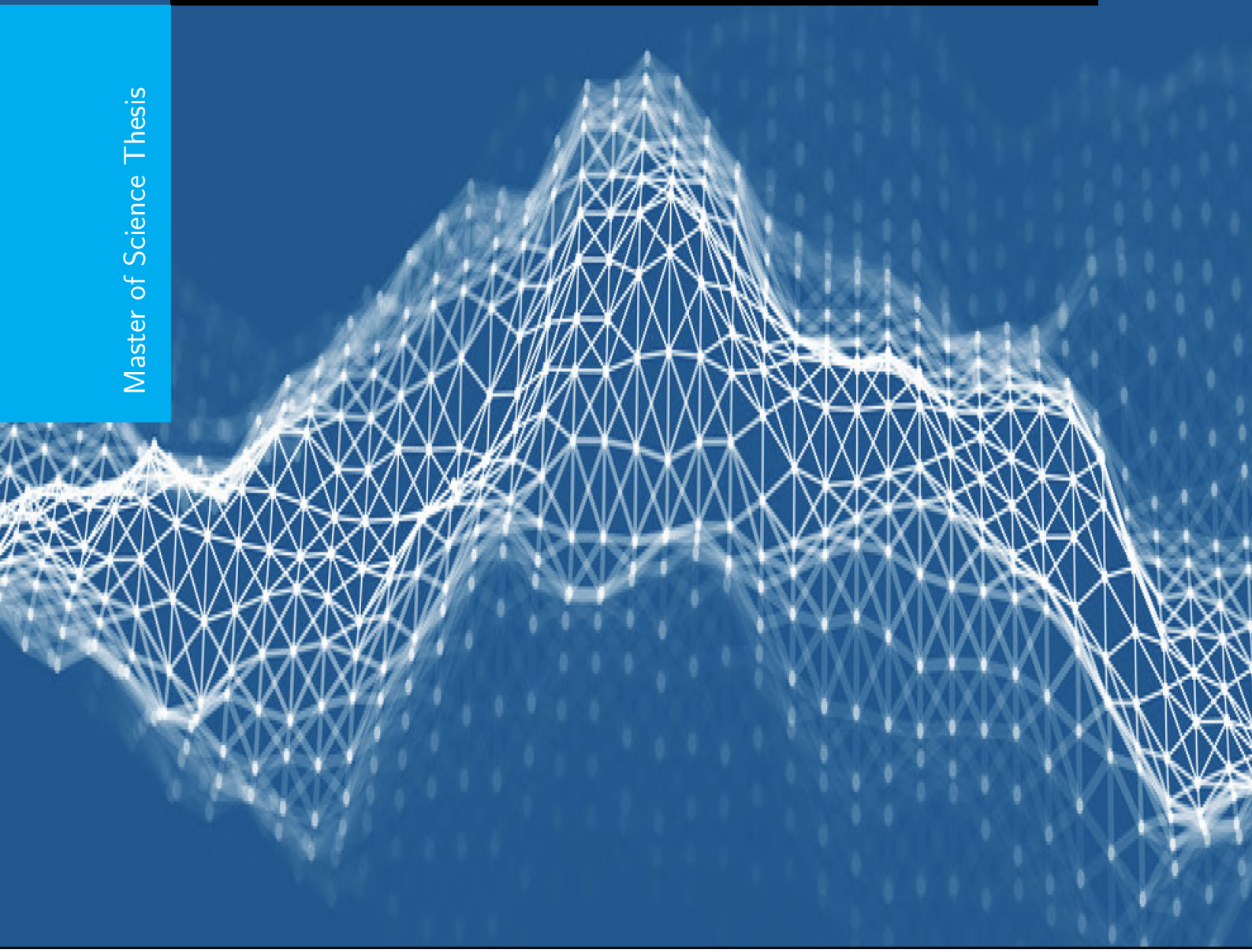
Symmetric Canonical Polyadic Decomposition And Gauss-Newton Optimizer For Nonlinear Volterra System Identification

Zhehan Li

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Symmetric Canonical Polyadic Decomposition And Gauss-Newton Optimizer For Nonlinear Volterra System Identification

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Zhehan Li

July 18, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Most real-life systems are nonlinear in nature so that empirical modeling of a nonlinear system from the measured input–output data becomes an important tool for accurately reflecting the system under research. High order discrete-time nonlinear Volterra systems have been widely used in various application areas. The main drawback of these systems is their parametric complexity implying the need to estimate a huge number of parameters. Recent research has applied various tensor decomposition techniques on the Volterra kernels to induce a substantial parametric complexity reduction. One of these tensor decomposition techniques is the parallel factor (PARAFAC) decomposition, whose obtained Volterra system model is called the Volterra-PARAFAC model. This thesis applies the Gauss-Newton optimizer to estimate the parameter values of the Volterra-PARAFAC model by minimizing a nonlinear least square cost (NLS) function given the input and output measurements.

First, we model the multiple-input single-output (MISO) Volterra system as a linear system with a canonical polyadic decomposition constrained solution (LS-CPD), whose parameter values are estimated by the numerical optimization method. Next, we perform several internal validations on this innovative framework by using it to identify an artificial MISO Volterra system, with the aim to investigate the influence of its hyper-parameters on the identification performances. Meanwhile, we compare this developed framework with the existing frameworks by performing an external validation on identifying the same artificial MISO Volterra system under a storage complexity budget. In addition, we apply the developed framework to model the evoked cortical response (ECR) data set, which has been proven to be highly nonlinear. Finally, we test the plausibility of the modelled system on the ECR data set by examining its validation output residual signals. The advantages and disadvantages of the developed framework are found based on the validation and application experimental results.

Table of Contents

List of Figures

List of Tables

Acknowledgements

"Challenge the future" is the motto of Delft University of Technology, which has been motivating me to take on challenges of proactive self-improvement throughout my master studying. My decision to research in the field of nonlinear system identification for my master thesis comes from the grade of 6 in the course "Filtering And Identification". I discovered my weaknesses in both the theoretical and practical application of system identification throughout the integration project and other courses. Afterwards, I spent nine months working on my weak spots and preparing this report. This extremely educative experience highlights the intricate complexity of performing a thorough in-depth exploration of cutting-edge research. This report could not have been completed without the contribution of many people.

First of all, I would like to express my sincere gratitude to my daily supervisor dr. ir. Kim Batselier. Thanks for all the critical discussions and all the patience you had. Your perfectionism was challenging to pursue yet easy to admire. Your thorough and compelling feedback on every experiment report draft was often challenging but never misplaced. In addition, I would love to thank the research team who developed the Tensorlab. All the tensor computations in my report are based on the Matlab package from Tensorlab. I would also love to thank dr. ir. Alfred Schouten for all the advice about modeling the wrist dataset and indispensable feedback on my approach. Furthermore, I would like to thank the other two committee members of my thesis, dr. Riccardo Ferrari and MSc. Eva Memmel for interest in my research. Lastly, I would love to thank Heleen Sakkee-Zaal for the administrative and technical support for my thesis defense.

I had the chance to combine my master studying with some extra-master experience. Firstly, I would love to thank Prof. dr. Simon Watson and Lily Li for giving me the opportunity to assist the organization and participant in the TORQUE2022. Secondly, a big thank you to Marie Louise Verhagen and dr. Jens Kober for the chance to work as the technical assistant for the CCF2021 and AIM2021. In addition, I would love to thank dr. ir. Özge Okur and dr. ir. Iulia Left for having me as the teaching assistant for the courses, EPA1333 and TB133D. Your advice has improved my Python programming and communication skills, and stress-management ability. Lastly, I must also thank my internship supervisors Caspar Walhout and Paul Stefen Mooij from Husky Intelligent Fridges for your guidance and involvement in my internship project. Without your help, I could not have gone through all the difficult phases and stages of my internship research. I would also love to thank Prof. dr. Hans

Hallendorn and Birgit de Bruin for listening to my presentations and giving valuable feedback.

I was fortunate to have many colleagues and friends from whom to learn. Firstly, I would love to thank Vivek Varma (teammate for courses SC42056, SC42155, SC42150, SC42145, SC42025, and SC52035), Hassan Sewailem (teammate for courses SC42050 and SC42125), Yabin Wang (teammate for course AP3132) and many others. Big Thanks for your cooperation with me and many enjoyable sleepless nights we have spent together on the projects. Furthermore, I would love to thank my friends Roland Varga, Pierre Antoine Denarie, Baijing Yuan, Daniel Barroso Plata, Arjan Vonk, Surya Narayanan, Theodoulos Kapnisis, Valentin Wurzbauer, Csaba Balla Somogyi, Jingru Feng, Dong Shen, Rohan Chandrashekar, Daniel Varela, Chenghao Xu, Pepijn Bogaard and many others for your encouragement and our beautiful friendships. In addition, I would love to thank Robert Dujmovic for your mentorship and career guidance.

Finally, a special thanks goes to my parents, grandparents, cousins and all other relatives. Thank you mama and papa. I dedicate this manuscript to you. And thank my girlfriend Qingyi Ren for your love, patience, trust and joy, despite the nearly 10,000 km distance and the six-hour time difference between us.

Delft University of Technology Zhehan Li
July 18, 2022

“知己知彼，百战不殆”

— 《孙子 · 谋攻篇》

“If you know both the enemy and yourself, you will fight a hundred battles without danger of defeat”

— 《*SUN-TZU: The Art Of Warfare*》

Chapter 1

Introduction

Models of real-world systems are of fundamental importance in virtually all disciplines [1]. From a control engineering perspective, system modeling determines the quality of the final problem solution. This is because the quality of most advanced techniques in control engineering (such as controller design, optimization, supervision and fault detection, etc.) is often constrained by the quality of the modeling. As a result, there is a significant demand for advanced modeling techniques.

Empirical modeling based on measured input–output data is one of the essential tools in the field of system modeling. It begins with the selection of a suitable model structure, followed by the estimation of model parameters using an identification method by processing the input–output signals. The field of system identification uses statistical methods to build mathematical models of dynamical systems from measured data [2].

For the identification of linear systems, there are extensive advanced algorithms available in literature [3, 4, 5]. Most real-life systems are nonlinear in nature so that nonlinear models are often preferred over linear models for accurately representing the studied system. However, identification of complex nonlinear systems is a difficult endeavor due to several challenges, some of which are the lack of understanding of system dynamics, the huge computational and storage cost, and the availability of only short and partially observed measurements [6].

The convolution integral and the convolution summation that describe the behavior of linear time-invariant (LTI) systems in continuous and discrete time domain, respectively, can also be used for the description of the nonlinear time-invariant (NTI) systems. Examples of using convolution expressions to describe the LTI and NTI systems are shown in Figure 1-1. The mathematician and physicist, Vito Volterra, used series of convolution expressions to define the input-output relationship of NTI systems in the continuous time domain, mathematically

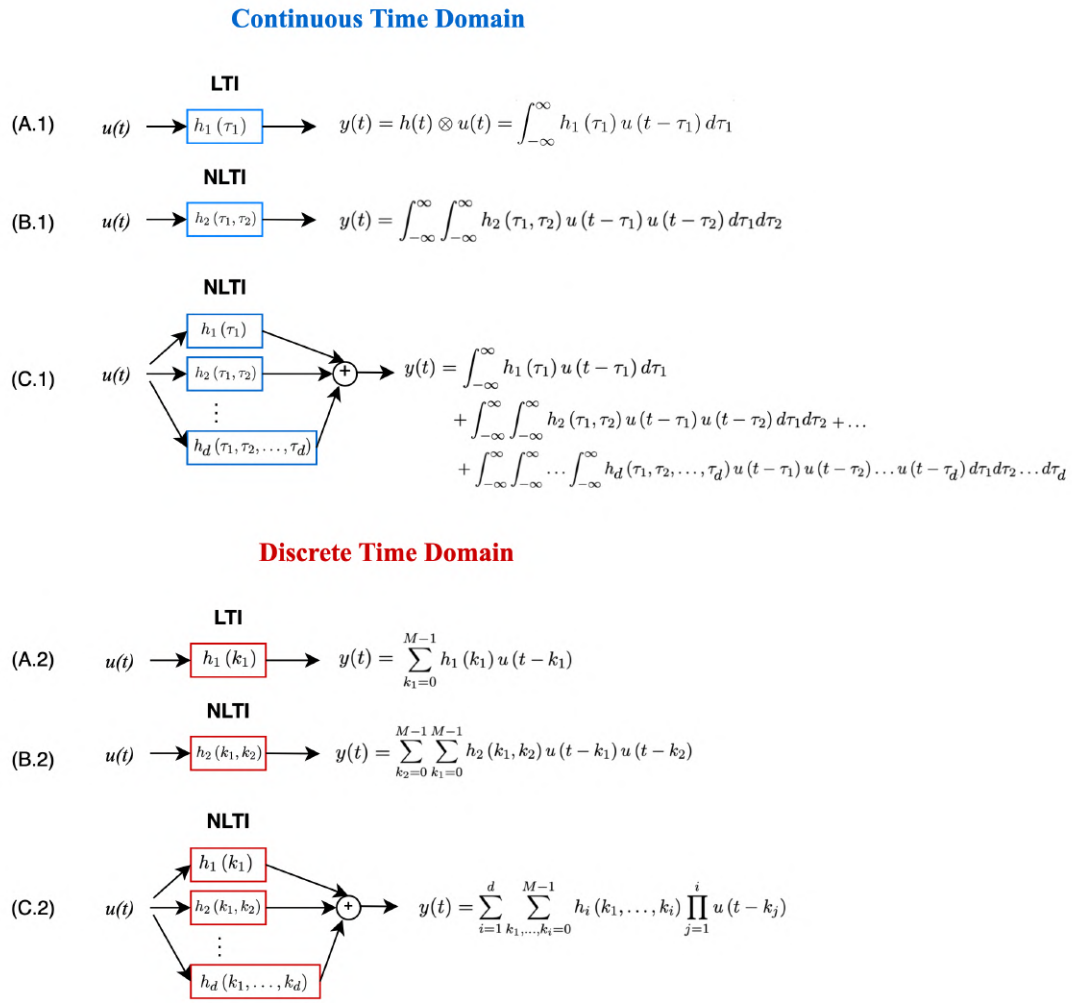


Figure 1-1: Example of LTI and NTI systems pairwise in continuous (.1) and discrete (.2) time domain. (A.1)(A.2): the first order LTI systems. (B.1)(B.2): the second-order NTI systems. (C.1)(C.2): the combined higher-order systems. Note that in the continuous time domain the delay is represented by $\tau \in (-\infty, \infty)$. And in the discretized time domain the truncated delay is denoted by $k \in [0, M-1]$, where M stands for the input memory. The output of the d^{th} order system is represented by a Volterra series consisting of the sum of the individual components, each determined by a convolution expressions in the continuous time domain, or summation expression in the discretized time domain.

expressed as [7]:

$$y(t) = \overbrace{\int_{-\infty}^{\infty} h_1(\tau_1)\, u(t-\tau_1)\, d\tau_1}^{1^{st}\text{ order Term}} + \overbrace{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_2(\tau_1,\tau_2)\, u(t-\tau_1)\, u(t-\tau_2)\, d\tau_1 d\tau_2}^{2^{nd}\text{ order Term}} + \dots + \underbrace{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\dots\int_{-\infty}^{\infty} h_n(\tau_1,\tau_2,\dots,\tau_n)\, u(t-\tau_1)\, u(t-\tau_2)\dots u(t-\tau_n)\, d\tau_1 d\tau_2 \dots d\tau_n}_{n^{th}\text{ order Term}} \tag{1-1}$$

where the output $y(t)$ of an n^{th} order system is the summation of the first, second, ..., n^{th} order convolution terms. These terms depend on the weighting functions $h_1, h_2, \dots, h_n$ named as the first, second, ..., n^{th} order Volterra kernels, and the multiple copies of the lagged input $u(t-\tau_1), u(t-\tau_2), \dots, u(t-\tau_n)$ with delay τ. In some texts [8], a 0^{th} order kernel h_0 representing the bias or offset, is added to Equation 1-1. Systems expressed by Equation 1-1 are called Volterra systems.

Regarding the identification of Volterra systems, most research focus on the discrete time domain [9, 10]. Therefore, this thesis also focuses on the identification of the discrete time truncated Volterra system. In the discrete time domain, the d^{th} order response of a single-input single-output (SISO) truncated Volterra system with finite memory M is defined as follows [11]:

Definition 1: Discrete time truncated d^{th} order SISO causal Volterra system with finite input memory M

$$\begin{aligned} y(t) &= y_0(t) + y_1(t) + y_2(t) + \dots + y_d(t) \\ &= h_0 + \sum_{i=1}^{d} \sum_{k_1,\dots,k_i=0}^{M-1} h_i(k_1,\dots,k_i) \prod_{j=1}^{i} u(t-k_j) \end{aligned} \tag{1-2}$$

where $y_i(t)$ and $h_i(k_1,\dots,k_i)$ denote the i^{th} order output at time instance t and the i^{th} order Volterra kernel with delay indices $(k_1,\dots,k_i)$, respectively. Only the causal system is considered, which implies $h_i(k_1,\cdots,k_i) = 0$ when any of the delay indices is negative. The notation $\sum_{k_1,\dots,k_i=0}^{M-1} = \sum_{k_1=0}^{M-1}\cdots\sum_{k_i=0}^{M-1}$ has been used for simplicity.

The output $y(t)$ relies on the input at time instant t as well as its values before t, which implies that the output $y(t)$ is described by polynomial expressions of the input $u(t-k_j)$ at different k_j. Therefore, the input–output relationship is nonlinear.

Volterra systems have been extensively researched and widely used in modeling the nonlinear systems, such as nonlinear networked system control [12, 13], nonlinear communication channel identification [14, 15], as well as human cortical responses modeling [16].

For the Volterra system in Definition 1, the kernels are playing the most important role. If the dynamic of a system is known precisely, such that its dynamic can be described by a (several) mathematical analytical function(s), the kernels of this system can be retrieved

directly. However, in most cases when only the input-output measurements are known, kernel estimation concerns a huge number of parameters, which suffers from the following two issues:

1. **Curse Of Dimensionality**: The second order Volterra kernel $\sum_{k_1,k_2=0}^{M-1} h_2(k_1,k_2)$ in Equation 1-1 can be arranged as a square matrix with dimension of $(M \times M)$, mathematically expressed as,

$$\sum_{k_1,k_2=0}^{M-1} h_2(k_1,k_2) = \begin{pmatrix} h_2(0,0) & h_2(0,1) & \cdots & h_2(0,M-1) \\ h_2(1,0) & h_2(1,1) & \cdots & h_2(1,M-1) \\ \vdots & \vdots & \ddots & \vdots \\ h_2(M-1,0) & h_2(M-1,1) & \cdots & h_2(M-1,M-1) \end{pmatrix} \tag{1-3}$$

   which can be extended to the third order kernel that consists of M square matrices having the dimension of $(M \times M)$. For the d^{th} order Volterra kernel, it is modelled by M^d coefficients. For a multiple-input multiple-output (MIMO) Volterra system with p inputs and l outputs, the situation becomes even worse. The d^{th} order Volterra kernel for one particular output is characterized by $(pM)^d$ coefficients, which increases exponentially as pM and d grow. It is practically infeasible to store and estimate these coefficients. Such issue is commonly known as the curse of dimensionality [17].

2. **Delay Index Permutation**: Each coefficient $h_d(k_1,\cdots,k_d)$ of the d^{th} order Volterra kernel corresponds to an input product term $\prod_{j=1}^{d} u(t-k_j)$. Both of them are determined by the delay indices $k_1,\cdots,k_d$. For all distinct permutations of the indices $k_1,\cdots,k_d$, the d^{th} order Volterra kernel contains repeated coefficients, which can be expressed as follows,

$$h_d(k_1,\cdots,k_d)\prod_{j=1}^{d} u(t-k_j) = h_d\left(k_{\pi(1)},\cdots,m_{\pi(d)}\right)\prod_{j=1}^{d} u\left(t-k_{\pi(j)}\right) \tag{1-4}$$

   where $\pi(.)$ stands for all distinct permutations of the indices $k_1,\cdots,k_d$. The number of unique coefficient of the SISO d^{th} order Volterra kernel is less than M^d. Similarly, the number of unique coefficient of the MIMO d^{th} order Volterra kernel for one particular output is less than $(pM)^d$. The issue of delay index permutation results in the unnecessary over-count of the number of kernel coefficients.

The issue of delay index permutation can be solved by grouping the product terms in Equation 1-4. Then a symmetric kernel $\sum_{k_1,k_2=0}^{M-1} h_d^{\text{sym}}(k_1,\cdots,k_d)$ is obtained by summing the coefficients $h_d(k_1,\cdots,k_d)$ over all distinct permutations π of the delay indices and dividing by the number of such permutations denoted as $\text{num}(\pi(k_1,\cdots,k_d))$. For example, all the repeated coefficients in Equation 1-4 are represented by one unique coefficient $h_d^{sym}(k_1,\cdots,k_d)$ of the symmetric kernel, expressed as:

$$\begin{aligned} h_d^{sym}(k_1,\cdots,k_d) &= \frac{\sum_{\pi(.)} h_d\left(k_{\pi(1)},\cdots,m_{\pi(d)}\right)}{\text{num}(\pi(k_1,\cdots,k_d))} \\ &= \frac{\sum_{\pi(.)} h_d\left(k_{\pi(1)},\cdots,m_{\pi(d)}\right)}{d\,!}. \end{aligned} \tag{1-5}$$

The number of unique coefficients of the d^{th} order SISO symmetric Volterra kernel is $C_d^{M+d-1} = \frac{(M+d-1)!}{d!(M-1)!}$ [18]. Figure 1-2 shows the growth of the number of parameters of the Volterra kernels h_d and the symmetric Volterra kernels h_d^{sym} with increasing d for $M \in [5, 10, 20, 40]$, respectively. Although the number of coefficients of h_d^{sym} with a certain M grows much slower than the number for h_d, it still requires a large storage complexity.

State-of-the-art research has reduced the storage complexity for symmetric Volterra kernel using the tensor decomposition approaches. The Parallel factor (PARAFAC) tensor [1] decomposition is applied on the symmetric Volterra kernels in [18]. The symmetric Volterra kernels consist of $h_0, h_1^{sym}, \cdots, h_d^{sym}$. The obtained model is called the Volterra-PARAFAC model. To make it easier to express the mathematical concepts, we denote this model by $f(.)$ as follows,

$$[h_0, h_1^{sym}, \cdots, h_d^{sym}] = f(A_1, \cdots, A_d) \tag{1-6}$$

where $A_1, \cdots, A_d$ are assumed to be the coefficients of the Volterra-PARAFAC model [2]. This model induces a significant storage complexity reduction. Several adaptive algorithms are proposed to identify the coefficients of the Volterra-PARAFAC model when input-output signals are complex-valued. However, the determination of the rank for PARAFAC decomposition is an NP-hard problem [19], and the identification performance suffers from the numerical instability when trying to find proper rank values. In addition, the Volterra systems are described in the Tensor Network (TN) formats by using the Tensor Train (TT) decomposition in [9, 11]. These approaches are explained in Appendix A-2.

The computation of PARAFAC decomposition of a given tensor has been extensively researched [20, 21]. Optimisation-based algorithms have been used to estimate the optimal value of the PARAFAC parameters by minimising the least-squares (LS) cost function, which is the Frobenius norm of the difference between the given tensor and the estimated tensor [22]. For convenience, we denote the given tensor as $\mathcal{V}_{tru}$, and the estimated tensor as $\mathcal{V}_{est}$, the PARAFAC decomposition as function $g(.)$, and the PARAFAC parameters as $B_1, \cdots, B_m$. The LS cost function can then be expressed as follows,

$$\min_{B_1, \cdots, B_m} \|\mathcal{V}_{\mathbf{tru}} - \mathcal{V}_{\mathbf{est}}\|_F, \quad \mathcal{V}_{est} = g(B_1, \cdots, B_m) \tag{1-7}$$

where the optimization variables are the PARAFAC parameters, and the estimated tensor is recovered as $g(B_1, \cdots, B_m)$. The Frobenius norm is denoted as $\|\|_F$.

The majority of optimisation-based methods for calculating the PARAFAC decomposition are first-order methods that use gradient term of the LS cost function in Equation 1-7 to determine the next optimization step [23, 24]. These methods have been shown to be susceptible to convergence problems, such as requiring a large number of iterations, being unstable to reach the global minimum, and being sensitive to the initial conditions. Using second-order information from the Hessian term of the cost function, like the Gauss-Newton (GN) optimizer, can lead to fewer iterations and improved convergence in general [25].

[1] The definition of tensor is introduced in the first paragraph of section 2-1. Readers who are unfamiliar with tensor are advised to read there first and then proceed with the remainder of this chapter.

[2] Equation 1-6 and Equation 1-7 are the conceptional models. The calculation of $f(.)$ and $g(.)$, as well as the construction of $A_1, \cdots, A_d$ and $B_1, \cdots, B_m$ are explained explicitly in the next chapter.

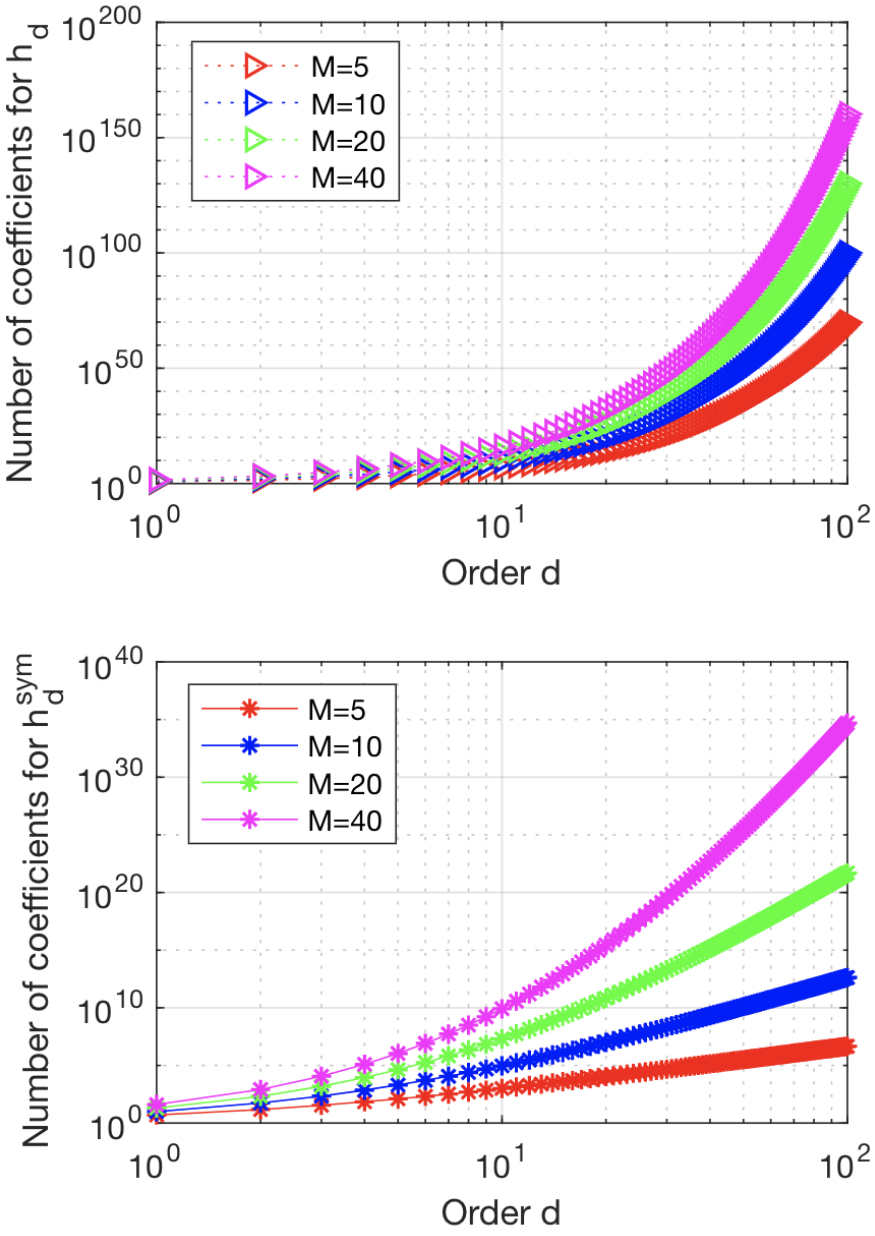


Figure 1-2: The number of parameters for SISO Volterra kernels h_d (top) and h_d^{sym} (bottom) for different M grows exponentially and factorially with d, respectively. The latter grows much slower than the former.

1-1 Thesis Motivation

The optimisation-based algorithms discussed in Equation 1-7 can not be applied to Volterra identification problem directly. The Volterra identification problem stands for the situation where only the input and output measurements $\{y(t), u(t)\}_{t=[0,N-1]}$ of Equation 1-2 are known, and the ground truth Volterra kernels $h_0, h_1^{sym}, \cdots, h_d^{sym}$ are unknown and needed to be estimated. The Volterra identification problem can be expressed by the following optimization framework,

$$\min_{h_0, h_1^{sym}, \cdots, h_d^{sym}} \|\mathbf{y} - \hat{\mathbf{y}}\|_F \tag{1-8}$$

where $\mathbf{y} = y(t)_{t=[0,N-1]}$ and $\hat{\mathbf{y}} = \hat{y(t)}_{t=[0,N-1]}$ stand for the ground truth and estimated output measurements, respectively. The estimated output measurements can be recovered based on the estimated Volterra kernels and the input measurements following Equation 1-2.

If the Volterra kernels $h_0, h_1^{sym}, \cdots, h_d^{sym}$ in Equation 1-8 are modeled using the Volterra-PARAFAC model in the form of Equation 1-6 , the optimization variables of Equation 1-8 become the coefficients of the Volterra-PARAFAC model. Furthermore, if the connection between Equation 1-8 and Equation 1-7 could be found, the second order optimization algorithm to solve Equation 1-7 could be applied (with some changes) to solve Equation 1-8. As a result, it is crucial to investigate how to establish one such connection and how to use the second order optimization technique to solve Equation 1-8 using the Volterra-PARAFAC model.

Recent research has applied the second and higher order Volterra systems [3] to model the highly nonlinear human cortical responses [16]. The PARAFAC decomposition has not yet been used to reduce the storage complexity of the modelled higher order Volterra system. Therefore, it is meaningful to investigate if the Volterra-PARAFAC model can be used to model the highly nonlinear human cortical responses and then estimated by the above mentioned optimization-based method.

1-2 Thesis Objective

The thesis objective is to answer the following main research question:

Research Question

How to develop, validate and apply an identification framework that can estimate the parameter values of the Volterra-PARAFAC model from the given input and output measurements using a second order optimization algorithm?

To answer the above defined main research question, the following phases are formulated. Each phase contains several research sub-questions that will be answered at the end of each following chapters.

[3] A higher order Volterra system denotes a Volterra system with an order value d equal to or greater than three.

Phase I: Development

1. *How does the Volterra-PARAFAC model address the issue of curse of dimensionality in identifying the Volterra kernel parameters? If it does not solve the issue, what is the lacking factor?*
2. *What is(are) the benefit(s) of using the Gauss-Newton (GN) optimizer to optimize the Least squares (LS) cost function? How to calculate the components of this optimizer?*
3. *The GN optimizer minimises the Least squares (LS) cost function starting from an initial value. How to initialise this optimization procedure?*
4. *Does the developed framework apply to the MIMO case? If not, which step (component) of the developed framework should be modified in order to do so?*

Phase II: Validation

1. *How to validate that the synthetic Volterra-PARAFAC model is estimated correctly using the developed framework?*
2. *What is(are) the suitable figure(s) of merit to reflect the identification performance of the proposed framework?*
3. *What is(are) the hyper-parameter(s) of the proposed framework? What is the effect of having different hyper-parameter(s) values on the identification performance?*
4. *Is the identification performance influenced by the initial value for the optimizer? What are the advantages and disadvantages of different initialization methods?*
5. *When there is a storage complexity budget, what are the advantages and disadvantages of the proposed framework compared to the existing frameworks on identifying the simulated artificial Volterra system?*

Phase III: Application

1. *The human cortical responses are modeled by the Volterra-PARAFAC model whose kernel coefficients are estimated by the developed framework. Does the modeling quality rely on the values of the hyper-parameter(s) of the framework?*
2. *How to test the plausibility of the modelled system? What are the potential reason(s) for the modelled system not representing the true underlying system?*

This research is innovative for the following reasons:

- The second order information from the cost function has not previously been used for identifying the Volterra-PARAFAC model.

- The experiment to compare the Volterra identification performance of different tensor-based frameworks under the storage complexity budget has not been explicitly performed before.
- The human cortical responses have not been modelled by the Volterra-PARAFAC model and estimated by the second order optimization method.

An open-source MATLAB implementation of the proposed framework and the experiments performed in this thesis is made available on Github.

1-3 Thesis Outline

This brings us to the following chapter-by-chapter outline of the thesis which serves as providing the reader with a structural overview of the thesis. The thesis is structured in such a way that the research questions of each phase are answered through a separate chapter.

Chapter 2 describes the development of the proposed Volterra system identification framework, which consists of two stages. The first stage models the input-output relationship using the tensor operations and a decomposition technique, while the second stage estimates the parameter values of the modelled system using a second-order numerical optimization algorithm.

Chapter 3 validates internally and externally the developed framework by identifying an artificial Volterra system. The internal validation focuses on the influence of the hyper-parameter and different initialization methods of the optimization algorithm on the identification performance, while the external validation compares the developed framework with three TN based frameworks on identifying the same artificial Volterra system under the storage complexity budget.

Chapter 4 applies the validated framework to model the evoked cortical responses. The modification of the data set to fit the validated framework and the update of the hyper-parameter values to improve the modeling quality are explained first. The challenges and the plausibility of the modelled system are then discussed.

Chapter 5 summarizes the thesis and provides an overview of future research.

Appendix A contains three sections. In Appendix A-1, the tensor operations that are used in this thesis are explained. In Appendix A-2, the storage complexity of three TN based frameworks is calculated individually and used in this thesis. In Appendix A-3, the persistently exciting condition for the input signal of the Volterra system is reviewed from [11].

Appendix B contains two sections. In Appendix B-1, several second-order optimization algorithms to approximate the Hessian term of the cost function are introduced. In Appendix B-2, it provides the algebraic initialization method for the developed optimization framework. The computational and storage complexity of this method are calculated and used in this thesis. This method is a modified version of the result from [22].

Chapter 2

Development: From Tensor To Numerical Optimization

This chapter aims to describe the development of the proposed tensor-based Volterra system identification framework. Section 2-1 covers the symbolic notations and the terminology of the tensor technique. The Volterra tensor is defined in section 2-2 to model the input-output relationship of the multiple-input single-output (MISO) Volterra system. The two following sections contain the essential innovation of this thesis. In section 2-3 the symmetric canonical polyadic decomposition (CPD) is applied on the Volterra tensor, and a linear framework is proposed to model the MISO Volterra system. The application of a second-order numerical optimization algorithm to estimate the linear framework parameters is introduced in section 2-4. This chapter ends by answering the research sub-questions of Phase I in section 2-5.

2-1 Tensor Basics

Tensors are multi-dimensional arrays that generalize the notions of vectors and matrices to higher orders. In a vector, each entry is determined by one index. In a matrix, there are two indices. In a d^{th} order tensor, each of its entries is determined by d indices. Scalars are denoted as Roman letters $x \in \mathbb{R}$, vectors as lower-case boldface letters $\mathbf{x} \in \mathbb{R}^{I_1}$, matrices as capital bold letters $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$. d^{th} order tensors are given as capital calligraphic letters $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$. $\mathbb{R}$ denotes the fields of real numbers. The positive integers $d, I_1, I_2, \ldots, I_d$ are called the order and the dimensions of the tensor, respectively. Note that the terminology *order*, *way* and *mode* of a tensor are interchangeable; and vectors or matrices can also be written as 1^{st} or 2^{nd} order tensors, respectively.

The notations of some basic linear algebra operations are defined as follows. $\mathbf{X}^T$, $\mathbf{X}^{-1}$ and $\mathbf{X}^\dagger$ stand for the transpose, inverse and Moore-Penrose pseudoinverse of $\mathbf{X}$, respectively. The n^{th} element in a sequence is denoted by a superscript in parentheses. For example, $x^{(1)}, x^{(2)}, x^{(3)}$ denote the first three elements of the vector $\mathbf{x}$. The exponentiation operation is denoted by

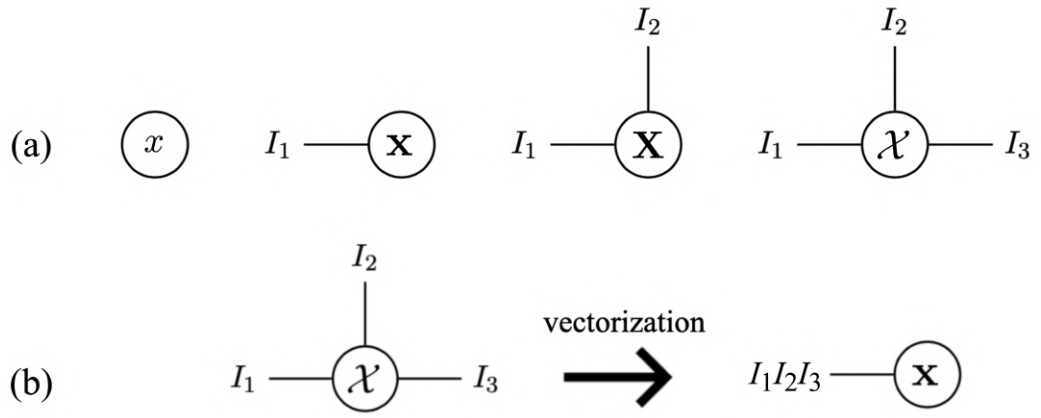


Figure 2-1: Diagrammatic notations of tensors. (a) From left to right: a scalar $x \in \mathbb{R}$, a vector $\mathbf{x} \in \mathbb{R}^{I_1}$, a matrix $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$, and a 3 way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$; (b) The vectorization of the 3 way tensor $\mathcal{X}$ into a vector $\mathbf{x} \in \mathbb{R}^{I_1 I_2 I_3}$.

the superscript of the exponent in the top right part of the base. For example, $\mathbf{u}^d$ stands for $\mathbf{u}$ raised to the power of d, where d is the exponent and $\mathbf{u}$ is the base.

Tensors are described and computed more easily when using diagrammatic notations adopted from physics and quantum chemistry [26]. The notations for a scalar, a vector, a matrix, and a three-way tensor are shown in Figure 2-1 (a). Each node corresponds to a tensor. The edges reflect the indices of tensor indices, and hence the number of edges correlates to the order of tensor. The edges are interchangeable. A node with one edge is a vector, or a 1^{st} order tensor.

Reshaping is a critical operation on tensors. The reshaping method used in this thesis is vectorization, which reorganizes the entries of a d way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ into a vector $\mathbf{x} \in \mathbb{R}^{I_1 \cdots I_d}$. The operation of vectorization of tensor $\mathcal{X}$ is denoted as $vec(\mathcal{X})$. In Figure 2-1 (b), the diagrammatic notation of the vectorization of a 3 way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ into a vector $\mathbf{x} \in \mathbb{R}^{I_1 I_2 I_3}$, denoted as $vec(\mathcal{X})$ is provided.

A tensor is called cubical if each mode has the same size, for example a d way cubical tensor $\mathcal{X} \in \mathbb{R}^{I \times \cdots \times I}$. Furthermore, this d way cubical tensor is symmetric if its elements stay constant regardless of any index permutation, expressed as follows:

$$\mathcal{X}^{(i_1, i_2, \ldots, i_d)} = \mathcal{X}^{\pi(i_1, i_2, \ldots, i_d)} \tag{2-1}$$

where $\pi(.)$ stands for all distinct index permutations.

Tensor operations are higher order equivalents to vector and matrix operations. Tensor contraction and product serve as the mathematical foundations in this chapter. The mode-n product is denoted as $\times_n$. The Kronecker, Khatri-Rao, row-wise Khatri-Rao, Hadamard and outer products are denoted as $\otimes$, $\odot$, $\odot^T$, $*$, and $\circ$, respectively. The calculation details of these tensor operations are provided in Appendix A-1.

The properties of the tensor operations that are used in this thesis are summarized as follows,

Property 1: Properties Of Tensor Operations [27]

The multidimensional contraction of d way symmetric tensor $\mathcal{A} \in \mathbb{R}^{I \times \cdots \times I}$ with a vector $\mathbf{x} \in \mathbb{R}^N$ is the scalar

$$\mathcal{A}\mathbf{x}^d := \mathcal{A} \times_1 \mathbf{x}^T \times_2 \mathbf{x}^T \times_3 \cdots \times_d \mathbf{x}^T \tag{2-2}$$

which equals to

$$\mathcal{A}\mathbf{x}^d := (\mathbf{x} \otimes \mathbf{x} \otimes \cdots \otimes \mathbf{x}) \operatorname{vec}(\mathcal{A}) \tag{2-3}$$

The product of the row-wise Khatri-rao product of $\mathbf{A} \in \mathbb{R}^{N \times I}$ and Khatri-rao product of $\mathbf{B} \in \mathbb{R}^{I \times R}$ can be calculated by the Hadamard product operation as follows,

$$(\mathbf{A} \odot^T \mathbf{A})(\mathbf{B} \odot \mathbf{B}) = \mathbf{AB} * \mathbf{AB} = (\mathbf{AB})^{.2} \in \mathbb{R}^{N \times R} \tag{2-4}$$

where $()^{.t}$ represents the element-wise t^{th} power.

2-2 Volterra Tensor

The above introduced tensor techniques are used to model the input-output relationship of the discrete time d^{th} order truncated Volterra system. Since the single-input single-output (SISO) system expressed in Equation 1-2 belongs to a multiple-input single-output (MISO) system, the following definitions and properties of MISO systems also apply to SISO systems.

Based on Equation 1-2, the input-output relationship of the d^{th} order MISO Volterra system is expressed as follows,

$$\begin{aligned} y(t) :&= y_0(t) + y_1(t) + y_2(t) + \cdots + y_d(t) \\ &= h_0 + \sum_{i=1}^{d} \sum_{z_1,\ldots,z_i=1}^{P} \sum_{k_1,\ldots,k_i=0}^{M-1} h_i\left(z_1, \ldots, z_i; k_1, \ldots, k_i\right) \prod_{j=1}^{i} u_{z_j}\left(t - k_j\right) \end{aligned} \tag{2-5}$$

where $h_i\left(z_1, \ldots, z_i; k_1, \ldots, k_i\right)$ denotes the i^{th} order Volterra kernel for the multiplication of the ${z_1}^{th}$ to ${z_i}^{th}$ inputs at time instance t with delay indices of k_1 to k_i for each input, respectively. The output $y(t)$ is a multivariate polynomial of degree d in inputs $u_i(t-\tau)$ for $\tau \in [0, M-1]$ and $i \in [1, p]$.

The d^{th} order Volterra kernel in Equation 2-5 has coefficient number of $(pM)^d$. These coefficients can be represented by a vectorized d way cubical tensor $\mathcal{V}_d \in \mathbb{R}^{pM \times \cdots \times pM}$. For simplicity, the input sequence $\boldsymbol{u}_t \in \mathbb{R}^{(pM+1)}$ is defined as a row vector with entries of 1, $u_1(t)$, $u_2(t)$,..., $u_p(t)$, ..., $u_1(t-M+1)$, $u_2(t-M+1)$,..., $u_p(t-M+1)$. The input-output relationship in Equation 2-5 can be formulated as the multidimensional contraction of the d way MISO Volterra tensor $\mathcal{V} \in \mathbb{R}^{(pM+1) \times \cdots \times (pM+1)}$ and $\mathbf{u}_t^T \in \mathbb{R}^{1 \times (pM+1)}$, by defining the d way MISO Volterra tensor as follows,

Definition 2: MISO Volterra Tensor [11]

For the MISO Volterra system in Equation 2-5, the d-way cubical Volterra tensor $\mathcal{V}$ is defined by

$$\begin{aligned} y(t) :&= y_0(t) + y_1(t) + y_2(t) + \cdots + y_d(t) \\ &= h_0 + \sum_{i=1}^{d} \sum_{z_1,\ldots,z_i=1}^{P} \sum_{k_1,\ldots,k_i=0}^{M-1} h_i\left(z_1,\ldots,z_i;k_1,\ldots,k_i\right) \prod_{j=1}^{i} u_{z_j}\left(t-k_j\right) \\ &= \mathcal{V}\,\mathbf{u}_t^d \\ &= \mathcal{V} \times_1 \mathbf{u}_t^T \times_2 \mathbf{u}_t^T \cdots \times_d \mathbf{u}_t^T \\ &= \left(\mathbf{u}_t^T \otimes \mathbf{u}_t^T \otimes \cdots \otimes \mathbf{u}_t^T\right) \mathrm{vec}(\mathcal{V}) \end{aligned} \tag{2-6}$$

where the diagrammatic notation of the multidimensional contraction of $\mathcal{V}$ and $\mathbf{u}_t^T$ is shown in Figure 2-2. The last three rows of Equation 2-6 are based on the tensor operation properties in Equation 2-2 and Equation 2-3.

By storing the output measurements $\{y(t)\}_{t=[0,N-1]}$ in a row vector, Equation 2-6 can be expressed as follows,

$$\begin{aligned} \mathbf{y} \in \mathbb{R}^N = \begin{pmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{pmatrix} &= \begin{pmatrix} \mathbf{u}_0^T \otimes \mathbf{u}_0^T \otimes \cdots \otimes \mathbf{u}_0^T \\ \mathbf{u}_1^T \otimes \mathbf{u}_1^T \otimes \cdots \otimes \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_{N-1}^T \otimes \mathbf{u}_{N-1}^T \otimes \cdots \otimes \mathbf{u}_{N-1}^T \end{pmatrix} \mathrm{vec}(\mathcal{V}) \\ &= \left(\begin{pmatrix} \mathbf{u}_0^T \\ \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_{N-1}^T \end{pmatrix} \odot^{\mathrm{T}} \begin{pmatrix} \mathbf{u}_0^T \\ \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_{N-1}^T \end{pmatrix} \odot^{\mathrm{T}} \cdots \odot^{\mathrm{T}} \begin{pmatrix} \mathbf{u}_0^T \\ \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_{N-1}^T \end{pmatrix} \right) \mathrm{vec}(\mathcal{V}) \\ &= \left(\mathbf{U} \odot^{\mathrm{T}} \mathbf{U} \odot^{\mathrm{T}} \cdots \odot^{\mathrm{T}} \mathbf{U}\right) \mathrm{vec}\left(\mathcal{V}\right) \\ &:= \tilde{\mathbf{U}}\,\mathrm{vec}\left(\mathcal{V}\right) \end{aligned} \tag{2-7}$$

where the symbol $\mathbf{U}$ stands for a matrix that stacks $\mathbf{u}_t^T$ row-wisely for $t \in [0, N-1]$, expressed as follows

$$\mathbf{U} = \begin{pmatrix} \mathbf{u}_0^T \\ \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_{N-1}^T \end{pmatrix} \in \mathbb{R}^{N \times (pM+1)} \tag{2-8}$$

Equation 2-7 is a linear system with d-times row wise Khatr-Rao product of $\mathbf{U}$ as the coefficient matrix, and $\mathrm{vec}\left(\mathcal{V}\right)$ as the solution. In [11], the condition for the input signal being persistently exciting of order d is provided and reviewed in Appendix A-3. According to [11], if the input is persistently exciting and $N \geq \mathrm{rank}(\tilde{\mathbf{U}})$, there are infinite number of solu-

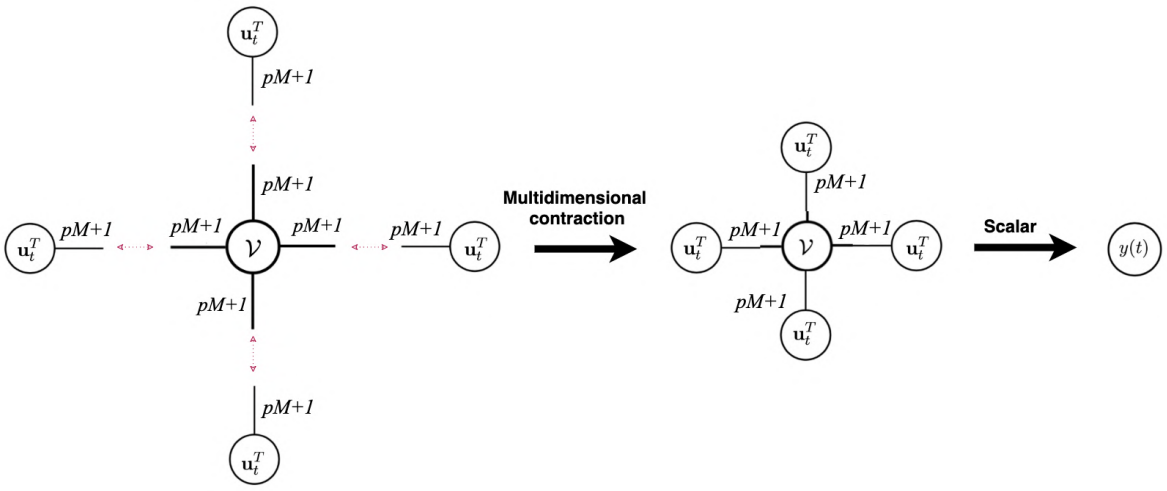


Figure 2-2: Diagrammatic notations of the multidimensional contraction operation of the 4-way Volterra tensor $\mathcal{V}$ and the transpose of the input sequence $\mathbf{u}_t^T$. The result is the scalar $y(t)$. The red arrow represents the product of $\mathcal{V}$ and $\mathbf{u}_t^T$ in each mode of $\mathcal{V}$.

tions $\text{vec}(\mathcal{V}) \in \mathbb{R}^{(pM+1)^d}$ satisfying Equation 2-7. One of the solutions that has $\begin{pmatrix} pM+d \\ pM \end{pmatrix}$ distinct coefficients must correspond to a vectorized d-way symmetric tensor, denoted as $\text{vec}(\mathcal{V}_{sym})$. The uniqueness of one symmetric solution implies the absence of more symmetric solutions.

By symmetrizing the Volterra tensor $\mathcal{V}$, the solution $\text{vec}(\mathcal{V}_{sym})$ is the unique solution of Equation 2-7, expressed as,

$$\mathbf{y} \in \mathbb{R}^N = \left(\mathbf{U} \odot^{\mathrm{T}} \mathbf{U} \odot^{\mathrm{T}} \cdots \odot^{\mathrm{T}} \mathbf{U}\right) \text{vec}(\mathcal{V}_{sym}) \tag{2-9}$$

where the storage complexity of the solution $\text{vec}(\mathcal{V}_{sym})$ is still large because all the unique elements of $\text{vec}(\mathcal{V}_{sym}) \in \begin{pmatrix} pM+d \\ pM \end{pmatrix}$ grow factorially with d.

In the next section, the tensor decomposition technique is introduced. After applying this technique on $\text{vec}(\mathcal{V}_{sym})$, the unique elements of $\text{vec}(\mathcal{V}_{sym})$ are replaced by only a few numbers.

2-3 Symmetric CPD On Volterra Tensor

Building on the well-established field of matrix decomposition technique such as the singular value decomposition (SVD) approach, various tensor decomposition techniques have been proposed to reveal the underlying information in a tensor in a compact way by minimizing the storage complexity [28]. One of these tensor decomposition techniques is the canonical polyadic decomposition (CPD). It was introduced by Hitchcock [20] and it is also named as PARAFAC, canonical decomposition (CANDECOMP), and rank decomposition. CPD is a versatile model that has been used in a wide variety of applications in machine learning, computer vision, data mining, signal processing, and biomedical sciences [29, 23, 30].

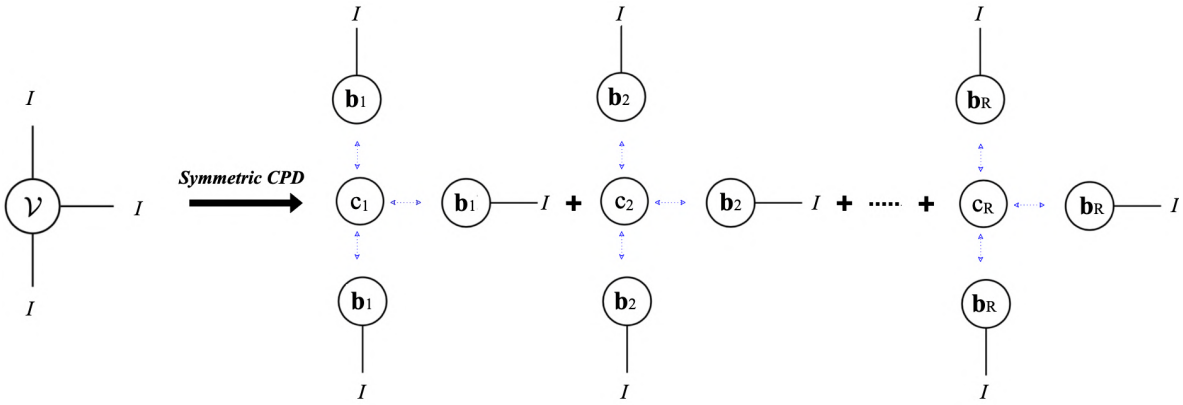


Figure 2-3: Diagrammatic notation of the symmetric CPD for a third-order Volterra tensor. The blue arrow represents the outer product of $\mathbf{b}_i$.

From now on, the notation I is used to replace $pM + 1$. For the d-way Volterra tensor $\mathcal{V}$, it can be represented by a sum of R rank-1 terms, mathematically expressed as,

$$\mathcal{V} \approx \sum_{r=1}^{R} c_r \mathbf{b}_r^{(1)} \circ \mathbf{b}_r^{(2)} \circ \cdots \circ \mathbf{b}_r^{(d)} \tag{2-10}$$

where $c_r \in \mathbb{R}$ is a scalar, $\mathbf{b}_r^{(i)} \in \mathbb{R}^I$ are vectors for $i = 1, 2 \cdots d$. Each element of the tensor $\mathcal{V}$ is determined by the element-wise product of vector $\mathbf{b}_r^{(i)}$:

$$v^{(i_1 i_2 \cdots i_d)} \approx \sum_{r=1}^{R} c_r \left(\prod_{n=1}^{d} b_r^{(n,i_n)} \right) \tag{2-11}$$

where the scalar $b_r^{(n,i_n)}$ stands for the ${i_n}^{th}$ element of $\mathbf{b}_r^{(n)}$.

The operation performed in Equation 2-10 is called the polyadic decomposition (PD) of the tensor $\mathcal{V}$. Moreover, in case R is the minimal number of the components in an *exact* PD, the operation in Equation 2-10 is also called CPD of the tensor, where the term *exact* applies if the decomposition is not an approximation but an equality. By defining $\mathbf{B}^{(i)} = [\mathbf{b}_1^{(i)}, \mathbf{b}_2^{(i)}, \cdots, \mathbf{b}_R^{(i)}] \in \mathbb{R}^{I \times R}$ and $\mathbf{c} = [c_1, c_2, \cdots, c_R] \in \mathbb{R}^{1 \times R}$, the vectorization of the CPD of tensor $\mathcal{V}$ can also be defined as follows,

$$\text{vec}(\mathcal{V}) := \left(\mathbf{B}^{(1)} \odot \mathbf{B}^{(2)} \odot \cdots \odot \mathbf{B}^{(d)} \right) \cdot \mathbf{c} \tag{2-12}$$

where $\mathbf{B}^{(i)}$ is called the ${i}^{th}$ component factor, and $\mathbf{c}$ is called the weighting vector. Determining the minimum rank R is an NP-hard problem [19].

For the symmetric Volterra tensor $\mathcal{V}_{sym}$, the component factors $\mathbf{B}^{(i)}$ for all values of i are the same. The corresponding symmetric CPD of the Volterra tensor is defined as below:

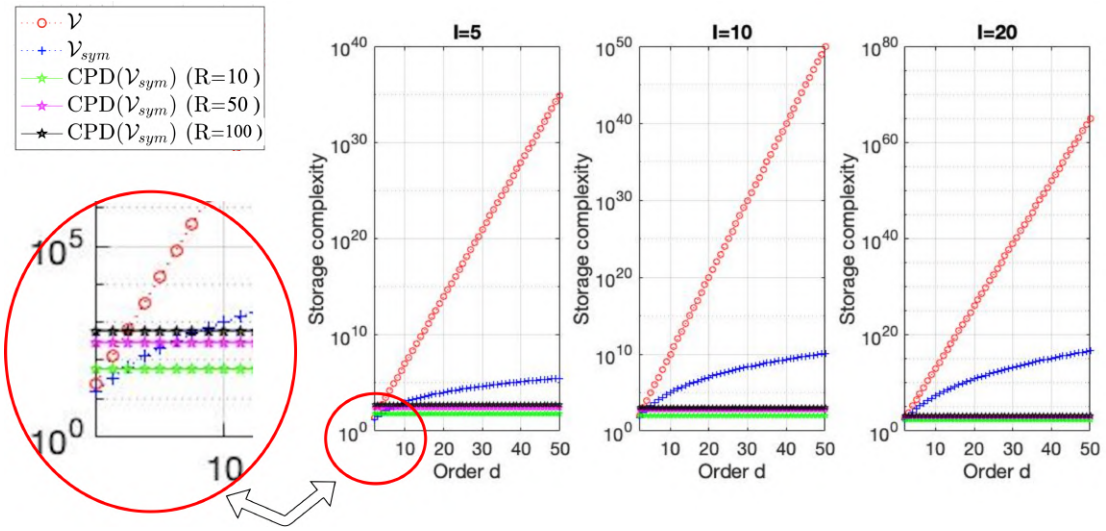


Figure 2-4: The storage complexities of three formats of Volterra tensor: $\mathcal{V}$, $\mathcal{V}_{sym}$ and symmetric CPD formats under different values of d I and R, respectively. In general, the symmetric CPD format reduces the storage complexity significantly for $d > 10$. However, the reduction is insignificant (even counteractive) when $d < 10$ and $R > 50$ as pointed out in the red circle.

Definition 3: Symmetric CPD Of Volterra Tensor

The symmetric CPD of the d-way Volterra tensor $\mathcal{V} \in \mathbb{R}^{I \times I \times \cdots \times I}$ is represented as a sum of R rank-1 terms, mathematically expressed as:

$$\mathcal{V} \in \mathbb{R}^{\overbrace{I \times I \times \cdots \times I}^{d\ times}} = \sum_{r=1}^{R} c_r \overbrace{\mathbf{b}_r \circ \mathbf{b}_r \circ \cdots \circ \mathbf{b}_r}^{d\ times} \tag{2-13}$$

whose diagrammatic notation for the third order system is shown in Figure 2-3. The storage complexity of the symmetric CPD is $\mathcal{O}\left(\mathbf{B} \in \mathbb{R}^{I \times R} + \mathbf{c} \in \mathbb{R}^{R}\right) = \mathcal{O}\left((I+1)R\right)$, which grows linearly with R and I.

Equation 2-13 is also called the Volterra-PARAFAC model, which is discussed in the introduction chapter. In this thesis, the terms symmetric CPD and Volterra-PARAFAC model refer to the operation represented in Equation 2-13.

The symmetric CPD addresses the parametric complexity problem in Equation 2-9, which can be evidenced by Figure 2-4. Figure 2-4 shows the number of coefficients of the Volterra tensor ($N_V = I^d$), the symmetric Volterra tensor $\left(N_{VS} = \begin{pmatrix} I-1+d \\ I-1 \end{pmatrix}\right)$ and its symmetric CPD format ($N_{VP} = R(I+1)$) under different values of I, d and R, respectively. The number of coefficients of the symmetric is not influenced by the value of d, thus it reduces the storage complexity significantly in general. However, as pointed out by the red circle for $d < 10$, the reduction is insignificant for $R = 10$, and even becomes counteractive for $R > 50$.

By performing the symmetric CPD on the Volterra tensor in Equation 2-9, the Volterra system is modeled as a linear system with a symmetric CPD constrained solution and a row-wise Khatri-Rao structured coefficient matrix. A linear system under the same structure is named LS-CPD in [22]. The d^{th} order MISO Volterra system modeling framework in the LS-CPD format is proposed in this thesis, mathematically expressed as follows:

Proposition 1: The d^{th} Order MISO Volterra System Modeling In The LS-CPD Framework

The d^{th} order MISO Volterra system with the input and output measurements can be modeled as:

$$\begin{aligned}
\mathbf{y} \in \mathbb{R}^N = \begin{pmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{pmatrix} &= \left(\mathbf{U} \odot^{\mathrm{T}} \mathbf{U} \odot^{\mathrm{T}} \cdots \odot^{\mathrm{T}} \mathbf{U}\right) \mathrm{vec}\left(\mathcal{V}_{sym}\right) \\
&= \left(\mathbf{U} \odot^{\mathrm{T}} \mathbf{U} \odot^{\mathrm{T}} \cdots \odot^{\mathrm{T}} \mathbf{U}\right) \left(\mathbf{B} \odot \mathbf{B} \odot \cdots \odot \mathbf{B}\right) \cdot \mathbf{c} \qquad (2\text{-}14) \\
&= \begin{pmatrix} d \\ * \\ i=1 \end{pmatrix} \mathbf{UB} \cdot \mathbf{c} \\
&= (\mathbf{UB})^{\cdot d} \cdot \mathbf{c}
\end{aligned}$$

whose derivation is based on the tensor operation property in Equation 2-4.

2-4 Nonlinear Least Squares Optimization Structure For LS-CPD

In this section, several second-order optimization algorithms are considered to search for the solution of Equation 2-14 by optimizing an Nonlinear Least Squares (NLS) objective function. In subsection 2-4-1, the NLS objective function is introduced, and the Gauss-Newton (GN) algorithm is selected. In subsection 2-4-2, the components and procedure of the Gauss-Newton (GN) algorithm to optimize the NLS objective function are explained.

2-4-1 NLS objective function

The solution $\mathcal{V}_{sym}$ of Equation 2-14 given the input-output measurements $\{y(t), u_i(t)\}_{t=[0,N-1]}^{i=[1,p]}$ can be obtained by the NLS method by finding the optimal variable $\mathbf{z}^* = \left[\mathrm{vec}\left(\mathbf{B}^*\right); \left(\mathbf{c}^*\right)^T\right] \in \mathbb{R}^{R(I+1)}$. The NLS method minimizes the squared error between the ground truth output $\mathbf{y}$ and the simulated output determined by $\mathbf{z}$ and $\mathbf{U}$ following Equation 2-14. The objective function of the NLS method can be expressed mathematically as,

$$\min_{\mathbf{z}} f(\mathbf{z}) \quad \text{with} \quad f(\mathbf{z}) \in \mathbb{R} = \frac{1}{2}\|\mathbf{r}(\mathbf{z})\|_F^2, \quad \mathbf{r}(\mathbf{z}) \in \mathbb{R}^N = (\mathbf{UB})^{\cdot d} \cdot \mathbf{c} - \mathbf{y} \qquad (2\text{-}15)$$

where $\|\cdot\|_F^2$ denotes the squared Frobenius norm calculated as:

$$\|\mathbf{r}(\mathbf{z})\|_F^2 = \sum_{n=1}^{N} |r(z)^{(n)}|^2 \tag{2-16}$$

where the scalar $r(z)^{(n)}$ denotes the n^{th} element (row) of the residual vector $\mathbf{r}(\mathbf{z})$.

Line search and trust region algorithms are two iterative methods for updating the variables of optimization problem in Equation 2-15 such that $\mathbf{z}_k$ is closer to a (possibly local) optimum. Line search methods use the objective function to generate a search direction $\mathbf{p}_k$, and then concentrate their efforts on finding a step length λ_k that is optimal for this direction, expressed as follows,

$$\begin{aligned} \mathbf{z}_k &= \mathbf{z}_{k-1} + \lambda_k \mathbf{p}_k \\ \lambda_k &= \arg\min_{\lambda>0} f\left(\mathbf{z}_{k-1} + \lambda \mathbf{p}_k\right) \end{aligned} \tag{2-17}$$

where $\mathbf{p}_k$ can be chosen by the steepest descent method. The steepest descent method is not used in this thesis because its convergence has been proven to be slow and not monotone [31].

Trust-region approaches identify an area Δ_k around the current iteration where they trust the model $\tilde{f}_k$ is a sufficient approximation of the objective function f_k and then decide the step $\mathbf{p}_k$ as the approximate minimizer of the model $\tilde{f}_k$ in this region, expressed as

$$\begin{aligned} \mathbf{p}_k &= \arg\min_{\mathbf{p}_k} \tilde{f}(\mathbf{p}_k) \quad \text{subject to} \quad \|\mathbf{p}_k\|^2 \leq \Delta_k, \\ \mathbf{z}_k &= \mathbf{z}_{k-1} + \mathbf{p}_k, \end{aligned} \tag{2-18}$$

where $\|.\|^2$ is the squared Euclidean norm.

The size of the trust region Δ_k is critical to the effectiveness of each step $\mathbf{p}_k$. If the region is too small, the algorithm misses an opportunity to take a substantial step that will move it much closer to the minimizer of the objective function. If too large, the minimizer of the model may be far from the minimizer of the objective function in the region, so we may have to reduce the size of the region and try again [31]. In this thesis, the size of the region is determined by the performance of the algorithm during previous iterations. The reader is recommend to reference the Algorithm 4.1 in [31] where a detailed derivation of the trust region calculation is provided and used in this thesis.

The step direction $\mathbf{p}_k$ in Equation 2-18 can be determined by the derivatives $\mathbf{g}_k$ and Hessian $\mathbf{H}_k$ of the objective function f_k locally approximated by a Taylor series at the current guess $\mathbf{z}_k$:

$$f\left(\mathbf{z}_k + \mathbf{p}\right) \approx f\left(\mathbf{z}_k\right) + \mathbf{p}^{\mathrm{T}} \cdot \underbrace{\nabla_{\mathbf{z}} f\left(\mathbf{z}_k\right)}_{\mathbf{g}_k} + \frac{1}{2}\mathbf{p}^{\mathrm{T}} \cdot \underbrace{\nabla_{\mathbf{z}}^2 f\left(\mathbf{z}_k\right)}_{\mathbf{H}_k} \cdot \mathbf{p} + \ldots \tag{2-19}$$

where $\mathbf{g}_k$ and $\mathbf{H}_k$ are calculated by the (partial) derivative as follows,

$$\mathbf{g}_k = \begin{bmatrix} \frac{\partial f}{\partial z_k^{(1)}} & \frac{\partial f}{\partial z_k^{(2)}} & \cdots & \frac{\partial f}{\partial z_k^{(R(I+1))}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{R(I+1)} \tag{2-20}$$

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial^2 f}{\partial (z_k^{(1)})^2} & \frac{\partial^2 f}{\partial z_k^{(1)} \partial z_k^{(2)}} & \cdots & \frac{\partial^2 f}{\partial z_k^{(1)} \partial z_k^{(R(I+1))}} \\ \frac{\partial^2 f}{\partial z_k^{(2)} \partial z_k^{(1)}} & \frac{\partial^2 f}{\partial (z_k^{(2)})^2} & \cdots & \frac{\partial^2 f}{\partial z_k^{(2)} \partial z_k^{(R(I+1))}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial z_k^{(R(I+1))} \partial z_k^{(1)}} & \frac{\partial^2 f}{\partial z_k^{(R(I+1))} \partial z_k^{(2)}} & \cdots & \frac{\partial^2 f}{\partial (z_k^{(R(I+1))})^2} \end{bmatrix} \in \mathbb{R}^{R(I+1) \times R(I+1)} \tag{2-21}$$

where $z_k^{(i)}$ represents the i^{th} element of the optimization variable $\mathbf{z}_k$ at iteration k.

The approximation model $\tilde{f}_k$ can be obtained by limiting the Taylor series in Equation 2-19 to the first three elements, expressed as

$$\widetilde{f}(\mathbf{p}) = f(\mathbf{z}_k) + \mathbf{p}^{\mathrm{T}} \mathbf{g}_k + \frac{1}{2} \mathbf{p}^{\mathrm{T}} \mathbf{H}_k \mathbf{p} \tag{2-22}$$

To obtain the optimal step direction $\mathbf{p}_k$, the gradient of $\widetilde{f}$, i.e., $\nabla_{\mathbf{p}} \widetilde{f} = \mathbf{g}_k + \mathbf{H}_k \mathbf{p}$, is set to zero, which results in the standard Newton's method:

$$\mathbf{H}_k \mathbf{p}_k = -\mathbf{g}_k. \tag{2-23}$$

where $\mathbf{p}_k$ is called the Newton step.

Equation 2-23 can be solved by calculating the inverse of the Hessian $\mathbf{H}_k$. However, this is ill-conditioned and expensive because of following two reasons:

1. The Hessian $\mathbf{H}_k$ is often difficult or expensive to compute explicitly. Each element of $\mathbf{H}_k$ requires a twice partial derivative evaluation of the cost function. For large I and R, the computational cost to calculate the twice partial derivations is high.
2. Even though the Hessian $\mathbf{H}_k$ could be calculated explicitly, $\mathbf{H}_k$ is not promised to be positive-definite, and thus it could not be invertible. For example, $\mathbf{H}_k$ could contain negative or zero eigenvalue(s) for some value combinations of $\mathbf{B}$ and $\mathbf{c}$.

Instead of calculating the inverse of Hessian $\mathbf{H}_k$, $\mathbf{H}_k$ can be approximated by several Newton type algorithms, such as nonlinear conjugate gradient , Gauss-Newton (GN) and Levenberg-Marquardt algorithms [25]. Some examples of such approximations of these algorithms are given in Appendix B-1. The GN algorithm is used in this thesis to find the step direction $\mathbf{p}_k$, which is expressed as follows,

$$\mathbf{J}_k^T \mathbf{J}_k \mathbf{p}_k = -\mathbf{J}_k^T \mathbf{r}_k \tag{2-24}$$

where $\mathbf{J}_k \in \mathbb{R}^{R(I+1)}$ is the Jacobian of the cost function, and the Newton step $\mathbf{p}_k$ can be solved exactly using the pseudoinverse of $\mathbf{J}_k^T \mathbf{J}_k$ or inexactly using conjugate gradient (CG) iterative technique. The reader is recommended to reference [31] for a detailed description of CG technique.

The advantages of using the GN algorithm to optimize the NLS objective function in Equation 2-15 are the motivation of using the GN algorithm, which are listed as follows,

1. The approximation of Hessian $\mathbf{H}_k$ by the Jacobian $\mathbf{J}_k$ saves the trouble of computing the individual element of the $\mathbf{H}_k$.

2. The multilinear information of tensor operations in the LS-CPD format can be used elegantly to calculate of the Jacobian $\mathbf{J}_k$, which is shown in the next subsection.

3. The convergence of the GN algorithm is quadratic when near to the local optimum. This property has been proven in Theorem 10.1 in [31].

4. Several recent research have demonstrated that the GN algorithm allows the constraints, symmetry, coupling and regularization to be implemented easily on the LS-CPD format [32, 33, 34].

The procedure of the GN algorithm to solve the NLS objective function will be explained in the next coming subsection.

2-4-2 GN optimization algorithm

The GN algorithm solves the NLS problem in Equation 2-15 by approximating the Hessian $\mathbf{H}_k$ at iteration k and then update the variable $\mathbf{z}_k$ along the direction $\mathbf{p}_k$ by solving the linear system in Equation 2-23, whose components are obtained in the following manners. For simplicity, the underscript k is dropped.

- Jacobian $\mathbf{J}$ can be partitioned into two parts $[\mathbf{J_B}, \mathbf{J_c}]$ corresponding to the variables $\mathbf{B}$ and $\mathbf{c}$ with:

$$\begin{aligned} \mathbf{J_B} &= \frac{\partial \mathbf{r}}{\partial \operatorname{vec}(\mathbf{B})} = \left(d(\mathbf{UB})^{\cdot(d-1)} \odot \mathbf{c}^{\mathrm{T}}\right) \odot^{\mathrm{T}} \mathbf{U} \in \mathbb{R}^{N \times RI} \\ \mathbf{J_c} &= \frac{\partial \mathbf{r}}{\partial \mathbf{c}} = (\mathbf{UB})^{\cdot d} \in \mathbb{R}^{N \times R} \end{aligned} \tag{2-25}$$

  whose computational complexity together are $\mathcal{O}\left(NR(I+d)\right)$. The reader is recommended to reference [22] for a detailed derivation of Equation 2-25.

- Hessian $\mathbf{H}$ is approximated by

$$\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{J} \in \mathbb{R}^{R(I+1) \times R(I+1)} \tag{2-26}$$

  where $\mathbf{J} \in \mathbb{R}^{N \times R(I+1)}$ is the Jacobian. The computational complexity of Equation 2-26 is $\mathcal{O}\left(NR^2I^2\right)$.

- Gradient $\mathbf{g}$ can be calculated explicitly by

$$\mathbf{g} = \mathbf{J}^{\mathrm{T}}\mathbf{r} \in \mathbb{R}^{R(I+1)} \tag{2-27}$$

  whose computational complexity is $\mathcal{O}\left(NRI\right)$.

The GN algorithm for solving the LS-CPD in Equation 2-14 with trust region method is summarized in Algorithm 1, where a random factor matrix $\mathbf{B}_0$ and weighting vector $\mathbf{c}_0$ can be used to initialize the algorithm. It can also start from the output of an algebraic framework proposed in [22] that is proven to provide a better initialization and explained in Appendix section B-2. The algorithm is available in the MATLAB ® script *lscpds_krt_nls.m* in the

Tensorlab package [35].

Algorithm 1: GN algorithm with trust region method to solve LS-CPD in Equation 2-14 [22]

Input: $\mathbf{U}$, $\mathbf{y}$, initial values for $\mathbf{B}_0$, $\mathbf{c}_0$

while *not converged* **do**
  Compute Jacobian $\mathbf{J}_k$ *and gradient* $\mathbf{g}_k$;
  Approximate the Hessian $\mathbf{H}_k$ *by* $\mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k$;
  Find the Newton step $\mathbf{p}_k$ *by solving* $\mathbf{H}_k\mathbf{p}_k = -\mathbf{g}_k$ *using pseudoinverse or CG* ;
  Update the variable $\mathbf{z}_k$ *along the Newton step* $\mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{p}_k$;
end

Output: $\mathbf{B}^*$, $\mathbf{c}^*$

The stopping criteria for Algorithm 1 is the fulfillment of any of the following conditions.

- The value of the objective function is lower than the tolerance with the default value as 10^{-12}. [1]
- The difference in objective function value between two successive iterations is lower than the tolerance with the default value as 10^{-12}.
- The number of iterations reach the maximum default value of 200.
- The value of the Newton step size reaches the tolerance with the default value as 10^{-6}.

If Equation 2-23 is solved by pseudoinverse of the Hessian $\mathbf{H}_k$, the computational complexity of Algorithm 1 is $\mathcal{O}\left(it_{GN}(NR^2I^2 + R^3I^3)\right)$ with it_{GN} stands for the number of GN iteration [22]. In each iteration, Algorithm 1 updates the values for the following components, $\mathbf{H}, \mathbf{J}, \mathbf{g}, \mathbf{r}, \mathbf{z}$ and $\mathbf{p}$. The storage complexity of Algorithm 1 is then the summation of the size of all these components and of the inputs $\mathbf{y}$ and $\mathbf{U}$. The result is $\mathcal{O}\left(R(I+1)(N+3) + N(I+2) + R^2(I+1)^2\right)$.

2-5 Conclusion On Phase I

In this chapter, the Volterra system identification problem is modelled within the LS-CPD framework introduced in Equation 2-14. The parameters values of this framework can be optimized by the NLS optimization problem which is solved with the GN algorithm with the trust region method explained in Algorithm 1.

Based on the content discussed this chapter, the research sub-questions related to Phase I are answered as follows.

[1] The default values in each condition are chosen to achieve a good balance between the quality of the final optimal solution and the extra unnecessary computational cost when the solution is already near to the global (local) minimum.

Phase I: Development

1. *How does the Volterra-PARAFAC model address the problem of parametric complexity in identifying the Volterra kernel parameters? If it does not solve the problem, what is the lacking factor?*

   (a) Firstly, it should be recalled that in this thesis, the Volterra-PARAFAC model is equivalent to the symmetric CPD of Volterra tensor defined in Equation 2-13. The symmetric CPD technology decomposes the Volterra tensor as a sum of R rank-1 terms. These terms are the same for each rank value, and are determined by the factor matrix and the weighting vector. The storage complexity of these two components grows linearly with the rank value and the dimension size of the Volterra tensor.

   (b) The effect of the symmetric CPD on lowering the parametric complexity is less attractive and even becomes negative, when with small order and dimension size but large rank values. Meanwhile, determining the rank R of the Volterra tensor is a NP-hard problem [9]. In many cases, the domain knowledge or trial-and-error are used to find the correct rank value. Because the rank R is a hyper-parameter in this proposed framework, it is crucial to investigate the impact of various rank values on the same identification configuration.

2. *What is(are) the benefit(s) of using the Gauss-Newton (GN) optimizer to optimize the Least squares (LS) cost function? How to calculate the components of this optimizer?*

   (a) The main advantage of the Gauss-Newton (GN) optimizer is that the calculationally expensive Hessian term of the cost function is approximated by the Jacobian term. The other benefits are: it leverages the multilinear information of the tensor product operation to compute the Jacobian, achieves quadratic convergence when near to the local optimum, and allows constraints, symmetry, coupling and regularization to be implemented easily.

   (b) The components for the GN algorithm consist of the Hessian, Jacobian, gradient and Newton step. The calculation of these terms is explained in section 2-4.

3. *The GN optimizer minimises the Least squares (LS) cost function starting from an initial value. How to initialise this optimization procedure?*

   (a) The random initialization and algebraic initialization methods have been introduced in subsection 2-4-2 and Appendix B-2. The former initializes the optimization variable from random values, while the latter initializes from the result of some algebraic operations, which could be already close to the optimal value.

4. *Does the developed framework apply to the MIMO case? If not, which step (component) of the developed framework should be modified in order to do so?*

(a) It does not apply to the MIMO case because the developed framework only updates a pair of factor matrix and weighting vector. For the MIMO system, each output corresponds to a Volterra tensor. The symmetric CPD should be applied to each Volterra tensor, respectively. This results in each output corresponds to a pair of factor matrix and weighting vector.

(b) Three components should be modified for the MIMO case. (1) The output measurements are stored in a matrix $\mathbf{Y}$ where each column corresponds to one output. The vector $\mathbf{y}$ in the developed cost function should be changed by the defined matrix $\mathbf{Y}$. (2) By stacking the elements of all the factor matrices and weighting vectors row-wisely in a vector, this vector becomes the new optimization variable for the new cost function, whose value is updated by the GN optimizer. (3) The Jacobian term of the GN optimizer should be calculated in a different manner, which takes all the factor matrices and weighting vectors into consideration.

Chapter 3

Validation: Identifying An Artificial Volterra System

This chapter aims to validate the developed Volterra system identification framework. Section 3-1 covers the simulation of an artificial MISO Volterra system, and definition of the figures of merit used to determine the quality of the identification performance. The artificial MISO Volterra systems with different parameter values are simulated and used in three proof-of-concept experiments. The fist two experiments performed in section 3-2 and section 3-3 are the internal validations. They focus on the influence of the hyper-parameter R and two initialization methods of the GN algorithm on the identification performance. The third experiment performed in section 3-4 is the external validation. It compares the developed framework with three TN based frameworks on identifying the same artificial Volterra system when the storage complexity budget is added. This chapter ends by answering the research sub-questions of Phase II in section 3-5. All computations were done on an Intel i5 quad-core processor running at 2.4 GHz with 8 GB RAM using MATLAB ® 2018a and Tensorlab 3.0 [35].

3-1 Proof-of-concept Experiment Setup

In this chapter, an artificial MISO Volterra system is simulated to demonstrate the validity of the developed identification framework. The default system order, the input memory, the input number, and the simulation time length are set to $d = 5$, $M = 5$, $p = 2$ and $N = 1000$[1], respectively. The factor matrix $\mathbf{B} \in \mathbb{R}^{I \times R}$ in the LS-CPD format in Equation 2-14 is created

[1]The selection of the system default setting is based by balancing the size of the synthetic Volterra tensor with the computational and storage capacity of the computer. If the size of the synthetic Volterra tensor (determined by d and $pM + 1$) is disproportionately large, the framework demands an excessive amount of resources. In this case, a single iteration of the GN algorithm may last a few minutes. Some experiments in this chapter are run thirty trials under same system parameters. Having an excessively large tensor size will add many hours to the completion of these studies.

as the sampled decaying exponential function with each column $i \in [1, R]$ expressed as [9]

$$\mathbf{B}(:, i) = |\alpha_i| \exp\left(-\beta_i[1 : I]\right) \tag{3-1}$$

where α_i is sampled from a standard normal distribution and β_i is an integer sampled uniformly from the range $[1, 10]$. The default rank value is set to $R = 5$. The weighting vector $\mathbf{c} \in \mathbb{R}^R$ contains pseudo-random values drawn from the standard uniform distribution on the open interval (0,1). The input measurements $u_i(t) \in \mathbb{R}$ for $t \in [0, N-1]$ and $i \in [1, p]$ are created as standard normal Gaussian white noise, which guarantees the input matrix $\mathbf{U} \in \mathbb{R}^{N \times I}$ in Equation 2-14 being persistently exciting of any order [9].

The corresponding symmetric Volterra tensor $\mathcal{V}_{sym}$ is recovered by following Definition 3. This can be achieved by using the MATLAB ® script *cpdgen.m* in the Tensorlab package [35]. The *cpdgen.m* function takes the user-defined $\mathbf{B}$ and $\mathbf{c}$ as the input, and returns $\mathcal{V}_{sym}$ as the output. The output measurements $\mathbf{y} \in \mathbb{R}^N$ are simulated within the LS-CPD format introduced in Equation 2-14. The measurement noise is also Gaussian white noise with a variance chosen such that different signal-to-noise ratio (SNR) can be created. The noiseless output is denoted as $\mathbf{y}$, and the output with noise SNR value of x is denoted as $\mathbf{y}(SNR = x)$. The SNR is defined as the ratio of the summed square magnitude of the signal to the summed squared magnitude of the noise, expressed as [18],

$$SNR = 20 \log_{10}\left(\frac{\|\mathbf{y}\|_F}{\|\mathbf{N}\|_F}\right) \tag{3-2}$$

where $\mathbf{N} = \mathbf{y}(SNR = x) - \mathbf{y}$, and $\|\cdot\|_F$ denotes the Frobenius norm.

The first 700 samples of the output measurements are used for training, denoted as $\mathbf{y}_t$, while the remaining 300 samples are used for validation, denoted as $\mathbf{y}_v$. The estimated training output is denoted as $\hat{\mathbf{y}}_t$, and the modelled validation output is denoted as $\hat{\mathbf{y}}_v$. Both of them are calculated within the LS-CPD format along with the input matrix $\mathbf{U}$, the estimated factor matrix $\hat{\mathbf{B}}$ and the weighting vector $\hat{\mathbf{c}}$. The latter two components are the output of Algorithm 1, mathematically expressed as:

$$\begin{aligned} \hat{\mathbf{y}}_t &= \left(\mathbf{U}(1 : 700, :)\hat{\mathbf{B}}\right)^{\cdot d} \hat{\mathbf{c}} \\ \hat{\mathbf{y}}_v &= \left(\mathbf{U}(701 : 1000, :)\hat{\mathbf{B}}\right)^{\cdot d} \hat{\mathbf{c}} \end{aligned} \tag{3-3}$$

where $\mathbf{U}(1 : 700, :)$ stands for the first 700 rows of the matrix $\mathbf{U}$. The estimated Volterra tensor, denoted as $\hat{\mathcal{V}}$, is recovered by the estimated factor matrix $\hat{\mathbf{B}}$ and the weighting vector $\hat{\mathbf{c}}$.

The identification performance is characterized by the following figures of merit. These are adopted from the recent research [9]:

- The relative training and validation error, denoted as r_t and r_v, respectively, are defined as follows,

$$\begin{aligned} r_t &:= \frac{\|\mathbf{y}_t - \hat{\mathbf{y}}_t\|_F}{\|\mathbf{y}_t\|_F} \\ r_v &:= \frac{\|\mathbf{y}_v - \hat{\mathbf{y}}_v\|_F}{\|\mathbf{y}_v\|_F} \end{aligned} \tag{3-4}$$

Table 3-1: Mean and standard deviation (Std) of the values regarding the figure of merits over 20 trails with $R = 5$ when starting with random initialization.

	r_t	r_v	s	$\mathrm{t}_{ir}(s)$	$\mathrm{t}_{gn}(s)$	$\mathrm{t}_t(s)$
Mean	$2.98*10^{-16}$	$1.22*10^{-13}$	$3.45*\ 10^{-19}$	$4*10^{-5}$	$2.86*10^{-2}$	$2.86*10^{-2}$
Std	$1.01*10^{-31}$	$2.29*10^{-17}$	$5.25*10^{-20}$	$1.43*10^{-6}$	$8.54*10^{-4}$	$8.54*10^{-4}$

- The symmetry of the estimated Volterra tensor $\hat{\mathcal{V}}$ is reflected by computing the symmetry coefficient s as follows [9],

$$s := \sum_p \|\hat{\mathcal{V}} - \mathrm{permute}(\hat{\mathcal{V}}, \pi)\|_F, \tag{3-5}$$

  which compares $\hat{\mathcal{V}}$ to all possible index permutations π, and is exactly zero when $\hat{\mathcal{V}}$ is symmetric.

- The running time required for the initialization of Algorithm 1 with the random initialization or the algebraic initialization, are denoted as t_{ir} and t_{ia}, respectively. The running time required from Algorithm 1 to converge is denoted t_{gn}. All the running times are measured by the MATLAB ® script *tic.m* and *toc.m* with unit of second. The total running time for the overall identification procedure, denoted as t_{tr} (random initialization) or t_{ta} (algebraic initialization), is calculated as follows,

$$\begin{aligned} t_{tr} &:= t_{ir} + t_{gn} \\ t_{ta} &:= t_{ia} + t_{gn} \end{aligned} \tag{3-6}$$

3-2 Experiment I: Influence Of Rank R

This experiment aims to (1) investigate the influence of the approximation rank value on the GN algorithm identification performance when the size of the Volterra tensor to be estimated is different; (2) discuss the compensation that choosing an overestimated rank value has on the figures of merit.

In the experiment, the default system parameter values and the noiseless output measurements are used. The random initialization is applied instead of the algebraic initialization because it has been checked that the latter already provides an (close-to-)optimal solution. Having an optimal initial value reduces the evidence of the GN optimization capability, which is not consistent with the aim of this experiment.

This experiment has two steps. First, the default value rank $R = 5$ is chosen, and the GN algorithm is run for twenty trials. Each trial starts with a different random initial value. The corresponding figures of merit of all trials are summarized in Table 3-1. The average number of iterations required for Algorithm 1 to converge is 11 (with deviation of $\pm$ 3). The values of several optimization components over iterations of one trial are displayed in Figure 3-1. In this trial, the second stopping criteria (the relative objective function value reaches

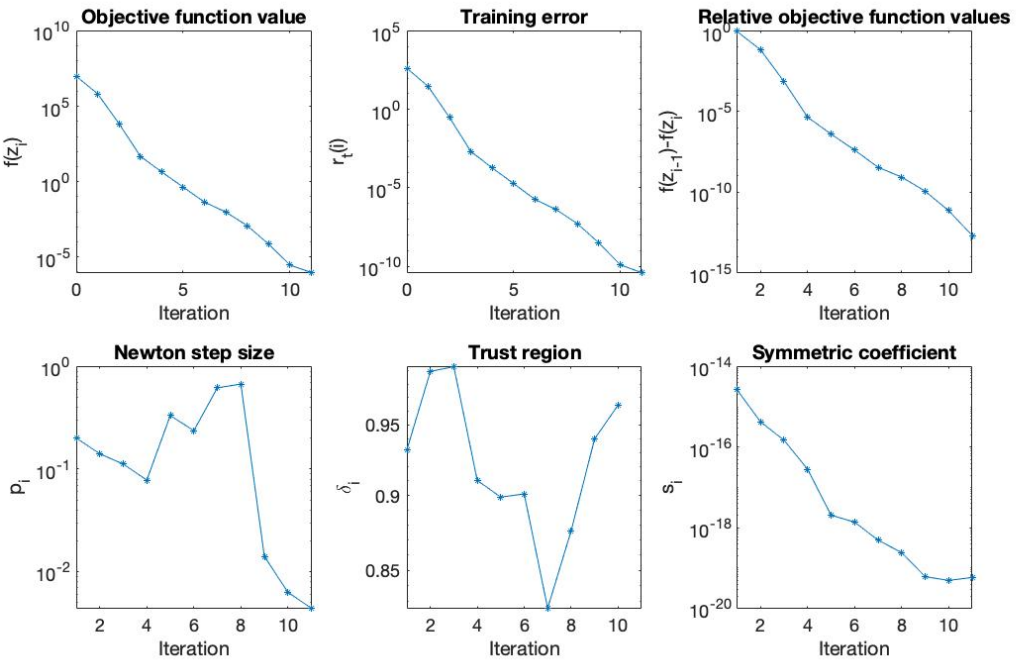


Figure 3-1: Evolution of the GN components' values over iterations in one trial with $R = 5$ and random initialization.

the minimum tolerance) is satisfied at the 11^{th} iteration when the training error r_t drops to 10^{-10}. The optimization variable $\mathbf{z}_i$ at the 8^{th} iteration may be already close to the optimal value, given the fact that both the Newton step size $\mathbf{p}_i$ and the relative objective function value $f(\mathbf{z}_{i-1}) - f(\mathbf{z}_i)$ keep as small value after 8^{th} iteration. Meanwhile, the estimated tensor is symmetric over all iterations, given the fact that the symmetric coefficient s_i always stays smaller than 10^{-14}. This is expected since the symmetric structure of the resultant tensor is enforced by the LS-CPD format.

The above experiment proves that the proposed framework is able to identify the artificial Volterra system with the default rank value. However, in practical applications, the number of input p, memory M, and dimension I of the tensor to be estimated are all or partially unknown.

The second step of the experiment is to compare the identification performances with various rank values $R \in [1, 2, 3, 4, 5, 6, 10, 20]$ and tensor dimensions $I \in [5, 10, 15, 20]$. For each value combination of I and R, the GN algorithm is run for thirty trials with random initialization. The corresponding figures of merit are displayed as a box-plot shown in Figure 3-2.

The black circle in the box-plot is the median value, representing the average level of the data. The length of the box represents the variance. The upper and lower bounds of the bins are the upper and lower quartiles of the data, respectively. This means that the bin contains 50% of the data. Therefore, the width of the bins reflects, to some extent, the degree of fluctuation in the data. The line above and below the box represents the maximum and

minimum value, respectively. The outlier data are plotted as small boxes outside these two lines.

In general, the identification performance is sensitive to the choice of R, especially with small tensor dimension. The sensitivity can be explained by the two following observations from Figure 3-2.

1. The training and validation error increase as the approximation rank values decreases when the approximation rank is under-estimated ($R < 5$). Both errors drop significantly when the approximation rank is greater than or equal to the true rank value $R = 5$. This implies that the GN algorithm does not converge to the local(global) minimum when the rank is under-estimated. This is more evident as I increases.

2. The training run time increases with I, which is expected due to the fact that the size of the optimization variable $\mathbf{z} \in \mathbb{R}^{R(I+1)}$, the computational complexity and the iterations of the GN algorithm, all increase with growing R and I. However, the running time unexpectedly drops to the shortest when R equals to the true rank value. This sudden drop is insignificant as I increases.

In conclusion, the good fit between the model estimated by the GN algorithm and the ground truth model is reached when the approximation rank is greater than or equal to the true rank. The approximation rank value R can be chosen in such a way that a trade-off between the computational (storage) complexity and the validation error is satisfied. For the trial-and-error approach, a significant drop in the training run time when the rank starts from a small number, might indicate that the current rank value is chosen sufficiently. When the dimension of the tensor to be estimated is large, the accuracy of this indication decreases. The method to find an appropriate rank value of the proposed framework for different tensor dimensions requires further research.

3-3 Experiment II: Choice Of Initialization Method

In the previous experiment, the simulation results summarised in Table 3-1 prove that the GN algorithm is able to converge to a local(global) minimum value when starting from a random value. The algebraic initialisation method explained in Append A-1 can offer a more accurate initial value for the GN algorithm [22].

A preparation experiment is designed to prove the effectiveness of the algebraic initialisation method. In the preparation experiment, two initialization methods are used for the GN algorithm to identify the default artificial system, respectively. Twenty trials are performed for the random initialization, and one trial for the algebraic initialization. The identification performances in terms of figures of merit of two initialization methods are compared in Table 3-2. The values of the components of the GN algorithm of two methods are also compared and shown in Figure 3-3, where one trial of the random initialization method is plotted.

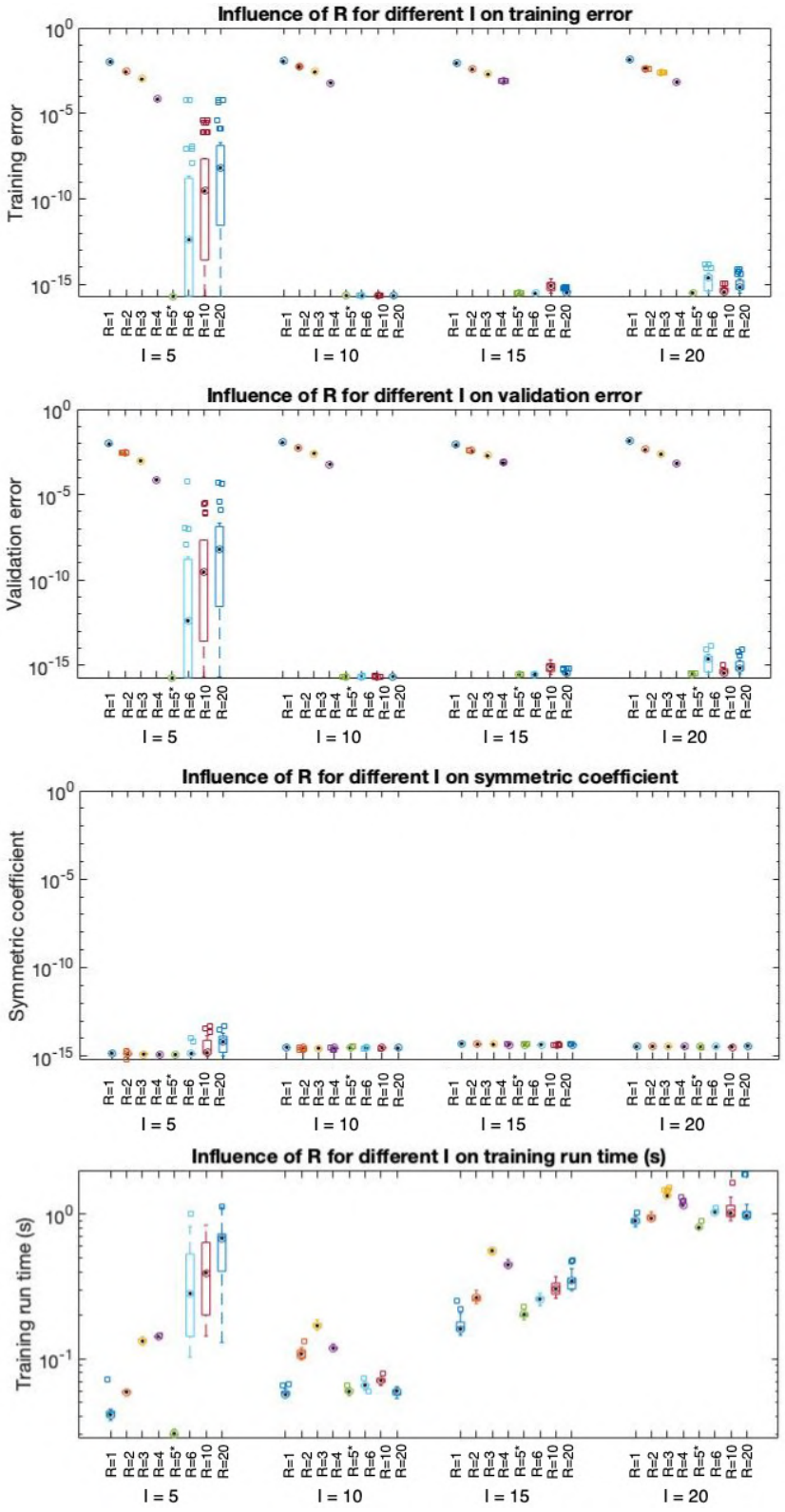


Figure 3-2: Influence of $R \in [1, 2, 3, 4, 5, 6, 10, 20]$ with increasing tensor dimension $I \in [5, 10, 15, 20]$ on training error (first plot), validation error (second plot), symmetric coefficient (third plot), and training run time (last plot). The results are from 30 trials of the GN algorithm with random initialization for each value combination of R and I.

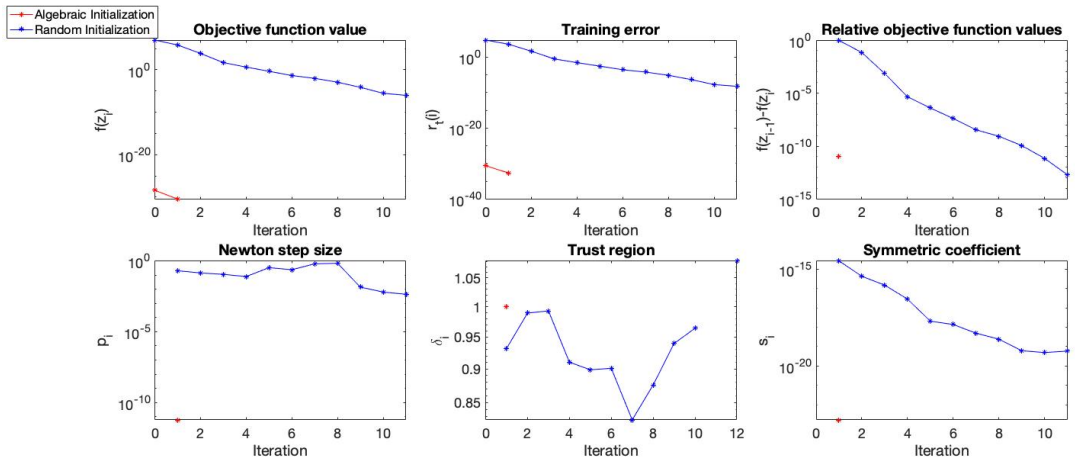


Figure 3-3: Evolution of the GN components' values over iterations in one trial with random (blue) or algebraic initialization (red) under the default system configuration.

Based on the results in Table 3-2 and Figure 3-3, the effectiveness of the algebraic initialization method comes at the expense of a long running time. This conclusion is provided based on the following analysis of the results:

1. In Figure 3-3, the objective function value (training error) at the zero iteration for the algebraic initialization method is 10^{-30} times smaller than that of the random method. This implies that the initial value provided by the algebraic initialization method is much closer to the ground truth than the random value.

2. In Table 3-2, the validation error with the algebraic initialization method is significantly lower (10^{-16} times less) than the random initialization. Meanwhile, the symmetric coefficient of the estimated Volterra tensor with the algebraic initialization method is also considerably lower (10^{-8} times less) than the random initialization.

3. In Table 3-2, it only takes 1 iteration for the GN algorithm to converge with the algebraic initialization method, while 11 iterations in average for the random initialization. In Figure 3-3, the first, second and fourth stopping criteria are met simultaneously with the algebraic initialization method after the 1^{st} iteration.

4. In Table 3-2, the running time for the algebraic initialization method is 10^5 times more than that of the random method. Meanwhile, when using the algebraic initialization, the total running time is dominated by the time consumed for initialization.

The longer total running time for the algebraic method results from its expensive computational and storage complexities. The computational and storage complexities of the algebraic initialization method are $\mathcal{O}\left(\frac{1}{2}NIF^2 + \frac{7}{6}F^3 + NF\right)$ and $\mathcal{O}\left(FI^d\right)$, respectively. The value of F equals $\begin{pmatrix} I-1+d \\ I-1 \end{pmatrix}$. The detailed derivations of the computational and storage complexities are given in Appendix B-2. The computational and storage complexities of the algebraic initialization depend on three system parameters: I, d and N. It is necessary to investigate

Table 3-2: The values of the figures of merit on identifying the default system with two initialization methods. Twenty trials are performed for the random initialization, and one trial for the algebraic initialization.

Element	Random (mean+Std)	Algebraic
r_t	$2.98*10^{-16}(1.01*10^{-31})$	$5.64*10^{-32}$
r_v	$1.22*10^{-13}(2.29*10^{-17})$	$6.23*10^{-29}$
s	$3.45*10^{-19}(5.25*10^{-20})$	$2.45*10^{-27}$
$t_{ir}(s)$	$4*10^{-5}(1.43*10^{-6})$	3.2
$t_{gn}(s)$	$2.86*10^{-2}(8.54*10^{-4})$	$4.32*10^{-4}$
$t_t(s)$	$2.86*10^{-2}(8.54*10^{-4})$	3.2
it_{gn}	11 (3)	1

under which situation the algebraic initialization is worth using. The purpose of this investigation is to optimize the overall computational and storage complexity of the framework, which is the summation of the computational and storage complexity of the initialization method and the GN algorithm.

The experiment in this section is designed with the aim of achieving this purpose by: (1) defining additional figures of merit to numerically reflect the balance between advantages and disadvantages of the algebraic initialization method; (2) recording the figures of merit of using the algebraic method for the GN algorithm to identify the artificial Volterra systems with different parameter values.

This experiment has two steps. Each step starts with the definition of the related figure of merit, followed by the simulation results and discussions. Each following subsection covers one step.

3-3-1 Complexity compensation rates

The figures of merit used in the first step are the rates between the computational and storage complexity of the algebraic initialization method and the GN algorithm. The complexity compensation rates (CCR) are defined to calculate the rates as expressed in Equation 3-7. These include two elements regarding the computational complexity denoted as CCR_c, and the storage complexity denoted as CCR_s, expressed respectively as follows,

$$\begin{aligned} CCR_c(\%) &:= 100\frac{|C_{gn} - C_a|}{C_{gn}} \\ CCR_s(\%) &:= 100\frac{|S_{gn} - S_a|}{S_{gn}} \end{aligned} \tag{3-7}$$

where C_{gn} and C_a (S_{gn} and S_a) stands for the computational (storage) complexities of the GN algorithm and the algebraic initialization method, respectively. The value of each term

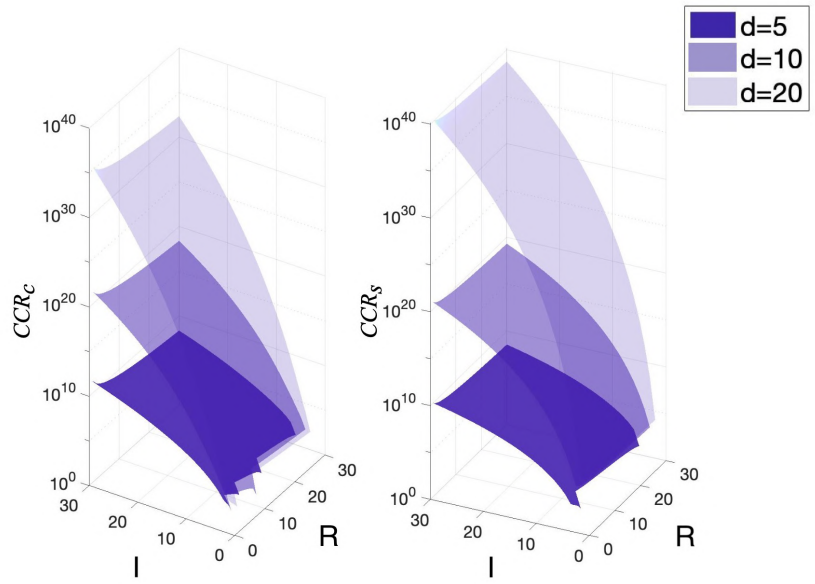


Figure 3-4: Growths of CCR_c (left) and CCR_s (right) for increasing I and R for three cases when $d = 5, 10, 20$. Both CCR_c and CCR_s grows exponentially with increasing I, and stay constant with increasing R. The growths are dominated by the value of d, while the influence of R is much weaker.

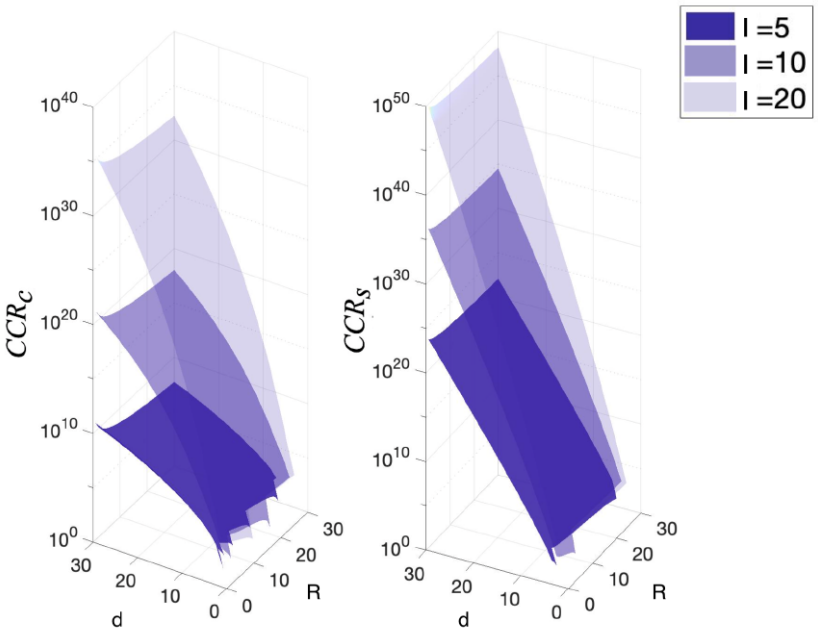


Figure 3-5: Growths of CCR_c (left) and CCR_s (right) for increasing d and R for three cases when $I = 5, 10, 20$. Both CCR_c and CCR_s grows exponentially with increasing d, and stay constant with increasing R. The growths are dominated by the value of I, while the influence of R is much weaker.

is recalled as follows,

$$\begin{aligned} C_{gn} &= \mathcal{O}\left(it_{GN}(NR^2I^2 + R^3I^3)\right) \\ C_a &= \mathcal{O}\left(\frac{1}{2}NIF^2 + \frac{7}{6}F^3 + NF\right) \\ S_{gn} &= \mathcal{O}\left(R(I+1)(N+3) + N(I+2) + R^2(I+1)^2\right) \\ S_a &= \mathcal{O}\left(FI^d\right) \end{aligned} \tag{3-8}$$

where an empirical value for $it_{GN} = 5$ is used. The values of CCR_c and CCR_s both depend on d, R, I and N. In this thesis, the influence of d, R and I is analyzed because these are usually unknown in practice.

The values of CCR_c and CCR_s under different value combinations of d, R and I are visualized Figure 3-4 (different R and I) and Figure 3-5 (different R and d). Following conclusions can be made by analysing the growth of CCR_c and CCR_s in these visualizations:

1. For large Volterra tensor that has higher values of d and I, C_a and S_a are both much higher than C_{gn} and S_{gn}. In this case, the algebraic initialization method is not recommended since it uses excessive resources in comparison to the GN algorithm, especially for the consideration of optimizing the overall computational and storage cost.
2. Although the rank value could influence the identification performance when starting with the random initialization as discussed in section 3-2, it does not influence CCR_c and CCR_s. This is expected because the rank value does not influence C_a and S_a.
3. The growth of CCR_s has the same pattern as CCR_c, but is relatively faster, especially for large Volterra tensor.

3-3-2 Performance compensation rate

The identification performances should also be taken into consideration. As shown in Figure 3-3, the advantage of the algebraic initialization method is that its validation error, denoted as r_{va}, is much smaller than that of the random initialization, denoted as r_{vr}. But its disadvantage is that the total running time t_{ta} is longer than that of the random initialization t_{tr}. The trade-off between its advantage and disadvantage should be made. The performance compensation rate PCR reflects the balance between the validation error reduction, denoted as PCR_r, and the total running time rise, denoted as PCR_t, of two initialization methods. The PCR is defined as follows,

$$\begin{aligned} PCR(\%) :&= 100\frac{PCR_r + PCR_t}{PCR_r} \\ &= 100\frac{log_{10}^{\frac{r_{vr}}{r_{va}}} + log_{10}^{\frac{t_{tr}}{t_{ta}}}}{log_{10}^{\frac{r_{vr}}{r_{va}}}} \end{aligned} \tag{3-9}$$

where $PCR_r > 0$ and $PCR_t < 0$. The logarithm operation is used to calculate the order of the reduction and growth of the PCR_r and PCR_t, respectively.

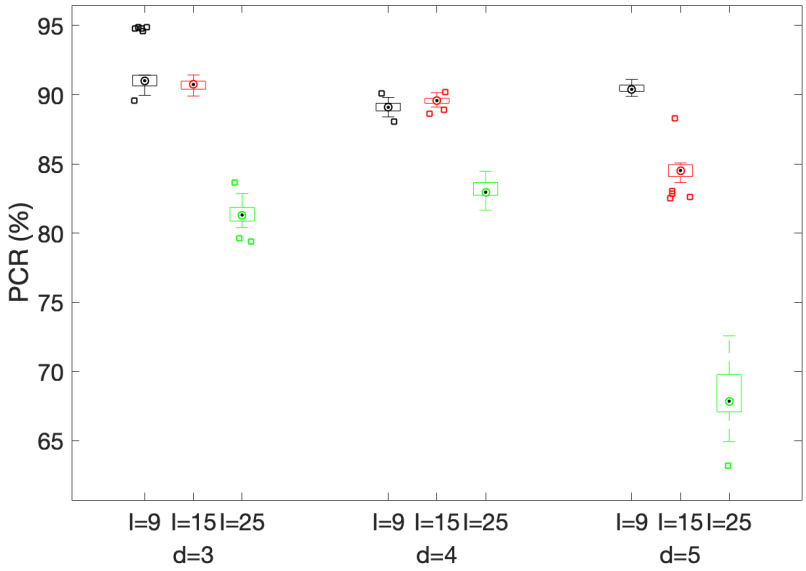


Figure 3-6: Distribution of the PCR values under different sizes of the Volterra tensor to be identified. For each tensor size, the random initialization method is performed 30 trails, and 1 trail for the algebraic initialization method.

The value of PCR is expected to be positive. The larger it is, the better balance between the advantage and disadvantage of the algebraic initialization method can be reached, and the more worthwhile it is to use this method. For the simulation results in Table 3-2, the average value of PCR is calculated as follows,

$$PCR(\%) = 100\frac{log_{10}^{\frac{1.22*10^{-13}}{6.23*10^{-29}}} + log_{10}^{\frac{2.86*10^{-2}}{3.2}}}{log_{10}^{\frac{1.22*10^{-13}}{6.23*10^{-29}}}} \approx 100\frac{16-2}{16} = 87.5 \tag{3-10}$$

which implies that the algebraic initialization method is preferred over the random initialization method when identifying the default artificial system .

The PCR values are recorded when identifying the artificial systems with different value combinations of I and d. Each value combination corresponds to an unique size of the Volterra tensor to be estimated. For each tensor size, the random initialization method is performed for thirty trails, and 1 trail for the algebraic initialization method. This results in the number of PCR for each tensor size being 30. The distribution of PCRs for different tensor sizes is visualized in Figure 3-6. The average value and the variance of PCRs decrease with increasing I and a constant d value. The decrease is more significant for large value of d. This implies that the advantage of the algebraic method is partially countered by its disadvantage when the size of the tensor to be estimated is large.

In conclusion, the algebraic initialization method can provide a better initial value for GN algorithm, but such gain is offset by its large computational and storage complexity, and long running time. The increase of the d and I of the Volterra tensor to be estimated results

Table 3-3: The storage complexities of three TN based and the proposed frameworks. The name of the algorithm that estimates the parameter value is used to represent the corresponding framework. The value of R in the first three algorithms represents the maximal TT rank value, which implies that the expressions in this table for the first three frameworks are over-estimated in favor of the simplification. The value of R for the GN algorithm represents the approximation rank value.

Algorithm	**Storage Complexity**
MPP	$\mathcal{O}\left(N + IR(N+1) + 2(d-1)R^2I + NR + R^2(I+1)\right)$
ALS	$\mathcal{O}\left(N + R^2(NI + 2I + 1)\right)$
MALS	$\mathcal{O}\left(N + R^2I^2(I+1) + 2R^2I\right)$
GN (proposed)	$\mathcal{O}\left(R(I+1)(N+3) + N(I+2) + R^2(I+1)^2\right)$

in the effect of such offset being more significant. This implies that the algebraic initialization method is more expensive and thus less attractive to use when identifying a larger Volterra tensor. The decision to use the algebraic initialization method depends on the total computational and storage complexity budget for the proposed identification framework.

3-4 Experiment III: Comparison With State-of-the-art

In Appendix A-2, three TN-based identification frameworks are reviewed. The Volterra tensor is decomposed into TN format in these frameworks, and the parameter values of the corresponding TN format are estimated by the alternating linear scheme (ALS) [11], modified alternating linear scheme (MALS) [11] and Moore-Penrose pseudoinverse (MPP) algorithms [9], respectively. The storage complexities of these three frameworks that are explicitly explained in Appendix A-2, and together with the storage complexity of the GN algorithm that has been introduced in subsection 2-4-2, are recalled in Table 3-3.

The four frameworks in Table 3-3 have been proven to be effective in solving the MISO Volterra system identification problem without suffering from the curse of dimensionality in [11, 9] and this thesis, respectively. The tensor decomposition techniques and parameter value estimation algorithms used in the four frameworks are different. It is not "fair" to compare the identification performances of these frameworks on the same Volterra system directly. The reasons are as follows,

1. As evident in Table 3-3, the storage complexity of each method is different, even if they may have the same identification performance on the same system. A framework that can estimate the parameters accurately but requires a large storage complexity is not recommended.
2. The TT rank values of the first three frameworks can either be chosen such that the condition of input signal being persistently exciting is satisfied, or being adapted during each iteration. However, there is no established algorithm to determine a proper rank value for the symmetric CPD. The performance of the proposed framework is sensitive to the value of rank value, as proven in section 3-2.

The aims of this experiment are (1) designing a relatively fair experiment set up to compare the performances of these four frameworks such that the above reasons can be taken into consideration; (2) Discussing the advantages and disadvantages of the proposed framework based on the comparison results.

The first aim is achieved by adding the same storage complexity threshold for each framework. The maximum rank value that satisfies the threshold is used in each framework to identify the same Volterra system. The artificial Volterra system with the default parameter values introduced in section 3-1 is used as the benchmark. Four different output noise values are chosen, SNR $\in [-20, 20, 40, +\infty]$. The threshold is chosen as 10^5 bits. The storage complexity growth with increasing R for each framework and the threshold are shown in Figure 3-7, where the maximum rank value for ALS=4, MALS=2, MPP=12 and GN=11.

The performance is evaluated by four figures of merit: training error r_t, validation error r_v, total running time tr (the random initialization method is used for GN), and symmetric coefficient s. The first 700 samples are used for training, and the last 300 samples for validation. For each SNR value, thirty trials are performed for each framework. The results are visualized by the box-plot in Figure 3-8.

Several conclusions can be drawn as follows by analysing Figure 3-8,

1. For noiseless case (SNR $= +\infty$), the TN based frameworks are all outperformed by the proposed method, given that the r_t and r_v of the TN based framework are much higher than that of the proposed method. The ALS and MALS framework even fail to find the symmetric result, given that the s of these two framework are close to 1.
2. The performance of the proposed framework is sensitive to the initial value, given that the length of its bins in each sub-plot is long, especially for t_r. This implies that the iteration number of the GN algorithm highly depends on the quality of the initial values.
3. All the frameworks fail to identify the system when the noise is added (SNR=40,20,-20). This implies that the robustness against the noise of discussed frameworks should be improved.

The reason for the first conclusion is that the maximum rank values under the storage complexity threshold for the TN based frameworks are all lower than the maximum of the suggested TT rank values. The suggested rank values are the least rank values for (1) identifying the unique parameters in the symmetric Volterra tensor and (2) satisfying the input signal being persistently exciting. Having lower rank values could result in the identification of the unique parameters being inadequate. The (maximum) suggested rank value and the corresponding storage complexity of each framework are summarized in Table 3-4. The reader is suggested to reference [11, 9] for detailed explanations of how the suggested rank values for the TN based frameworks are determined.

When the suggested rank values are used for the TN based frameworks, the performances are improved significantly as shown in Figure 3-9. Two conclusions are drawn by analysing Figure 3-9 as follows,

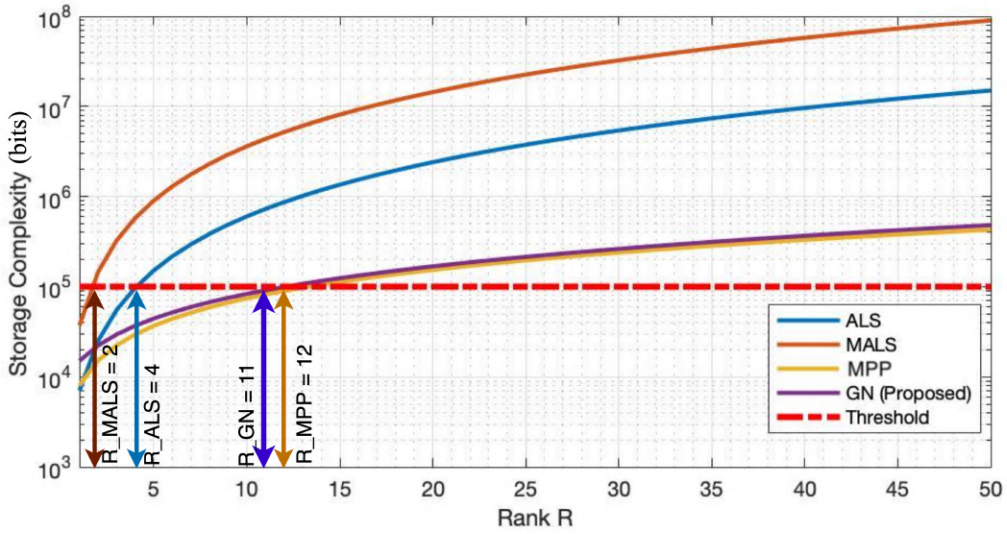


Figure 3-7: The maximum rank value for four frameworks named by corresponding parameter estimation algorithm when the storage complexity threshold of 10^5 bits is enabled.

Table 3-4: The suggested rank values and the corresponding storage complexity for the four frameworks.

Algorithm	ALS	MALS	MPP	GN
Suggested Rank Values (maximum)	6	6	56	5
Storage Complexity (bits)	$6.5 * 10^5$	$2.6 * 10^6$	$4.1 * 10^5$	$4.4 * 10^4$

1. For the noiseless case (SNR $= +\infty$), the average values of r_t, r_v and s for TN based frameworks are all much lower than those in Figure 3-8. The improvement can also be seen for the GN algorithm. With the ground truth rank value $R = 5$, the influence of the random initialization on all the figures of merit decrease, given that the lengths of its bins in four sub-plots are short than those in Figure 3-8. The non-iterative MPP algorithm requires the least amount of running time compared to the other three iterative methods.

2. For the cases with noise (SNR $= -20, 20, 40$), the symmetric solutions can be found in four frameworks, given that the values of s are all lower than 10^{-10}. However, the symmetric solutions are not accurately representing the ground truth systems, given that all r_t and r_v are closer to 1.

In conclusion, the proposed framework outperforms the TN based frameworks to identify the default artificial Volterra system, when the storage complexity threshold is enabled. This is the first advantage of the proposed method. When the threshold and the noise are removed, all the frameworks identify the ground truth system accurately and the resultant Volterra tensors are symmetric. The second advantage of this method is that it requires 10 times less storage complexity than the other frameworks. However, all the frameworks are unable to

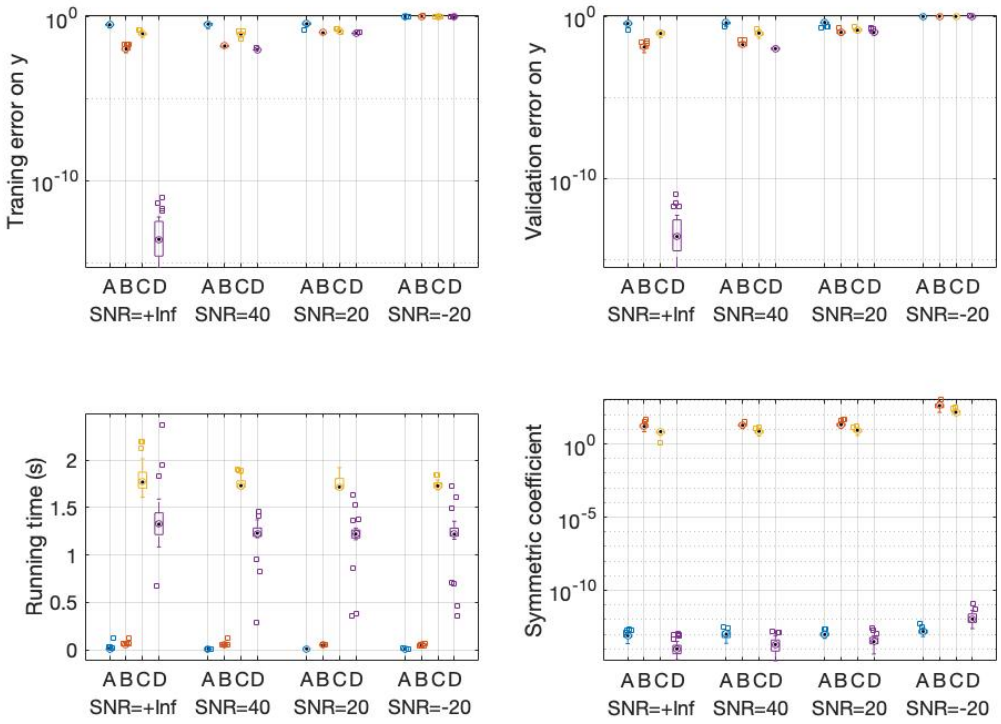


Figure 3-8: Identification performances of four frameworks when the storage complexity threshold is enabled. A stands for MPP, B for ALS, C for MALS, and D for GN.

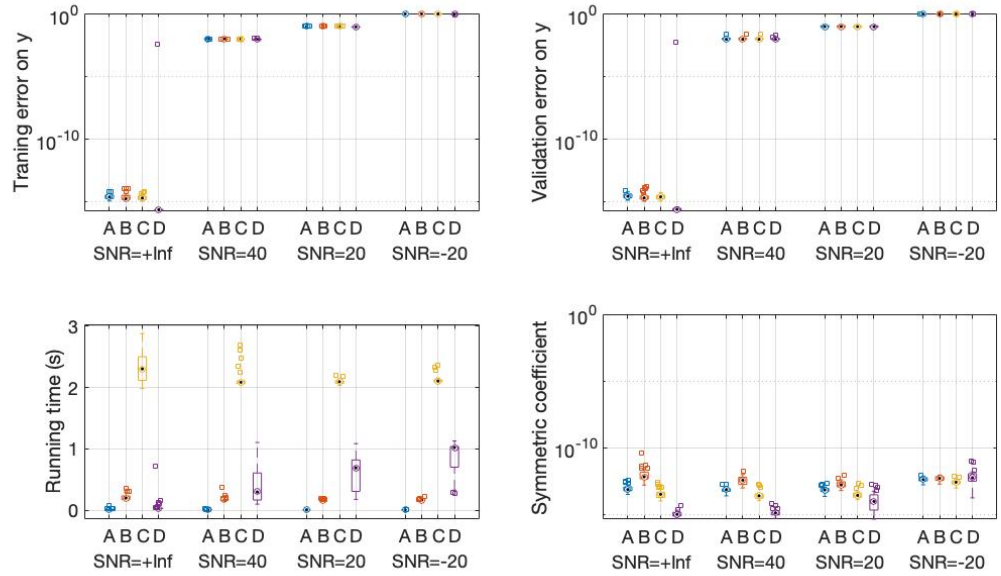


Figure 3-9: Identification performances of four frameworks when the suggested rank values are used. A stands for MPP, B for ALS, C for MALS, and D for GN.

identify the default system when the output noise is added, which is a disadvantage of this method. The performance of this method is sensitive to the initial value and the rank value. This is another disadvantage.

3-5 Conclusion On Phase II

This chapter has proven the validity of the proposed framework to identify the artificial MISO Volterra systems. The conclusions from the three experiments point out that the performance of the proposed framework is sensitive to the choice of the rank value R and the initial value for the GN algorithm.

Based on the content discussed this chapter, the research sub-questions related to Phase II are answered as follows,

Phase II: Validation

1. *How to validate that the synthetic Volterra-PARAFAC model is estimated correctly using the developed framework?*
   (a) The synthetic Volterra-PARAFAC model are determined by two main components, the factor matrix and the weighting vector. Given the estimated factor matrix and weighting vector, the input data and the value of d, the estimated (training and validation) output measurements are simulated within the LS-CPD format, and compared to the ground truth (training and validation) output measurements. The developed framework can be validated in terms of the difference between the estimated and ground truth output measurements.
2. *What is(are) the suitable figure(s) of merit to reflect the identification performance of the proposed framework?*
   (a) The figures of merit should be able to qualitatively describe (1) the closeness of the identified system to the ground truth system, and (2) the resources the framework takes to convergence.
   (b) Five figures of merits that qualify the above conditions are used and explained in section 3-1.
3. *What is(are) the hyper-parameter(s) of the proposed framework? What is the effect of having different hyper-parameter(s) values, on the identification performance?*
   (a) The system order d, the tensor dimension I and the symmetric CPD rank value R are the hyper-parameters of the proposed framework when identifying an unknown system. If the objective system is partially known, such that the values of d and I are given, only R is the hyper-parameter. An example of the partially known system is the artificial Volterra system.

(b) The effect of having an overestimated (estimation rank value is higher than the ground truth value) and underestimated (estimation rank value is lower than the ground truth value) rank value R is opposite. With an overestimated rank value, the framework can accurately identify the artificial Volterra system. It fails to do so with an underestimated rank value.

(c) The influence of d and I can be investigated on an unknown system identification, such as the human cortical response system. This is because the simulated output measurements of the artificial Volterra system depend on the values of d and I. It is not appropriate to use the performance results under different output measurements to discuss the hyper-parameter effect. However, for an unknown system, the influence of d and I will yield an empirical result and may not reflect the underlying true system.

4. *Is the identification performance influenced by the initial value for the optimizer? What are the advantages and disadvantages of different initialization methods?*

   (a) Yes, the influence has been shown in the experiment results in Table 3-2, where the difference between the validation errors of two initialization methods (random and algebraic methods) is significant.

   (b) The random initialization does not require extra resources but its validation error is much higher than the algebraic method. The algebraic method returns an almost-optimal initial value, but it consumes large storage and computational complexities. Three figures of merit are designed in section 3-3 to reflect the balance between the quality and the extra resources required from the algebraic method. Simulation results indicate the need for extra resources when the Volterra tensor to be estimated has large values of d and I.

   (c) The decision of which initialization method is used depends on the priority in practice. The algebraic initialisation approach is preferred if the accuracy of the estimated model is a priority and the total computational and storage complexity budget for the framework is sufficient. Otherwise, if the budget takes priority, the random initialization method is used.

5. *When there is a storage complexity budget, what are the advantages and disadvantages of the proposed framework compared to the existing frameworks on identifying the simulated artificial Volterra system?*

   (a) The experiment in section 3-4 is designed specifically to answer this question. The reader is suggested to reference the conclusion of section 3-4.

Chapter 4

Application: Modeling Evoked Cortical Responses

This chapter aims to apply the developed framework to model the evoked cortical responses. Section 4-1 describes the data acquisition process and the structure of the data set. The procedure of modifying the data set and the figures of merit used to reflect the modelling quality are covered in section 4-2. The purposes, expected challenges and results of two modelling experiments are explained in section 4-3. This chapter ends by answering the research sub-questions of Phase III in section 4-4. All the computations were performed in the same environment as the previous chapter.

4-1 Data Set Description

The wrist joint manipulation elicits a response from the sensors in the periphery which, via the spinal cord, arrives in the cortex. The evoked cortical responses can be measured on the scalp using electroencephalography (EEG). The EEG data set features the response in the human cortex (output) to robotic manipulations of the wrist joint (input) [36]. It has been used as the benchmark for testing the nonlinear system identification framework.

The EEG data set is collected from ten healthy right-handed participants. Participants were seated with their right forearm fixed to an arm support and their right hand fastened to the handle of a robotic manipulator, as shown in Figure 4-1. The right forearm of the subject is strapped into an armrest and the right hand is strapped to the handle, requiring no hand force to hold the handle. Participants were instructed to gaze at the screen, which showed a static target. Participants were asked to relax their wrists and not respond to the robotic manipulator's continual rotational disturbance.

The input perturbation were the summation of several multisine signals with frequencies range from 1 to 23 Hz with a duration of 1 s. For each participant, seven different phase realizations of the input multisine signals (i.e. same amplitude per frequency, yet other random

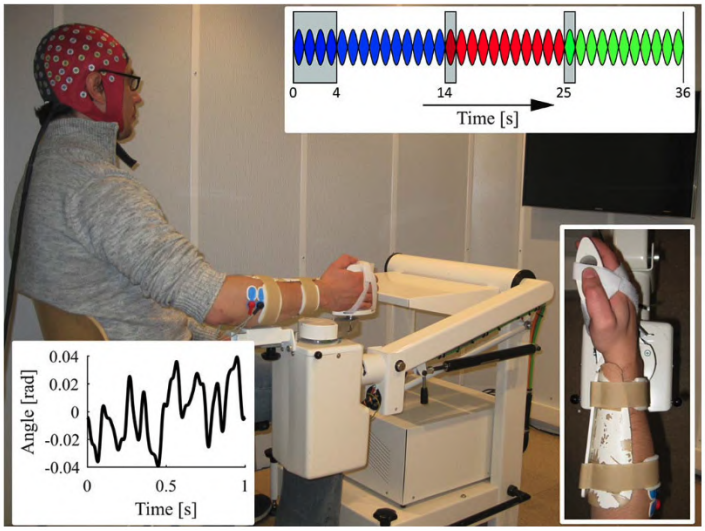


Figure 4-1: EEG experimental setup and overview. Participants were seated with their right forearm fixated to an arm support and their hand strapped to the handle of a robotic manipulator (figure from [16]). (1) The top-right inset provides a schematic illustration of one 36s trial. The three various colors reflect different multisine realizations, and each lobe represents one 1s period of the perturbation signal. Highlighted periods are removed, leaving 10 periods per realization for examination in each experiment. (2) One of the perturbation signal realizations is shown in the bottom-left inset. (3) The bottom-right inset depicts a close-up of the hand in the robotic manipulator. The axis of rotation of the manipulator was aligned with the wrist joint.

phases) are used. Seven corresponding output measurements were recorded from the EEG amplifier. The output measurements are assumed to be corrupted by the white noise that is defined as a normal distribution with stationary and finite variance. The reader is suggested to reference [16] for a detailed explanation of the experimental setup, the input perturbation signals and the data post-processing.

Recent research has applied the Volterra system to model the EEG data set. [16] concludes that the linear model can only explain 10% of the variance of the evoked response, and over 80% of the response is generated by nonlinear behavior. In this article, a second order truncated Volterra system is used to model the non-linearity in the EEG data set. The obtained model is able to explain 46% of the variance of the evoked response. This result offers insight into the relevance between modeling the evoked response and Volterra system identification.

The experiments in this chapter aim to (1) modify the EEG data set into the LS-CPD format that can be identified by the proposed framework; (2) choose the figure(s) of merit to describe the modeling performance; (3) investigate the influence of the values of hyper-parameters on the modeling performance; (4) test the plausibility of the modeled system.

4-2 Modeling Experiments Setup

The input and output measurements of the EEG data set are modified to fit the LS-CPD format. The training and validation data set for the modeling experiments are built based on

the modified measurements. The procedure is explained as follows,

1. For every realization, the corresponding input matrix $\mathbf{U} \in \mathbb{R}^{N\times(M+1)}$ is built in the same structure as in Equation 2-7. There are 210 samples in each realization. The value of N is 210. The corresponding output vector $\mathbf{y} \in \mathbb{R}^N$ stores the output measurements of 210 samples row-wise.

2. The first M rows of the resultant input matrix $\mathbf{U}$ include some elements with a negative time index $t-\tau < 0$, where τ stands for the delay index. The value of these elements are zero (unknown). In order to decrease the transient effect, the first M rows of $\mathbf{U}$ and $\mathbf{y}$ for each realization are removed. This result in the new input matrix $\mathbf{U} \in \mathbb{R}^{(N-M)\times(M+1)}$ and output vector $\mathbf{y} \in \mathbb{R}^{N-M}$ for each realization.

3. For each participant, six realizations are used for training and the remaining realization is used for validation.

4. In the training data set, the training output data is built by storing the output vectors of the six realizations in a row vector, denoted as $\mathbf{y}_t^i \in \mathbb{R}^{6(N-M)}$. The value of i represents the sequence number of the realization that is used for validation. The corresponding training input data is built in the same manner, denoted as $\mathbf{U}_t^i \in \mathbb{R}^{6(N-M)\times(M+1)}$.

5. The input matrix and output vector of the i^{th} realization are used to build the validation data in which the validation input data is denoted as $\mathbf{U}_v^i \in \mathbb{R}^{(N-M)\times(M+1)}$ and the validation output data is denoted as $\mathbf{y}_v^i \in \mathbb{R}^{N-M}$.

6. The above procedure is repeated seven times to achieve seven-fold cross-validation. The repetition results in seven groups of $\mathbf{U}_t^i$, $\mathbf{y}_t^i$, $\mathbf{U}_v^i$, and $\mathbf{y}_v^i$. Each group has a corresponding estimated model.

The same figure of merit from [16] is used as an unified criterion for the comparisons among research. The performance of each estimated model is evaluated by the variance accounted for (VAF) on the training and validation output data, mathematically expressed as,

$$\begin{aligned} VAF_t &= \left(1 - \frac{\text{var}(\mathbf{y}_t^i - \hat{\mathbf{y}}_t^i)}{\text{var}(\mathbf{y}_t^i)}\right) \cdot 100\% \\ VAF_v &= \left(1 - \frac{\text{var}(\mathbf{y}_v^i - \hat{\mathbf{y}}_v^i)}{\text{var}(\mathbf{y}_v^i)}\right) \cdot 100\% \end{aligned} \tag{4-1}$$

where VAF_t and VAF_v represent the training and validation VAF, respectively. var(.) represents the variance, and $\hat{\mathbf{y}}$ represents the modelled output.

4-3 Experiments Results

In this section, two modelling experiments with the modified EEG data set of one single participant are performed. The first experiment focuses on improving the modeling quality on one validation realization. It involves tuning the hyper-parameters empirically, and investigating the influence of each hyper-parameter on the modeling performance in terms of

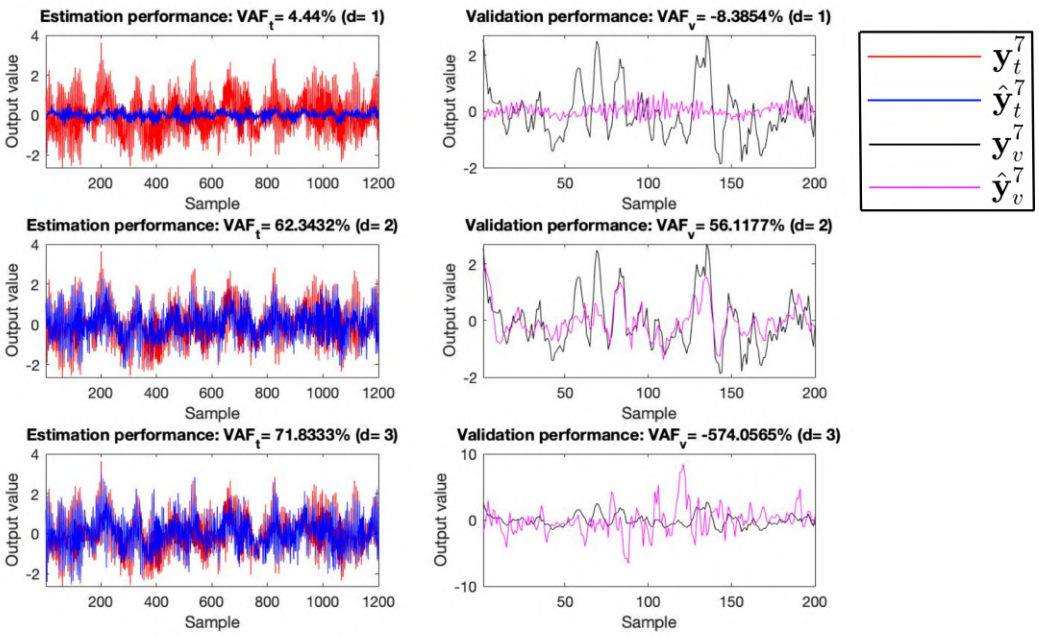


Figure 4-2: Training (left) and validation (right) performances of three system order values $d = 1$ (first row), $d = 2$ (second row), $d = 3$ (third row), while $M = 25$, $R = 20$ are kept as constant. The first order system have low values of VAF_t and VAF_v. The third order system suffers from the over-fitting issue. The second order system has the highest VAF_v.

VAF_t and VAF_v. The second experiment focuses on the similarity within the seven models estimated from the seven-fold cross-validation. In both experiments, twenty trials with the algebraic initialization method are performed for each training data set, while only one trial is plotted in the following figures. This is because that the algebraic initialization method has been proven to provide an almost-optimal solution for the GN algorithm, and the variance of VAF_t and VAF_v for twenty trials is found to be small.

4-3-1 Hyper-parameter tuning

The first six realizations of the first participant are modified as the training data set, denoted as $\mathbf{U}_t^7$ and $\mathbf{y}_t^7$. The remaining realization is modified as the validation data set, denoted as $\mathbf{U}_v^7$ and $\mathbf{y}_v^7$. The symmetric CPD rank is initially set as $R = 20$ [1], and the memory value is set initially as $M = 25$ [2].

[1] This number is chosen to ensure that the storage and computational complexity and the training time for the GN algorithm are moderate. The rank value is tuned afterwards.

[2] 25 samples corresponds to approximately 120 ms at a sampling rate of 256 Hz. The choice is inspired by [16].The memory value is also tuned afterwards.

Tuning d

Three system order values are used, $d = 1, 2, 3$. The simulated training and validation output, denoted as $\hat{\mathbf{y}}_t^7$ and $\hat{\mathbf{y}}_v^7$, respectively, for three order values are visualized in Figure 4-2. The corresponding VAF_t and VAF_v are provided in the title of each plot. Several observations and conclusions based on the analysis of Figure 4-2 are listed as follows,

1. The first order system have both low value of VAF_t and VAF_v. This indicates that modeling the EEG data set with linear system is insufficient.
2. The second order system has the highest VAF_v. However, this only represents one validation realization. There is no claim being made that the actual underlying system can be modelled by the second order Volterra system.
3. The third order system suffers from the over-fitting problem, given that the corresponding VAF_t is the highest but the VAF_v is the lowest. One potential solution to solve this problem is to recollect the data set with a richer perturbation signal, in terms of more excited frequencies and longer period length.

The assumption is recalled that the output measurements are corrupted by the white noise. In an ideal situation (the modeled system represents exactly the true underlying system), the estimated validation output has the following properties:

Property 2: Properties Of The Estimated Validation Output [37]

The estimated validation output residuals $\mathbf{r}_v^i$, calculated as $\mathbf{r}_v^i = \mathbf{y}_v^i - \hat{\mathbf{y}}_v^i$, are also white noise. This implies that $\mathbf{r}_v^i$ are normally distributed with zero mean and finite variance σ. This further means that the auto-correlation function of $\mathbf{r}_v^i$ has an impulse value at zero leg, mathematically expressed as follows,

$$\mathbb{E}[\mathbf{r}_v^i(n)\mathbf{r}_v^i(n-\tau)] = \begin{cases} \sigma^2, & \tau = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4-2}$$

The relationship between the validation input measurements $\mathbf{U}_v^i[:, 1]$, denoted as $\mathbf{u}_v^i$, and the estimated validation output $\hat{\mathbf{y}}_v^i$ is fully embedded in the model class. This implies that $\mathbf{r}_v^i$ and $\mathbf{u}_v^i$ are uncorrelated, which further implies that the cross-correlation of $\mathbf{r}_v^i$ and $\mathbf{u}_v^i$ has zero value at any leg value, mathematically expressed as follows,

$$\mathbb{E}[\mathbf{r}_v^i(n)\mathbf{u}_v^i(n-\tau)] = 0, \quad \text{for any } \tau \tag{4-3}$$

One way to test the plausibility of the modeled systems in Figure 4-2 is to examine the properties of the corresponding validation output residual $\mathbf{r}_v^i$. If $\mathbf{r}_v^i$ could satisfy Equation 4-2 and Equation 4-3 simultaneously, it is plausible that the modeled systems resemble the true underlying system.

The MATLAB ® script *autocorr.m* is used to calculate the auto-correlation first with the

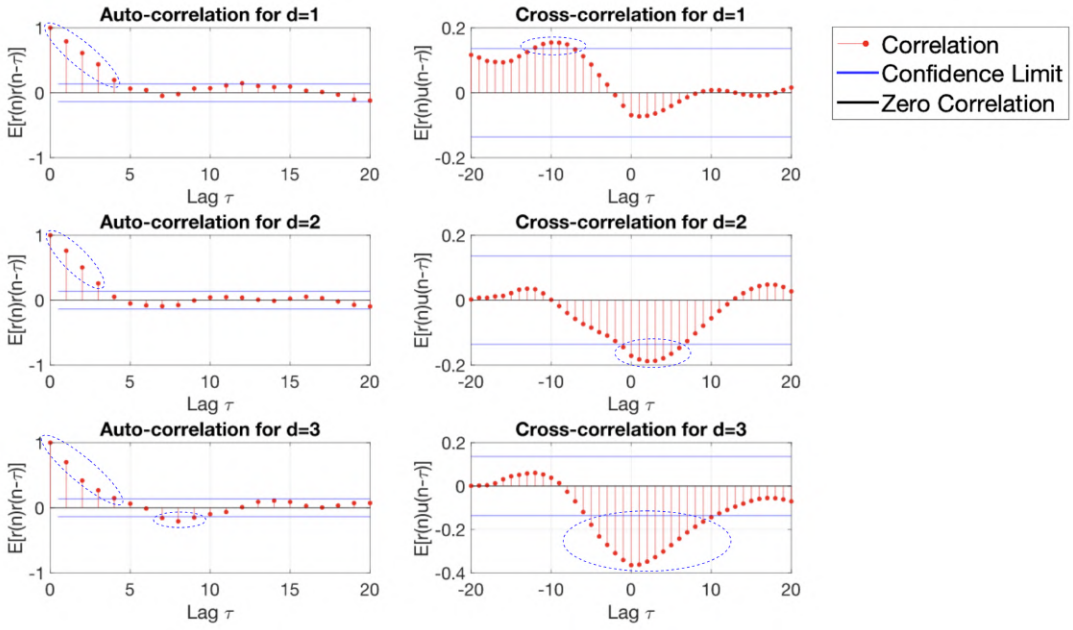


Figure 4-3: Auto-correlation (left) and cross-correlation (right) of the validation residuals for three modelled systems in Figure 4-2. (1) Left: the spikes at zero leg are obvious for three order cases. The second order system has the least spikes at non-zero leg that exceed the confidence interval. (2) Right: the second order system has the less spikes at positive leg that exceed the confidence interval than the third order system. The first order system has spikes that exceed the confidence interval only for a few negative leg values.

confidence interval. The left three plots in Figure 4-3 depict the auto-correlation of the residual for each modelled system in Figure 4-2, respectively. The spike at leg $\tau = 0$ is clear in three cases. However, as pointed out by the blue marks, the spikes at non-zero leg values with $\tau = 1, 2, 3, 4$ exceed the confidence intervals in the first and third order system. For the third order system, the spikes at $\tau = 7, 8, 9$ also exceed the confidence interval. For the second order system, it is a bit closer to the properties, given that less spikes at non-zero leg values exceed the confidence interval.

The results indicate that residual signals are not white for three modelled systems. The dynamics active in the residuals do not fully come from the noise. Therefore, three modelled systems do not meet the properties, and thus are not precise enough to resemble the true underlying system. This conclusion can be substantiated by the visualization of the cross-correlation between the residuals and the input measurements for three modeled systems, shown in Figure 4-3(right).

The MATLAB ® script *crosscorr.m* is used to calculate the cross-correlation and the confidence interval. In Figure 4-3(right), the magnitude of the correlation function for $d = 2$ has much lower and less spikes that exceed the confidence interval for positive leg values, compared to the case of $d = 3$. This implies that the influence of previous input exists in the current residuals for both systems, while the second-order system has less influence. For the first order system, the spikes exceed the confidence interval for negative leg values. This

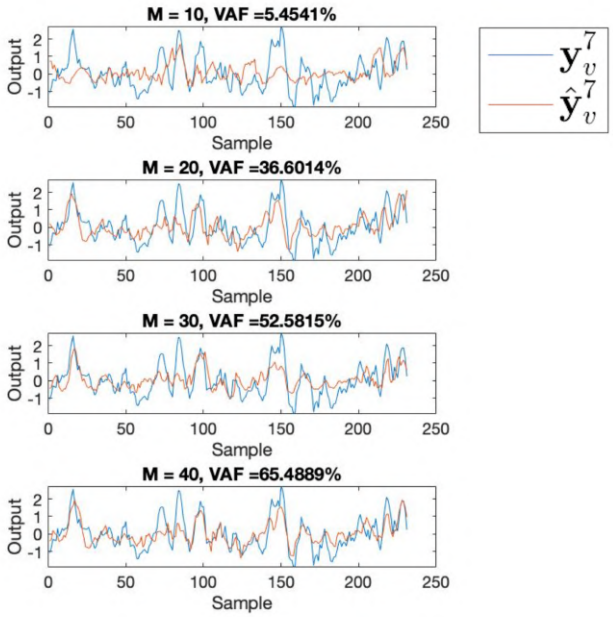


Figure 4-4: Validation performances for $M = [10, 20, 30, 40]$ with $d = 2, R = 20$. The VAF_v increases with M.

implies that the current residual affect future inputs, which does not mean that the model is faulty because this could be seen as an indication of output feedback.

In conclusion, the properties of the modelled system for $d = 2$ are more closer to the ideal situation, given the current data set. This is aligned with the modeling results in [16], where the dynamic in the EEG data set is modeled by a second-order Volterra system. However, from the results in this sub-section, it can not be concluded that there are no nonlinearities in the system higher than the second order. The estimation of higher order Volterra kernels would increase the storage and computational complexities, and the over-fitting issue might require an experiment to recollect the data set with a richer perturbation signal.

Tuning M

In the previous experiment, the memory value is set as $R = 20$. The spikes of the cross-correlation function for the second order system exceed the confidence interval for some positive leg values. This implies the past inputs influence the current residual, which indicates that the modeling quality could be improved by increasing the memory value. The larger memory values is, the more past input measurements are considered to predict the current output measurement.

Four memory values are used, $M = [10, 20, 30, 40]$[3], while $d = 2$ and $R = 20$ are kept

[3]This range contains both the memory values that are less and more than the initial value, which can be used to verify the above indication from both aspects. In order to have the moderate storage and computational

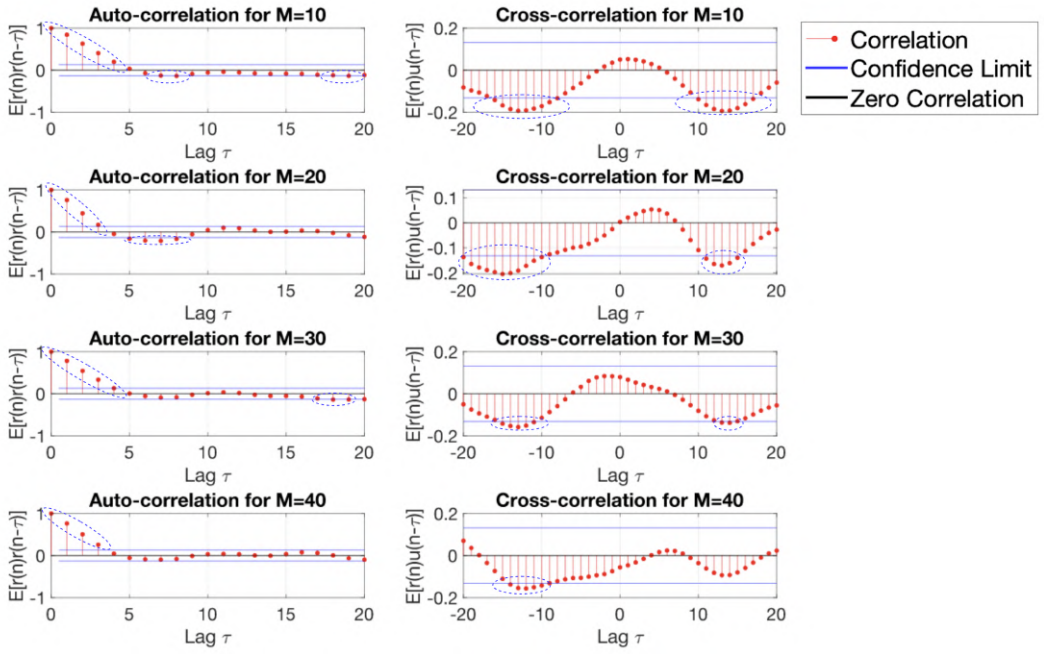


Figure 4-5: Auto-correlation (left) and cross-correlation (right) functions of the validation residuals for four modeled systems in Figure 4-4. For both functions, the number of spikes that exceed the confidence interval with positive lag values decreases with increasing M.

constant. Each memory value corresponds to a pair of training and validation input data. The validation performances are shown in Figure 4-4, where VAF_v is expected to increase with M. Meanwhile, Figure 4-5 shows the auto-correlation (left) and cross-correlation (right) functions of the validation residual signals $\mathbf{r}_v^i$ of the four modelled system, respectively. The spikes that exceed the confidence interval at the positive leg values for both auto-correlation and cross-correlation functions shrink with increasing M. For $M = 40$, there is no spike at the positive lag value that exceeds the confidence interval, which further supports that the validation residual of the modelled second order system with $M = 40$ is closer to the ideal situation than $M < 40$.

Tuning R

Another question is that is the low-rank identification possible for modeling this system? To answer this question, a big range of rank values are adopted, $R = [10, 15, 20, 30, 40, 70]$, while $d = 2, M = 30$ remain constant. The small memory value $M = 30$ is used instead of $M = 40$ to decrease the computational complexity. The corresponding validation performances are shown in Figure 4-6. The VAF_V stays almost the same when $R > 20$. and decreases insignificantly when $R < 20$. This implies that choosing a large rank value is unnecessary due to the higher computational complexity and insignificant improvement of the performance. The low-rank identification is possible in exchange of the model accuracy. The choice of rank value is left as an open question.

complexity, the maximal rank value $R = 40$ is used.

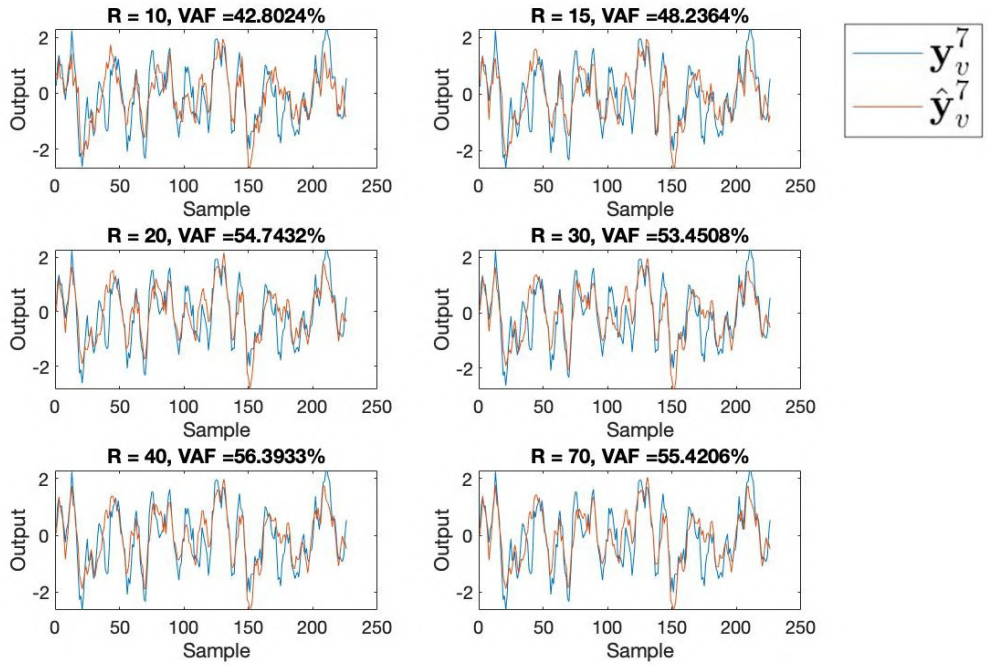


Figure 4-6: Validation performance for $R = [10, 12, 20, 30, 40, 70]$ with $d = 2, M = 30$. The VAF_V stays almost the same when $R > 20$. and decreases insignificantly when $R < 20$.

4-3-2 Similarity

The true underlying system for the cortical response of the same participant should be consistent regardless of disturbance signals. This requires that the seven modelled systems corresponding to seven-fold cross-validation for the same participant (participant 1) are supposed to be similar. Therefore, the plausibility of the seven modelled systems can be tested by calculating the relative differences between the modelled Volterra tensors that are recovered from the estimated factor matrix and weighting vector following Equation 2-13. The relative differences $r_{i,j}$ are calculated as follows,

$$r_{i,j} = \frac{\|\mathbf{V}^i - \mathbf{V}^j\|_F}{\|\mathbf{V}^i\|_F} \tag{4-4}$$

where $\mathbf{V}^i$ stands for the recovered Volterra tensor with the i^{th} realization used for validation. In the ideal situation, the value of $r_{i,j}$ is close to zero for every value combination of i and j.

The relative differences between any two of the seven modelled systems for the first participant are shown in Figure 4-7. Although the differences are smaller for two adjacent realizations, given that the elements near the left-diagonal are smaller, the large value and inconsistency of the differences cast doubt on the credibility of the seven modeled systems all being the true underlying system.

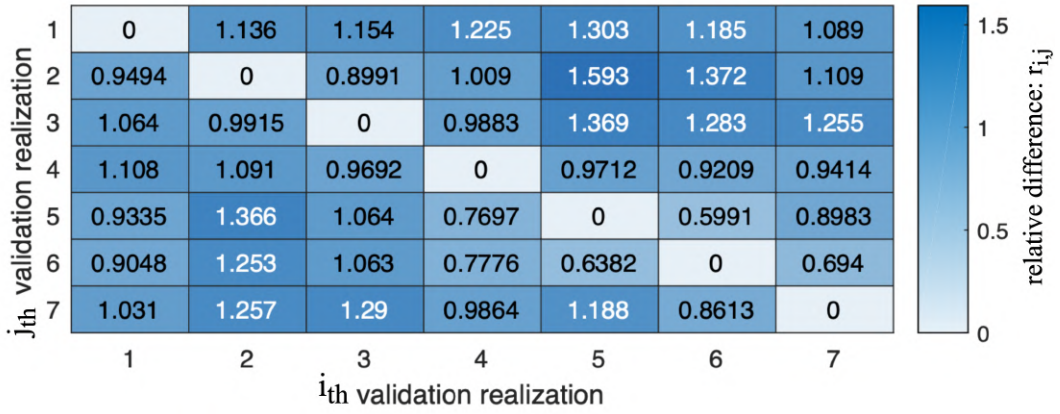


Figure 4-7: Relative differences between any two of the seven modelled systems for the first participant. In the ideal situation, each element in this table should be close to zero. The results cast doubt on the credibility of the seven modeled systems all being the true underlying system.

4-4 Conclusion On Phase III

This chapter has shown that the second order Volterra system is able to model the modified EEG data set. Although it is not necessarily believed that the modelled system can accurately represent the true underlying system by examining the residuals, the potential of applying the framework in modeling the highly-nonlinear systems is demonstrated.

Based on the content discussed this chapter, the research sub-questions related Phase III are answered as follows,

Phase III: Application

1. *The human cortical responses are modeled by the Volterra-PARAFAC model whose kernel coefficients are estimated by the developed framework. Does the modeling quality rely on the values of the hyper-parameter(s) of the framework?*

   (a) The modeling quality is described by the the variance accounted for (VAF) on the training and validation output measurements. The corresponding calculations are expressed in Equation 4-1.

   (b) The hyper-parameters contain d, M and R. In section 4-3, results have shown that the validation VAF is influenced significantly by d and M. The second order system with $M = 40, R = 20$ returns the highest validation VAF. Low-rank ($R < 20$) approximation is also possible in exchange of lower validation VAF. The large rank value, however, requires high computational and storage complexity. The decision of R value depends on the overall computational and storage complexity budget.

2. *How to test the plausibility of the modelled system? What are the potential reason(s) for the modelled system not representing the true underlying system?*

(a) The output measurement is assumed to be disturbed by the white noise. The validation output residual signal can be used. In an ideal situation, the auto-correlation function of the residuals has an impulse value at leg $\tau = 0$ and stays within the confidence interval for non-zero leg values. The cross-correlation function of the residual and input measurement stays within the confidence interval for all leg values. If the validation residual of the modelled system meets this assumption, it is plausible that the modelled system resembles the true underlying system.

(b) Results in section 4-3 cast doubt on the credibility of the modeled system being the true underlying system. The reasons could be that (1) the modification of the data set is not appropriate so that the transient effect still exists; (2) The hyper-parameters are tuned empirically in the experiments. The prior information about the data set is insufficient; (3) The framework has been shown to have a weak robustness against the noise for identifying the artificial system. There are white noise in the output measurements of the EEG data set. The robustness of the framework is weak.

Chapter 5

Conclusions And Future Work

In the final chapter, a summary of the thesis and an overview of open questions for future research are provided.

5-1 Concluding Remarks

In this thesis, the connection between the symmetric canonical polyadic decomposition (CPD, also called PARAFAC) and the second order numerical optimization method is established to identify the high order multiple-input single-output (MISO) Volterra system. As a result, we now have an innovative identification framework to estimate the parameter values of the MISO Volterra-PARAFAC model by minimising the nonlinear least-square (NLS) cost function via the Gauss-Newton (GN) algorithm. In the following, we give a chapter-by-chapter overview of this thesis.

In Chapter 2, we have used the tensor operations and the symmetric CPD technique to model the Volterra system within the LS-CPD framework. The parameter values of this framework is estimated by minimising the NLS cost function. The cost function is calculated as the squared error between the ground truth output and the simulated output that is recovered based on the LS-CPD framework and the components of the Volterra-PARAFAC model. The optimization variable of the cost function contains the components of the Volterra-PARAFAC model, and is optimized by the GN algorithm with the trust region method. The random initialization and algebraic initialization methods are provided for the GN algorithm. The former initializes the optimization variable from random values, while the latter initializes from the result of some algebraic operations. The algebraic initialization method can return an almost-optimal initial value, but it consumes large storage and computational complexities.

In Chapter 3, we have proved the validity of the developed framework to identify an artificial MISO Volterra system. The output of the artificial Volterra system is simulated by the synthetic Volterra-PARAFAC model and input measurements. The synthetic Volterra-PARAFAC model contains two components that are the factor matrix and the weighting

vector. The former has each column created as the sampled decaying exponential function, while the latter contains pseudo-random values drawn from the standard uniform distribution on the open interval $(0, 1)$. The input measurements are created as standard normal Gaussian white noise to guarantee the input signal being persistently exciting. Several figures of merit are defined to reflect the identification performances of the developed framework on identifying the artificial Volterra systems under different configurations.

Experiments result show that the developed framework can accurately identify the artificial Volterra system in terms of validation figures of merit when an over-estimated rank value is used. It fails to do so with an underestimated rank value. Meanwhile, three figures of merit are designed to reflect the balance between the quality and the extra resources required from the algebraic initialization method. Simulation results indicate the need for extra resources when the size of the Volterra tensor to be estimated is large. In addition, when the noise on the output is removed, the developed framework outperforms the Tensor-Network (TN) based frameworks to identify the same artificial Volterra system when the storage complexity budget is added, and performs equally but requires 10 times less storage complexity than the other frameworks when the budget is removed. However, the performance of the developed method is sensitive to the choice of the initial value and the rank value, and is not robust against the white noise added to the output.

In Chapter 4, we have applied the validated framework to model the evoked cortical responses data set that are measured on the scalp using electroencephalography (EEG) and its input-output relationship has been previously proven to be highly nonlinear. This nonlinearity is planned to be modelled by the unknown order Volterra-PARAFAC model. The realizations (input-output measurements) are first modified to fit the LS-CPD format as a single-input single-output (SISO) system. The GN algorithm is then used to estimate the factor matrices and weighting vectors of several Volterra-PARAFAC models with different value combinations of system order d, input memory M and approximation rank R. The modeling quality is reflected by the variance accounted for (VAF) on the training and validation output data, respectively. The results have shown that the validation VAF is influenced significantly by d and M. The second order systems with $M = 40$ and $R \in [20, 40]$ return the highest validation VAF. The low-rank ($R < 20$) approximation is also possible in exchange of lower validation VAF. The large rank value ($R > 40$), however, requires high computational and storage complexity. Meanwhile, the validation output residual signals are used to test the plausibility of the modelled systems. The test results cast doubt on the credibility of the modeled systems being the true underlying system. The reasons could be the inappropriate data set modification that results in the transient effect, the insufficient prior information on the choice of framework hyper-parameters, the limited range of hyper-parameter values where they are tuned empirically, and the poor robustness of the developed framework against the noise.

5-2 Future Research

Many theoretical and computational challenges remain unsolved because this thesis represents the humble beginning of a Volterra system identification framework. Besides the permanent

objective of applying the tensor techniques to solve engineering problems, we would suggest the following topics as the future research for each chapter of this thesis:

5-2-1 Development

The MISO Volterra system is modeled within the LS-CPD format. This format is originally proposed in [38] and has been proven to provide a broad framework for the analysis of multilinear systems of equations [22]. Based on the development of the proposed identification framework, many tensor techniques can be used to extend its functionality in system identification.

Multiple-input Multiple-output (MIMO) Volterra System Identification: A d^{th} order MIMO Volterra kernel can be defined as a $d+1$ way Volterra tensor, which can be seen as a vector with d elements and each element is a d way tensor. Each output of the MIMO system is determined by one corresponding element. The current LS-CPD format is expected to be modified in the manner how the $d+1$ way MIMO Volterra tensor is decomposed and stacked.

Acceleration On Newton Step Calculation: The New-step is obtained by solving the linear system expressed in Equation 2-23 by the pseudoinverse or the conjugate gradient method. The convergence rate of solving this linear system can be improved by clustering the eigenvalues of the Hessian $\mathbf{H}$ using the preconditioner $\mathbf{M}$ to replace Equation 2-23 by the equation as follows,

$$\mathbf{M}^{-1}\mathbf{H}\mathbf{p} = -\mathbf{M}^{-1}\mathbf{g} \tag{5-1}$$

where the sparsity of the eigenvalues of $\mathbf{M}^{-1}\mathbf{H}$ can be controlled by different choice of $\mathbf{M}$ such that $\mathbf{M}^{-1}$ is not expensive to compute. The recently published results in [32, 25] are both useful starting points for this direction.

Complex-valued Input-output Measurements: The developed framework only considers the case where the input-output measurements are all real-valued. In real-life application, having the complex-valued measurements is possible [1]. The adaptive algorithms to estimate the parameter values of the Volterra-PARAFAC model with complex-valued measurements in [18] can serve as a starting point for extending the complex-valued property of our framework.

5-2-2 Validation

Both the internal and external validation experiments are performed on a limited range of parameter values due to the computation power of our computer. As a result, the conclusions made in this thesis can not represent the true property of the developed framework. The following experiments can be performed to address this gap.

[1] In electrical engineering, the Fourier transform is used to analyze varying voltages and currents. The treatment of resistors, capacitors, and inductors can be unified by introducing imaginary resistors that are frequency-dependent and combining them into a single complex number called the impedance. In fluid mechanics, complex numbers are used to describe potential flow in two dimensions.

Higher Order Artificial Volterra System: The order value of the default artificial Volterra System in this thesis is $d = 5$, which is much smaller than the potential true system order values of the real-life systems. It is meaningful to investigate the identification performances when the order value is larger, for example $d = 20$. Meanwhile, a larger value range for the default CPD rank can also be used to examine the influence of the approximation rank values more precisely.

Robustness Against Noise: The experiment results indicate that the development framework has a weak robustness against noise. Measurements from real world applications are noisy, hence the coefficient of the Volterra tensor will always be subject to some uncertainty. Starting from the recently published results on [39, 40], further research will need to investigate how these uncertainties can be used to determine a suitable confidence interval.

5-2-3 Application

The EEG data set is modelled only by the developed framework in this thesis. The horizontal comparisons of the modeling performance with TN based frameworks are missing. This constrains the comprehensiveness of the strengths and weaknesses of the developed framework. One of the key research points is therefore obviously to look for ways to further exploit the limitations of the developed framework.

Modeling Performances Of All Participants: Only the modelled systems for one out of ten participants in the EEG data set are studied in this thesis. The signal characteristics and models for each participant have been proven to be diverse in [16]. It is necessary to further compare the modeling performances of the developed framework on each participant and test the plausibility of these modelled systems.

Application On More Benchmark Nonlinear Systems: A list of hosted nonlinear dynamical system data sets can provided in [41]. The Volterra system has been used to model most of these nonlinear systems [42, 43]. These research can serve as a starting point to determine whether it is possible to apply the developed framework to obtain a better performance.

Appendix A

State-of-the-art Volterra Identification Frameworks

A-1 Tensor Operations

1. The Mode-n product of an order-N tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ will yield an order- N tensor which is mathematically given by

$$\mathcal{C} = \mathcal{A} \times_n \mathbf{U} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N} \tag{A-1}$$

where the operator indicates the index (mode) of contraction in term of its subscript. The size of this index is required to match the one of the corresponding matrix. The entries of the resultant tensor are determined by element-wise summations written as

$$c_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} = \sum_{i_n}^{I_n} a_{i_1,\ldots,i_n,\ldots,i_d} u_{j,i_n} \tag{A-2}$$

2. The Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$, denoted by $\mathbf{A} \otimes \mathbf{B}$, results in a matrix expressed as

$$\mathbf{C} \in \mathbb{R}^{IK \times JL} := \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix} \tag{A-3}$$

3. The Khatri-Rao product of two matrices $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$, denoted by $\mathbf{A} \odot \mathbf{B}$, results in a matrix expressed by

$$\mathbf{C} \in \mathbb{R}^{IJ \times K} := \mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_K \otimes \mathbf{b}_K \end{bmatrix} \tag{A-4}$$

where $\mathbf{a}_i \in \mathbb{R}^I$ represents the i^{th} column of $\mathbf{A}$.

4. The row-wise Khatri-Rao product of $\mathbf{A} \in \mathbb{R}^{I\times K}$ and $\mathbf{B} \in \mathbb{R}^{I\times J}$ denoted by $\mathbf{A} \odot^T \mathbf{B}$, results in a matrix expressed by

$$\mathbf{C} \in \mathbb{R}^{I\times KJ} := \mathbf{A} \odot^T \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 \\ \mathbf{a}_2 \otimes \mathbf{b}_2 \\ \cdots \\ \mathbf{a}_I \otimes \mathbf{b}_I \end{bmatrix} \tag{A-5}$$

where $\mathbf{a}_i \in \mathbb{R}^{1\times K}$ represents the i^{th} row of $\mathbf{A}$.

5. The Hadamard product of two matrices $\mathbf{A}$ and $\mathbf{B}$ with the same size $\in \mathbb{R}^{I\times J}$, denoted by $\mathbf{A} * \mathbf{B}$, results in a same size matrix defined by

$$\mathbf{C} \in \mathbb{R}^{I\times J} := \mathbf{A} * \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \cdots & a_{IJ}b_{IJ} \end{bmatrix} \tag{A-6}$$

6. The outer product of two vectors $\mathbf{a} \in \mathbb{R}^{I_1}$ and $\mathbf{b} \in \mathbb{R}^{I_2}$ results in a matrix, expressed as

$$\mathbf{C} \in \mathbb{R}^{I_1\times I_2} := \mathbf{a} \circ \mathbf{b} = \mathbf{a}\mathbf{b}^\top \tag{A-7}$$

where the entries of the $\mathbf{C}$ are determined by the element-wise summations defined by

$$c_{i_1,i_2} = \sum_{i=1}^{1} a_{i_1} b_{i_1} \tag{A-8}$$

A-2 Three Tensor Network Based Frameworks

In this section, the procedure to calculate the storage complexity of three TN-based frameworks are explained. For simplicity, three frameworks are denoted as, ALS and MALS from [11], and MPP from [9]. The reader is recommend to reference [11][9] for a detailed derivation of these frameworks.

The TN refers to the network that utilizes the tensor train (TT) decomposition tool to decompose the objective tensor. The TT decomposition of the d-way Volterra tensor is defined as follows,

Definition 4: Tensor Train Decomposition Of Volterra Tensor

The TT decomposition of the d-way Volterra tensor $\mathcal{V} \in \mathbb{R}^{I\times I\times\cdots\times I}$ is represented by the TT cores interconnected by the *Mode 1-3 tensor contraction* operation denoted by $\times_3^1$, mathematically expressed as:

$$\mathcal{V} \in \mathbb{R}^{\overbrace{I \times I \times \cdots \times I}^{d\ times}} = \mathcal{V}^{(1)} \times_3^1 \mathcal{V}^{(2)} \times_3^1 \cdots \times_3^1 \mathcal{V}^{(d)} \tag{A-9}$$

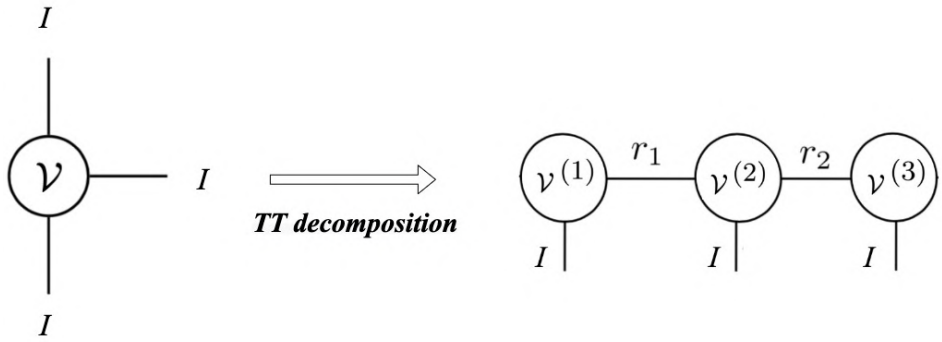


Figure A-1: Diagrammatic notation of the TT decomposition of a 3-way Volterra tensor.

where the 3-way tensors $\mathcal{V}^{(k)}$ are called the TT cores with the size of $\mathbb{R}^{R_{k-1}\times I\times R_k}$. The indices R_{k-1} and R_k in 1^{st} and 3^{rd} modes of the k^{th} TT core are called TT ranks. The boundary rank values for Equation A-9 are $R_0 = R_d = 1$. Each element of $\mathcal{V}$ can be calculated by

$$v_{i_1,i_2,\ldots,i_d} = \sum_{r_1=1}^{R_1}\sum_{r_2=1}^{R_2}\cdots\sum_{r_{d-1}=1}^{R_{d-1}} v^{(1)}_{1,i_1,r_1} v^{(2)}_{r_1,i_2,r_2}\cdots v^{(d)}_{r_{d-1},i_d,1} \tag{A-10}$$

where scalar $v^{(n)}_{r_i,i_t,r_{i+1}}$ represents the r_i^{th} in mode-1, i_t^{th} in mode-2 and r_{1+1}^{th} in mode-3 element of the n^{th} TT core.

The Mode-1,3 tensor contraction of the first and second TT cores means that the 3^{nd} mode of $\mathcal{V}^{(1)}$ is contracted with the 1^{st} mode of $\mathcal{V}^{(2)}$. It results in a new 4-way tensor, mathematically expressed as

$$\mathcal{C}\in\mathbb{R}^{1\times I\times I\times R_2} = \mathcal{V}^{(1)}\times_3^1 \mathcal{V}^{(2)} \tag{A-11}$$

where the entries of the resultant tensor $\mathcal{C}$ are calculated element-wise as

$$c_{1,i_1,i_2,r_2} = \sum_{r_1=1}^{R_1} v^{(1)}_{1,i_1,r_1} v^{(2)}_{r_1,i_2,r_2} \tag{A-12}$$

The storage complexity of the Volterra tensor in the TN format is $\mathcal{O}\left(IdR^2\right)$, where all TT ranks values are presented by R. $\mathcal{O}\left(IdR^2\right)$ is linearly increasing with d and I. Therefore Equation A-9 can also solve the curse of dimensionality issue, especially with small TT ranks values. Figure A-1 shows the diagrammatic notation of the TT decomposition of a 3-way Volterra tensor.

The advantage of applying the TT decomposition on Volterra tensor is that the output $y(t)$ in Equation 2-6 can also be modelled by the multidimensional contraction operation as follows

[11],

$$\begin{aligned} y(t) \in \mathbb{R} :&= \mathcal{V} \times_1 \mathbf{u}_t^T \times_2 \mathbf{u}_t^T \cdots \times_d \mathbf{u}_t^T \\ &= \left(\mathcal{V}^{(1)} \times_2 \mathbf{u}_t^T\right) \left(\mathcal{V}^{(2)} \times_2 \mathbf{u}_t^T\right) \cdots \left(\mathcal{V}^{(d)} \times_2 \mathbf{u}_t^T\right) \\ &= \left(\mathcal{V}_{k+1}^T \otimes \mathbf{u}_t^T \otimes \mathcal{V}_{k-1}\right) \text{vec}\left(\mathcal{V}^{(k)}\right) \end{aligned} \tag{A-13}$$

where

$$\begin{aligned} \mathcal{V}_{k+1}^T &= \left(\mathcal{V}^{(1)} \times_2 \mathbf{u}^T\right) \cdots \left(\mathcal{V}^{(k-1)} \times_2 \mathbf{u}^T\right) \in \mathbb{R}^{1 \times R_k} \\ \mathcal{V}_{k-1} &= \left(\mathcal{V}^{(k+1)} \times_2 \mathbf{u}^T\right) \cdots \left(\mathcal{V}^{(d)} \times_2 \mathbf{u}^T\right) \in \mathbb{R}^{1 \times R_{k-1}} \end{aligned} \tag{A-14}$$

By stacking the output $y(t)$ in a row vector for $t \in [0, N-1]$, Equation 2-7 can also be expressed as follows,

$$\mathbf{y} \in \mathbb{R}^N = \mathbf{U}^{(k)} \text{vec}\left(\mathcal{V}^{(k)}\right) \tag{A-15}$$

where $\mathbf{U}^{(k)} \in \mathbb{R}^{N \times R_{k-1} I R_k}$.

A-2-1 ALS

ALS is used to estimate the parameters values of $\text{vec}\left(\mathcal{V}^{(k)}\right)$ in Equation A-15 sequentially. The advantage of ALS is that each small sub-problem related to only one TT core is addressed sequentially, rather than solving a single massive problem related with the simultaneous update of all TT cores. The ALS is formulated in Algorithm 2:

Algorithm 2: ALS for TT decomposition [11][44]

Input: TT ranks $R_0, R_1, \cdots, R_d$, the value of d and I, and the input output measurements

1 Initialize right-orthogonal TT cores $\mathcal{V}^1, \cdots, \mathcal{V}^d$ with specified ranks $R_0, R_1, \cdots, R_d$
2 **while** *termination criterion not satisfied* **do**
3 **for** $i = 1, \cdots, d-1$ **do**
4 $vec\left(\mathcal{V}^{(i)}\right) \leftarrow$ *Compute and solve Equation* $A-15$
5 $\mathbf{V}^{(i)} \leftarrow reshape\left(\mathcal{V}^{(i)}, [R_{i-1}I, R_i]\right)$
6 *Compute thin QR decomposition of matrix* $\mathbf{V}^{(i)}$
7 $\tilde{\mathcal{V}}^{(i)} \leftarrow reshape\left(\mathbf{Q}, [R_{i-1}, I, R_i]\right)$
8 $\mathbf{V}^{(i+1)} \leftarrow reshape\left(\mathcal{V}^{(i+1)}, [R_i, IR_{i+1}]\right)$
9 $\mathcal{V}^{(i+1)} \leftarrow tensorization\left(\mathbf{R}\mathbf{V}^{(i+1)}, [R_i, I, R_{i+1}]\right)$
10 **end**
11 *Repeat the above loop in reverse order*
12 **end**

Output: Optimal TT cores $\tilde{\mathcal{V}}^{(1)}, \cdots, \tilde{\mathcal{V}}^{(d)}$ that solves Equation A-15

The thin QR decomposition of matrix $\mathbf{V}^{(i)} \in \mathbb{R}^{R_i I \times R_i}$ in the 6^{th} line of Algorithm 2 is expressed as follows,

$$[\mathbf{Q} \in \mathbb{R}^{R_{i-1} I \times R_i}, \mathbf{R} \in \mathbb{R}^{R_i \times R_i}] = qr(\mathbf{V}^{(i)}) \tag{A-16}$$

where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix. Half sweep of 2 is visualized in Figure A-2(a).

The storage complexity of ALS algorithm is the summation of the size of $\mathbf{y} \in \mathbb{R}^N$, $\mathbf{U}^{(k)} \in \mathbb{R}^{N \times R_{k-1} I R_k}$, $\text{vec}\left(\mathcal{V}^{(k)}\right) \in \mathbb{R}^{R_{k-1} I R_k}$, $\mathbf{Q} \in \mathbb{R}^{R_{i-1} I \times R_i}$ and $\mathbf{R} \in \mathbb{R}^{R_i \times R_i}$. The storage complexity is approximated as $\mathcal{O}\left(N + R^2(NI + 2I + 1)\right)$, with R represents the maximal TT rank value.

A-2-2 MALS

The drawback of ALS is that the TT ranks should be specified as priori. The quality of the output of ALS is sensitive to the choice of the TT ranks. MALS is developed to not only updates the TT cores but also adapts the TT ranks during each iteration.

The modification is to contract two subsequent cores $\mathcal{V}^{(i)}, \mathcal{V}^{(i+1)}$ into one compound core $\mathcal{W}^{(i)} \in \mathbb{R}^{R_{i-1} \times I \times I \times R_{i+1}}$ by summing over all possible R_i values, expressed as

$$\mathcal{W}^{(i)} := \sum_{r_i=1}^{R_i} \mathcal{V}^{(i)}(R_{i-1}, I, r_i) \mathcal{V}^{(i+1)}(r_i, I, R_{i+1}) \tag{A-17}$$

which can be applied to Equation A-13 as follows,

$$\begin{aligned} y(t) :&= \mathcal{V}_{k-1} \left(\mathcal{V}^{(k)} \times_2 \mathbf{u}^T\right) \left(\mathcal{V}^{(k+1)} \times_2 \mathbf{u}^T\right) \mathcal{V}_{k+2} \\ &= \mathcal{V}_{k-1} \left(\mathcal{W}^{(k)} \times_2 (\mathbf{u}^T)^{②}\right) \mathcal{V}_{k+2} \\ &= \left(\mathcal{V}_{k+2}^T \otimes (\mathbf{u}^T)^{②} \otimes \mathcal{V}_{k-1}\right) \text{vec}\left(\mathcal{W}^{(k)}\right) \end{aligned} \tag{A-18}$$

where $y(t)$ can be stacked row-wise for $t \in [0, N-1]$, and Equation A-19 can be modified as

$$\mathbf{y} \in \mathbb{R}^N = \mathbf{U}^{(k,k+1)} \text{vec}\left(\mathcal{W}^{(k)}\right) \tag{A-19}$$

where $\mathbf{U}^{(k,k+1)} \in \mathbb{R}^{N \times R_{k-1} I^2 R_{k+1}}$.

The MALS is formulated in Algorithm 3.

Algorithm 3: MALS for TT decomposition [11][44]

Input: TN The value of d and I, and the input output measurements

Initialize right-orthogonal TT cores $\mathcal{V}^1, \cdots, \mathcal{V}^d$ with ranks 1
while *termination criterion not satisfied* **do**
  for $i = 1, \cdots, d-1$ **do**
    $vec\left(\mathcal{W}^{(i)}\right) \leftarrow$ *Compute and solve Equation* $A-19$
    $\mathbf{W}^{(i)} \leftarrow reshape\left(\mathcal{W}^{(i)}, [r_{i-1}n_i, n_{i+1}r_{i+1}]\right)$
    Compute SVD of matrix $\mathbf{W}^{(i)}$
    Determine the numerical rank r_i *from tolerance* τ
    $\tilde{\mathcal{V}}^{(i)} \leftarrow reshape\left(\mathbf{U}, [r_{i-1}, n_i, r_i]\right)$
    $\mathcal{V}^{(i+1)} \leftarrow reshape\left(\mathbf{SV}^T, [r_i, n_{i+1}, r_{i+1}]\right)$
  end
  Repeat the above loop in reverse order
end

Output: Optimal TT cores $\tilde{\mathcal{V}}^{(1)}, \cdots, \tilde{\mathcal{V}}^{(d)}$ that solves Equation A-19

The SVD of $\mathbf{W}^{(i)} \in \mathbb{R}^{R_{i-1}I \times IR_{i+1}}$ in the 6^{th} line of 3 is expressed as follows,

$$[\mathbf{U} \in \mathbb{R}^{R_{i-1}I \times R_i}, \mathbf{S} \in \mathbb{R}^{R_i \times R_i}, \mathbf{V} \in \mathbb{R}^{R_i \times IR_{i+1}}] = SVD(\mathbf{W}^{(i)}) \tag{A-20}$$

where $\mathbf{U}, \mathbf{V}$ are orthogonal matrices and $\mathbf{S}$ is a diagonal matrix with positive entries $s_1 \geq \ldots \geq s_q$, with $q = \min(R_{i-1}I, IR_{i+1})$. The rank R_i can be determined from a tolerance τ such that $s_1 \geq \cdots \geq s_{r_1} \geq \tau \geq \cdots \geq s_q$, which is decided by $\tau = \epsilon s_1 \max(R_{i-1}I, IR_{i+1})$, where ϵ is the machine precision. Half sweep of 3 is visualized in Figure A-2(b).

The storage complexity of MALS algorithm is the summation of the size of $\mathbf{y} \in \mathbb{R}^N$, $\mathbf{U}^{(k,k+1)} \in \mathbb{R}^{N \times R_{k-1}I^2R_k}$, $\text{vec}\left(\mathcal{W}^{(k)}\right) \in \mathbb{R}^{R_{k-1}I^2R_k}$, $\mathbf{U} \in \mathbb{R}^{R_{i-1}I \times R_i}$, $\mathbf{S} \in \mathbb{R}^{R_i \times R_i}$ and $\mathbf{D} \in \mathbb{R}^{R_i \times IR_{i+1}}$. The storage complexity is approximated as $\mathcal{O}\left(N + R^2I^2(I+1) + 2R^2I\right)$, with R represents the maximal TT rank value.

A-2-3 MPP

The input matrix $\tilde{\mathbf{U}} \in \mathbb{R}^{N \times I^d}$ in Equation 2-7 can be expressed in the TT format as follows,

$$\begin{aligned} \tilde{\mathbf{U}} &= \left(\mathbf{U} \odot^{\mathrm{T}} \mathbf{U} \odot^{\mathrm{T}} \cdots \odot^{\mathrm{T}} \mathbf{U} \odot^{\mathrm{T}}\right) \\ &= \mathcal{U}^{(1)} \times_4^1 \mathcal{U}^{(2)} \times_4^1 \cdots \times_4^1 \mathcal{U}^{(d)} \end{aligned} \tag{A-21}$$

where $\mathcal{U}^{(k)} \in \mathbb{R}^{R_k \times 1 \times I \times R_{k+1}}$ and $R_1 = N, R_{d+1} = 1$. The TT cores can be calculated by in Algorithm 4 [9], which is a modified version of Algorithm 2 in [5].

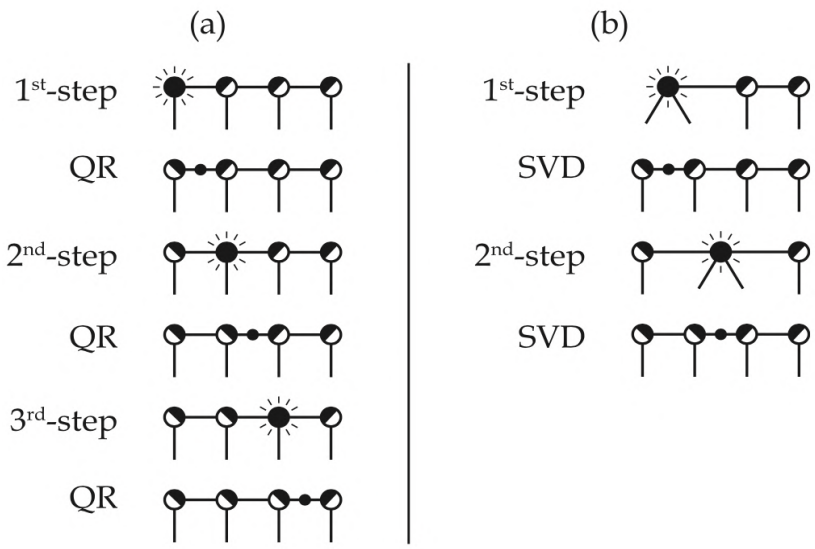


Figure A-2: TN diagram of the "half sweep" of the (a) ALS algorithm and (b) MALS algorithm for 4-way tensor. Image is reproduced from [44]. The highlighted TT core $\mathcal{V}^{(i)}$ (or compound core $\mathcal{W}^{(i)}$ in the MALS instance) is optimized in each i^{th} iteration. Afterwards, the non-orthogonal TT core is shifted to the next one $\mathcal{V}^{(i+1)}$ (or $\mathcal{W}^{(i+1)}$) by performing the QR decomposition of $\mathbf{V}^{(i)}$ (or SVD of $\mathbf{W}^{(i)}$). When the procedure reaches the rightmost TT core, it is reversed in the "back sweep" (not depicted here).

Algorithm 4: Constructing TT cores $\mathcal{U}^{(1,2...d)}$ from $\mathbf{U}$ [9][5]

Input: The value of d, and matrix $\mathbf{U}$

$\mathcal{U}^{(d)} \leftarrow \text{reshape}(\mathbf{U}, [1, N, I, 1])$

for $k = d : -1 : 2$ **do**

  $\boldsymbol{T} \leftarrow \text{reshape}\left(\mathcal{U}^{(k)}, [N, IR_{k+1}]\right)$

  $\boldsymbol{T} \leftarrow \boldsymbol{T} \odot \mathbf{U}$

  $\boldsymbol{T} \leftarrow \text{reshape}\left(\boldsymbol{T}, [NI, IR_{k+1}]\right)$

  $[\mathbf{Q}, \mathbf{S}, \mathbf{V}] \leftarrow \textit{Compute SVD of matrix } \boldsymbol{T}$

  Determine the rank $R_k = \begin{pmatrix} d-k+I-1 \\ I-1 \end{pmatrix}$

  $\mathcal{U}^{(k)} \leftarrow \text{reshape}\left(\mathbf{V}^T, [R_k, 1, I, R_{k+1}]\right)$

  $\mathcal{U}^{(k-1)} \leftarrow \text{reshape}\left(\mathbf{QS}, [1, N, I, R_k]\right)$

end

Output: TT cores $\mathcal{U}^{(1,2...d)}$ in Equation A-21

The rank condition R_k can be further bounded by the proposed Theorem 3 in [9] when considering the symmetric solution of $\mathbf{V}_{sym}$, expressed as

$$R_k \leq \min\left(\begin{pmatrix} k+I-1 \\ I-1 \end{pmatrix}, \begin{pmatrix} d-k+I-1 \\ I-1 \end{pmatrix}\right) \tag{A-22}$$

Compared with $\tilde{\mathbf{U}} \in \mathbb{R}^{N \times I^d}$, the TT formats $\mathcal{U}^{(1,2...d)}$ obtained from Algorithm 4 only requires the storage complexity of $\mathcal{O}(NIR)$, with R stands for the maximal rank value. The

diagrammatic notation of TT format $\mathbf{U}$ is depicted in Figure A-3(b).

The TT cores of the symmetric Volterra tensor can be obtained by the MPP. The MPP depends on thin SVD performed on $\mathcal{U}^{(1)} \in \mathbb{R}^{1\times N\times I\times R_2}$ that is explained in Algorithm 5.

Algorithm 5: Thin SVD on $\mathcal{U}^{(1)}$ [9]

Input: TT cores $\mathcal{U}^{(1,2\ldots d)}$ from Algorithm 4

1 $\boldsymbol{U}_1 \leftarrow$ reshape $\left(\mathcal{U}^{(1)}, [N, IR_2]\right)$

2 $[\mathbf{Q}_1, \mathbf{S}, \mathbf{H}] \leftarrow$ *Compute* SVD *of matrix* $(\boldsymbol{U}_1)$

3 *Truncate* $\mathbf{Q}_1, \mathbf{S}, \mathbf{H}$ *to the rank* $R = \begin{pmatrix} d+I-1 \\ I-1 \end{pmatrix}$

4 $\mathcal{H}^{(1)} \leftarrow$ reshape $(\mathbf{H}, [1, R, I, R_2])$

5 $\mathcal{H}^{(k)} \leftarrow \mathcal{U}^{(k)} (k = 2, \ldots, d)$

Output: Components matrices $\mathbf{S}$, $\mathbf{Q}_1$, and $\mathcal{H}^{(1)}$

The thin SVD of $\mathbf{U}_1 \in \mathbb{R}^{N\times IR_2}$ in the 2^{nd} line of 5 is expressed as follows,

$$[\mathbf{Q}_1 \in \mathbb{R}^{N\times R}, \mathbf{S} \in \mathbb{R}^{R\times R}, \mathbf{H} \in \mathbb{R}^{R\times IR_2}] = SVD(\mathbf{U}_1) \tag{A-23}$$

where $\mathbf{Q}_1, \mathbf{H}$ are orthogonal matrices and $\mathbf{S}$ is a diagonal matrix.

The TT format of symmetric Volterra tensor in Equation A-9 can be obtained by the MPP, expressed as

$$\begin{aligned} \mathcal{V}^{(1)}_{sym} \in \mathbb{R}^{1\times I\times R_1} &= \mathcal{H}^{(1)}\mathbf{S}^{-1}\mathbf{Q}_1^T\mathbf{y} \\ \mathcal{V}^{(k)}_{sym} \in \mathbb{R}^{R_{k-1}\times I\times R_k} &= \mathcal{H}^{(k)}, \quad \textit{for k=[2,d]} \end{aligned} \tag{A-24}$$

whose diagrammatic notation is visualized in Figure A-3 (c).

The storage complexity of MPP framework is the summation of the size of $\mathbf{y} \in \mathbb{R}^N$, $\mathcal{U}^{(1)} \in \mathbb{R}^{NIR_1}$, $\mathcal{V}^{(1)}_{sym} \in \mathbb{R}^{IR_1}$, $\sum_{k=2}^{k=d} \mathcal{U}^{(k)} \in \mathbb{R}^{(d-1)IR^2}$, $\sum_{k=2}^{k=d} \mathcal{V}^{(k)}_{sym} \in \mathbb{R}^{(d-1)IR^2}$, $\mathcal{H}^{(1)} \in \mathbb{R}^{1\times R\times I\times R_2}$, $\mathbf{S} \in \mathbb{R}^{R\times R}$ and $\mathbf{Q}_1 \in \mathbb{R}^{N\times R}$. The storage complexity is approximated as $\mathcal{O}\left(N + IR(N+1) + 2(d-1)R^2I + NR + R^2(I+1)\right)$, with R represents the maximal TT rank value.

A-3 Persistently Exciting Condition

The condition for the input measurement $\mathbf{U}$ in Equation 2-7 being persistently exciting of order d is defined in Lemma 1.

Lemma 1: Persistent Exciting Input Rank Condition [11]

The input sequence in matrix $\mathbf{U}$ is called persistent exciting inputs of order d if the column rank of $\tilde{\mathbf{U}}$ in **??** equals $\begin{pmatrix} pM+d \\ pM \end{pmatrix}$.

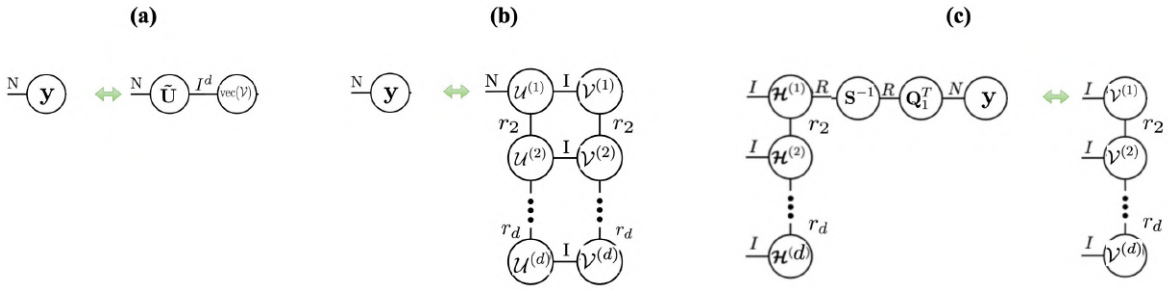


Figure A-3: Diagrammatic notation of (a) Equation 2-7 (b) TT decomposition of $\tilde{\mathbf{U}}$ and $\text{vec}\left(\mathcal{V}^{(k)}\right)$ where $\mathcal{U}^{(k)}$ are constructed from Algorithm 4; (c) constructing $\mathcal{V}_{sym}^{(k)}$ from Algorithm 5. Image modified on [9].

.

Proof: The vector $\mathbf{u}_t$ is polynomial in pM variables of degree d. With $\mathbf{u}_t^{@} \in \mathbb{R}^{(pM+1)^d} := \mathbf{u}_t \otimes \mathbf{u}_t \otimes \cdots \otimes \mathbf{u}_t$ denoting the d-times repeated Kronecker product of vector $\mathbf{u}_t$, there are only $\begin{pmatrix} pM+d \\ pM \end{pmatrix}$ entries of $\mathbf{u}_t^{@}$ being unique. Therefore, the rank of $\left((\mathbf{u}_0^{@})^T, (\mathbf{u}_1^{@})^T, \ldots, (\mathbf{u}_{N-1}^{@})^T\right)^T$, denoted as $\tilde{\mathbf{U}}$, equals to $\begin{pmatrix} pM+d \\ pM \end{pmatrix}$.

Appendix B

Algebraic Initialization Method

B-1 Newton Type Algorithms Examples

Several Newton type algorithms estimate the Hessian $\mathbf{H}_k$ term in Equation 2-23 in the following manners,

1. For gradient or steepest descent algorithm, $\mathbf{H}_k = \mathbf{I}$ and the step direction is simply the direction of the steepest slope.
2. For nonlinear conjugate gradient method algorithm,$\mathbf{H}_k = \mathbf{I} - \gamma \mathbf{p}_{k-1}\boldsymbol{\delta}^{\mathrm{T}}$, i.e., the identity plus a rank-1 correction. The values γ and δ depend on the current and previous step $\mathbf{p}_{k-1}$.
3. For Gauss-Newton algorithm, $\mathbf{H}_k = \mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k$ with $\mathbf{J}_k$ denoting the Jacobian matrix, and $\mathbf{H}_k$ is then called the Gramian (of the Jacobian).
4. For Levenberg-Marquardt algorithm, $\mathbf{H}_k = \mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k + \lambda\mathbf{I}$ with $\mathbf{J}_k$ the Jacobian matrix and $\lambda \geq 0$ a chosen constant.

B-2 Algebraic Initialization Method For Algorithm 1

The result of the algebraic method proposed in [22] that solves Equation 2-14 can be used as the initial value for Algorithm 1. This section aims to explain how the computational complexity of the algebraic method is calculated. The reader is advised to consult [22] for the detail derivation. For simplicity, the LS-CPD in Equation 2-14 is referred as follows:

$$\begin{aligned} \mathbf{y} \in \mathbb{R}^N = \begin{pmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{pmatrix} &= \left(\mathbf{U} \odot^{\mathrm{T}} \mathbf{U} \odot^{\mathrm{T}} \cdots \odot^{\mathrm{T}} \mathbf{U}\right) \mathrm{vec}(\mathcal{V}) \\ &= \mathbf{A}\mathbf{t} \end{aligned} \tag{B-1}$$

where $\mathbf{A} \in \mathbb{R}^{N \times I^d}$ denotes the d-time row-wise Khatri-rao products of $\mathbf{U}$, and $\mathbf{t} \in \mathbb{R}^{I^d}$ represents the vectorization of Volterra tensor $\text{vec}(\mathcal{V}) \in \mathbb{R}^{I^d}$.

If $\mathbf{A}$ has full column rank, the naive technique for determining the CPD factors of tensor $\mathcal{V}$ is to solve the linear system $\mathbf{t} = \mathbf{A}^{-1}\mathbf{y}$, reshape (tensorization [26]) $\mathbf{t}$ into tensor $\mathcal{V}$, and then decompose $\mathcal{V}$ into CPD format using the MATLAB ® script *cpd.m* in the Tensorlab package [35], expressed as:

$$\begin{aligned}[\mathbf{B}, \mathbf{c}] &= \text{CPD}(\mathcal{V}) \\ &= \text{CPD}(\text{tensorization}(\mathbf{t})) \\ &= \text{CPD}\left(\text{tensorization}\left(\mathbf{A}^{-1}\mathbf{y}\right)\right)\end{aligned} \tag{B-2}$$

where $\mathbf{B} = [\mathbf{B}^1, \cdots, \mathbf{B}^d]$ is the factor matrices and $\mathbf{c}$ is the weighting vector, which are defined in Equation 2-12, and becomes symmetric CPD if $\mathbf{B}^i$ are same for $i = [1, d]$. Because $\mathbf{A}$ contains repeated columns and does not have full column rank, the calculation of the inverse of $\mathbf{A}$ directly is ill-conditioned.

The algebraic method adopts the Cholesky decomposition [31] method to solve the inverse of $\mathbf{A}$ expressed as:

$$\begin{aligned}\mathbf{y} &= \mathbf{A}\mathbf{t} \\ \mathbf{y} &= \mathbf{W}\mathbf{P}\mathbf{t}, \quad \text{with } \mathbf{W} \in \mathbb{R}^{N \times F}, \text{ and } \mathbf{P} = \{0,1\}^{F \times I^d} \in \mathbb{R}^{F \times I^d} \\ \mathbf{y} &= \mathbf{W}\mathbf{z}, \quad \text{with } \mathbf{z} = \mathbf{P}\mathbf{t} \\ \mathbf{W}^T\mathbf{y} &= \mathbf{W}^T\mathbf{W}\mathbf{z} \\ \mathbf{W}^T\mathbf{y} &= \mathbf{c}\mathbf{z}, \quad \text{with } \mathbf{c} = \text{chol}(\mathbf{W}^T\mathbf{W}) \\ \mathbf{c}^{-1}\mathbf{W}^T\mathbf{y} &= \mathbf{z} \\ \mathbf{c}^{-1}\mathbf{W}^T\mathbf{y} &= \mathbf{P}\mathbf{t} \\ \mathbf{P}^+\mathbf{c}^{-1}\mathbf{W}^T\mathbf{y} &= \mathbf{t} \\ \mathbf{P}^+\left(\mathbf{W}^T\mathbf{W}\right)^{-1}\mathbf{W}^T\mathbf{y} &= \mathbf{t} \\ [\mathbf{B}, \mathbf{c}] &= \text{CPD}(\text{tensorization}(\mathbf{t}))\end{aligned} \tag{B-3}$$

where $\mathbf{W}$ is the matrix containing the unique columns of $\mathbf{A}$. The value of F equals to the rank of $\mathbf{A}$ that is $\begin{pmatrix} I-1+d \\ I-1 \end{pmatrix}$. $\mathbf{P}$ is the binary matrix containing the location of $\mathbf{W}$ in $\mathbf{A}$. $\mathbf{P}^+$ is the pseudoinverse of $\mathbf{P}$.

The first 6 lines are the common Cholesky decomposition method to solve $\mathbf{z}$, which requires the computational complexity of $\mathcal{O}\left(\frac{1}{2}NIF^2 + \frac{1}{6}F^3 + NF\right)$, and the computational complexity of last three lines are dominated by the pseudoinverse of $\mathbf{P}$ with $\mathcal{O}\left(F^3\right)$. In total, the computational complexity of Equation B-3 is $\mathcal{O}\left(\frac{1}{2}NIF^2 + \frac{7}{6}F^3 + NF\right)$.

The result is supposed to be symmetric CPD, thus the last step of the algebraic method is to check the identity of the resultant factor matrices $\mathbf{B}^i$. By defining $\mathbf{B}_1 = [\mathbf{B}^1, \cdots, \mathbf{B}^{d-1}]$,

and $\mathbf{B}_{N-1} = [\mathbf{B}^2, \cdots, \mathbf{B}^d]$, the identity of the resultant factor matrices can be calculated by the relative difference in Frobenius norm between $\mathbf{B}_1$ and $\mathbf{B}_{N-1}$, mathematically expressed as:

$$\frac{\sum_p \|\mathbf{B}_1 - \text{ permute } (\mathbf{B}_{N-1}, \pi)\|_F}{\|\mathbf{B}_1\|_F} < \text{SymmetryTolerance} \tag{B-4}$$

where π is the index permutation and scaling matrix such that each element of $\mathbf{B}_1$ is optimally permuted and scaled to fit the $\mathbf{B}_{N-1}$, $\|\|_F$ is the Frobenius norm, and the SymmetryTolerance has the default value of 10^{-6}. If Equation B-4 does not hold, which implies that the algebraic method does not find a suitable symmetric solution, the pseudo random factor matrix and a weighting vector for symmetric CPD are returned as the output of the algebraic method, using the MATLAB ® script *cpd_rnd.m* in the Tensorlab package [35].

The computational cost of the algebraic method is dominated by Equation B-3 as $\mathcal{O}\left(\frac{1}{2}NIF^2 + \frac{7}{6}F^3 + NF\right)$. The storage complexity is dominated by the three main components $\mathbf{P}$, $\mathbf{Z}$ and $\mathbf{y}$ in Equation B-3. The total size of these three components is $\mathcal{O}\left(FI^d + NF + N\right)$, which can be approximated as $\mathcal{O}\left(FI^d\right)$ given that $I^d >> N > F$.

Bibliography

[1] Oliver Nelles. *Nonlinear System Identification: From Classical Approaches to Neural Networks, Fuzzy Models, and Gaussian Processes*. Springer Nature, 2020.

[2] Tohru Katayama et al. *Subspace methods for system identification*. Vol. 1. Springer, 2005.

[3] Michel Verhaegen and Vincent Verdult. *Filtering and system identification: a least squares approach*. Cambridge university press, 2007.

[4] Constantin Paleologu, Jacob Benesty, and Silviu Ciochină. "Linear system identification based on a Kronecker product decomposition". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.10 (2018), pp. 1793–1808.

[5] Kim Batselier, Ching-Yun Ko, and Ngai Wong. "Tensor network subspace identification of polynomial state space models". In: *Automatica* 95 (2018), pp. 187–196.

[6] Rolf Isermann and Marco Münchhof. *Identification of dynamic systems: an introduction with applications*. Vol. 85. Springer, 2011.

[7] Angelo Guerraggio and Giovanni Paoloni. *Vito Volterra*. Vol. 15. Springer-Verlag, 2010.

[8] Wim Van Drongelen. *Signal processing for neuroscientists*. Academic press, 2018.

[9] Kim Batselier. "Enforcing symmetry in tensor network MIMO Volterra identification". In: *IFAC-PapersOnLine* 54.7 (2021), pp. 469–474.

[10] Kim Batselier et al. "A tensor-based volterra series black-box nonlinear system identification and simulation framework". In: *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2016, pp. 1–7.

[11] Kim Batselier, Zhongming Chen, and Ngai Wong. "Tensor Network alternating linear scheme for MIMO Volterra system identification". In: *Automatica* 84 (2017), pp. 26–35.

[12] Francis J Doyle, Ronald K Pearson, and Babatunde A Ogunnaike. *Identification and control using Volterra models*. Springer, 2002.

[13] Li Tan and Jean Jiang. "Adaptive Volterra filters for active control of nonlinear noise processes". In: *IEEE Transactions on signal processing* 49.8 (2001), pp. 1667–1676.

[14] Chi-Hao Cheng and Edward J Powers. "Optimal Volterra kernel estimation algorithms for a nonlinear communication system for PSK and QAM inputs". In: *IEEE Transactions on Signal processing* 49.1 (2001), pp. 147–163.

[15] Edward Bedrosian and Stephen O Rice. "The output properties of Volterra systems (nonlinear systems with memory) driven by harmonic and Gaussian inputs". In: *Proceedings of the IEEE* 59.12 (1971), pp. 1688–1707.

[16] Martijn P Vlaar et al. "Modeling the nonlinear cortical response in EEG evoked by wrist joint manipulation". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 26.1 (2017), pp. 205–215.

[17] Mario Köppen. "The curse of dimensionality". In: *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)*. Vol. 1. 2000, pp. 4–8.

[18] Gérard Favier, Alain Y Kibangou, and Thomas Bouilloc. "Nonlinear system modeling and identification using Volterra-PARAFAC models". In: *International Journal of Adaptive Control and Signal Processing* 26.1 (2012), pp. 30–53.

[19] Johan Håstad. "Tensor rank is NP-complete". In: *Journal of Algorithms* 11.4 (1990), pp. 644–654.

[20] Frank L Hitchcock. "The expression of a tensor or a polyadic as a sum of products". In: *Journal of Mathematics and Physics* 6.1-4 (1927), pp. 164–189.

[21] Evrim Acar, Daniel M Dunlavy, and Tamara G Kolda. "A scalable optimization approach for fitting canonical tensor decompositions". In: *Journal of Chemometrics* 25.2 (2011), pp. 67–86.

[22] Martijn Boussé et al. "Linear systems with a canonical polyadic decomposition constrained solution: Algorithms and applications". In: *Numerical Linear Algebra with Applications* 25.6 (2018), e2190.

[23] Dana Lahat, Tülay Adali, and Christian Jutten. "Multimodal data fusion: an overview of methods, challenges, and prospects". In: *Proceedings of the IEEE* 103.9 (2015), pp. 1449–1477.

[24] Anh Huy Phan, Petr Tichavsky, and Andrzej Cichocki. "On fast computation of gradients for CANDECOMP/PARAFAC algorithms". In: *arXiv preprint arXiv:1204.1586* (2012).

[25] N Vervliet and L De Lathauwer. "Numerical optimization-based algorithms for data fusion". In: *Data Handling in Science and Technology*. Vol. 31. Elsevier, 2019, pp. 81–128.

[26] Ulrich Schollwöck. "The density-matrix renormalization group in the age of matrix product states". In: *Annals of physics* 326.1 (2011), pp. 96–192.

[27] Tamara G Kolda and Brett W Bader. "Tensor decompositions and applications". In: *SIAM review* 51.3 (2009), pp. 455–500.

[28] Andrzej Cichocki et al. "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions". In: *Foundations and Trends® in Machine Learning* 9.4-5 (2016), pp. 249–429.

[29] Andrzej Cichocki et al. "Tensor decompositions for signal processing applications: From two-way to multiway component analysis". In: *IEEE signal processing magazine* 32.2 (2015), pp. 145–163.

[30] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. "Tensors for data mining and data fusion: Models, applications, and scalable algorithms". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 8.2 (2016), pp. 1–44.

[31] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[32] Michiel Vandecappelle, Nico Vervliet, and Lieven De Lathauwer. "A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost". In: *IEEE Transactions on Signal Processing* 68 (2020), pp. 4454–4465.

[33] Martijn Boussé and Lieven De Lathauwer. "Nonlinear least squares algorithm for canonical polyadic decomposition using low-rank weights". In: *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. IEEE. 2017, pp. 1–5.

[34] Michiel Vandecappelle, Nico Vervliet, and Lieven De Lathauwer. "Nonlinear least squares updating of the canonical polyadic decomposition". In: *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE. 2017, pp. 663–667.

[35] N. Vervliet et al. *Tensorlab 3.0*. Available online. Mar. 2016. URL: https://www.tensorlab.net.

[36] Martijn P Vlaar et al. "Modeling the nonlinear cortical response in EEG evoked by wrist joint manipulation". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 26.1 (2017), pp. 205–215.

[37] Mike de Pont. "Non-Linear Bayesian System Identification of Cortical Responses Using Volterra Series". In: (2020).

[38] Stijn Hendrikx et al. "Algebraic and optimization based algorithms for multivariate regression using symmetric tensor decomposition". In: *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. IEEE. 2019, pp. 475–479.

[39] Eva Memmel and Kim Batselier. "Bayesian tensor network-based Volterra system identification". In: *Delft University of Technology* (2021).

[40] Kim Batselier, Zhongming Chen, and Ngai Wong. "A Tensor Network Kalman filter with an application in recursive MIMO Volterra system identification". In: *Automatica* 84 (2017), pp. 17–25.

[41] "WORKSHOP ON NONLINEAR SYSTEM IDENTIFICATION BENCHMARKS". In: (Apr. 2021). Available online. URL: https://drive.google.com/file/d/1YE396kMM68NbrM_NRih58c_OtLgIy2t/view.

[42] SB Shiki et al. "Characterization of the nonlinear behavior of a F-16 aircraft using discrete-time Volterra series". In: *ISMA*. 2014.

[43] Carl Andersson et al. "Deep convolutional networks in system identification". In: *2019 IEEE 58th conference on decision and control (CDC)*. IEEE. 2019, pp. 3670–3676.

[44] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. "The alternating linear scheme for tensor optimization in the tensor train format". In: *SIAM Journal on Scientific Computing* 34.2 (2012), A683–A713.

Glossary

List of Acronyms

3mE	Mechanical, Maritime and Materials Engineering
LTI	Linear time-invariant
NTI	Nonlinear time-invariant
SISO	Single-input single-output
MIMO	Multiple-input multiple-output
TN	Tensor network
TT	Tensor train
PD	Polyadic decomposition
CPD	Canonical polyadic decomposition
LS	Least squares
LS-CPD	Linear system with a canonical polyadic decomposition constrained solution
SVD	Singular value decomposition
ALS	Alternating linear scheme
MALS	Modified alternating linear scheme
PARAFAC	Parallel factor model
MISO	Multiple input single output
CANDECOMP	Canonical decomposition
MPP	Moore-Penrose pseudoinverse
SNR	Signal-to-noise ratio
NLS	Nonlinear Least Square
GN	Gauss-Newton
CG	Conjugate gradients
CCR	Complexity compensation rates

PCR Performance compensation rate

EEG Electroencephalography

VAF Variance accounted for