A Khatri-Rao product tensor network for efficient symmetric MIMO Volterra identification

Batselier, Kim

DOI
10.1016/j.ifacol.2023.10.339

Publication date
2023

Document Version
Final published version

Published in
IFAC-PapersOnLine

Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 56-2 (2023) 7282–7287

A Khatri-Rao product tensor network for efficient symmetric MIMO Volterra identification

Kim Batselier *

* *Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands.*

Abstract: The identification of symmetric tensor network MIMO Volterra models has been studied earlier via the computation of a Moore-Penrose pseudoinverse in tensor network form. The current state of the art requires the construction of a tensor network of a repeated Khatri-Rao product of a matrix with itself. This construction has a computational complexity that is dominated by one singular value decomposition (SVD) of an $RI \times IN$ matrix, where N is the number of measurements, I depends linearly on the number of inputs and input lags and R is the maximal tensor network rank. In this article, we prove an alternative method for constructing this tensor network without any computation whatsoever. The pseudoinverse can then be computed through an orthogonalization of the newly proposed tensor network. Furthermore, the proposed algorithm allows for the recursive identification of symmetric Volterra models of increasing degree D, which reduces the computation to one SVD of a $RI \times N$ matrix per step. Through numerical experiments we demonstrate how the proposed algorithm enables up to ten times faster identification of symmetric tensor network MIMO Volterra systems.

Keywords: Nonlinear system identification, truncated Volterra systems, tensors

1. INTRODUCTION

Truncated Volterra series are a popular nonlinear model structure due to their conceptual simplicity as higher-order generalizations of finite impulse response models, which also makes them easier to interpret. The parametric identification of truncated Volterra series, however, suffers from the curse of dimensionality as the total number of parameters grows exponentially with the degree of the model. For example, using I time-lagged inputs the Dth Volterra kernel contains I^D parameters. As a consequence, learning these parameters from data comes with a computational complexity of $O(I^{3D})$, which limits the direct identification of multiple-input-multiple output (MIMO) truncated Volterra systems through standard methods to weakly-nonlinear systems.

One way to lift this curse of dimensionality is through solving the equivalent nonparametric dual problem by choosing a suitably parameterized kernel function (Birpoutsoukis et al., 2017, 2018; Dalla Libera et al., 2021). These methods suffer however from a computational complexity that is cubic in the number of measurements $O(N^3)$. Tensor decompositions, on the other hand, allow to solve the parametric primal problem iteratively through the addition of a low-rank constraint (Shi and Townsend, 2021). In Favier et al. (2012) both the canonical polyadic (Harshman, 1970; Carroll and Chang, 1970) and Tucker tensor decompositions (Tucker, 1966) were used to describe each Volterra kernel separately. A more convenient description is found in Batselier et al. (2017a,b); Batselier (2022), where tensor networks are used to describe all Volterra kernel coefficients at once through one tensor, which can be directly identified from data. Low-rank tensor networks also have the advantage that the complexity of estimating the model parameters scales linearly in both the degree D of the truncated Volterra series and in the number of measurements N. This linear scaling of the computational complexity opens up the possibility of identifying systems that are highly nonlinear and hence require high-degree approximations.

The problem of enforcing symmetry on the estimated Volterra kernels directly through low-rank tensor networks was recently solved in Batselier (2021). The main idea of the proposed solution for a degree-D Volterra system is to first compute a tensor train matrix of a D-times repeated Khatri-Rao product of a matrix $\tilde{\boldsymbol{U}}$ with itself. This computation can be done through Algorithm 2 (Batselier et al., 2018, p. 191) with a computational complexity of $O(NI^3R^2)$ and a storage complexity of $O(DIR^2)$, where N is the sample size, I depends linearly on the number of inputs P and input lags M, and R is the maximal rank of the resulting tensor network.

The main contribution of this article is the proof for an exact representation of this required tensor network that can be constructed without any computation and with a storage complexity of $O(NI)$. This efficient representation is key in enabling the computation of previously infeasible symmetric Volterra systems. A second contribution is an algorithm that computes the pseudoinverse of this new representation and allows for the recursive identification of Volterra models of increasing degree D with a computational complexity of $O(NI^2R^2)$ per step. Numerical experiments demonstrate the effectiveness of the proposed

10.1016/j.ifacol.2023.10.339

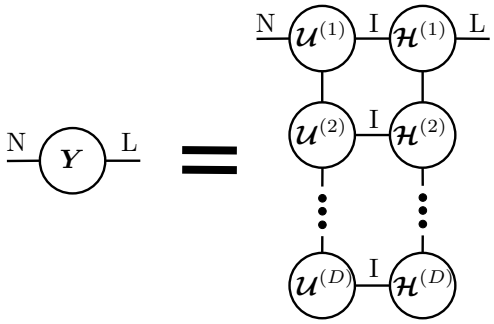

Fig. 1. Tensor diagram of equation (3). Both matrices $\boldsymbol{U}$ and $\boldsymbol{H}$ are represented by tensor train matrices. Row indices point to the left and column indices point to the right.

algorithm, where a speedup of up to factor of 10 is observed compared to the state of the art described in Batselier (2021).

2. TENSOR NETWORK MIMO VOLTERRA SYSTEMS

In this section a brief overview of the MIMO Volterra identification problem as described in Batselier et al. (2017a); Batselier (2022) is given. A multiple-input-multiple-output (MIMO) truncated Volterra model extends the single-input-single-output (SISO) linear finite impulse response model

$$y(n) = h_0 + \sum_{m_1=0}^{M} h_1(m_1)\, u(n-m_j) \tag{1}$$

to higher degrees of nonlinearity by adding homogeneous polynomials in the lagged inputs $u(n)$ to (1). Suppose there are L outputs and P inputs, which implies that

$$\boldsymbol{y}(n) = (y_1(n)\ y_2(n)\ \cdots\ y_L(n))^T \in \mathbb{R}^L,$$
$$\boldsymbol{u}(n) = (u_1(n)\ u_2(n)\ \cdots\ u_P(n))^T \in \mathbb{R}^P.$$

Defining the notation shorthand $I := (1 + (M+1)P)$ and using the same notation as in Batselier et al. (2017a), for a given memory (= input lags) M the vector $\boldsymbol{u}_n$ is defined as

$$\boldsymbol{u}_n := \left(1\ \boldsymbol{u}(n)^T\ \cdots\ \boldsymbol{u}(n-M)^T\right)^T \in \mathbb{R}^I.$$

All N vectors $\boldsymbol{u}_n$ can be collected into the $N \times I$ matrix

$$\tilde{\boldsymbol{U}} := \begin{pmatrix} \boldsymbol{u}_1^T \\ \boldsymbol{u}_2^T \\ \vdots \\ \boldsymbol{u}_N^T \end{pmatrix}.$$

All monomials in the lagged inputs from degree 0 up to D are constructed by taking the D-times repeated Kronecker product

$$\boldsymbol{u}_n^D := \overbrace{\boldsymbol{u}_n \otimes \boldsymbol{u}_n \otimes \cdots \otimes \boldsymbol{u}_n}^{D \text{ times}} \in \mathbb{R}^{I^D}.$$

The output of a MIMO Volterra system can then be written as

$$\boldsymbol{y}(n)^T = \left(\boldsymbol{u}_n^D\right)^T \boldsymbol{H}\,, \tag{2}$$

where column l of the matrix $\boldsymbol{H} \in \mathbb{R}^{I^D \times L}$ contains all Volterra kernel coefficients from degree 0 up to D responsible for output l. Writing out equation (2) for $n = 1, \ldots, N$ results in a system of N linear equations

$$\underbrace{\boldsymbol{Y}}_{N\times L} = \underbrace{\boldsymbol{U}}_{N\times I^D}\ \underbrace{\boldsymbol{H}}_{I^D\times L}. \tag{3}$$

The nth row of $\boldsymbol{U}$ is the D-times repeated Kronecker product of $\boldsymbol{u}_n$, which implies that $\boldsymbol{U}$ can be written as the D-times repeated row-wise Khatri-Rao product

$$\boldsymbol{U} = \overbrace{\tilde{\boldsymbol{U}} \odot \tilde{\boldsymbol{U}} \odot \cdots \odot \tilde{\boldsymbol{U}}}^{D \text{ times}}. \tag{4}$$

It is proven in Proposition 4.1 of Batselier et al. (2017a) that this Khatri-Rao structure of $\boldsymbol{U}$ results in a rank-deficiency, which implies that the linear system (3) has infinitely many solutions. Some form of regularization is therefore recommended in order to select a particular solution.

One way of regularizing the problem is the symmetric Volterra system identification problem, which is to solve equation (3) for the unknown $\boldsymbol{H}$ matrix with the constraint that each column l of $\boldsymbol{H}$ is the vectorization of a symmetric tensor $\boldsymbol{\mathcal{H}}_l$. A symmetric tensor $\boldsymbol{\mathcal{H}}_l \in \mathbb{R}^{I\times\cdots\times I}$ is a D-dimensional array for which

$$\boldsymbol{\mathcal{H}}_l(i_1, i_2, \ldots, i_D) = \boldsymbol{\mathcal{H}}_l(\pi(i_1, i_2, \ldots, i_D)),$$

where $\pi(i_1, i_2, \ldots, i_D)$ denotes any possible permutation of the D indices. It was proven in Proposition 4.2 of Batselier et al. (2017a) that the minimum norm solutions of (3) are the desired symmetric tensors. Therefore, the desired symmetric Volterra models can be obtained by solving the linear system (3) using the pseudoinverse of $\boldsymbol{U}$.

However, the exponential number I^D columns of $\boldsymbol{U}$ and rows of $\boldsymbol{H}$ quickly make it infeasible to explicitly construct these matrices and compute the pseudoinverse of $\boldsymbol{U}$. This curse of dimensionality is lifted by replacing these matrices by particular tensor networks called tensor train matrices (Oseledets, 2010, 2011), as shown in the tensor diagram of Figure 1. In a tensor diagram each node represents a tensor, and each edge represents an index. An edge that connects two nodes represents a summation over that particular index.

The tensor train matrix of the $N \times I^D$ matrix $\boldsymbol{U}$ splits the single column index i of size I^D into D indices $i_1, i_2, \ldots, i_D$, each of size I, over D separate tensors

$$\boldsymbol{\mathcal{U}}^{(d)} \in \mathbb{R}^{R_d \times N_d \times I \times R_{d+1}}.$$

These D separate indices $1 \leq i_d \leq I$ $(1 \leq d \leq D)$ relate to the original column index i through

$$i = [i_1 i_2 \ldots i_D] := i_1 + \sum_{d=2}^{D} (i_d - 1)\ I^{d-1}.$$

The storage complexity of $\boldsymbol{U}$ is reduced in this way from NI^D to the sum of the storage complexities of each of the node tensors. It follows from Figure 1 that $N_1 = N$ and $N_2 = \cdots = N_D = 1$. The dimensions R_d $(1 \leq d \leq D+1)$ are called the ranks of the tensor train matrix. As shown in Figure 1 by the connected edges, the indices corresponding with these ranks are all summed over. A similar tensor train matrix structure is used for the representation of the matrix $\boldsymbol{H}$.

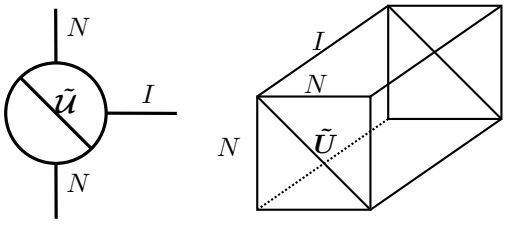

Fig. 2. The figure on the left is the tensor diagram of the 3-way tensor $\tilde{\mathcal{U}}$ constructed from a matrix $\tilde{\boldsymbol{U}}$. The figure on the right shows how the nonzero elements of $\tilde{\mathcal{U}}$ are the matrix $\tilde{\boldsymbol{U}}$ fixed as a diagonal slice of the tensor.

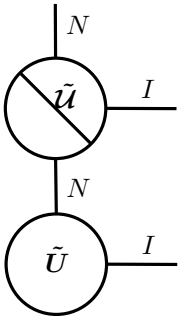

Fig. 3. Tensor network diagram of the tensor train matrix that represents $\tilde{\boldsymbol{U}} \odot \tilde{\boldsymbol{U}} \in \mathbb{R}^{N \times I^2}$.

3. EXACT TENSOR TRAIN MATRIX OF REPEATED KHATRI-RAO PRODUCT

In Batselier (2021) the symmetric solution for $\boldsymbol{H}$ is computed via the pseudoinverse of $\boldsymbol{U}$ in tensor train form. A first and necessary step is the computation of the tensor train matrix of $\boldsymbol{U}$ via Algorithm 2 (Batselier et al., 2018, p. 191), which computes repeated Khatri-Rao products together with truncated singular value decompositions (SVDs). The total computational complexity to construct the tensor train matrix of $\boldsymbol{U}$ in this way is $O(N^2 I^3 R)$. In this article we replace the construction of this tensor train matrix through an exact representation that involves no computation whatsoever as only instance of the matrix $\tilde{\boldsymbol{U}}$ needs to be stored. The following 3-way tensor constructed from a matrix plays a crucial role in the representation.

Definition 1. For a given matrix $\tilde{\boldsymbol{U}} \in \mathbb{R}^{N \times I}$ we define the 3-way tensor $\tilde{\mathcal{U}} \in \mathbb{R}^{N \times I \times N}$ such that

$$\tilde{\mathcal{U}}(n, i, n) = \tilde{\boldsymbol{U}}(n, i),$$

and all other elements of $\tilde{\mathcal{U}}$ are exactly zero.

Figure 2 illustrates how all nonzero elements of $\tilde{\mathcal{U}}$ are the matrix $\tilde{\boldsymbol{U}}$ itself, placed on the diagonal slice that is obtained from restricting the first and third index of $\tilde{\mathcal{U}}$ to be the same value n.

Before we can state the main theorem we first need the following lemma that relates the outer product of two vectors with the corresponding Kronecker product.

Lemma 1. (Golub and Van Loan, 2013, p.28) Let $\boldsymbol{u} \in \mathbb{R}^I$, then

$$\mathrm{vec}(\boldsymbol{u}\boldsymbol{u}^T) = \boldsymbol{u} \otimes \boldsymbol{u},$$

where vec() denotes the vectorization operation that concatenates all columns of a given matrix into a single vector.

We can now formulate the main theorem on which the exact tensor train matrix representation of the matrix $\boldsymbol{U}$ is based.

Theorem 1. Let $\boldsymbol{U}_2 := \tilde{\boldsymbol{U}} \odot \tilde{\boldsymbol{U}} \in \mathbb{R}^{N \times I^2}$, then

$$\boldsymbol{U}_2^T([i_1 i_2], n) = \sum_{n_1=1}^{N} \tilde{\boldsymbol{U}}(n_1, i_1)\, \tilde{\mathcal{U}}(n_1, i_2, n) \tag{5}$$

Proof. The index summation of equation (5) can be rewritten as the following matrix-matrix product

$$\tilde{\boldsymbol{U}}^T \tilde{\boldsymbol{U}}_{(1)} = (\boldsymbol{u}_1\ \boldsymbol{u}_2\ \cdots\ \boldsymbol{u}_N) \begin{pmatrix} \boldsymbol{u}_1^T & 0 & 0 & \cdots & 0 & 0 \\ 0 & \boldsymbol{u}_2^T & 0 & \cdots & 0 & 0 \\ & & & \ddots & & \\ 0 & 0 & \cdots & \cdots & 0 & \boldsymbol{u}_N^T \end{pmatrix}$$

where $\tilde{\boldsymbol{U}}_{(1)} \in \mathbb{R}^{N \times IN}$ is the matrix obtained from reshaping the tensor $\tilde{\mathcal{U}}$ along the first dimension. The diagonal matrix slice structure of $\tilde{\mathcal{U}}$ then results in a row-diagonal structure of $\tilde{\boldsymbol{U}}_{(1)}$ such that its nth main-diagonal vector is $\boldsymbol{u}_n^T$. Computing the matrix-matrix product results in the $I \times IN$ matrix

$$\left(\boldsymbol{u}_1\boldsymbol{u}_1^T\ \ \boldsymbol{u}_2\boldsymbol{u}_2^T\ \cdots\ \boldsymbol{u}_N\boldsymbol{u}_N^T\right),$$

which consists of N rank-1 matrix blocks of size $I \times I$ concatenated column-wise. Applying the vec() operator from Lemma 1 on each of the $I \times I$ rank-1 matrices $\boldsymbol{u}_n\boldsymbol{u}_n^T$ results in the $I^2 \times N$ matrix

$$(\boldsymbol{u}_1 \otimes \boldsymbol{u}_1\ \ \boldsymbol{u}_2 \otimes \boldsymbol{u}_2\ \cdots\ \boldsymbol{u}_N \otimes \boldsymbol{u}_N) = \boldsymbol{U}_2^T,$$

which concludes the proof.

The tensor diagram representation of Theorem 1 is shown in Figure 3. The summation over the n_1 index is indicated in the diagram by the connected edge between the two nodes. The extension of Theorem 1 to D factor matrices in the Khatri-Rao product is now straightforward. Since

$$\boldsymbol{U}_3 := \boldsymbol{U}_2 \odot \tilde{\boldsymbol{U}} \in \mathbb{R}^{N \times I^3},$$

we can apply Theorem 1 again by connecting the top edge of dimension N in Figure 3 to another $\tilde{\mathcal{U}}$ node. Repeated use of Theorem 1 for D factor matrices $\tilde{\boldsymbol{U}}$ results in the tensor diagram of $\boldsymbol{Y} = \boldsymbol{U}\,\boldsymbol{H}$ as shown in Figure 4. Contrary to Batselier (2021), the tensor train matrix representation of $\boldsymbol{U}$ requires no computation as it consists of D copies of $\tilde{\boldsymbol{U}}$ and $\boldsymbol{U}$ is never explicitly computed. It is therefore sufficient to store the matrix $\tilde{\boldsymbol{U}}$ only once in memory, which reduces the storage complexity of $\boldsymbol{U}$ from NI^D down to NI.

4. ALGORITHM TO COMPUTE PSEUDOINVERSE

Once the tensor train matrix of $\boldsymbol{U}$ is available, the next step in the identification of a symmetric Volterra model is the computation of the pseudoinverse $\boldsymbol{U}^\dagger$ of $\boldsymbol{U}$ through the thin SVD

$$\boldsymbol{U} = \boldsymbol{Q}\,\boldsymbol{S}\,\boldsymbol{V}^T,$$

where $\boldsymbol{Q} \in \mathbb{R}^{N \times R}, \boldsymbol{V} \in \mathbb{R}^{I^D \times R}$ have orthonormal columns and $\boldsymbol{S} \in \mathbb{R}^{R \times R}$ is diagonal. If the inputs are persistently

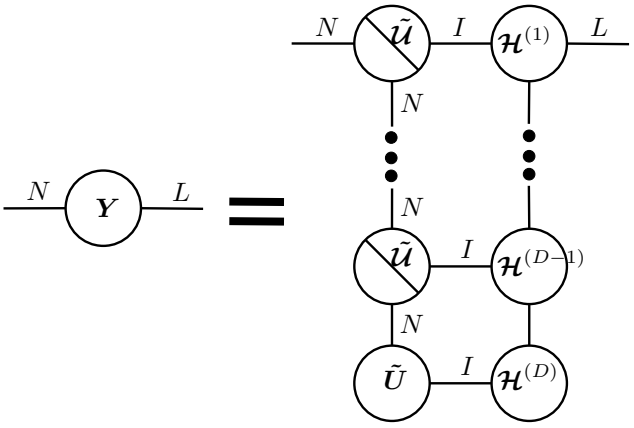

Fig. 4. Tensor diagram of equation (3) where the tensor train matrix of $\boldsymbol{U}$ has been constructed using Theorem 1. This construction involves no computation and has a total storage complexity of NI.

exciting of at least degree D, then it was proven in Batselier et al. (2017a) that $R = \binom{D+I-1}{I-1}$. The pseudoinverse is then

$$\boldsymbol{U}^{\dagger} = \boldsymbol{V}\ \boldsymbol{S}^{-1}\ \boldsymbol{Q}^{T},$$

from which the symmetric Volterra kernels can be computed in tensor train matrix form as $\boldsymbol{H} = \boldsymbol{U}^{\dagger}\ \boldsymbol{Y}$. The exponential number of rows of $\boldsymbol{V}$ means that this matrix needs to be computed in tensor train matrix form.

4.1 Proposed algorithm

With the newly proposed way to represent the tensor train matrix of $\boldsymbol{U}$ based on Theorem 1, also a new way of computing the pseudoinverse is required. This is now achieved through D orthogonalization steps. The whole algorithm is presented as pseudocode in Algorithm 1 and involves computing D SVDs, retaining the left singular vectors as $\boldsymbol{\mathcal{V}}^{(d)}$ and multiplying $\boldsymbol{SQ}^T$ with the next node of the tensor train matrix. It is always assumed that all inputs are persistently exciting of at least degree D.

Figure 5 illustrates the first orthogonalization of Algorithm 1 as a tensor network diagram. First, the thin SVD of $\tilde{\boldsymbol{U}}^T = \boldsymbol{V}\,\boldsymbol{S}\,\boldsymbol{Q}^T$ is computed. All zero singular values are truncated and the resulting matrix $\boldsymbol{V}$ is retained as the new first node in the network. The matrix factor $\boldsymbol{SQ}^T$is then multiplied with the second node $\tilde{\boldsymbol{\mathcal{U}}}$ and the process repeats.

The multiplication of $\boldsymbol{SQ}^T$ with $\tilde{\boldsymbol{\mathcal{U}}}$ is done via Theorem 1 as $\tilde{\boldsymbol{U}}^{\boldsymbol{T}} \odot \boldsymbol{SQ}^{\boldsymbol{T}}$. It is therefore not necessary to ever construct the tensors $\tilde{\boldsymbol{\mathcal{U}}}$ tensors during the algorithm. This process of orthogonalization and multiplying $\boldsymbol{SQ}^{\boldsymbol{T}}$ with the next node in the network results finally in the tensor train matrix for $\boldsymbol{V}$ and two matrices $\boldsymbol{S}$ and $\boldsymbol{Q}$ of the thin SVD. The most expensive computational step of Algorithm 1 is the SVD computation of the Dth matrix $\tilde{\boldsymbol{U}}^T \odot \boldsymbol{SQ}^{\boldsymbol{T}}$ with a complexity of $O(N^2 R_D I)$, since $IR_D > N$.

4.2 Alternatives to full SVD in Algorithm 1

When the input signals are persistently exciting of at least degree D, then according to Batselier (2021) the tensor train matrix-ranks of $\boldsymbol{U}$ are guaranteed to be

$$R_{d+1} = \binom{d+I-1}{I-1}\ (0 \leq d \leq D).$$

As a consequence, the full SVD of the $R_d I \times N$ matrix $\tilde{\boldsymbol{U}}^{\boldsymbol{T}} \odot \boldsymbol{SQ}^{\boldsymbol{T}}$ does not need to be computed since its rank is known to be R_{d+1}. Instead, a rank-R_{d+1} truncation can be computed directly with methods such as the implicitly restarted Lanczos method (Baglama and Reichel, 2005) or the randomized SVD (Halko et al., 2011). The computational benefit of using such methods is expected to be larger when the full SVD cannot be stored in memory anymore.

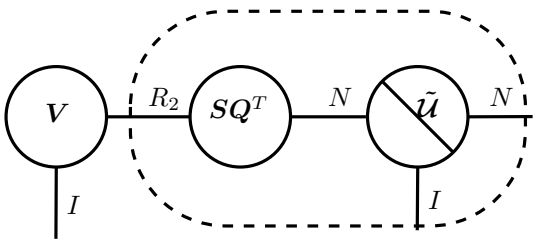

Fig. 5. Tensor network diagram of the first orthogonalization in Algorithm 1. The multiplication of $\boldsymbol{SQ}^T$ with the second node $\tilde{\boldsymbol{\mathcal{U}}}$, indicated by the dashed line, can be computed through Theorem 1 as $\tilde{\boldsymbol{U}}^{\boldsymbol{T}} \odot \boldsymbol{SQ}^{\boldsymbol{T}}$.

Algorithm 1 Pseudoinverse of $\boldsymbol{U}$

Input: $N \times I$ matrix $\tilde{\boldsymbol{U}}$, degree D of Volterra model
Output: $\boldsymbol{U}^{\dagger} = \boldsymbol{V}\ \boldsymbol{S}^{-1}\ \boldsymbol{Q}^T$

1: $[\boldsymbol{V}, \boldsymbol{S}, \boldsymbol{Q}] \leftarrow \text{SVD}(\tilde{\boldsymbol{U}}^T)$
2: Truncate $\boldsymbol{V}, \boldsymbol{S}, \boldsymbol{Q}$ to rank $R_2 = I$
3: $\boldsymbol{\mathcal{V}}^{(1)} \leftarrow \text{reshape}(\boldsymbol{V}, [1, I, I])$ % $R_1 = 1$
4: **for** $d = 2 : D$ **do**
5: $\quad[\boldsymbol{V}, \boldsymbol{S}, \boldsymbol{Q}] \leftarrow \text{SVD}(\tilde{\boldsymbol{U}}^{\boldsymbol{T}} \odot \boldsymbol{SQ}^{\boldsymbol{T}})$
6: $\quad$Truncate $\boldsymbol{V}, \boldsymbol{S}, \boldsymbol{Q}$ to rank $R_{d+1} = \binom{d+I-1}{I-1}$
7: $\quad\boldsymbol{\mathcal{V}}^{(d)} \leftarrow \text{reshape}(\boldsymbol{V}, [R_d, I, R_{d+1}])$
8: **end for**
9: Return tensor train matrix $\boldsymbol{\mathcal{V}}$ and matrices $\boldsymbol{S}^{-1}, \boldsymbol{Q}^T$

4.3 Efficiency for multiple experiments with increasing D

The identification of a symmetric MIMO Volterra tensor network model requires the specification of the two nonnegative integer-valued hyperparameters D and M. A common way to find suitable values for these hyperparameters is via a grid search. In the case where the value of M is fixed and D increases with unit increments, an additional computational gain can be achieved with Algorithm 1 since all computations for D can be reused for the case $D + 1$. The only computation required to update the results for D to D+1 is one Khatri-Rao product $\tilde{\boldsymbol{U}}^{\boldsymbol{T}} \odot \boldsymbol{SQ}^{\boldsymbol{T}}$, and one additional SVD with a total computational complexity of $O(N^2 R_{D+1} I)$. A major bottleneck of the algorithm is still the binomial increase of required N. Nevertheless, numerical experiments in Section 5 show that a speedup of up to ten times can be achieved via this recursive way.

5. NUMERICAL EXPERIMENTS

The main contribution of this article is an alternative method to the state of the art algorithm described in Batselier (2021) for constructing the tensor network of $\boldsymbol{U}$ and its pseudoinverse. For this reason we compare in this section the proposed Algorithm 1 to the aforementioned state of the art. All experiments were run on a desktop pc running an 8-core Intel Xeon processor at 3.60 GHz and with 16 GB RAM. A Matlab implementation to reproduce these experiments can be freely downloaded from `https://github.com/kbatseli/SymmetricVolterra`.

5.1 Incremental increase degree D experiment

We consider the identification of symmetric SISO Volterra systems with a fixed value $M = 5$ and where D is increased from 3 up to 10. A zero-mean Gaussian white noise signal is used as the input signal to ensure persistency of excitation for any degree D. The total number of samples is fixed to $N = \binom{10+6-1}{6-1} = 3003$ such that all systems up to degree 10 can be uniquely identified.

For each value of D the matrix $\boldsymbol{U}$ is then constructed in tensor network-form and its pseudoinverse is computed using the state of the art algorithm in Batselier (2021) and Algorithm 1. Figure 6 shows the mean runtime over 10 runs. Starting from $D = 7$ a corner can be observed in the mean runtime of Algorithm 1, which results in a ten times faster identification for $D = 10$. The numerical values of the mean runtimes are shown in Table 1, together with the total number of entries I^D of the identified Volterra tensor $\boldsymbol{\mathcal{H}}$ and a "Speedup" column that is obtained from dividing the mean runtime of (Batselier, 2021) by the mean runtime of Algorithm 1.

Table 1. Numerical values of mean runtimes over 10 runs that are visualized in Figure 6. SOTA stands for state of the art (Batselier, 2021). The column Speedup was computed from dividing column 2 by column 3.

Degree D	I^D	Runtime SOTA (s)	Runtime Algorithm 1 (s)	Speedup
3	216	.028	0.02	1.7
4	1296	.117	0.04	2.6
5	7776	.496	0.15	3.2
6	46656	1.89	0.78	2.4
7	279936	8.45	5.13	1.6
8	1679616	19.1	7.05	2.7
9	10077696	54.0	8.16	6.6
10	60466176	137	9.93	13.8

6. CONCLUSIONS

This article described a new representation of the tensor train matrix for $\boldsymbol{U}$ without any computation. Furthermore, an algorithm was provided to efficiently determine the pseudoinverse of this matrix for the identification of symmetric MIMO tensor network Volterra systems. Given the computational bottleneck of the binomial growth of the required number of measurements N due to the pseudoinverse, an alternative method would be to iteratively identify low-rank symmetric Volterra tensors $\boldsymbol{\mathcal{H}}_l$. Future work is then required on how to impose the symmetry directly during the optimizations. Other future work is on imposing other regularization structures such as a triangular structure onto the Volterra kernel coefficients in tensor network form.

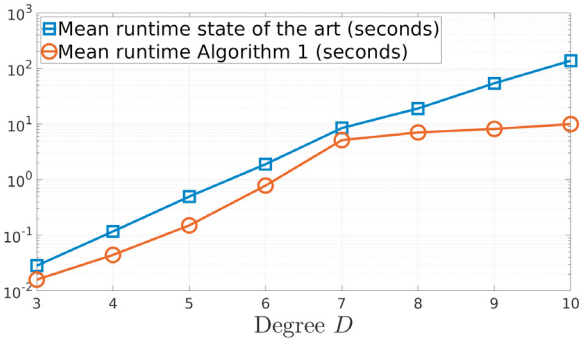

Fig. 6. The mean runtime over 10 runs to compute the pseudoinverse of $\boldsymbol{U}$ in tensor network form for a Volterra system with $M = 5$ and where D varies from 3 up to 10. The blue line with square markers is the mean runtime of the state of the art (Batselier, 2021). The red line with circle markers is the mean runtime of the proposed Algorithm 1. Starting from $D = 7$ there is a sudden change in growth of the average runtime, resulting in a speedup of 10 for $D = 10$.

REFERENCES

Baglama, J. and Reichel, L. (2005). Augmented implicitly restarted lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing*, 27(1), 19–42.

Batselier, K., Chen, Z.M., and Wong, N. (2017a). Tensor Network alternating linear scheme for MIMO Volterra system identification. *Automatica*, 84, 26–35.

Batselier, K., Chen, Z.M., and Wong, N. (2017b). A Tensor Network Kalman filter with an application in recursive MIMO Volterra system identification. *Automatica*, 84, 17–25.

Batselier, K. (2021). Enforcing symmetry in tensor network MIMO Volterra identification. *IFAC-PapersOnLine*, 54(7), 469–474.

Batselier, K. (2022). Low-Rank Tensor Decompositions for Nonlinear System Identification: A Tutorial with Examples. *IEEE Control Systems Magazine*, 42(1), 54–74.

Batselier, K., Ko, C.Y., and Wong, N. (2018). Tensor network subspace identification of polynomial state space models. *Automatica*, 95, 187–196.

Birpoutsoukis, G., Csurcsia, P.Z., and Schoukens, J. (2018). Efficient multidimensional regularization for Volterra series estimation. *Mechanical Systems and Signal Processing*, 104, 896–914.

Birpoutsoukis, G., Marconato, A., Lataire, J., and Schoukens, J. (2017). Regularized nonparametric Volterra kernel estimation. *Automatica*, 82, 324–327.

Carroll, J.D. and Chang, J.J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3), 283–319.

Dalla Libera, A., Carli, R., and Pillonetto, G. (2021). Kernel-based methods for volterra series identification. *Automatica*, 129, 109686.

Favier, G., Kibangou, A.Y., and Bouilloc, T. (2012). Nonlinear system modeling and identification using Volterra-PARAFAC models. *Int. J. Adapt. Control Signal Process*, 26(1), 30–53.

Golub, G.H. and Van Loan, C.F. (2013). *Matrix computations*. JHU press.

Halko, N., Martinsson, P.G., and Tropp, J.A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2), 217–288.

Harshman, R.A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16(1), 84.

Oseledets, I.V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5), 2295–2317.

Oseledets, I. (2010). Approximation of $2^d \times 2^d$ Matrices Using Tensor Decomposition. *SIAM J. Matrix Anal. Appl.*, 31(4), 2130–2145.

Shi, T. and Townsend, A. (2021). On the compressibility of tensors. *SIAM Journal on Matrix Analysis and Applications*, 42(1), 275–298.

Tucker, L.R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3), 279–311.