On Adaptive PARAFAC Decomposition of Three-Way Tensors

Viet-Dung Nguyen, Karim Abed-Meraim, Nguyen Linh-Trung

On Adaptive PARAFAC Decomposition of Three-Way Tensors

Viet-Dung Nguyen, Karim Abed-Meraim, and Nguyen Linh-Trung

Abstract

We propose a fast adaptive PARAFAC algorithm for a class of third-order tensors which have one dimension growing linearly with time. The proposed method is based on alternating least squares approach used in conjunction with a Newton-type optimization technique. By exploiting the Khatri-Rao product and the reduced rank structure at each time instant, our algorithm has the advantages of both linear complexity and superior convergence performance. A modified version of the algorithm is also proposed to tackle the nonnegative adaptive PARAFAC problem. Parallel implementation issues and algorithm performance are finally investigated. The latter is achieved through simulation comparison with the state-of-the-art algorithms to better highlight the effectiveness of the proposed method.

Index Terms

Fast adaptive PARAFAC, big data, parallel computing, non-negative constraint.

I. Introduction

With the recent advances on sensor and streaming technologies, processing massive volume data or big data with time constraints or even real-time is not only crucial but also challenging [1], in a wide range of applications including MIMO radars [2], biomedical imaging [3], and signal processing [4].

A typical situation in those cases is that the data are acquired in multidimensional form with time as one dimension. Such data can be presented naturally by multi-way arrays, also called

V. D. Nguyen and K. Abed-Meraim are with PRISME laboratory, University of Orléans, 12 rue de Blois BP 6744, Orléans, France. Emails: {viet-dung.nguyen, karim.abed-meraim}@univ-orleans.fr.

N. Linh-Trung is with the University of Engineering and Technology, Vietnam National University Hanoi, 144 Xuan Thuy, Cau Giay, Hanoi, Vietnam. E-mail: linhtrung@vnu.edu.vn.

tensors. Tensor decomposition, thereby, can be used as an important tool to analyze, understand or eventually compress data. Tucker decomposition and PARAFAC (Parallel Factor) decomposition are widely-used and can be considered as generalization of the singular value decomposition (SVD) for multi-way arrays. While the Tucker decomposition lacks of uniqueness and often requires to deal with imposed constraints such as orthogonality, non-negativity, sparseness [5], the PARAFAC decomposition is unique up to scale and permutation indeterminacy under a mild condition. For recent surveys on tensor decomposition, we refer the reader to [4], [6], [7], and references therein for more details. In this paper, the PARAFAC decomposition is the method of interest.

It is observed that applying the batch (offline) PARAFAC decomposition directly to streaming tensors is infeasible because of its high computational cost. Thus, an *adaptive* (incremental) approach is more suitable and provides a good trade-off between quality and efficiency. In contrast to adaptive filtering [8] or subspace tracking [9], [10] which have a long standing history and are well-understood, there exist only few works dedicated for adaptive tensor decomposition. In [2], Nion and Sidiropoulos proposed adaptive models and corresponding adaptive algorithms for third-order tensors having one dimension growing with time, namely recursive least-squares tracking (PARAFAC-RLST) and simultaneous diagonalization tracking (PARAFAC-SDT). In another recent work [3], Mardani, Mateos and Giannakis also proposed an adaptive PARAFAC for incomplete streaming data. In these works, one common basis is that they use first-order methods (i.e., using gradients) to optimize an exponentially-weighted least-squares cost function.

In this paper, we provide an alternative way to tackle the challenging problem of adaptive PARAFAC decomposition. In particular, we propose a new adaptive algorithm for a class of third-order tensors having the dimensions as $I \times J(t) \times K$; the dimension J varies with time. This algorithm is called *second-order optimization based adaptive PARAFAC decomposition* (SOAP). Three main contributions of the proposed algorithm are presented below.

The first contribution of SOAP is that its complexity is *linear* with respect to the tensor rank R, i.e. $O(IKR)$ flops per iteration, as compared to the quadratic cost, i.e. $O(IKR^2)$, required by PARAFAC-RLST and PARARAC-SDT, and its convergence performance is comparable or even superior. To achieve this, we first exploit a second-order stochastic gradient algorithm, in lieu of the first-order gradient algorithm as in [2], to improve the estimation accuracy. In addition, at each step in the algorithm, we force a column of the estimated subspace to have a Kronecker

product structure. As a result, the overall subspace approximately preserves a Khatri-Rao product structure. When possible, we also exploit a rank-1 update to obtain a linear complexity in general.

The second contribution is a proposal of an adaptive version of SOAP in order to handle the constraint of non-negativity. It is known that, imposing a non-negativity constraint on PARAFAC, when applicable, not only improves the physical interpretation [11] but also helps to avoid diverging components [12]. We thereby develop an adaptive non-negative PARAFAC algorithm based on appropriate modifications of SOAP. To the best of our knowledge, adaptive non-negative PARAFAC algorithms have not been addressed in the literature.

The third contribution of SOAP is its readiness to be implemented in the form of parallel/decentralized computing, which has not been considered in [2], [3]. This is especially important when performing online large-scale processing tasks. In particular, SOAP allows us to reduce both the algorithm complexity and the storage usage when we have several computing units (DSP) available in a parallel architecture.

Notations: We follow the notations used in [7]. Calligraphic letters are used for tensors ($\mathcal{A}, \mathcal{B}$, ...). Matrices, vectors (both row and column), and scalars are denoted by boldface uppercase, boldface lowercase, and lowercase respectively; for example $\mathbf{A}, \mathbf{a}$, and a. Element (i, j, k) of a tensor $\mathcal{A} \in \mathbb{C}^{I \times J \times K}$ is symbolized as a_{ijk}, element (i, j) of a matrix $\mathbf{A} \in \mathbb{C}^{I \times J}$ as a_{ij}, and i-th entry of a vector $\mathbf{a} \in \mathbb{C}^{I}$ as a_i. Moreover, $\mathbf{A} \otimes \mathbf{B}$ defines the Kronecker product of $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{A} \odot \mathbf{B}$ the Khatri-Rao (column-wise Kronecker) product, $\mathbf{A} * \mathbf{B}$ the Hadamard product which is the element-wise matrix product, $\mathbf{a} \circ \mathbf{b}$ the outer product of $\mathbf{a}$ and $\mathbf{b}$, and $[\mathbf{A}]^{+} = \max\{0, a_{ij}\}$, for all i, j, is a positive orthant projection of a real-valued matrix $\mathbf{A}$. The transpose, complex conjugate, complex conjugate transpose and pseudo-inverse of $\mathbf{A}$ are denoted by $\mathbf{A}^T$, $\mathbf{A}^*$, $\mathbf{A}^H$ and $\mathbf{A}^{\#}$, respectively. A non-negative $\mathbf{A}$ is denoted by $\mathbf{A} \geqslant 0$, where its entries $a_{ij} \geq 0$ for all i, j.

II. Batch and Adaptive PARAFAC Decompositions

A. Batch PARAFAC

Consider a tensor $\mathcal{X} \in \mathbb{C}^{I \times J \times K}$. The PARAFAC decomposition of $\mathcal{X}$ can be written as follows:

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{1}$$

which is the sum of R rank-one tensors; R is the rank of the tensor. The set of vectors, $\{\mathbf{a}_r\}, \{\mathbf{b}_r\}, \{\mathbf{c}_r\}$ can be grouped into the so-called loading matrices $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_R] \in \mathbb{C}^{I \times R}$, $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_R] \in \mathbb{C}^{J \times R}$, and $\mathbf{C} = [\mathbf{c}_1 \dots \mathbf{c}_R] \in \mathbb{C}^{K \times R}$.

In practice, (1) is only an approximate tensor. In other words, in a noisy environment we should have

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r + \mathcal{N}, \tag{2}$$

where $\mathcal{N}$ is a noise tensor. Thus, given a data tensor $\mathcal{X}$, PARAFAC tries to achieve an R-rank best least squares approximation. Equation (1) also can be formulated in matrix form as

$$\mathbf{X}^{(1)} = (\mathbf{A} \odot \mathbf{C})\mathbf{B}^T, \tag{3}$$

where $\mathbf{X}^{(1)} \in \mathbb{R}^{IK \times J}$ whose entry $\mathbf{X}^{(1)}_{(i-1)K+k,j} = x_{ijk}$. We can write analogous expressions for $\mathbf{X}^{(2)}$ and $\mathbf{X}^{(3)}$ [2]. PARAFAC is generically unique if it satisfies the following condition [13]:

$$2R(R-1) \leq I(I-1)K(K-1), \qquad R \leq J. \tag{4}$$

B. Adaptive PARAFAC

In batch PARAFAC, the dimensions of $\mathcal{X}$ are fixed. In contrast, in adaptive PARAFAC, the dimensions grow with time, that is $\mathcal{X}(t) \in \mathbb{C}^{I(t) \times J(t) \times K(t)}$ in general case. In this paper, we only consider the case where only one dimension grows with time, for example $J(t)$.

For the ease of later comparison, we follow the basic model and assumptions in [2]. In particular, the matrix representation of the adaptive PARAFAC model is presented as

$$\mathbf{X}^{(1)}(t) \simeq \mathbf{H}(t)\mathbf{B}^T(t), \tag{5}$$

where $\mathbf{H}(t) = \mathbf{A}(t) \odot \mathbf{C}(t) \in \mathbb{C}^{IK \times R}$, $\mathbf{B}(t) \in \mathbb{C}^{J(t) \times R}$. Considering two successive times, $t-1$ and t, we have

$$\mathbf{X}^{(1)}(t) = [\mathbf{X}^{(1)}(t-1), \mathbf{x}(t)], \tag{6}$$

where $\mathbf{x}(t) \in \mathbb{C}^{IK}$ is a vectorized representation of new data concatenated to $\mathbf{X}(t-1)$ as shown in Figure 1.

If we further assume that the loading matrices $\mathbf{A}$ and $\mathbf{C}$ follow an unknown but slowly time-varying model (i.e., $\mathbf{A}(t) \simeq \mathbf{A}(t-1)$ and $\mathbf{C}(t) \simeq \mathbf{C}(t-1)$), it leads to $\mathbf{H}(t) \simeq \mathbf{H}(t-1)$. Therefore, we have

$$\mathbf{B}^T(t) \simeq [\mathbf{B}^T(t-1), \mathbf{b}^T(t)]. \tag{7}$$

It means that, at each time, we only need to estimate the *row vector* $\mathbf{b}(t)$ and append it to $\mathbf{B}(t-1)$ instead of updating the whole $\mathbf{B}(t)$.

Two other assumptions also imposed here are: the tensor rank R is known and constant such that at each time when new data is added to the old tensor, the uniqueness of the new tensor fulfills (4). Now we have enough materials to present our algorithm.

III. Algorithm derivation

We consider the following exponentially-weighted least- squares cost function:

$$\Phi(t) = \frac{1}{2}\sum_{\tau=1}^{t} \lambda^{t-\tau}\phi(\tau) \tag{8}$$

where

$$\phi(\tau) = \| \mathbf{x}(\tau) - \mathbf{H}(t)\mathbf{b}^T(\tau) \|_2^2 \tag{9}$$

and $\lambda \in (0, 1]$ is referred to as the forgetting factor. Now, finding the loading matrices of the adaptive PARAFAC model of (5) corresponds to minimizing (8), that is,

$$\min_{\mathbf{H}(t),\mathbf{B}(t)} \Phi(t), \quad \text{s.t. } \mathbf{H}(t) = \mathbf{A}(t) \odot \mathbf{C}(t). \tag{10}$$

This cost function is well-known in adaptive filter theory [8] and can be solved by a recursive least-square method as used in [2]. Here, we provide an alternative way to not only reduce the complexity but also improve the performance of the algorithm.

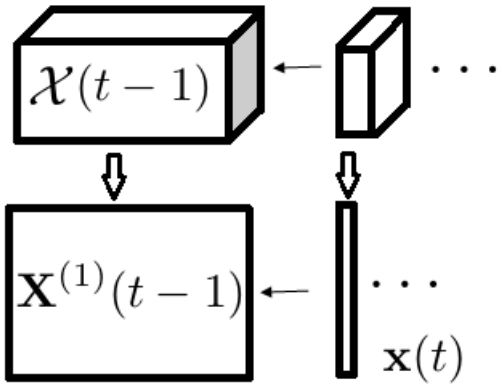


Fig. 1: Adaptive third-order tensor model and its equivalent matrix form.

In this section, given the estimates of $\mathbf{A}(t-1), \mathbf{B}(t-1)$ and $\mathbf{C}(t-1)$, we use the alternating minimization approach to construct the recursive update expressions for $\mathbf{A}(t), \mathbf{B}(t)$ and $\mathbf{C}(t)$. Our algorithm can be summarized into the following four steps:

1) Given $\mathbf{H}^{\#}(t-1)$ and $\mathbf{x}(t)$, estimate $\mathbf{b}^T(t)$
2) Given $\mathbf{b}(t)$, estimate $\mathbf{H}(t)$
3) Extract $\mathbf{A}(t)$ and $\mathbf{C}(t)$, and estimate one column of $\mathbf{H}(t)$
4) Calculate $\mathbf{H}^{\#}(t)$ and update $\mathbf{b}(t)$

We now explain these steps in detail.

Step 1: Estimate $\mathbf{b}^T(t)$

Vector $\mathbf{b}^T(t)$ can be obtained as the least-square solution of (9), according to

$$\begin{aligned}\hat{\mathbf{b}}^T(t) &= \underset{\mathbf{b}^T}{\operatorname{argmin}} \parallel \mathbf{x}(t) - \mathbf{H}(t-1)\mathbf{b}^T(t) \parallel_2^2 \\ &= \mathbf{H}^{\#}(t-1)\mathbf{x}(t), \end{aligned} \tag{11}$$

where $\mathbf{H}^{\#}(t-1)$ has been calculated in the previous iteration.

Step 2: Estimate $\mathbf{H}(t)$

Unlike [2], we use here a Newton-type method to find $\mathbf{H}(t)$. Let $\mathbf{h} = \mathrm{vec}(\mathbf{H})$. Function $\phi(\tau)$ can be rewritten as

$$\begin{aligned}\phi(\tau) =& \parallel \mathbf{x}(\tau) - \mathbf{H}(t)\mathbf{b}^T(\tau) \parallel_2^2 \\ =& \parallel \mathbf{x}(\tau) - (\mathbf{b}(\tau) \otimes \mathbf{I}_{IK})\mathbf{h}(t) \parallel_2^2, \end{aligned} \tag{12}$$

wherein we have exploited the fact that $\mathrm{vec}(\mathbf{ABC}^T) = (\mathbf{C} \otimes \mathbf{A})\,\mathrm{vec}(\mathbf{B})$.

As mentioned in [14], the direction at the maximum rate of change to $\mathbf{h}$ of a function Φ is given by $[\mathfrak{D}_{\mathbf{h}^*}\Phi(\mathbf{h}, \mathbf{h}^*)]^T$, where $[\mathfrak{D}_{\mathbf{h}^*}\Phi(\mathbf{h}, \mathbf{h}^*)]^T$ is the derivative of Φ with respect to $\mathbf{h}^*$. Consequently, we have

$$[\mathfrak{D}_{\mathbf{h}^*}\Phi(\mathbf{h}, \mathbf{h}^*)]^T \Big|_{\mathbf{h}=\mathbf{h}(t-1)} = -\sum_{\tau=1}^{t} \lambda^{t-\tau} \quad [(\mathbf{b}^H(\tau) \otimes \mathbf{I}_{IK})\mathbf{x}(\tau) + (\mathbf{b}^H(\tau)\mathbf{b}^T(\tau) \otimes \mathbf{I}_{IK})\mathbf{h}(t-1)]. \tag{13}$$

Thus, its Hessian is computed as

$$\mathfrak{H} = \mathfrak{D}_{\mathbf{h}}([\mathfrak{D}_{\mathbf{h}^*}\Phi(\mathbf{h},\mathbf{h}^*)]^T)\Big|_{\mathbf{h}=\mathbf{h}(t-1)} = \sum_{\tau=1}^{t}\lambda^{t-\tau}[(\mathbf{b}^H(\tau)\mathbf{b}^T(\tau))\otimes\mathbf{I}_{IK}] = \mathbf{R}(t)\otimes\mathbf{I}_{IK},$$

where

$$\mathbf{R}(t) = \sum_{\tau=1}^{t}\lambda^{t-\tau}\mathbf{b}^H(\tau)\mathbf{b}(\tau) \tag{14}$$

$$= \lambda\mathbf{R}(t-1) + \mathbf{b}^H(t)\mathbf{b}(t) \tag{15}$$

To achieve linear complexity, we replace (13) by instant gradient estimation (i.e., stochastic gradient)

$$[\mathfrak{D}_{\mathbf{h}^*}\phi(\mathbf{h},\mathbf{h}^*,t)]^T\Big|_{\mathbf{h}=\mathbf{h}(t-1)} = -\left[(\mathbf{b}^H(t)\otimes\mathbf{I}_{IK})\mathbf{x}(t) - (\mathbf{b}^H(t)\mathbf{b}^T(t))\otimes\mathbf{I}_{IK}\mathbf{h}(t-1)\right]. \tag{16}$$

The update rule of $\mathbf{h}$ is thus given by

$$\mathbf{h}(t) = \mathbf{h}(t-1) + \eta\mathfrak{H}^{-1}[\mathfrak{D}_{\mathbf{h}^*}\phi(\mathbf{h},\mathbf{h}^*,t)]^T, \tag{17}$$

where η is a step size. By substituting (14) and (16) into (17), we obtain

$$\mathbf{h}(t) = \mathbf{h}(t-1) + \eta[\mathbf{R}^{-1}(t)\mathbf{b}(t)\otimes\mathbf{I}_{IK}\mathbf{x}(t) - \mathbf{R}^{-1}(\mathbf{b}^H(t)\mathbf{b}^T(t))\otimes\mathbf{I}_{IK}\mathbf{h}(t-1)]. \tag{18}$$

We can stack (i.e., unvec) (18) in matrix form as follows:

$$\mathbf{H}(t) = \mathbf{H}(t-1) + \eta[\mathbf{x}(t) - \mathbf{H}(t-1)\mathbf{b}^T(t)]\mathbf{b}^*(t)\mathbf{R}^{-1}(t). \tag{19}$$

Here, we can see that calculating and storing the Hessian explicitly as in (14) is not necessary. Instead, we only need to calculate the pseudo-inverse of $\mathbf{R}(t)$. Since $\mathbf{R}(t)$ has a rank-1 update structure, its pseudo-inverse can be efficiently updated using the inversion lemma as

$$\mathbf{R}^{-1}(t) = [\lambda\mathbf{R}(t-1) + \mathbf{b}^T(t)\mathbf{b}^*(t)]^{-1} = \lambda^{-1}\mathbf{R}^{-1}(t-1) - \beta^{-1}(t)\mathbf{u}(t)\mathbf{u}^H(t). \tag{20}$$

where

$$\beta(t) = 1 + \lambda^{-1}\mathbf{b}^*(t)\mathbf{R}^{-1}(t-1)\mathbf{b}^T(t), \tag{21}$$

$$\mathbf{u}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1)\mathbf{b}^T(t), \tag{22}$$

Substituting (20) into (19) yields

$$\mathbf{H}(t) = \mathbf{H}(t-1) + \mathbf{d}(t)\mathbf{u}^H(t), \tag{23}$$

where

$$\mathbf{d}(t) = \gamma(t)[\mathbf{x}(t) - \mathbf{H}(t-1)\mathbf{b}^T(t)], \tag{24}$$

$$\gamma(t) = \eta(1 - \beta^{-1}(t)\mathbf{b}^*(t)\mathbf{u}(t)). \tag{25}$$

Step 3: Extract $\mathbf{A}(t)$ *and* $\mathbf{C}(t)$ *&* *update of one column of* $\mathbf{H}(t)$

The purpose of this step is three-fold: 1) to preserve an approximate Khatri-Rao product structure of $\mathbf{H}(t)$ in order to improve the estimation accuracy and ensure the algorithm's convergence, 2) to provide a reduced rank update structure which allows the calculation of $\mathbf{H}^{\#}(t)$ with a linear complexity in the next step, and 3) to extract from $\mathbf{H}(t)$ the loading matrices $\mathbf{A}(t)$ and $\mathbf{C}(t)$.

A solution to this problem can be implemented efficiently as follows. Before proceeding further, we first consider the way to extract $\mathbf{A}(t)$ and $\mathbf{C}(t)$ from $\mathbf{H}(t)$ [15], [2], based on

$$\begin{aligned} \mathbf{H}(t) &\simeq \mathbf{A}(t) \odot \mathbf{C}(t) \\ &\simeq [\mathbf{a}_1(t) \otimes \mathbf{c}_1(t) \cdots \mathbf{a}_R(t) \otimes \mathbf{c}_R(t)] \\ &\simeq [\mathrm{vec}(\mathbf{c}_1(t)\mathbf{a}_1^T(t)) \cdots \mathrm{vec}(\mathbf{c}_R(t)\mathbf{a}_R^T(t))]. \end{aligned} \tag{26}$$

First, we observe that each column is a vectorized representation of rank-1 matrix. The loading vectors $\mathbf{c}_i(t)$ and $\mathbf{a}_i(t)$ are, thus, the principal left and right singular vectors respectively of matrix $\mathbf{H}_i(t) = \mathrm{unvec}(\mathbf{a}_i(t) \otimes \mathbf{c}_i(t))$. Since using batch SVD is not suitable in adaptive tracking, a single Bi-SVD iteration [16] can be used to update $\mathbf{a}_i(t)$ and $\mathbf{c}_i(t)$ recursively according to

$$\text{for } \; i = 1, \ldots, R$$

$$\mathbf{a}_i(t) = \mathbf{H}_i^T(t)\mathbf{c}_i(t-1) \tag{27}$$

$$\mathbf{c}_i(t) = \frac{\mathbf{H}_i(t)\mathbf{a}_i(t)}{\| \mathbf{H}_i(t)\mathbf{a}_i(t) \|}. \tag{28}$$

Then, having obtained $\mathbf{A}(t)$ and $\mathbf{C}(t)$, $\mathbf{H}(t)$ can be re-estimated according to (26).

However, to achieve the linear complexity of our algorithm, we choose to re-update only one column of $\mathbf{H}(t)$ at each iteration. In other words, at time instant t, we select the j-th column of $\mathbf{H}(t)$ in a cyclic way (i.e., $j = (t \mod R) + 1$) and update it as

$$\hat{\mathbf{h}}_j(t) = \mathbf{a}_j(t) \otimes \mathbf{c}_j(t). \tag{29}$$

Because of the best rank-1 approximation property of the SVD, we take advantage of the denoised loading vectors $\mathbf{a}_j(t)$ and $\mathbf{c}_j(t)$ and, thereby, improve the accuracy of $\mathbf{H}(t)$ estimation. The other columns of $\mathbf{H}(t)$ are left unchanged to preserve the reduced rank structure of the updated matrix. The updated version of $\mathbf{H}(t)$ can be expressed as

$$\hat{\mathbf{H}}(t) = \mathbf{H}(t) + \mathbf{z}(t)\mathbf{e}_j^T(t), \tag{30}$$

where

$$\mathbf{z}(t) = \hat{\mathbf{h}}_j(t) - \mathbf{h}_j(t), \tag{31}$$

and $\mathbf{e}_j(t)$ is the unit vector whose j-th entry is one. It is straightforward to see that $\hat{\mathbf{H}}(t)$ has a rank-2 update structure by substituting (23) into (30).

Step 4: Calculate $\mathbf{H}^{\#}(t)$ and update $\mathbf{b}(t)$

As mentioned in Step 3, we can compute the pseudo-inverse of $\hat{\mathbf{H}}(t)$ efficiently thanks to its rank-2 update structure. Our case corresponds to the fifth one in [17]. Now we can re-estimate $\mathbf{b}(t)$ as

$$\mathbf{b}^T(t) = \hat{\mathbf{H}}(t)^{\#}\mathbf{x}(t), \tag{32}$$

and hence obtain $\mathbf{B}(t)$ from (7).

Algorithm initialization

To initialize $\mathbf{A}(0)$, $\mathbf{B}(0)$, $\mathbf{C}(0)$, $\mathbf{H}^{\#}(0)$ and $\mathbf{R}^{-1}(0)$ before tracking, we can capture J_0 slices (J_0 is chosen to satisfy the uniqueness condition of (4)), and then run a batch PARAFAC algorithm to obtain $\mathbf{A}(0)$, $\mathbf{B}(0)$, and $\mathbf{C}(0)$. After that, we compute $\mathbf{H}^{\#}(0) = (\mathbf{A}(0) \odot \mathbf{C}(0))^{\#}$ and $\mathbf{R}^{-1}(0) = (\mathbf{B}^T(0)\mathbf{B}(0))^{-1}$.

The above steps are summarized in Table I.

IV. Adaptive non-negative PARAFAC

Now, we consider the case where the constraint of non-negativity is imposed, and modify the proposed SOAP algorithm accordingly. Given the non-negative estimates of $\mathbf{A}(t-1) \geqslant 0$, $\mathbf{B}(t-1) \geqslant 0$, and $\mathbf{C}(t-1) \geqslant 0$, we want to find recursive updates of $\mathbf{A}(t) \geqslant 0$, $\mathbf{B}(t) \geqslant 0$, and $\mathbf{C}(t) \geqslant 0$, which are the loading matrices of the PARAFAC decomposition. Before describing

TABLE I: Summary of SOAP

Inputs:

$\mathbf{A}(t-1)$, $\mathbf{B}(t-1)$, $\mathbf{C}(t-1), \mathbf{H}(t-1)$, $\mathbf{H}^{\#}(t-1)$, $\mathbf{R}^{-1}(t-1)$

Step 1: Estimate $\mathbf{b}^T(t)$

$$\mathbf{b}^T(t) = \mathbf{H}^{\#}(t-1)\mathbf{x}(t) \tag{11}$$

Step 2: Estimate $\mathbf{H}(t)$

$$\beta(t) = 1 + \mathbf{b}(t)\mathbf{R}^{-1}(t-1)\mathbf{b}(t)^T \tag{21}$$
$$\mathbf{u}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1)\mathbf{b}^T(t) \tag{22}$$
$$\mathbf{R}^{-1}(t) = \lambda^{-1}\mathbf{R}(t-1)^{-1} - \beta^{-1}(t)\mathbf{u}(t)\mathbf{u}^H(t) \tag{20}$$
$$\gamma(t) = \eta(1 - \beta^{-1}(t)\mathbf{b}^*(t)\mathbf{u}(t)) \tag{25}$$
$$\mathbf{d}(t) = \gamma(t)[\mathbf{x}(t) - \mathbf{H}(t-1)\mathbf{b}^T(t)] \tag{24}$$
$$\mathbf{H}(t) = \mathbf{H}(t-1) + \mathbf{d}(t)\mathbf{u}^T(t) \tag{23}$$

Step 3: Extract $\mathbf{A}(t)$ and $\mathbf{C}(t)$ and update one column of $\mathbf{H}(t)$

- Extract $\mathbf{A}(t)$ and $\mathbf{C}(t)$

for $i = 1, \ldots, R$

$$\mathbf{H}_i(t) = \text{unvec}(\mathbf{h}_i(t))$$
$$\mathbf{a}_i(t) = \mathbf{H}_i^T(t)\mathbf{c}_i(t-1)$$
$$\mathbf{c}_i(t) = \frac{\mathbf{H}_i(t)\mathbf{a}_i(t)}{\| \mathbf{H}_i(t)\mathbf{a}_i(t) \|}$$

- Select j-th column of $\mathbf{H}(t)$ in a cyclic way

$$\hat{\mathbf{h}}_j(t) = \mathbf{a}_j(t) \otimes \mathbf{c}_j(t) \tag{29}$$
$$\mathbf{z}(t) = \hat{\mathbf{h}}_j(t) - \mathbf{h}_j(t) \tag{31}$$
$$\mathbf{H}(:, j)(t) = \hat{\mathbf{h}}_j(t)$$

Step 4: Calculate $\mathbf{H}^{\#}(t)$ and re-update $\mathbf{b}(t)$

- Calculate $\mathbf{H}^{\#}(t)$ using fast matrix inversion lemma

$$\mathbf{b}^T(t) = \mathbf{H}(t)^{\#}\mathbf{x}(t) \tag{32}$$
$$\mathbf{B}^T(t) = [\mathbf{B}^T(t-1), \mathbf{b}^T(t)] \tag{7}$$

Outputs:

$\mathbf{A}(t)$, $\mathbf{B}(t)$, $\mathbf{C}(t), \mathbf{H}(t)$, $\mathbf{H}^{\#}(t)$, $\mathbf{R}^{-1}(t)$

our method, we note that SOAP works with the general case of complex values, while in the section we only consider real non-negative ones.

A simple approach is to use the positive orthant projection. That is, at each step of SOAP, we project the result on the positive orthant, for example, in Step 1, set $\mathbf{b}^T(t) = [\mathbf{b}^T(t)]^+$. However, this naive combination does not work for Step 2 (the so-called projected Newton-type method),

as indicated in the context of constrained optimization [18] or least-squares non-negative matrix approximation [19].

In practice, for batch processing, the projected Newton-type method requires a combination of restrictions on the Hessian (e.g., diagonal or partly diagonal [20]) and the Armijo-like step rule [18] to guarantee a quadratic rate of convergence. In spite of their advantage, computing the Armijo-like step rule is expensive and not suitable for adaptive algorithms. It is even more difficult in our case because the global optimum value can be changed continuously, depending on new data.

Therefore, we propose to use a simpler strategy. Particularly, because of the slowly time-varying model assumption, we restrict ourselves to only calculating the diagonal of the Hessian and using a fixed step rule. Even though the convergence proof is not available yet, this strategy still gives an acceptable performance and represents a good trade-off between the performance and the complexity, as indicated in Section VI.

Now, for the details, several modifications of SOAP for handling the non-negativity constraint are as follows. At the end of Step 1, we add one minor step after obtaining $\mathbf{b}^T(t)$, by setting

$$\mathbf{b}^T(t) = [\mathbf{b}^T(t)]^+. \tag{33}$$

In Step 2, after calculating $\mathbf{R}^{-1}(t)$, we extract the diagonal matrix (which is non-negative since R is positive Hermitian)

$$\hat{\mathbf{R}}^{-1}(t) = \mathrm{diag}(\mathrm{diag}(\mathbf{R}^{-1}(t))), \tag{34}$$

and then calculate

$$\hat{\mathbf{u}}(t) = \hat{\mathbf{R}}^{-1}(t)\mathbf{b}^T(t). \tag{35}$$

Thus, $\mathbf{H}(t)$ is updated, using $\mathbf{d}(t)$ and $\hat{\mathbf{u}}(t)$ instead of $\mathbf{d}(t)$ and $\mathbf{u}(t)$, as

$$\mathbf{H}(t) = \mathbf{H}(t-1) + \mathbf{d}(t)\hat{\mathbf{u}}^T(t). \tag{36}$$

Then, we project $\mathbf{H}(t)$ on the positive orthant to obtain

$$\tilde{\mathbf{H}}(t) = [\mathbf{H}(t)]^+. \tag{37}$$

As previously discussed in Step 3, $\mathbf{c}_i(t)$ and $\mathbf{a}_i(t)$ are respectively the principal left and right singular vectors of the matrix $\mathbf{H}_i(t) = \mathrm{unvec}(\mathbf{a}_i(t) \otimes \mathbf{c}_i(t))$. The updated loading matrices $\mathbf{A}(t)$

TABLE II: Summary of NSOAP

Inputs:

$\mathbf{A}(t-1) \geqslant 0$, $\mathbf{B}(t-1) \geqslant 0$, $\mathbf{C}(t-1) \geqslant 0, \mathbf{H}(t-1)$, $\mathbf{H}^{\#}(t-1)$, $\mathbf{R}^{-1}(t-1)$

Step 1: Estimate $\mathbf{b}^T(t)$

Perform Step 1 of SOAP

$$\mathbf{b}^T(t) = [\mathbf{b}^T(t)]^+ \tag{33}$$

Step 2: Estimate $\mathbf{H}(t)$

$$\beta(t) = 1 + \mathbf{b}(t)\mathbf{R}^{\#}(t-1)\mathbf{b}(t)^T \tag{21}$$

$$\mathbf{u}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1)\mathbf{b}(t)^T \tag{22}$$

$$\mathbf{R}^{-1}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1) - \beta^{-1}(t)\mathbf{u}(t)\mathbf{u}(t)^T \tag{20}$$

$$\hat{\mathbf{R}}^{-1}(t) = \mathrm{diag}(\mathrm{diag}(\mathbf{R}^{-1}(t))) \tag{34}$$

$$\gamma(t) = \eta(1 - \beta^{-1}(t)\mathbf{b}^*(t)\mathbf{u}(t)) \tag{25}$$

$$\mathbf{d}(t) = \gamma(t)(\mathbf{x}(t) - \mathbf{H}(t-1)\mathbf{b}^T(t)) \tag{24}$$

$$\hat{\mathbf{u}}(t) = \hat{\mathbf{R}}^{-1}(t)\mathbf{b}^T(t) \tag{35}$$

$$\mathbf{H}(t) = \mathbf{H}(t-1) + \mathbf{d}(t)\hat{\mathbf{u}}^T(t)$$

$$\tilde{\mathbf{H}}(t) = [\mathbf{H}(t)]^+ \tag{37}$$

Step 3: Same as Step 3 of SOAP but with $\tilde{\mathbf{H}}(t)$

Step 4: Calculate $\mathbf{H}^{\#}(t)$ and re-update $\mathbf{b}(t)$

Perform Step 4 of SOAP

$$\mathbf{b}^T(t) = [\mathbf{b}^T(t)]^+ \tag{33}$$

$$\mathbf{B}^T(t) = [\mathbf{B}^T(t-1), \mathbf{b}^T(t)] \tag{7}$$

Outputs:

$\mathbf{A}(t) \geqslant 0$, $\mathbf{B}(t) \geqslant 0$, $\mathbf{C}(t) \geqslant 0, \mathbf{H}(t)$, $\mathbf{H}^{\#}(t)$, $\mathbf{R}^{-1}(t)$

and $\mathbf{C}(t)$, obtained by (27) and (28), are still non-negative since $\tilde{\mathbf{H}}_i(t)$ is non-negative and so are $\mathbf{A}(t-1)$ and $\mathbf{C}(t-1)$ (already obtained in the previous time instant). In Step 4, a positive orthant projection like (33) is used.

We call this modification of SOAP as non-negative SOAP (NSOAP). A summary of all the steps of the NSOAP algorithm is given in Table II. The initialization of NSOAP is similar to that of SOAP, except that a batch non-negative PARAFAC is used instead of the standard PARAFAC.

V. Discussions

In this section, we provide some important comments on the similarities and differences between our algorithms and those developed in [2]. Discussions on parallel implementations are also given.

First, it is straightforward to realize that in all steps the main cost of SOAP comes from the matrix-vector product. Thus, it has linear computational complexity of $O(IKR)$. NSOAP is slightly more expensive but has the same complexity order as SOAP.

Second, obviously, in Step 3 of SOAP, we can choose to update $d > 1$ columns of $\mathbf{H}(t)$ instead of only 1 column. d can be chosen as a parameter to balance between the estimation accuracy and the numerical cost.

Third, the main reason that helps maintain the linear complexity, while still having a comparable or even superior performance (as shown in the next section) stems from Steps 2 and 3. In Step 2, both the gradient (stochastic) and Hessian are used instead of only the gradient as in PARAFAC-RLST. In Step 3, we exploit the Khatri-Rao product structure, which is not fully exploited in both PARAFAC-RLST and PARAFAC-SDT, to enhance the performance. The fast calculation of $\mathbf{H}^{\#}(t)$ using the pseudo-inverse lemma in Step 4 is a consequence of designing $\mathbf{H}(t)$ to have a rank-2 update in Steps 2 and 3.

Finally, in Step 3, we can update all columns of $\mathbf{H}(t)$ using (27) and (28) for $i = 1, \ldots, R$. It, however, leads to calculate $\mathbf{H}^{\#}(t)$ without a rank-2 update structure as in SOAP. Hence, $\mathbf{H}^{\#}(t)$ can be obtained by using the Khatri-Rao product structure leading to

$$\mathbf{H}^{\#}(t) = [\mathbf{A}^T(t)\mathbf{A}(t) * \mathbf{C}^T(t)\mathbf{C}(t)]^{-1}[\mathbf{A}(t) \odot \mathbf{C}(t)]^T. \tag{38}$$

The cost for this implementation is $O(IKR^2)$ and is thus disregarded in this paper.

Now, we show that both SOAP and NSOAP are easy to realize in a parallel scheme. This implementation is important when used for massive data (large dimensional systems). It can be observed that the main computational cost comes from the matrix-vector product. Assume that R computational units (DSPs) are available. Then, in Step 1, Equation (11) corresponds to

$$\mathbf{b}_i^T(t) = \tilde{\mathbf{h}}_i(t-1)\mathbf{x}(t), \qquad i = 1, \ldots, R, \tag{39}$$

where $\tilde{\mathbf{h}}_i(t-1)$ is i-th row of $\mathbf{H}^{\#}(t-1)$. It means that we have replaced the matrix-vector product by the vector-vector product. This procedure can also be applied in Step 4. Steps 2 and 3 themselves have already a parallel structure and, again, note that each column of $\mathbf{H}(t)$ can be estimated independently. By this way, the overall cost can be reduced, by approximately a factor of R, to $O(IK)$ flops per iteration.

VI. SIMULATIONS

In this section, we illustrate the performance of proposed algorithms using both synthetic and real data, from [24].

A. Performance comparison

First, we use the framework provided by the authors in [2] to verify and compare the performance of the considered algorithms. A time-varying model is thereby constructed so that, at time instant t, we generate the loading matrices $\mathbf{A}(t)$ and $\mathbf{C}(t)$ as

$$\mathbf{A}(t) = (1 - \varepsilon_A)\mathbf{A}(t-1) + \varepsilon_A \mathbf{N}_A, \tag{40}$$

$$\mathbf{C}(t) = (1 - \varepsilon_C)\mathbf{C}(t-1) + \varepsilon_C \mathbf{N}_C, \tag{41}$$

where ε_A and ε_C control the speed of variation for $\mathbf{A}$ and $\mathbf{C}$ between two successive observations, $\mathbf{N}_A$ and $\mathbf{N}_C$ are random matrices with identical sizes with $\mathbf{A}$ and $\mathbf{C}$. Generate a vector $\mathbf{b}(t)$ randomly and the noiseless input data $\mathbf{x}(t)$ is given by

$$\mathbf{x}(t) = [\mathbf{A}(t) \odot \mathbf{C}(t)]\mathbf{b}^T(t).$$

Thus, this observation vector follows the model described in Section II-B and are constrained by the assumptions therein. Then, the noisy observation is given by

$$\tilde{\mathbf{x}}(t) = \mathbf{x}(t) + \sigma \mathbf{n}(t), \tag{42}$$

where $\mathbf{n}(t)$ is a zero mean, unit-variance noisy vector while parameter σ is introduced to control the noise level. We set a default value of σ to 10^{-3}. To have a fair comparison, we keep all default parameters of the algorithms and the model as offered by the authors of [2]. A summary of parameters used in our experiments is showed in Table III.

The performance measures for the loading matrices $\mathbf{A}(t)$ and $\mathbf{C}(t)$ are the standard deviations (STD) between the true loading matrices, $\mathbf{A}(t)$ and $\mathbf{C}(t)$, and their estimates, $\mathbf{A}_{\text{es}}(t)$ and $\mathbf{C}_{\text{es}}(t)$, up to a scale and permutation indeterminacy at each time

$$\text{STD}_{\mathbf{A}}(t) = \| \mathbf{A}(t) - \mathbf{A}_{\text{es}}(t) \|_F, \tag{43}$$

$$\text{STD}_{\mathbf{C}}(t) = \| \mathbf{C}(t) - \mathbf{C}_{\text{es}}(t) \|_F, \tag{44}$$

TABLE III: Experimental set-up parameters

Fig.	I	J_0	K	R	T	$\varepsilon_A, \varepsilon_C$	λ	η
2	20	50	20	8	1000	10^{-3}	0.8	NA
3	20	50	20	8	1000	10^{-3}	0.8	0.02
4	5	70	61	3	201	NA	0.8	0.002
5	20	50	20	8	1000	10^{-2} 10^{-3} 10^{-5}	0.8	0.02
6-7	5	50	61	3	1000	0	0.8	0.02

For the loading matrix $\mathbf{B}(t)$, because of its time-shift structure, we verify its performance through $\mathbf{x}(t)$ by

$$\mathrm{STD}_{\mathbf{B}}(t) = \| \mathbf{x}(t) - \mathbf{x}_{\mathrm{es}}(t) \|_2 . \tag{45}$$

To assess the convergence rate of algorithms, we set up the following scenario: always keep the speed of variation of $\mathbf{A}$ and $\mathbf{C}$ constant except at a few specific time instants at which the speed of variation arbitrarily increases. Thus, the algorithm which recovers faster yields a better convergence rate. This scenario is similar to the convergence rate assessment in the context of subspace tracking, see for example [21], [22].

The first experiment is to compare the performance of SOAP with that of PARAFAC-RLST, PARAFAC-SDT (exponential window) and batch PARAFAC-ALS (Alternating Least-Squares). Batch PARAFAC here serves as a "lower bound" for adaptive algorithms. As shown in Figure 2, SOAP outperforms both PARAFAC-RLST and PARAFAC-SDT and is close to batch PARAFAC. In addition, SOAP, PARAFAC-RLST, and PARAFAC-SDT approximately have the same convergence rate. Again, we note that the computational complexity of SOAP is $O(IKR)$ as compared to $O(IKR^2)$ of PARAFAC-RLST and PARAFAC-SDT.

In the second experiment for non-negative data, we take absolute value of the previous model,

i.e.,

$$\mathbf{A}^{+}(t) = |\mathbf{A}(t)|, \tag{46}$$

$$\mathbf{C}^{+}(t) = |\mathbf{C}(t)|, \tag{47}$$

$$\mathbf{x}^{+}(t) = |\mathbf{x}(t)|, \tag{48}$$

where $(^{+})$ means non-negative. Since there exist no other adaptive non-negative PARAFAC algorithms apart from our NSOAP, we compare NSOAP with the batch non-negative PARAFAC (Batch N-PARAFAC) algorithm implemented in the N-way toolbox [23]. The results are shown in Figure 3.

To further exemplify the performance of NSOAP, we apply it to a typical example using a fluorescence dataset [24] which includes five samples of fluorescence excitation-emission of size 5 samples $\times$ 201 emission wavelengths $\times$ 61 excitation wavelengths. It is showed that the estimated loading matrices from PARAFAC with nonnegative constraint are similar to the pure spectra. Here, we use an initialization tensor of size $5 \times 70 \times 61$. Note that emission wavelength is relatively short and during the interval $[250, 300]$, one of three components is almost zero. Figure 4 shows that NSOAP can recover the tensor components in this particular example.

B. Effect of the speed of variation

In this section, we consider different values of the speed of variation ε_A and ε_C to evaluate its effect on the algorithm's performance. Figure 5 shows that SOAP adapts better to fast variation ($\varepsilon_A = \varepsilon_C = 10^{-2}$) than PARAFAC-RLST and PARAFAC-SDT. In this case, SOAP still converges while the others diverge. When the variation is smaller ($\varepsilon_A = \varepsilon_C = 10^{-3}$ or 10^{-5}), SOAP is comparable to PARAFAC-RLST and PARAFAC-SDT.

C. Waveform-preserving character

The waveform-preserving character is important in communication and bio-medical applications, as for example in the blind receivers for direct-sequence code-division multiple access (DS-CDMA) [25] and multi-subject Functional Magnetic Resonance Imaging (fMRI) analysis [15]. In this section, we illustrate this property of our algorithms via a synthetic example. We first generate the loading matrix $\mathbf{B}(t)$ including three kinds of signals: a chirp, a rectangular wave

and a sawtooth wave. The signal length is 1 s and the sample rate is 1 kHz. For the chirp, the instantaneous frequency is 0 at $t = 0$ and crosses 300 Hz at $t = 1$ s. The rectangular wave has a frequency of 50 Hz, and the symmetric sawtooth has a repetition frequency of 20 Hz with a sawtooth width of 0.05 s. For $\mathbf{A}(t)$ and $\mathbf{C}(t)$, we use the loading matrices from the fluoresence example where components of $\mathbf{A}(t)$ have a sharp change and components of $\mathbf{C}(t)$ have a smooth change. The PARAFAC model is then disturbed by a Gaussian noise with signal-to-noise-ratio (SNR) of 15 dB. The SNR (in dB) is defined by

$$\mathrm{SNR} = 10 \log_{10} \frac{\mathrm{E}\{\| \mathbf{A}(t) \odot \mathbf{C}(t)\mathbf{b}(t) \|\}^2}{\sigma^2}. \tag{49}$$

The simulation results when applying SOAP and NOSAP are shown in Figures 6 and 7, corresponding to the first 200 data samples (iterations). As we can see, both algorithms lead to a good restoration of the original components.

VII. Conclusions

In this paper, we have proposed two efficient adaptive PARAFAC decomposition algorithms: SOAP for a standard setup and NSOAP for the non-negative constraint case. To our best knowledge, none adaptive non-negative PARAFAC has been addressed before. By exploiting the data structure, the proposed algorithms have a linear computational complexity $O(IKR)$ per iteration while enjoying a good performance as compared to the state-of-the-art algorithms. These algorithms can be considered as a starting point of real-time PARAFAC-based applications.

Acknowledgment

This work was supported by the National Foundation for Science and Technology Development of Vietnam under Grant No. 102.02-2015.32.

References

References

[1] K. Slavakis, G. Giannakis, G. Mateos, Modeling and optimization for big data analytics:(statistical) learning tools for our era of data deluge, Signal Processing Magazine, IEEE 31 (5) (2014) 18–31.

[2] D. Nion, N. D. Sidiropoulos, Adaptive algorithms to track the PARAFAC decomposition of a third-order tensor, Signal Processing, IEEE Transactions on 57 (6) (2009) 2299–2310.

[3] M. Mardani, G. Mateos, G. B. Giannakis, Subspace learning and imputation for streaming big data matrices and tensors, Signal Processing, IEEE Transactions on 63 (10) (2015) 2663–2677.

[4] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, H. A. Phan, Tensor decompositions for signal processing applications: From two-way to multiway component analysis, Signal Processing Magazine, IEEE 32 (2) (2015) 145–163.

[5] M. Mørup, L. K. Hansen, S. M. Arnfred, Algorithms for sparse nonnegative Tucker decompositions, Neural computation 20 (8) (2008) 2112–2131.

[6] P. Comon, Tensors: a brief introduction, IEEE Signal Processing Magazine 31 (3) (2014) 44–53.

[7] T. G. Kolda, B. W. Bader, Tensor decompositions and applications, SIAM review 51 (3) (2009) 455–500.

[8] S. S. Haykin, Adaptive filter theory, Pearson Education India, 2007.

[9] P. Comon, G. H. Golub, Tracking a few extreme singular values and vectors in signal processing, Proceedings of the IEEE 78 (8) (1990) 1327–1343.

[10] X. G. Doukopoulos, G. V. Moustakides, Fast and stable subspace tracking, Signal Processing, IEEE Transactions on 56 (4) (2008) 1452–1465.

[11] A. Cichocki, R. Zdunek, A. H. Phan, S.-i. Amari, Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation, John Wiley & Sons, 2009.

[12] A. Stegeman, Finding the limit of diverging components in three-way Candecomp/Parafaca demonstration of its practical merits, Computational Statistics & Data Analysis 75 (2014) 203–216.

[13] J. B. Kruskal, Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics, Linear algebra and its applications 18 (2) (1977) 95–138.

[14] A. Hjorungnes, D. Gesbert, Complex-valued matrix differentiation: Techniques and key results, Signal Processing, IEEE Transactions on 55 (6) (2007) 2740–2746.

[15] C. F. Beckmann, S. M. Smith, Tensorial extensions of independent component analysis for multisubject FMRI analysis, Neuroimage 25 (1) (2005) 294–311.

[16] P. Strobach, Bi-iteration SVD subspace tracking algorithms, Signal Processing, IEEE Transactions on 45 (5) (1997) 1222–1240.

[17] C. D. Meyer, Jr, Generalized inversion of modified matrices, SIAM Journal on Applied Mathematics 24 (3) (1973) 315–323.

[18] D. P. Bertsekas, Projected newton methods for optimization problems with simple constraints, SIAM Journal on control and Optimization 20 (2) (1982) 221–246.

[19] D. Kim, S. Sra, I. S. Dhillon, Fast Newton-type methods for the least squares nonnegative matrix approximation problem., in: SDM, Vol. 7, SIAM, 2007, pp. 343–354.

[20] M. Schmidt, D. Kim, S. Sra, Projected Newton-type methods in machine learning, Optimization for Machine Learning (2012) 305.

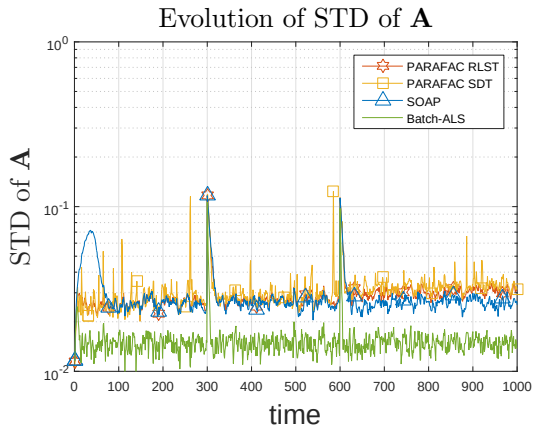
[21] P. Strobach, Fast recursive subspace adaptive ESPRIT algorithms, Signal Processing, IEEE Transactions on 46 (9) (1998) 2413–2430.

[22] R. Badeau, G. Richard, B. David, Fast and stable YAST algorithm for principal and minor subspace tracking, Signal Processing, IEEE Transactions on 56 (8) (2008) 3437–3446.

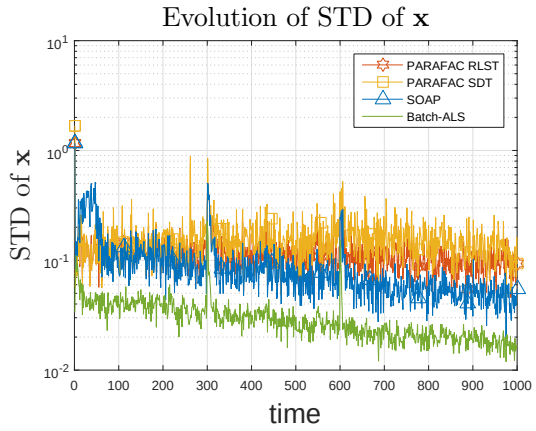
[23] C. A. Andersson, R. Bro, The n-way toolbox for MATLAB, Chemometrics and Intelligent Laboratory Systems 52 (1) (2000) 1–4.

[24] R. Bro, PARAFAC. tutorial and applications, Chemometrics and intelligent laboratory systems 38 (2) (1997) 149–171.

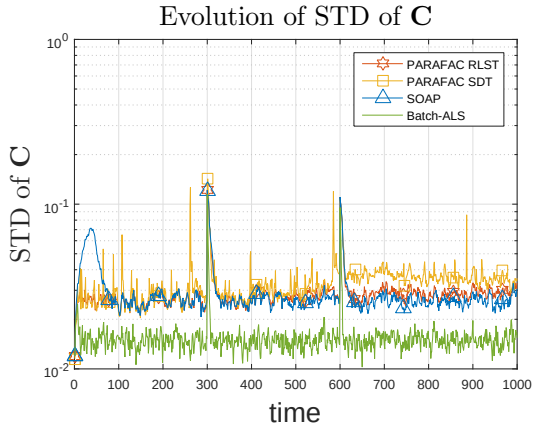
[25] N. D. Sidiropoulos, G. B. Giannakis, R. Bro, Blind PARAFAC receivers for DS-CDMA systems, Signal Processing, IEEE Transactions on 48 (3) (2000) 810–823.



(a) Loading matrix $\mathbf{A}(t)$

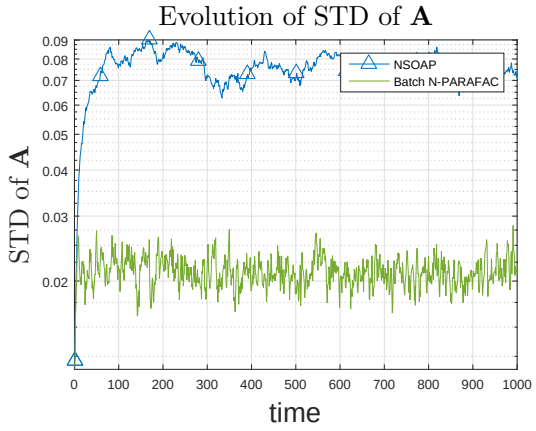


(b) Observation vector $\mathbf{x}(t)$

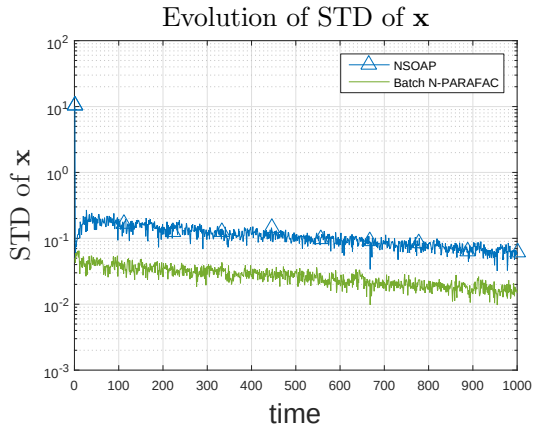


(c) Loading matrix $\mathbf{C}(t)$

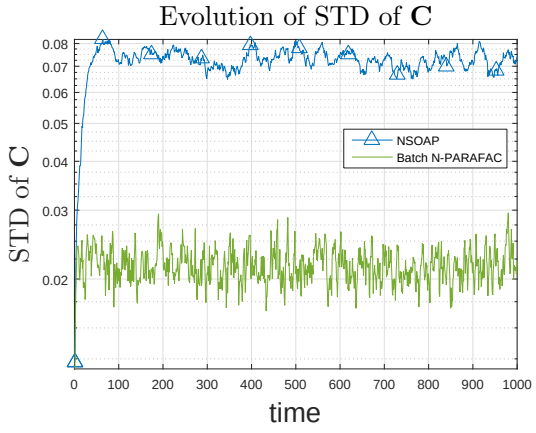
Fig. 2: Performance comparison of four algorithms when loading matrices change relatively fast, $\varepsilon_A = \varepsilon_C = 10^{-3}$.



(a) Loading matrix $\mathbf{A}(t)$

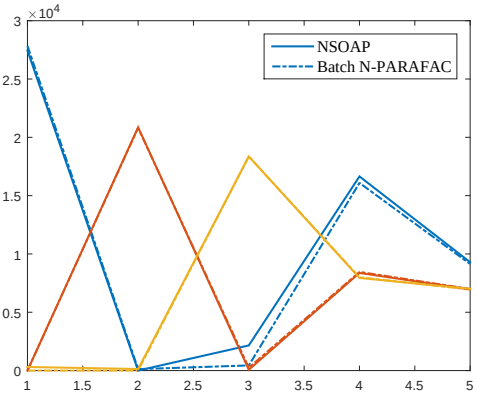


(b) Observation vector $\mathbf{x}(t)$

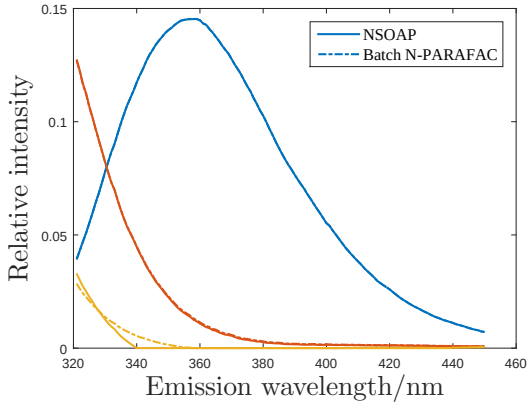


(c) Loading matrix $\mathbf{C}(t)$

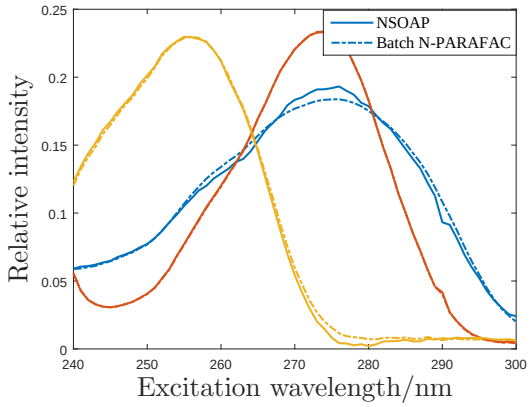
Fig. 3: Performance comparison of NSOAP with batch non-negative PARAFAC when loading matrices change relatively fast, $\varepsilon_A = \varepsilon_C = 10^{-3}$.



(a) Loading matrix $\mathbf{A}(t)$

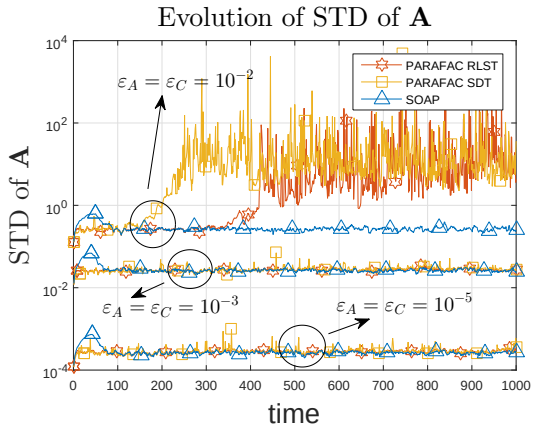


(b) Loading matrix $\mathbf{B}(t)$

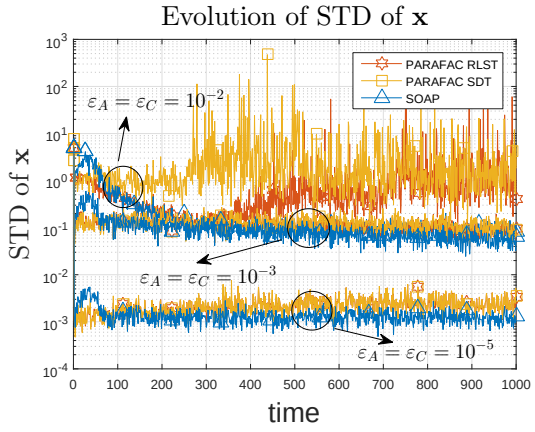


(c) Loading matrix $\mathbf{C}(t)$

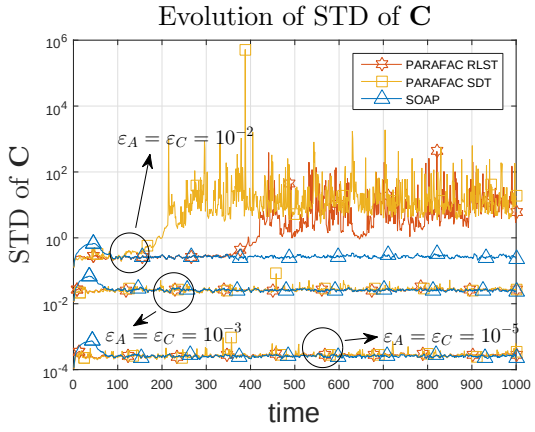
Fig. 4: Performance comparison of NSOAP with batch non-negative PARAFAC in fluorescence data set. For $\mathbf{B}(t)$, we present only a part of recovered loading matrix; initialization part is disregarded.



(a) Loading matrix $\mathbf{A}(t)$

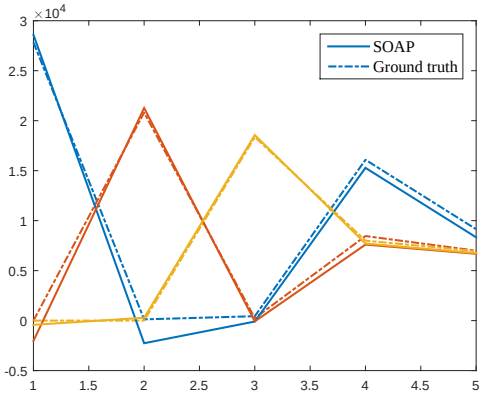


(b) Observation vector $\mathbf{x}(t)$

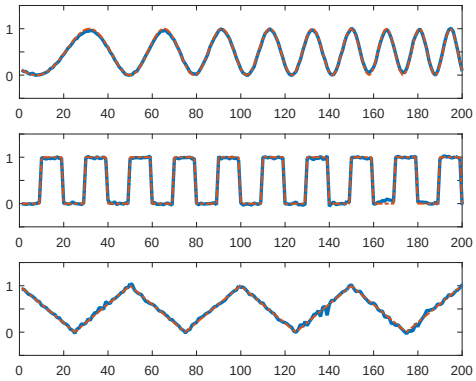


(c) Loading matrix $\mathbf{C}(t)$

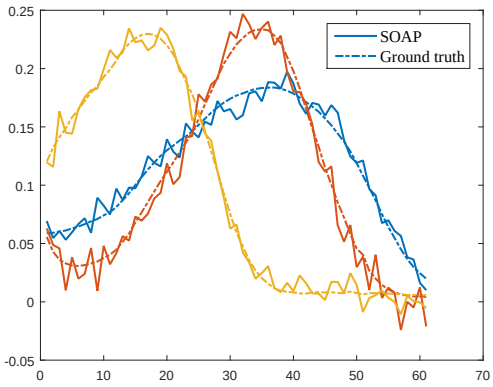
Fig. 5: The effect of the speed of variation on the algorithm performance.



(a) Loading matrix $\mathbf{A}(t)$

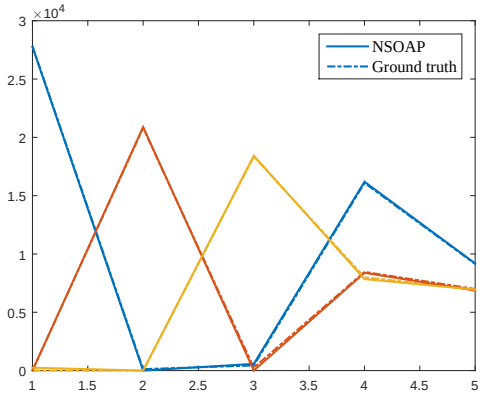


(b) Loading matrix $\mathbf{B}(t)$

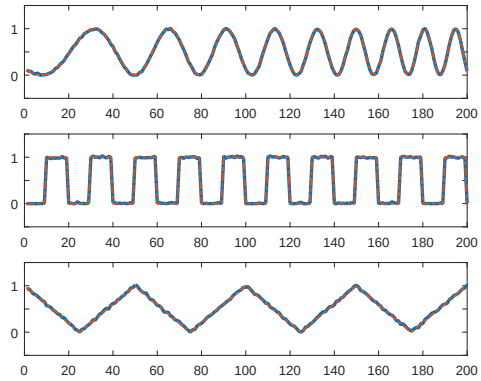


(c) Loading matrix $\mathbf{C}(t)$

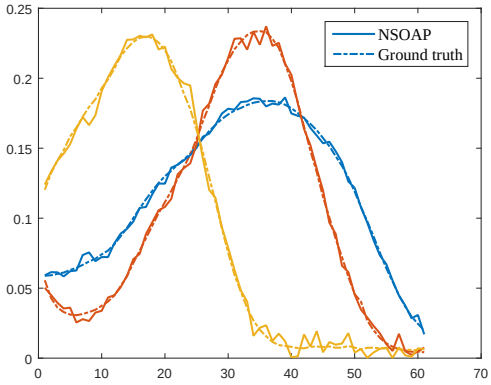
Fig. 6: Illustration of waveform-preserving character of SOAP through synthetic example with SNR = 15 dB. Solid line represents solution of SOAP while dash-dot line represents ground-truth.



(a) Loading matrix $\mathbf{A}(t)$



(b) Loading matrix $\mathbf{B}(t)$



(c) Loading matrix $\mathbf{C}(t)$

Fig. 7: Illustration of waveform-preserving character of NSOAP through synthetic example with SNR = 15 dB.