A Tensor-Based Volterra Series Black-Box Nonlinear System Identification And Simulation Framework

Kim Batselier
Department of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong
kimb@eee.hku.hk

Zhongming Chen
Department of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong
zmchen@eee.hku.hk

Haotian Liu
Cadence Design Systems, Inc.
USA
haotian@cadence.com

Ngai Wong
Department of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong
nwong@eee.hku.hk

ABSTRACT

Tensors are a multi-linear generalization of matrices to their d-way counterparts, and are receiving intense interest recently due to their natural representation of high-dimensional data and the availability of fast tensor decomposition algorithms. Given the input-output data of a nonlinear system/circuit, this paper presents a nonlinear model identification and simulation framework built on top of Volterra series and its seamless integration with tensor arithmetic. By exploiting partially-symmetric polyadic decompositions of sparse Toeplitz tensors, the proposed framework permits a pleasantly scalable way to incorporate high-order Volterra kernels. Such an approach largely eludes the curse of dimensionality and allows computationally fast modeling and simulation beyond weakly nonlinear systems. The black-box nature of the model also hides structural information of the system/circuit and encapsulates it in terms of compact tensors. Numerical examples are given to verify the efficacy, efficiency and generality of this tensor-based modeling and simulation framework.

Keywords

black box; Volterra series; nonlinear system identification; tensors; simulation

1. INTRODUCTION

Automatic system identification and model selection of a nonlinear system or circuit from a given set of input-output data is an important goal in electronic design automation (EDA). For linear systems this goal has been largely achieved, which is evident from the rich literature and many sophisticated algorithms that are available, e.g., [1, 2]. In contrast, it is a much more difficult task for nonlinear systems [3].

ICCAD '16 Doubletree Hotel, Austin, TX, USA

DOI: XX.XXX/XXX_X

This article makes a step forward towards this goal by introducing a tensor-based framework for the automatic identification and simulation of nonlinear systems and circuits with Volterra series. Our proposed framework is illustrated as a block diagram in Fig. 1. First, a set of Volterra kernels $\boldsymbol{\mathcal{H}}_1, \boldsymbol{\mathcal{H}}_2, \ldots, \boldsymbol{\mathcal{H}}_n$ are identified from a given set of measured discrete-time input/output data. These kernels are then used together with a new input to generate a simulated output of the nonlinear system. We present an algorithm for both the identification and simulation block in this article. The main tool used for modeling nonlinear systems in our framework is the Volterra series, which has been successfully applied to modeling and simulating weakly nonlinear systems, e.g., [4–7]. We show that there is an inherent link between Volterra series and tensors and exploit this link to develop a fast simulation algorithm. We remark that the integration of Volterra theory and tensors has appeared in previous works on nonlinear simulation [8–10] and in system identification [11]. The key novelties in this work are:

- **Representation of the Volterra kernels as tensors** – This is not an entirely new idea and has appeared in prior work. However, we introduce, for the first time in the literature, particular *Toeplitz tensors* that allow the development of a remarkably fast simulation algorithm via polyadic tensor decompositions. The obtained reduction in required simulation runtime and storage cost is demonstrated by means of numerical experiments.
- **Fast simulation by means of multi-mode convolution** – The curse of dimensionality appears in the standard implementation of Volterra kernels wherein multi-mode convolutions are needed in computing higher order responses. This has limited the application of Volterra series to mostly mildly nonlinear systems. Via polyadic tensor decompositions we manage to compute the multi-mode convolution by means of *only* linear convolutions, regardless of the Volterra kernel order. This means that strongly nonlinear systems can now be efficiently captured with high-order kernels whereby the dimensionality curse is gratefully removed during simulation.

This article is organized as follows. Section 2 reviews basic tensor notation and operations, followed by a succinct account on the Volterra series. The development of the nonlinear system identification and accelerated tensor-based simulation is elaborated in Sec-

tions 3 and 4. Section 5 presents three numerical examples demonstrating the efficacy of the proposed black-box modeling and simulation approach. Some further remarks are given in Section 6 and Section 7 draws the conclusions.

2. BACKGROUND

2.1 Tensors

First, we give a short introduction into the topic of tensors and the notation that we use. Tensors are denoted by boldface capital calligraphic letters (e.g. $\boldsymbol{\mathcal{A}}$), matrices by boldface capital letters (e.g. $\boldsymbol{A}$), vectors by boldface letters (e.g. $\boldsymbol{a}$), and scalars by either Roman (e.g. a) or Greek (e.g. α) letters. The nth element in a sequence is denoted by a superscript in parentheses. For example, $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \boldsymbol{u}^{(3)}$ denote the first three elements in a sequence of $\boldsymbol{u}$ vectors. Tensors are in our context generalizations of matrices and vectors. A d-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is simply a d-way array. Each element of the tensor $\boldsymbol{\mathcal{A}}$ is hence specified by d indices $i_1, \ldots, i_d$ and denoted by $a_{i_1 \cdots i_d}$. The positive integers $d, n_1, n_2, \ldots, n_d$ are called the order and the dimensions of the tensor, respectively. Fig. 2 illustrates a 3-way or 3rd-order tensor with dimensions $4, 3, 2$.

The matrix vector multiplication is generalized to the tensor case in the following way. The k-mode product of a d-way tensor $\boldsymbol{\mathcal{A}}$ with a vector $\boldsymbol{u} \in \mathbb{R}^{n_k}$ is defined by

$$(\boldsymbol{\mathcal{A}} \times_k \boldsymbol{u}^T)_{i_1 \cdots i_{k-1} i_{k+1} \cdots i_d} = \sum_{i_k=1}^{n_k} u_{i_k} a_{i_1 \cdots i_k \cdots i_d}.$$

This operation effectively removes the kth mode, resulting in a $(d-1)$-way tensor. Note that for a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and vector $\boldsymbol{u} \in \mathbb{R}^n$, we can write the familiar matrix vector products as $\boldsymbol{A} \times_1 \boldsymbol{u}^T \triangleq \boldsymbol{u}^T \boldsymbol{A}$ and as $\boldsymbol{A} \times_2 \boldsymbol{u}^T \triangleq \boldsymbol{A}\,\boldsymbol{u}$.

A d-way rank-1 tensor is the outer product of d vectors

$$\boldsymbol{\mathcal{A}} = \boldsymbol{a}^{(1)} \circ \boldsymbol{a}^{(2)} \circ \cdots \circ \boldsymbol{a}^{(d)},$$

where $\boldsymbol{a}^{(1)} \in \mathbb{R}^{n_1}, \ldots, \boldsymbol{a}^{(d)} \in \mathbb{R}^{n_d}$. The entries of $\boldsymbol{\mathcal{A}}$ are completely determined by $a_{i_1 i_2 \cdots i_d} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(d)}_{i_d}$.

A cubical tensor is a tensor for which $n_1 = n_2 = \cdots = n_d$. A cubical tensor $\boldsymbol{\mathcal{A}}$ is symmetric if $a_{i_1 \cdots i_d} = a_{\pi(i_1, \ldots, i_d)}$ where $\pi(i_1, \ldots, i_d)$ is any permutation of the indices and a Toeplitz tensor if $a_{i_1 \cdots i_d} = a_{i_1+1 \cdots i_d+1}$ holds for all entries.

The Kronecker product [12, 13] and Hadamard product are denoted by $\otimes$ and by $\odot$ respectively. We introduce the shorthand notation $x^d \triangleq x \otimes x \otimes \cdots \otimes x$ for the d-times repeated Kronecker product.

An important operation on tensors is reshaping. The most common reshapings are the matricization and vectorization [14], which reorder the entries of $\boldsymbol{\mathcal{A}}$ into a matrix $\boldsymbol{A}$ and vector vec($\boldsymbol{\mathcal{A}}$). The mode-n matricization of a tensor $\boldsymbol{\mathcal{A}}$ rearranges the entries of $\boldsymbol{\mathcal{A}}$ such that the rows of the resulting matrix are indexed by the nth tensor index i_n. The remaining indices are grouped in ascending order. The only reshapings used in this paper are the mode-1 matricization and vectorization.

EXAMPLE 1. *The mode-1 matricization of the tensor $\boldsymbol{\mathcal{A}}$ from Fig. 2 is*

$$\boldsymbol{A} = \begin{pmatrix} 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \\ 4 & 8 & 12 & 16 & 20 & 24 \end{pmatrix}.$$

and its vectorization is

$$vec(\boldsymbol{\mathcal{A}}) = \begin{pmatrix} 1 & 2 & \cdots & 24 \end{pmatrix}^T.$$

The importance of the matricization and vectorization lies in the following two equations

$$\boldsymbol{\mathcal{A}} \times_1 \boldsymbol{u}^T \times_2 \cdots \times_n \boldsymbol{u}^T = \text{vec}(\boldsymbol{\mathcal{A}})^T \boldsymbol{u}^d \tag{1}$$

and

$$\boldsymbol{\mathcal{A}} \times_2 \boldsymbol{u}^T \times_3 \cdots \times_n \boldsymbol{u}^T = \boldsymbol{A}\,\boldsymbol{u}^{d-1}, \tag{2}$$

which tell us how the k-mode products of a tensor $\boldsymbol{\mathcal{A}}$ with a vector $\boldsymbol{u}$ can be computed.

A polyadic decomposition [15, 16] of a d-way tensor $\boldsymbol{\mathcal{A}}$ is its decomposition into a sum of R rank-1 tensors

$$\boldsymbol{\mathcal{A}} = \sum_{i=1}^{R} \boldsymbol{a}_i^{(1)} \circ \boldsymbol{a}_i^{(2)} \circ \cdots \circ \boldsymbol{a}_i^{(d)}.$$

In order to store a d-way tensor $\boldsymbol{\mathcal{A}}$ in memory, we need to store all of its $n_1 \times \cdots \times n_d$ elements. In contrast, storing its polyadic decomposition needs only $R(n_1 + \cdots + n_d)$ elements, which can be quite a reduction when R is small. When R is minimal, the polyadic decomposition is called canonical. Probably the most well-known canonical polyadic decomposition of a matrix is its singular value decomposition [17]. A symmetric tensor $\boldsymbol{\mathcal{A}}$ can always be decomposed into a symmetric polyadic decomposition. This means that every rank-1 term is also a symmetric tensor and therefore

$$\boldsymbol{\mathcal{A}} = \sum_{i=1}^{R} \boldsymbol{a}_i \circ \boldsymbol{a}_i \circ \cdots \circ \boldsymbol{a}_i.$$

Storage of a symmetric polyadic decomposition therefore needs only Rn elements when $\boldsymbol{a}_i \in \mathbb{R}^n$.

2.2 Volterra Series

Volterra theory has been developed over a century ago and has found applications in analyzing communication systems and nonlinear control [4, 5]. The object of interest in this theory are the Volterra series, which can be regarded as a Taylor series with memory effects since its evaluation at a particular time instance requires information from the past. Specifically, a nonlinear discrete-time time-invariant system with an input $u(t) \in \mathbb{R}$ and an output $y(t) \in \mathbb{R}$ is described by a Volterra series as

$$y(t) = y_1(t) + y_2(t) + y_3(t) + \cdots,$$

where

$$y_n(t) = \sum_{k_1=0}^{M-1} \cdots \sum_{k_n=0}^{M-1} h_n(k_1, \cdots, k_n) \cdot u(t-k_1) \cdots u(t-k_n), \tag{3}$$

with $h_n(k_1, \cdots, k_n)$ the nth-order Volterra kernel and M the memory of the kernel. We only consider causal systems, which implies that $h_n(k_1, \cdots, k_n) = 0$ when any of the indices $k_1, \ldots, k_n$ is negative. Note that y_1 is the usual first-order convolution of the input $u(t)$ with the impulse response h_1, which is well-known from linear system theory. For orders $n \geq 2$ the kernels are not unique, since the products $u(t-k_1) \cdots u(t-k_n)$ are commutative. This becomes problematic when system properties are to be described in terms of properties of the kernels. It therefore becomes important to consider restricted forms of the kernels that impose uniqueness. The form we assume throughout this work is the symmetric

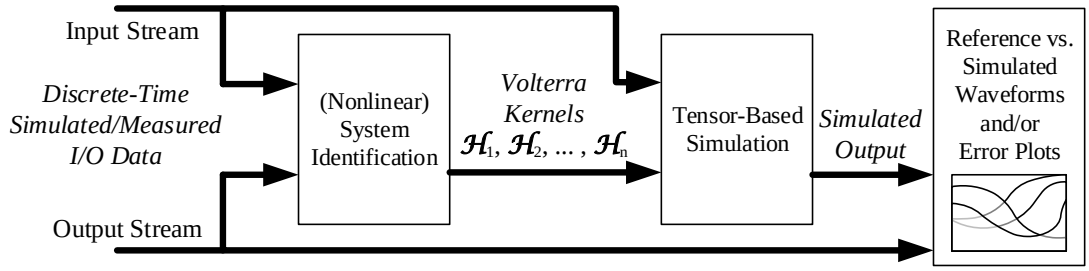

Figure 1: General overview of the tensor-based black-box Volterra series identification and simulation framework.

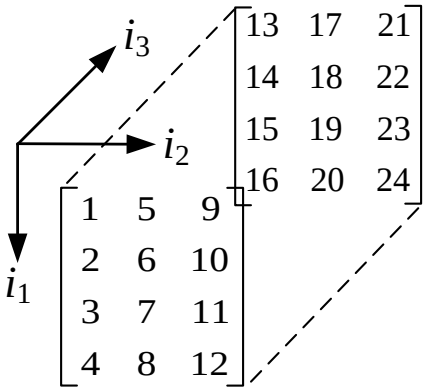

Figure 2: An example tensor $\mathcal{A} \in \mathbb{R}^{4\times3\times2}$.

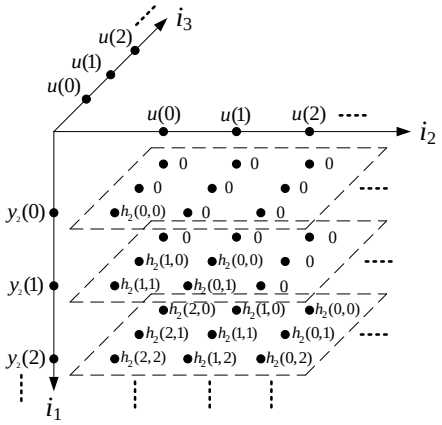

Figure 3: Toeplitz tensor $\mathcal{T}_2$ corresponding with the Volterra kernel h_2.

one. This implies that $h(k_1,\ldots,k_n) = h(\pi(k_1,\ldots,k_n))$, where $\pi(k_1,\ldots,k_n)$ is any permutation of the indices $k_1,\ldots,k_n$, and reduces the number of distinct kernel values from M^n to $\binom{M-1+n}{M-1} \approx (M-1)^n/n!$. It is straightforward to see that each Volterra kernel h_n corresponds with a symmetric n-way tensor $\mathcal{H}_n$, which allows us to rewrite (3) as

$$y_n(t) = \mathcal{H}_n \times_1 \boldsymbol{u}^T \times_2 \cdots \times_n \boldsymbol{u}^T, \tag{4}$$

with $\boldsymbol{u}^T = \begin{pmatrix} u(t) & u(t-1) & \cdots & u(t-M+1) \end{pmatrix}$. By using (1), (4) can be rewritten as

$$y_n(t) = \text{vec}(\mathcal{H}_n)^T \boldsymbol{u}^n = (\boldsymbol{u}^n)^T \text{vec}(\mathcal{H}_n), \tag{5}$$

which needs $O(M^n)$ multiplications. Defining

$$\boldsymbol{y_n} \triangleq \begin{pmatrix} y_n(0) & y_n(1) & \cdots & y_n(N) \end{pmatrix}^T,$$
$$\boldsymbol{u}_N \triangleq \begin{pmatrix} u(0) & u(1) & \cdots & u(N) \end{pmatrix}^T,$$

we can write (5) for $t = 0,\ldots,N$ for a linear system $(n=1)$ to obtain

$$\boldsymbol{y_1} = \begin{pmatrix} u(0) & 0 & \cdots & 0 \\ u(1) & u(0) & \cdots & 0 \\ \vdots & & & \\ u(N) & u(N-1) & \cdots & u(N-M+1) \end{pmatrix} \text{vec}(\mathcal{H}_1),$$

which is the well-known expression for the discrete convolution of $\boldsymbol{u}_N$ with the impulse response $\mathcal{H}_1$. The convolution is here performed as a matrix vector product of a $N \times M$ Toeplitz matrix containing the values of $\boldsymbol{u}_N$ with the vectorization of $\mathcal{H}_1$. Likewise, it is possible to rewrite the convolution as a matrix vector product of a $N \times N$ Toeplitz matrix $\boldsymbol{T}_1$ containing the values of $\mathcal{H}_1$ with the vector $\boldsymbol{u}_N$ as

$$\boldsymbol{y_1} = \begin{pmatrix} h(0) & 0 & \cdots & & \cdots & 0 \\ h(1) & h(0) & \cdots & & \cdots & 0 \\ \vdots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & \cdots & h(M) & \cdots & h(1) & h(0) \end{pmatrix} \boldsymbol{u}_N. \tag{6}$$

By following the same procedure of writing (5) for an arbitrary n for all values $t = 0,\ldots,N$ we obtain

$$\boldsymbol{y_n} = \mathcal{T}_n \times_2 \boldsymbol{u}_N^T \times_3 \cdots \times_{n+1} \boldsymbol{u}_N^T, \tag{7}$$

where $\mathcal{T}_n$ is a $n+1$-way cubical Toeplitz tensor of dimension $N+1$ containing the coefficients of the Volterra kernel $\mathcal{H}_n$. Observe how (7) reduces to (6) when $n=1$. Fig. 3 illustrates the Toeplitz tensor $\mathcal{T}_n$ for the case $n=2$. We call (7) a multi-mode convolution of $\boldsymbol{u}_N$ with $\mathcal{H}_n$. The computational complexity of computing such a multi-mode convolution can be deduced from using the matricization of $\mathcal{T}_n$ as described in (2). Indeed, we can rewrite (7) as

$$\boldsymbol{y_n} = \boldsymbol{T}_n \boldsymbol{u}_N^d, \tag{8}$$
$$= \boldsymbol{U}_n \text{vec}(\mathcal{H}_n), \tag{9}$$

with $\boldsymbol{U}_n$ a $N \times M^n$ matrix. Computing the multi-mode convolution as described in (8) needs $O((N+1)^{n+1})$ multiplications, while using (9) has a computational complexity of $O(NM^n)$. Since in

practice $N \gg M$, the second way is better but still suffers from the curse of dimensionality due to the M^n factor. We resolve this curse in Section 4 using a polyadic tensor decomposition. Also note that for the linear case ($n = 1$) the convolution can be computed using the Fast Fourier Transform (FFT) with a computational complexity of $O(N \log N)$.

3. IDENTIFICATION

It is well-known that the estimation of the Volterra kernel values for an unknown system from measured input/output is a linear problem. Once the values for the maximal order n and memory M are chosen, one then uses the measured input $\boldsymbol{u}_N$ and output $\boldsymbol{y}$ signal to estimate the values of $\boldsymbol{\mathcal{H}}_1, \boldsymbol{\mathcal{H}}_2, \ldots, \boldsymbol{\mathcal{H}}_n$. By repeated use of (9), $\boldsymbol{y} = \boldsymbol{y_1} + \boldsymbol{y_2} + \cdots + \boldsymbol{y_n}$ can be rewritten as the linear problem

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 & \cdots & \boldsymbol{U}_n \end{pmatrix} \begin{pmatrix} \mathrm{vec}(\boldsymbol{\mathcal{H}}_1) \\ \mathrm{vec}(\boldsymbol{\mathcal{H}}_2) \\ \vdots \\ \mathrm{vec}(\boldsymbol{\mathcal{H}}_n) \end{pmatrix}. \tag{10}$$

The $N \times (M + M^2 + \cdots + M^d)$ matrix $\boldsymbol{U} \triangleq \begin{pmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 & \cdots & \boldsymbol{U}_n \end{pmatrix}$ introduces a constraint on how many samples need to be collected in order for the Volterra kernel values to be uniquely defined. Indeed, the matrix $\boldsymbol{U}$ is square when $N = M + M^2 + \cdots + M^d$. If more samples are measured then the linear problem can be solved using a pseudoinverse approach. Again, the curse of dimensionality appears in the growing size of the linear system as M and n are increased. One way to alleviate this problem somewhat is by exploiting the symmetry of each of the Volterra kernels. The vectorization $\boldsymbol{\mathcal{H}}_n$ contains many repeated entries, which could be removed on the condition that the corresponding values of $\boldsymbol{U}$ are adjusted. This reduces the dimensionality of the $\boldsymbol{U}$ matrix to $N \times M + \binom{M-1+2}{M-1} + \cdots + \binom{M-1+n}{M-1}$, but does not resolves the curse.

An alternative identification procedure that uses higher order tensors explicitly is described in [11]. Instead of identifying the symmetric Volterra kernels, approximations to their symmetric polyadic decompositions are identified instead by means of three iterative algorithms. The estimated kernels are always approximations since the number of terms R in each of the symmetric polyadic decompositions is not known *a priori*. Quantifying the approximation error is difficult in this case since then one needs to compute the output to Algorithm 1 as well.

An interesting development is the use of kernel methods from machine learning in the identification of the Volterra kernels [18]. By representing the Volterra series as elements of a reproducing kernel Hilbert space, the complexity of the estimation process becomes independent of the order of nonlinearity. Even infinite Volterra series expansions can be estimated. Whether this identification method can be used in our proposed framework requires further research. For this article, Algorithm 1 was implemented and used in the numerical experiments for prototyping and verification of the proposed ideas.

4. SIMULATION

Once the symmetric Volterra kernels are estimated, one can use them to efficiently simulate the nonlinear system. Remember from (9) that a naive implementation of the multi-mode convolution has a computational complexity of $O(NM^n)$. We will illustrate how the simulation can be made significantly faster by a polyadic tensor decomposition of $\boldsymbol{\mathcal{H}}_n$ and by exploiting the Toeplitz structure of $\boldsymbol{\mathcal{T}}_n$. The main idea is that a polyadic decomposition of the Volterra tensor $\boldsymbol{\mathcal{H}}_n$ suffices to reconstruct the Toeplitz structure of $\boldsymbol{\mathcal{T}}_n$. Suppose we have a polyadic decomposition of $\boldsymbol{\mathcal{H}}_n$, then for a single output sample $y(t)$ we can rewrite equation (4) as

$$\begin{aligned} y_n(t) &= \boldsymbol{\mathcal{H}}_n \times_1 \boldsymbol{u}^T \times_2 \cdots \times_n \boldsymbol{u}^T, \\ &= (\sum_{i=1}^{R} h_i^{(n)} \circ \cdots \circ h_i^{(1)}) \times_1 \boldsymbol{u}^T \times_2 \cdots \times_n \boldsymbol{u}^T, \\ &= \sum_{i=1}^{R} (\boldsymbol{u}^T h_i^{(n)}) \cdots (\boldsymbol{u}^T h_i^{(1)}). \end{aligned} \tag{11}$$

If we now consider the kth mode and write out the inner products $\boldsymbol{u}^T h_i^{(k)}$ for all $t = 0, \ldots, N$ we obtain

$$\begin{pmatrix} u(0) & 0 & \cdots & 0 \\ u(1) & u(0) & \cdots & 0 \\ \vdots & & & \\ u(N) & u(N-1) & \cdots & u(N-M+1) \end{pmatrix} h_i^{(k)}, \tag{12}$$

which is the convolution of $\boldsymbol{u}_N$ with $h_i^{(k)}$ that can be computed with a computational complexity of $O(N \log N)$. The total computational complexity is hence $O(nRN \log N)$ since there are n modes and R vectors per mode. Let $\boldsymbol{u}_i^{(kk)}$ denote the convolution of $\boldsymbol{u}_N$ with $h_i^{(k)}$, then (11) is computed for all $t = 0, \ldots, N$ as $\boldsymbol{y_n} = \sum_{i=1}^{R} \boldsymbol{u}_i^{(1)} \odot \cdots \odot \boldsymbol{u}_i^{(n)}$. Using a symmetric polyadic decomposition of $\boldsymbol{\mathcal{H}}_n$ can result in an additional speedup since then $\boldsymbol{y_n} = \sum_{i=1}^{R} \boldsymbol{u}_i \odot \cdots \odot \boldsymbol{u}_i$ with a computational complexity of $O(RN \log N)$. The algorithm for fast time-domain simulation using polyadic decompositions of the Volterra kernels is presented in Algorithm 2. Note that the second for-loop in Algorihm 2 disappears when symmetric polaydic decompositions of the Volterra kernels are used.

An additional way of reducing computational complexity of Algorithm 2 is truncating the polyadic decomposition of each $\boldsymbol{\mathcal{H}}_j$. This reduces the number of iterations of the third for-loop at the cost of introducing an approximation error.

Algorithm 1 Volterra kernel identification

Input: $\boldsymbol{u}_N, \boldsymbol{y}, M, n$
Output: $\boldsymbol{\mathcal{H}}_1, \boldsymbol{\mathcal{H}}_2, \ldots, \boldsymbol{\mathcal{H}}_n$

```
for j = 1 ... n do
    construct U_j from u_N, considering the symmetry of H_j
end for
h̄ ← solution of linear system (10)
for j = 1 ... n do
    extract symmetric H_j tensor from h̄ vector
end for
```

Algorithm 2 Fast time-domain Volterra simulation

Input: $\boldsymbol{u}_N, \boldsymbol{\mathcal{H}}_1,$ polyadic decompositions of $\boldsymbol{\mathcal{H}}_2, \ldots, \boldsymbol{\mathcal{H}}_n$
Output: $\boldsymbol{y}$

```
y ← conv(u_N, H_1)
for j = 1 ... n do
    for k = 1 ... j do
        for i = 1 ... R do
            u_i^(k) ← conv(u_N, h_i^(k))
        end for
    end for
    y ← y + Σ_{i=1}^{R} u_i^(1) ⊙ ··· ⊙ u_i^(j)
end for
```

5. NUMERICAL EXPERIMENTS

Algorithms 1 and 2 were implemented in Matlab and tested in the following examples. We compare its runtime with the traditional method of computing the multi-mode convolution. All computations were done on an Intel i5 quad-core processor running at 3.3 GHz with 16 GB RAM. The polyadic and symmetric polyadic decompositions were computed with the freely available TTr1SVD [19] and STEROID [20] algorithms. The quality of the identification for unknown Volterra kernels was quantified by the mean squared error $||\boldsymbol{y} - \hat{\boldsymbol{y}}||_2^2/N$, where $\hat{\boldsymbol{y}}$ denotes the simulated output of the identified Volterra series.

5.1 Decaying multi-dimensional exponentials

First, we demonstrate the validity of Algorithms 1 and 2 by means of an artificial example. Symmetric Volterra kernels were generated up to order $d = 5$ and with memory $M = 10$ and containing exponentially decaying coefficients in the following way. We first define the first-order Volterra kernels as $h_1(k_1) = \exp(-k_1^2)$ with $k_1 = 0, .1, .2, \ldots, .9$. These coefficients are stored in the $M \times 1$ $\boldsymbol{\mathcal{H}}_1$ tensor. The other remaining symmetric higher-order Volterra kernels are then generated as the following outer products of $\boldsymbol{\mathcal{H}}_1$

$$\begin{aligned}
\boldsymbol{\mathcal{H}}_2 &= \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1, \\
\boldsymbol{\mathcal{H}}_3 &= \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1, \\
\boldsymbol{\mathcal{H}}_4 &= \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1, \\
\boldsymbol{\mathcal{H}}_5 &= \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1 \circ \boldsymbol{\mathcal{H}}_1.
\end{aligned}$$

This implies that all generated symmetric Volterra kernels correspond, per definition, with rank-1 tensors $\boldsymbol{\mathcal{H}}_2, \ldots, \boldsymbol{\mathcal{H}}_5$. We also have that each entry of the nth-order Volterra kernel is given by

$$h_n(k_1, \ldots, k_n) = \exp(-k_1^2 - k_2^2 - \cdots - k_n^2).$$

A random input signal of 4000 samples was generated from a standard normal distribution. Computing the corresponding output using (9) took 18 seconds. Exploiting the symmetric rank-1 property of the Volterra kernels in Algorithm 2 to generate the output reduces the computation time to 0.01 seconds, which corresponds to a speedup with a factor of 1169. The input/output signals were used in Algorithm 1 to estimate the Volterra kernels. Since the correct Volterra kernels are known in this case, we use the relative identification error to quantify the accuracy of our identification algorithm. The relative identification error is defined as

$$\frac{||\boldsymbol{\mathcal{H}}_k - \hat{\boldsymbol{\mathcal{H}}}_k||_F}{||\boldsymbol{\mathcal{H}}_k||_F},$$

where $\hat{\boldsymbol{\mathcal{H}}}_k$ denotes the estimated Volterra kernel and $||\boldsymbol{\mathcal{X}}||_F$ denotes the Frobenius norm of a tensor $\boldsymbol{\mathcal{X}}$ (the square root of the sum of squares of all entries in $\boldsymbol{\mathcal{X}}$). Table 1 lists the relative identification errors for each order of the estimated Volterra kernels, confirming the validity of Algorithm 1.

Table 1: Relative identification errors - decaying exponentials.

order	1	2	3	4	5
error	8.6e−8	1.0e−8	5.9e−10	1.0e−11	4.2e−13

5.2 Nonlinear inductance

Next, we consider a nonlinear inductance which is described by

$$v = 0.02\,(1 - (\tanh^2(i/5))\,\frac{di}{dt},$$

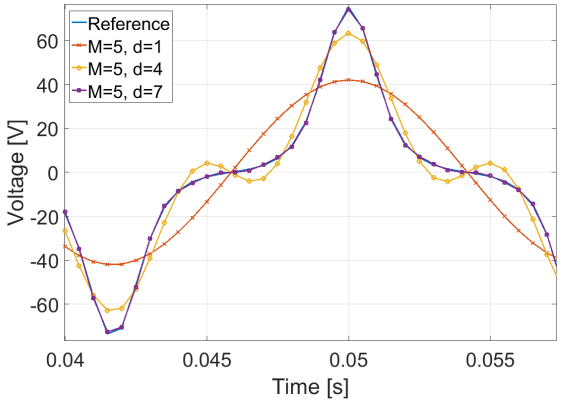

Figure 4: Output voltage of the nonlinear inductance for Volterra series of maximal order $1, 4, 7$.

where v is the output voltage and i is the input current. A 60Hz sinusoidal current and its corresponding voltage were sampled at 100kHz for 0.1 seconds. Volterra kernels were estimated by Algorithm 1 with memories M ranging from 1 to 5 and orders d from 1 up to 7. Fig. 4 shows one period of the simulated output voltage for different orders of the Volterra kernel. Table 2 lists the mean squared error of the output voltage for all identified Volterra series. Notice how the mean squared error is independent from M when $d < 7$ and decreases as soon as an extra odd order is included in the Volterra series. For $d = 7$ a steady decrease of the mean squared error is observed as a function of M. A 7th-order Volterra series with memory $M = 5$ manages to model the output voltage with a mean squared error of 0.0063.

Table 2: Mean squared error - nonlinear inductance.

d \ M	1	2	3	4	5
1	.339	.162	.162	.162	.162
2	.339	.162	.162	.162	.162
3	.339	.061	.061	.061	.061
4	.339	.061	.061	.061	.061
5	.339	.021	.021	.020	.020
6	.339	.021	.021	.020	.020
7	.339	.009	.008	.007	.006

Several input series were then used to compare the different simulation methods. Table 3 lists the typical runtime and storage cost for one set using three different simulation methods on the 7th-order Volterra series with memory $M = 5$. The 'naive' method computes the multi-mode convolution using (9). TTr1SVD and STEROID use Algorithm 2 with the respective polyadic and symmetric polyadic decompositions of the Volterra tensors. The larger storage requirement for the TTr1SVD method is due to that each ith Volterra kernel requires the storage of i mode vectors, while STEROID only needs to store only one vector for every term. The STEROID method is superior compared to the two other simulation methods with a speedup of 93 in runtime compared to the naive method and a reduction of required storage by a factor of 29.

5.3 Double balanced mixer

In this example we consider a double balanced mixer used for upconversion. The output RF signal is determined by the input LO

Table 3: Runtimes and storage costs - nonlinear inductance.

Simulation method	Runtime [s]	Storage [kB]
Naive	48.27	763
TTr1SVD [19]	4.01	845
STEROID [20]	0.52	26

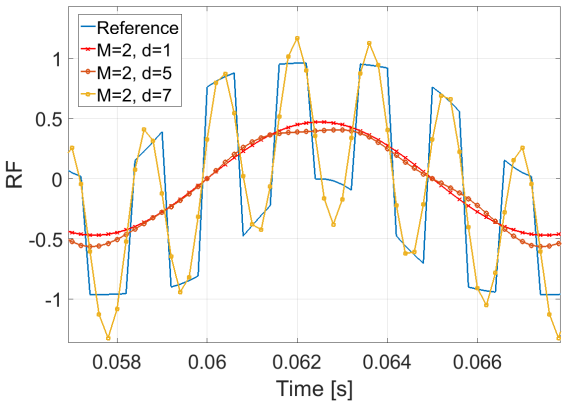

Figure 5: Output RF signal of the mixer for Volterra series of maximal order $1, 5, 7$.

and IF signals. All signals were sampled at 10 kHz for 1 second. The LO and IF signals have frequencies of 600 and 100 Hz respectively, with a phase difference of $\pi/8$. Again, Volterra kernels were estimated by Algorithm 1 with memories ranging from 1 to 5 and orders from 1 up to 7. Table 4 lists the mean squared error of the RF output signal as a function of M and d. One needs a minimal order and memory of 6 and 2 respectively in order to see a decrease in the mean squared error. The mean squared error seems to be small even for the linear system, however, these models are not able to capture the high frequency variations that are relatively small in amplitude while the Volterra series can.

Table 4: Mean squared error - double balanced mixer.

d \ M	1	2	3	4	5
1	.006	.006	.006	.006	.006
2	.006	.006	.006	.006	.006
3	.006	.006	.006	.006	.006
4	.006	.006	.006	.006	.006
5	.006	.006	.006	.006	.006
6	.006	.003	.003	.003	.003
7	.006	.003	.003	.003	.003

Fig. 5 shows one period of the simulated output RF signal for different orders of the Volterra kernel. Notice the small difference between the simulated output of the linear and 5th-degree Volterra series, indicating the high nonlinearity of the mixer. A 6th-degree Volterra series with memory $M = 2$ is able to follow the high frequency RF output. Increasing the memory M or order d did not lead to any further improvement of the simulated output. Again, the same input/output series were used for comparing the different simulation methods for the $d = 6$, $M = 2$ model. Table 5 lists the runtime and storage cost for the three different simulation methods. The same patterns as in the previous example can be observed. The TTr1SVD method needs more storage due to the increasing number of mode vectors that it needs to store. The STEROID method is far superior with a speedup of 266 in runtime compared to the naive method and needing only about one third of the storage.

Table 5: Runtimes and storage costs - double balanced mixer.

Simulation method	Runtime [s]	Storage [kB]
Naive	5.749	0.98
TTr1SVD [19]	0.081	2.11
STEROID [20]	0.021	0.39

6. REMARKS

Some key contributions are summarized again:

1. The (symmetric) polyadic tensor decomposition effectively breaks down the multi-mode convolution of an input vector with any higher order Volterra kernel into purely linear convolutions. Tensor decompositions are therefore key in overcoming the curse of dimensionality.

2. Continuing from the previous remark, the relatively expensive tensor decomposition computation is done only *once* and is the overhead to pay. The simulation algorithm then benefits from the recurring and cheap linear convolution operation.

3. The least-squares framework for kernel identification as described by (10), though being effective, is generic to this work and is employed due to its ease of implementation for validating ideas. Other, more sophisticated, Volterra kernel identification methods can be readily substituted for robust Volterra kernel identification subject to, say, noisy input-output samples.

4. The identified Volterra kernels completely hide the circuit or topological details, while allowing efficient user black-box simulation via compact tensor mode factor representation. In other words, the proposed framework also suggests an effective means of intellectual property (IP) protection.

5. The proposed framework now handles only the identification and simulation of single-input single-output (SISO) systems. The extension to the direct handling of multiple-input multiple-output (MIMO) data streams is underway. Moreover, research is being conducted on the characterization of stability and passivity of the identified models.

7. CONCLUSIONS

This paper has proposed a highly efficient black-box identification and simulation framework for nonlinear systems/circuits based solely on the provision of discrete-time input-output samples. A natural link has been established between Volterra kernels, multi-mode convolution and tensor decompositions. A new tensor-based Volterra simulation algorithm has been developed. Numerical examples have been given to demonstrate the significant reduction in runtime and required storage cost for the simulation of some highly nonlinear systems. Compared to conventional Volterra-based simulation, a runtime speedup up to $1000\times$ and a decrease in storage up to $29\times$ have been observed.

8. REFERENCES

[1] L. Ljung, Ed., *System Identification (2Nd Ed.): Theory for the User*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[2] T. Katayama, *Subspace Methods for System Identification*, ser. Communications and Control Engineering. Springer London, 2005.

[3] O. Nelles, *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*, ser. Engineering online library. Springer, 2001.

[4] W. Rugh, *Nonlinear System Theory – The Volterra-Wiener Approach*. Baltimore, MD: Johns Hopkins Univ. Press, 1981.

[5] E. Bedrosian and S. O. Rice, "The output properties of Volterra systems (nonlinear systems with memory) driven by harmonic and Gaussian inputs," *Proc. IEEE*, vol. 59, no. 12, pp. 1688–1707, Dec. 1971.

[6] J. R. Phillips, "Projection-based approaches for model reduction of weakly nonlinear time-varying systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 2, pp. 171–187, Feb. 2003.

[7] P. Li and L. Pileggi, "Compact reduced-order modeling of weakly nonlinear analog and RF circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 2, pp. 184–203, Feb. 2005.

[8] R. Boyer, R. Badeau, and G. Favier, "Fast orthogonal decomposition of Volterra cubic kernels using oblique unfolding," in *Proc. Very Large Scale Integration (VLSI-SoC)*, Oct. 2011, pp. 160–163.

[9] G. Favier and T. Bouilloc, "Parametric complexity reduction of Volterra models using tensor decompositions," in *17th European Signal Processing Conference (EUSIPCO)*, Aug 2009.

[10] H. Liu, X. Y. Z. Xiong, K. Batselier, L. Jiang, L. Daniel, and N. Wong, "STAVES: Speedy tensor-aided volterra-based electronic simulator," in *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, Nov 2015, pp. 583–588.

[11] G. Favier, A. Y. Kibangou, and T. Bouilloc, "Nonlinear system modeling and identification using Volterra-PARAFAC models," *Int. J. Adapt. Control Signal Process*, vol. 26, no. 1, pp. 30–53, Jan. 2012.

[12] P. A. Regalia and M. K. Sanjit, "Kronecker products, unitary matrices and signal processing applications," *SIAM Review*, vol. 31, no. 4, pp. 586–613, 1989.

[13] C. F. V. Loan, "The ubiquitous Kronecker product," *J. Comp. Appl. Math.*, vol. 123, no. 1-2, pp. 85–100, Nov. 2000.

[14] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[15] J. D. Carroll and J. J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[16] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, no. 1, p. 84, 1970.

[17] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, Oct. 1996.

[18] M. O. Franz and B. Schölkopf, "A unifying view of Wiener and Volterra theory and polynomial kernel regression," *Neural Computation*, vol. 18, no. 12, pp. 3097–3118, 2006.

[19] K. Batselier, H. Liu, and N. Wong, "A constructive algorithm for decomposing a tensor into a finite sum of orthonormal rank-1 terms," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 3, pp. 1315–1337, Sep. 2015.

[20] K. Batselier and N. Wong, "Symmetric tensor decomposition by an iterative eigendecomposition algorithm," *ArXiv e-prints*, Sep. 2014.