Contributions to fast matrix and tensor decompositions

Viet-Dung Nguyen

UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE MATHÉMATIQUES, INFORMATIQUE, PHYSIQUE THÉORIQUE ET INGÉNIERIE DES SYSTÈMES

Laboratoire Pluridisciplinaire de Recherche en Ingénierie des Systèmes, Mécanique et Énergétique (PRISME)

THÈSE présentée par :

Viet-Dung NGUYEN

Soutenue le : 16 Novembre 2016

pour obtenir le grade de : Docteur de l'université d'Orléans

Discipline / Spécialité : Traitement du Signal

Contribution aux décompositions rapides des matrices et tenseurs

THÈSE dirigée par :

Karim ABED-MERAIM	PR, Université d'Orléans, directeur de thèse
Linh-Trung NGUYEN	MCF–HDR, VNU University of Engineering and Technology, co-encadrant

RAPPORTEURS :

Pierre COMON	Directeur de recherche, CNRS, GIPSA-lab, Grenoble
Roland BADEAU	MCF–HDR, TELECOM ParisTech

JURY :

Rachid HARBA	PR, Université d'Orléans, président du jury
Karim ABED-MERAIM	PR, Université d'Orléans, directeur de thèse
Roland BADEAU	MCF–HDR, TELECOM ParisTech, rapporteur
Pierre COMON	Directeur de recherche, CNRS, GIPSA-lab, Grenoble, rapporteur
Marco DI RENZO	Directeur de recherche, CNRS, LSS-lab, Université Paris-Saclay, examinateur
Philippe FORSTER	PR, Université Paris-Ouest Nanterre, examinateur
Linh-Trung NGUYEN	MCF–HDR, VNU University of Engineering and Technology, co-encadrant

Acknowledgements

I would like to express deep gratitude to my supervisors, Prof. Karim Abed-Meraim and Prof. Nguyen Linh-Trung for their tremendous support and guidance. During my thesis, they not only share with me their solid knowledge to solve technical problems but also transmit to me their passion for science. Moreover, they give me freedom to choose my favorite subjects and work independently.

I would like to thank other collaborators with whom I have opportunity to work together : Dr. Rodolephe Weber, Prof. Adel Belouchrani, Prof. Philippe Ravier, and Dr. Abdelouahab Boudjellal. I am also thankful to the PRISME Laboratory staff for all the help.

I would like to thank the thesis committee members: Prof. Rachid Harba, Prof. Pierre Common, Prof. Roland Badeau, Prof. Philippe Forster, and Prof. Marco Di Renzo. They gave me many useful comments to improve the presentation of this thesis.

It has been a pleasure to share office and ideas with two lab-mates: Mohamed Nait Meziane and Abdelbassit Boualem.

I would like to acknowledge the fellowship and grant that kindly supported for my research: "Bourse Ministère de l' Enseignement Supérieur et de la Recherche" and NAFOSTED 102.02-2015.32 .

Finally, I would like to thank my family, my wife and my son. They always support me with constant love, encouragement, and sacrifice.

To my family,

and to my wife and son

Contents

List of Figures

List of Tables

Acronyms

PARAFAC	Parallel Factor Analysis
LOFAR	Low-Freqency-Array
RFI	Radio Frequency Interference
SVD	Singular Value Decomposition
EVD	Eigenvalue Decomposition
LBA	Low Band Antennas
HBA	High Band Antennas
MNS	Minimum Noise Subspace
GMNS	Generalized Minimum Noise Subspace
OPAST	Orthonormal Projection Approximation Subspace Tracking
PSA	Principal Subspace Analysis
MCA	Minor Component Analysis
MIMO	Multiple-Input-Multiple-Output
SKA	Square Kilometer Array
MUSIC	MUltiple SIgnal Classification
ESPRIT	Estimation of Signal Parameters via Rotational Invariance Techniques
PCS	Properly Connected Sequence

GPCS Generalized Properly Connected Sequence

PST Principal Subspace Tracking

MST Minor Subspace Tracking

PET Principal Eigenvector Tracking

DoA Direction of Arrival

SEP Subspace Estimation Performance

EEP Eigenvector Estimation Performance

SNR Signal-to-Noise Ratio

CANDECOMP Canonical Decompostion

HOOI Higher Order Orthogonal Iteration

HOSVD Higher-Order SVD

ALS Alternating Least-Squares

RS Random sampling

dGN damped Gauss-Newton

ADMM Alternating Direction Method of Multipliers

CRLB Cramer-Rao Lower Bound

NJD Non-symmetrical Joint Diagonalization

ASNP Sparse Nonegative PARAFAC

ASNT Sparse Non-negative Tucker

SOAP Second-Order Optimization based Adaptive PARAFAC

NSOAP Non-negative SOAP

STD Standard Deviations

RLS Recursive Least Squares

LMS Least Mean Squares

DS-CDMA Direct-Sequence Code-Division Multiple Access

fMRI Functional Magnetic Resonance Imaging

Chapter 1

Introduction

We are interested in fast matrix decomposition and its natural extension, tensor decomposition. Here, by "fast", we target at low-complexity and distributed/parallelizable algorithms, which allow the handling of massive data for both batch and adaptive settings. For matrix decomposition, particularly, we will focus on low-rank matrix decomposition, also known as subspace estimation in signal processing and array processing. For tensor decomposition, we will concentrate on low rank tensor decomposition where two widely used models, Parallel Factor Analysis (PARAFAC) and Tucker, will be considered.

Matrix and tensor decompositions are versatile tools used in diverse disciplines including signal processing, linear and multilinear algebra, data communication, data mining, machine learning, to name a few, because they help to process, model, analyze, understand or eventually compress data. We give concrete examples to illustrate this point from several perspectives. Figure 1.1 describes a radio-astronomy system including several Low-Freqency-Array (LOFAR) stations. After beamforming to the direction of interest, signals from all stations are transmitted to a central correlator to performs the correlation. Then applying beamforming to the covariance matrix (in all direction in the skype) can help to produce a skymap. However, the observa-

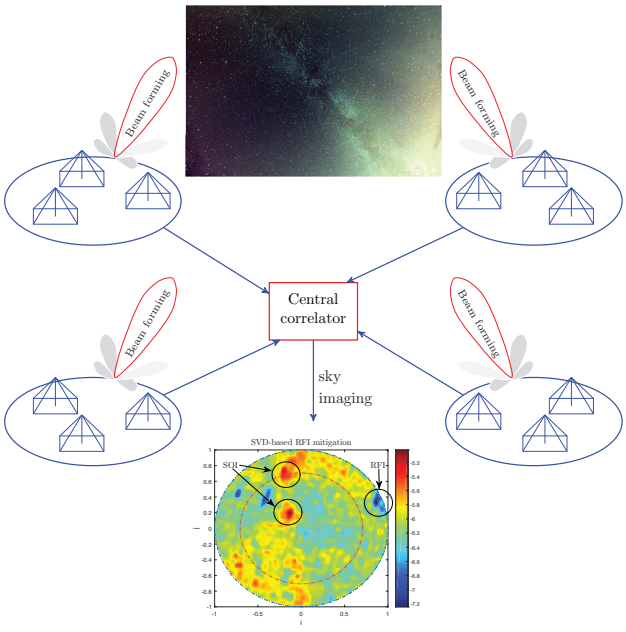


Figure 1.1: LOFAR stations.

tional signals is often accompanied by Radio Frequency Interference (RFI), leading to data loss and observation inefficiency. To mitigate RFI, subspace-based methods have been proven to be a simple and efficient way (see [1] and references therein). This approach estimates the principal subspaces using SVD of the covariance matrix and then build an orthogonal projection to eliminate RFIs (see Chapter 2 for a deeper analysis). For tensor case, we illustrate the efficiency of PARAFAC decomposition in EEG signal analysis (Figure 1.2). A multi-channel EEG record is transformed to the time-frequency domain to create a three-way tensor. A wavelet or time-frequency transform can be used in this situation. Then, PARAFAC decompose three-way tensor in terms of temporal, frequency and topography components. The information from those components can be exploited in the diagnosis process. On the other hand, from Table 1.1, it is straightforward to see that representation of data by matrix or tensor decomposition (i.e., number of parameters) is much less than data itself. Thus, matrix or tensor decomposition can serve as a compression tool when appli-

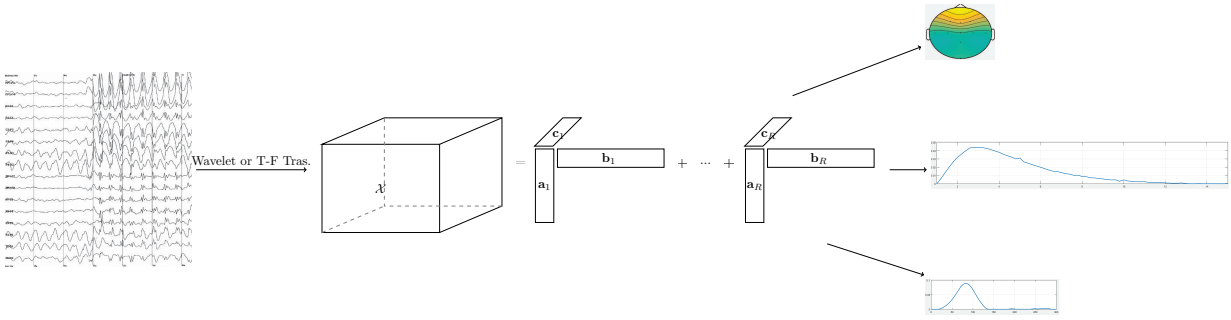


Figure 1.2: Time-Frequency representation of EEG signals and its PARAFAC decomposition.

	Raw data vs.	Representation
Number of parameters	mn	$mr + nr$ (Matrix decomposition)
	IJK	$IR + JR + KR$ (PARAFAC)
	IJK	$IP + JQ + KR + PQR$ (Tucker)

Table 1.1: A comparison on number of parameters of raw data and its presentation using matrix and tensor decomposition.

cable. In particular, a rank-r matrix of size $m \times n$, assume that $r \ll m, n$, has only $mr + nr$ parameters by way of truncated SVD. Similarly, we only need to consider $IR + JR + KR$ or $IP + JQ + KR + PQR$ parameters as a replacement for IJK ones if we can use PARAFAC or Tucker decomposition respectively. Here we assume that $P \ll I, Q \ll J$, and $R \ll K$.

However, the following common characteristics shared by many large-scale datasets make the problem exceedingly difficult [2] (i) extremely large on amount of data (up to million to billion entries) (ii) high dimensional to capture many aspects of the considered object (iii) time constraint or real time requirement for streaming data sources [3]. Sometimes, those characteristics can join together to form even harder issues. For examples, we will consider large-scale batch tensor data (thus, (i) and (ii)) in Chapter 4, and streaming tensor data (thus, (ii) and (iii)) in Chapter 6). Back to the example of radio-astronomy, a real implementation of this description is the LOFAR system [4] which is comprised of 18 core stations, 18 remote stations

and 8 international stations. Each station has 96 Low Band Antennas (LBA) and 48 High Band Antennas (HBA). Thus, even if we consider only the HBA and want to build a skymap from data of all stations, the number of sensors to process in total is quite large (more than 2000). In the EEG analysis example, if we take into account other essential aspects such as trials, conditions, subjects and groups, a huge amount of data can produce, making it difficult for traditional analysis tools to handle. A real-life example of large EEG dataset is MindBigData[1].

To tackle those problems, it naturally leads to using distributed/parallel computing and contemplating structure of the problem. However, how we can exploit the structure of problem as well as parallel computing efficiently for each specific task remains an open and challenging problem. Our work will focus on these two aspects and, thus, develop corresponding fast algorithms.

1.1 Thesis Overview and Contributions

1.1.1 Thesis overview

The structure of the thesis is divided into two main parts: fast matrix decomposition, and fast tensor decomposition. Next, we will describe their contents in detail.

Subspace estimation. In the first part (Chapter 2), we explore the problem of parallel/distributed subspace estimation for array processing applications. Subspace based methods are widely used because of its simplicity and accuracy. However, computational complexity of subspace estimation is also known to be expensive and inappropriate for large-scale problems. Thus, we use a divide-and-conquer approach and exploit parallel computing architectures to improve scalability of this task. In

[1]This dataset can be found at http://www.mindbigdata.com/opendb/.

particular, our method divides large data into smaller ones, estimates the principal and minor subspace from covariance matrices and fuses the local results into global one. Different batch and adaptive algorithms are taken into account for fast and parallel computation of both signal (principal) and noise (minor) subspaces. The method is motivated by the theory of minimum noise subspace (MNS) previously introduced in the context of blind channel identification; hence named Generalized MNS (GMNS). We then apply GMNS to RFI mitigation which is a challenging problem in radio astronomy. The result shows that GMNS represents an excellent trade-off between the computational gain and the subspace estimation accuracy, as compared to several standard subspace methods.

In the second part, we explore fast tensor decompositions for large scale data.

A short survey of fast tensor decompositions for big data processing. Before presenting the proposed algorithms of large scale-tensor decomposition , Chapter 3 presents a short survey on several recent state-of-the-art approaches for large-scale tensor data which is a crucial part of big data. We cover both batch and adaptive settings but limited to only PARAFAC and Tucker models.

Parallelizable PARAFAC decomposition. Inspired by the divide-and-conquer based GMNS method, we propose a parallelizable PARAFAC decomposition in Chapter 4. In the "divide" step, we take into account overlapping and non-overlapping cases. The simulation result reveals that the performance of the proposed algorithm is close to that of algorithm processing the whole data. Moreover, we also indicate that solving PARAFAC decomposition is essentially equivalent to solving non-symmetric joint diagonalization which is developed for blind source separation (BSS).

Robust tensor decompositions. Since determining tensor rank in advance is an NP-complete problem, with few exceptions, most tensor algorithms are computed with a guess (overestimated value) of the exact rank, known as the overfactoring.

Moreover, among various constraints, non-negativity and sparsity are the most popular ones because many problems come naturally with them such as for text, image, and EEG signal analysis. We thus adapt the all-at-once optimization framework which is robust to the overfactoring to impose either non-negativity constraint or both non-negativity and sparseness constraint in Chapter 5. It is shown that our proposed algorithms eliminate artifacts and improve accuracy as compared to the state-of-the-art algorithms. This work can be combined with the techniques developed in Chapter 4 to handle large-scale sparse and non-negative tensor decomposition.

Fast adaptive PARAFAC algorithms for streaming tensors. In Chapter 6, we develop two adaptive PARAFAC algorithms for streaming tensors that have one dimension growing with time. The main problem of the state-of-the-art algorithms is that their computational complexity is quadratic with respect to the tensor rank. Our algorithms, in contrast, achieve linear computational complexity while keeping the accuracy equivalent or even superior than the previous proposed algorithms. The first proposed algorithm generalizes the Orthonormal Projection Approximation Subspace Tracking (OPAST) approach along with a Khatri-Rao product constraint on each column of the estimated subspace. The second algorithm is based on an alternating least squares approach in conjunction with a Newton-type optimization technique. On top of that, they preserve the Khatri-Rao product approximately and exploits the reduced-rank update structure of the estimated subspace at each time instant. In addition, we also introduced adaptive non-negative PARAFAC decomposition and adapted one of the proposed algorithm to tackle this problem.

1.1.2 Publications

Most of the above results above have been published/submitted in the following papers:

Journal Papers

1. V.-D. Nguyen, K. Abed-Meraim, N. Linh-Trung, and R. Weber. Generalized Minimum Noise Subspace for Array Processing. *Submitted to IEEE Transactions on Signal Processing*, 2016.
2. V.-D. Nguyen, K. Abed-Meraim, and N. Linh-Trung. Second-Order Optimization based Adaptive PARAFAC Decomposition of Three-Way Tensors. *Submitted to Digital Signal Processing*, 2016

Conference Papers

1. V.-D. Nguyen, K. Abed-Meraim, N. Linh-Trung, and R. Weber. Generalized MNS Method for Parallel Minor and Principal Subspace Analysis. *22nd European Signal Processing Conference (EUSIPCO) 2013*, pages: 2265-2269, September 2014.
2. V.-D. Nguyen, K. Abed-Meraim, and N. Linh-Trung. Parallelizable PARAFAC Decomposition of 3-way Tensors. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2015*, pages : 5505-5509, April 2015.
3. V.-D. Nguyen, K. Abed-Meraim, and N. Linh-Trung. Fast Adaptive PARAFAC Decomposition Algorithm with Linear Complexity. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2016*, pages : 6235-6239, March 2016.
4. V.-D. Nguyen, K. Abed-Meraim, and N. Linh-Trung. New Robust Algorithms for Sparse Non-negative Three-way Tensor Decompositions. *24nd European Signal Processing Conference (EUSIPCO) 2016*, September 2016.
5. V.-D. Nguyen, K. Abed-Meraim, and N. Linh-Trung. Fast Tensor Decompositions for Big Data Processing (Invited paper). *The International Conference on Advanced Technologies for Communications (ATC) 2016*, October 2016.

1.1.3 Beyond This Thesis

I also contribute to two other works which are the outside of the scope of this thesis. The first one is the performance analysis of time-frequency sparsity prior to localization performance [5], in collaboration with Dr. Boudjellal, Prof. Abed-Meraim, Prof. Belouchrani and Prof. Ravier. The second is adaptive PARAFAC decomposition for third-order tensor completion [6], in collaboration with Mr. Truong, Prof. Abed-Meraim and Prof. Nguyen. We refer interested reader to those publications for more details.

Part I

Fast Matrix Decomposition

Chapter 2

Matrix decomposition

Contents

2.1 Introduction

Principal subspace analysis (PSA) and minor subspace analysis (MSA) play important roles in many practical signal processing applications such as high resolution parameter estimation [7], blind source separation [8], and radio frequency interference mitigation [1]. Important problems closely related to PSA and MSA are principal component analysis (PCA) and minor component analysis (MCA) that usually require eigen-subspaces of the data covariance matrix [9].

For batch systems, standard subspace techniques based on singular value decomposition (SVD) or eigenvalue decomposition (EVD) are often applied. Although these techniques have high performance advantages, they face high computational complexity, generally of $O(n^3)$ operations, where n is the dimension of the observation vector or the number of sensors. This complexity causes a serious handicap for large-dimensional systems such as large sensor networks [10], massive multiple-input-multiple-output (MIMO) systems [11], or large antenna arrays like the square kilometer array (SKA) used in radio astronomy [12]. In addition, SVD-like methods are not

suitable for adaptive subspace tracking because it requires repeated decompositions. In such a case, the use of distributed algorithms [13], parallel computation schemes [1], and fast adaptive techniques [14–16] becomes of high interest and generally leads to large gains in terms of computational complexity and memory requirements.

To concretely motivate our research, we now give an example about the LOw Frequency Array (LOFAR) telescope [4], which can be considered as a pathfinder for SKAs [12]. A LOFAR telescope provides a flexible way for all-sky monitoring. In such a system, to attain desired sensitivity (at least several orders of magnitude higher than most communication systems) and accurate spatial resolution, a huge amount of small phased arrays for future SKAs are used and distributed over a large area instead of a small number of big dishes. According to [12], several hundreds of mid- to high-frequency 15 m dishes will be located in South Africa and Africa, and several hundreds of thousands of low-frequency antennas will be located in Western Australia. Thus, it allows an observation of multiple directions at the same time on the sky at a reasonable cost. However, it also leads to challenging problems as follows. First, a massive amount of data, which are collected from some distributed stations and then transmitted to a data center, need to be processed. Second, due to high sensitivity, observed radio astronomical signals are very weak; they are typically 40 dB to over 100 dB weaker than signals from active services. If we suppose further that subspace approaches such as multiple signal classification (MUSIC) [17], estimation of signal parameters via rotational invariance techniques (ESPRIT) [18] and their variants are adopted, it is then problematic to deal with both computational complexity and "subspace fusion" of the massive data. The latter, similar to data fusion, is the process of integration of multiple subspaces representing the same real-world object into consistent, accurate, and useful representations of the "global" minor and principal subspaces.

From a technical point of view, suppose that we have at hand a parallel computing architecture with K computing units (A parallel computing architecture can be, but not limited to, a multi-core processor, a graphics processing unit (GPU) or a field-programmable gate array (FPGA) board.). The question of interest in this chapter is: How can we exploit this architecture to fuse subspaces, and then to achieve the "global" minor or principal subspaces, as well as to reduce the computational cost for extracting these subspaces? A simple and efficient subspace method called minimum noise subspace (MNS) was proposed in [19]. This method estimates the noise subspace via a set of noise vectors (a basis of the noise subspace) that can be computed *in parallel* from a set of tuples of system outputs. MNS is much more efficient because it avoids large-scale EVD/SVD computation in the standard subspace method. MNS has been applied to blind system identification [19], source localization [20], array calibration [20], multichannel blind image deconvolution [21], and adaptive subspace tracking [22]. However, the number of parallel computing units based on which MNS is implemented is the same as the dimension of the noise subspace and is, generally not equal to the actually available number of computing units (K) of the parallel computing architecture in use. In addition, the performance of MNS is degraded if the number of outputs is very large compared to that of the inputs.

In this work, we are not only to generalize MNS from [19] in such a way that we can handle MNS computation with a given K number of parallel computing units but also to handle principal subspace computation that was not dealt with in [19]. Our main contributions are summarized as follows.

1. Via the introduction of the concept of a generalized properly connected sequence (GPCS), we propose the GMNS concept and then several efficient GMNS-based algorithms for MSA and PSA, given a fixed number of parallel computing units. It is noted that we have disseminated partial results on GMNS in our con-

ference contribution [23], and here we provide detailed proofs and extensive experiments.

2. We then propose an algorithm to estimate the principal eigenpairs from the corresponding principal subspaces (i.e., solving the PCA problem) by solving a joint diagonalization problem.

3. We develop efficient adaptive versions of the above proposed GMNS-based batch algorithms for both principal subspace tracking (PST) and minor subspace tracking (MST) by integrating several existing (non-GMNS-based) subspace tracking algorithms in our parallel framework of GMNS. As will be shown, our GMNS-based adaptive algorithms have advantages in terms of computational complexity and convergence rate. It should be noted that some adaptive algorithms for MNS have already been proposed in [24] but they are limited to least minor subspace analysis.

4. We further propose efficient GMNS-based adaptive algorithms for principal eigenvector tracking (PET) from the corresponding GMNS-based PST algorithms. The performance of our algorithms is nearly identical to that of standard SVD-based algorithms and the computational complexity is lower.

5. We apply our GMNS-based PSA method to a real-life application– radio frequency interference (RFI) mitigation in radio astronomy.

The minor and principal subspaces can be exploited in different ways in different algorithms. For example, while MUSIC and its variants use the minor subspace for direction of arrival (DOA) estimation, ESPRIT and its variants use the principal subspace to achieve the desired result. In this work, GMNS considers both minor and principal subspace estimation, whereas MNS only considers minor subspace estimation. The potential of GMNS are two-fold. First, GMNS provides a framework that

allows the estimation of both minor and principal subspaces in parallel. Second, it can be used to fuse data from several data sources. In the practical application on RFI mitigation studied in Section 2.7, it will be shown that it is difficult or expensive to directly estimate the "global" principal subspace from the available data covariances of several stations.

The rest of this chapter is organized as follows. The MNS concept and its implementation are briefly reviewed in Section 2.2. The GMNS concept and the GMNS-based batch algorithms for MSA and PSA/PCA are proposed in Sections 2.3 and 2.4, respectively. GMNS-based adaptive algorithms are then developed in Section 2.5, with some details on the computational complexity. The performance of the proposed GMNS-based batch and adaptive algorithms is presented in Sections 2.6 and 2.7, including the real-life application on RFI mitigation.

2.2 Minimum Noise Subspace: A Review

Let us consider a general linear system with p inputs and n outputs ($p < n$), which obeys the following model for the input-output relationship:

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t), \tag{2.1}$$

where $\mathbf{x}(t) \in \mathbb{C}^n$ is the observation vector, $\mathbf{A} \in \mathbb{C}^{n\times p}$ is the system matrix of full column rank, $\mathbf{s}(t) \in \mathbb{C}^p$ is the random source vector, and $\mathbf{n}(t) \in \mathbb{C}^p$ is the additive white noise vector with unknown variance σ^2. The data covariance matrix is then given by

$$\mathbf{R}_{xx} = E\{\mathbf{x}(t)\mathbf{x}(t)^H\} = \mathbf{A}\mathbf{R}_{ss}\mathbf{A}^H + \sigma^2\mathbf{I}, \tag{2.2}$$

where $\mathbf{R}_{ss}$ is the source covariance matrix of full rank.

It is of interest to compute the minor subspace of $\mathbf{R}_{ss}$ for that MNS was proposed

in order to achieve fast computation in a parallel manner [19]. In particular, the concept of properly connected sequence (PCS) is defined and the system is re-organized into $n - p$ subsystems based on a selected PCS. Then, the minor subspace will be efficiently estimated by computing the least (noise) eigenvector, corresponding to the least eigenvalue, of each subsystem. The PCS concept is to guarantee that the noise vectors computed from the subsystems form a basis of the noise subspace. In the following, we will review the concept of PCS and the implementation of MNS.

2.2.1 Properly Connected Sequence

Denote the n system outputs by a set of members $m_1, m_2, \ldots, m_n$. Consider a sequence of $n - p$ subsets of outputs, wherein each subset i contains $p + 1$ members and is denoted by the $(p + 1)$-tuple $t_i = (m_{i_1}, m_{i_2}, \ldots, m_{i_{p+1}})$, $i = 1, \ldots, n - p$. This sequence is said to be *properly connected*, if the following conditions are satisfied:

$$\{m_{i_1}, \ldots, m_{i_p}\} \subset \{m_{j_k} \mid j < i, 1 \leqslant k \leqslant p + 1\},$$
$$m_{i_{p+1}} \notin \{m_{j_k} \mid j < i, 1 \leqslant k \leqslant p + 1\}.$$

These conditions mean that each tuple in the sequence has p members in common with its preceding tuples, along with another member not so. For instance, if we consider a system with $p = 2$ inputs and $n = 6$ outputs, the following sequence of $n - p = 4$ tuples is a PCS:

$$t_1 = (m_1, m_2, m_3); t_2 = (m_1, m_2, m_4);$$
$$t_3 = (m_2, m_3, m_5); t_4 = (m_2, m_5, m_6).$$

2.2.2 MNS Implementation

Now, given a PCS of $n-p$ tuples of system outputs, the (partial) observation vector $\mathbf{x}_i(t) = [x_{i_1}(t), x_{i_2}(t), \ldots, x_{i_{p+1}}(t)]^T$ of $p+1$ outputs corresponding to the i-th tuple has the following structure:

$$\mathbf{x}_i = \mathbf{A}_i\mathbf{s}(t) + \mathbf{n}_i(t), \tag{2.3}$$

and its covariance matrix is given by

$$\mathbf{R}_i = E\{\mathbf{x}_i(t)\mathbf{x}_i^H(t)\} = \mathbf{A}_i\mathbf{R}_{ss}\mathbf{A}_i^H + \sigma^2\mathbf{I}, \tag{2.4}$$

where $\mathbf{A}_i$ is the response matrix of the i-th subsystem and $\mathbf{n}_i(t)$ is the corresponding additive white noise. Each subsystem in (2.4) has a noise subspace of minimum dimension (i.e., equal to one), suggesting the naming of MNS.

From each $\mathbf{R}_i$, a noise vector $\mathbf{v}_i$ is constructed by first computing the least eigenvector $\hat{\mathbf{v}}_i$ of $\mathbf{R}_i$, and then zero-padding (ZP) it according to

$$\mathbf{v}_i(j) = \begin{cases} 0, & \text{if the } j\text{-th system output does} \\ & \text{not belong to the } i\text{-th tuple,} \\ \hat{\mathbf{v}}_i(j'), & \text{if the } j\text{-th system output is the} \\ & j'\text{-th entry of } i\text{-th tuple,} \end{cases} \tag{2.5}$$

for $1 \leqslant j \leqslant n$.

In practice, $\mathbf{R}_i$ is estimated by the sample average as $\hat{\mathbf{R}}_i = \frac{1}{T}\sum_t \mathbf{x}_i(t)\mathbf{x}_i^H(t)$, with T being the sample size. It is proved in [19] that the resulting set of noise vectors $\{\mathbf{v}_i\}$, for $1 \leqslant i \leqslant n-p$, forms a basis of the noise subspace. Figure 2.1 illustrates the MNS implementation.

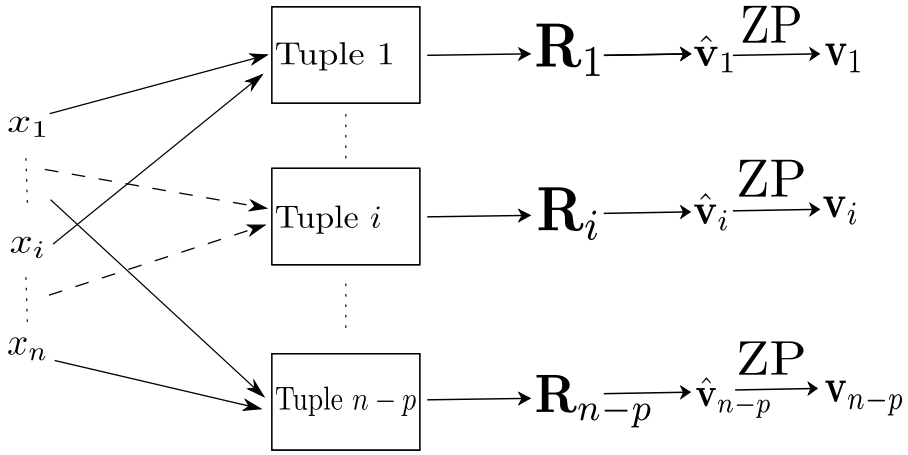


Figure 2.1: MNS implementation.

2.3 Minor Subspace Analysis using GMNS

As stated in Section 2.1, the objective of this work is to exploit the available parallel computing architecture, when having access to exactly K computing units, in order to reduce the computational cost for extracting the minor or principal subspaces of $\mathbf{R}_{xx}$. We approach this by generalizing MNS.

It is observed that each noise vector in MNS is estimated by the use of a minimum number of system outputs (i.e., p+1) which might lead to a non-negligible performance loss if $n \gg p$. In addition, to achieve the parallel computation of the noise vectors, $(n-p)$ computing units are needed, a number which depends on the impinging source number p and is usually a non-controllable system parameter.

Next, we propose a GMNS-based method for MSA, hereafter referred to as GMNS-MSA, to overcome the above-mentioned shortcomings. Given K computing units, we need to compute the $(n-p)$ noise vectors.

Let us write $n-p = dK + r$, where d and r are integers and $0 \leqslant r < K$, and for simplicity we assume that $r = 0$, i.e., $(n-p)/K$ is integer-valued. Now, we propose a concept of generalized properly connected sequence (GPCS), which generalizes the

PCS concept used in MNS.

Definition 1. *A GPCS is a sequence of K $(p+d)$-tuples $t_i = (m_{i_1}, \ldots, m_{i_{p+d}})$ for $1 \leqslant i \leqslant K$ that satisfies the following conditions:*

$$\{m_{i_1}, \ldots, m_{i_p}\} \subset \{m_{j_k} \mid j < i, 1 \leqslant k \leqslant p+d\}, \tag{2.6}$$

$$\{m_{i_{p+1}}, \ldots, m_{i_{p+d}}\} \notin \{m_{j_k} \mid j < i, 1 \leqslant k \leqslant p+d\}. \tag{2.7}$$

In other words, each tuple in the sequence has p members in common with its preceding tuples along with d other members not so.

Given a GPCS of K $(p+d)$-tuples of the outputs, the noise vectors are computed as follows. First, for each i-th subsystem, we compute the covariance matrix $\mathbf{R}_i$ of size $(p+d) \times (p+d)$, and hence its d least eigenvectors, represented by matrix $\hat{\mathbf{V}}_i$. Then, we construct the desired noise submatrix $\mathbf{V}_i$ according to the following rule[1]:

$$\mathbf{V}_i(j,:) = \begin{cases} 0, & \text{if the } j\text{-th system output does} \\ & \text{not belong to the } i\text{-th tuple,} \\ \hat{\mathbf{V}}_i(j',:), & \text{if the } j\text{-th system output is the} \\ & j'\text{-th entry of } i\text{-th tuple.} \end{cases} \tag{2.8}$$

Finally, we concatenate the K submatrices $\mathbf{V}_i$, $1 \leqslant i \leqslant K$, to have the global noise matrix $\mathbf{V} = [\mathbf{V}_1, \ldots, \mathbf{V}_K]$ whose columns form a basis of the noise subspace under the conditions given in the following theorem, with proof given in Appendix A.1.

Theorem 1. *Under the assumption that every p rows of $\mathbf{A}$ are linearly independent, the noise matrix $\mathbf{V}$ has full column rank (i.e., $\operatorname{rank}(\mathbf{V}) = n-p$) and hence its columns span the desired noise subspace of the data covariance matrix $\mathbf{R}_{xx}$.*

[1] In the sequel, $\mathbf{V}(j,:)$ refers to j-th row vector of $\mathbf{V}$.

The GMNS-MSA method is illustrated in Figure 2.2 and its implementation is summarized in Table 2.1.

The main advantage of GMNS-MSA is the reduction of the computational cost in comparison with the standard method. In particular, GMNS-MSA requires $O((p + (n-p)/K)^2 T)$ flops for the computation of the subsystem covariance matrices plus $O((p + (n-p)/K)^2 (n-p)/K)$ flops for the estimation of the least eigenvectors. Whereas, the standard method requires $O(n^2 T)$ flops for the estimation of the global covariance matrix plus $O(n^2(n-p))$ flops for the extraction of the noise vectors. Obviously, if $n \gg p$, the overall cost is reduced by almost a factor of K^2 for the covariance matrix estimation and a factor of K^3 for the noise subspace estimation.

Table 2.1: Summary of GMNS for MSA

Input: Observed data $\mathbf{X}$

1. Extract subsystems $\mathbf{X}_i$ as described in 2.3
2. Compute covariance matrix of each subsystem: $\mathbf{R}_i = \frac{1}{T}\mathbf{X}_i\mathbf{X}_i^H$
3. Extract d least eigenvectors of $\mathbf{R}_i$: $\hat{\mathbf{V}}_i = \text{eig}(\mathbf{R}_i)$
4. Construct desired noise matrix:

$$\mathbf{V}_i(j,:) = \begin{cases} 0, & \text{system output } j \notin \text{tuple } i \\ \hat{\mathbf{V}}_i(j',:), & \text{system output } j = \text{entry } j' \text{ of tuple } i \end{cases}$$

Output: Minor subspace weight matrix: $\mathbf{V} = [\mathbf{V}_1, \ldots, \mathbf{V}_K]$

Following are some remarks on the GMNS-MSA method.

Remark 1. Generally, $(n-p)/K$ is non integer-valued; that is, $0 < r < K$ for r in $n - p = dK + r$. In that case, we will use r tuples of length $p + d'$, with $d' = d + 1$, and $K - r$ tuples of length $p + d$.

Remark 2. It should be noted that the GPCS concept is just a practical way to

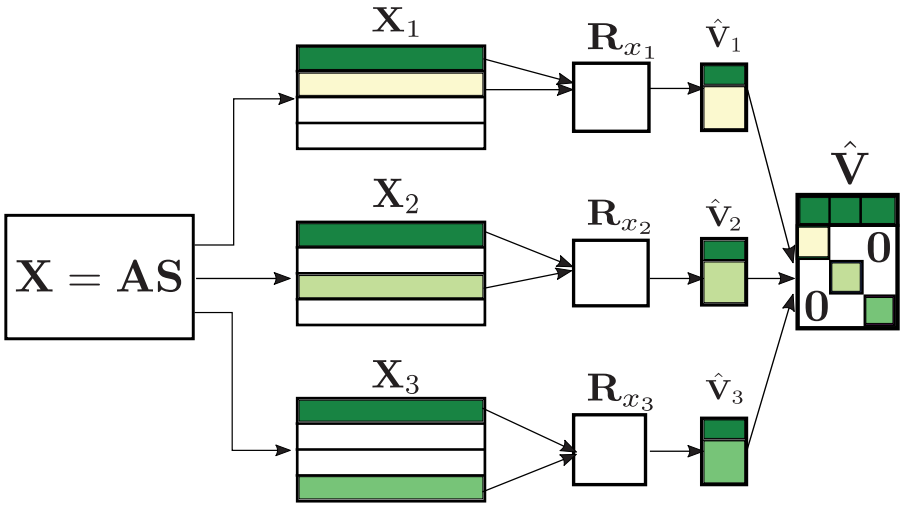


Figure 2.2: GMNS for MSA, with $K = 3$ subsystems. The green (bold) part represents the p outputs shared by three subsystems.

guarantee that the obtained vectors form a basis of the desired subspace. In other words, GPCS does not represent necessary conditions to meet but only sufficient conditions.

Remark 3. For large dimensional systems when $n \gg p$, using only $p + 1$ system outputs as in the original MNS to compute a noise vector may result in non-negligible performance loss. Now if $\lfloor (n-p)/K \rfloor = d$ is relatively large, we will instead use $p+d+1$ outputs to estimate a given noise vector which improves its estimation accuracy, as later shown in Figure 2.5. It also means that we need $K < (n-d)/p$ so that the size of subsystems is larger than p.

Remark 4. When $p \ll n$, one way to estimate the minor subspace is to first estimate the signal subspace (with a low cost) and then obtain the noise subspace as its orthogonal complement. However, it is quite expensive to compute the orthogonal complement subspace; i.e., $O(p^2 n)$. In addition, extraction of an orthogonal basis of the noise subspace would require a further cost.

Remark 5. So far, we have assumed that p is known in advance. When this assumption is violated, many subspace algorithms still work well in practice if we replace p with

an overestimated value, for example in the MUSIC algorithm. In such a case, the value used by the algorithm can be just a guess (overestimated value) of the exact value p.

Remark 6. In [25], a fast subspace estimation method was proposed, exploiting the spatial whiteness of the additive noise. In this work, we exploit this property together with parallel computing to achieve a much higher computational gain.

2.4 Principal Subspace Analysis and Principal Component Analysis using GMNS

The original MNS was dedicated to MSA and above we have proposed a GMNS-based method for MSA. In this section, we propose a GMNS-based method for PSA using K subsystems in a parallel manner. This method is hereafter referred to as GMNS-PSA. In particular, we proposed algorithms for overlapping and non-overlapping subsystems respectively. In addition, we extend the method for PCA, which is hereafter referred to as GMNS-PCA.

2.4.1 Principal Subspace Analysis using GMNS

2.4.1.1 Subsystems without Overlapping Parts

Let us assume that we have a large dimensional system such that $l = n/K > p$ and, for simplicity, l is integer-valued. We divide the n system outputs into K subsystems of length l each represented by

$$(m_{(i-1)l+1}, \ldots, m_{il}), \quad i = 1, \ldots, K.$$

Now, for each subsystem i, we compute the corresponding covariance matrix $\mathbf{R}_i$ and its principal subspace matrix $\mathbf{W}_i = \mathbf{A}_i\mathbf{Q}_i$, where $\mathbf{Q}_i$ is an unknown nonsingular matrix of size $p \times p$.

To have a global estimate of the signal subspace (i.e., a matrix $\mathbf{W} = \mathbf{AQ}$ of size $n \times p$ where $\mathbf{Q}$ is any $p \times p$ nonsingular matrix), we need to get rid of the unknown matrices $\mathbf{Q}_i$. For that, we exploit the fact that all subsystems receive the same source $\mathbf{S}$ of size $p \times T$, that is,

$$\mathbf{X}_i = \mathbf{A}_i\mathbf{S} + \mathbf{N}_i, \quad i = 1, \ldots, K, \tag{2.9}$$

where $\mathbf{S} = [\mathbf{s}(1), \cdots, \mathbf{s}(T)]$ and $\mathbf{N}_i$ is noise affecting the i-th subsystem. Let us define

$$\mathbf{S}_i = \mathbf{W}_i^{\#}\mathbf{X}_i, \tag{2.10}$$

where $\#$ denotes the pseudo-inverse operator[1]. Then, thanks to (2.9), we have

$$\mathbf{S}_i = \mathbf{Q}_i^{-1}\mathbf{S} + \mathbf{W}_i^{\#}\mathbf{N}_i, \quad i = 1, \ldots, K. \tag{2.11}$$

In the noiseless case, it can be shown that

$$\mathbf{S}_i = \mathbf{T}_i\mathbf{S}_1,$$

where $\mathbf{T}_i = \mathbf{Q}_i^{-1}\mathbf{Q}_1$. Matrix $\mathbf{T}_i$ can be estimated by solving the following least square problem:

$$\min_{\mathbf{T}_i} \| \mathbf{S}_i - \mathbf{T}_i\mathbf{S}_1 \|_2^2 .$$

[1]Most subspace estimation methods compute an orthonormal basis of the desired subspace (see, e.g., [26]) in which case we have $\mathbf{W}_i^{\#} = \mathbf{W}_i^H$.

Its solution is given by

$$\hat{\mathbf{T}}_i = \mathbf{S}_i \mathbf{S}_1^{\#}. \tag{2.12}$$

Finally, the principal subspace weight matrix is obtained as

$$\mathbf{W} = \left[\mathbf{W}_1^T, (\mathbf{W}_2\mathbf{T}_2)^T, \ldots, (\mathbf{W}_K\mathbf{T}_K)^T\right]^T \approx \mathbf{A}\mathbf{Q}_1. \tag{2.13}$$

In the noisy case, the estimate of the principal subspace weight matrix in (2.13) is *biased* due to the effect of the noise term on the estimation of $\mathbf{T}_i$ in (2.12). In fact, (2.12) can be rewritten as

$$\hat{\mathbf{T}}_i = \left(\frac{\mathbf{S}_i\mathbf{S}_1^H}{T}\right)\left(\frac{\mathbf{S}_1\mathbf{S}_1^H}{T}\right)^{-1} \tag{2.14}$$

$$\approx (\mathbf{Q}_i^{-1}\hat{\mathbf{R}}_{ss}\mathbf{Q}_1^{-H})(\mathbf{Q}_1^{-1}\hat{\mathbf{R}}_{ss}\mathbf{Q}_1^{-H} + \sigma^2\mathbf{I})^{-1}. \tag{2.15}$$

Here we have substituted (2.11) into (2.14), and used the facts that the subsystems are non-overlapping, their noise terms are uncorrelated (spatially white noise assumption) and $\mathbf{W}_i$ are unitary matrices that leads to $E[\mathbf{W}_1^H\mathbf{n}_1(t)\mathbf{n}_1^H(t)\mathbf{W}_1] = \sigma^2\mathbf{I}$ and, hence, $\mathbf{W}_1^H(\mathbf{N}_1\mathbf{N}_1^H/T)\mathbf{W}_1 \approx \sigma^2\mathbf{I}$. Moreover, we remind that the signal and the noise are uncorrelated, i.e., $E[\mathbf{S}\mathbf{N}_i^H] = 0$ and hence $\mathbf{S}\mathbf{N}_i^H/T \approx 0$. Because of the additive term $\sigma^2\mathbf{I}$, $\hat{\mathbf{T}}_i$ deviates from its desired value and leads to an estimation bias for the global weight matrix, especially, at low signal-to-noise ratio (SNR).

To overcome this problem, we replace the previous estimate of $\mathbf{T}_i$ by the following asymptotically unbiased estimate:

$$\tilde{\mathbf{T}}_i = \left(\frac{\mathbf{S}_i\mathbf{S}_1^H}{T}\right)\left(\frac{\mathbf{S}_1\mathbf{S}_1^H}{T} - \hat{\sigma}^2\mathbf{I}\right)^{-1}$$

$$= (\mathbf{W}_i^H\mathbf{R}_{i,1}\mathbf{W}_1)(\mathbf{W}_1^H\mathbf{R}_1\mathbf{W}_1 - \hat{\sigma}^2\mathbf{I})^{-1}, \tag{2.16}$$

where

$$\mathbf{R}_{i,1} = E[\mathbf{x}_i(t)\mathbf{x}_1^H(t)] \tag{2.17}$$

$$\hat{\sigma}^2 = [\mathrm{Tr}(\mathbf{R}_1) - \mathrm{Tr}(\mathbf{W}_1^H\mathbf{R}_1\mathbf{W}_1)]/(l-p). \tag{2.18}$$

To obtain the noise power estimate in (2.18), we first obtain the EVD of $\mathbf{R}_1$ as

$$\begin{aligned}\mathbf{R}_1 &= \mathbf{A}_1\mathbf{R}_{ss}\mathbf{A}_1^H + \sigma^2\mathbf{I} \\ &= \mathbf{U}_s(\mathbf{\Lambda}_s + \sigma^2\mathbf{I})\mathbf{U}_s^H + \sigma^2\mathbf{U}_n\mathbf{U}_n^H,\end{aligned} \tag{2.19}$$

in which the columns of $\mathbf{U}_s$ span the signal subspace of $\mathbf{R}_1$ and those of $\mathbf{U}_n$ span the noise subspace of $\mathbf{R}_1$. From (2.19) and using $\mathbf{W}_1^H\mathbf{U}_n = \mathbf{0}$, we then have

$$\mathbf{W}_1^H\mathbf{R}_1\mathbf{W}_1 = \mathbf{W}_1^H\mathbf{U}_s(\mathbf{\Lambda}_s + \sigma^2\mathbf{I})\mathbf{U}_s^H\mathbf{W}_1. \tag{2.20}$$

Since $\mathbf{W}_1^H\mathbf{U}_s$ is unitary, we have $\mathrm{Tr}(\mathbf{R}_1 - \mathbf{W}_1^H\mathbf{R}_1\mathbf{W}_1) = (l-p)\sigma^2$, and thus (2.18).

The above biased and unbiased algorithms are referred to as GMNS-N-PSA (N stands for non-overlapping) and GMNS-NU-PSA (NU stands for non-overlapping and unbiased), respectively. They are summarized in Table 2.2. Figure 2.3 illustrates the steps described in this section.

2.4.1.2 Subsystems with Overlapping Parts

In the above non-overlapping case, we have assumed that $n/K > p$. To relax this assumption and extend our method to cover also the case[1] where $n/K < p$, we consider here overlapping subsystems of size $p+q$ sharing q system outputs, and represented

[1]When using overlapping subsystems, we can deal with either $n/K \geqslant p$ or $n/K < p$, while in the non-overlapping case, it is necessary to have $n/K > p$.

Table 2.2: Summary of GMNS for PSA: non-overlapping and overlapping subsystems

Input: Captured data $\mathbf{X}$

1. Extract non-overlapping subsystems $\mathbf{X}_i$ as described in 2.4.1.1, or overlapping ones as in 2.4.1.2
2. Compute covariance matrix of each subsystem: $\mathbf{R}_i = \frac{1}{T}\mathbf{X}_i\mathbf{X}_i^H$.
3. Extract p principal eigenvectors of $\mathbf{R}_i$: $\mathbf{W}_i = \text{eig}(\mathbf{R}_i, p)$

4*) *Non-overlapping*:

Compute $\tilde{\mathbf{T}}_i$

- Biased: $\mathbf{S}_i = \mathbf{W}_i^{\#}\mathbf{X}_i, \quad \tilde{\mathbf{T}}_i = \mathbf{S}_i\mathbf{S}_1^{\#}$.
- Unbiased: $\hat{\sigma}^2 = [\text{Tr}(\mathbf{R}_1) - \text{Tr}(\mathbf{W}_1^H\mathbf{R}_1\mathbf{W}_1)]/(l-p)$, $\tilde{\mathbf{T}}_i = \left(\frac{\mathbf{S}_i\mathbf{S}_1^H}{T}\right)\left(\frac{\mathbf{S}_1\mathbf{S}_1^H}{T} - \hat{\sigma}^2\mathbf{I}\right)^{-1}$.

4**) *Overlapping*:

Extract overlap: $\mathbf{W}_1^{\text{olap}}(t)$, $\mathbf{W}_i^{\text{olap}}(t)$

Compute: $\mathbf{T}_i = (\mathbf{W}_i^{\text{olap}})^{\#}(\mathbf{W}_1^{\text{olap}}), i = 2, \ldots, K$

Compute principal weight matrix: $\tilde{\mathbf{W}}_1 = \mathbf{W}_1^{\text{olap}}$, $\tilde{\mathbf{W}}_i = \mathbf{W}_i^{\text{olap}}\mathbf{T}_i$

Output: Principal subspace

- *Non-overlapping*: $\mathbf{W} = [\mathbf{W}_1^T, (\mathbf{W}_2\tilde{\mathbf{T}}_2)^T, \ldots, (\mathbf{W}_K\tilde{\mathbf{T}}_K)^T]^T$
- *Overlapping*: $\mathbf{W} = [\tilde{\mathbf{W}}_1^T, \tilde{\mathbf{W}}_2^T, \ldots, \tilde{\mathbf{W}}_K^T]^T$

by the K tuples. For example, here, we choose all subsystems which overlap with the first one:

$$(m_1, \ldots, m_q, m_{q+1}, \ldots m_{p+q}),$$
$$(m_1, \ldots, m_q, m_{(i-1)d+1}, \ldots, m_{id}), \quad i = 2, \ldots, K.$$

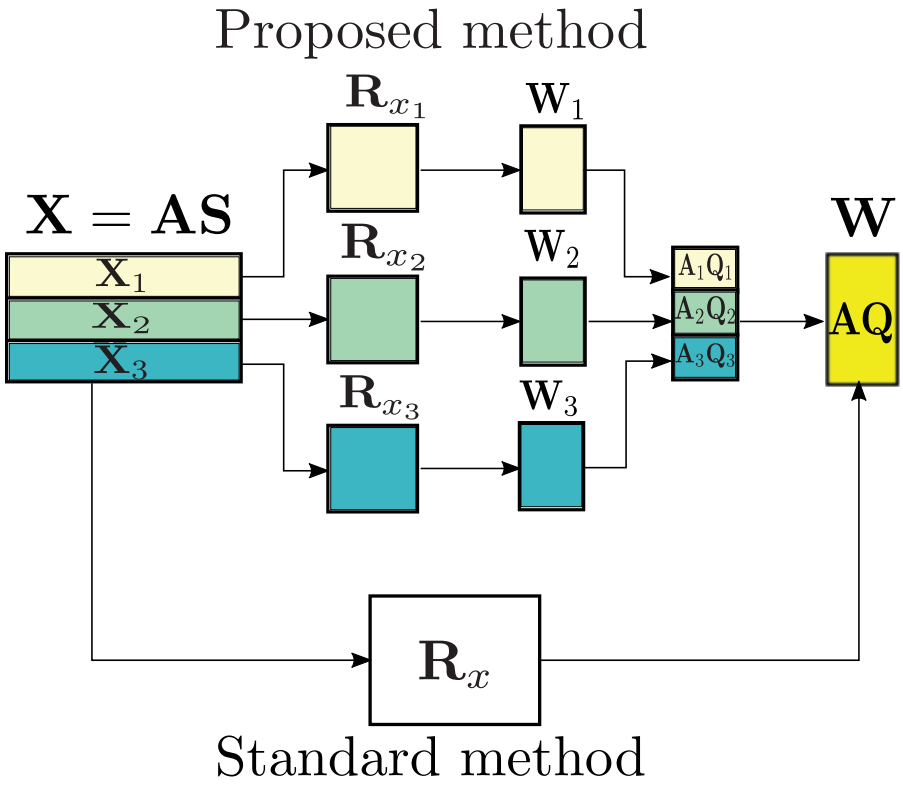


Figure 2.3: GMNS for PSA with non-overlapping parts; $K = 3$.

In other words, the q first members of the first subsystem are the q first members of the i-th subsystem, for $i = 2, \ldots, K$. For simplicity, we assume that $d = (n-p)/K$ is integer-valued. Now, for each subsystem, we compute the covariance matrix $\mathbf{R}_i$ and its corresponding weight matrix $\mathbf{W}_i$ which can be written as

$$\mathbf{W}_i = \begin{bmatrix} \mathbf{W}_i^{\text{olap}} \\ \mathbf{W}_i' \end{bmatrix} = \begin{bmatrix} \mathbf{A}_i^{\text{olap}} \\ \mathbf{A}_i' \end{bmatrix} \mathbf{Q}_i. \tag{2.21}$$

To get rid of the matrices $\mathbf{Q}_i$, one exploits the overlap between the first subsystem and the i-th subsystem by assuming that any $p \times p$ submatrix of $\mathbf{A}$ has full rank. In that case, the global weight matrix is estimated as

$$\mathbf{W} = \left[\tilde{\mathbf{W}}_1^T, \tilde{\mathbf{W}}_2^T, \ldots, \tilde{\mathbf{W}}_K^T\right]^T, \tag{2.22}$$

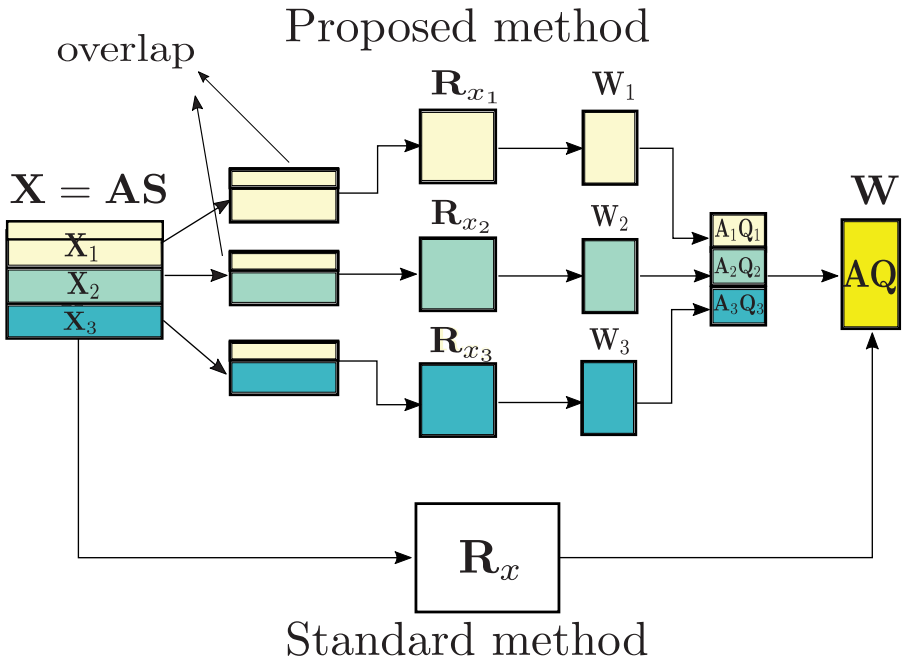


Figure 2.4: GMNS for PSA with overlapping parts; $K = 3$.

where

$$\tilde{\mathbf{W}}_1 = \mathbf{W}_1, \tag{2.23}$$

$$\tilde{\mathbf{W}}_i = \mathbf{W}_i \mathbf{T}_i, \quad i = 2, \ldots, K, \tag{2.24}$$

$$\mathbf{T}_i = (\mathbf{W}_i^{\text{olap}})^{\#} (\mathbf{W}_1^{\text{olap}}). \tag{2.25}$$

This algorithm is referred to as GMNS-O-PSA (where O stands for overlapping) and is summarized in Table 2.2 and illustrated in Figure 2.4.

2.4.1.3 Complexity

Similar to GMNS-MSA in Section 2.3, the main advantage of the proposed GMNS-based PSA algorithms resides in its reduced computational cost. GMNS-NU-PSA costs $O((n/K)^2 p + p^2(n/K + p))$ flops for the computation of p signal subspace vectors

and $O(2(n/K)^2T)$ flops for the computation of the covariance matrices $\mathbf{R}_i$ and the correlation matrices $\mathbf{R}_{i,1}$, for $i = 1, \ldots, K$. This overall cost is approximately K^2 less than the cost of a direct computation of the signal subspace using the global covariance matrix, which takes $O(n^2(T+p))$ flops. GMNS-O-PSA costs $O((p+q)^2p+p^2(2p+q))$ flops for the computation of p signal subspace vectors and $O((p+q)^2T)$ flops for the parallel computation of the covariance matrices. If $n \gg p$, $n \gg K$ and $T \gg 1$, then $q \approx n/K$, and hence GMNS-O-PSA is slightly cheaper than GMNS-NU-PSA since it does not require computing the correlation matrices[1].

2.4.2 Principal Component Analysis using GMNS

Having estimated the principal subspace by the GMNS-PSA method as presented in Section 2.4.1, we can further extract the principal eigenvalues and eigenvectors of the covariance matrix $\mathbf{R}_{xx}$, resulting in a GMNS-based method for PCA. This method is hereafter referred to as GMNS-PCA and is described below.

First, we note that

$$\mathbf{W} = \mathbf{A}\mathbf{Q} = \mathbf{U}_s\tilde{\mathbf{Q}}, \tag{2.26}$$

where the columns of $\mathbf{U}_s$ are p principal eigenvectors of $\mathbf{R}_{xx}$, and $\tilde{\mathbf{Q}}$ is a non-singular matrix of size $p \times p$.

Now, consider the following expression:

$$\mathbf{W}^{\#}\mathbf{R}_{xx}\mathbf{W} = \tilde{\mathbf{Q}}^{-1}(\mathbf{U}_s^H\mathbf{R}_{xx}\mathbf{U}_s)\tilde{\mathbf{Q}} \tag{2.27}$$

$$= \tilde{\mathbf{Q}}^{-1}\begin{bmatrix} \lambda_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \lambda_p \end{bmatrix}\tilde{\mathbf{Q}}, \tag{2.28}$$

[1]For certain applications, e.g., radio astronomy, the global covariance matrix is available (or already computed) for other needs. Hence, GMNS-NU-PSA becomes more advantageous than GMNS-O-PSA in term of computational complexity.

Table 2.3: Summary of GMNS for PCA

Input: Principal subspace $\mathbf{W}$

1. Compute expression: $\mathbf{W}^{\#}\mathbf{R}_{xx}\mathbf{W}$
2. Solve diagonalization problem to find $\tilde{\mathbf{Q}}^{-1}$ and $\{\lambda_i\}$
3. Extract p principal eigenvectors: $\mathbf{U}_s = \mathbf{W}\tilde{\mathbf{Q}}^{-1}$

Output: Principal eigenpairs: $\mathbf{U}_s$ and $\{\lambda_i\}_{i=1:p}$

where $\lambda_1, \ldots, \lambda_p$ are the principal eigenvalues of $\mathbf{R}_{xx}$.

Therefore, the principal eigenvectors and eigenvalues can be found by computing matrix $\tilde{\mathbf{Q}}$ that diagonalizes the matrix $\mathbf{W}^{\#}\mathbf{R}_{xx}\mathbf{W}$.

We note that since GMNS-PCA estimates the principal eigenvectors from the principal subspace obtained by GMNS-PSA, the eigenvectors are arranged in descending order of the corresponding eigenvalues, i.e., $\lambda_1 > \lambda_2 > \cdots > \lambda_p$. Therefore, the performance of principal eigenvector estimation depends on that of principal subspace estimation. The whole method is summarized in Table 2.3.

2.5 Adaptive GMNS-based Algorithms

In this section, we are interested in estimating the minor and principal subspaces at each time index t from streaming observations $\{\mathbf{x}(t)\}_{t\geqslant 1}$.

2.5.1 Minor Subspace Tracking using GMNS

Thanks to the parallel structure of GMNS-MSA, the conversion from a batch system to an adaptive one is quite simple. This leads to our proposed GMNS-based method for MST, which is hereafter referred to as GMNS-MST.

In practice, to track the underlying minor subspace, we replace the SVD-based

Table 2.4: Summary of GMNS for MST

Initialization
Input: Observed data at snapshot t: $\mathbf{x}(t)$

1. Extract subsystems $\mathbf{x}_i(t)$ as described in 2.3
2. Track minor subspace of each subsystem using existing algorithms (e.g. FOOja, FDPM):

$$\hat{\mathbf{V}}_i(t) = \text{TrackingAlgorithm}(\mathbf{V}_i(t-1), \mathbf{x}_i(t))$$

3. Construct the desired noise matrix $\mathbf{V}_i(t)$:

$$\mathbf{V}_i(j,:) = \begin{cases} 0, & \text{if system output } j \notin \text{tuple } i \\ \hat{\mathbf{V}}_i(j',:), & \text{if system output } j = \text{entry } j' \text{ of tuple } i \end{cases}$$

Output: Minor subspace $\mathbf{V}(t) = [\mathbf{V}_1(t), \ldots, \mathbf{V}_K(t)]$

computation of the minor subspace from the correlation matrix (i.e., Step 2 in Table 2.1) with any existing MST algorithms while keeping the remaining steps unchanged. In our GMNS-MST, we integrate the FOOja algorithm [27] and the FDPM algorithm [16].

It should be noted that the estimation of the covariance matrix (i.e., Step 1 in Table 2.1) is not required in FOOja and FDPM. As a result, GMNS-MST provides a way to reduce the complexity of the algorithms by a factor of K. The implementation of GMNS-MST is summarized in Table 2.4.

Because of the way we construct the desired noise matrix (i.e., Step 3 in Table 2.4), the computational cost of GMNS-MST equals the cost of the tracking algorithm in use reduced by a factor of K. For example, it costs $O(np/K)$ if FOOja or FDPM is used.

2.5.2 Principal Subspace Tracking using GMNS

By extending GMNS-PSA as given in Section 2.4.1 to adaptive processing, we now propose a GMNS-based method for PST, which is hereafter referred to as GMNS-PST. Similar to GMNS-PSA, we deal with two cases whether we have the subsystems with or without overlapping parts.

2.5.2.1 Subsystems without Overlapping Parts

We observe that matrix $\mathbf{T}_i$ can be expressed by

$$\mathbf{T}_i = \mathbf{W}_i^{\#}(\mathbf{X}_i\mathbf{X}_1^{\#})\mathbf{W}_1. \tag{2.29}$$

By this way, we can track the principal subspace $\mathbf{W}_i$ of each subsystem and then compute the global weight matrix $\mathbf{W}$ as in (2.13).

Because calculating the term $\mathbf{X}_i\mathbf{X}_1^{\#}$ in full scale is expensive and not suitable for adaptive processing, we propose to use a sliding window technique to overcome this problem. Denote by N the window size. At time instant t, the subsystem $\mathbf{X}_i(t)$ can be written as

$$\mathbf{X}_i(t) = [\mathbf{x}_i(t-(N-1)) \quad \cdots \quad \mathbf{x}_i(t)]. \tag{2.30}$$

In our GMNS-PST, we use the sliding window version of the orthogonal projection approximation subspace tracking (OPAST) algorithm [28].

Now, to update $\mathbf{T}_i(t)$ efficiently, we first rewrite the term $\mathbf{X}_i(t)\mathbf{X}_1^{\#}(t)$ as

$$\mathbf{X}_i(t)\mathbf{X}_1^{\#}(t) = \mathbf{P}_i(t)\mathbf{M}^{-1}(t), \tag{2.31}$$

where

$$\mathbf{P}_i(t) = \mathbf{X}_i(t)\mathbf{X}_1^H(t),$$
$$\mathbf{M}(t) = \mathbf{X}_1(t)\mathbf{X}_1^H(t).$$

Then, we obtain

$$\begin{aligned}\mathbf{P}_i(t) &= \sum_{\tau=0}^{N-1} \mathbf{x}_i(t-\tau)\mathbf{x}_1^H(t-\tau) \\ &= \mathbf{P}_i(t-1) + \mathbf{x}_i(t)\mathbf{x}_1^H(t) - \mathbf{x}_i(t-N)\mathbf{x}_1^H(t-N), \qquad (2.32)\\ \mathbf{M}(t) &= \sum_{\tau=0}^{N-1} \mathbf{x}_1(t-\tau)\mathbf{x}_1^H(t-\tau) \\ &= \mathbf{M}(t-1) + \mathbf{x}_1(t)\mathbf{x}_1^H(t) - \mathbf{x}_1(t-N)\mathbf{x}_1^H(t-N). \qquad (2.33)\end{aligned}$$

Since $\mathbf{M}(t)$ has a rank-2 update structure, it can be efficiently inverted by applying the matrix inversion lemma to yield $\mathbf{M}^{-1}(t)$. However, substituting (2.32) and (2.33) into (2.31) and hence into (2.29) still includes expensive matrix-matrix multiplications. Fortunately, many fast tracking algorithms have a rank-1 update structure (e.g., the OPAST algorithm [15]) expressed in the form of

$$\mathbf{W}(t) = \mathbf{W}(t-1) + \mathbf{p}(t)\mathbf{q}^H(t). \qquad (2.34)$$

Thus, we can use this observation to compute (2.29) recursively with only matrix-vector multiplications. To initialize $\mathbf{P}_i(0)$ and $\mathbf{M}(0)$, we can either choose them arbitrarily or use the N first snapshots in a batch way.

This algorithm is referred to as GMNS-N-PST and summarized in Table 2.5.

Remark 7. Recall that in GMNS-PSA we have developed both biased and unbiased estimators of $\mathbf{T}_i$. However, in adaptive processing, we have observed that the per-

formance improvement of PST in the unbiased case is negligible as compared to the biased case. Thus, we only presented the latter, as above, due to its simplicity.

2.5.2.2 Subsystems with overlapping parts

Similar to modifying GMNS-MST to deal with the case where the subsystems have overlapping parts, at each time instant in the adaptive version of GMNS-O-PSA, we simply replace the SVD-based computation of the principal subspace from the correlation matrix (i.e., Steps 2 and 3 in Table 2.2) with any existing PST algorithms while keeping the remaining steps unchanged. Again, a similar observation for efficiently computing $\mathbf{T}_i$ in the non-overlapping case can also be applied here. This algorithm is referred to as GMNS-O-PST and summarized in Table 2.5.

2.5.3 Principal Eigenvector Tracking using GMNS

Here we present an adaptive version of GMNS-PCA, as described in Section 2.4.2, to track the principal eigenvectors from the estimated principal subspace. It is hereafter referred to as GMNS-PET.

According to (2.27), at time instant t, the following diagonalization is performed:

$$\mathbf{W}^{\#}(t)\mathbf{R}_{xx}(t)\mathbf{W}(t) = \tilde{\mathbf{Q}}^{-1}(t)\begin{bmatrix} \lambda_1(t) & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \lambda_p(t) \end{bmatrix}\tilde{\mathbf{Q}}(t). \tag{2.35}$$

Then, the principal eigenvectors are found to be

$$\mathbf{U}_s(t) = \mathbf{W}(t)\tilde{\mathbf{Q}}^{-1}(t). \tag{2.36}$$

Table 2.5: Summary of GMNS for PST

Initialization
Input: Observed data at snapshot t: $\mathbf{x}(t)$

1. Extract non-overlapping subsystems $\mathbf{x}_i(t)$ as described in 2.4.1.1, or overlapping ones as described in 2.4.1.2

2. Track principal subspace of each subsytem using existing algorithms (e.g. YAST, OPAST):

$$\mathbf{W}_i(t) = \text{TrackingAlgorithm}(\mathbf{W}_i(t-1), \mathbf{x}_i(t))$$

3*) *Non-overlapping*:

Compute recursive updates of $\mathbf{P}_i(t)$ and $\mathbf{M}(t)^{-1}$

Compute matrix: $\mathbf{T}_i(t) = \mathbf{W}_i^{\#}(t)(\mathbf{P}_i(t)\mathbf{M}^{-1}(t))\mathbf{W}_1(t)$

Compute principal weight matrix:

$$\mathbf{W}(t) = \left[\mathbf{W}_1^T(t), (\mathbf{W}_2^T(t)\mathbf{T}_2^T(t)), \ldots, (\mathbf{W}_K^T(t)\mathbf{T}_K^T(t))\right]^T$$

3**) *Overlapping*:

Extract overlap: $\mathbf{W}_1^{\text{olap}}(t)$, $\mathbf{W}_i^{\text{olap}}(t)$

Compute matrix:

$$\mathbf{T}_i(t) = (\mathbf{W}_i^{\text{olap}}(t))^{\#}(\mathbf{W}_1^{\text{olap}}(t)), i = 2, \ldots, K$$

Compute principal weight matrix:

$$\tilde{\mathbf{W}}_i(t) = \mathbf{W}_i(t)\mathbf{T}_i(t), \quad \text{with } \tilde{\mathbf{W}}_1(t) = \mathbf{W}_1(t)$$

Output: Principal subspace $\mathbf{W}(t)$ and $\tilde{\mathbf{W}}_i(t)$

A naive implementation of the above estimation is not numerically efficient. Ob-

served that (2.27) can be written as

$$\begin{aligned}\mathbf{W}^{\#}(t)\mathbf{R}_{xx}(t)\mathbf{W}(t) &\cong E\{\mathbf{W}^{\#}(t)\mathbf{x}(t)\mathbf{x}(t)^{H}\mathbf{W}(t)\} \\ &\cong E\{\tilde{\mathbf{y}}(t)\mathbf{y}^{H}(t)\},\end{aligned}$$

where

$$\tilde{\mathbf{y}}(t) = \mathbf{W}^{\#}(t)\mathbf{x}(t), \tag{2.37}$$

$$\mathbf{y}(t) = \mathbf{W}^{H}(t)\mathbf{x}(t). \tag{2.38}$$

The main cost thus comes from the calculation of the pseudo-inverse operator in (2.37), which is then expressed as

$$\tilde{\mathbf{y}}(t) = [\mathbf{W}(t)^{H}\mathbf{W}(t)]^{-1}\mathbf{W}(t)^{H}\mathbf{x}(t) = \mathbf{D}(t)\mathbf{y}(t),$$

where

$$\mathbf{D}(t) = [\mathbf{W}(t)^{H}\mathbf{W}(t)]^{-1} = \Big[\mathbf{I} + \sum_{k=2}^{K}\mathbf{T}_{k}^{H}(t)\mathbf{T}_{k}(t)\Big]^{-1}. \tag{2.39}$$

Equation (2.39) yields because of the fact that $\mathbf{W}_k(t)$ $(k = 1, \ldots, K)$ are orthogonal, i.e., $\mathbf{W}_k^H(t)\mathbf{W}_k(t) = \mathbf{I}$. Thus, $\tilde{\mathbf{y}}(t)$ and $\mathbf{y}(t)$ can be obtained as

$$\tilde{\mathbf{y}}(t) = \sum_{k=1}^{K}\tilde{\mathbf{y}}_{k}(t), \tag{2.40}$$

$$\mathbf{y}(t) = \sum_{k=1}^{K}\mathbf{y}_{k}(t), \tag{2.41}$$

where

$$\tilde{\mathbf{y}}_k = \mathbf{D}(t)\mathbf{y}_k(t), \tag{2.42}$$

$$\mathbf{y}_k = \mathbf{W}_k^H(t)\mathbf{x}_k(t), \tag{2.43}$$

which can be implemented in parallel in each computing unit.

Then, according to (2.35), $\tilde{\mathbf{Q}}^{-1}(t)$ can be estimated by performing EVD of $\mathbf{Z}(t)$. Here, $\mathbf{Z}(t)$ is either obtained by

$$\begin{aligned}\mathbf{Z}(t) &= \sum_{\tau=0}^{L-1} \tilde{\mathbf{y}}(t-\tau)\mathbf{y}^H(t-\tau) \\ &= \mathbf{Z}(t-1) + \tilde{\mathbf{y}}(t)\mathbf{y}^H(t) - \tilde{\mathbf{y}}(t-L+1)\mathbf{y}^H(t-L+1),\end{aligned} \tag{2.44}$$

if a sliding window of length L is used, or by

$$\begin{aligned}\mathbf{Z}(t) &= \sum_{\tau=0}^{t} \beta^{t-\tau}\tilde{\mathbf{y}}(\tau)\mathbf{y}^H(\tau) \\ &= \beta\mathbf{Z}(t-1) + \tilde{\mathbf{y}}(t)\mathbf{y}^H(t),\end{aligned} \tag{2.45}$$

if the exponential window is used (β being the forgetting factor, $0 < \beta \leqslant 1$). Once $\tilde{\mathbf{Q}}^{-1}(t)$ has been estimated, the principal eigenvectors are obtained by (2.36), which can again be estimated in parallel. The algorithm costs $O(np^2/K) + O(p^3)$ and is summarized in Table 2.6.

2.6 Simulated Experiments

In this section, the performance of subspace estimation is studied by simulation. In all experiments, the system matrix $\mathbf{A}$ is randomly generated but kept fixed for all Monte Carlo runs. The sources are i.i.d. Gaussian processes of unit power. The noise

Table 2.6: Summary of GMNS for PET

Input: Principal subspaces $\mathbf{W}_k(t)$, $\mathbf{T}_k(t)$

1. Compute $\mathbf{D}(t) = \left[\mathbf{I} + \sum_{k=2}^{K} \mathbf{T}_k^H(t)\mathbf{T}_k(t)\right]^{-1}$
2. Compute $\mathbf{Z}(t)$ as in (2.44) or (2.45)
3. Compute $\tilde{\mathbf{Q}}^{-1}(t) = \text{eig}(\mathbf{Z}(t))$
4. Extract p principal eigenvectors: $\mathbf{U}_s(t) = \mathbf{W}(t)\tilde{\mathbf{Q}}^{-1}(t)$

Output: Principal eigenpairs: $\mathbf{U}_s(t)$ and $\{\lambda_i(t)\}_{i=1:p}$

is spatially white with variance of σ^2. The SNR is defined as

$$\text{SNR} = 10\log_{10}\frac{\| \mathbf{A} \|^2}{\sigma^2}. \tag{2.46}$$

To assess the performance of the proposed algorithms, the following two criteria are used: subspace estimation performance (SEP) and eigenvector estimation performance (EEP). The lower the values of SEP or EEP, the better the performance.

The SEP is defined as

$$\text{SEP}(t) = \frac{\text{Tr}\{\mathbf{W}_i^H(t)(\mathbf{I} - \mathbf{W}_{\text{ex}}(t)\mathbf{W}_{\text{ex}}^H(t))\mathbf{W}_i(t)\}}{\text{Tr}\{\mathbf{W}_i^H(t)(\mathbf{W}_{\text{ex}}(t)\mathbf{W}_{\text{ex}}^H(t))\mathbf{W}_i(t)\}}, \tag{2.47}$$

where $\mathbf{W}_i$ is the estimated subspace at the i-th run, and $\mathbf{W}_{\text{ex}}$ is the exact subspace weight matrix computed by orthorgonalizing $\mathbf{A}$. In the case of a batch system, we can drop t from SEP.

The EEP is defined as

$$\text{EEP}(t) = \| \mathbf{U}(t) - \mathbf{U}_{\text{ex}} \|_F^2, \tag{2.48}$$

where $\mathbf{U}(t)$ is the matrix of the estimated eigenvectors[1], and $\mathbf{U}_{\text{ex}}$ is the matrix of the exact eigenvectors computed from the exact covariance matrix (i.e., noiseless case) using the full SVD algorithm. Similarly, we can drop t when considering a batch system.

In all cases, the results are reported by taking the average over 100 Monte Carlo runs. To assess the performance of the algorithms with respect to the number of sources, p, we present two different scenarios ($p = 2$ and $p = 6$) for relatively large dimensional systems. A summary of parameters used in the experiments is given in Table 2.7. For principal subspace estimation/tracking, parameters are chosen based on the configuration of one of real radio astronomy systems from which we collected data, as described in Section 2.7. For minor subspace estimation/tracking, the number of sensors is chosen randomly but in such a way that $(n-d)/p$ is integer-valued. The value of T is considered following the radio astronomy application (Section 2.7) where the sample size was large enough and may even exceed 100 times n.

2.6.1 Minor Subspace Analysis

First, we assess the performance of GMNS-MSA against the standard MSA method using SVD (SVD-MSA) with both small ($p = 2$) and large ($p = 6$) numbers of sources. The results indicate that GMNS-MSA has performance close to SVD-MSA when $p = 2$, and it losses some accuracy at low SNR (i.e., SNR < 5 dB) when $p = 6$, as shown in Figure 2.5(a). However, in both experiments, the dominant cost of GMNS-MSA is reduced by a factor of K^2, as compared to SVD-MSA. The selected GPCS is given in Table 2.8.

We further consider the effect of the number of sources on the performance of GMNS-MSA by fixing n and changing p. Moreover, we take into account two scenar-

[1]To remove inherent ambiguities, the eigenvector norm is set to one and its first entry is chosen to be positive real-valued.

Table 2.7: Parameters used in our experiments

Experiment	Figure	n	p	K	T	q
1	5	30	2	4	500	N/A
2	5	30	6	4	500	N/A
3	6	30	1:10	4	500	N/A
4	6	30	1:10	4	500	N/A
5	7	48	2	4	500	4
6	7	48	6	4	500	8
7	8	48	1:10	4	500	8
8	8	48	1:10	4	500	8
9	9	48	2	4	500	4
10	9	48	6	4	500	8
11	10	30	2	4	4000	N/A
12	11	30	6	4	4000	N/A
13	12	48	2	4	3000	4
14	13	48	6	4	3000	8
15	14	48	2	4	3000	4
16	15	48	6	4	3000	8
17	16	48	2	4	20971	4

Table 2.8: GPCS used for first experiment

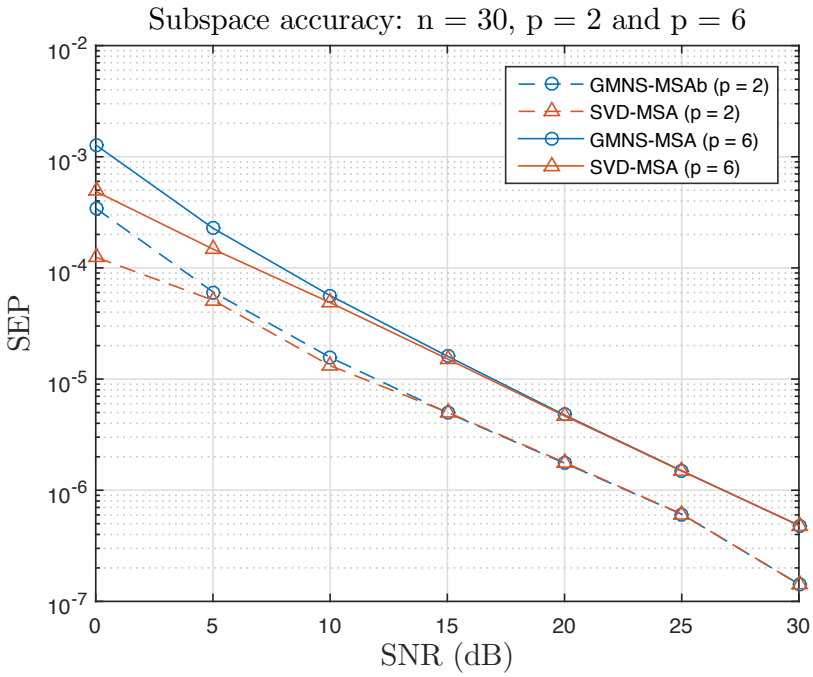
Tuples	Members
t_1	$(m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9)$
t_2	$(m_1, m_2, m_{10}, m_{11}, m_{12}, m_{13}, m_{14}, m_{15}, m_{16})$
t_3	$(m_1, m_2, m_{17}, m_{18}, m_{19}, m_{20}, m_{21}, m_{22}, m_{23})$
t_4	$(m_1, m_2, m_{24}, m_{25}, m_{26}, m_{27}, m_{28}, m_{29}, m_{30})$

ios: low SNR (SNR = 5 dB) and high SNR (SNR = 20 dB). Figure 2.5(b) suggests that GMNS-MSA is robust and comparable with SVD-MSA in both scenarios.

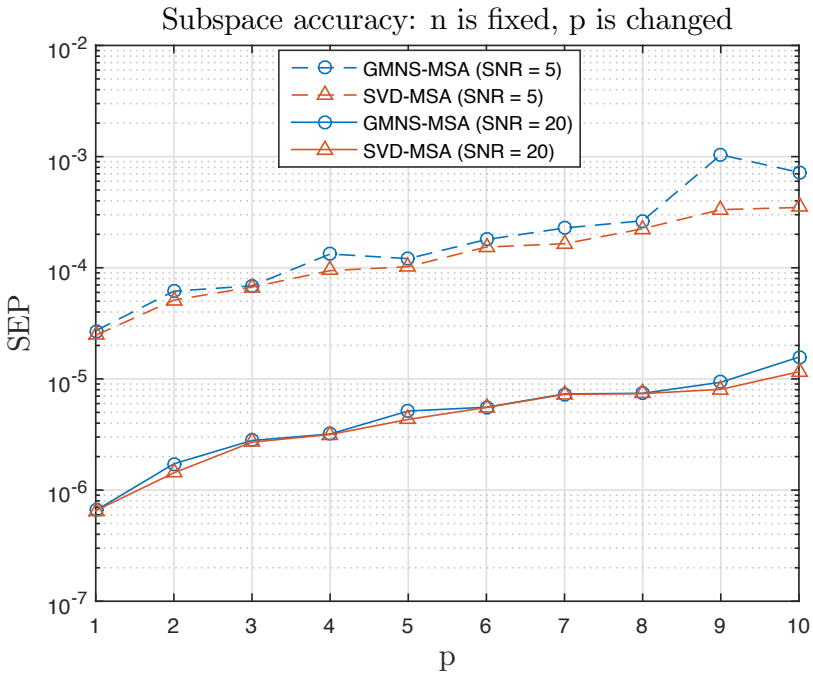
2.6.2 Principal Subspace/Component Analysis

For PSA, we compare the performance of GMNS-N-PSA, GMNS-O-PSA and GMNS-NU-PSA with the standard PSA method using SVD (SVD-PSA), for $p = 2$ and $p = 6$ as shown in Figures 2.6(a). Among the four algorithms, GMNS-N-PSA has the lowest performance, as noticed in Section 2.4.1.1. The three other methods reach the SVD, except at low SNRs. Additionally, we conduct a similar experiment as in the MSA case to evaluate the effect of p on the performance of GMNS-N-PSA, GMNS-O-PSA and GMNS-NU-PSA (Figure 2.6(b)). At low SNR, GMNS-N-PSA degrades the performance when p increases while GMNS-O-PSA, GMNS-NU-PSA and SVD-PSA are comparable. At high SNR, all algorithms are nearly identical in terms of estimation accuracy.

For PCA, the same observation is done as for PSA, as shown in Figures 2.7. Similar to GMNS-MSA, GMNS-PCA based on the three GMNS-MSA algorithms have the advantage of lower computational cost with a gain approximately being equal to K^2, as compared to the standard PCA method based on SVD.

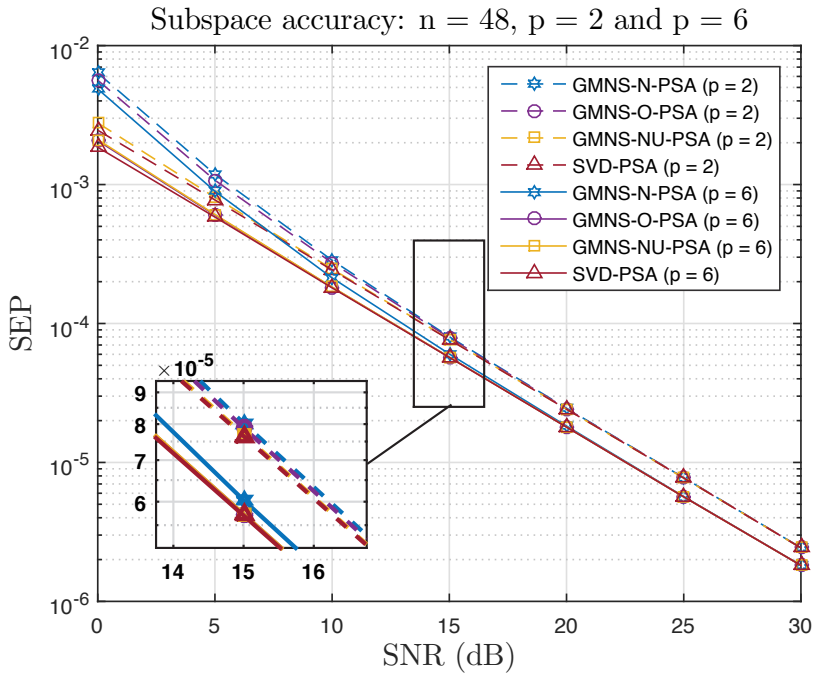


(a) SEP vs. SNR

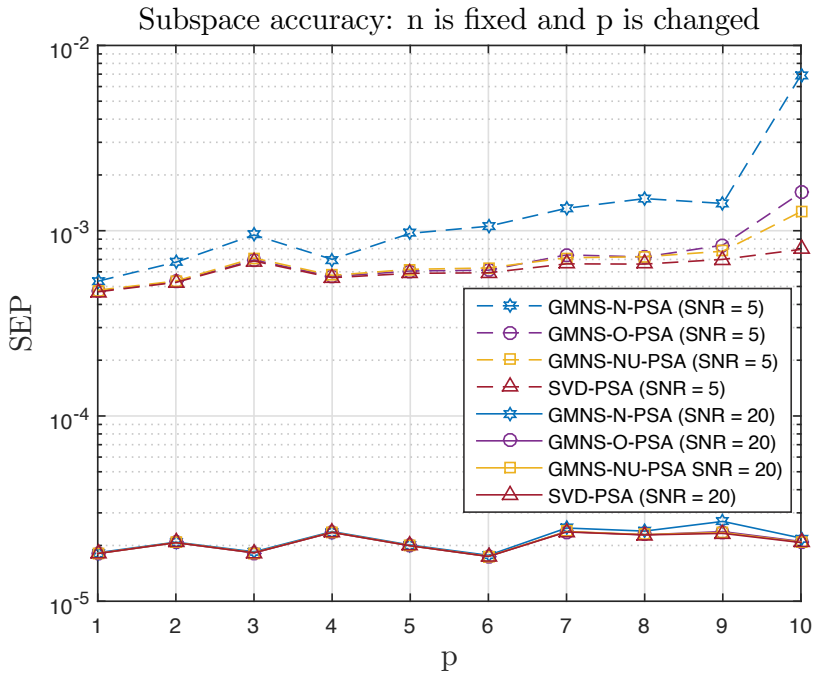


(b) SEP vs. p

Figure 2.5: Minor subspace estimation



(a) SEP vs. SNR



(b) SEP vs. p

Figure 2.6: Principal subspace estimation.

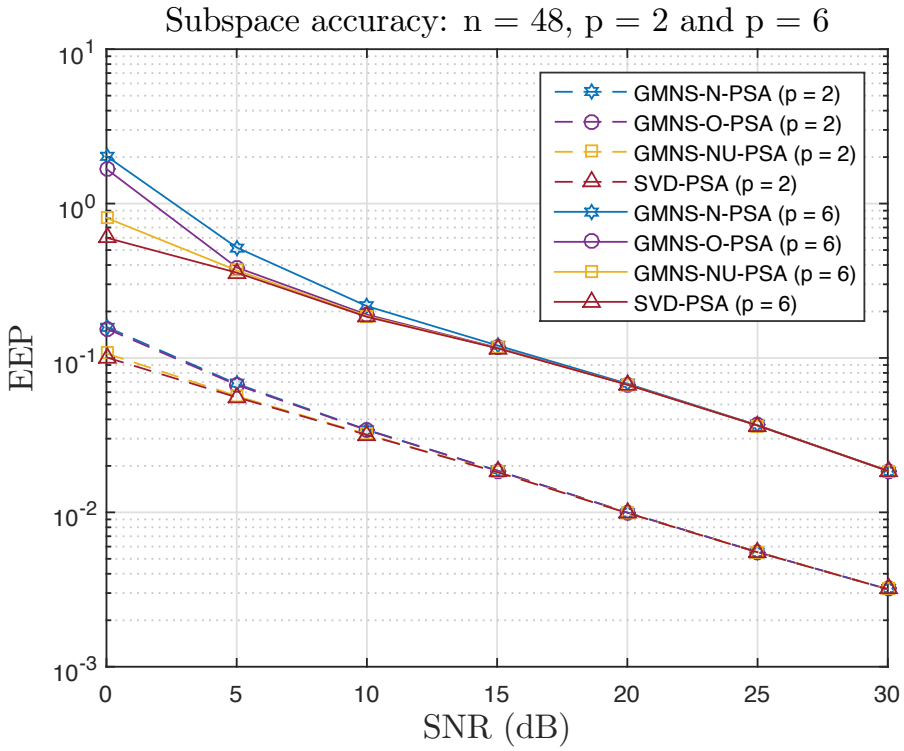


Figure 2.7: Principal eigenvector estimation: EEP vs. SNR.

2.6.3 Minor Subspace Tracking

For MST, we choose two low-cost algorithms, FOOja [27] and FDPM [16], and one moderate-cost one, YAST (yet another subspace tracking) [29], and compare them with the corresponding GMNS-based algorithms: GMNS-MST-FOOja, GMNS-MST-FDPM and GMNS-MST-YAST. All algorithms run in a noisy environment with SNR = 15 dB. The performance results with respect to $p = 2$ and $p = 6$ are shown in Figure 2.8, respectively.

Interestingly, the performance of GMNS-MST-FOOja is better than FOOja even though its convergence rate is slower. The reason is that dividing data into small subsystems reduces the search space which then mitigates the local minima convergence problem and enhances the overall performance. An analogous observation can be seen for GMNS-MST-FDPM and FDPM. Better convergence rate and estimation

accuracy are obtained by both YAST[1] and GMNS-MST-YAST but at the expense of higher computational complexity.

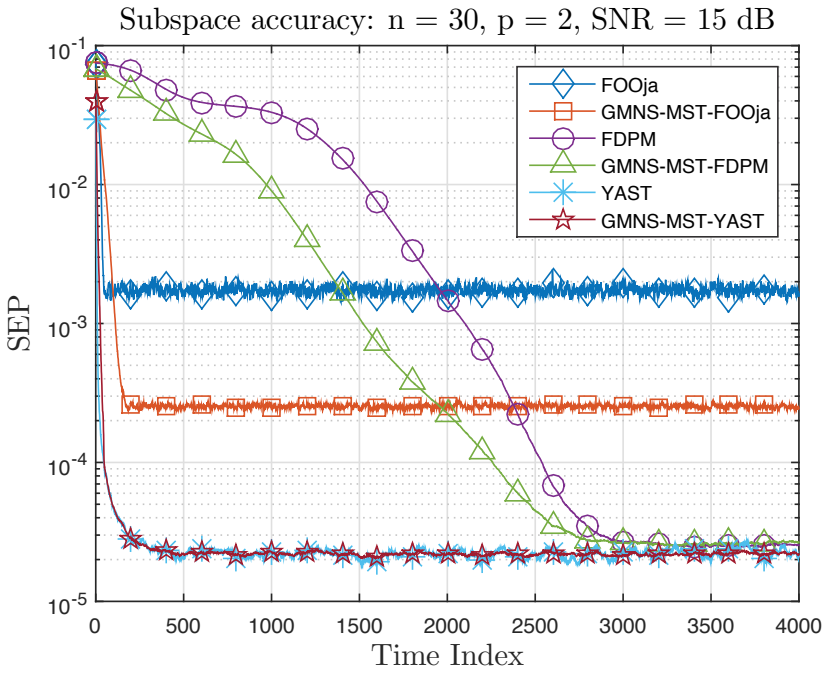
2.6.4 Principal Subspace Tracking

For PST, we compare OPAST and FDPM[2], with their corresponding GMNS-based algorithms: GMNS-N-PST-OPAST, GMNS-O-PST-OPAST, GMNS-N-PST-FDPM and GMNS-O-PST-FDPM. It can be seen from Figure 2.9 that the GMNS-based algorithms have the same performance as their original algorithms, but with a reduced cost. Also, we observe a clear advantage in favor of OPAST-based algorithms as compared to FDPM-based ones.

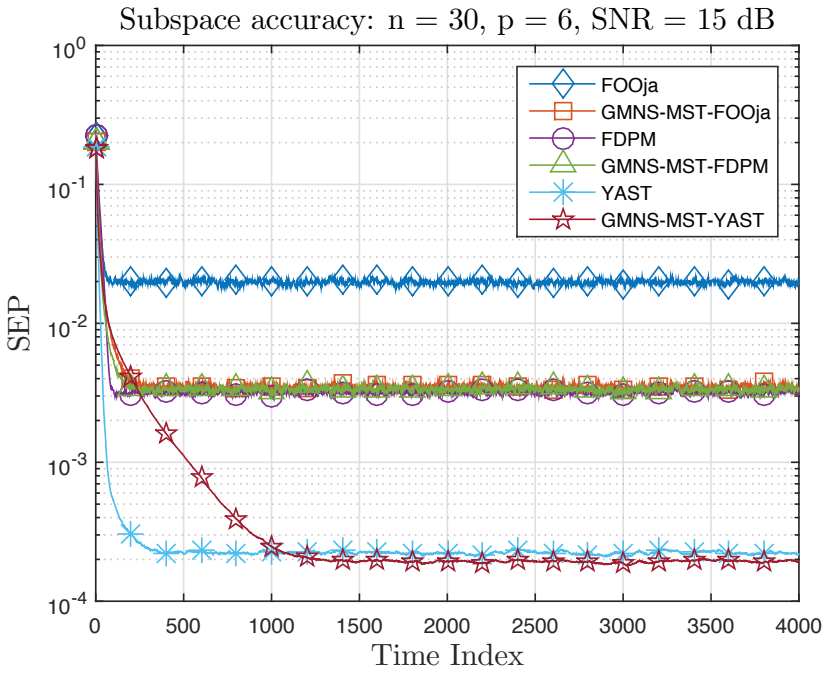
2.6.5 Principal Eigenvector Tracking

Since the performance of GMNS-based FDPM is worse than GMNS-based OPAST, as just shown above, we apply the GMNS-PET method in Section 2.5.3 using OPAST to track the principal eigenvectors, with two parallelized GMNS-based algorithms: GMNS-N-PST-OPAST and GMNS-O-PST-OPAST. Then, we compare their results with the stand SVD-based algorithm (SVD-PST). As shown in Figure 2.10, both GMNS-N-PST-OPAST and GMNS-O-PST-OPAST have the same performance as that of SVD-PST.

[1]YAST has complexity of $O(np^2)$ as presented in [29].
[2]The principal subspace can be obtained by FDPM by changing the sign of step size parameter.

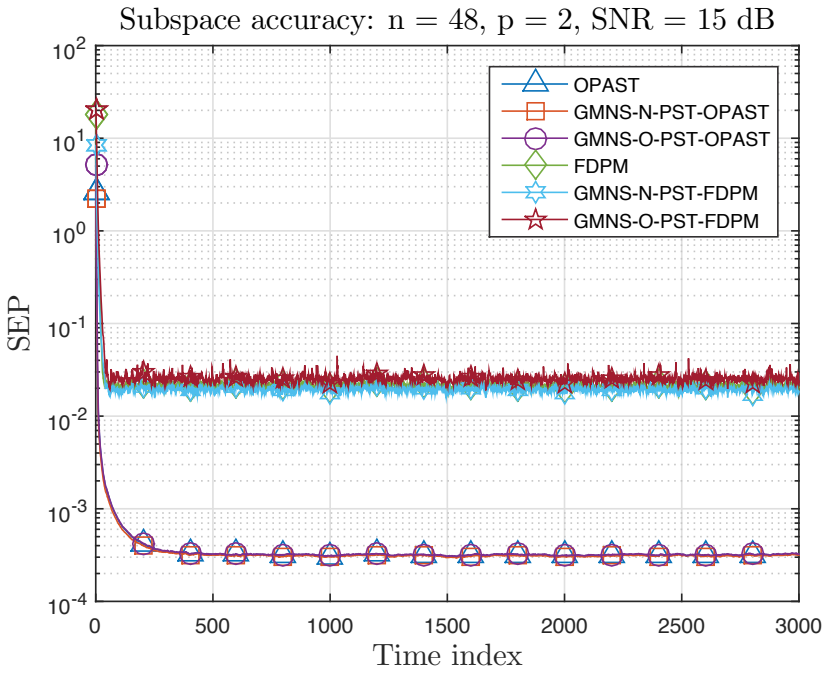


(a) $p = 2$

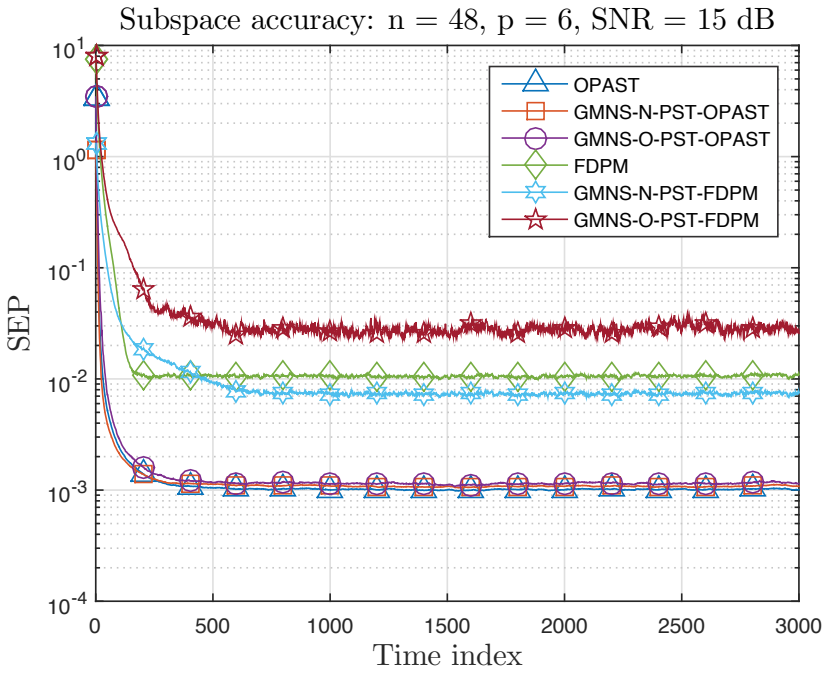


(b) $p = 6$

Figure 2.8: Minor subspace tracking.

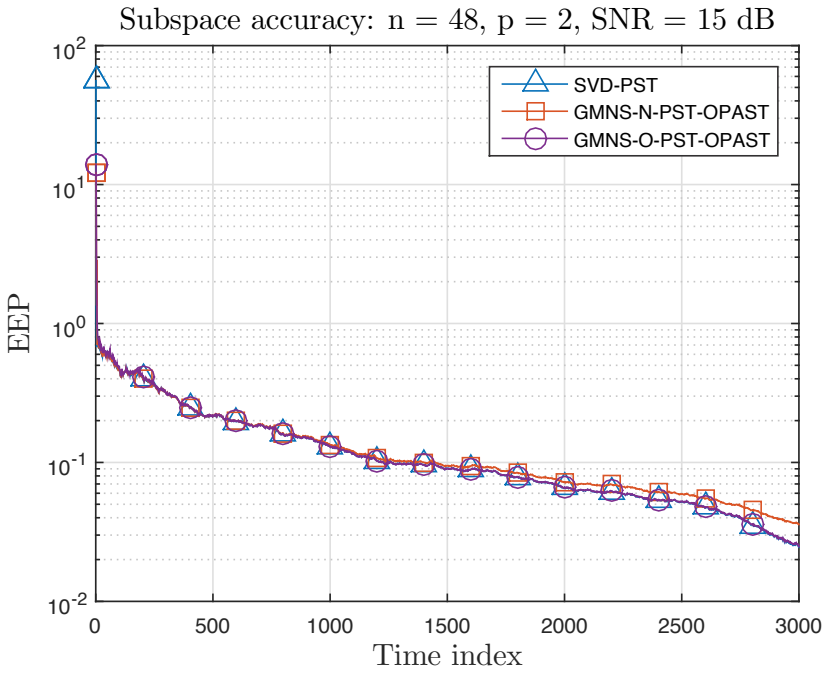


(a) $p = 2$

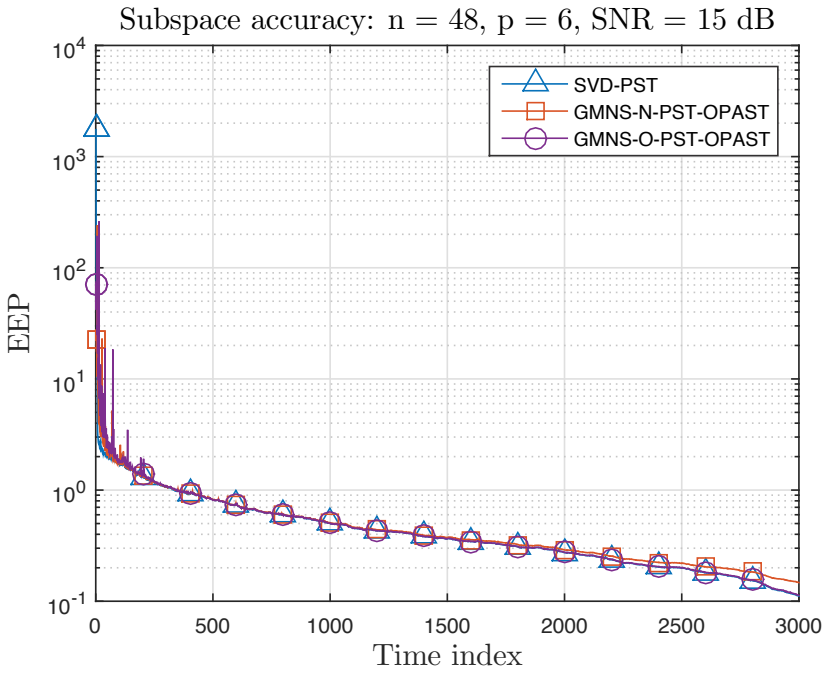


(b) $p = 6$

Figure 2.9: Principal subspace tracking.



(a) $p = 2$



(b) $p = 6$

Figure 2.10: Principal eigenvector tracking.

2.7 Application to RFI Mitigation in Radio Astronomy

Now, we consider RFI mitigation which is a challenging problem in radio astronomy [30]. In general, RFI is difficult to void even with the spectrum being protected and the deployed area being relatively remote. The interference sources in radio astronomy can stem from various man-made wireless services such as mobile cellular telephone, global positioning system satellites, digital audio and video broadcasting, and so on.

The effect of RFI can be observed in Figure 2.11(b). In this example, the dataset comes from a single LOFAR station with $n = 48$ antennas, as shown in Figure 2.11(a). Usually, those 48 antenna outputs are beamformed in real time and the corresponding signal is sent to a central correlator for further radio astronomical processing with other remote stations.

In our experiment, a 5.2 ms signal of those 48 antennas has been stored on disk, and a 2 MHz band around a specific terrestrial RFI (a land mobile at 55 MHz) has been selected to produce a 48×48 covariance matrix. The different sky images presented in this section are derived from this matrix.

In Figure 2.11(b), while signals-of-interest (SOIs) are not really well-defined, a "strong" RFI appears at the horizon. This image with distortions and artifacts is referred to as "dirty image". RFI can lead to distorted data and unwanted artifacts, causing difficulty in astronomical observation.

To mitigate RFI, an efficient method is based on projection in which a key step is subspace estimation. However, as stated in the introduction, SVD- or EVD-based subspace estimation methods are quite expensive in terms of computational complexity. Here we will illustrate how we can apply the proposed GMNS-based methods to

tackle this problem while still preserving the imaging quality.

The model under consideration can be presented as [31]

$$\mathbf{x}(t) = \mathbf{A}_c\mathbf{c}(t) + \mathbf{A}_r\mathbf{r}(t) + \mathbf{n}(t), \tag{2.49}$$

where $\mathbf{A}_c \in \mathbb{C}^{n\times m}$ is the cosmic source spatial signature matrix, $\mathbf{c} \in \mathbb{C}^m$ is the cosmic source signal vector, $\mathbf{A}_r \in \mathbb{C}^{n\times p}$ is the interference spatial signature matrix, $\mathbf{r}(t) \in \mathbb{C}^p$ is the RFI signal vector, and $\mathbf{n}(t) \in \mathbb{C}^n$ is the additive white noise vector with unknown variance σ^2.

Hence, we can estimate the data covariance matrix as

$$\mathbf{R}_{xx} = \mathbf{A}_c\mathbf{R}_{cc}\mathbf{A}_c^H + \mathbf{A}_r\mathbf{R}_{rr}\mathbf{A}_r^H + \sigma^2\mathbf{I}, \tag{2.50}$$

where $\mathbf{R}_{cc}$ and $\mathbf{R}_{rr}$ are cosmic and RFI covariance matrices, respectively. Here we have assumed that the cosmic sources, the RFIs and the system noise are uncorrelated. The cosmic sources are point sources because of relative distance between the source and the instrument.

2.7.1 Orthogonal Projection based RFI Mitigation Algorithm

This method can be implemented by first computing an orthogonal projection matrix $\mathbf{P}_r$ and then applying it to a "dirty" covariance matrix to produce "clean" one (see [1, 32, 33] and references therein). In particular, the orthogonal projector is computed as

$$\mathbf{P}_r = \mathbf{I} - \mathbf{W}_r(\mathbf{W}_r^H\mathbf{W}_r)^{-1}\mathbf{W}_r^H, \tag{2.51}$$

where $\mathbf{W}_r$ is the estimated RFI principal subspace using SVD/EVD. We can estimate the "clean" covariance matrix as

$$\bar{\mathbf{R}} = \mathbf{P}_r \mathbf{R} \mathbf{P}_r^H. \tag{2.52}$$

In fact, we can apply the projection matrix at the pre-correlation stage (i.e., at antenna array output). However, because the data covariance matrix is produced by the radio astronomy system by default, the described method is preferred.

In Figure 2.11(c), the SVD-based subspace projector is applied according to (2.51). A subspace of dimension 48×2, corresponding to two principal eigenvalues, is selected. The Milky Way as well as Cassiopeia A and Cygnus A are now strongly visible and the land mobile signal has been mitigated.

2.7.2 Qualitative Comparison

In Section 2.6, we have compared the subspace estimation accuracy between the proposed algorithms and the corresponding SVD-based algorithms quantitatively via numerical simulation. Now we conduct a qualitative experiment using real-life data for further evaluation.

We replace the principal subspace in (2.51), estimated by the standard SVD-PSA method, with the principal subspace estimated by our GMNS-PSA method and build skymaps after RFI mitigation. Again, the subspace corresponding to two principal eigenvalues defines the RFI subspace ($p = 2$). As can be seen from Figures 2.11(d)–(f), SOIs are enhanced while RFIs are removed (only the strongest RFI is circled in these figures). From those figures, we can see that the imaging quality based on GMNS-PSA is comparable with that of SVD-PSA. Again, our main advantage is the

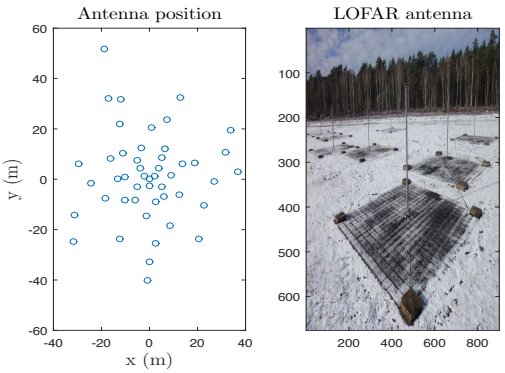
fact that the cost is reduced by a factor of K^2 compared to SVD-PSA.

While this experiment is based on a significant but still limited number of antennas, our method is potential when massive data like in the SKA project [12] need to be processed or when data from some stations need to be fused.

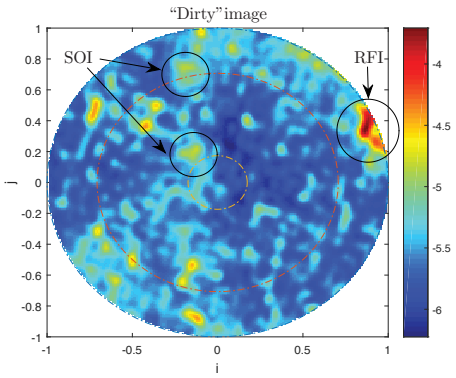
2.8 Conclusions

In this chapter, we have proposed a simple but efficient approach for estimation and tracking of the signal and noise subspaces. The different problems considered in this work are quite common in many array processing applications and are known as the most expensive tasks in source localization and separation to the extent that many efficient spatial filtering methods have been disregarded in real-life applications which use large antenna arrays, e.g., RFI mitigation in radio astronomy [30,31][1]. Our GMNS solution exploits the specific array processing model together with a parallel computing architecture to reduce the overall cost by a factor close to K^2, where K is the number of parallel computing units, for large dimensional systems. At the same time, it can be used to fuse data from a number of data sources. Several algorithmic versions of the GMNS have been developed and the performance was assessed via simulated and real-life experiments. The performance results showed that GMNS represents an excellent solution to deal with large size arrays when distributed resources or parallel computing units are available.

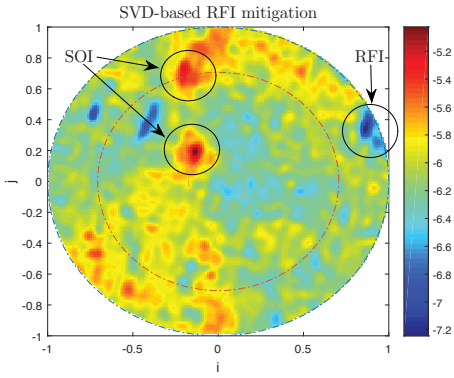
[1]Due to their high computational cost, efficient subspace-based RFI mitigation methods are replaced in practice by a simple RFI or 'No RFI' labeling method.



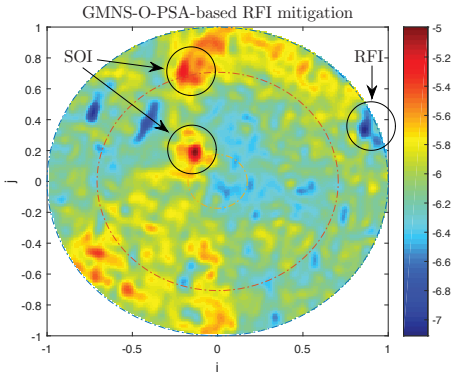
(a) Antenna position and LOFAR antenna



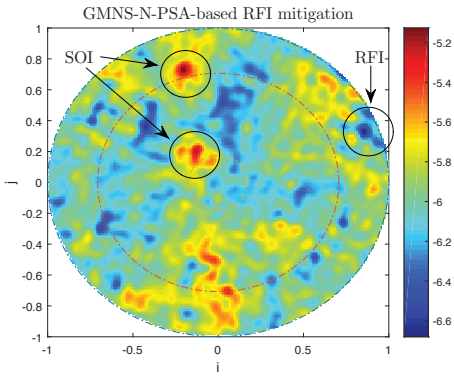
(b) "Dirty" image



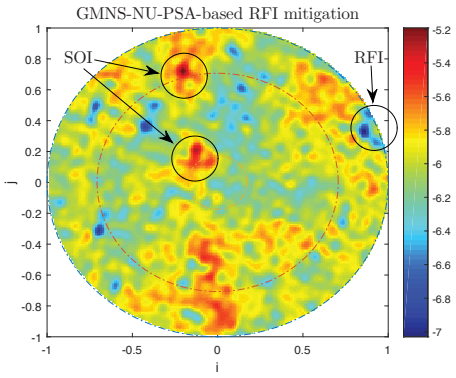
(c) SVD-based RFI mitigation



(d) GMNS-O-PSA-based RFI mitigation



(e) GMNS-N-PSA-based RFI mitigation



(f) GMNS-NU-PSA-based RFI mitigation

Figure 2.11: Image formation before (b) and after RFI mitigation (c-f). Qualitative comparison between SVD-based (c) and GMNS-based (d-f) subspace estimation on RFI mitigation in radio astronomy. Signals-Of-Interest (SOIs): Cassiopeia A and Cygnus A; Radio-Frequency-Interference (RFI): a land mobile signal. (Best view in color)

Appendix A

Appendix

A.1 Proof of Theorem 1

To prove Theorem 1, we need to show that the columns of $\mathbf{V}_i$ belong to the noise subspace and the columns of $\mathbf{V}$ form a vector basis of the noise subspace.

First, note that $\tilde{\mathbf{V}}_i^H \mathbf{A}_i = \mathbf{0}$ leads to $\mathbf{V}_i^H \mathbf{A} = \mathbf{0}$ because of the zero-padding procedure. Hence, the columns of $\mathbf{V}_i$ belong to the noise subspace.

To prove that the columns of $\mathbf{V}$ form a vector basis, let us show that the noise matrix has (up to row permutation) a block diagonal structure as illustrated in Figure A.1 with non-singular $d \times d$ diagonal blocks, which guarantee its full column rank. Indeed, according to the GPCS concept, the i-th $d \times d$ diagonal block represents the entries of $\mathbf{V}_i$ corresponding to the d system outputs not shared by the preceding subsystems (i.e., associated to tuples $1, 2, \ldots, i-1$). The block diagonal structure is then a direct consequence of the zero-padding technique used to build $\mathbf{V}_i$ from $\tilde{\mathbf{V}}_i$.

Let us prove now that the $d \times d$ diagonal blocks are non-singular. Indeed, if a given diagonal block matrix is singular, then there exists a noise vector with at most p non-zero entries; this is in contradiction with the assumption that any p rows of matrix $\mathbf{A}$ are linearly independent, as assumed in Theorem 1.

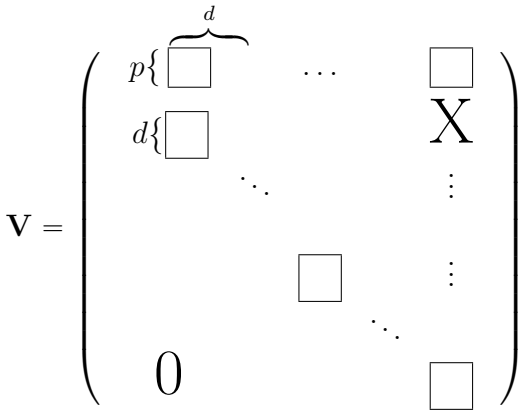


Figure A.1: Block diagonal structure of matrix $\mathbf{V}$.

Part II

Fast Tensor Decomposition

Chapter 3

Background and Related Works

Contents

In part 1 of this thesis, we have considered matrix decomposition. We now move to tensor decomposition. This chapter aims to review several recent state-of-the-art approaches for large-scale tensor data and introduce the fundamentals of tensor decomposition. This serves as a background for next chapters.

3.1 Introduction

Large volumes of data are being generated at any given time, especially from transactional databases, multimedia content, social media, and applications of sensors in the Internet of Things. When the size of datasets is beyond the ability of typical database software tools to capture, store, manage, and analyze, we face the phenomenon of big data for which new and smarter data analytic tools are required. Big data provides opportunities for new form of data analytics, resulting in substantial productivity.

For datasets collected in a multi-dimensional form, they can be naturally represented by multi-way arrays, which are called tensors[1]. If we consider a vector as a first-order tensor, a matrix as a second-order tensor, we will work with higher-order tensors (of order larger than two) for multiway arrays. In other words, while a matrix is indexed with two indices, a higher order tensor is a data structure with more than two indices. For example, a three-way tensor can be used to easily store the time-frequency representation of an EEG dataset. The first way is for the spatial dimension, storing the locations of the electrodes (channels) that measure the electricity of the brain. The two other dimensions are the time and the frequency which store the time-varying frequency content of the EEG signal. These tensor decompositions, which reveal different structures/components hidden in the underlying tensors,

[1]More precisely, tensors are introduced in linear algebra as multilinear forms which are naturally represented, for a given basis of the considered Euclidean space, by multi-way array.

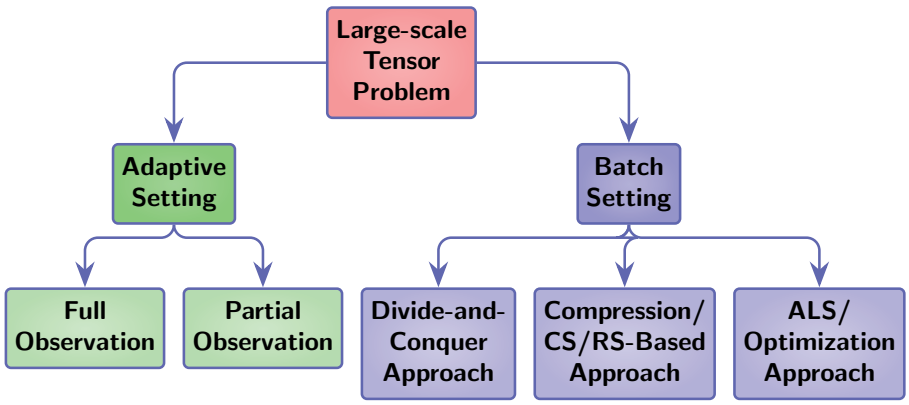


Figure 3.1: Taxonomy of large-scale tensor problem.

thereby provide efficient tools to analyze, compress and understand data.

Two widely-used tensor decompositions are: (i) parallel factor analysis (PARAFAC) [34], also known as canonical decompostion (CANDECOMP) [35] used for latent parameter estimation, and (ii) Tucker decomposition [36] often used for compression and subspace estimation. While matrix decompositions (e.g., singular value decomposition-SVD, non-negative matrix decomposition) are used as powerful tools to analyze two dimensional data, tensor decompositions are more versatile because they enjoy the following main advantages for multi-dimensional data: (i) Tensors are a natural generalization of matrices; (ii) The PARAFAC decomposition possesses the uniqueness property under mild conditions [37]. Note that additional constraints (such as non-negativity, sparseness) imposed on the tensor model, when possible, can improve the uniqueness property and/or interpretation [38]; (iii) The Tucker decomposition takes into account the multi-way structure of data and captures multiple interactions instead of pairwise interactions, which will be destroyed if applying matrix decomposition to collapse some of the modes of data [39]; (iv) Tensor decompositions outperform matrix decompositions in some practical applications as shown in [40].

Tensor decomposition is encountered in diverse applications, such as: psychometrics, chemistry, signal processing, linear and multilinear algebra, data communication,

data mining, computer vision, machine learning, to name a few. Thus, a comprehensive survey which covers all those disciplines is difficult. We list here several important surveys arranged in chronological order: basics of tensor decomposition and its applications [37], unsupervised multiway data analysis [41], multi-linear subspace learning for tensor data [42], tensor factorization and decomposition in data mining [43], low-rank tensor approximation [44] and tensor decomposition for signal processing [40,45]. This list is by no means exhaustive and for more details, we refer to them and the references therein.

Our short survey, in complementary with the mentioned surveys, focuses on large-scale tensor problem and the potential approaches to solve it. Here we only consider the two most popular models: PARAFAC and Tucker and disregard other important models such as block tensor decomposition [46–48], tensor networks including tensor trains [49] and hierarchical Tucker [50], coupled matrix/tensor-tensor decomposition [51], [52].

Notations: We follow the notations used in [37]. Calligraphic letters are used for tensors ($\mathcal{A}, \mathcal{B}, \dots$). Matrices, vectors, 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 $\mathcal{A} \in \mathbb{R}^{I \times J \times K}$ is symbolized as a_{ijk}, element (i, j) of a matrix $\mathbf{A} \in \mathbb{R}^{I \times J}$ as a_{ij}, and i-th entry of a vector $\mathbf{a} \in \mathbb{R}^{I}$ 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 and $\mathbf{A} * \mathbf{B}$ the Hadamard product which is the element-wise matrix product, $\mathbf{a} \circ \mathbf{b}$ the outer product of $\mathbf{a}$ and $\mathbf{b}$.

3.2 From Matrix Decomposition to Tensor Decomposition

Before starting with basic operators and models of tensor decomposition, we provide "the bridge" between matrix and tensor decomposition through a comparison of their uniqueness. Here, we are interested in computing a low-rank approximation. In general form, low rank matrix decomposition can be written as

$$\mathbf{X} = \mathbf{P}\mathbf{Q}^T, \tag{3.1}$$

which is non-unique. It means that there always exists a non-singular matrix $\mathbf{S}$ such that

$$\mathbf{X} = \mathbf{P}\mathbf{Q}^T = (\mathbf{P}\mathbf{S}^{-1})(\mathbf{S}\mathbf{Q}^T) = \hat{\mathbf{P}}\hat{\mathbf{Q}}^T \tag{3.2}$$

where $\hat{\mathbf{P}} = \mathbf{P}\mathbf{S}^{-1}$ and $\hat{\mathbf{Q}} = \mathbf{Q}\mathbf{S}^T$. To be unique, additional constraints must be imposed. Among various constraints, the most popular ones include orthogonality, sparseness and non-negativity. For example, SVD of $\mathbf{X}$ is given by

$$\mathbf{P} = \mathbf{U}\mathbf{E}, \mathbf{Q} = \mathbf{V} \tag{3.3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and $\mathbf{E}$ is a diagonal matrix with non-negative real singular values. In convention, singular values are arranged in descending order. As a result, the uniqueness of SVD , up to a sign, is due to orthogonal constraint on $\mathbf{U}$ and $\mathbf{V}$, and ordered diagonal matrix $\mathbf{E}$.

Let's consider, for example, the PARAFAC tensor decomposition. The PARAFAC model in matrix form (unfolded tensor) can be represented as (see next section for

more details)

$$\mathbf{X} = \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T \tag{3.4}$$

If we decompose $\mathbf{X}$ by using SVD, we will obtain

$$\mathbf{P} = \mathbf{A}\hat{\mathbf{S}}^{-1} \tag{3.5}$$

$$\mathbf{Q} = (\mathbf{C} \odot \mathbf{B})\hat{\mathbf{S}}^T \tag{3.6}$$

where $\hat{\mathbf{S}}$ is a non-singular matrix. However, different from matrix case, we know that $\mathbf{Q}$ has Khatri-Rao product structure. Without any additional constraints, by "restoring" the Khatri-Rao structure of $\mathbf{Q}$, we can recover matrix $\mathbf{B}$ and $\mathbf{C}$, then $\mathbf{A}$ uniquely (up to scale and permutation). Thus, PARAFAC model can be seen as matrix decomposition of the unfolded tensor with the Khatri-Rao product structure imposed on matrix $\mathbf{Q}$. The point here is that, using tensor decomposition, if possible, often provides rich structures which can be exploited efficiently to improve the performance and convergence rate of certain algorithms.

3.3 Basic Tensor Operations and Models

In this section, we present basic tensor operators which are often used in developing algorithms for tensor decomposition. We also present intuitive idea of PARAFAC and Tucker model and their uniqueness properties. This section is a summary of rich literature which can be found in the listed surveys. For simplicity, we will present most results in the case of 3-way tensor.

3.3.1 Basic Tensor Operations

3.3.1.1 Tensor Unfolding

Tensor unfolding, also known as matricization and flattening, is an operation which reorders tensor into a matrix. Using tensor unfolding allows to exploit well-defined properties developed in linear algebra for vectors and matrices and provides a convenient way to process tensors. We note that tensor operators can be implemented without tensor unfolding.

Mode-n unfoldings of a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ are defined as

$$\mathbf{X}_{(1)} : [\mathbf{X}_{(1)}]_{i,j+(k-1)J} = x_{ijk}$$
$$\mathbf{X}_{(2)} : [\mathbf{X}_{(2)}]_{j,i+(k-1)I} = x_{ijk}$$
$$\mathbf{X}_{(3)} : [\mathbf{X}_{(3)}]_{k,i+(j-1)I} = x_{ijk}.$$

There are different ways to choose ordering of columns. In literature, three ways are considered: forward cyclic [53], backward cyclic [54] and ascending order [37]. The mode-n-unfolding here corresponds to the ascending case. Using different tensor unfoldings leads to slightly different formula of tensor models. However, it does not affect the final results (i.e., recovered factors) as long as it is chosen consistently.

3.3.1.2 Tensor Multiplication

Entries of mode-1 product of a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ and a matrix $\mathbf{A} \in \mathbb{R}^{N \times I}$, denoted by $(\mathcal{X} \times_n \mathbf{A})$, is given by

$$(\mathcal{X} \times_1 \mathbf{A})_{njk} = \sum_{i=1}^{I} x_{ijk} a_{ni}$$

More generally, entries of *mode-n product* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \cdots \times I_n \times \cdots \times I_N}$ and a matrix $\mathbf{A} \in \mathbb{R}^{J \times I_n}$, denoted by $(\mathcal{X} \times_n \mathbf{A})$, is defined as

$$(\mathcal{X} \times_n \mathbf{A})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 \cdots i_N} a_{j i_n}$$

Let $\mathcal{Y} = \mathcal{X} \times_n \mathbf{A}$. We can write an equivalent expression in unfolded tensor form as follows

$$\mathbf{Y}_{(n)} = \mathbf{A}\mathbf{X}_{(n)}$$

We also have the following properties

$$\begin{aligned} \mathcal{X} \times_n \mathbf{A} \times_m \mathbf{B} &= \mathcal{X} \times_m \mathbf{B} \times_n \mathbf{A} \\ \mathcal{X} \times_n \mathbf{A} \times_n \mathbf{B} &= \mathcal{X} \times_n (\mathbf{B}\mathbf{A}) \end{aligned} \tag{3.7}$$

The *inner product* of two same-size tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I \times J \times K}$ is defined as

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} x_{ijk} y_{ijk}.$$

As a consequence, we have $\langle \mathcal{X}, \mathcal{X} \rangle = \|\mathcal{X}\|^2$. Moreover, the l_1 norm of a tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is defined as

$$\|\mathcal{X}\|_1 = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} |x_{ijk}|.$$

3.3.1.3 Useful Matrix Property

Several useful matrix properties are summarized here

$$
\begin{aligned}
(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) &= (\mathbf{AC}) \otimes (\mathbf{BD}) \\
(\mathbf{A} \odot \mathbf{B})^T (\mathbf{A} \odot \mathbf{B}) &= (\mathbf{A}^T\mathbf{A}) * (\mathbf{B}^T\mathbf{B}) \\
\text{vec}(\mathbf{ABC}^T) &= (\mathbf{C} \otimes \mathbf{A})\,\text{vec}(\mathbf{B}), \\
(\mathbf{A} \otimes \mathbf{B})^{\#} &= ((\mathbf{A}^T\mathbf{A}) * (\mathbf{B}^T\mathbf{B}))^{\#}(\mathbf{A} \otimes \mathbf{B})^T \\
(\mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C})(\mathbf{D} \odot \mathbf{E} \odot \mathbf{F}) &= (\mathbf{AD}) \odot (\mathbf{BE}) \odot (\mathbf{CF})
\end{aligned}
$$

where vec() performs vectorization of a matrix or a tensor that stacks the columns of the matrix or the tensor into a vector (e.g., given $\mathbf{B} \in \mathbb{R}^{I \times J}$, $\text{vec}(\mathbf{B}) = [\mathbf{b}_1^T, \cdots, \mathbf{b}_J^T]^T$ and $()^{\#}$ is the pseudo-inverse operator. Dimensions of each matrix are assumed to match.

3.3.2 PARAFAC model

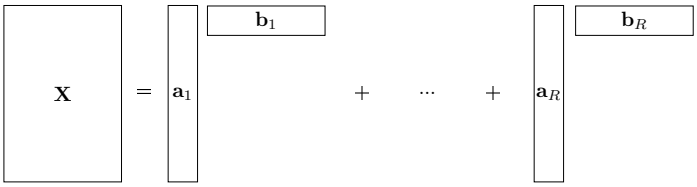
Intuitive idea behind PARAFAC can be captured through Figure 3.2. While SVD for matrices can be written as sum of R rank one matrices (Figure 3.2 (a)), the PARAFAC decomposition of $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ can be defined as

$$
\mathcal{X} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \equiv \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{3.8}
$$

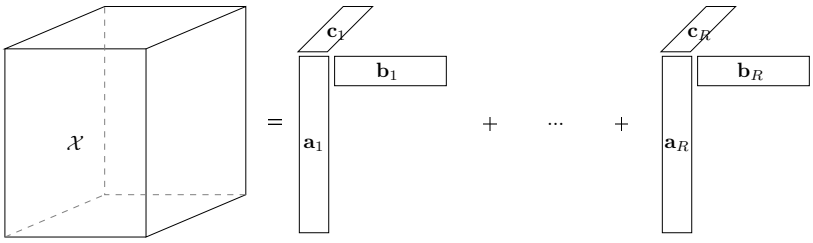
or equivalently

$$
\mathbf{x}_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr} \tag{3.9}
$$

which is the sum of R rank-one tensors (Figure 3.2 (b)), with R being the tensor rank. The set of vectors, $\{\mathbf{a}_r\}, \{\mathbf{b}_r\}, \{\mathbf{c}_r\}$ can be grouped into the so-called loading matrices



(a) SVD of a matrix as sum of R rank-1 matrices.



(b) PARAFAC of a tensor as sum of R rank-1 tensors.

Figure 3.2: PARAFAC can be seen as a generalization of SVD.

$\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_R] \in \mathbb{R}^{I \times R}$, $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_R] \in \mathbb{R}^{J \times R}$, and $\mathbf{C} = [\mathbf{c}_1 \dots \mathbf{c}_R] \in \mathbb{R}^{K \times R}$.

Equation 3.8 can also be formulated in matrix and vector form using mode-n unfolding as

$$\mathbf{X}_{(1)} = \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T \tag{3.10}$$

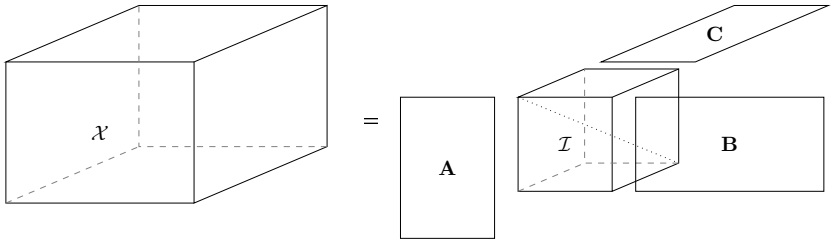
$$\mathbf{X}_{(2)} = \mathbf{B}(\mathbf{C} \odot \mathbf{A})^T \tag{3.11}$$

$$\mathbf{X}_{(3)} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^T \tag{3.12}$$

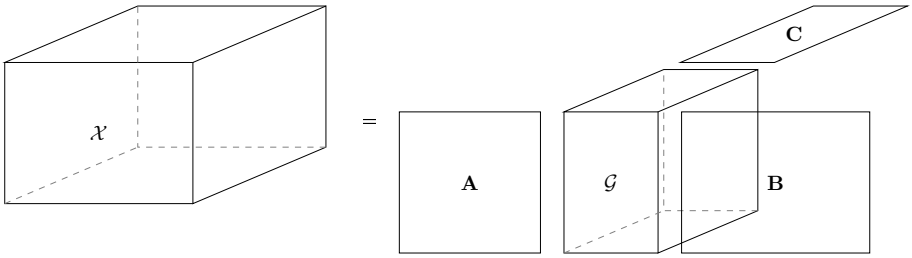
$$\mathbf{x} = (\mathbf{C} \odot \mathbf{B} \odot \mathbf{A})\mathbf{1} \tag{3.13}$$

where $\mathbf{x} = \text{vec}(\mathbf{X}_{(1)})$ and $\mathbf{1} \in \mathbb{R}^{R \times 1}$ whose all entries are one. It is straightforward to see from Figure 3.2 (b) and (3.8) that the sum is unchangeable if we reorder and re-scale rank-one tensors (hence, order of vectors in loading matrices). Thus, we have

$$\mathcal{X} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] = [\![\mathbf{A}\mathbf{\Pi}\mathbf{\Lambda}_1, \mathbf{B}\mathbf{\Pi}\mathbf{\Lambda}_2, \mathbf{C}\mathbf{\Pi}\mathbf{\Lambda}_3]\!] \tag{3.14}$$



(a) Representation of PARAFAC using loading matrices and identity core.



(b) Tucker model with a non-identity core tensor $\mathcal{G}$.

Figure 3.3: PARAFAC can be seen as a special case of Tucker.

where $\mathbf{\Pi}$ is a permutation matrix and $\mathbf{\Lambda}_i, i = 1, \cdots, 3$, are scale diagonal matrices satisfying $\mathbf{\Lambda}_1\mathbf{\Lambda}_2\mathbf{\Lambda}_3 = \mathbf{I}$. PARAFAC decomposition is generically unique (up to scales and permutation) if the following condition is satisfied [55]:

$$2R(R-1) \leqslant I(I-1)K(K-1), \qquad R \leqslant J.$$

This condition is developed based on Kruskal's result [56].

3.3.3 Tucker Model

We note that PARAFAC can also be illustrated as Figure 3.3 (a) with an unit super-diagonal tensor[1] $\mathcal{I}$. If we relax this constraint (i.e., $\mathcal{I}$ now can be sparse or dense tensor), we form the Tucker decomposition of $\mathcal{Y} \in \mathbb{R}^{I\times J\times K}$ which can be written as

[1] an unit super-diagonal tensor is a cubical tensor with ones along the superdiagonal.

follows:

$$\mathcal{Y} = [\![\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \equiv \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} \mathbf{a}_p \circ \mathbf{b}_q \circ \mathbf{c}_r, \tag{3.15}$$

where $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_P] \in \mathbb{R}^{I \times P}$, $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_Q] \in \mathbb{R}^{J \times Q}$ and $\mathbf{C} = [\mathbf{c}_1 \dots \mathbf{c}_R] \in \mathbb{R}^{K \times R}$ are the factor matrices and $\mathcal{G} \in \mathbb{R}^{P \times Q \times R}$ is called the core tensor. In general, factor matrices of Tucker model are not necessarily orthogonal. However, in practice, column-wise orthogonal constraint is, in most cases, imposed.

A special case of orthogonal Tucker decomposition is Higher-Order SVD (HOSVD) [54] where the core tensor has all-orthogonal property besides orthogonal factors. All-orthogonal property means that by considering a 3-way core tensor, the matrices, extracted by fixing one index and releasing two the others, are mutually orthogonal.

Matrix and vector forms of (3.15) can be presented as

$$\mathbf{Y}_{(1)} = \mathbf{A}\mathbf{G}_{(1)}(\mathbf{C} \otimes \mathbf{B})^T \tag{3.16}$$

$$\mathbf{Y}_{(2)} = \mathbf{B}\mathbf{G}_{(2)}(\mathbf{C} \otimes \mathbf{A})^T \tag{3.17}$$

$$\mathbf{Y}_{(3)} = \mathbf{C}\mathbf{G}_{(3)}(\mathbf{B} \otimes \mathbf{A})^T \tag{3.18}$$

$$\mathbf{y} = (\mathbf{C} \otimes \mathbf{B} \otimes \mathbf{A})\mathbf{g}, \tag{3.19}$$

where $\mathbf{y} = \text{vec}(\mathcal{Y})$ and $\mathbf{g} = \text{vec}(\mathcal{G})$.

In contrast to the PARAFAC model, the Tucker model is non-unique since

$$\mathcal{Y} = [\![\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!] = [\![\mathcal{G} \times_1 \mathbf{P} \times_2 \mathbf{Q} \times_3 \mathbf{R}; \mathbf{A}\mathbf{P}^{-1}, \mathbf{B}\mathbf{Q}^{-1}, \mathbf{C}\mathbf{R}^{-1}]\!]. \tag{3.20}$$

Uniqueness can be achieved only if specific constraints are added, for example both sparsity and non-negativity constraints.

3.4 Batch Setting

In batch setting, we categorize the existing algorithms based on their tensor decomposition approaches. For each one, we first describe the main idea which is shared by all algorithms and then present specific solutions and their differences.

3.4.1 Divide-and-Conquer Approach

The main idea of this approach (Figure 3.4) is to divide a big tensor data into number of smaller tensor data; then run specific tensor decomposition algorithms on those data (possibly in parallel scheme) before joining "local" results into "global" one. The difference resides on the way each algorithm handle data (i.e., decentralized or distributed) and on the used optimization techniques.

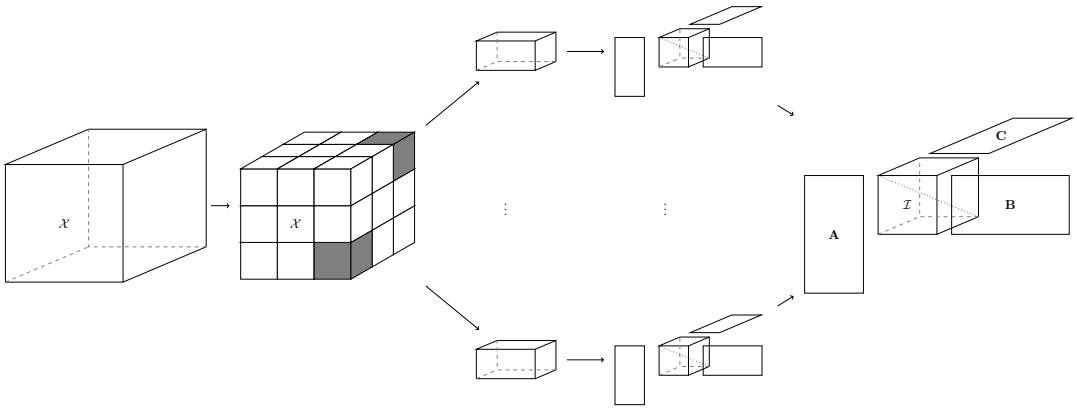


Figure 3.4: Divide-and-Conquer approach for large-scale tensor (An example of PARAFAC model)

3.4.1.1 PARAFAC Model

In [57], the authors proposed to use this approach combined with a fast Alternating Least-Squares (ALS) algorithm. To speed up the joining procedure, they also proposed a multistage reconstruction step where local factors are merged from results

of neighbor sub-tensors. The algorithm works with underlying assumption that the decomposition of each sub-tensor is strictly unique.

In [58], we also used a decentralized approach but the way the structure of PARAFAC is used is different from [57]. In fact, our algorithm is an extension in spirit of the Generalized Minimum Noise Subspace (GMNS) method [23, 59] which permits to use (stable) SVD at small scale for subspace estimation. Our algorithm also use the same uniqueness condition assumption as in [57].

To relax uniqueness condition on sub-tensors, in [60, 61], the authors developed a distributed ALS algorithm. The main idea is to permit collaboration across the three modes of the tensor.

3.4.1.2 Tucker Model

A distributed memory Tucker decomposition for data compression was proposed in [62]. While each data block is fixed in each processor, factor matrices are exchanged between processor grid. Parallel implementation Higher Order Orthogonal Iteration (HOOI) [63] using Sequentially-Truncated HOSVD [64] as an initialization, was taken into account. Those algorithms can also be implemented efficiently by using level 3 Basic Linear Algebra Subprograms (BLAS) routines[1].

3.4.2 Compression/Compressive Sensing/Random Sampling based Approach

The spirit of these approaches is to process a reduced-size or a sparse representation which essentially keeps the same or approximately the same information as the original form. An illustration of this approach is given in Figure 3.5.

[1]Level 3 BLAS aims to implement matrix-matrix operations and supports block-partitioned algorithms

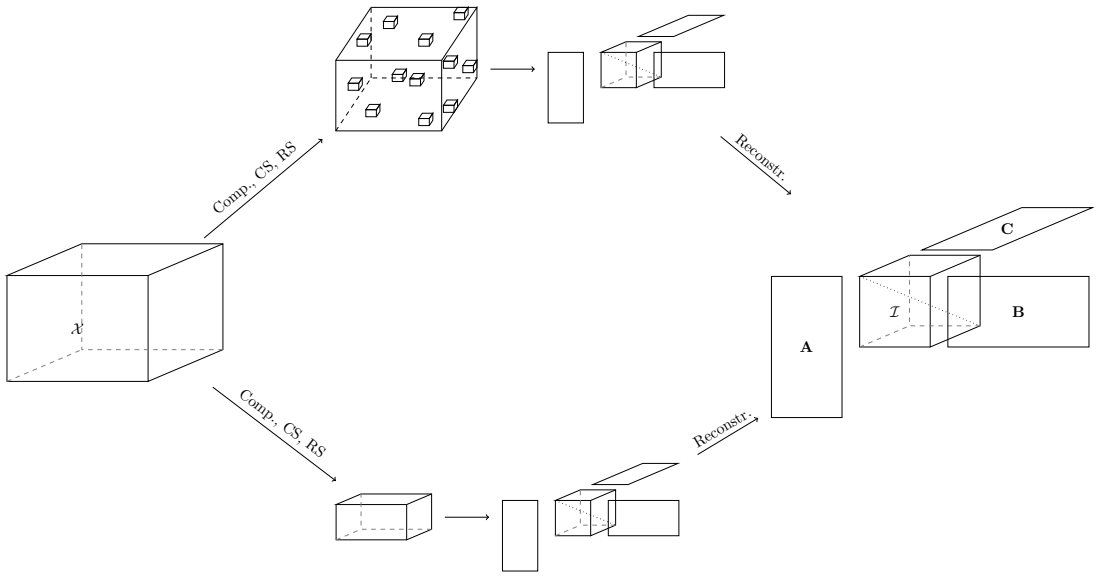


Figure 3.5: Compression/compressive sensing/random sampling approach for large-scale tensor (PARAFAC model)

In tensor decomposition, *compression approach*-based algorithms always use Tucker decomposition or HOSVD to compress data; and then depending on applications, a desired decomposition (e.g., PARAFAC) is applied on the core tensor to extract loading factors. To recover the factors of the original form, the extracted factors are simply multiplied with the corresponding factors of Tucker decomposition. As a consequence, it allows to avoid running desired decomposition on large dimensional tensors (i.e., avoid running iterative algorithm[1] in large-scale data). Since both Tucker and HOSVD algorithms require SVD or EVD computation of large dimensional matrices, this approach would be appropriate only for small or moderate size tensor decomposition. For more details, we refer the reader to [65].

Random sampling (RS) method represents data in smaller size or sparse form while approximately preserving essential information (see [66] for an introduction and review). Depending on data form (e.g., matrix or tensor) and algorithms, there

[1]Most PARAFAC decomposition algorithms are iterative.

are different kinds of sampling strategies such as element-wise, row/column, slide and block. Moreover, those strategies can be chosen following several specific probability distributions (e.g., uniform, non-uniform, data-dependent).

For Tucker model, in [67], along each mode, a column sampling strategy, which can be one-pass or multiple-pass, is first applied to the unfolded tensor. Then the factor matrices are computed as principal singular vectors of the sampled matrix. A similar method but using element-wise sampling is developed in [68]. We note that element-wise sampling yields sparse representation instead of reduced-size form.

For PARAFAC model, Parcube algorithm [69] uses data-dependent-based sampling to determine important part of tensor (i.e., marginal sum of tensor for each mode), runs PARAFAC for each sampled tensor and then merges results. This algorithm only works for sparse tensors and offers no identifiablity guarantee.

In [70], authors further developed compression approach by assuming that the big tensor has underlying low-rank structure (i.e., PARAFAC model) and factor matrices are sparse. While low-rank structure allows to design a special compression matrix which has Kronecker product structure, the sparse factors help to guarantee identifiability (i.e., uniqueness of the recovered factors from results of the compressed tensor). Here, the Kronecker product structure of the compression matrix keeps the compressed tensor having a low-rank as the original one. Thus, authors claim that this approach can be considered as a generalization of *compressive sensing* (CS) idea for multilinear case. A combination of this approach and divide-and-conquer strategy can be found in [71].

3.4.3 Alternating Least-Square/Optimization Approach

For tensor decomposition, *alternating least-squares* yields workhorse algorithm for long time. The idea of alternating approach is simple; at each step, we optimize a

factor while keeping the others fixed. At the beginning there were several efforts to accelerate the convergence rate and overcome the degeneracy[1] problem using line-search approach [72]. The direct implementation of ALS is difficult to handle for large scale data because (i) the size of intermediate computation results between unfolded tensor and all-but-one factors is much larger than that of the interested loading matrices (ii) multiple accesses of original tensor data in different orders are necessary. Several techniques to tackle those problems have been addressed in [73] and [74].

The *general optimization approach* casts the tensor decomposition problem into a nonlinear equation problem and then solves it using standard optimization tools, such as gradient methods. In some difficult situations such as degeneracy or over-factoring case[2], the gradient approach based-algorithms can outperform ALS in terms of accuracy and performance. Within this class, various algorithms have been developed, including the non-linear conjugate gradient [75], also see [76] for the case of missing data, [77] for the case of sparse and nonnegative PARAFAC and Tucker, the fast damped Gauss-Newton (dGN) algorithms [78] and the Alternating Direction Method of Multipliers (ADMM) [79].

3.5 Adaptive Setting

In adaptive setting (also known as *online* or *incremental* setting), we classify the existing algorithms w.r.t. the characteristic of streaming data: full observation (i.e., without missing data) and partial observation (i.e., with missing data). An illustration of adaptive tensor setting is given in Figure 3.6.

[1] Degeneracy refers to problem when collinearity of two or more components in the factor matrices exists.

[2] Over-factoring means that the chosen tensor rank is larger than the true one.

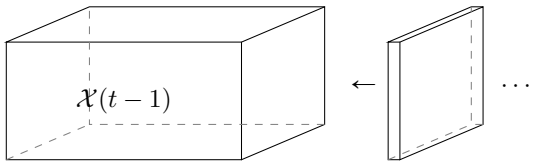


Figure 3.6: Adaptive tensor setting: at time instant t, tensor $\mathcal{X}(t)$ captures a new data slice.

3.5.1 Full Observation

Adaptive PARAFAC model for third-order tensors having one dimension growing with time has been introduced in [80]. Two algorithms using recursive least-squares tracking (PARAFAC-RLST) and simultaneous diagonalization tracking (PARAFAC-SDT) have been proposed. While the former uses first-order methods (i.e., using gradients) to optimize an exponentially-weighted least-squares cost function, the latter exploits SVD tracking algorithm combined with recursive simultaneous diagonalization step. The computational complexity of both algorithms is quadratic in terms of the tensor rank.

To deal with computational complexity problem of [80], we have proposed a linear complexity adaptive PARAFAC algorithm [81] which generalizes the Orthonormal Projection Approximation Subspace Tracking (OPAST) approach [15]. This algorithm, named 3DOPAST, uses a special interpretation of Khatri-Rao product as collinear vectors inside each column. Its performance is equal or even superior to PARAFAC-RLST and PARAFAC-SDT while keeping the computational complexity linear to tensor rank. An improved version of 3DOPAST, named SOAP, using second-order stochastic gradient as well as preserving Khatri-Rao product structure is also proposed in [82]. Moreover, we also adapt SOAP to handle adaptive non-negative PARAFAC model. It is shown that SOAP is stable for very long time run. For more details, we invite the reader to our papers [81, 82].

Adaptive Tucker decomposition has several different names in the literature, for examples: dynamic tensor analysis [83], incremental tensor subspace learning [84], tensor subspace tracking [85]. Streaming tensor analysis was proposed in [83]. In this work, the eigensubspace is updated via tracking a projection matrix for all modes. Dynamic tensor analysis, which is a more general model for streaming tensor analysis, allows a changeable amount of data to come at each time instant. The main idea of the proposed algorithm for this kind of model is to track the covariance matrix and the eigensubspace of unfolded tensors for each mode. With the same spirit, the incremental tensor subspace learning applies an incremental SVD algorithm [86] to three unfolded tensors. In tensor subspace tracking, based on Kronecker-structured projection, the tensor subspace is tracked by using the matrix-based subspace tracking PAST algorithm [14].

3.5.2 Partial Observation

In another recent work [87], authors have proposed an adaptive PARAFAC for incomplete streaming data. They proposed to use first-order method to minimize exponentially-weighted least-squares cost function with regularization terms. Second-order methods have been considered independently in [88] as well as in our work [6].

Chapter 4

Parallelizable Tensor Decomposition

Contents

In previous chapter, we have surveyed the approaches for large-scale tensor problem. In contrast, this chapter will consider a specific solution which is an extension of GMNS method introduced in Chapter 2.

4.1 Introduction

As stated in the previous chapter, in literature, many algorithms are developed for fitting the PARAFAC model. One of the "workhorse" algorithms for long time is the alternating least-squares (ALS). The drawback of ALS is that it may require many iterations before convergence and is not guaranteed to reach the global optimum [37].

Few works giving approximate "closed-form" [1] solutions are proposed such as [55] and [89]. These algorithms transform PARAFAC model into joint diagonalization (JD) of several sets of matrices. Although having promising results, they are either not robust with noise (as shown later) or/and expensive in term of computation complexity (required to compute HOSVD (Higher-Order-Singular Value Decomposition) [89] or a bi-linear mapping [55] before solving JD).

In this chapter, we propose a new closed-form algorithm to solve the PARAFAC problem. The key steps of our algorithm is based on solving a non-symmetrical joint diagonalization (NJD) problem. These steps comparing to [55], [89] are simpler and more robust to noise. Moreover, the performance of our algorithm is near identical with ALS's which is close to Cramer-Rao Lower Bound (CRLB) as show in [90]. We also develop a procedure which allows to integrate our algorithm in distributed/parallel computing scheme, thereby can be suitable for large-scale problem.

4.2 PARAFAC-NJD Decomposition

Let consider the matrix representation of a tensor $\mathcal{X}$ (i.e.(3.11))

$$\mathbf{X} = (\mathbf{C} \odot \mathbf{A})\mathbf{B}^T + \mathbf{N} \tag{4.1}$$

Our algorithm can be divided into three steps:

Step 1: Estimate the subspace which spans column space of matrix $\mathbf{C} \odot \mathbf{A}$ (i.e. $(\mathbf{C} \odot \mathbf{A})\mathbf{Q}$ whereas $\mathbf{Q}$ is an unknown (non-singular) matrix.

Step 2: Estimate the matrices $\mathbf{A}$ and $\mathbf{C}$,and $\mathbf{Q}$ simultaneously using non-symmetrical joint diagonalization.

[1] As commented in [89], the 'closed-form' in the literature is conflicting. Here, we consider JD and NJD to be closed-form.

Step 3: Estimate the loading matrix $\mathbf{B}$.

Now we explain these steps in detail. To estimate the subspace of the matrix $\mathbf{A} \odot \mathbf{B}$, we use the approach based on computing the covariance matrix of $\mathbf{X}$

$$\mathbf{R_x} = \frac{1}{J}(\mathbf{X}\mathbf{X}^T) = (\mathbf{C} \odot \mathbf{A})\mathbf{R_b}(\mathbf{C} \odot \mathbf{A})^T + \mathbf{R_n} \tag{4.2}$$

where $\mathbf{R_n}$ is assumed either negligible or asymptotically (i.e. for $J \gg 1$) proportional to identity matrix. Then the principal subspace is extracted by partial eigenvalue decomposition (i.e. by extracting $\mathbf{U}$, the matrix of the R principal eigenvectors of $\mathbf{R_x}$). It is straightforward to demonstrate:

$$\mathbf{U} = (\mathbf{C} \odot \mathbf{A})\mathbf{Q} \tag{4.3}$$

where $\mathbf{Q}$ is a non-singular $R \times R$ unknown matrix. The subspace estimation here can be thought as compression or dimension reduction step. In the second step, we propose to use a non-symmetrical joint diagonalization to recover the loading matrices $\mathbf{A}$ and $\mathbf{C}$, and estimate the matrix $\mathbf{Q}$ simultaneously. Observe that (4.3) can be rewritten as

$$\mathbf{M}_k = \mathbf{A}\mathbf{D}_k\mathbf{Q}, \qquad k = 1, \ldots, K \tag{4.4}$$

where $\mathbf{M}_k = \mathcal{U}_{:,:,k}$ (i.e., in slides [37]) and $\mathbf{D}_k = \text{diag}(\mathbf{C}_{k:})$. Thus, $\mathbf{A}$, $\{\mathbf{D}_k\}$, and $\mathbf{Q}$ can be found by minimizing the following function

$$f(\mathbf{A}, \mathbf{Q}, \{\mathbf{D}_k\}) = \sum_{k=1}^{K} \parallel \mathbf{M}_k - \mathbf{A}\mathbf{D}_k\mathbf{Q} \parallel_F^2 \tag{4.5}$$

There are many existing algorithms which tackle this problem, for example [91], [92], [93] or ALS [72]. At the final step, we can estimate $\mathbf{B}$ from the estimated subspace

$\mathbf{U}$ and $\mathbf{Q}$ in the first and second steps respectively

$$\mathbf{B} = (\mathbf{U}^T\mathbf{X})^T\mathbf{Q}^T \tag{4.6}$$

Here, we exploit the fact that $\mathbf{U}$ is orthogonal. Thus, we do not need to calculate $\mathbf{C} \odot \mathbf{A}$ after the second step. This algorithm is referred to as PARAFAC-NJD (NJD stands for Non-symmetrical Joint Diagonalization).

Remarks:

- First, while our approach solves a non-symmetrical joint diagonalization directly, the method in [55] needs to build a bi-linear mapping which has high computational complexity in the case of the captured data with large dimension before solving a symmetrical joint diagonalization problem. More importantly, in a noisy environment, our algorithm is more robust to noise when comparing with [55] as indicated in the simulation section.

- Second, by exploiting structure of the estimated subspaces, our method is easy to incorporate into parallel and dynamic schemes as presented in next section.

- Third, the solution of (4.4) is unique only up to scaling and permutation indeterminacy. More specifically, any set of matrices of the form $\mathbf{A}\mathbf{P}\mathbf{\Lambda}$, $\{\mathbf{\Lambda}^{-1}\mathbf{P}^T\mathbf{D}_k\mathbf{P}\tilde{\mathbf{\Lambda}}^{-1}\}$, and $\tilde{\mathbf{\Lambda}}\mathbf{P}^T\mathbf{Q}$ is also a solution where $\mathbf{P}$ is a permutation matrix, and $\mathbf{\Lambda}$ and $\tilde{\mathbf{\Lambda}}$ are diagonal matrices.

4.2.1 Parallel PARAFAC-NJD Decomposition

In this section, we develop a parallel version for the proposed algorithm. In the previous chapter, GMNS method is proposed to estimate principal and minor subspaces in a parallel scheme. It allows to reduce overall numerical cost by a factor of L^2 (L is number of available DSPs) while still preserving the estimation accuracy. This

Table 4.1: Summary of Parallel PARAFAC-NJD algorithm

Input: Matrix representation of large-scale tensor $\mathcal{X}$

1. Extract non-overlapping sub-tensors as described in 4.2.1.1, or overlapping ones as described in 4.2.1.2
2. Apply PARAFAC-NJD algorithm:
   (a) Parallel principal subspace estimation using GMNS
   (b) Extract sub-loading matrices using NJD
3. Solve permutation and scale problem

Output: estimated loading matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$

algorithm can be applied to estimate the principal subspace $\mathbf{U}$. Moreover, we further reduce the complexity of solving NJD problem in our algorithm by proposing a new procedure which has the spirit close to the GMNS algorithm.

Our new procedure has two key steps besides applying PARAFAC-NJD procedure. The first one is to divide a big tensor into sub-tensors. We consider non-overlapping and overlapping cases. The second one is to solve permutation problem between the estimated loading matrices from each sub-tensor. Summary of all steps are presented in Table 4.1.

4.2.1.1 Parallel PARAFAC-NJD Decomposition with Non-overlapping Tensor Partitioning

Assume that, we want to divide a big tensor into L non-overlapping sub-tensors along K-mode such that $d = K/L$ (for simplicity, d is assumed to be integer-valued). Thus, l-th subtensor can be represented in matrix and slab form respectively as follows

$$\mathbf{X}^l = (\mathbf{C}_{d(l-1)+1:dl,:} \odot \mathbf{A})\mathbf{B}^T \tag{4.7}$$

and,

$$\mathbf{X}_k^l = \mathbf{A}\mathbf{D}_k^l\mathbf{B}^T, \qquad k = 1, \ldots, d \tag{4.8}$$

Following this partitioning procedure, the subtensors still keep the same structure like the original one but having a smaller dimension (i.e $I \times J \times d$ for each).

After partitioning tensor, we apply the PARAFAC-NJD algorithm for each subtensor. Specifically, the estimated subspace $\mathbf{U}^l$ of tensor $\mathcal{X}^l$ yields

$$\mathbf{U}^l = (\mathbf{C}^l \odot \mathbf{A}^l)\mathbf{Q}^l. \tag{4.9}$$

where $\mathbf{C}^l \in \mathbb{R}^{d\times R}$, $\mathbf{A}^l \in \mathbb{R}^{I\times R}$, and $\mathbf{Q}^l \in \mathbb{R}^{R\times R}$. Then we find $\mathbf{C}^l$, $\mathbf{A}^l$ and $\mathbf{Q}^l$ by minimizing the function

$$f(\mathbf{A}^l, \mathbf{Q}^l, \{\mathbf{D}_k^l\}) = \sum_{k=1}^{d} \parallel \mathbf{M}_k^l - \mathbf{A}^l\mathbf{D}_k^l\mathbf{Q}^l \parallel^2 \tag{4.10}$$

Similar to (4.6), the loading matrix $\mathbf{B}^l$ can be calculated as

$$\mathbf{B}^l = ((\mathbf{U}^l)^T\mathbf{X}^l)^T(\mathbf{Q}^l)^T \tag{4.11}$$

where $\mathbf{B}^l \in \mathbb{R}^{J\times R}$. Until now, we have a set of matrices $\{\mathbf{A}^l\}$, $\{\mathbf{B}^l\}$ and $\{\mathbf{C}^l\}$. Note that, $\{\mathbf{A}^l\}$ and $\{\mathbf{B}^l\}$ are estimates of $\mathbf{A}$ and $\mathbf{B}$ correspondingly up to scale and permutation. On the other hand, each $\mathbf{C}^l$ presents a sub-matrix of $\mathbf{C}$ (see (4.20) and (4.21)).

We now solve scaling and permutation problem. For loading matrix $\mathbf{A}$, we consider loading matrices of two successive subtensors, $\mathbf{A}^l = \mathbf{A}\mathbf{P}^l\mathbf{\Lambda}^l$ and $\mathbf{A}^{l+1} = \mathbf{A}\mathbf{P}^{l+1}\mathbf{\Lambda}^{l+1}$.

Then, for each column of matrix $\mathbf{A}^l$, we search

$$i = \underset{j}{\operatorname{argmax}} \frac{|(\mathbf{A}^l_{:,k})^T(\mathbf{A}^{l+1}_{:,j})|}{\|\mathbf{A}^l_{:,k}\|\|\mathbf{A}^{l+1}_{:,j}\|} \qquad k = 1, \ldots, R \tag{4.12}$$

$$\mathbf{P}^{l+1,l}(i,k) = 1 \tag{4.13}$$

$$\mathbf{\Lambda}^{l+1,l}(i,i) = \frac{(\mathbf{A}^{l+1}_{:,i})^T(\mathbf{A}^l_{:,k})}{\|\mathbf{A}^{l+1}_{:,i}\|} \tag{4.14}$$

where initialization of $\mathbf{P} = \mathbf{0}$ which is an $R \times R$ zero matrix and initialization of $\mathbf{\Lambda} = \mathbf{I}$ which is an identity matrix. Thus, loading matrix $\mathbf{A}$ can be found by (by informal way)

$$\mathbf{A}^{l+1} = \mathbf{A}^{l+1}(\mathbf{\Lambda}^{l+1,l})(\mathbf{P}^{l+1,l}) \tag{4.15}$$

For loading matrix $\mathbf{B}$, we note that

$$\mathbf{B}^l = ((\mathbf{U}^l)^T\mathbf{X}^l)^T(\mathbf{Q}^l)^T \tag{4.16}$$

$$= \mathbf{B}\mathbf{P}^l(\tilde{\mathbf{\Lambda}}^l)^T \tag{4.17}$$

We thus can apply the same procedure in previous description to $\mathbf{B}^l$ and $\mathbf{B}^{l+1}$ to find out $\tilde{\mathbf{\Lambda}}^{l+1,l}$. Consequently,

$$\mathbf{B}^{l+1} = \mathbf{B}^{l+1}(\tilde{\mathbf{\Lambda}}^{l+1,l})(\mathbf{P}^{l+1,l}) \tag{4.18}$$

Finally, from results of $\mathbf{A}$ and $\mathbf{B}$, each row of loading matrix $\mathbf{C}^{l+1}$ is given by

$$\mathbf{D}^{l+1}_k = (\mathbf{\Lambda}^{l+1,l})^{-1}(\mathbf{P}^{l+1,l})^T\mathbf{D}^{l+1}_k(\mathbf{P}^{l+1,l})(\tilde{\mathbf{\Lambda}}^{l+1,l})^{-1} \tag{4.19}$$

We refer to this procedure as PARAFAC-NJD-NO.

4.2.1.2 Parallel PARAFAC-NJD Decomposition with Overlapping Tensor Partitioning

Similar to non-overlapping case, we want to divide a big tensor into L overlapping sub-tensors along K-mode. Let q be the number of overlapping slides. Hence, each overlapping tensor has $d+q$ slabs. The matrix and slab representation of this procedure for two successive subtensor $\mathcal{X}^l$ and $\mathcal{X}^{l+1}$ are given by

$$\mathbf{X}^l = (\mathbf{C}_{d(l-1)+1:dl+q,:} \odot \mathbf{A})\mathbf{B}^T \tag{4.20}$$

$$\mathbf{X}^{l+1} = (\mathbf{C}_{dl+1:d(l+1)+q,:} \odot \mathbf{A})\mathbf{B}^T \tag{4.21}$$

and

$$\mathbf{X}_k^l = \mathbf{A}\mathbf{D}_k^l\mathbf{B}^T, \qquad k = 1, \ldots, d+q \tag{4.22}$$

$$\mathbf{X}_k^{l+1} = \mathbf{A}\mathbf{D}_k^{l+1}\mathbf{B}^T \tag{4.23}$$

respectively, where $\mathbf{D}_k^l = \text{diag}(\mathbf{C}_{k:}^l)$ and $\mathbf{D}_k^{l+1} = \text{diag}(\mathbf{C}_{k:}^{l+1})$ and $\mathbf{C}^l = \mathbf{C}_{d(l-1)+1:dl+q,:}$ and $\mathbf{C}^{l+1} = \mathbf{C}_{dl+1:d(l+1)+q,:}$. Each subtensor now has size of $I \times J \times (d+q)$. We then apply the proposed algorithm for each sub-tensor to get a set of matrices $\{\mathbf{A}^l\} \in \mathbb{R}^{I\times R}$, $\{\mathbf{B}^l\} \in \mathbb{R}^{J\times R}$ and $\{\mathbf{C}^l\} \in \mathbb{R}^{(d+q)\times R}$. The scale and permutation problem can be solved by applying the same procedure in non-overlapping case for $\mathbf{A}$ and $\mathbf{B}$.

For the loading matrix $\mathbf{C}$, we consider two successive sub-tensors which yield $\mathbf{C}^l$ and

$\mathbf{C}^{l+1}$

$$\mathbf{C}^{l} = \begin{bmatrix} \mathbf{C}^{l,nonoverlap} \\ \mathbf{C}^{l,overlap} \end{bmatrix} \mathbf{P}^{l}\mathbf{\Lambda}^{l} = \begin{bmatrix} \mathbf{C}_{d(l-1)+1:dl,:} \\ \mathbf{C}_{dl+1:dl+q,:} \end{bmatrix} \mathbf{P}^{l}\mathbf{\Lambda}^{l} \tag{4.24}$$

$$\mathbf{C}^{l+1} = \begin{bmatrix} \mathbf{C}^{l+1,overlap} \\ \mathbf{C}^{l+1,nonoverlap} \end{bmatrix} \mathbf{P}^{l+1}\mathbf{\Lambda}^{l+1} = \begin{bmatrix} \mathbf{C}_{dl+1:d(l+1)} \\ \mathbf{C}_{d(l+1)+1:d(l+1)+q} \end{bmatrix} \mathbf{P}^{l+1}\mathbf{\Lambda}^{l+1} \tag{4.25}$$

We then apply the described procedure for overlapping part of $\mathbf{C}^{l}$ and $\mathbf{C}^{l+1}$. Thus, loading matrix $\mathbf{C}$ is given by

$$\mathbf{C}^{l+1} = \mathbf{C}^{l+1}(\mathbf{\Lambda}^{l+1,l})(\mathbf{P}^{l+1,l}) \tag{4.26}$$

After matrices alignment, we can improve the estimates of $\mathbf{A}$ and $\mathbf{B}$ by averaging

$$\hat{\mathbf{A}} = \frac{1}{L}\sum_{1}^{L}\mathbf{A}^{l} \tag{4.27}$$

and

$$\hat{\mathbf{B}} = \frac{1}{L}\sum_{1}^{L}\mathbf{B}^{l} \tag{4.28}$$

We refer to this procedure as PARAFAC-NJD-O.

4.3 Simulations

To assess the performance of the proposed algorithms, we use the same set-up as in [55]. The entries of a tensor $\mathcal{X}$ whose rank is R are drawn from a zero-mean unit-variance Gaussian distribution. A Gaussian noise term $\mathcal{N}$ is then added to this tensor

$$\tilde{\mathcal{X}} = \frac{\mathcal{X}}{\| \mathcal{X} \|_F} + \sigma_N \frac{\mathcal{N}}{\| \mathcal{N} \|_F} \tag{4.29}$$

I	J	K	R	L	q
20	20	40	8	2;4	4

Table 4.2: Particular parameters are set in our experiments.

where the value of σ_N controls the noise level. The accuracy of the estimated loading matrices ($\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$) and matrix representation of tensor $\mathcal{X}$ ($\mathbf{X}$) is the relative error defined as

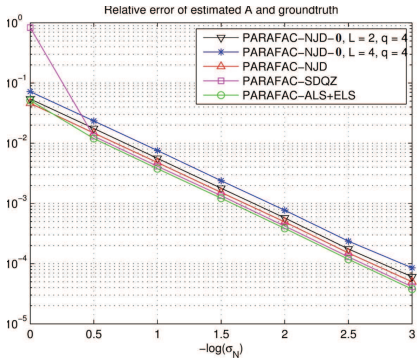
$$\rho(N) = \frac{1}{N}\sum_{n=1}^{N}\frac{\parallel \mathbf{H}_n - \mathbf{H}_{ex} \parallel_F}{\parallel \mathbf{H}_{ex} \parallel_F} \tag{4.30}$$

where N is the number of Monte Carlo runs and $\mathbf{H}_i$ is the estimate of the exact matrix $\mathbf{H}_{ex}$. In all experiments, N is set to 100. We compare the performance of our algorithms with two state-of-the-art ones, ALS+ELS from N-way toolbox[1] and PARAFAC-SDQZ [55]. Parameters for experiments are shown in Table4.2. Rank R is chosen to be small comparing to dimensions of tensor. The results from Figure 4.1 and 4.2 show that performance of PARAFAC-NJD is comparable to ALS+ELS and PARAFAC-SDQZ but with reduced computational cost. Moreover, it is more robust than PARAFAC-SDQZ in the noisy environment (i.e. $-log(\sigma_N) < 0.5$). Figures 4.1 and 4.2 also illustrate the effectiveness of proposed procedures in parallel computing scheme (likewise, for dynamic tensor analysis with the same parameters). The PARAFAC-NJD-0 ($L = 2, q = 4$) has slight performance loss comparing to the three batch algorithms. When number of processors L increases, PARAFAC-NJD-0 ($L = 4, q = 4$) gain more in term of complexity and have small performance loss. Thus changing L allows to balance between the accuracy and the complexity.

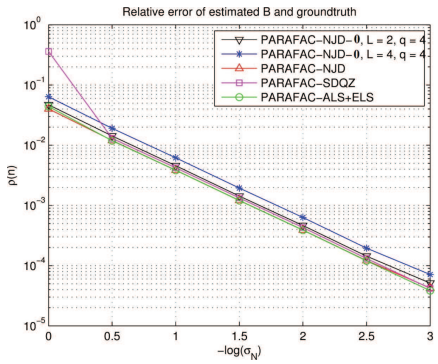
[1]N-way toolbox version 3.30 (including enhanced line search procedure [72]) is downloaded from http://www.models.life.ku.dk/nwaytoolbox

4.4 Conclusion

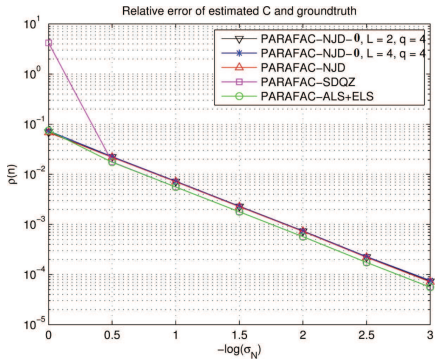
We have described a new algorithm for tackling PARAFAC problem based on solving a non-symmetrical joint diagonalization problem. Furthermore, a new procedure for solving scale and permutation ambiguities between the estimated loading matrices from different DSPs has been proposed to reduce complexity in a distributed/parallel computing scheme. In simulation, we have compared the proposed methods to ALS+ELS and PARAFAC-SDQZ and observed that the performance of our algorithm is similar to or even slightly better in specific contexts.



(a) Loading matrix **A**



(b) Loading matrix **B**



(c) Loading matrix **C**

Figure 4.1: Performance comparison of the proposed algorithms with ALS+ELS and PARAFAC-SDQZ (loading matrices)

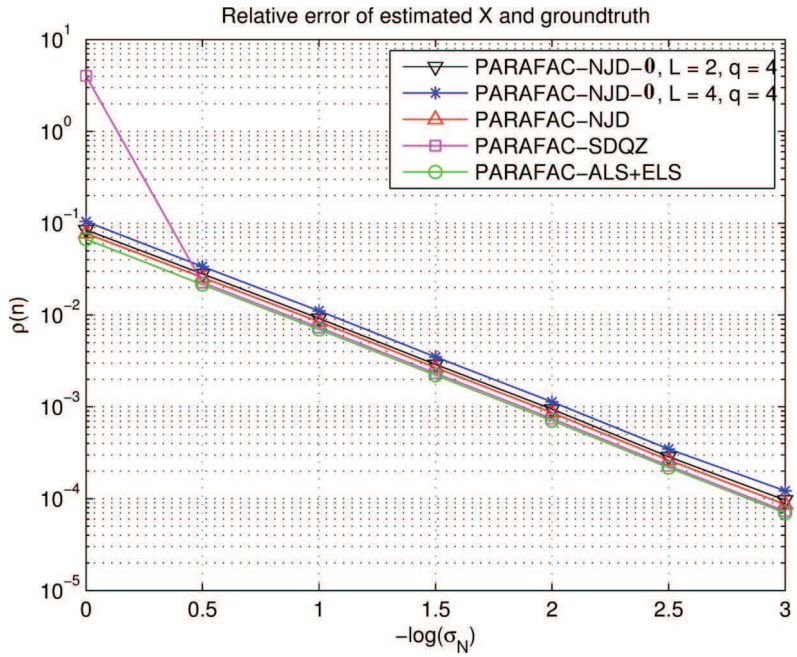


Figure 4.2: Performance comparison of the proposed algorithms with ALS+ELS and PARAFAC-SDQZ (matrix representation of tensor $\mathcal{X}$).

Chapter 5

Robust Tensor Decomposition

Contents

In Chapter 4, we have assumed that tensor rank is known in advance. In practice, Moreover, we did not examine constraints on the model. We thus address to bring those problems together to cope with in this chapter. Our algorithms here, when stand alone, are inappropriate for large-scale problem. However, they can be integrated into the divide-and-conquer framework as described in Chapter 3.

5.1 Introduction

To date, various methods have been introduced and developed for computing tensor decomposition. We can classify them into three main approaches: alternating approach, general optimization approach and algebraic approach. The alternating approach optimizes one factor at each step while keeping the others fixed. The general optimization approach casts the tensor decomposition problem into a nonlinear equation problem and then solve it using standard optimization tools, such as gradient methods. In a different aspect, as its name reveals, the algebraic approach aims to solve the tensor decomposition problem by using only algebra operators. We refer the reader to [76, 94] for a comparion of different algorithms for PARAFAC decomposition.

For a long time, the alternating approach is considered as the "workhorse" one because of its simplicity and relative efficiency. However, an *all-at-once* optimization framework (see [76] and references therein), which has been recently proposed, has been reported to be as accurate as alternating-based algorithms while being more robust in the over-factoring case[1]. While estimating the rank of a tensor is NP-complete [37] and still an open problem, robustness is a desirable characteristic.

Moreover, among various constraints, two popular ones are non-negativity and sparsity because many problems come naturally with them, such as in text, image, and EEG signal analyses. As a result, many tensor decomposition algorithms with non-negativity and sparseness constraints have been proposed. We refer the reader to [37, 38, 40] for comprehensive review. For PARAFAC, imposing a non-negativity constraint, when applicable, not only improves the physical interpretation [38] but also helps to avoid diverging components [95, 96]. Enforcing non-negativity and sparsity

[1] Over-factoring means that for example in PARAFAC we choose a tensor rank is larger than its true value.

on factors and/or the core tensor also helps to improve the uniqueness of the Tucker decomposition [97]. However, as will be indicated in the sequel, even enforcing non-negativity and sparsity on factors, the state-of-the-art algorithms are not robust to over-factoring. Our work aims to overcome this shortcoming by bring these two desired characteristics together. In particular, we would like to have algorithms which are robust with over-factoring and easy to use with either non-negativity constraint or both non-negativity and sparseness constraints.

Our contributions are two folds: First, we tailor the all-at-one optimization framework [76] to impose non-negativity constraint or both non-negativity and sparseness constraints for tensor decomposition. Second, we apply this framework to two case studies: (i) sparse non-negative PARAFAC decomposition, and (ii) sparse non-negative Tucker decomposition. As shown in the simulation section, our sparse non-negative algorithms are as accurate as state-of-the-art algorithms but more robust to over-factoring. We would like to highlight that the all-at-once optimization approach has been applied to the PARAFAC decomposition with missing entries [98], coupled matrix/tensor decomposition without/with missing value [99]. However, to our best knowledge, sparse non-negative PARAFAC and Tucker ones have not been proposed in the context of all-at-one optimization.

5.2 All-at-once Optimization with Nonnegative and Sparse Constraints

The all-at-once optimization approach [76] includes three steps:

Step 1: Define a cost function f and variable $\mathbf{x}$.

Step 2: Compute the gradient $\nabla f(\mathbf{x})$.

Step 3: Optimize using nonlinear conjugate gradient.

Table 5.1: All-at-once optimization with non-negativity constraints

Step 1: Define a cost function f and variable $\mathbf{x}$.
Step 2: Compute the gradient $\nabla f(\mathbf{x})$.
Step 3: Optimize using first order projected gradient.

The special point comes from variable $\mathbf{x}$ which is a concatenation of the vectorized forms of the loading matrices for PARAFAC, or vectorized forms of the loading factors and the core tensor for Tucker (see next sections for more details).

To impose the non-negativity constraint or both the non-negativity and sparseness constraints, we propose to replace the nonlinear conjugate gradient by a first-order projected gradient method. This framework is simple and easy to implement. Moreover, it allows us to solve a large class of problems (*e.g.*, different models) with non-negativity and sparseness constraints. When the sparseness constraint comes after the non-negativity constraint, it is straightforward to calculate the gradient of variable with l_1 norm following the rule $\frac{\partial \|\mathbf{x}\|_1}{\partial \mathbf{x}} = \mathbf{1}$ where $\mathbf{1}$ is a vector[1] whose entries are one. Thus, we can use the proposed framework without modifying Step 3 in the optimization algorithm. A summary of the framework is presented in Algorithm 5.1. We adapt the projected gradient algorithm [100] to use in Step 3. We choose this algorithm because it is well-understood and the results for convergence are available [101]. The algorithm is a variant of the projected gradient algorithm using the Armijo rule along the projection arc [102]. In particular, the step size in the following update step:

$$\mathbf{x}^{(k+1)} = [\mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k)]^+ \tag{5.1}$$

[1]Here, we will "over-use" notation $\mathbf{1}$ which can be vector or matrix depending on the variable size

should be such that the following condition is satisfied:

$$f(\mathbf{x}^{k+1}) - f(\mathbf{x}^k) \leqslant \sigma \nabla f(\mathbf{x}^k)^T (\mathbf{x}^{k+1} - \mathbf{x}^k), \tag{5.2}$$

where $\alpha_k = \beta^{t_k}$ and t_k is the first integer which makes (5.2) hold and σ is a pre-defined positive scalar. The main idea now is to reduce the search time of α_k by using α_{k-1} as an initial guess. This search strategy stems from the fact that α_k and α_{k-1} may be close. Thus we can increase or decrease α_k until the largest β^{t_k} that satisfies (5.2) is found. A summary of the projected gradient is presented in Table 1.

For stopping condition, we terminate the algorithm if $\|[\nabla f(\mathbf{x}^k)]^+\| \leqslant \epsilon \|[\nabla f(\mathbf{x}^1)]\|$ or if the algorithm reaches a maximum number of iterations. The former means that we stop if a solution at the k-th step $\mathbf{x}^k$ is close to a stationary point where $\epsilon > 0$ is a small user-defined number. The latter is to avoid a too-long run-time.

Algorithm 1: Modified projected gradient [100].

Input : $\mathbf{x}_0 \in \mathbb{R}_+$
Output: $\mathbf{x} \in \mathbb{R}_+$ such that (5.2) is minimized

initialization for $\beta \in (0,1)$, $\sigma \in (0,1)$, $\alpha_0 = 1$
while *a stopping condition is not met* **do**
  if *(5.2) holds,* **then**
    repeat
      $\alpha_k \leftarrow \alpha_k / \beta$
      $\hat{\mathbf{x}} \leftarrow [\mathbf{x} - \alpha_k \nabla f(\mathbf{x}^k)]^+$
    until *(5.2) does not hold, or* $\hat{\mathbf{x}} = \mathbf{x}$;
  else
    repeat
      $\alpha_k \leftarrow \alpha_k \beta$
      $\hat{\mathbf{x}} \leftarrow [\mathbf{x} - \alpha_k \nabla f(\mathbf{x}^k)]^+$
    until *(5.2) holds, or* $\hat{\mathbf{x}} = \mathbf{x}$;
  end
end

5.3 First Case Study: PARAFAC Model

In this section, we define the cost function for the sparse non-negative PARAFAC decomposition as

$$\begin{aligned} &\text{minimize} && f(\mathbf{A},\mathbf{B},\mathbf{C}) \\ &\text{subject to} && \mathbf{A} \geqslant 0, \mathbf{B} \geqslant 0, \mathbf{C} \geqslant 0 \end{aligned}$$

where

$$\begin{aligned} f(\mathbf{A},\mathbf{B},\mathbf{C}) =& \frac{1}{2}\|\mathcal{X} - [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\|_F^2 \\ &+ \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1 \end{aligned} \tag{5.3}$$

and λ_A, λ_B and λ_C are penalty terms which control the regularization strength. We can write out three equivalent expressions of (5.3) in terms of unfolded tensors as follows:

$$\begin{aligned} f(\mathbf{A},\mathbf{B},\mathbf{C}) =& \frac{1}{2}\|\mathbf{X}_{(1)} - \mathbf{A}(\mathbf{C}\odot\mathbf{B})^T\|_F^2 \\ &+ \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1 \end{aligned} \tag{5.4}$$

$$\begin{aligned} f(\mathbf{A},\mathbf{B},\mathbf{C}) =& \frac{1}{2}\|\mathbf{X}_{(2)} - \mathbf{B}(\mathbf{C}\odot\mathbf{A})^T\|_F^2 \\ &+ \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1 \end{aligned} \tag{5.5}$$

$$\begin{aligned} f(\mathbf{A},\mathbf{B},\mathbf{C}) =& \frac{1}{2}\|\mathbf{X}_{(3)} - \mathbf{C}(\mathbf{B}\odot\mathbf{A})^T\|_F^2 \\ &+ \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1. \end{aligned} \tag{5.6}$$

These expressions are convenient when computing the partial derivatives corresponding to variables $\mathbf{A}$ or $\mathbf{B}$ or $\mathbf{C}$. To be specific, we have

$$\frac{\partial f}{\partial \mathbf{A}} = -\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B}) + \mathbf{A}(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B}) + \lambda_A \mathbf{1} \tag{5.7}$$

$$\frac{\partial f}{\partial \mathbf{B}} = -\mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A}) + \mathbf{B}(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A}) + \lambda_B \mathbf{1} \tag{5.8}$$

$$\frac{\partial f}{\partial \mathbf{C}} = -\mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A}) + \mathbf{C}(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A}) + \lambda_C \mathbf{1}. \tag{5.9}$$

We then obtain the gradient by concatenating their vectorized forms as

$$\nabla f(\mathbf{x}) = \left[\text{vec}(\frac{\partial f}{\partial \mathbf{A}})^T, \text{vec}(\frac{\partial f}{\partial \mathbf{B}})^T, \text{vec}(\frac{\partial f}{\partial \mathbf{C}})^T \right]^T, \tag{5.10}$$

where $\mathbf{x} = [\text{vec}(\mathbf{A})^T, \text{vec}(\mathbf{B})^T, \text{vec}(\mathbf{C})^T]^T$. Finally, we use this gradient in Step 2 of the framework given by Table 5.1. We refer to this implementation as the All-at-once optimization based Sparse Nonegative PARAFAC (ASNP) algorithm.

5.4 Second Case Study: Tucker Model

Similar to the PARAFAC case, we present a sparse non-negative Tucker decomposition. Depending on applications, we can choose to impose a sparseness constraint on several factors and the core tensor, or only the core tensor. If dimensions of the core tensor are smaller than those of the underlying tensor, we have the standard Tucker model. Otherwise, we have a tensor dictionary learning problem. Consider the following cost function:

$$\begin{aligned} &\text{minimize} && h(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathcal{G}) \\ &\text{subject to} && \mathbf{A} \geqslant 0, \mathbf{B} \geqslant 0, \mathbf{C} \geqslant 0, \mathcal{G} \geqslant 0 \end{aligned} \tag{5.11}$$

where

$$h(\mathbf{A},\mathbf{B},\mathbf{C},\mathcal{G}) = \frac{1}{2}\|\mathcal{Y} - [\![\mathcal{G};\mathbf{A},\mathbf{B},\mathbf{C}]\!]\|_F^2 + \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1 + \lambda_G\|\mathcal{G}\|_1 \tag{5.12}$$

and λ_A, λ_B, λ_C and λ_G are penalty terms of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathcal{G}$ respectively. We can write out four equivalent expressions of (5.12) in terms of unfolded tensors and vectorized form as follows:

$$h(\mathbf{A},\mathbf{B},\mathbf{C},\mathcal{G}) = \frac{1}{2}\|\mathbf{Y}_{(1)} - \mathbf{A}\mathbf{G}_{(1)}(\mathbf{C}\otimes\mathbf{B})^T\|_F^2 + \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1 + \lambda_G\|\mathbf{G}_{(1)}\|_1 \tag{5.13}$$

$$h(\mathbf{A},\mathbf{B},\mathbf{C},\mathcal{G}) = \frac{1}{2}\|\mathbf{Y}_{(2)} - \mathbf{B}\mathbf{G}_{(2)}(\mathbf{C}\otimes\mathbf{A})^T\|_F^2 + \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1 + \lambda_G\|\mathbf{G}_{(2)}\|_1 \tag{5.14}$$

$$h(\mathbf{A},\mathbf{B},\mathbf{C},\mathcal{G}) = \frac{1}{2}\|\mathbf{Y}_{(3)} - \mathbf{C}\mathbf{G}_{(3)}(\mathbf{B}\otimes\mathbf{A})^T\|_F^2 + \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1 + \lambda_G\|\mathbf{G}_{(3)}\|_1 \tag{5.15}$$

$$h(\mathbf{A},\mathbf{B},\mathbf{C},\mathcal{G}) = \frac{1}{2}\|\mathbf{y} - (\mathbf{C}\otimes\mathbf{B}\otimes\mathbf{A})\mathbf{g}\|_F^2 + \lambda_A\|\mathbf{A}\|_1 + \lambda_B\|\mathbf{B}\|_1 + \lambda_C\|\mathbf{C}\|_1 + \lambda_G\|\mathbf{g}\|_1. \tag{5.16}$$

The partial derivatives of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathcal{G}$ can then be obtained as

$$\begin{aligned}
\frac{\partial h}{\partial \mathbf{A}} &= -\mathbf{X}_{(1)}(\mathbf{C}\otimes\mathbf{B})\mathbf{G}_{(1)}^T \\
&\quad + \mathbf{A}\mathbf{G}_{(1)}(\mathbf{C}^T\mathbf{C}\otimes\mathbf{B}^T\mathbf{B})\mathbf{G}_{(1)}^T + \lambda_A\mathbf{1} \qquad (5.17)\\
\frac{\partial h}{\partial \mathbf{B}} &= -\mathbf{X}_{(2)}(\mathbf{C}\otimes\mathbf{A})\mathbf{G}_{(2)}^T \\
&\quad + \mathbf{B}\mathbf{G}_{(2)}(\mathbf{C}^T\mathbf{C}\otimes\mathbf{A}^T\mathbf{A})\mathbf{G}_{(2)}^T + \lambda_B\mathbf{1} \qquad (5.18)\\
\frac{\partial h}{\partial \mathbf{C}} &= -\mathbf{X}_{(3)}(\mathbf{B}\otimes\mathbf{A})\mathbf{G}_{(3)}^T \\
&\quad + \mathbf{C}\mathbf{G}_{(3)}(\mathbf{B}^T\mathbf{B}\otimes\mathbf{A}^T\mathbf{A})\mathbf{G}_{(3)}^T + \lambda_C\mathbf{1} \qquad (5.19)\\
\frac{\partial h}{\partial \mathbf{g}} &= -(\mathbf{C}^T\otimes\mathbf{B}^T\otimes\mathbf{A}^T)\mathbf{g} \\
&\quad + (\mathbf{C}^T\mathbf{C}\otimes\mathbf{B}^T\mathbf{B}\otimes\mathbf{A}^T\mathbf{A})\mathbf{g} + \lambda_G\mathbf{1}. \qquad (5.20)
\end{aligned}$$

By concatenating their vectorized form, we attain

$$\nabla h(\mathbf{x}) = \left[\mathrm{vec}\left(\frac{\partial h}{\partial \mathbf{A}}\right)^T, \mathrm{vec}\left(\frac{\partial h}{\partial \mathbf{B}}\right)^T, \mathrm{vec}\left(\frac{\partial h}{\partial \mathbf{C}}\right)^T, \left(\frac{\partial h}{\partial \mathbf{g}}\right)^T\right]^T, \qquad (5.21)$$

where $\mathbf{x} = [\mathrm{vec}(\mathbf{A})^T, \mathrm{vec}(\mathbf{B})^T, \mathrm{vec}(\mathbf{C})^T, \mathbf{g}^T]^T$. We use the above gradient in place of the gradient in Step 2 of the framework of Table 5.1. We refer to this implementation as the All-at-once optimization based Sparse Non-negative Tucker (ASNT) algorithm.

5.5 Simulation

In this section, the performance of the proposed algorithms are assessed. We implement the considered algorithms using the Matlab Tensor Toolbox in Matlab [103]. Parameters of the algorithms are summarized in Table 6.5. Because of limited space, we select two experiments to compare our algorithms with several available algorithms from literature. In all experiments, we keep all default parameters of the comparing

α	β	σ	maxIter	ϵ	λ_A	λ_B	λ_C	λ_G
1	0.1	0.01	1000	10^{-4}	0.01	0	0	N/A
1	0.1	0.01	1000	10^{-4}	0	0	0	0

Table 5.2: Particular parameters set in our experiment.

algorithms.

To assess the accuracy of ASNP[1] and its robustness, we compare our algorithm in over-factoring case with two state-of-the-art algorithms: CPOPT [76] from the Tensor Toolbox and the Non-negative Alternating Least-Squares (NALS-PARAFAC) from the N-way Toolbox [104]. In particular, we use amino acids fluorescence data from [105]. This data is a third-order tensor corresponding to 5 chemical samples measured by fluorescence at 61 excitation and 201 emission wavelengths. The true rank of the tensor is three. Since PARAFAC is unique up to scales and permutation, we follow the normalization method proposed in [105]. Particularly, we normalize the loading vectors of the second and third modes and keep the variance of the corresponding vectors in the first mode. Then we arrange them in descending order of amplitude. This normalization method was also used for CPOPT when comparing with the CP-ALS algorithm in [76]. As shown in Figure 5.1, our method is accurate and more robust than CPOPT and NALS-PARAFAC.

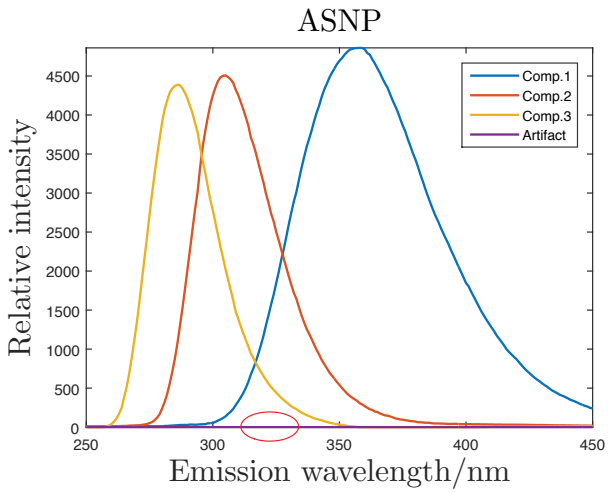
To assess the performance of ANST, we randomly generate a $20 \times 20 \times 20$ non-negative tensor following the Tucker model (3.15). The core tensor size is set to $5 \times 5 \times 5$. Then, the noisy observation is $\tilde{\mathcal{Y}} = \mathcal{Y} + \eta \frac{\|\mathcal{Y}\|_F}{\|\mathcal{N}\|_F} \mathcal{N}$, where $\mathcal{N}(t)$ is the noise whose size is identical to that of $\mathcal{Y}$ and parameter η controls the noise level. We compare performance of ANST with the Block Coordinate Descent based Tucker (BCDT) in [106]. The relative error was used as the performance criterion $\varrho = \frac{\|\mathcal{Y} - \mathcal{Y}_{est}\|_F}{\|\mathcal{Y}\|_F}$ where $\mathcal{Y}_{est}$ is the estimated tensor. The average result obtained from 100 Monte Carlo

[1] With chosen parameters, the result of the ASNP algorithm for the non-negativity constraint is almost identical to that for both the sparseness and non-negativity constraints. Thus, we decided to present the latter only.

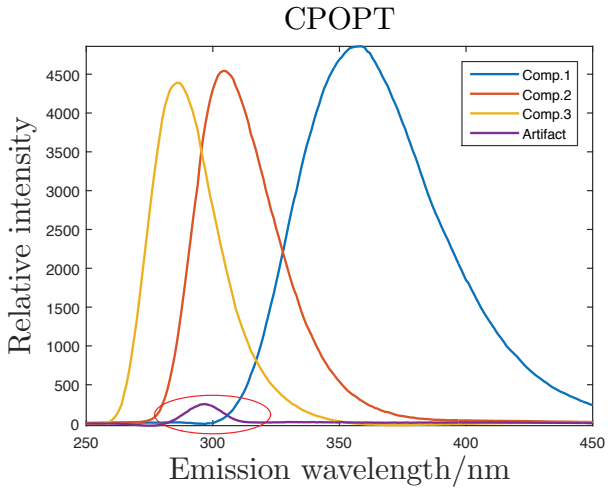
simulation runs is shown in Figure 6.2. We can see that the performance of ANST is slightly better than that of BCDT. However, we confirm that our algorithm is slower than BCDT and our future work will focus on improving this drawback.

5.6 Conclusion

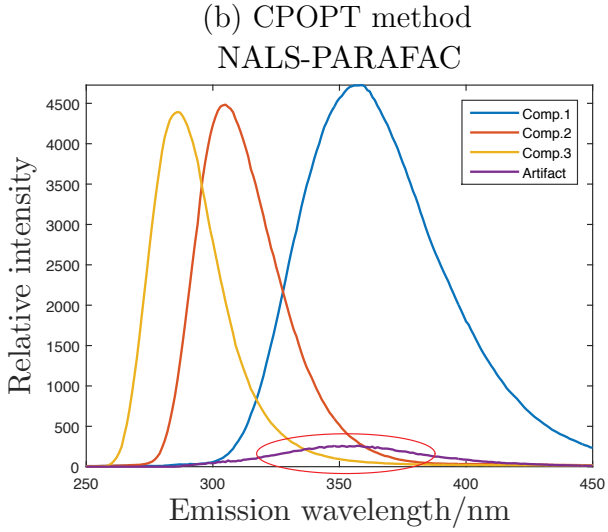
We have introduced in this chapter two new tensor decomposition algorithms that take into account the sparsity and non-negativity of the factors. Compared to other existing methods, the proposed solution has the advantages of simplicity and robustness to tensor-rank estimation errors. Simulation results have been provided to illustrate the effectiveness and robustness of the algorithms for both simulated and real-life data.



(a) Our method



(b) CPOPT method



(c) NALS-PARAFAC method

Figure 5.1: Illustration of loading components estimated from three methods for overfactoring case, $R = R_{true} + 1$.

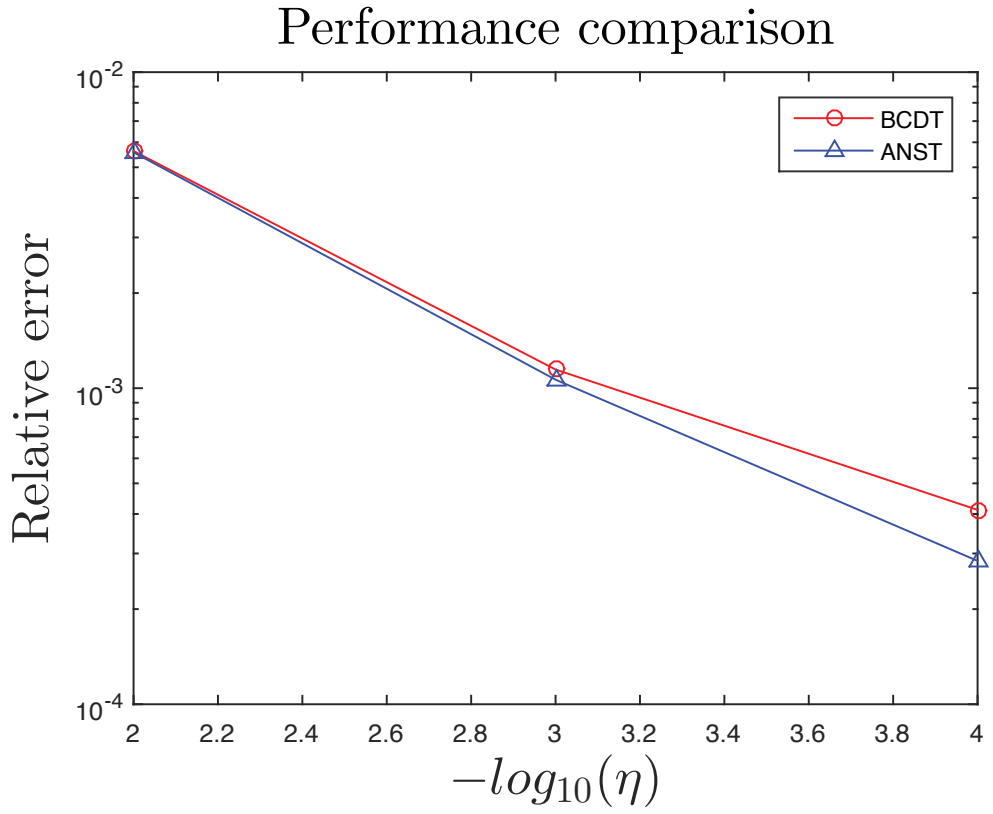


Figure 5.2: Performance comparison of ANST and BCDT.

Chapter 6

Fast Adaptive Tensor Decomposition

Contents

So far we have discussed about tensor decomposition in batch setting in previous chapters of the second part. However many applications requires processing with time constraint or even realtime. In this chapter, we will study PARAFAC model in adaptive setting, so-called adaptive PARAFAC.

6.1 Introduction

With the recent advances on sensor and streaming technologies, processing massive volumes of data (or "big data") with time constraints or even in real-time is not only crucial but also challenging [3], in a wide range of applications including MIMO radars [80], biomedical imaging [87], and signal processing [40].

A typical situation in those cases is that data are acquired along multiple dimensions, one among which is time. Such data can be naturally represented by tensors. Tensor decomposition, thereby, can be used as an important tool to analyze, understand or eventually compress data. In this chapter, adaptive PARAFAC decomposition is the method of interest.

For streaming tensors, direct application of batch (i.e., offline) PARAFAC decomposition is computationally demanding. Instead, an *adaptive* (i.e., incremental) approach is more suitable and, hence, should provide a good trade-off between quality

and efficiency. In contrast to adaptive filtering [107] or subspace tracking [16,108,109], which have a long standing history and are well-understood, adaptive tensor decomposition has received little attention so far.

In [80], Nion and Sidiropoulos proposed an adaptive decomposition model for a class of third-order tensors that have one dimension growing with time. Accordingly, they proposed two algorithms: recursive least-squares tracking (PARAFAC-RLST) and simultaneous diagonalization tracking (PARAFAC-SDT). In [87], Mardani, Mateos and Giannakis also proposed an adaptive PARAFAC method for streaming data under partial observation. A common basis in these studies is the use of first-order methods (i.e., using gradients) to optimize an exponentially weighted least-squares cost function.

In this chapter, we propose two algorithms for full streaming data. In particular, first algorithm, called 3D-OPAST, generalizes the orthonormal projection approximation subspace tracking (OPAST) algorithm [15] The second algorithm, called SOAP, is based on second-order optimization. The main contributions of the proposed algorithms are summarized as follows.

1. 3D-OPAST and SOAP have lower complexity and comparable or superior convergence as compared to PARAFAC-RLST and PARAFAC-SDT. In terms of complexity, if I and K are the two tensor dimensions other than time, and R is the tensor rank, then 3D-OPAST and SOAP require only $O(IKR)$ flops per iteration (linear complexity with respect to R) while PARAFAC-RLST and PARARAC-SDT require $O(IKR^2)$ (quadratic complexity). For 3D-OPAST, this complexity is achieved by exploiting a special interpretation of the Khatri-Rao product as collinear vectors inside each column of the estimated subspace. For SOAP, we first take advantage of a second-order stochastic gradient algorithm, in replace of the first-order gradient algorithm used in [80], to improve

estimation accuracy. Then, at each step in the algorithm, a column of the estimated subspace is forced to have a Kronecker product structure so that the overall subspace approximately preserves the Khatri-Rao product structure. When possible, a rank-one update is also exploited to achieve linear complexity.

2. A variant of SOAP is proposed for adaptive PARAFAC decomposition with a non-negativity constraint. It is known that imposing a non-negativity constraint on PARAFAC, when applicable, not only improves the physical interpretation [38] but also helps to avoid diverging components [95,96]. To the best of our knowledge, adaptive non-negative PARAFAC has not been addressed in the literature.

3. Both 3DOPAST and SOAP are ready for parallel/decentralized computing implementation, an advantage not considered in [80]. This is especially important when performing large-scale online processing tasks. SOAP allows reduction of algorithm complexity and storage when several parallel computing units (DSP) are available.

6.2 Batch and Adaptive PARAFAC

To make this chapter self-contained, we will recall batch PARAFAC model in this section. In addition, for the ease of comparison, we follow the basic adaptive PARAFAC model and assumptions that were introduced in [80]. This model is slightly different to the PARAFAC model in chapter 3 in terms of unfolded tensor, as described in detail below.

6.2.1 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:

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \tag{6.1}$$

summing R rank-one tensors, where R is the rank of $\mathcal{X}$. The set of vectors $\{\mathbf{a}_r\}$, $\{\mathbf{b}_r\}$, and $\{\mathbf{c}_r\}$ 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, (6.1) is only an approximate tensor. In other words, in a noisy environment we should have

$$\mathcal{X} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r + \mathcal{N}, \tag{6.2}$$

where $\mathcal{N}$ is a noise tensor. Thus, given a data tensor $\mathcal{X}$, PARAFAC tries to achieve R-rank best least squares approximation. Equation (6.1) can also be re-formulated in matrix form as

$$\mathbf{X}^{(1)} = (\mathbf{A} \odot \mathbf{C})\mathbf{B}^T, \tag{6.3}$$

where $\mathbf{X}^{(1)} \in \mathbb{R}^{IK \times J}$ with $\mathbf{X}^{(1)}_{(i-1)K+k,j} = x_{ijk}$. We can write analogous expressions for $\mathbf{X}^{(2)}$ and $\mathbf{X}^{(3)}$ [80].

PARAFAC is generically unique if it satisfies the following condition [56]:

$$2R(R-1) \leqslant I(I-1)K(K-1), \qquad R \leqslant J. \tag{6.4}$$

6.2.2 Adaptive PARAFAC

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

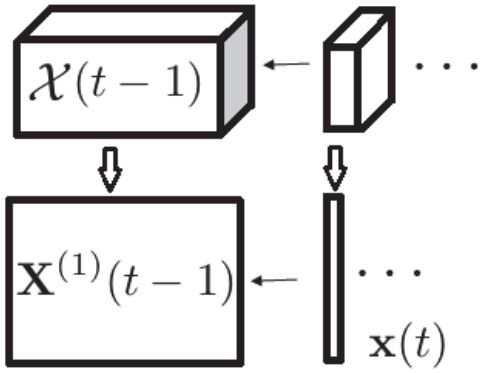


Figure 6.1: Adaptive third-order tensor model and its equivalent matrix form.

$\mathcal{X}(t) \in \mathbb{C}^{I \times J(t) \times K}$.

Under this model, the mode-1 tensor represented in matrix form at time t, $\mathbf{X}^{(1)}(t)$, is given by

$$\mathbf{X}^{(1)}(t) \simeq \mathbf{H}(t)\mathbf{B}^T(t), \tag{6.5}$$

where $\mathbf{H}(t) = \mathbf{A}(t) \odot \mathbf{C}(t) \in \mathbb{C}^{IK \times R}$, $\mathbf{B}(t) \in \mathbb{C}^{J(t) \times R}$. When taking into account two successive times, it can be expressed as a concatenation of the mode-1 tensor of past data at time $t-1$ and a vector of new data at time t, i.e.,

$$\mathbf{X}^{(1)}(t) = (\mathbf{X}^{(1)}(t-1), \mathbf{x}(t)), \tag{6.6}$$

where $\mathbf{x}(t) \in \mathbb{C}^{IK}$ is obtained from vectorizing the new slide of data at time t. Figure 6.1 illustrates this formulation.

The loading matrices $\mathbf{A}$ and $\mathbf{C}$ assumed to follow unknown but slowly time-varying models such that $\mathbf{A}(t) \simeq \mathbf{A}(t-1)$ and $\mathbf{C}(t) \simeq \mathbf{C}(t-1)$, and hence $\mathbf{H}(t) \simeq \mathbf{H}(t-1)$. Accordingly, we have

$$\mathbf{B}^T(t) \simeq (\mathbf{B}^T(t-1), \mathbf{b}^T(t)). \tag{6.7}$$

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

Also, the tensor rank, R, is assumed to be known and constant so that at each time instant, when new data is added to the old tensor, the uniqueness property of the new tensor is fulfilled (6.4). We note that estimating tensor rank is NP-complete problem [37]. Several heuristic methods can be found in [110] and references therein.

6.3 Principle of 3D-OPAST Algorithm

Recall the matrix representation of the PARAFAC model in (6.3) without the superscript (1) for simplicity

$$\mathbf{X} = (\mathbf{A} \odot \mathbf{C})\mathbf{B}^T. \tag{6.8}$$

By using matrix factorization [55], (6.8) can be rewritten as

$$\mathbf{X} = \mathbf{W}\mathbf{E},$$

where

$$\mathbf{W} = (\mathbf{A} \odot \mathbf{C})\mathbf{Q}^{-1},$$
$$\mathbf{E} = \mathbf{Q}\mathbf{B}^T,$$

for some nonsingular $\mathbf{Q}$. The objective is now to find the ambiguity matrix $\mathbf{Q}$ or $\mathbf{Q}^{-1}$ to recover the Khatri-Rao product structure $\mathbf{H} = \mathbf{A} \odot \mathbf{C}$. Matrix $\mathbf{Q}$ can be estimated by solving a simultaneous diagonalization [55], [89] or non-symmetric joint diagonalization as in chapter 4. Hence, $\mathbf{A}$ and $\mathbf{C}$ can be found, based on the following observation of $\mathbf{H}$:

$$\begin{aligned}\mathbf{H} = \mathbf{A} \odot \mathbf{C} &= [\mathbf{a}_1 \otimes \mathbf{c}_1 \cdots \mathbf{a}_R \otimes \mathbf{c}_R] \\ &= [\mathrm{vec}\{\mathbf{c}_1\mathbf{a}_1^T\} \cdots \mathrm{vec}\{\mathbf{c}_R\mathbf{a}_R^T\}].\end{aligned}$$

Since each column of $\mathbf{H}$ has the form of a vectorized rank-1 matrix, we can achieve $\mathbf{c}_i$ and $\mathbf{a}_i$ as left and right singular vectors of matrix $\mathbf{H}_i = \text{unvec}(\mathbf{a}_i \otimes \mathbf{c}_i)$. It is straightforward to obtain $\mathbf{B}$ when either $\mathbf{Q}$ or $\mathbf{Q}^{-1}$ and $\mathbf{E}$ are available.

Because of its high complexity, applying the above procedure is not suitable for the adaptive model. In [80], one of the proposed methods requires sliding-window SVD tracking to estimate both the left and right orthogonal subspaces, $\mathbf{W}(t)$ and $\mathbf{E}(t)$, and then exploits the block common between $\mathbf{B}(t-1)$ and $\mathbf{B}(t)$ to construct the recursive update of $\mathbf{Q}(t)$ and $\mathbf{Q}^{-1}(t)$. We refer the reader to [80] for more details.

In this work, we only track the left orthogonal subspace $\mathbf{W}(t)$ using the OPAST method in [15], and then use orthonormal projection approximation of the two successive subspaces $\mathbf{W}(t-1)$ and $\mathbf{W}(t)$ to find out the recursive update of $\mathbf{H}(t)$ and $\mathbf{Q}(t)$. At each step, we preserve the Khatri-Rao product structure of $\mathbf{H}(t)$ by minimizing a cost function that measures the collinear deviation of sub-vectors inside each column of $\mathbf{H}(t)$. Then, $\mathbf{b}(t)$ is estimated by calculating the pseudo-inverse of $\mathbf{H}(t)$ exploiting its reduced rank update structure. As a consequence, the proposed algorithm has linear computational complexity of order $O(IKR)$ while sustaining good performance.

6.3.1 Algorithm Derivation

6.3.1.1 Overview of OPAST Principle

Consider the cost function

$$\begin{aligned} f(\mathbf{W}) &= E\{ \parallel \mathbf{x}(t) - \mathbf{W}\mathbf{W}\mathbf{x}(t) \parallel^2 \} \\ &= \text{tr}\left\{\mathbf{R}_{xx} - 2\mathbf{W}^H\mathbf{R}_{xx}\mathbf{W} + \mathbf{W}^H\mathbf{R}_{xx}\mathbf{W}\mathbf{W}^H\mathbf{W}\right\}, \end{aligned} \tag{6.9}$$

where $\mathbf{W}$ is an unitary matrix spanning the principal subspace of $\mathbf{R}_{xx} = E\{\mathbf{x}(t)\mathbf{x}^H(t)\}$. Minimizing (6.9) yields the abstract form of the OPAST [15] as follows (using informal

Table 6.1: OPAST algorithm

$$
\begin{aligned}
\mathbf{y}(t) &= \mathbf{W}^H(t-1)\mathbf{x}(t) \\
\mathbf{q}(t) &= \frac{1}{\beta}\mathbf{Z}(t-1)\mathbf{y}(t) \\
\gamma &= 1/(1+\mathbf{y}^H(t)\mathbf{q}(t)) \\
\tau &= \frac{1}{\|\mathbf{q}(t)\|^2}\left(\frac{1}{\sqrt{1+\|\mathbf{q}(t)\|^2(\|\mathbf{x}(t)\|^2-\|\mathbf{y}(t)\|^2)}}-1\right) \\
\mathbf{p}(t) &= \mathbf{W}(t-1)(\tau\mathbf{q}(t)-\gamma(1+\tau\parallel\mathbf{q}(t)\parallel^2)\mathbf{y}(t))+(1+\tau\parallel\mathbf{q}(t)\parallel^2)\gamma\mathbf{x}(t) \\
\mathbf{Z}(t) &= \frac{1}{\beta}\mathbf{Z}(t-1)-\gamma\mathbf{q}(t)\mathbf{q}^H(t) \\
\mathbf{W}(t) &= \mathbf{W}(t-1)+\mathbf{p}(t)\mathbf{q}^H(t)
\end{aligned}
$$

notation)

$$
\begin{aligned}
\mathbf{W}(t) &= \mathbf{R}_{xx}\mathbf{W}(t-1)[\mathbf{W}^H(t-1)\mathbf{R}_{xx}\mathbf{W}(t-1)]^{-1} \\
\mathbf{W}(t) &= \mathbf{W}(t)[\mathbf{W}^H(t)\mathbf{W}(t)]^{-1/2}
\end{aligned} \tag{6.10}
$$

Equation (6.10) is an orthogonalization of $\mathbf{W}(t)$. By using the projection approximation, i.e., $\mathbf{W}(t) \simeq \mathbf{W}(t-1)$, a fast implementation of the algorithm has been derived in [15] and summarized in Table 6.1 where $\mathbf{Z}(0)$ is initialized by identity matrix.

6.3.1.2 3D-OPAST

In this section, we assume that $\mathbf{A}(t-1)$, $\mathbf{B}(t-1)$ and $\mathbf{C}(t-1)$ are available, as well as their related matrices $\mathbf{H}^{\#}(t-1)$ and $\mathbf{H}(t-1)$. We then build on the projection approximation approach to construct the recursive update expressions for $\mathbf{A}(t)$, $\mathbf{B}(t)$ and $\mathbf{C}(t)$. Our algorithm can be summarized in four steps as follows: (i) given $\mathbf{x}(t)$, estimate $\mathbf{W}(t)$ using OPAST; (ii) estimate $\mathbf{H}(t)$ from $\mathbf{W}(t)$ and $\mathbf{Q}(t-1)$ by iteratively minimizing a criterion which measures its deviation from a Khatri-Rao structure; (iii) extract $\mathbf{A}(t)$ and $\mathbf{C}(t)$ from $\mathbf{H}(t)$; and (iv) update the estimate of $\mathbf{H}(t)$ and its pseudo-inverse and calculate $\mathbf{b}^T(t)$. We now consider these steps in detail.

Step 1: Estimate $\mathbf{W}(t)$

To estimate $\mathbf{W}(t)$, at time instant t, we run the OPAST algorithm as described in Table 6.1.

Step 2: Estimate $\mathbf{H}(t)$ from $\mathbf{W}(t)$ and $\mathbf{Q}(t-1)$

At this step, ideally, we want to recover $\mathbf{H}(t)$ which "preserves" the Khatri-Rao product structure. To this end, we first consider the following expression:

$$\begin{aligned}\mathbf{H}(t) &= \mathbf{W}(t)\mathbf{Q}(t) \\ &\simeq [\mathbf{W}(t-1) + \mathbf{p}(t)\mathbf{q}^T(t)][\mathbf{Q}(t-1)(\mathbf{I} + \boldsymbol{\varepsilon}(t)),\end{aligned} \tag{6.11}$$

where we use last equation of OPAST and the updating rule

$$\mathbf{Q}(t) \simeq \mathbf{Q}(t-1)(\mathbf{I} + \boldsymbol{\varepsilon}(t)), \tag{6.12}$$

where $\boldsymbol{\varepsilon}(t)$ is an $R \times R$ unknown matrix. The objective is to find $\boldsymbol{\varepsilon}(t)$ which imposes the Khatri-Rao product structure on $\mathbf{H}(t)$. We note that finding entries of $\boldsymbol{\varepsilon}(t)$ all at once leads to a computational complexity of order $O(IKR^2)$. Instead, to preserve the linear complexity, we choose $\boldsymbol{\varepsilon}(t)$ to be zero except for its $j-$th column vector (with $j = t \mod (R)$), i.e.,

$$\boldsymbol{\varepsilon}(t) = [\mathbf{0} \quad \varepsilon_j \quad \mathbf{0}] = \varepsilon_j \mathbf{e}_j^T, \tag{6.13}$$

where ε_j is j-th column of $\boldsymbol{\varepsilon}(t)$ and $\mathbf{e}_j$ is the unit vector whose j-th entry is one. Substitute (6.13) into (6.11) yields

$$\mathbf{H}(t) \simeq \mathbf{M}(t) + (\mathbf{M}(t)\boldsymbol{\varepsilon}_j)\mathbf{e}_j^T, \tag{6.14}$$

where

$$\begin{aligned}\mathbf{M}(t) &= [\mathbf{W}(t-1) + \mathbf{p}(t)\mathbf{q}^T(t)][\mathbf{Q}(t-1) \\ &= \mathbf{H}(t-1) + \mathbf{p}(t)\tilde{\mathbf{q}}^T(t),\end{aligned} \tag{6.15}$$

with $\tilde{\mathbf{q}}(t) = \mathbf{Q}^T(t-1)\mathbf{q}(t)$. Therefore, only the j-th column of $\mathbf{H}(t)$ is affected by ε_j according to

$$\mathbf{h}_j(t) \simeq \mathbf{m}_j(t) + \mathbf{M}(t)\varepsilon_j \tag{6.16}$$

$$\begin{pmatrix} \mathbf{h}_j^1(t) \\ \vdots \\ \mathbf{h}_j^i(t) \\ \vdots \\ \mathbf{h}_j^I(t) \end{pmatrix} \simeq \begin{pmatrix} \mathbf{m}_j^1(t) + \mathbf{M}_j^1(t)\varepsilon_j \\ \vdots \\ \mathbf{m}_j^i(t) + \mathbf{M}_j^i(t)\varepsilon_j \\ \vdots \\ \mathbf{m}_j^I(t) + \mathbf{M}_j^I(t)\varepsilon_j \end{pmatrix} \tag{6.17}$$

where $\mathbf{h}_j^i(t)$ and $\mathbf{m}_j^i(t), i = 1, \ldots, I$ are $K \times 1$ sub-vectors of $\mathbf{h}_j(t)$ and $\mathbf{m}_j(t)$, respectively, and $\mathbf{M}_j^i(t)$ are sub-matrices of $\mathbf{M}(t)$.

Observe that two successive sub-vectors $\mathbf{h}_j^i(t)$ and $\mathbf{h}_j^{i+1}(t)$ are collinear by following the definition of $\mathbf{H}(t) = \mathbf{A}(t) \odot \mathbf{C}(t)$. Thus, preserving the collinear subvectors inside each column of $\mathbf{H}(t)$ corresponds to imposing the Khatri-Rao structure on $\mathbf{H}(t)$.

Before proceeding further, we define the following cost function which measures the deviation from the collinear condition of two vectors $\mathbf{u} = [u_1, \ldots, u_I]^T$ and $\mathbf{v} = [v_1, \ldots, v_I]^T$:

$$f(\mathbf{u}, \mathbf{v}) = \Big(\sum_{i=1}^{I-1} (u_i v_{i+1} - v_i u_{i+1})^2 \Big) + (u_I v_1 - v_I u_1)^2. \tag{6.18}$$

Let $\tilde{\mathbf{u}} = \mathbf{D}_I \mathbf{u} = [a_I, a_1, \ldots, a_{I-1}]^T$, where $\mathbf{D}_I$ is a downshift permutation matrix [26]. The cost function (6.18) can be written as $f(\mathbf{u}, \mathbf{v}) = \| \tilde{\mathbf{u}} * \mathbf{v} - \tilde{\mathbf{v}} * \mathbf{u} \|^2$, where $*$ refers

to the Hadamard product (elementwise product). Thus, we minimize the following cost function:

$$\min_{\varepsilon_j} \sum_{i=1}^{I-1} \| \tilde{\mathbf{h}}_j^i * \mathbf{h}_j^{i+1} - \tilde{\mathbf{h}}_j^{i+1} * \mathbf{h}_j^i \|^2 + \| \tilde{\mathbf{h}}_j^I * \mathbf{h}_j^1 - \tilde{\mathbf{h}}_j^1 * \mathbf{h}_j^I \|^2 \tag{6.19}$$

To solve (6.19), we used a first order linearization (expansion) in term of ε_j. By this way, ε_j is computed by solving an $R \times R$ linear system. The detail calculation can be found in the Appendix B.1. Once we have ε_j, the j-th columns of $\mathbf{H}(t)$ and $\mathbf{Q}(t)$ are updated using (6.12) and (6.16) respectively.

Step 3: Extract $\mathbf{A}(t)$ and $\mathbf{C}(t)$ from $\mathbf{H}(t)$

At this step, since using batch SVD is not suitable for adaptive tracking, we use a single Bi-SVD iteration [111] to update $\mathbf{a}_i(t)$ and $\mathbf{c}_i(t)$ recursively according to$\mathbf{C}(t)$ from $\mathbf{H}(t)$

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

$$\mathbf{a}_i(t) = \mathbf{H}_i^T(t)\mathbf{c}_i(t-1) \tag{6.20}$$

$$\mathbf{c}_i(t) = \frac{\mathbf{H}_i(t)\mathbf{a}_i(t)}{\| \mathbf{H}_i(t)\mathbf{a}_i(t) \|}. \tag{6.21}$$

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

By combining (6.13), (6.14) and (6.15), $\mathbf{H}(t)$ reveals a rank-2 update structure

$$\mathbf{H}(t) = \mathbf{H}(t-1) + \mathbf{p}(t)\tilde{\mathbf{q}}^T(t) + \mathbf{z}(t)\mathbf{e}_j^T,$$

where $\mathbf{z}(t) = \mathbf{M}(t)\varepsilon_j$. Thus, given the knowlege of $\mathbf{H}^{\#}(t-1)$, we can use the matrix inversion lemma to calculate $\mathbf{H}^{\#}(t)$ with a linear complexity.

A summary of 3DOPAST is given in Table 6.2.

Initialization:

Table 6.2: Summary of 3DOPAST

Inputs:
$\mathbf{A}(t-1)$, $\mathbf{B}(t-1)$, $\mathbf{C}(t-1), \mathbf{H}(t-1)$, $\mathbf{H}^{\#}(t-1)$, $\mathbf{Q}(t-1)$
1. Estimate $\mathbf{W}(t)$
Estimate $\mathbf{W}(t)$ using OPAST
2. Step 2: Estimate $\mathbf{H}(t)$ from $\mathbf{W}(t)$ and $\mathbf{Q}(t-1)$
$\tilde{\mathbf{q}}(t) = \mathbf{Q}^T(t-1)\mathbf{q}(t)$
$\mathbf{M}(t) = \mathbf{H}(t-1) + \mathbf{p}(t)\tilde{\mathbf{q}}^T(t)$ (6.15)
Estimate ε_j by minimizing the cost function (6.19)
$\mathbf{h}_j(t) \simeq \mathbf{m}_j(t) + \mathbf{M}(t)\varepsilon_j$ (6.16)
$\mathbf{Q}(t) \simeq \mathbf{Q}(t-1)(\mathbf{I} + \boldsymbol{\varepsilon}(t))$ (6.12)
3. Extract $\mathbf{A}(t)$ and $\mathbf{C}(t)$
- *Extract* $\mathbf{A}(t)$ *and* $\mathbf{C}(t)$
for $i = 1, \ldots, R$
$\mathbf{H}_i(t) = \text{unvec}(\mathbf{h}_i(t))$
$\mathbf{a}_i(t) = \mathbf{H}_i^T(t)\mathbf{c}_i(t-1)$
$\mathbf{c}_i(t) = \dfrac{\mathbf{H}_i(t)\mathbf{a}_i(t)}{\| \mathbf{H}_i(t)\mathbf{a}_i(t) \|}$
4. Calculate $\mathbf{H}^{\#}(t)$ and update $\mathbf{b}(t)$
Calculate $\mathbf{H}^{\#}(t)$ using fast matrix inversion lemma
$\mathbf{b}^T(t) = \mathbf{H}(t)^{\#}\mathbf{x}(t)$ (6.43)
$\mathbf{B}^T(t) = [\mathbf{B}^T(t-1), \mathbf{b}^T(t)]$ (6.7)
Outputs:
$\mathbf{A}(t)$, $\mathbf{B}(t)$, $\mathbf{C}(t), \mathbf{H}(t)$, $\mathbf{H}^{\#}(t)$, $\mathbf{Q}(t)$

In our simulation, we choose to capture J_0 slices, then run the batch PARAFAC algorithm to obtain initial estimation. J_0 can be chosen to be the smallest number satisfying the uniqueness condition in (6.4).

6.4 Second-order Optimization based Adaptive PARAFAC

Consider the following exponentially weighted least-square cost function:

$$\Phi(t) = \frac{1}{2}\sum_{\tau=1}^{t} \lambda^{t-\tau}\phi(\tau), \tag{6.22}$$

where

$$\phi(\tau) = \| \mathbf{x}(\tau) - \mathbf{H}(t)\mathbf{b}^T(\tau) \|_2^2 \tag{6.23}$$

and $\lambda \in (0, 1]$ is referred to as the forgetting factor. Now, finding the loading matrices of the adaptive PARAFAC model of (6.5) corresponds to minimizing (6.22), that is,

$$\min_{\mathbf{H}(t),\mathbf{B}(t)} \Phi(t), \quad \text{s.t. } \mathbf{H}(t) = \mathbf{A}(t) \odot \mathbf{C}(t). \tag{6.24}$$

This cost function is well-known in adaptive filter theory [107] and can be solved by a recursive least-square method as used in [80]. In this paper, we provide an alternative way to not only improve the performance of the algorithm but also reduce the complexity. For the performance, our idea is to first optimize the exponentially-weighted least- squares cost function by using second-order stochastic gradient, then approximately preserve the Khatri-Rao product of the estimated subspace $\mathbf{H}(t)$ at each step. To achieve linear complexity, we propose to update each column of the subspace at a time instant using a cyclic strategy. Thus, our algorithm is called Second-Order Optimization based Adaptive PARAFAC (SOAP).

Given the estimates of $\mathbf{A}(t-1), \mathbf{B}(t-1)$ and $\mathbf{C}(t-1)$, the objective of SOAP is to construct the recursive update expressions for $\mathbf{A}(t), \mathbf{B}(t)$ and $\mathbf{C}(t)$ using alternating minimization. The algorithm includes four main steps as follows:

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

Step 2: Given $\mathbf{b}(t)$, estimate $\mathbf{H}(t)$

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

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

While the steps are similar to those in [80], the details in these steps contain our various improvements. We now explain these steps in detail. The summary of SOAP is given in Table 6.3.

6.4.1 Estimate $\mathbf{b}^T(t)$

This step is the same as in [80]. Vector $\mathbf{b}^T(t)$ can be obtained as the least-square solution of (6.23), according to

$$\arg\min_{\mathbf{b}^T} \| \mathbf{x}(t) - \mathbf{H}(t-1)\mathbf{b}^T(t) \|_2^2, \tag{6.25}$$

which is

$$\hat{\mathbf{b}}^T(t) = \mathbf{H}^{\#}(t-1)\mathbf{x}(t), \tag{6.26}$$

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

6.4.2 Estimate $\mathbf{H}(t)$

Unlike [80], we use here a Newton-type method to find $\mathbf{H}(t)$. Let $\mathbf{h} = \text{vec}(\mathbf{H})$. Function $\phi(\tau)$ in (6.23) can be rewritten as

$$\phi(\tau) = \| \mathbf{x}(\tau) - (\mathbf{b}(\tau) \otimes \mathbf{I}_{IK})\mathbf{h}(t) \|_2^2, \tag{6.27}$$

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

As mentioned in [112], the direction at the maximum rate of change of a function Φ with respect to $\mathbf{h}$ is given by the derivative of Φ with respect to $\mathbf{h}^*$, i.e., $[\mathfrak{D}_{\mathbf{h}^*}\Phi(\mathbf{h}, \mathbf{h}^*)]^T$. Consequently, we have

$$\begin{aligned}[\mathfrak{D}_{\mathbf{h}^*}\Phi(\mathbf{h}, \mathbf{h}^*)]^T \Big|_{\mathbf{h}=\mathbf{h}(t-1)} = \\ -\sum_{\tau=1}^{t} \lambda^{t-\tau}[(\mathbf{b}^H(\tau) \otimes \mathbf{I}_{IK})\mathbf{x}(\tau) \\ - (\mathbf{b}^H(\tau)\mathbf{b}(\tau) \otimes \mathbf{I}_{IK})\mathbf{h}(t-1)].\end{aligned} \tag{6.28}$$

Thus, its Hessian is computed as

$$\begin{aligned}\mathfrak{H} &= \mathfrak{D}_{\mathbf{h}}([\mathfrak{D}_{\mathbf{h}^*}\Phi(\mathbf{h},\mathbf{h}^*)]^T)\Big|_{\mathbf{h}=\mathbf{h}(t-1)} \\ &= \sum_{\tau=1}^{t}\lambda^{t-\tau}[(\mathbf{b}^H(\tau)\mathbf{b}(\tau))\otimes\mathbf{I}_{IK}] = \mathbf{R}(t)\otimes\mathbf{I}_{IK},\end{aligned} \tag{6.29}$$

where

$$\begin{aligned}\mathbf{R}(t) &= \sum_{\tau=1}^{t}\lambda^{t-\tau}\mathbf{b}^H(\tau)\mathbf{b}(\tau) \\ &= \lambda\mathbf{R}(t-1)+\mathbf{b}^H(t)\mathbf{b}(t)\end{aligned}$$

To achieve linear complexity, we replace (6.28) by instant gradient estimation (i.e., stochastic gradient)

$$\begin{aligned}&[\mathfrak{D}_{\mathbf{h}^*}\phi(\mathbf{h},\mathbf{h}^*,t)]^T\Big|_{\mathbf{h}=\mathbf{h}(t-1)} = \\ &-[(\mathbf{b}^H(t)\otimes\mathbf{I}_{IK})\mathbf{x}(t)- \\ &\quad(\mathbf{b}^H(t)\mathbf{b}(t))\otimes\mathbf{I}_{IK}\mathbf{h}(t-1)].\end{aligned} \tag{6.30}$$

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

$$\mathbf{h}(t) = \mathbf{h}(t-1)+\eta\mathfrak{H}^{-1}[\mathfrak{D}_{\mathbf{h}^*}\phi(\mathbf{h},\mathbf{h}^*,t)]^T, \tag{6.31}$$

where η is a step size. By substituting (6.29) and (6.30) into (6.31), we obtain

$$\begin{aligned}\mathbf{h}(t) &= \mathbf{h}(t-1)+\eta[\mathbf{R}^{-1}(t)(\mathbf{b}^H(t)\otimes\mathbf{I}_{IK})\mathbf{x}(t) \\ &\quad-\mathbf{R}^{-1}(t)(\mathbf{b}^H(t)\mathbf{b}(t)\otimes\mathbf{I}_{IK})\mathbf{h}(t-1)].\end{aligned} \tag{6.32}$$

We can stack (i.e., unvec) (6.32) in matrix form as follows:

$$\begin{aligned}\mathbf{H}(t) =&\mathbf{H}(t-1)+\\ &\eta[\mathbf{x}(t)-\mathbf{H}(t-1)\mathbf{b}^T(t)]\mathbf{b}^*(t)\mathbf{R}^{-1}(t).\end{aligned} \tag{6.33}$$

Here, we can see that calculating and storing the Hessian explicitly as in (6.29) is not necessary. Instead, we only need to calculate the inverse of $\mathbf{R}(t)$. Since $\mathbf{R}(t)$ has a rank-1 update structure, its inverse can be efficiently updated using the inversion lemma as

$$\begin{aligned}\mathbf{R}^{-1}(t) &= [\lambda\mathbf{R}(t-1)+\mathbf{b}^T(t)\mathbf{b}^*(t)]^{-1}\\ &= \lambda^{-1}\mathbf{R}^{-1}(t-1)-\beta^{-1}(t)\mathbf{u}(t)\mathbf{u}^H(t).\end{aligned} \tag{6.34}$$

where

$$\beta(t) = 1+\lambda^{-1}\mathbf{b}^*(t)\mathbf{R}^{-1}(t-1)\mathbf{b}^T(t), \tag{6.35}$$

$$\mathbf{u}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1)\mathbf{b}^T(t), \tag{6.36}$$

Substituting (6.34) into (6.33) yields

$$\mathbf{H}(t) = \mathbf{H}(t-1)+\mathbf{d}(t)\mathbf{u}^H(t), \tag{6.37}$$

where

$$\mathbf{d}(t) = \gamma(t)[\mathbf{x}(t)-\mathbf{H}(t-1)\mathbf{b}^T(t)], \tag{6.38}$$

$$\gamma(t) = \eta(1-\beta^{-1}(t)\mathbf{b}^*(t)\mathbf{u}(t)). \tag{6.39}$$

6.4.3 Extract $\mathbf{A}(t)$ and $\mathbf{C}(t)$ & Update One Column of $\mathbf{H}(t)$

The purpose of this step is to (i) preserve an approximate Khatri-Rao product structure of $\mathbf{H}(t)$ in order to improve the estimation accuracy and ensure the convergence of the algorithm, (ii) provide a reduced rank update structure that allows the calculation of $\mathbf{H}^{\#}(t)$ in the next step with linear complexity, and (iii) extract from $\mathbf{H}(t)$ the loading matrices $\mathbf{A}(t)$ and $\mathbf{C}(t)$. This can be implemented efficiently as step 3 of 3D-OPAST.

Then, having obtained $\mathbf{A}(t)$ and $\mathbf{C}(t)$, $\mathbf{H}(t)$ may be re-updated as done in [80]. However, to achieve linear complexity, in this paper we choose to re-update only one column of $\mathbf{H}(t)$ at each iteration. In particular, at time instant t, we select the column of $\mathbf{H}(t)$ to be updated in a cyclic way; that is, select column j where $j = (t \mod R) + 1$. This column is then updated it as

$$\hat{\mathbf{h}}_j(t) = \mathbf{a}_j(t) \otimes \mathbf{c}_j(t). \tag{6.40}$$

Because of the best rank-1 approximation property of 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

$$\hat{\mathbf{H}}(t) = \mathbf{H}(t) + \mathbf{z}(t)\mathbf{e}_j^T(t), \tag{6.41}$$

where

$$\mathbf{z}(t) = \hat{\mathbf{h}}_j(t) - \mathbf{h}_j(t), \tag{6.42}$$

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 (6.37) into (6.41).

6.4.4 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 Theorem 5 in [113], as given in the Appendix.

Then, we can update $\mathbf{b}(t)$ by

$$\mathbf{b}^T(t) = \hat{\mathbf{H}}^{\#}(t)\mathbf{x}(t), \tag{6.43}$$

and hence obtain $\mathbf{B}(t)$ from (6.7).

6.4.5 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, where J_0 is chosen to satisfy the uniqueness condition of (6.4), 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 $\mathbf{R}^{-1}(0) = (\mathbf{B}^T(0)\mathbf{B}(0))^{-1}$.

6.5 Adaptive Non-negative PARAFAC

In this section, we consider the case where the constraint of non-negativity is imposed, and modify SOAP accordingly. We call this modification of SOAP as non-negative SOAP (NSOAP).

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

Table 6.3: Summary of SOAP

Inputs:

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

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

$$\mathbf{b}^T(t) = \mathbf{H}^{\#}(t-1)\mathbf{x}(t) \tag{6.26}$$

2. Estimate $\mathbf{H}(t)$

$$\beta(t) = 1 + \lambda^{-1}\mathbf{b}^*(t)\mathbf{R}^{-1}(t-1)\mathbf{b}^T(t) \tag{6.35}$$

$$\mathbf{u}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1)\mathbf{b}^T(t) \tag{6.36}$$

$$\mathbf{R}^{-1}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1) - \beta^{-1}(t)\mathbf{u}(t)\mathbf{u}^H(t) \tag{6.34}$$

$$\gamma(t) = \eta(1 - \beta^{-1}(t)\mathbf{b}^*(t)\mathbf{u}(t)) \tag{6.39}$$

$$\mathbf{d}(t) = \gamma(t)[\mathbf{x}(t) - \mathbf{H}(t-1)\mathbf{b}^T(t)] \tag{6.38}$$

$$\mathbf{H}(t) = \mathbf{H}(t-1) + \mathbf{d}(t)\mathbf{u}^T(t) \tag{6.37}$$

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

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

for $i = 1, \ldots, R$

$$\mathbf{H}_i(t) = \text{unvec}(\mathbf{h}_i(t))$$

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

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

- *Update column* j *of* $\mathbf{H}(t)$

$$j = (t \mod R) + 1$$

$$\hat{\mathbf{h}}_j(t) = \mathbf{a}_j(t) \otimes \mathbf{c}_j(t) \tag{6.40}$$

$$\mathbf{z}(t) = \hat{\mathbf{h}}_j(t) - \mathbf{h}_j(t) \tag{6.42}$$

$$\mathbf{H}(:, j)(t) = \hat{\mathbf{h}}_j(t)$$

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

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

$$\mathbf{b}^T(t) = \mathbf{H}(t)^{\#}\mathbf{x}(t) \tag{6.43}$$

$$\mathbf{B}^T(t) = [\mathbf{B}^T(t-1), \mathbf{b}^T(t)] \tag{6.7}$$

Outputs:

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

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 [114] or least-squares non-negative matrix approximation [115].

In practice, for batch processing, the projected Newton-type method requires a combination of restrictions on the Hessian (e.g., diagonal or partly diagonal [116]) and the Armijo-like step rule [114] 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 6.7.

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

$$\mathbf{b}^T(t) \coloneqq [\mathbf{b}^T(t)]^+. \tag{6.44}$$

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

$$\hat{\mathbf{R}}^{-1}(t) = \mathrm{diag}(\mathrm{diag}(\mathbf{R}^{-1}(t))), \tag{6.45}$$

and then calculate

$$\hat{\mathbf{u}}(t) = \hat{\mathbf{R}}^{-1}(t)\mathbf{b}^T(t). \tag{6.46}$$

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

$$\mathbf{H}(t) = \mathbf{H}(t-1) + \mathbf{d}(t)\hat{\mathbf{u}}^T(t). \tag{6.47}$$

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

$$\tilde{\mathbf{H}}(t) = [\mathbf{H}(t)]^{+}. \tag{6.48}$$

As previously discussed in Step 3, $\mathbf{c}_i(t)$ and $\mathbf{a}_i(t)$ are respectively the principal left singular vector and the conjugate of the principal right singular vector of matrix $\mathbf{H}_i(t) = \text{unvec}(\mathbf{a}_i(t) \otimes \mathbf{c}_i(t))$. The updated loading matrices $\mathbf{A}(t)$ and $\mathbf{C}(t)$, obtained by (6.20) and (6.21), 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 (6.44) is used.

A summary of all the steps of the NSOAP algorithm is given in Table 6.4. The initialization of NSOAP is similar to that of SOAP, except that a batch non-negative PARAFAC algorithm is used instead of the standard PARAFAC.

6.6 Discussions

In this section, we provide some important comments on the similarities and differences between our algorithms and those developed in [80]. Discussions on parallel implementations are also given.

First, it is straightforward to realize that in all steps the main cost of 3DOPAST and SOAP come from the matrix-vector product. Thus, they have linear 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. This parameter d can be chosen to balance between the estimation accuracy and the numerical cost.

Third, the main reason that helps SOAP achieve linear complexity, while still

Table 6.4: Summary of NSOAP

Inputs:
$\mathbf{A}(t-1) \geqslant 0$, $\mathbf{B}(t-1) \geqslant 0$, $\mathbf{C}(t-1) \geqslant 0$,
$\mathbf{H}(t-1)$, $\mathbf{H}^{\#}(t-1)$, $\mathbf{R}^{-1}(t-1)$
1. Estimate $\mathbf{b}^T(t)$
Perform Step 1 of SOAP
$\mathbf{b}^T(t) = [\mathbf{b}^T(t)]^+$ (6.44)
2. Estimate $\mathbf{H}(t)$
$\beta(t) = 1 + \mathbf{b}(t)\mathbf{R}^{\#}(t-1)\mathbf{b}(t)^T$ (6.35)
$\mathbf{u}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1)\mathbf{b}(t)^T$ (6.36)
$\mathbf{R}^{-1}(t) = \lambda^{-1}\mathbf{R}^{-1}(t-1) - \beta^{-1}(t)\mathbf{u}(t)\mathbf{u}(t)^T$ (6.34)
$\hat{\mathbf{R}}^{-1}(t) = \text{diag}(\text{diag}(\mathbf{R}^{-1}(t)))$ (6.45)
$\gamma(t) = \eta(1 - \beta^{-1}(t)\mathbf{b}^*(t)\mathbf{u}(t))$ (6.39)
$\mathbf{d}(t) = \gamma(t)(\mathbf{x}(t) - \mathbf{H}(t-1)\mathbf{b}^T(t))$ (6.38)
$\hat{\mathbf{u}}(t) = \hat{\mathbf{R}}^{-1}(t)\mathbf{b}^T(t)$ (6.46)
$\mathbf{H}(t) = \mathbf{H}(t-1) + \mathbf{d}(t)\hat{\mathbf{u}}^T(t)$
$\tilde{\mathbf{H}}(t) = [\mathbf{H}(t)]^+$ (6.48)
3. Same as Step 3 of SOAP but with $\tilde{\mathbf{H}}(t)$
4. Calculate $\mathbf{H}^{\#}(t)$ and update $\mathbf{b}(t)$
Perform Step 4 of SOAP
$\mathbf{b}^T(t) = [\mathbf{b}^T(t)]^+$ (6.44)
$\mathbf{B}^T(t) = [\mathbf{B}^T(t-1), \mathbf{b}^T(t)]$ (6.7)
Outputs:
$\mathbf{A}(t) \geqslant 0$, $\mathbf{B}(t) \geqslant 0$, $\mathbf{C}(t) \geqslant 0$,
$\mathbf{H}(t)$, $\mathbf{H}^{\#}(t)$, $\mathbf{R}^{-1}(t)$

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 (6.20) and (6.21) for $i = 1, \ldots, R$. However, it leads to calculate $\mathbf{H}^{\#}(t)$ without a rank-2 update structure

as in SOAP, by using the Khatri-Rao product structure as follows:

$$\mathbf{H}^{\#}(t) = [\mathbf{A}^T(t)\mathbf{A}(t) * \mathbf{C}^T(t)\mathbf{C}(t)]^{-1}[\mathbf{A}(t) \odot \mathbf{C}(t)]^T. \tag{6.49}$$

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 (6.26) corresponds to

$$\mathbf{b}_i^T(t) = \tilde{\mathbf{h}}_i(t-1)\mathbf{x}(t), \qquad i = 1, \ldots, R, \tag{6.50}$$

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.

6.7 Simulations

In this section, we study the performance of the proposed algorithms using both synthetic and real data, from [105]. We also consider the effect of different system parameters on the performance.

6.7.1 Performance Comparison

First, we use the framework provided by the authors in [80] 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

$$\mathbf{A}(t) = (1 - \varepsilon_A)\mathbf{A}(t-1) + \varepsilon_A \mathbf{N}_A, \tag{6.51}$$

$$\mathbf{C}(t) = (1 - \varepsilon_C)\mathbf{C}(t-1) + \varepsilon_C \mathbf{N}_C, \tag{6.52}$$

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 6.2.2 and are constrained by the assumptions therein. Then, the noisy observation is given by

$$\tilde{\mathbf{x}}(t) = \mathbf{x}(t) + \sigma \mathbf{n}(t), \tag{6.53}$$

where $\mathbf{n}(t)$ is a zero mean, unit-variance noise 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 [80]. A summary of parameters used in our experiments is showed in Table 6.5.

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

Table 6.5: 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	50	50	50	20	1000	10^{-3}	0.8	NA
7	20	50	20	8	10000	0	0.8	NA
8-9	5	50	61	3	1000	0	0.8	0.02

estimates, $\mathbf{A}_{\text{es}}(t)$ and $\mathbf{C}_{\text{es}}(t)$, up to a scale and permutation indeterminacy at each time

$$\text{STD}_{\mathbf{A}}(t) = \| \mathbf{A}(t) - \mathbf{A}_{\text{es}}(t) \|_F, \tag{6.54}$$

$$\text{STD}_{\mathbf{C}}(t) = \| \mathbf{C}(t) - \mathbf{C}_{\text{es}}(t) \|_F . \tag{6.55}$$

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

$$\text{STD}_{\mathbf{B}}(t) = \| \mathbf{x}(t) - \mathbf{x}_{\text{es}}(t) \|_2 . \tag{6.56}$$

To assess the convergence rate of the 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 that 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 [29, 117].

The first experiment is to compare the performance of SOAP with that of PARAFAC-

RLST, PARAFAC-SDT (exponential window), 3DOPAST and batch PARAFAC-ALS (Alternating Least-Squares). Batch PARAFAC here serves as a "lower bound" for adaptive algorithms. As shown in Figure 6.2, SOAP outperforms PARAFAC-RLST, PARAFAC-SDT, 3DOPAST and is closer to batch PARAFAC than the others. In addition, SOAP, 3DOPAST, PARAFAC-RLST, and PARAFAC-SDT approximately have the same convergence rate. Again, we note that the computational complexity of SOAP and 3DOPAST are $O(IKR)$ 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.,

$$\mathbf{A}^{+}(t) = |\mathbf{A}(t)|, \tag{6.57}$$

$$\mathbf{C}^{+}(t) = |\mathbf{C}(t)|, \tag{6.58}$$

$$\mathbf{x}^{+}(t) = |\mathbf{x}(t)|, \tag{6.59}$$

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 [104]. The results are shown in Figure 6.3.

To further study the performance of NSOAP, we apply it to a typical example using a fluorescence dataset [105] 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 the nonnegative constraint are similar to the pure spectra. Here, we use an initialization tensor of size $5 \times 70 \times 61$. Note that the emission wavelength is relatively short, and during the interval $[250, 300]$ one of three components is almost zero. Figure 6.4 shows that

NSOAP can recover the tensor components in this particular example.

6.7.2 Relative Execution Time Comparison

In Section 6.6, we indicate that our algorithms have linear complexity $O(IKR)$. Here, we provide a rough complexity assessment of the algorithms using *CPU execution time* as a measure. We emphasize the relativity of this comparison because the CPU execution time depends on various factors including platform, programming language, implementation. We compare SOAP with PARAFAC-RLST and PARAFAC-SDT [1]. Again, all parameters of PARAFAC-RLST and PARAFAC-SDT are kept default. The algorithms are run in a computer Intel Core i7 2.8 Ghz with 8Gb RAM and Maltab R2015a. Figure 6.6 shows that SOAP is faster than PARAFAC-RLST and PARAFAC-SDT.

6.7.3 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 performance. Figure 6.5 shows that SOAP adapts better to fast variation ($\varepsilon_A = \varepsilon_C = 10^{-2}$) than PARAFAC-RLST, PARAFAC-SDT, and 3DOPAST. 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.

6.7.4 Long Run-time Stability

Performance of various algorithms including the famous recursive least squares (RLS) and least mean square (LMS) can suffer when running for long time. This is referred to as long run-time stability or limited precision effect [118] caused by accumulated

[1] We disregard 3DOPAST in this experiment because the code is not optimized yet.

quantization error in time. We show that, by experiment, SOAP is more stable than PARAFAC-RLST,PARAFAC-SDT and 3DOPAST in this aspect[1] (Figure 6.7), at least in the described context and given parameters. A theoretical analysis to explain why SOAP is more stable is still an open problem and deserves a future work. We note that, in practice, the limited precision effect can be resolved by reinitializing algorithms after typically several thousands of iterations.

6.7.5 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) [119] and multi-subject Functional Magnetic Resonance Imaging (fMRI) analysis [120]. 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

$$\mathrm{SNR(dB)} = \frac{\mathrm{E}\{\| \mathbf{A}(t) \odot \mathbf{C}(t)\mathbf{b}(t) \|_2\}^2}{\sigma^2}. \tag{6.60}$$

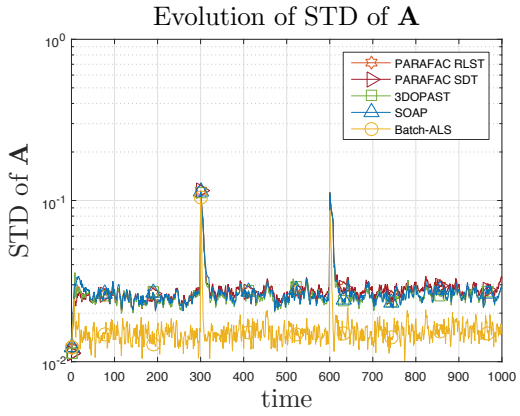
[1] An experiment where we run SOAP 10^6 iterations was conducted to confirm the stability of the proposed algorithm.

The simulation results when applying SOAP and NOSAP are shown in Figures 6.8 and 6.9, corresponding to the first 200 data samples (iterations). As we can see, both algorithms lead to a good restoration of the original components.

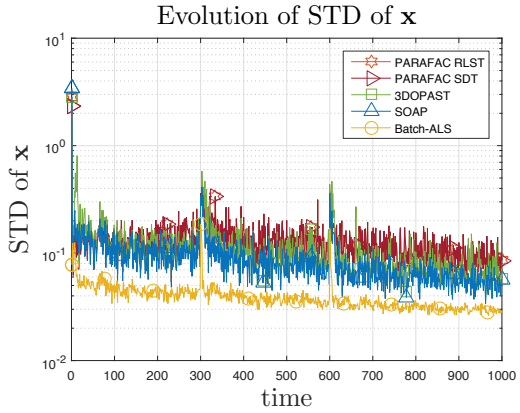
6.8 Conclusions

In this chapter, we have proposed three efficient adaptive PARAFAC decomposition algorithms: 3D-OPAST and SOAP for a standard setup and NSOAP for the non-negative constraint case. To our best knowledge, no adaptive non-negative PARAFAC algorithms have been addressed before. By exploiting the data structure, the proposed algorithms achieve linear computational complexity of $O(IKR)$ per iteration while enjoying a good performance as compared to the state-of-the-art algorithms. These algorithms[1] can be considered as a starting point of real-time PARAFAC-based applications.

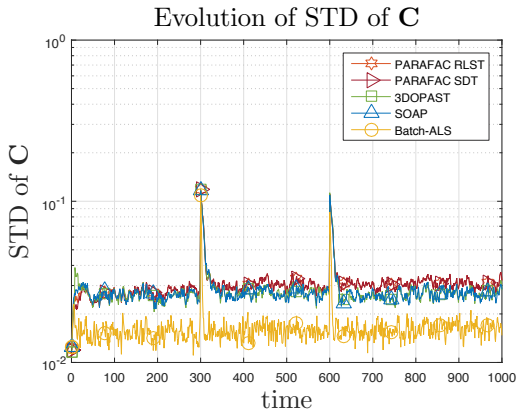
[1]Program codes will be made available on-line after publication of this work.



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

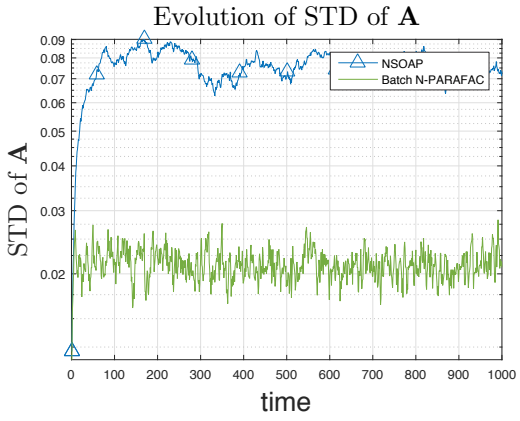


(b) Observation vector $\mathbf{x}(t)$

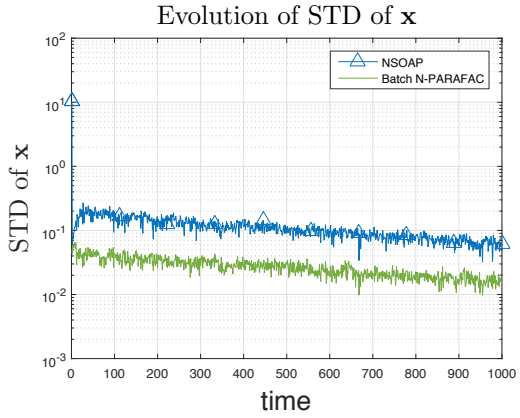


(c) Loading matrix $\mathbf{C}(t)$

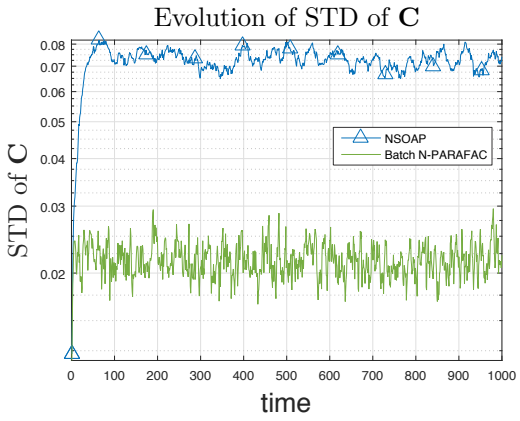
Figure 6.2: Performance and speed convergence rate comparison of five algorithms when loading matrices change relatively fast, $\varepsilon_A = \varepsilon_C = 10^{-3}$.



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

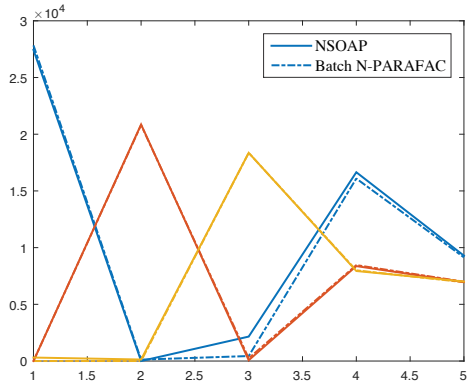


(b) Observation vector $\mathbf{x}(t)$

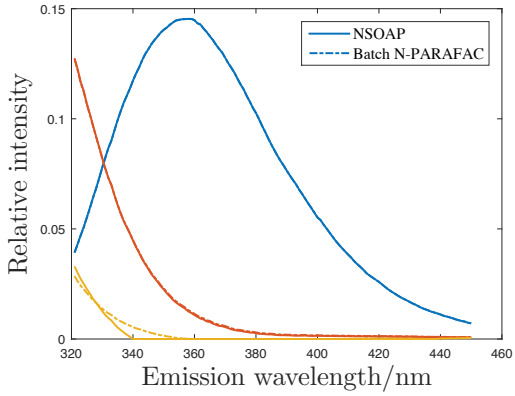


(c) Loading matrix $\mathbf{C}(t)$

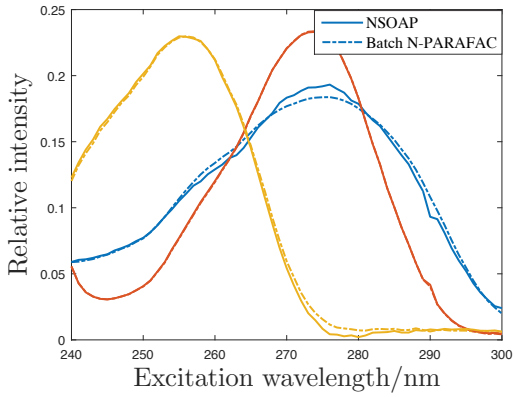
Figure 6.3: Performance comparison of NSOAP with batch non-negative PARAFAC when loading matrices change relatively fast, $\varepsilon_A = \varepsilon_C = 10^{-3}$.



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

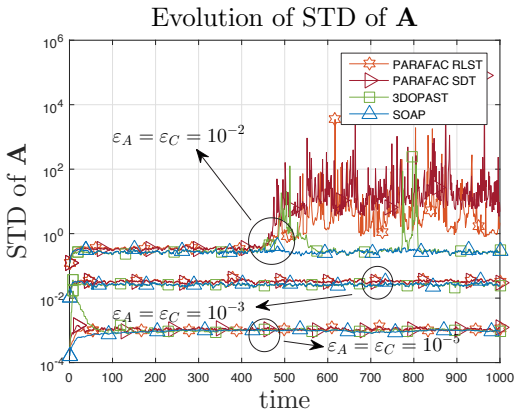


(b) Loading matrix $\mathbf{B}(t)$

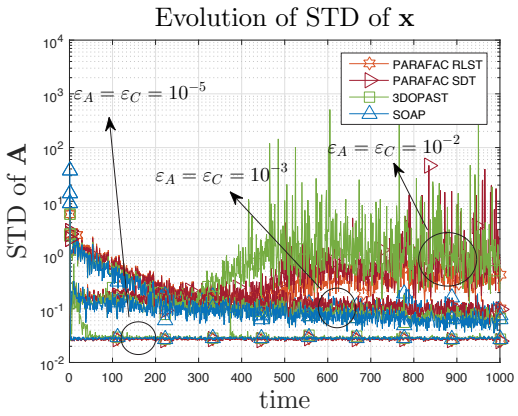


(c) Loading matrix $\mathbf{C}(t)$

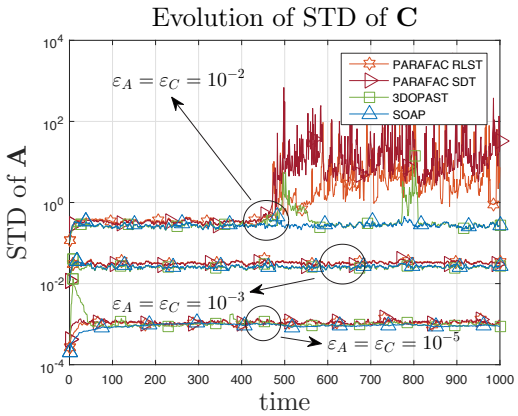
Figure 6.4: Performance comparison of NSOAP with batch non-negative PARAFAC in fluorescence data set. For $\mathbf{B}(t)$, we present only a part of recovered loading matrix; initialization part is disregarded.



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



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



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

Figure 6.5: The effect of the speed of variation on the algorithm performance.

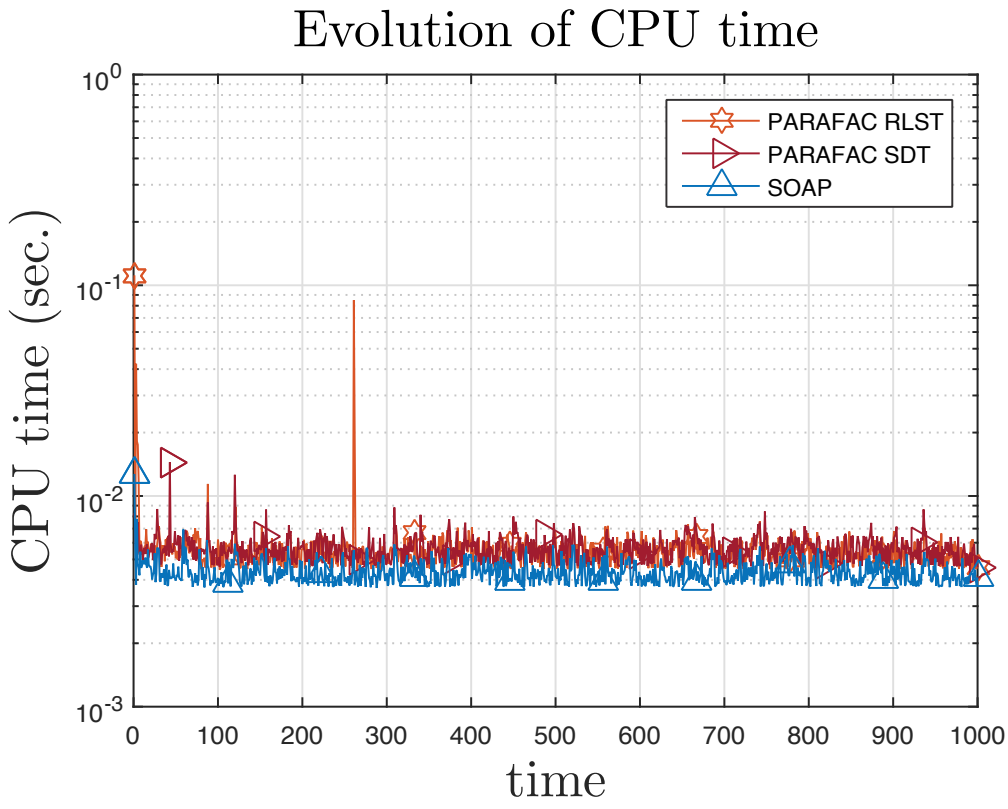
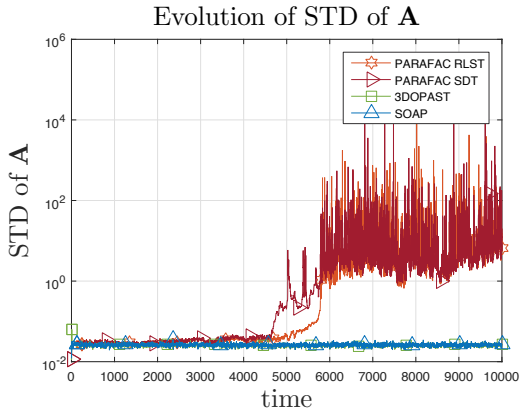
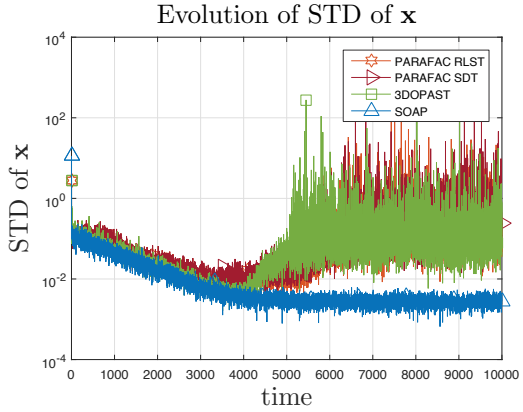


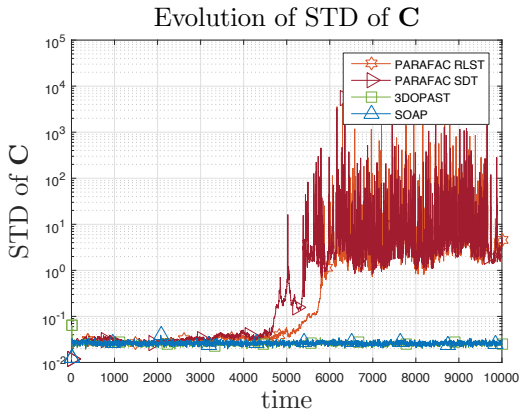
Figure 6.6: CPU time-based comparison when loading matrices change relatively fast, $\varepsilon_A = \varepsilon_C = 10^{-3}$.



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

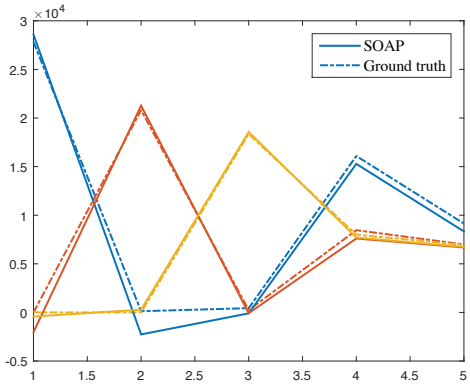


(b) Observation vector $\mathbf{x}(t)$

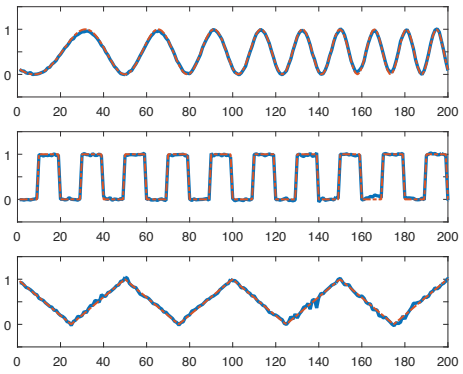


(c) Loading matrix $\mathbf{C}(t)$

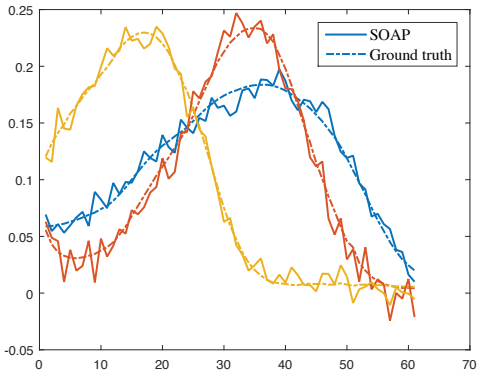
Figure 6.7: Long time run stability of four algorithms when loading matrices change relatively fast, $\varepsilon_A = \varepsilon_C = 10^{-3}$.



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

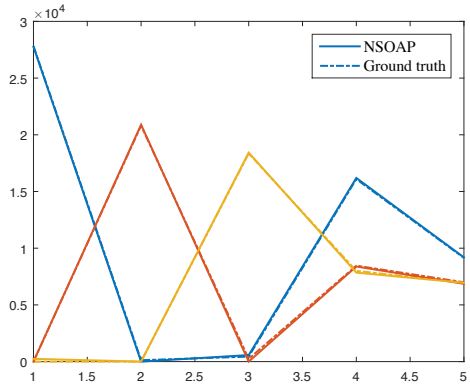


(b) Loading matrix $\mathbf{B}(t)$

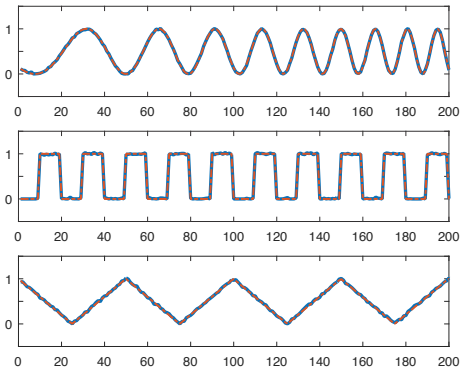


(c) Loading matrix $\mathbf{C}(t)$

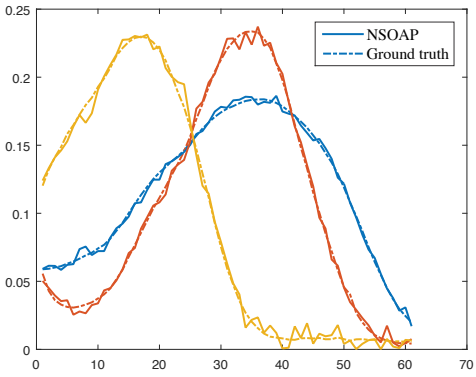
Figure 6.8: Illustration of waveform-preserving character of SOAP through synthetic example with SNR = 15 dB. Solid line represents solution of SOAP while dash-dot line represents ground-truth.



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



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



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

Figure 6.9: Illustration of waveform-preserving character of NSOAP through synthetic example with SNR = 15 dB.

Appendix B

Appendix

B.1 Linearization

In this section, we present in details the procedure which minimizes the cost function defined in Section 6.3.1.2.

$$\min_{\varepsilon_j} \Phi(\varepsilon_j) \tag{B.1}$$

where

$$\Phi(\varepsilon_j) = \Big(\sum_{i=1}^{I-1} \parallel \tilde{\mathbf{h}}_j^i * \mathbf{h}_j^{i+1} - \tilde{\mathbf{h}}_j^{i+1} * \mathbf{h}_j^i) \parallel^2 \Big) + \parallel \tilde{\mathbf{h}}_j^I * \mathbf{h}_j^1 - \tilde{\mathbf{h}}_j^1 * \mathbf{h}_j^I) \parallel^2 \tag{B.2}$$

To simplify the notation , we will drop subscript j from now. Thus, we have

$$\begin{aligned} \Phi(\varepsilon) = \Big(\sum_{i=1}^{I-1} \parallel (\tilde{\mathbf{m}}^i + \tilde{\mathbf{M}}^i\varepsilon) * (\mathbf{m}^{i+1} + \mathbf{M}^{i+1}\varepsilon) - (\tilde{\mathbf{m}}^{i+1} + \tilde{\mathbf{M}}^{i+1}\varepsilon) * (\mathbf{m}^i + \mathbf{M}^i\varepsilon) \parallel^2 \Big) \\ + \parallel (\tilde{\mathbf{m}}^I + \tilde{\mathbf{M}}^I\varepsilon) * (\mathbf{m}^1 + \mathbf{M}^1\varepsilon) - (\tilde{\mathbf{m}}^1 + \tilde{\mathbf{M}}^1\varepsilon) * (\mathbf{m}^I + \mathbf{M}^I\varepsilon) \parallel^2 \end{aligned} \tag{B.3}$$

where

$$\tilde{\mathbf{m}}^i = \mathbf{D}_I * \mathbf{m}^i, \qquad i = 1, \cdots, I \tag{B.4}$$

$$\tilde{\mathbf{M}}^i = \mathbf{D}_I * \mathbf{M}^i, \qquad i = 1, \cdots, I \tag{B.5}$$

By neglecting the second order terms, the cost function $\Phi(\varepsilon)$ can be expressed as follows

$$\Phi(\varepsilon) \simeq \sum_{i=1}^{I-1} \parallel \tilde{\mathbf{m}}^i * (\mathbf{M}^{i+1}\varepsilon) + (\tilde{\mathbf{M}}^i\varepsilon) * \mathbf{m}^{i+1} - \tilde{\mathbf{m}}^{i+1} * (\mathbf{M}^i\varepsilon) - (\tilde{\mathbf{M}}^{i+1}\varepsilon) * \mathbf{m}^i + \tilde{\mathbf{m}}^i * \mathbf{m}^{i+1} - \tilde{\mathbf{m}}^{i+1} * \mathbf{m}^i \parallel^2 \tag{B.6}$$

We note that

$$\tilde{\mathbf{m}}^i * (\mathbf{M}^{i+1}\varepsilon) = (\mathbf{M}^{i+1} * \underbrace{[\tilde{\mathbf{m}}^i, \cdots, \tilde{\mathbf{m}}^i]}_{R \text{ columns}})\varepsilon \qquad = \mathbf{U}^i\varepsilon \tag{B.7}$$

$$(\tilde{\mathbf{M}}^i\varepsilon) * \mathbf{m}^{i+1} = (\tilde{\mathbf{M}}^i * \underbrace{[\mathbf{m}^{i+1}, \cdots, \mathbf{m}^{i+1}]}_{R \text{ columns}})\varepsilon \qquad = \mathbf{V}^i\varepsilon \tag{B.8}$$

$$\tilde{\mathbf{m}}^{i+1} * (\mathbf{M}^i\varepsilon) = (\mathbf{M}^i * \underbrace{[\tilde{\mathbf{m}}^{i+1}, \cdots, \tilde{\mathbf{m}}^{i+1}]}_{R \text{ columns}})\varepsilon \qquad = \mathbf{P}^i\varepsilon \tag{B.9}$$

$$(\tilde{\mathbf{M}}^{i+1}\varepsilon) * \mathbf{m}^i = (\tilde{\mathbf{M}}^{i+1} * \underbrace{[\mathbf{m}^i, \cdots, \mathbf{m}^i]}_{R \text{ columns}})\varepsilon \qquad = \mathbf{Q}^i\varepsilon \tag{B.10}$$

where

$$\mathbf{U}^i = \mathbf{M}^{i+1} * \underbrace{[\tilde{\mathbf{m}}^i, \cdots, \tilde{\mathbf{m}}^i]}_{R \text{ columns}} \tag{B.11}$$

$$\mathbf{V}^i = \tilde{\mathbf{M}}^i * \underbrace{[\mathbf{m}^{i+1}, \cdots, \mathbf{m}^{i+1}]}_{R \text{ columns}} \tag{B.12}$$

$$\mathbf{P}^i = \mathbf{M}^i * \underbrace{[\tilde{\mathbf{m}}^{i+1}, \cdots, \tilde{\mathbf{m}}^{i+1}]}_{R \text{ columns}} \tag{B.13}$$

$$\mathbf{Q}^i = \tilde{\mathbf{M}}^{i+1} * \underbrace{[\mathbf{m}^i, \cdots, \mathbf{m}^i]}_{R \text{ columns}} \tag{B.14}$$

Let

$$\mathbf{s}^i = \tilde{\mathbf{m}}^i * \mathbf{m}^{i+1} \tag{B.15}$$

$$\mathbf{f}^i = \tilde{\mathbf{m}}^{i+1} * \mathbf{m}^i \tag{B.16}$$

We obtain the following form of cost function $\Phi(\varepsilon)$

$$\Phi(\varepsilon) \simeq \sum_{i=1}^{I-1} \parallel \mathbf{Z}^i \varepsilon + \mathbf{z}^i \parallel^2 \tag{B.17}$$

where

$$\mathbf{Z}^i = (\mathbf{U}^i + \mathbf{V}^i) - (\mathbf{P}^i + \mathbf{Q}^i) \tag{B.18}$$

$$\mathbf{z}^i = \mathbf{s}^i - \mathbf{f}^i \tag{B.19}$$

By setting (B.17) to zero and again neglecting the second order terms, we obtain the final solution

$$\varepsilon = \frac{\rho \mathbf{v}}{\parallel \mathbf{v} \parallel^2} \tag{B.20}$$

where

$$\rho = \sum_{i=1}^{I-1} (\mathbf{z}^i)^T (\mathbf{z}^i) \tag{B.21}$$

$$\mathbf{v} = \sum_{i=1}^{I-1} \mathbf{Z}^i \mathbf{z}^i \tag{B.22}$$

B.2 Rank-1 Update Formula

To make our paper self-contained, we present here rank-1 update for the pseudo-inverse as discussed in Step 4 of the proposed algorithm (Section 6.4.4). Because the matrix $\hat{\mathbf{H}}(t)$ has a rank-2 structure, we can apply formula (B.23) twice to obtain its pseudo-inverse $\hat{\mathbf{H}}^{\#}(t)$.

Given matrix $\mathbf{A} \in \mathbb{C}^{I \times J}$, its pseudo-inverse $\mathbf{A}^{\#} \in \mathbb{C}^{I \times J}$ and two vectors, $\mathbf{c} \in \mathbb{C}^{I \times 1}$ $\mathbf{d} \in \mathbb{C}^{J \times 1}$, fast update of $(\mathbf{A} + \mathbf{c}\mathbf{d}^H)^{\#}$, corresponding to Theorem 5 in [113], is given by

$$(\mathbf{A} + \mathbf{c}\mathbf{d}^H)^{\#} = \mathbf{A}^{\#} + \frac{1}{\beta^*}\mathbf{A}^{\#}\mathbf{h}^H\mathbf{u}^H - \frac{\beta^*}{\sigma}\mathbf{p}\mathbf{q}^H, \tag{B.23}$$

where

$$\begin{aligned}
\beta &= 1 + \mathbf{d}^H\mathbf{A}^{\#}\mathbf{c} \\
\mathbf{h} &= \mathbf{d}^H\mathbf{A}^{\#} \\
\mathbf{k} &= \mathbf{A}^{\#}\mathbf{c} \\
\mathbf{u} &= \mathbf{c} - \mathbf{A}\mathbf{k} \\
\mathbf{p} &= -\frac{\| \mathbf{u} \|^2}{\beta^*}\mathbf{A}^{\#}\mathbf{h}^H - \mathbf{k} \\
\mathbf{q}^H &= -\frac{\| \mathbf{h} \|^2}{\beta^*}\mathbf{u}^H - \mathbf{h} \\
\sigma &= \| \mathbf{h} \|^2 \| \mathbf{u} \|^2 + | \beta |^2 .
\end{aligned}$$

We note that this update includes only matrix-vector multiplications and, thus, preserves linear complexity of our algorithm.

Chapter 7

Conclusion

We have presented novel techniques for fast matrix and tensor decomposition with real-life applications. The contributions of this thesis are summarized as follows:

For estimating subspace in parallel, in Chapter 2 we have introduced *subspace fusion* concept inspired by the real-life processing and application on radio-astronomy. We then generalize MNS theory to solve this problem in both batch and adaptive settings. The proposed GMNS consists of the following two main contributions (i) we extend the concept of PCS (properly connected sequence) used in the MNS method in such a way one can extract the minor subspace with a fixed number K of DSPs in a parallel architecture or otherwise to improve the estimation of noise vectors in large dimensional systems, and (ii) we propose new algorithms for the computation of the principal subspace using again K properly chosen subsystems in a parallel scheme. The overall numerical cost is approximately reduced by a factor of K^2 while preserving the estimation accuracy close to optimal. We have also adapted GMNS to several important variants such as principal component analysis, minor and principle subspace tracking, principal eigenvector tracking. As a result, GMNS can be applied to a wider applications such as high resolution parameter estimation, data compression or blind source separation, besides RFI mitigation.

For batch tensor decomposition, first we have did a survey on recent approaches for large-scale tensor problem in Chapter 3. We then have proposed two solutions, non-overlapping and overlapping, for massive-scale PARAFAC in Chapter 4. The proposed algorithms use similar divide-and-conquer strategy as GMNS case. Moreover, since selecting number of components in PARAFAC and Tucker with sparse and non-negative constraints is a difficult problem, we adapted all-at-once optimization approach for those models to resolve it in Chapter 5. Consequently, we can set an overfactoring as tensor rank for PARAFAC model instead of heuristic or time consuming model section approaches. All-at-once optimization for Tucker can also run on top of other algorithms to improve the accuracy of estimated loading factors. Our main point is to show that combining of robust overfactoring algorithms and appropriate constraint (here, nonnegativity and sparsity) can help to eliminate artifacts and obtain the superior result. Besides, we can combine the solutions for massive-scale with the robust-overfactoring algorithms.

For adaptive PARAFAC decomposition, we have set a new limit on computational complexity, *linear complexity* to tensor rank instead of quadratic one in Chapter 6. To this end, we have used the following three main ideas in two different ways. First, since adaptive PARAFAC decomposition can be seen as principal subspace tracking with Khatri-Rao structure, we can generalize results of the well-developed adaptive subspace tracking algorithms. Secondly, reduced-rank structure of estimated subspace can preserve low complexity. Finally, exploiting Khatri-Rao product structure and cyclic updating a column of the estimated subspace can help to improve the accuracy. We have also introduced the adaptive non-negative PARAFAC problem and refined solution of adaptive PARAFAC to tackle it.

We expect that the proposed techniques in this thesis will bring a step forward real-time applications using matrix and tensor decomposition. In the future, we plan

to extend the following aspects from our work.

For fast matrix decomposition:

- more efficient GMNS method for short observational time interval and explore real-time applications of the adaptive tensor decomposition
- combination of GMNS method and random matrix theory
- statistical performance analysis of GMNS

For fast tensor decomposition:

- third-order tensor with two or three dimensions growing with time
- missing data case (both adaptive matrix and tensor completion)

Also, we would like to emphasize that adaptive tensor decomposition is at early stage of development as compared to well-understood adaptive subspace tracking.

References

[1] G. Hellbourg, "RFI spatial processing for modern radio telescopes," *PhD Thesis, Univ. of Orléans*, Jan. 2014. 1, 2.1, 2.7.1

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011. 1

[3] K. Slavakis, G. Giannakis, and G. Mateos, "Modeling and optimization for big data analytics:(statistical) learning tools for our era of data deluge," *Signal Processing Magazine, IEEE*, pp. 18–31, 2014. 1, 6.1

[4] M. Van Haarlem, M. Wise, A. Gunst, G. Heald, J. McKean, J. Hessels, A. De Bruyn, R. Nijboer, J. Swinbank, R. Fallows *et al.*, "LOFAR: The low-frequency array," *Astronomy & Astrophysics*, vol. 556, p. A2, 2013. 1, 2.1

[5] A. Boudjellal, V.-D. Nguyen, K. Abed-Meraim, A. Belouchrani, and P. Ravier, "Time-frequency sparsity prior impact on the localization performance," in *Proc. EUSIPCO*. IEEE, 2016. 1.1.3

[6] T. Minh-Chinh, V.-D. Nguyen, N. Linh-Trung, and K. Abed-Meraim, "Adaptive parafac decomposition for third-order tensor completion," in *Proc. ICCE*, 2016. 1.1.3, 3.5.2

[7] P. Stoica, R. Moses, *Spectral Analysis of Signals*. Prentice Hall, 2005. 2.1

[8] P. Comon and Ch. Jutten, *Handbook of Blind Source Separation: ICA and Applications*. Elsevier, 2010. 2.1

[9] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002. 2.1

[10] S. Haykin, K. J. Ray Liu, *Handbook on Array Processing and Sensor Networks*. Wiley-IEEE Press, 2010. 2.1

[11] E. G. Larsson, F. Tufvesson, O. Edfors, and T. L. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE Commun. Mag.*, 2013. 2.1

[12] "Square kilometre array: Exploring the universe with the world's largest radio telescope," *https://www.skatelescope.org/*. 2.1, 2.7.2

[13] A. Bertrand and M. Moonen, "Distributed adaptive eigenvector estimation of the sensor signal covariance matrix in a fully connected sensor network," in *Proc. ICASSP*, May 2013, pp. 4236–4240. 2.1

[14] B. Yang, "Projection approximation subspace tracking," *IEEE-T-SP*, pp. 95–107, 1995. 2.1, 3.5.1

[15] K. Abed-Meraim, A. Chkeif, and Y. Hua, "Fast orthonormal PAST algorithm," *Signal Processing Letters, IEEE*, pp. 60–62, 2000. 2.1, 2.5.2.1, 3.5.1, 6.1, 6.3, 6.3.1.1, 6.3.1.1

[16] X. G. Doukopoulos and G. V. Moustakides, "Fast and stable subspace tracking," *Signal Processing, IEEE Transactions on*, pp. 1452–1465, 2008. 2.1, 2.5.1, 2.6.3, 6.1

[17] R. O. Schmidt, "Multiple emitter location and signal parameter estimation," *Antennas and Propagation, IEEE Transactions on*, pp. 276–280, 1986. 2.1

[18] R. Roy and T. Kailath, "Esprit-estimation of signal parameters via rotational invariance techniques," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, pp. 984–995, 1989. 2.1

[19] K. Abed-Meraim and Y. Hua, "Blind identification of multi-input multi-output system using minimum noise subspace," *IEEE Tr-SP*, no. 1, pp. 254–258, 1997. 2.1, 2.2, 2.2.2

[20] K. Abed-Meraim, Y. Hua and A. Belouchrani, "Minimum noise subspace: concepts and applications," in *Proc. ICICS*, 1997, pp. 118–121. 2.1

[21] W. Souidene, K. Abed-Meraim, and A. Beghdadi, "A new look to multichannel blind image deconvolution," *Image Processing, IEEE Transactions on*, pp. 1487–1500, 2009. 2.1

[22] M. Thameri, K. Abed-Meraim and A. Belouchrani, "Minor subspace tracking using MNS technique," in *Proc. ICASSP*, 2012, pp. 2433–2436. 2.1

[23] V.-D. Nguyen, K. Abed-Meraim, N. Linh-Trung, and R. Weber, "Generalized MNS method for parallel minor and principal subspace analysis," in *Proc. EUSIPCO*. IEEE, 2014, pp. 2265–2269. 1, 3.4.1.1

[24] M. Thameri, K. Abed-Meraim, and A. Belouchrani, "Minor subspace tracking using mns technique," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, March 2012, pp. 2433–2436. 3

[25] G. Xu and T. Kailath, "Fast subspace decomposition," *Signal Processing, IEEE Transactions on*, pp. 539–551, 1994. 6

[26] G.H. Golub, C. F. Van Loan, *Matrix computations*. JHU Press, 2012. 1, 6.3.1.2

[27] S. Bartelmaos and K. Abed-Meraim, "Principal and minor subspace tracking: Algorithms and stability analysis," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, May 2006, pp. III–III. 2.5.1, 2.6.3

[28] R. Badeau, K. Abed-Meraim, G. Richard, and B. David, "Sliding window orthonormal PAST algorithm," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2003, pp. 261–264. 2.5.2.1

[29] R. Badeau, G. Richard, and B. David, "Fast and stable YAST algorithm for principal and minor subspace tracking," *Signal Processing, IEEE Transactions on*, pp. 3437–3446, 2008. 2.6.3, 1, 6.7.1

[30] R. Weber, G. Hellbourg, C. Dumez-Viou, A. Boonstra, S. Torchinsky, C. Capdessus, and K. Abed-Meraim, "RFI mitigation in radio astronomy: an overview," in *Les Journées Scientifiques d'URSI-France L'électromagnétisme,*, 2013. 2.7, 2.8

[31] G. Hellbourg, R. Weber, K. Abed-Meraim, and A. Boonstra, "RFI spatial processing at Nancay observatory: Approaches and experiments," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 5387–5391. 2.7, 2.8

[32] A.-J. Boonstra, *Radio frequency interference mitigation in radio astronomy*. TU Delft, Delft University of Technology, 2005. 2.7.1

[33] A. J. van der Veen and S. J. Wijnholds, "Signal processing tools for radio astronomy," in *Handbook of Signal Processing Systems*. Springer, 2013, pp. 421–463. 2.7.1

[34] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multi-modal factor analysis," *UCLA Working Papers in Phonetics*, no. 1, p. 84, 1970. 3.1

[35] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition," *Psychometrika*, pp. 283–319, 1970. 3.1

[36] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, pp. 279–311, 1966. 3.1

[37] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, pp. 455–500, 2009. 3.1, 3.1, 3.3.1.1, 4.1, 4.2, 5.1, 6.2.2

[38] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009. 3.1, 5.1, 2

[39] A. Cichocki, "Tensors decompositions: New concepts for brain data analysis?" *Journal of Control Measurement, and System Integration*, pp. 507–517, 2011. 3.1

[40] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *Signal Processing Magazine, IEEE*, pp. 145–163, 2015. 3.1, 3.1, 5.1, 6.1

[41] E. Acar and B. Yener, "Unsupervised multiway data analysis: A literature survey," *IEEE transactions on knowledge and data engineering*, pp. 6–20, 2009. 3.1

[42] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "A survey of multilinear subspace learning for tensor data," *Pattern Recognition*, pp. 1540–1551, 2011. 3.1

[43] M. Mørup, "Applications of tensor (multiway array) factorizations and decompositions in data mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, pp. 24–40, 2011. 3.1

[44] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, pp. 53–78, 2013. 3.1

[45] P. Comon, "Tensors: a brief introduction," *IEEE Signal Processing Magazine*, pp. 44–53, 2014. 3.1

[46] L. De Lathauwer, "Decompositions of a higher-order tensor in block terms-part i: Lemmas for partitioned matrices," *SIAM Journal on Matrix Analysis and Applications*, pp. 1022–1032, 2008. 3.1

[47] ——, "Decompositions of a higher-order tensor in block terms-part ii: Definitions and uniqueness," *SIAM Journal on Matrix Analysis and Applications*, pp. 1033–1066, 2008. 3.1

[48] L. De Lathauwer and D. Nion, "Decompositions of a higher-order tensor in block terms-part iii: Alternating least squares algorithms," *SIAM journal on Matrix Analysis and Applications*, pp. 1067–1083, 2008. 3.1

[49] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, pp. 2295–2317, 2011. 3.1

[50] W. Hackbusch and S. Kühn, "A new scheme for the tensor representation," *Journal of Fourier Analysis and Applications*, pp. 706–722, 2009. 3.1

[51] E. Acar, R. Bro, and A. K. Smilde, "Data fusion in metabolomics using coupled matrix and tensor factorizations," *Proceedings of the IEEE*, pp. 1602–1620, Sept 2015. 3.1

[52] R. C. Farias, J. Cohen, and P. Comon, "Exploring multimodal data fusion through joint decompositions with flexible couplings," *IEEE-T-SP*, pp. 4830 – 4844, 2016. 3.1

[53] H. A. Kiers, "Towards a standardized notation and terminology in multiway analysis," *Journal of chemometrics*, pp. 105–122, 2000. 3.3.1.1

[54] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM journal on Matrix Analysis and Applications*, pp. 1253–1278, 2000. 3.3.1.1, 3.3.3

[55] L. De Lathauwer, "A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization," *SIAM journal on Matrix Analysis and Applications*, pp. 642–666, 2006. 3.3.2, 4.1, 4.2, 4.3, 4.3, 6.3

[56] J. B. Kruskal, "Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics," *Linear algebra and its applications*, pp. 95–138, 1977. 3.3.2, 6.2.1

[57] A.-H. Phan and A. Cichocki, "PARAFAC algorithms for large-scale problems," *Neurocomputing*, pp. 1970–1984, 2011. 3.4.1.1

[58] V.-D. Nguyen, K. Abed-Meraim, and N. Linh-Trung, "Parallelizable PARAFAC decomposition of 3-way tensors," in *Proc. ICASSP*. IEEE, 2015, pp. 5505–5509. 3.4.1.1

[59] V.-D. Nguyen, K. Abed-Meraim, N. Linh-Trung, and R. Weber, "Array processing using generalized minimum noise subspace," *HAL-https://hal.inria.fr/hal-01295030*, 2016. 3.4.1.1

[60] A. L. de Almeida and A. Y. Kibangou, "Distributed computation of tensor decompositions in collaborative networks," in *Proc. CAMSAP*. IEEE, 2013, pp. 232–235. 3.4.1.1

[61] A. De Almeida and A. Y. Kibangou, "Distributed large-scale tensor decomposition," in *Proc. ICASSP*. IEEE, 2014, pp. 26–30. 3.4.1.1

[62] W. Austin, G. Ballard, and T. G. Kolda, "Parallel tensor compression for large-scale scientific data," in *Proc. IPDPS*, May 2016. 3.4.1.2

[63] L. De Lathauwer, B. De Moor, and J. Vandewalle, "On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors," *SIAM Journal on Matrix Analysis and Applications*, pp. 1324–1342, 2000. 3.4.1.2

[64] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition," *SIAM Journal on Scientific Computing*, pp. A1027–A1052, 2012. 3.4.1.2

[65] R. Bro and C. A. Andersson, "Improving the speed of multiway algorithms: Part ii: Compression," *Chemometrics and intelligent laboratory systems*, pp. 105–113, 1998. 3.4.2

[66] P. Drineas and M. W. Mahoney, "Randnla: Randomized numerical linear algebra," *Communications of the ACM*, pp. Pages 80–90, 2016. 3.4.2

[67] ——, "A randomized algorithm for a tensor-based generalization of the singular value decomposition," *Linear algebra and its applications*, pp. 553–571, 2007. 3.4.2

[68] C. E. Tsourakakis, "MACH: Fast randomized tensor decompositions." SIAM, 2010. 3.4.2

[69] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 521–536. 3.4.2

[70] N. D. Sidiropoulos and A. Kyrillidis, "Multi-way compressed sensing for sparse low-rank tensors," *IEEE Signal Processing Letters*, pp. 757–760, 2012. 3.4.2

[71] N. Sidiropoulos, E. Papalexakis, and C. Faloutsos, "Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition," *Signal Processing Magazine, IEEE*, pp. 57–70, 2014. 3.4.2

[72] M. Rajih, P. Comon, and R. A. Harshman, "Enhanced line search: A novel method to accelerate parafac," *SIAM journal on matrix analysis and applications*, pp. 1128–1147, 2008. 3.4.3, 4.2, 1

[73] A.-H. Phan, P. Tichavskỳ, and A. Cichocki, "Fast alternating ls algorithms for high order candecomp/parafac tensor factorizations," *IEEE-T-SP*, pp. 4834–4846, 2013. 3.4.3

[74] N. Ravindran, N. D. Sidiropoulos, S. Smith, and G. Karypis, "Memory-efficient parallel computation of tensor and matrix products for big tensor decomposition," in *Proc. Asilomar*. IEEE, 2014, pp. 581–585. 3.4.3

[75] E. Acar, D. M. Dunlavy, and T. G. Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *Journal of Chemometrics*, pp. 67–86, February 2011. 3.4.3

[76] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, pp. 41–56, 2011. 3.4.3, 5.1, 5.2, 5.5, 5.5

[77] V.-D. Nguyen, K. Abed-Meraim, and N. Linh-Trung, "New robust algorithms for sparse non-negative three-way tensor decompositions," in *Proc. EUSIPCO*, 2016. 3.4.3

[78] A.-H. Phan, P. Tichavsky, and A. Cichocki, "Low complexity damped gauss–newton algorithms for candecomp/parafac," *SIAM Journal on Matrix Analysis and Applications*, pp. 126–147, 2013. 3.4.3

[79] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers," *IEEE-T-SP*, pp. 5450–5463, 2015. 3.4.3

[80] D. Nion and N. D. Sidiropoulos, "Adaptive algorithms to track the PARAFAC decomposition of a third-order tensor," *IEEE-T-SP*, pp. 2299–2310, 2009. 3.5.1, 6.1, 1, 3, 6.2, 6.2.1, 6.3, 6.4, 6.4.1, 6.4.2, 6.4.3, 6.6, 6.7.1, 6.7.1

[81] V.-D. Nguyen, K. Abed-Meraim, and N. Linh-Trung, "Fast adaptive PARAFAC decomposition algorithm with linear complexity," in *Proc. ICASSP*, 2016. 3.5.1

[82] ——, "On adaptive PARAFAC decomposition of three-way tensors," *HAL-https://hal.inria.fr/hal-01295020*, 2016. 3.5.1

[83] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, and C. Faloutsos, "Incremental tensor analysis: Theory and applications," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, p. 11, 2008. 3.5.1

[84] W. Hu, X. Li, X. Zhang, X. Shi, S. Maybank, and Z. Zhang, "Incremental tensor subspace learning and its applications to foreground segmentation and tracking," *International Journal of Computer Vision*, pp. 303–327, 2011. 3.5.1

[85] Y. Cheng, F. Roemer, O. Khatib, and M. Haardt, "Tensor subspace tracking via Kronecker structured projections (TeTraKron) for time-varying multidimensional harmonic retrieval," *EURASIP Journal on Advances in Signal Processing*, pp. 1–14, 2014. 3.5.1

[86] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International Journal of Computer Vision*, pp. 125–141, 2008. 3.5.1

[87] M. Mardani, G. Mateos, and G. B. Giannakis, "Subspace learning and imputation for streaming big data matrices and tensors," *IEEE-T-SP*, pp. 2663–2677, 2015. 3.5.2, 6.1

[88] H. Kasai, "Online low-rank tensor subspace tracking from incomplete data by CP decomposition using recursive least squares," in *Proc. ICASSP*, March 2016, pp. 2519–2523. 3.5.2

[89] F. Roemer and M. Haardt, "A closed-form solution for parallel factor (parafac) analysis," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on.* IEEE, 2008, pp. 2365–2368. 4.1, 1, 6.3

[90] X. Liu and N. D. Sidiropoulos, "Cramer-rao lower bounds for low-rank decomposition of multidimensional arrays," *Signal Processing, IEEE Transactions on*, pp. 2074–2086, 2001. 4.1

[91] A.-J. Van Der Veen, "Joint diagonalization via subspace fitting techniques," vol. 5, pp. 2773–2776, 2001. 4.2

[92] M. Congedo, R. Phlypo, and D.-T. Pham, "Approximate joint singular value decomposition of an asymmetric rectangular matrix set," *Signal Processing, IEEE Transactions on*, vol. 59, no. 1, pp. 415–424, 2011. 4.2

[93] K. Abed-Meraim and Y. Hua, "A least-squares approach to joint schur decomposition," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 4. IEEE, 1998, pp. 2541–2544. 4.2

[94] N. K. M. Faber, R. Bro, and P. K. Hopke, "Recent developments in candecomp/parafac algorithms: a critical review," *Chemometrics and Intelligent Laboratory Systems*, vol. 65, no. 1, pp. 119–137, 2003. 5.1

[95] L.-H. Lim and P. Comon, "Nonnegative approximations of nonnegative tensors," *Journal of chemometrics*, vol. 23, no. 7-8, pp. 432–441, 2009. 5.1, 2

[96] A. Stegeman, "Finding the limit of diverging components in three-way Candecomp/Parafaca demonstration of its practical merits," *Computational Statistics & Data Analysis*, pp. 203–216, 2014. 5.1, 2

[97] M. Mørup, L. K. Hansen, and S. M. Arnfred, "Algorithms for sparse nonnegative Tucker decompositions," *Neural computation*, pp. 2112–2131, 2008. 5.1

[98] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, 2011. 5.1

[99] E. Acar, T. G. Kolda, and D. M. Dunlavy, "All-at-once optimization for coupled matrix and tensor factorizations," *arXiv preprint arXiv:1105.3422*, 2011. 5.1

[100] C. J. Lin, "Projected gradient methods for nonnegative matrix factorization," *Neural computation*, vol. 19, no. 10, pp. 2756–2779, 2007. 5.2, 1

[101] P. H. Calamai and J. J. Moré, "Projected gradient methods for linearly constrained problems," *Mathematical programming*, vol. 39, no. 1, pp. 93–116, 1987. 5.2

[102] D. P. Bertsekas, "Nonlinear programming," *Athena scientific*, 1999. 5.2

[103] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.6," Available online, February 2015. [Online]. Available: http://www.sandia.gov/~tgkolda/TensorToolbox/ 5.5

[104] C. A. Andersson and R. Bro, "The n-way toolbox for MATLAB," *Chemometrics and Intelligent Laboratory Systems*, pp. 1–4, 2000. 5.5, 6.7.1

[105] R. Bro, "PARAFAC. tutorial and applications," *Chemometrics and intelligent laboratory systems*, pp. 149–171, 1997. 5.5, 5.5, 6.7, 6.7.1

[106] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM Journal on imaging sciences*, vol. 6, no. 3, pp. 1758–1789, 2013. 5.5

[107] S. S. Haykin, *Adaptive filter theory*. Pearson Education India, 2007. 6.1, 6.4

[108] P. Comon and G. H. Golub, "Tracking a few extreme singular values and vectors in signal processing," *Proceedings of the IEEE*, vol. 78, no. 8, pp. 1327–1343, 1990. 6.1

[109] Y. Hua, Y. Xiang, T. Chen, K. Abed-Meraim, and Y. Miao, "A new look at the power method for fast subspace tracking," *Digital Signal Processing*, vol. 9, no. 4, pp. 297–314, 1999. 6.1

[110] K. Liu, J. P. C. da Costa, H. C. So, L. Huang, and J. Ye, "Detection of number of components in candecomp/parafac models via minimum description length," *Digital Signal Processing*, vol. 51, pp. 110–123, 2016. 6.2.2

[111] P. Strobach, "Bi-iteration SVD subspace tracking algorithms," *Signal Processing, IEEE Transactions on*, pp. 1222–1240, 1997. 6.3.1.2

[112] A. Hjorungnes and D. Gesbert, "Complex-valued matrix differentiation: Techniques and key results," *Signal Processing, IEEE Transactions on*, pp. 2740–2746, June 2007. 6.4.2

[113] C. D. Meyer, Jr, "Generalized inversion of modified matrices," *SIAM Journal on Applied Mathematics*, pp. 315–323, 1973. 6.4.4, B.2

[114] D. P. Bertsekas, "Projected newton methods for optimization problems with simple constraints," *SIAM Journal on control and Optimization*, pp. 221–246, 1982. 6.5

[115] D. Kim, S. Sra, and I. S. Dhillon, "Fast Newton-type methods for the least squares nonnegative matrix approximation problem." in *SDM*. SIAM, 2007, pp. 343–354. 6.5

[116] M. Schmidt, D. Kim, and S. Sra, "Projected Newton-type methods in machine learning," *Optimization for Machine Learning*, p. 305, 2012. 6.5

[117] P. Strobach, "Fast recursive subspace adaptive ESPRIT algorithms," *Signal Processing, IEEE Transactions on*, pp. 2413–2430, 1998. 6.7.1

[118] J. M. Cioffi, "Limited-precision effects in adaptive filtering," *Circuits and Systems, IEEE Transactions on*, pp. 821–833, 1987. 6.7.4

[119] N. D. Sidiropoulos, G. B. Giannakis, and R. Bro, "Blind PARAFAC receivers for DS-CDMA systems," *Signal Processing, IEEE Transactions on*, pp. 810–823, 2000. 6.7.5

[120] C. F. Beckmann and S. M. Smith, "Tensorial extensions of independent component analysis for multisubject FMRI analysis," *Neuroimage*, pp. 294–311, 2005. 6.7.5

Viet-Dung NGUYEN

Contribution aux décompositions rapides des matrices et tenseurs

Résumé :

De nos jours, les grandes masses de données se retrouvent dans de nombreux domaines relatifs aux applications multimédia, sociologiques, biomédicales, radio astronomiques, etc. On parle alors du phénomène 'Big Data' qui nécessite le développement d'outils appropriés pour la manipulation et l'analyse appropriée de telles masses de données. Ce travail de thèse est dédié au développement de méthodes efficaces pour la décomposition rapide et adaptative de tenseurs ou matrices de grandes tailles et ce pour l'analyse de données multidimensionnelles.
Nous proposons en premier une méthode d'estimation de sous espaces qui s'appuie sur la technique dite 'divide and conquer' permettant une estimation distribuée ou parallèle des sous-espaces désirés. Après avoir démontré l'efficacité numérique de cette solution, nous introduisons différentes variantes de celle-ci pour la poursuite adaptative ou bloc des sous espaces principaux ou mineurs ainsi que des vecteurs propres de la matrice de covariance des données. Une application à la suppression d'interférences radiofréquences en radioastronomie a été traitée.
La seconde partie du travail a été consacrée aux décompositions rapides de type PARAFAC ou Tucker de tenseurs multidimensionnels. Nous commençons par généraliser l'approche 'divide and conquer' précédente au contexte tensoriel et ce en vue de la décomposition PARAFAC parallélisable des tenseurs. Ensuite nous adaptons une technique d'optimisation de type 'all-at-once' pour la décomposition robuste (à la méconnaissance des ordres) de tenseurs parcimonieux et non négatifs. Finalement, nous considérons le cas de flux de données continu et proposons deux algorithmes adaptatifs pour la décomposition rapide (à complexité linéaire) de tenseurs en dimension 3. Malgré leurs faibles complexités, ces algorithmes ont des performances similaires (voire parfois supérieures) à celles des méthodes existantes de la littérature.
Au final, ce travail aboutit à un ensemble d'outils algorithmiques et algébriques efficaces pour la manipulation et l'analyse de données multidimensionnelles de grandes tailles.

Mots clés : décompositions rapides des matrices et tenseurs, PARAFAC, Tucker, Big Data, suivi sous-espace adaptatif, contrainte clairsemée et non négative.

Viet-Dung NGUYEN

Contributions to Fast Matrix and Tensor Decompositions

Abstract :

Large volumes of data are being generated at any given time, especially from transactional databases, multimedia content, social media, and applications of sensor networks. When the size of datasets is beyond the ability of typical database software tools to capture, store, manage, and analyze, we face the phenomenon of big data for which new and smarter data analytic tools are required. Big data provides opportunities for new form of data analytics, resulting in substantial productivity. In this thesis, we will explore fast matrix and tensor decompositions as computational tools to process and analyze multi-dimensional massive-data.

We first aim to study fast subspace estimation, a specific technique used in matrix decomposition. Traditional subspace estimation yields high performance but suffers from processing large-scale data. We thus propose distributed/parallel subspace estimation following a divide-and-conquer approach in both batch and adaptive settings. Based on this technique, we further consider its important variants such as principal component analysis, minor and principal subspace tracking and principal eigenvector tracking. We demonstrate the potential of our proposed algorithms by solving the challenging radio frequency interference (RFI) mitigation problem in radio astronomy.

In the second part, we concentrate on fast tensor decomposition, a natural extension of the matrix one. We generalize the results for the matrix case to make PARAFAC tensor decomposition parallelizable in batch setting. Then we adapt all-at-once optimization approach to consider sparse non-negative PARAFAC and Tucker decomposition with unknown tensor rank. Finally, we propose two PARAFAC decomposition algorithms for a class of third-order tensors that have one dimension growing linearly with time. The proposed algorithms have linear complexity, good convergence rate and good estimation accuracy. The results in a standard setting show that the performance of our proposed algorithms is comparable or even superior to the state-of-the-art algorithms. We also introduce an adaptive nonnegative PARAFAC problem and refine the solution of adaptive PARAFAC to tackle it.

The main contributions of this thesis, as new tools to allow fast handling large-scale multidimensional data, thus bring a step forward real-time applications..

Keywords : Fast matrix and tensor decompositions, PARAFAC, Tucker, Big Data, adaptive subspace tracking, sparse and non-negative constraint.

Laboratoire Pluridisciplinaire de Recherche en Ingénierie des Systèmes, Mécanique et Energétique (PRISME)
12 rue de Blois, 45067 Orléans, France