIAM Int. Conf. Data Min. 81–89. https://doi.org/10.1137/1.9781611975321.10.
19. Thanh, L.T., Abed-Meraim, K., Linh-Trung, N., and Hafiane, A. (2021). A fast randomized adaptive CP decomposition for streaming tensors. In IEEE Int. Conf. Acoust. Speech Signal Process., pp. 2910–2914. https://doi.org/10.1109/ICASSP39728.2021.9413554.
20. Zeng, C., and Ng, M.K. (2021). Incremental CP tensor decomposition by alternating minimization method. SIAM J. Matrix Anal. Appl. *42*, 832–858. https://doi.org/10.1137/20M1319097.
21. Lyu, H., Wang, J., Wu, W., Duong, V., Zhang, X., Dye, T.D., and Luo, J. (2022). Online nonnegative CP-dictionary learning for Markovian data. J. Mach. Learn. Res. *23*, 1–50.
22. Kasai, H., and Mishra, B. (2016). Low-rank tensor completion: a Riemannian manifold preconditioning approach. Int. Conf. Mach. Learn. 1012–1021.
23. Fang, S., Kirby, R.M., and Zhe, S. (2021). Bayesian streaming sparse Tucker decomposition. In Conf. Uncertain. Artif. Intell., pp. 558–567.
24. Zdunek, R., and Fonał, K. (2022). Incremental nonnegative Tucker decomposition with block-coordinate descent and recursive approaches. Symmetry *14*, 113. https://doi.org/10.3390/sym14010113.
25. Jang, J.-G., and Kang, U. (2023). Static and streaming Tucker decomposition for dense tensors. ACM Trans. Knowl. Discov. Data *17*, 1–34. https://doi.org/10.1145/3568682.
26. Sun, J., Lin, Z., Feng, J., Li, Y., and Shen, B. (2008). Incremental tensor analysis: theory and applications. ACM Trans. Knowl. Discov. Data *2*, 1–37. https://doi.org/10.1145/1409620.1409621.
27. Traoré, A., Berar, M., and Rakotomamonjy, A. (2019). Online multimodal dictionary learning. Neurocomputing *368*, 163–179. https://doi.org/10.1016/j.neucom.2019.08.053.
28. Li, P., Feng, J., Jin, X., Zhang, L., Xu, X., and Yan, S. (2019). Online robust low-rank tensor modeling for streaming data analysis. IEEE Transact. Neural Networks Learn. Syst. *30*, 1061–1075. https://doi.org/10.1109/TNNLS.2018.2860964.
29. Chachlakis, D.G., Dhanaraj, M., Prater-Bennette, A., and Markopoulos, P.P. (2021). Dynamic L1-norm Tucker tensor decomposition. IEEE J. Sel. Top. Signal Process. *15*, 587–602. https://doi.org/10.1109/JSTSP.2021.3058846.
30. Thanh, L.T., Duy, T.T., Abed-Meraim, K., Linh Trung, N., and Hafiane, A. (2022). Robust online Tucker dictionary learning from multidimensional data streams. IEEE Asia-Pacific Signal Inf. Process. Assoc. Annu. Conf. 1815–1820. https://doi.org/10.23919/APSIPAASC55919.2022.9980029.
31. Gilman, K., Tarzanagh, D.A., and Balzano, L. (2022). Grassmannian optimization for online tensor completion and tracking with the t-SVD. IEEE Trans. Signal Process. *70*, 2152–2167. https://doi.org/10.1109/TSP.2022.3164837.
32. Martin, C.D., Shafer, R., and LaRue, B. (2013). An order-p tensor factorization with applications in imaging. SIAM J. Sci. Comput. *35*, A474–A490. https://doi.org/10.1137/110841229.
33. Zhang, Z., Wu, Y., Yu, F., Niu, C., Du, Z., Chen, Y., and Du, J. (2017). Exact tensor completion using t-SVD. IEEE Trans. Signal Process. *65*, 1511–1526. https://doi.org/10.1109/TSP.2016.2639466.
34. Jiang, F., Liu, X.-Y., Lu, H., and Shen, R. (2018). Efficient multi-dimensional tensor sparse coding using t-linear combination. AAAI Conf. Artif. Intell. *32*, 3326–3333. https://doi.org/10.1609/aaai.v32i1.11620.
35. De Lathauwer, L. (2008). Decompositions of a higher-order tensor in block terms – Part II: definitions and uniqueness. SIAM J. Matrix Anal. Appl. *30*, 1033–1066. https://doi.org/10.1137/070690729.
36. Gujral, E., and Papalexakis, E.E. (2020). OnlineBTD: streaming algorithms to track the block term decomposition of large tensors. IEEE Int. Conf. Data Sci. Adv. Anal. 168–177. https://doi.org/10.1109/DSAA49011.2020.00029.
37. Rontogiannis, A.A., Giampouras, P.V., and Kofidis, E. (2021). Online rank-revealing block-term tensor decomposition. Asilomar Conf. Signals Syst. Comput. 1678–1682. https://doi.org/10.1109/ICASSP39728.2021.9415104.
38. Thanh, L.T., Abed-Meraim, K., Trung, N.L., and Boyer, R. (2021). Adaptive algorithms for tracking tensor-train decomposition of streaming tensors. Eur.

Signal Process. Conf. 995–999. https://doi.org/10.23919/Eusipco47968.2020.9287780.

39. Thanh, L.T., Abed-Meraim, K., Trung, N.L., and Hafiane, A. (2022). Robust tensor tracking with missing data under tensor-train format. Eur. Signal. Process. Conf. 832–836. https://doi.org/10.23919/EUSIPCO55093.2022.9909702.
40. Yu, J., Zou, T., and Zhou, G. (2022). Online subspace learning and imputation by tensor-ring decomposition. Neural Network. *153*, 314–324. https://doi.org/10.1016/j.neunet.2022.05.023.
41. Song, Q., Huang, X., Ge, H., Caverlee, J., and Hu, X. (2017). Multi-aspect streaming tensor completion. ACM Int. Conf. Knowl. Disc. Data Min. 435–443. https://doi.org/10.1145/3097983.3098007.
42. Najafi, M., He, L., and Yu, P.S. (2019). Outlier-robust multi-aspect streaming tensor completion and factorization. Int. Joint Conf. Artificial Intell. 3187–3194. https://doi.org/10.24963/ijcai.2019/442.
43. Nimishakavi, M., Mishra, B., Gupta, M., and Talukdar, P. (2018). Inductive framework for multi-aspect streaming tensor completion with side information. ACM Int. Conf. Inf. Knowl. Manag. 307–316. https://doi.org/10.1145/3269206.3271713.
44. Mahoney, M.W. (2011). Randomized algorithms for matrices and data. Found. Trends Mach. Learn. *3*, 123–224.
45. Wang, Y., Tung, H.-Y., Smola, A.J., and Anandkumar, A. (2015). Fast and guaranteed tensor decomposition via sketching. Adv. Neural Inf. Process. Syst. 991–999.
46. Song, Z., Woodruff, D., and Zhang, H. (2016). Sublinear time orthogonal tensor decomposition. Adv. Neural Inf. Process. Syst. 793–801.
47. Battaglino, C., Ballard, G., and Kolda, T.G. (2018). A practical randomized CP tensor decomposition. SIAM J. Matrix Anal. Appl. *39*, 876–901. https://doi.org/10.1137/17M1112303.
48. Malik, O.A., and Becker, S. (2018). Low-rank Tucker decomposition of large tensors using Tensorsketch. Adv. Neural Inf. Process. Syst. 10096–10106.
49. Che, M., and Wei, Y. (2019). Randomized algorithms for the approximations of Tucker and the tensor train decompositions. Adv. Comput. Math. *45*, 395–428. https://doi.org/10.1007/s10444-018-9622-8.
50. Che, M., Wei, Y., and Yan, H. (2021). Randomized algorithms for the low multilinear rank approximations of tensors. J. Comput. Appl. Math. *390*, 113380. https://doi.org/10.1016/j.cam.2020.113380.
51. Minster, R., Saibaba, A.K., and Kilmer, M.E. (2020). Randomized algorithms for low-rank tensor decompositions in the Tucker format. SIAM J. Math. Data Sci. *2*, 189–215. https://doi.org/10.1137/19M1261043.
52. Ahmadi-Asl, S., Abukhovich, S., Asante-Mensah, M.G., Cichocki, A., Phan, A.H., Tanaka, T., and Oseledets, I. (2021). Randomized algorithms for computation of Tucker decomposition and higher-order SVD (HOSVD). IEEE Access *9*, 28684–28706. https://doi.org/10.1109/ACCESS.2021.3058103.
53. Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D.P. (2016). Tensor networks for dimensionality reduction and large-scale optimization: Part I low-rank tensor decompositions. FNT. in Machine Learning *9*, 249–429.
54. De Silva, V., and Lim, L.-H. (2008). Tensor rank and the ill-posedness of the best low-rank approximation problem. SIAM J. Matrix Anal. Appl. *30*, 1084–1127. https://doi.org/10.1137/06066518X.
55. Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2010). Online learning for matrix factorization and sparse coding. J. Mach. Learn. Res. *11*, 19–60.
56. Thanh, L.T., Dung, N.V., Trung, N.L., and Abed-Meraim, K. (2021). Robust subspace tracking with missing data and outliers: novel algorithm with convergence guarantee. IEEE Trans. Signal Process. *69*, 2070–2085. https://doi.org/10.1109/TSP.2021.3066795.
57. Thanh, L.T., Abed-Meraim, K., Hafiane, A., and Trung, N.L. (2022). Sparse subspace tracking in high dimensions. In IEEE Int. Conf. Acoust. Speech Signal Process., pp. 5892–5896. https://doi.org/10.1109/ICASSP43922.2022.9746546.
58. Chatterjee, S. (2020). A deterministic theory of low rank matrix completion. IEEE Trans. Inf. Theor. *66*, 8046–8055. https://doi.org/10.1109/TIT.2020.3019569.
59. Candes, E.J., and Tao, T. (2010). The power of convex relaxation: near-optimal matrix completion. IEEE Trans. Inf. Theor. *56*, 2053–2080. https://doi.org/10.1109/TIT.2010.2044061.
60. Mairal, J. (2015). Incremental majorization-minimization optimization with application to large-scale machine learning. SIAM J. Optim. *25* (2), 829–855. https://doi.org/10.1137/140957639.
61. Raskutti, G., and Mahoney, M.W. (2016). A statistical perspective on randomized sketching for ordinary least-squares. J. Mach. Learn. Res. *17*, 7508–7538.
62. Farrar, D.E., and Glauber, R.R. (1967). Multicollinearity in regression analysis: the problem revisited. Rev. Econ. Stat. *49*, 92–107. https://doi.org/10.2307/1937887.
63. Allen, M.P. (1997). The problem of multicollinearity. Understanding Regression Analysis, 176–180. https://doi.org/10.1007/978-0-585-25657-3_37.
64. Tropp, J.A. (2011). Improved analysis of the subsampled randomized Hadamard transform. Adv. Adapt. Data Anal. *03* (01n02), 115–126. https://doi.org/10.1142/S1793536911000787.
65. Balzano, L., Chi, Y., and Lu, Y.M. (2018). Streaming PCA and subspace tracking: the missing data case. Proc. IEEE *106*, 1293–1310. https://doi.org/10.1109/JPROC.2018.2847041.
66. Chi, Y., Eldar, Y.C., and Calderbank, R. (2013). PETRELS: parallel subspace estimation and tracking by recursive least squares from partial observations. IEEE Trans. Signal Process. *61*, 5947–5959. https://doi.org/10.1109/TSP.2013.2282910.
67. Spall, J.C. (2005). Introduction to Stochastic Search and Optimization (John Wiley & Sons).
68. Langville, A.N., and Stewart, W.J. (2004). The Kronecker product and stochastic automata networks. J. Comput. Appl. Math. *167*, 429–447. https://doi.org/10.1016/j.cam.2003.10.010.
69. Diao, H., Jayaram, R., Song, Z., Sun, W., and Woodruff, D. (2019). Optimal sketching for Kronecker product regression and low rank approximation. Adv. Neural Inf. Process. Syst. 4739–4750.
70. Feng, J., Xu, H., and Yan, S. (2013). Online robust PCA via stochastic optimization. Adv. Neural Inf. Process. Syst. 404–412.
71. Linh-Trung, N., Nguyen, V.D., Thameri, M., Minh-Chinh, T., and Abed-Meraim, K. (2018). Low-complexity adaptive algorithms for robust subspace tracking. IEEE J. Sel. Top. Signal Process. *12*, 1197–1212. https://doi.org/10.1109/JSTSP.2018.2876626.
72. Xu, P., Xu, A., Chen, B., Zheng, S., Xu, Y., Li, H., Li, B., Huang, P., Zhang, Y., Ge, Y., and Liu, C. (2017). Fast algorithms for higher-order singular value decomposition from incomplete data. J. Comput. Math. *35*, 395–420. https://doi.org/10.4208/jcm.1608-m2016-0641.
73. Filipović, M., and Jukić, A. (2015). Tucker factorization with missing data with application to low-*n*-rank tensor completion. Multidim. Syst. Signal Process. *26*, 677–692. https://doi.org/10.1007/s11045-013-0269-9.
74. Acar, E., Dunlavy, D.M., Kolda, T.G., and Mørup, M. (2011). Scalable tensor factorizations for incomplete data. Chemometr. Intell. Lab. *106*, 41–56. https://doi.org/10.1016/j.chemolab.2010.08.004.
75. Linh-Trung, N., Minh-Chinh, T., Nguyen, V.D., and Abed-Meraim, K. (2018). A non-linear tensor tracking algorithm for analysis of incomplete multi-channel EEG data. IEEE Int. Symp. Medical Inf. Commun. Tech. 114–119. https://doi.org/10.1109/ISMICT.2018.8573711.
76. Thanh, L.T., Dao, N.T.A., Dung, N.V., Trung, N.L., and Abed-Meraim, K. (2020). Multi-channel EEG epileptic spike detection by a new method of tensor decomposition. J. Neural. Eng. *17*, 016023. https://doi.org/10.1088/1741-2552/ab5247.