<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>A tensor-based Volterra series black-box nonlinear system identification and simulation framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Batselier, K; Chen, Z; Liu, H; Wong, N</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD ’16 ), Austin, TX., 7-10 November 2016, p. Article No. 17</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2016</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/229783">http://hdl.handle.net/10722/229783</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.</td>
</tr>
</tbody>
</table>
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  
wong@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].

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 $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n$ 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 gracefully 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. \( \mathbf{A} \)), matrices by boldface capital letters (e.g. \( \mathbf{A} \)), vectors by boldface letters (e.g. \( \mathbf{a} \)), and scalars by either Roman (e.g. a) or Greek (e.g. \( \alpha \)) letters. The \( n \)th element in a sequence is denoted by a superscript in parentheses. For example, \( \mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)} \) denote the first three elements in a sequence of \( \mathbf{u} \) vectors. Tensors are in our context generalizations of matrices and vectors. A \( d \)-way tensor \( \mathbf{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d} \) is simply a \( d \)-way array. Each element of the tensor \( \mathbf{A} \) is hence specified by \( d \) indices \( i_1, \ldots, i_d \) and denoted by \( a_{i_1, \ldots, i_d} \). 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 \( \mathbf{A} \) with a vector \( \mathbf{u} \in \mathbb{R}^{n_k} \) is

\[
(\mathbf{A} \times_k \mathbf{u})^{(1)} = \sum_{i_k=1}^{n_k} u_{i_k} a_{i_1, \ldots, i_k, \ldots, i_d},
\]

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

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

\[
\mathbf{A} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(d)},
\]

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

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

The Kronecker product [12, 13] and Hadamard product are denoted by \( \otimes \) and by \( \circ \) respectively. We introduce the shorthand notation \( \mathbf{x}^d \triangleq \mathbf{x} \otimes \mathbf{x} \otimes \cdots \otimes \mathbf{x} \) 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 \( \mathbf{A} \) into a matrix \( \mathbf{A} \) and vector \( \text{vec}(\mathbf{A}) \). The mode-\( n \) matricization of a tensor \( \mathbf{A} \) rearranges the entries of \( \mathbf{A} \) such that the rows of the resulting matrix are indexed by the \( n \)th 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 \( \mathbf{A} \) from Fig. 2 is

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

and its vectorization is

\[
\text{vec}(\mathbf{A}) = (1 \ 2 \ \cdots \ 24)^T.
\]

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

\[
\mathbf{A} \times_1 \mathbf{u}^T \times_2 \cdots \times_n \mathbf{u}^T = \text{vec}(\mathbf{A})^T \mathbf{u}^d
\]

and

\[
\mathbf{A} \times_2 \mathbf{u}^T \times_3 \cdots \times_n \mathbf{u}^T = \mathbf{A} \mathbf{u}^{d-1},
\]

which tell us how the \( k \)-mode products of a tensor \( \mathbf{A} \) with a vector \( \mathbf{u} \) can be computed.

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

\[
\mathbf{A} = \sum_{i=1}^{R} \mathbf{a}^{(1)}(i) \circ \cdots \circ \mathbf{a}^{(d)}(i).
\]

In order to store a \( d \)-way tensor \( \mathbf{A} \) 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 \( \mathbf{A} \) can always be decomposed into a symmetric polyadic decomposition. This means that every rank-1 term is also a symmetric tensor and therefore

\[
\mathbf{A} = \sum_{i=1}^{R} \mathbf{a}_i \circ \cdots \circ \mathbf{a}_i.
\]

Storage of a symmetric polyadic decomposition therefore needs only \( R n \) elements when \( \mathbf{a}_i \in \mathbb{R}^{n} \).

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

\[
y_n(t) = \sum_{k_1=0}^{M-1} \cdots \sum_{k_n=0}^{M-1} h_n(k_1, \ldots, k_n) u(t-k_1) \cdots u(t-k_n),
\]

with \( h_n(k_1, \ldots, k_n) \) the \( n \)-th order Volterra kernel and \( M \) the memory of the kernel. We only consider causal systems, which implies that \( h_n(k_1, \ldots, k_n) = 0 \) 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
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 \( (M-1)^n / n! \). It is straightforward to see that each Volterra kernel \( h_n \) corresponds with a symmetric \( n \)-way tensor \( H_n \), which allows us to rewrite (3) as

\[
y_n(t) = H_n \times_1 u^T \times_2 \cdots \times_n u^T,
\]

(4)

with \( u^T = (u(t) \ u(t-1) \ \cdots \ u(t-M+1)) \). By using (1), (4) can be rewritten as

\[
y_n(t) = \text{vec}(H_n)^T u^n = (u^n)^T \text{vec}(H_n),
\]

(5)

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

\[
y_n \triangleq (y_n(0) \ y_n(1) \ \cdots \ y_n(N))^T,
\]

\[
u_N \triangleq (u(0) \ u(1) \ \cdots \ u(N))^T,
\]

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

\[
y_1 = \begin{pmatrix} u(0) \\ u(1) \\ \vdots \\ u(N) \end{pmatrix} \text{vec}(H_1),
\]

which is the well-known expression for the discrete convolution of \( u_N \) with the impulse response \( H_1 \). The convolution is here performed as a matrix vector product of a \( N \times M \) Toeplitz matrix containing the values of \( u_N \) with the vectorization of \( H_1 \). Likewise, it is possible to rewrite the convolution as a matrix vector product of a \( N \times N \) Toeplitz matrix \( T_1 \) containing the values of \( H_1 \) with the vector \( u_N \) as

\[
y_1 = \begin{pmatrix} h(0) & 0 & \cdots & 0 \\ h(1) & h(0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & h(1) \end{pmatrix} \begin{pmatrix} u(0) \\ u(1) \\ \vdots \\ u(N) \end{pmatrix},
\]

(6)

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

\[
y_n = T_n \times_2 u_N^T \times_3 \cdots \times_n u_N^T,
\]

(7)

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

\[
y_n = T_n \ u_N^d, \quad \text{or} \quad y_n = U_n \text{vec}(H_n),
\]

(8)

(9)

with \( U_n \) 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 $u_n$ and output $y$ signal to estimate the values of $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n$. By repeated use of (9), $y = y_1 + y_2 + \cdots + y_n$ can be rewritten as the linear problem

$$y = (U_1 \ U_2 \ \cdots \ U_n) \begin{pmatrix} \text{vec}(\mathcal{H}_1) \\ \text{vec}(\mathcal{H}_2) \\ \vdots \\ \text{vec}(\mathcal{H}_n) \end{pmatrix}.$$  \hspace{1cm} (10)

The $N \times (M + M^2 + \cdots + M^n)$ matrix $U \triangleq (U_1 \ U_2 \ \cdots \ U_n)$ 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 $U$ is square when $N = M + M^2 + \cdots + M^n$. 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 $\text{vec}(\mathcal{H}_n)$ contains many repeated entries, which could be removed on the condition that the corresponding values of $U$ are adjusted. This reduces the dimensionality of the $U$ matrix to $N \times M = (M - 2) + (M - 3) + \cdots + (M-n)$, but does not resolve the curse.

An alternative identification procedure that uses higher order tensors explicitly is described in [11]. Instead of identifying the symmetric polyadic decompositions of $\mathcal{H}_n$, an alternative way of reducing computational complexity of Algorithm 2 is truncating the polyadic decomposition of each $\mathcal{H}_j$ tensor from $\bar{h}$ vector.

$$\text{Algorithm 1 Volterra kernel identification}$$

**Input:** $u_N, y, M, n$
**Output:** $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n$

1. $j = 1 \ldots n$ do
2. construct $U_j$ from $u_n$, considering the symmetry of $\mathcal{H}_j$
3. $j = 1 \ldots n$ do
4. $\bar{h} \leftarrow$ solution of linear system (10)
5. $j = 1 \ldots n$ do
6. extract symmetric $\mathcal{H}_j$ tensor from $\bar{h}$ vector
7. end for
8. end for
9. end for

$$\text{Algorithm 2 Fast time-domain Volterra simulation}$$

**Input:** $u_N, \mathcal{H}_1$, polyadic decompositions of $\mathcal{H}_2, \ldots, \mathcal{H}_n$
**Output:** $y$

1. $y \leftarrow \text{conv}(u_N, \mathcal{H}_1)$
2. $j = 1 \ldots n$ do
3. $k = 1 \ldots j$ do
4. $i = 1 \ldots R$ do
5. $u^{(k)}(i) \leftarrow \text{conv}(u_N, h^{(k)}(i))$
6. $i = 1 \ldots R$ do
7. $j = 1 \ldots n$ do
8. $y \leftarrow y + \sum_{i=1}^{R} u^{(1)}(1) \cdots u^{(j)}(j)$
9. $j = 1 \ldots n$ do

pose we have a polyadic decomposition of $\mathcal{H}_n$, then for a single output sample $y(t)$ we can rewrite equation (4) as

$$y_n(t) = \mathcal{H}_n \times_{1} u^T \times_{2} \cdots \times_{n} u^T$$

$$= (\sum_{i=1}^{R} h^{(n)}(1) \cdots h^{(1)}(1)) \times_{1} u^T \times_{2} \cdots \times_{n} u^T$$

$$= \sum_{i=1}^{R} (u^T \ h^{(n)}(1) \cdots u^T \ h^{(1)}(1)).$$ \hspace{1cm} (11)

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

$$\begin{pmatrix} u(0) & 0 & \cdots & 0 \\ u(1) & u(0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ u(N) & u(N-1) & \cdots & u(N-M+1) \end{pmatrix} \ h^{(k)}(l),$$ \hspace{1cm} (12)

which is the convolution of $u_N$ with $h^{(k)}(l)$ that can be computed with a computational complexity of $O(N \log N)$. The total computational complexity is hence $O(nRM^2 \log N)$ since there are $n$ modes and $R$ vectors per mode. Let $u^{(kk)}(k)$ denote the convolution of $u_N$ with $h^{(k)}(l)$, then (11) is computed for all $t = 0, \ldots, N$ as $y_n = \sum_{i=1}^{R} u^{(1)}(1) \cdots u^{(n)}(n)$. Using a symmetric polyadic decomposition of $\mathcal{H}_n$ can result in an additional speedup since then $y_n = \sum_{i=1}^{R} u^{(1)}(1) \cdots u^{(n)}(n)$ with a computational complexity of $O(RM^2 \log N)$. 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 Algorithm 2 disappears when symmetric polyadic decompositions of each $\mathcal{H}_j$ are used.

An additional way of reducing computational complexity of Algorithm 2 is truncating the polyadic decomposition of each $\mathcal{H}_j$. This reduces the number of iterations of the third for-loop at the cost of introducing an approximation error.
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 $||y - \hat{y}||^2/N$, where $\hat{y}$ 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 $M \times 1$ $\mathcal{H}_1$ tensor. The other remaining symmetric higher-order Volterra kernels are then generated as the following outer products of $\mathcal{H}_1$

- $\mathcal{H}_2 = \mathcal{H}_1 \circ \mathcal{H}_1$,
- $\mathcal{H}_3 = \mathcal{H}_1 \circ \mathcal{H}_1 \circ \mathcal{H}_1$,
- $\mathcal{H}_4 = \mathcal{H}_1 \circ \mathcal{H}_1 \circ \mathcal{H}_1 \circ \mathcal{H}_1$,
- $\mathcal{H}_5 = \mathcal{H}_1 \circ \mathcal{H}_1 \circ \mathcal{H}_1 \circ \mathcal{H}_1 \circ \mathcal{H}_1$.

This implies that all generated symmetric Volterra kernels correspond, per definition, with rank-1 tensors $\mathcal{H}_2, \ldots, \mathcal{H}_5$. We also have that each entry of the $n$th-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

$$\frac{||\mathcal{H}_k - \mathcal{H}_k^*||_F}{||\mathcal{H}_k^*||_F},$$

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

<table>
<thead>
<tr>
<th>order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>8.6e−8</td>
<td>1.0e−8</td>
<td>5.9e−10</td>
<td>1.0e−11</td>
<td>4.2e−13</td>
</tr>
</tbody>
</table>

5.2 Nonlinear inductance

Next, we consider a nonlinear inductance which is described by

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

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.

<table>
<thead>
<tr>
<th>$d$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>.339</td>
<td>.162</td>
<td>.162</td>
<td>.162</td>
<td>.162</td>
</tr>
<tr>
<td>1</td>
<td>.339</td>
<td>.162</td>
<td>.162</td>
<td>.162</td>
<td>.162</td>
</tr>
<tr>
<td>2</td>
<td>.339</td>
<td>.162</td>
<td>.162</td>
<td>.162</td>
<td>.162</td>
</tr>
<tr>
<td>3</td>
<td>.339</td>
<td>.061</td>
<td>.061</td>
<td>.061</td>
<td>.061</td>
</tr>
<tr>
<td>4</td>
<td>.339</td>
<td>.061</td>
<td>.061</td>
<td>.061</td>
<td>.061</td>
</tr>
<tr>
<td>5</td>
<td>.339</td>
<td>.021</td>
<td>.021</td>
<td>.020</td>
<td>.020</td>
</tr>
<tr>
<td>6</td>
<td>.339</td>
<td>.021</td>
<td>.021</td>
<td>.020</td>
<td>.020</td>
</tr>
<tr>
<td>7</td>
<td>.339</td>
<td>.009</td>
<td>.008</td>
<td>.007</td>
<td>.006</td>
</tr>
</tbody>
</table>

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 nth 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

![Figure 4: Output voltage of the nonlinear inductance for Volterra series of maximal order 1, 4, 7.](Image)
Table 3: Runtimes and storage costs - nonlinear inductance.

<table>
<thead>
<tr>
<th>Simulation method</th>
<th>Runtime [s]</th>
<th>Storage [kB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>48.27</td>
<td>763</td>
</tr>
<tr>
<td>TTr1SVD [19]</td>
<td>4.01</td>
<td>845</td>
</tr>
<tr>
<td>STEROID [20]</td>
<td>0.52</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 4: Mean squared error - double balanced mixer.

<table>
<thead>
<tr>
<th>d</th>
<th>M=2, d=1</th>
<th>M=2, d=5</th>
<th>M=2, d=7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.006</td>
<td>.006</td>
<td>.006</td>
</tr>
<tr>
<td>2</td>
<td>.006</td>
<td>.006</td>
<td>.006</td>
</tr>
<tr>
<td>3</td>
<td>.006</td>
<td>.006</td>
<td>.006</td>
</tr>
<tr>
<td>4</td>
<td>.006</td>
<td>.006</td>
<td>.006</td>
</tr>
<tr>
<td>5</td>
<td>.006</td>
<td>.006</td>
<td>.006</td>
</tr>
<tr>
<td>6</td>
<td>.006</td>
<td>.003</td>
<td>.003</td>
</tr>
<tr>
<td>7</td>
<td>.006</td>
<td>.003</td>
<td>.003</td>
</tr>
</tbody>
</table>

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.

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

<table>
<thead>
<tr>
<th>Simulation method</th>
<th>Runtime [s]</th>
<th>Storage [kB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>5,749</td>
<td>0.98</td>
</tr>
<tr>
<td>TTr1SVD [19]</td>
<td>0.081</td>
<td>2.11</td>
</tr>
<tr>
<td>STEROID [20]</td>
<td>0.021</td>
<td>0.39</td>
</tr>
</tbody>
</table>

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


