Low-Rank Tensor Decompositions for Nonlinear System Identification
A Tutorial with Examples

Batselier, Kim

DOI
10.1109/MCS.2021.3122268

Publication date
2022

Document Version
Final published version

Published in
IEEE Control Systems

Low-Rank Tensor Decompositions for Nonlinear System Identification

A TUTORIAL WITH EXAMPLES

KIM BATSELIER

Nonlinear parametric system identification is the estimation of nonlinear models of dynamical systems from measured data. Nonlinear models are parameterized, and it is exactly these parameters that must be estimated. Extending familiar linear models to their nonlinear counterparts quickly leads to practical problems. For example, the generalization of a multivariate linear function to a multivariate polynomial implies that the number of parameters grows exponentially with the total degree of the polynomial. This exponential explosion of model parameters is an instance of the so-called curse of dimensionality. Both the storage and computational complexities are limiting factors in the development of system identification methods for such models.

The solution to this problem has been sought in the limitation of the complexity of these models through various ways. For example, state-of-the-art identification methods for nonlinear autoregressive moving average models with exogenous inputs try to determine which model terms are important during identification [1]. The goal is to obtain a sparse model that is described by only a relatively small subset of the exponential number of terms. An alternative approach involves block-oriented models, where the complexity of the model is reduced by constructing a nonlinear model through two kinds of blocks: a linear dynamical block and a nonlinear static transformation block [2], [3]. Two simple examples of such models are the Hammerstein model (which consists of a static nonlinearity followed by a linear block) and the Wiener model (which has the opposite order). More complexity can be achieved by combining different branches of block-oriented models [4]. Sparse and block-oriented model structures, however, are mostly meaningful when the control engineer has

Digital Object Identifier 10.1109/MCS.2021.3122268
Date of current version: 18 January 2022

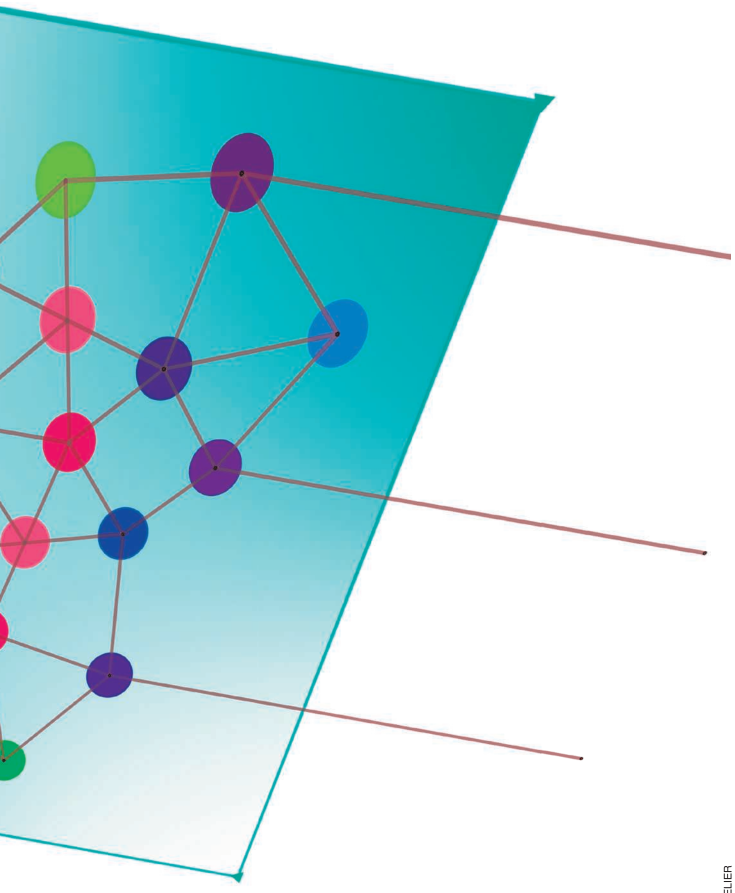
1066-033X/22©2022IEEE

strong reasons to believe that the system really has such a sparse or particular block structure.

The approach to address the curse of dimensionality described in this article is the use of tensor decompositions (see "Summary"). Tensors are multidimensional arrays that are generalizations of vectors and matrices to higher orders. The total number of entries of a tensor also grows exponentially with the number of dimensions, and tensor decompositions can alleviate this problem. The key idea in lifting the curse of dimensionality lies in tensor decompositions, where a given tensor is decomposed into a set of tensors of much smaller size. These smaller tensors are often called factor matrices or core tensors. In this way, the original tensor never has to be explicitly kept in memory. Instead, the core tensors can be stored in memory. The "rank" of a tensor decomposition plays an important role, as it determines how small the dimensions of the core tensors can be. Different tensor decompositions have their own definition of rank, and the question then naturally rises whether low-rank tensor decompositions can be meaningful in the context of dynamical systems.

One goal of this article is to show that this is effectively the case. A low-rank representation of a (nonlinear) model can be intuitively understood as a way of implicitly adding the constraint that the model parameters are not all completely independent from one another. For example, "Low-Rank Motivation" shows how a low-rank tensor structure emerges when modeling a simple linear mass spring system. The entries of the $\boldsymbol{A}$ matrix in the corresponding linear state-space model are not independent from one another. The same is true in nonlinear models. Volterra kernels, for example, are generalizations of finite-impulse responses (FIRs) to higher orders and are therefore expected to be smoothly decaying. Recent research enforces these constraints explicitly through regularization [5], [6], but is unfortunately limited to third-order kernels. Smooth functions have been shown to result in low-rank tensor decompositions [7], [8]. A low-rank tensor decomposition of the Volterra kernel coefficients can therefore be interpreted as implicitly encoding this smoothness and allows for the identification of models with orders higher than three. The link between sparse parametric models and low-rank tensor decompositions is explained in "Sparse Models as Restricted Low-Rank Tensor Decompositions."

The goal of this article is twofold. First, it serves as a basic introduction to tensor decompositions, as they can be a powerful tool for addressing large-scale problems in systems and control. A second objective of this article lies in the presentation of three applications in nonlinear system identification where a low-rank tensor approach is used. The identification of both Volterra systems and state-space models with polynomial inputs is discussed. This article starts with an overview of some basic tensor notations, tensor operations, and decompositions. This overview is followed by a first application: the low-rank, tensor-based identification of truncated Volterra systems. The second application discusses a Kalman filter approach to the Volterra identification problem, where both the mean vector and covariance matrix of the distributions are represented by low-rank tensor decompositions. A third and final application is a tensor-based subspace


KIM BATSELIER

Summary

Tensor decompositions can be a powerful tool when faced with the curse of dimensionality and have been applied in myriad applications. Their application to problems in the control community remains largely unexplored. This article aims at filling this gap by introducing tensor decompositions, where the key idea is always to exploit structure in the problem to lift the curse of dimensionality. This structure leads to the notion of low rank, which can be intuitively understood as parameters in the problem being correlated. The potential of low-rank tensor decompositions is illustrated by means of three applications, specifically in nonlinear system identification. The parametric identification of both Volterra systems and state-space models with polynomial inputs is discussed.

identification algorithm for state-space models with polynomial inputs. Finally, an overview of online resources and software is given that can help tensor decomposition users with particular problems in control.

TENSOR NOTATION AND BASICS

Tensors are multidimensional arrays. A D-way or Dth-order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ is a D-dimensional array where each entry $\mathcal{A}(i_1, \ldots, i_D)$ is completely determined by D indices $i_1, \ldots, i_D$. Commonly used tensors in control are scalars $(D = 0)$, vectors $(D = 1)$, and matrices $(D = 2)$. One of the goals of this article is to show that tensors of order $D > 0$ and their decompositions can be a beneficial tool in control, with a specific focus on nonlinear system identification. More detailed information about tensors and their decompositions can be found in [9]–[11].

Tensor notation and terminology differ across fields, with a particular distinction between the field of applied mathematics and quantum mechanics. In this article, the applied mathematics terminology will be used. Notation conventions are summarized in Table 1. Following the convention used in Matlab, all indices are one based and hence start counting from one rather than from zero. Indices are always denoted by lowercase letters, and their corresponding capital letters denote their respective upper bounds. For example, an entry of an $M \times N$ matrix $\boldsymbol{A}$ is denoted $\boldsymbol{A}(m, n)$. Indices can also be combined into a single multi-index. The conversion of D separate indices $i_1, i_2, \ldots, i_D$ into one multi-index $[i_1 i_2 \ldots i_D]$ follows the definition

$$[i_1 i_2 \cdots i_D] = i_1 + \sum_{d=2}^{D} (i_d - 1) \prod_{l=1}^{d-1} I_l. \tag{1}$$

The same formula is used when a single index $i = [i_1 i_2 \cdots i_D]$ is split into D seperate indices. The diagonal of a tensor $\mathcal{A}$ is the entries when all indices attain equal values; that is,

$$\mathcal{A}(i, i, \ldots, i), \quad i = 1, \ldots I.$$

Low-Rank Motivation

To illustrate the emergence of low-rank tensor decompositions in dynamical systems, consider a linear sequence of N masses all connected through $N-1$ springs, as shown in Figure S1. The origin of the local coordinate x_n lies at the equilibrium position of mass m_n, and each spring has a spring constant k_n. Assuming no friction, each mass m_n, excluding the ones at the border, is then described by the differential equation

$$m_n \ddot{x}_n = -k_{n-1}(x_n - x_{n-1}) + k_n(x_{n+1} - x_n). \tag{S1}$$

The state of each mass m_n can be chosen to contain $x_n, \dot{x}_n$, and the concatenation of these state variables for all masses in a state vector $\boldsymbol{x}(t) \in \mathbb{R}^{2N}$ allows us to write the continuous-time state-space model

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\,\boldsymbol{x}(t), \\ \boldsymbol{y}(t) &= \boldsymbol{x}(t), \end{aligned}$$

with state dynamics matrix $\boldsymbol{A} \in \mathbb{R}^{2N \times 2N}$. The inputs to the system are not important in this discussion, as we focus completely on the state dynamics.

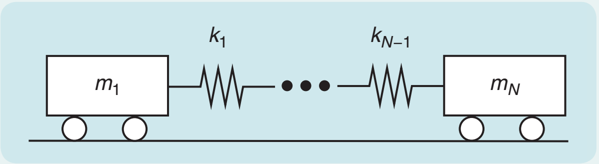


FIGURE S1 A linear sequence of N masses $m_1, \ldots, m_N$ connected through springs with spring constants $k_1, \ldots, k_{N-1}$.

From a white-box perspective, the system identification problem with regard to the state dynamics is defined as the estimation of the $2 + 2(N-2)$ unique entries (up to a sign) of the $\boldsymbol{A}$ matrix. These entries are comprised of contributions from the N masses m_n and $N-1$ spring constants k_n. For example, when $N = 4$, the state dynamics matrix is

$$\boldsymbol{A} = \begin{pmatrix} & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \\ -\frac{k_1}{m_1} & \frac{k_1}{m_1} & & & & & & \\ \frac{k_1}{m_2} & -\frac{k_1+k_2}{m_2} & \frac{k_2}{m_2} & & & & & \\ & \frac{k_2}{m_3} & -\frac{k_2+k_3}{m_3} & \frac{k_3}{m_3} & & & & \\ \frac{k_3}{m_4} & -\frac{k_3}{m_4} & & & & & & \end{pmatrix}, \tag{S2}$$

where empty spaces denote zero entries. The $2 + 2(4-2) = 6$ unique parameters are then found in the lower-left corner of the $\boldsymbol{A}$ matrix. Although the number of $\boldsymbol{A}$ entries scales quadratically in the number of masses $O(N^2)$, the number of parameters to be estimated in identification grows only linearly in the number of masses $O(N)$. In this example, Newton's laws enforce a particular structure to the $\boldsymbol{A}$ matrix, reducing the number of unique parameters.

Now, assume a black-box perspective, and suppose that the knowledge that the system consists of a linear sequence of masses connected through springs is not available. Furthermore, assume that the size of the state vector $\boldsymbol{x}(t)$ is known. The question then arises whether it is possible to encode the

For the diagonal of a tensor to exist, it is required that $I_d \geq I$, where d assumes values between one and D. A diagonal tensor is defined as a tensor for which all off-diagonal entries are zero. Three important products are the outer product, Kronecker product, and Khatri–Rao product. The outer product $\circ$ of D vectors $\boldsymbol{a}^{(d)} \in \mathbb{R}^{I_d}$, where d assumes values between one and D, is, per definition, the tensor

$$\mathcal{A} = \boldsymbol{a}^{(1)} \circ \boldsymbol{a}^{(2)} \circ \cdots \circ \boldsymbol{a}^{(D)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}, \tag{2}$$

such that each tensor entry is computed from the product of the corresponding vector entries

$$\mathcal{A}(i_1, i_2, \ldots, i_D) = \boldsymbol{a}^{(1)}(i_1)\, \boldsymbol{a}^{(2)}(i_2) \cdots \boldsymbol{a}^{(D)}(i_D).$$

From this definition, the outer product is not commutative. The observant reader probably has already noticed that (2) can be interpreted as the tensor $\mathcal{A}$ being "decomposed" into a set of D tensors. This is correct, as the outer product plays a fundamental role in tensor decompositions for exactly this reason. More details are discussed in "Tensor Decompositions." A related product is the Kronecker product, which is usually defined between matrices. The Kronecker

TABLE 1 A summary of tensor notations used throughout this article.

$a, \boldsymbol{a}, \boldsymbol{A}, \mathcal{A}$	A scalar, vector matrix, higher-order tensor
$\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(D)}$	An enumeration of D tensors
$\mathcal{A}(i_1, \ldots, i_D)$	The $(i_1, \ldots, i_D)$th entry of tensor $\mathcal{A}$
$\boldsymbol{A}^T$	The transpose of a matrix $\boldsymbol{A}$
$\boldsymbol{A}^{-1}$	The (Moore–Penrose pseudo-) inverse of a matrix $\boldsymbol{A}$
$\circ$	An outer product
$\otimes$	The Kronecker product
$\odot$	The Khatri–Rao product
$\boldsymbol{a} = \text{vec}(\mathcal{A})$	The vectorization of a tensor $\mathcal{A}$
$\boldsymbol{1}_N$	A vector of ones of length N

assumption that there is only a limited number of unique entries (due to physical laws) inside the $\boldsymbol{A}$ matrix. The answer to this question is a resounding yes, and a key part of the answer lies in a particular notion of rank. Figure S2 shows the total number of parameters that must be stored in memory for the $\boldsymbol{A}$ matrix, the singular value decomposition (SVD) of the $\boldsymbol{A}$ matrix, the tensor train matrix representation of the $\boldsymbol{A}$ matrix, and the white-box model as a function of the number of masses in the system. Remarkably, the SVD requires more storage than the original matrix, due to the $\boldsymbol{A}$ matrix being almost full rank. Only one singular value is zero.

The tensor train matrix of $\boldsymbol{A}$ is a specific tensor decomposition that is designed to represent matrices. This tensor decomposition is also characterized by a notion of rank, which is much lower in the case of this $\boldsymbol{A}$ matrix and hence results in fewer parameters that must be stored. The total number of parameters of the white-box model is also shown in Figure S2 as a reference. The low-rank tensor train matrix representation requires a storage space that is similar in orders of magnitude to the white-box parameters. This beneficial representation is completely due to the structure present in the $\boldsymbol{A}$ matrix, which can be captured by a low-rank tensor decomposition.

The fact that all entries of the $\boldsymbol{A}$ matrix are not completely random results in the corresponding low-rank tensor representation. Precise conditions for when such a low-rank representation is available are given in [7] and [8]. These conditions relate to different structures, such as an algebraic and displacement structure as well as smoothness, that are present in the tensor. Smoothness relates to tensors for which the entries are the evaluations of smooth functions. An example of a displacement structure is the block Hankel structure that is common in subspace identification algorithms. A more intuitive interpretation of *"structure"* is that all tensor entries can be computed from a smaller set of numbers. For example, the rank 1 matrix decomposition

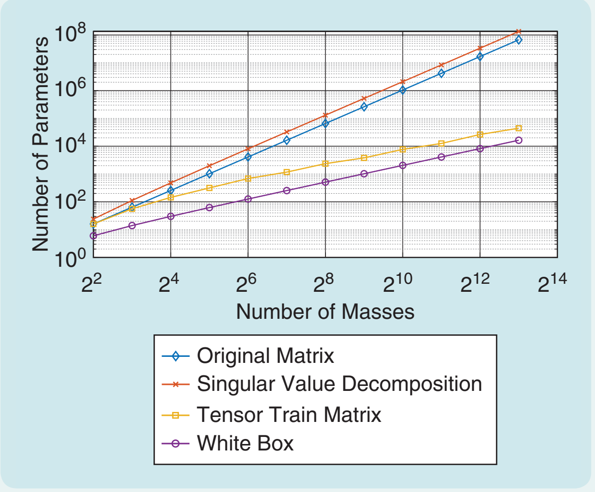


FIGURE S2 A plot of the number of parameters of each representation of the state dynamics $\boldsymbol{A}$ matrix versus the total number of masses in the dynamic system. The tensor train matrix representation requires a storage cost that is similar to the white-box model.

$$\begin{pmatrix} 3 & 4 & 5 \\ 6 & 8 & 10 \\ 9 & 12 & 15 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$$

implies that all nine matrix entries on the left-hand side can be computed from the six numbers on the right-hand side. Hence, the nine matrix entries are not entirely "independent" from one another. The different tensor ranks discussed in this article generalize this idea to higher dimensions.

product of matrices $A \in \mathbb{R}^{I \times J}$, and $B \in \mathbb{R}^{K \times L}$ is the $KI \times LJ$ matrix

$$C = A \otimes B = \begin{pmatrix} A(1,1)B & \cdots & A(1,J)B \\ \vdots & \ddots & \vdots \\ A(I,1)B & \cdots & A(I,J)B \end{pmatrix}.$$

In terms of indices, the Kronecker product is written as

$$C([ki],[lj]) = A(i,j)\,B(k,l).$$

Observe that each of the indices of B is first in the multi-indices of C. This is also the reason why the Kronecker product is not commutative. Finally, the Khatri–Rao product of matrices $A \in \mathbb{R}^{I \times J}$, and $B \in \mathbb{R}^{K \times J}$ is the $KI \times J$ matrix

$$C = A \odot B = (A(:,1) \otimes B(:,1) \;\cdots\; A(:,J) \otimes B(:,J)),$$

formed by taking the column-wise Kronecker product A with B. The notation $A(:,j)$ denotes the jth column of the matrix A, where the colon operator (:) replaces the row index to include the whole range of possible index values.

At first, one might think that higher-order tensors cannot be visualized. However, a visualization of higher dimensions is possible. In fact, tensor diagrams can be a powerful tool to represent complicated tensor expressions. More information about tensor diagrams can be found in "Tensor Diagrams." These diagrams will be extensively used throughout this article.

Tensor Operations

Manipulating tensors will be crucial when describing the different applications. In this section, an overview is given of three important tensor operations. A first operation on tensors is changing the order D of the tensor through the "reshape" operator. The operation

$$\mathcal{B} = \text{reshape}(\mathcal{A}, [J_1, J_2, \ldots, J_K])$$

reshapes a D-way tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ into a K-way tensor $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_K}$. The total number of entries does not change through this reshape operation; that is, $\Pi_{d=1}^{D} I_d = \Pi_{k=1}^{K} J_k$. A tensor order decrease through the

Sparse Models as Restricted Low-Rank Tensor Decompositions

Sparse parametric models can be interpreted in terms of low-rank tensor decompositions. Consider, for example, the single-input, single-output discrete time model

$$y(n) = f(u(n), u(n-1), \ldots, u(n-29)),$$

where $f(\cdot)$ is a 30-variate polynomial in the time-shifted input signal $u(n)$ of total degree 3. A parameterization of this model structure requires $\binom{3+30}{30} = 5456$ parameters, one for each distinct monomial. Now, consider the sparse model

$$y(n) = c_0 u(n) + c_1 u(n-10) u(n-20) + c_2 u(n-29)^3 \qquad \text{(S3)}$$

parameterized by the unknown model coefficients c_0, c_1, and c_2. All remaining 5453 parameters are considered to be exactly zero in this model. How can this sparse parametric model be rewritten as a low-rank tensor decomposition?

First, all possible 5456 monomials are written as the Kronecker product

$$\boldsymbol{u}_n^3 = \begin{pmatrix} 1 \\ u(n) \\ u(n-1) \\ \vdots \\ u(n-29) \end{pmatrix} \otimes \begin{pmatrix} 1 \\ u(n) \\ u(n-1) \\ \vdots \\ u(n-29) \end{pmatrix} \otimes \begin{pmatrix} 1 \\ u(n) \\ u(n-1) \\ \vdots \\ u(n-29) \end{pmatrix} \in \mathbb{R}^{27{,}000},$$

which is the vectorization of a rank 1 symmetric tensor that contains $30^3 = 27{,}000$ entries. The first term of (S3) can then be written as the inner product

$$c_0 u(n) = (\boldsymbol{u}_n^3)^T \left(\begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ c_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right) = (\boldsymbol{u}_n^3)^T (\boldsymbol{e}_1 \otimes \boldsymbol{e}_1 \otimes c_0 \boldsymbol{e}_2),$$

where $\boldsymbol{e}_k$ denotes the kth canonical basis vector in $\mathbb{R}^{30}$. The sparse model (S3) can hence be written as

$$y(n) = (\boldsymbol{u}_n^3)^T (\boldsymbol{e}_1 \otimes \boldsymbol{e}_1 \otimes c_0 \boldsymbol{e}_2 + \boldsymbol{e}_1 \otimes \boldsymbol{e}_{11} \otimes c_1 \boldsymbol{e}_{21} + \boldsymbol{e}_{30} \otimes \boldsymbol{e}_{30} \otimes c_2 \boldsymbol{e}_{30}),$$

where the sum of the three Kronecker product terms is the vectorization of a rank 3 canonical polyadic tensor decomposition. Any sparse model of R terms can in this way be written as a rank R canonical polyadic decomposition. The sparse model can, in fact, be interpreted as a very restricted tensor decomposition in that the vectors in the decomposition are limited to canonical basis vectors $\boldsymbol{e}_k$.

By lifting this restriction, the rank 3 model is written as

$$y(n) = (\boldsymbol{u}_n^3)^T (\boldsymbol{a}_1^{(1)} \otimes \boldsymbol{a}_1^{(2)} \otimes \boldsymbol{a}_1^{(3)} + \boldsymbol{a}_2^{(1)} \otimes \boldsymbol{a}_2^{(2)} \otimes \boldsymbol{a}_2^{(3)} + \boldsymbol{a}_3^{(1)} \otimes \boldsymbol{a}_3^{(2)} \otimes \boldsymbol{a}_3^{(3)}),$$

where $\boldsymbol{a}_1^{(1)}, \ldots, \boldsymbol{a}_3^{(3)}$ are arbitrary vectors in $\mathbb{R}^{30}$. The total number of model parameters increases from three to $3 \cdot 3 \cdot 30 = 270$, which is still a storage gain of a factor 100 compared to the original tensor of 27,000 entries. In this way, the model becomes more expressive in that all 5456 monomials can now contribute in predicting $y(n)$. If the control engineer has no reason to believe that the "true model" is sparse and therefore belongs to this restricted low-rank tensor decomposition model class, then the use of canonical basis vectors $\boldsymbol{e}_k$ in the decomposition can be removed while keeping the low-rank structure. Low-rank tensor decompositions can thus be seen as a more generic choice than sparse models.

reshape operation is equivalent to grouping several indices together into one multi-index according to (1). Likewise, an increase of the tensor order is equivalent to the splitting of an index $i = [i_1 i_2 \cdots i_D]$ into D separate indices $i_1, i_2, \ldots, i_D$, such that (1) is also satisfied. The most common use of the reshape operator is for converting a tensor into a vector or matrix and vice versa. The conversion of a D-way tensor $\mathcal{A}$ into a vector $\boldsymbol{a}$ is called the vectorization, denoted $\text{vec}(\mathcal{A})$. For example, the vectorization of the matrix

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

is the vector

$$\boldsymbol{a} = \text{vec}(A) = (1 \;\; 2 \;\; 3 \;\; 4 \;\; 5 \;\; 6)^T.$$

In terms of indices, the relationship between the entries of a matrix A and its vectorization $\boldsymbol{a}$ can be written as

$$A(i_1, i_2) = \boldsymbol{a}([i_1 i_2]).$$

Using the vectorization operator, it is also possible to denote the relationship between the outer product and the Kronecker product

$$\text{vec}(\boldsymbol{a}^{(1)} \circ \boldsymbol{a}^{(2)} \circ \cdots \circ \boldsymbol{a}^{(D)}) = \boldsymbol{a}^{(D)} \otimes \cdots \otimes \boldsymbol{a}^{(2)} \otimes \boldsymbol{a}^{(1)},$$

which follows from the various definitions. Note that the order of the vectors in the Kronecker product is reversed. Essentially, both the outer product and Kronecker product compute products among all possible combinations of the

Tensor Diagrams

Writing down tensor equations quickly becomes cumbersome as the number of indices increases. A simplified version of the Penrose tensor diagram notation [57] is used in this article to visualize tensor operations. Every tensor is represented by a node (a circle), and each index corresponds to an edge (a line). Tensors of order zero up to three are illustrated in Figure S3. The Penrose diagram of an index summation is obtained by connecting the lines between two nodes in the diagram. For example, the matrix–matrix product of $\boldsymbol{A} \in \mathbb{R}^{N \times R}$ with $\boldsymbol{B} \in \mathbb{R}^{R \times M}$,

$$C(n, m) = \sum_r \boldsymbol{A}(n, r)\, \boldsymbol{B}(r, m),$$

is represented in Figure S4 as the connecting edge between the nodes $\boldsymbol{A}$ and $\boldsymbol{B}$. The order of the resulting tensor in a diagram can always be deduced from the number of edges that are not connected, the so-called dangling edges. In Figure S4, for example, there are two such edges. Hence, the diagram represents a matrix. Of course, index summations involving tensors of higher order can also be visualized with a diagram. For example, a Tucker decomposition of a third-order tensor $\mathcal{A}$ is a contraction of a third-order tensor $\mathcal{S}$ with a matrix on each of its dimensions:

$$\mathcal{A} = \mathcal{S} \times_1 \boldsymbol{A}^{(1)} \times_2 \boldsymbol{A}^{(2)} \times_3 \boldsymbol{A}^{(3)}. \tag{S4}$$

Figure S5 shows the corresponding tensor diagram. The resulting tensor $\mathcal{A}$ also has an order of three, and its entries are determined by

$$\mathcal{A}(i_1, i_2, i_3) = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \mathcal{S}(r_1, r_2, r_3)\, \boldsymbol{A}^{(1)}(i_1, r_1)\, \boldsymbol{A}^{(2)}(i_2, r_2) \times \boldsymbol{A}^{(3)}(i_3, r_3). \tag{S5}$$

The power of tensor diagrams becomes more apparent as the number of contractions increases since the need to write out all index summations explicitly, as in (S5), or define a new notation, as in (S4), is removed. Tensor decompositions are also called tensor networks, as different decompositions correspond to various topologies of interconnected tensors in a diagram.

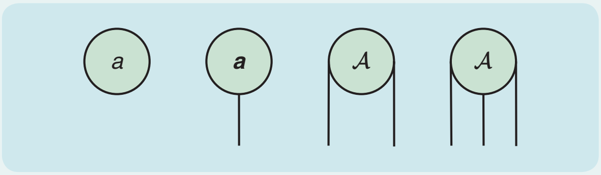


FIGURE S3 Tensor diagrams of a scalar a, vector $\boldsymbol{a}$, matrix $\boldsymbol{A}$, and three-way tensor $\mathcal{A}$. Each edge represents a dimension of the tensor, thus allowing the visualization of high-dimensional tensors.

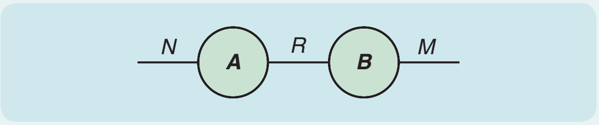


FIGURE S4 A tensor diagram of the multiplication of a matrix $\boldsymbol{A}$ with a matrix $\boldsymbol{B}$. The diagram has two "dangling" edges and therefore represents a matrix. The connected edge of size R represents the summation over the column index of $\boldsymbol{A}$ and row index of $\boldsymbol{B}$.

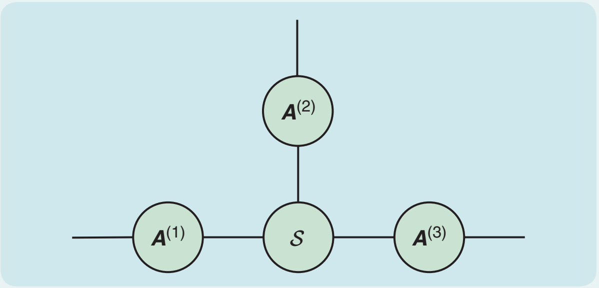


FIGURE S5 A tensor diagram of a Tucker decomposition of a three-way tensor. The three-way tensor $\mathcal{S}$ is contracted along each of its three dimensions with matrices $\boldsymbol{A}^{(1)}$, $\boldsymbol{A}^{(2)}$, $\boldsymbol{A}^{(3)}$. The resulting tensor is also of the third order, since there are three "dangling" edges in the diagram.

vector entries. The outer product stores the result in a tensor, while the Kronecker product stores it in a vector.

A second important operation is the generalization of the matrix transpose to three or more indices. The operator "permute($\mathcal{A}, \boldsymbol{p}$)" rearranges the indices of $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ so that they are in the order specified by the vector $\boldsymbol{p}$. The resulting tensor has the same values of $\mathcal{A}$, but the order of the subscripts needed to access any particular element is rearranged, as specified by $\boldsymbol{p}$. All the elements of $\boldsymbol{p}$ must be unique, real, positive, integer values from one to D. For example, applying the following permutation on the tensor $\mathcal{A} \in \mathbb{R}^{2\times 5\times 3}$,

$$\mathcal{B} = \text{permute}(\mathcal{A}, [3,1,2]),$$

results in a tensor $\mathcal{B} \in \mathbb{R}^{3\times 2\times 5}$, such that

$$\mathcal{A}(i_1, i_2, i_3) = \mathcal{B}(i_3, i_1, i_2).$$

In the same way, the conventional matrix transpose $\boldsymbol{B} = \boldsymbol{A}^T$ can be written as

$$\boldsymbol{B} = \text{permute}(A, [2,1]).$$

The third and last tensor operation discussed in this section is the summation over indices, also called the contraction of indices. A matrix multiplication is probably the most familiar instance of an index contraction. Indeed, rewriting the matrix product $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$ in terms of the matrix entries

$$\boldsymbol{C}(i_1, j_1) = \sum_{i_2=1}^{I_2} \boldsymbol{A}(i_1, i_2)\boldsymbol{B}(i_2, j_1)$$

shows that the index i_2 is summed over all its values. A particularly common index contraction in the context of tensors is the d-mode product of a tensor with a matrix. The d-mode product $\mathcal{A} \times_d \boldsymbol{U}_d$ of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ with a matrix $\boldsymbol{U}_d \in \mathbb{R}^{S_d \times I_d}$ is the tensor $\mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_{d-1} \times S_d \times I_{d+1} \times \cdots \times I_D}$ with elements

$$\begin{aligned}&\mathcal{B}(i_1, \ldots, i_{d-1}, s_d, i_{d+1}, \ldots, i_D)\\ &\quad = \sum_{i_d=1}^{I_d} \mathcal{A}(i_1, \ldots, i_{d-1}, i_d, i_{d+1}, \ldots, i_D)\boldsymbol{U}_d(s_d, i_d).\end{aligned} \tag{3}$$

The index i_d has effectively been summed over and therefore has been replaced by the s_d index. Defining the matrix $\boldsymbol{A}_{(d)}$ as

$$\begin{aligned}\boldsymbol{A}_{(d)} = \text{reshape}(&\text{permute}(\mathcal{A}, [d, 1, 2, \ldots, d-1, d+1, \ldots, D]),\\ &[I_d, I_1 \cdots I_D]),\end{aligned}$$

the index contraction (3) can be implemented through the matrix product

$$\boldsymbol{B} = \boldsymbol{U}_d\, \boldsymbol{A}_{(d)}.$$

The desired $\mathcal{B} = \mathcal{A} \times_d \boldsymbol{U}_d$ tensor is then obtained from

$$\begin{aligned}\mathcal{B} = \text{permute}(&\text{reshape}(\boldsymbol{B}, [S_d, I_1, I_2, \ldots, I_D]),\\ &[2, 3, \ldots, d, 1, d+1, \ldots, D]).\end{aligned}$$

Tensor Decompositions

Low-rank approximation of a matrix through a decomposition is well-known, and the singular value decomposition (SVD) is an essential tool in this respect [12]. The SVD of a matrix $\boldsymbol{A} \in \mathbb{R}^{I\times J}$

$$\boldsymbol{A} = \boldsymbol{U}\,\boldsymbol{S}\,\boldsymbol{V}^T \tag{4}$$

consists of two orthogonal matrices $\boldsymbol{U} \in \mathbb{R}^{I\times I}, \boldsymbol{V} \in \mathbb{R}^{J\times J}$ and a diagonal matrix $\boldsymbol{S} \in \mathbb{R}^{I\times J}$ of nonnegative real numbers $\sigma_1 > \sigma_2 > \ldots$, also called singular values. A matrix is, per definition, rank R when it has R nonzero singular values. It follows then that $R = \min(I, J)$. The SVD factor matrices of a rank R matrix can be truncated to dimensions $\boldsymbol{U} \in \mathbb{R}^{I\times R}$, $\boldsymbol{V} \in \mathbb{R}^{J\times R}$, and $\boldsymbol{S} \in \mathbb{R}^{R\times R}$. When $R \ll I, J$, a storage benefit can be obtained by storing the factor matrices $\boldsymbol{US}$, $\boldsymbol{V}$ in memory instead of $\boldsymbol{A}$. The storage complexity of $\boldsymbol{A}$ is in this way reduced from IJ to $(I+J)R$. Truncating the SVD to a dimension $R' < R$ results in an optimal rank R' approximation

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T + \boldsymbol{E},$$

where the approximation error $\boldsymbol{E}$ satisfies $\|\boldsymbol{E}\|_2 = \sigma_{R'+1}$. This optimal approximation property does not apply for higher-order tensors [13].

The generalization of the SVD from matrices to higher-order tensors has led to the discovery of many different tensor decompositions, and each decomposition comes with its own properties and notion of rank. For this reason, it is good to have an overview of commonly used decompositions and how they relate to one another. Three tensor decompositions discussed in this article are the canonical polyadic decomposition (CPD), Tucker decomposition, and tensor train decomposition. These decompositions are summarized in terms of their storage complexity and uniqueness in Table 2. The applications of these three tensor decompositions go far beyond the field of control [11], [14]. The main idea is always the same: a tensor $\mathcal{A}$ that is too large to be stored explicitly in memory

TABLE 2 A summary of commonly used tensor decompositions of a D-way tensor $\mathcal{A}$. The storage complexity calculation assumes uniform tensor dimensions I and uniform tensor rank R.

	Storage Complexity	Unique?
Original tensor $\mathcal{A}$	I^D	
Canonical polyadic decomposition	$O(DIR)$	Yes, under mild conditions
Tucker decomposition	$O(R^D + DIR)$	No
Tensor train decomposition	$O(DIR^2)$	No
Tensor train matrix decomposition	$O(DI^2R^2)$	No

is replaced or approximated by a set of smaller tensors. Both the CPD and Tucker decomposition can be interpreted as direct generalizations of the matrix SVD to higher-order tensors. The key idea to generalize the SVD is to rewrite this decomposition in terms of outer products. Labeling column r of $\boldsymbol{U}$ and $\boldsymbol{V}$ by $\boldsymbol{u}_r$ and $\boldsymbol{v}_r$, respectively, rewrites (4) as

$$\boldsymbol{A} = \sum_{r=1}^{R} \sigma_r \boldsymbol{u}_r \boldsymbol{v}_r^T = \sum_{r=1}^{R} \sigma_r \boldsymbol{u}_r \circ \boldsymbol{v}_r.$$

The notation $\circ$ for the outer product is used here, as it generalizes more easily to the case of an outer product of $D > 2$ vectors

$$\boldsymbol{\mathcal{A}} = \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(D)}. \tag{5}$$

Such a decomposition is called a CPD, as well as CANDECOMP and PARAFAC decomposition [15], [16]. The CP rank is defined as the smallest R for which (5) is an equality. A tensor $\boldsymbol{\mathcal{A}}$, as in (2), is, per definition, a rank 1 tensor. The analog of the matrix singular values σ_r in the CPD is the product of the norms of each of the vectors $\boldsymbol{a}_r^{(d)}$:

$$\boldsymbol{\mathcal{A}} = \sum_{r=1}^{R} \sigma_r \frac{\boldsymbol{a}_r^{(1)}}{\left\|\boldsymbol{a}_r^{(1)}\right\|_2} \circ \frac{\boldsymbol{a}_r^{(2)}}{\left\|\boldsymbol{a}_r^{(2)}\right\|_2} \circ \cdots \circ \frac{\boldsymbol{a}_r^{(D)}}{\left\|\boldsymbol{a}_r^{(D)}\right\|_2}.$$

Unlike the SVD, the CPD does not require the vectors $\boldsymbol{a}^{(d)}$ to be mutually orthogonal. If the R vectors $\boldsymbol{a}_r^{(d)} \in \mathbb{R}^{I_d}$ are concatenated into an $I_d \times R$ matrix $\boldsymbol{A}^{(d)}$, then the CPD can be rewritten as

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{S}} \times_1 \boldsymbol{A}^{(1)} \times_2 \boldsymbol{A}^{(2)} \times_3 \cdots \times_D \boldsymbol{A}^{(D)}, \tag{6}$$

where $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{R \times R \times \cdots \times R}$ is a diagonal D-way tensor that contains the σ_r scalars on the diagonal.

The CPD has been shown to be unique under mild conditions [17], [18]. This uniqueness has proven to be very useful, especially in the field of signal separation [19], [20]. Another application of the CPD in the field of nonlinear system identification is for the approximation of a set of multivariate real polynomials into linear combinations of univariate polynomials [21], [22]. The storage complexity of this decomposition is completely determined by the dimensions of the factor matrices $\boldsymbol{A}^{(d)}$. Assuming $\boldsymbol{\mathcal{A}}$ is a D-way tensor with uniform dimensions I and CP rank R, then the CPD has a storage complexity of DIR. The main problem in the usage of the CPD is the determination of the CP rank, which is an NP-hard problem [23]. A more general tensor decomposition is obtained through a relaxation of the diagonal tensor requirement in (6). Allowing $\boldsymbol{\mathcal{S}}$ to have a different size for each dimension and relaxing the diagonal constraint results in the Tucker decomposition

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{S}} \times_1 \boldsymbol{A}^{(1)} \times_2 \boldsymbol{A}^{(2)} \times_3 \cdots \times_D \boldsymbol{A}^{(D)}, \tag{7}$$

where $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_D}$ is also called the Tucker core tensor [24], [25]. The single CP rank is in this way replaced by a multilinear rank $(R_1, R_2, \ldots, R_D)$ with the constraint that $R_d \leq I_d$.

Assuming a uniform multilinear rank R, the storage complexity of the Tucker decomposition is $R^D + DIR$. The exponential R^D term is due to the Tucker core $\boldsymbol{\mathcal{S}}$. Contrary to the CPD, the Tucker decomposition is not unique. An identity matrix $\boldsymbol{I} = \boldsymbol{T}\boldsymbol{T}^{-1}$ can always be "inserted" between the Tucker core $\boldsymbol{\mathcal{S}}$ and factor matrix $\boldsymbol{A}^{(d)}$. A new Tucker core and factor matrix can then be defined as

$$\tilde{\boldsymbol{\mathcal{S}}} = \boldsymbol{\mathcal{S}} \times_d \boldsymbol{T}^T,$$
$$\tilde{\boldsymbol{A}}^{(d)} = \boldsymbol{T}^{-1} \boldsymbol{A}^{(d)},$$

without changing the underlying tensor. This nonuniqueness provides additional flexibility. For example, a common way to "fix" the Tucker decomposition into a particular form is to require that each of the factor matrices $\boldsymbol{A}^{(d)}$ consists of orthonormal vectors, leading to the so-called higher-order SVD [26]. Diagrams of CPD and Tucker decompositions are provided in "Tensor Diagrams" and consist of a central core tensor that is "surrounded" in each of its dimensions by a factor matrix.

A third tensor decomposition is the tensor train, also called the matrix product state [27]. Rather than being a core tensor surrounded by factor matrices, a tensor train is a linear sequence of tensor cores. A tensor train of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ is defined as a set of D three-way tensor cores $\boldsymbol{\mathcal{A}}^{(d)} \in \mathbb{R}^{R_d \times I_d \times R_{d+1}}$, such that

$$\begin{aligned}\boldsymbol{\mathcal{A}}(i_1, i_2, \ldots, i_D) = &\sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_D=1}^{R_D} \boldsymbol{\mathcal{A}}^{(1)}(r_1, i_1, r_2) \\ &\times \boldsymbol{\mathcal{A}}^{(2)}(r_2, i_2, r_3) \cdots \boldsymbol{\mathcal{A}}^{(D)}(r_D, i_D, r_1).\end{aligned} \tag{8}$$

A diagram of a tensor train is presented in Figure 1, and it illustrates the power of this visualization method: the complexity of (8) is captured in a simple illustration. The minimal values of $R_1, R_2, \ldots, R_D$ such that (8) is an equality are called the tensor train ranks of $\boldsymbol{\mathcal{A}}$. A tensor train satisfies $R_1 = 1$, which implies that the index connecting the first core to the last core is of unit size. A tensor train is called a tensor ring or tensor chain when $R_1 > 1$ [28]–[30]. The ring case will not be considered in the remainder of the article, and the corresponding unit size edge will not be drawn in consequent diagrams anymore. The tensor train is also not unique. An identity matrix $\boldsymbol{I} = \boldsymbol{T}\boldsymbol{T}^{-1}$ can be inserted between any two tensor train cores $\boldsymbol{\mathcal{A}}^{(d-1)}, \boldsymbol{\mathcal{A}}^{(d)}$, such that these cores can be rewritten as

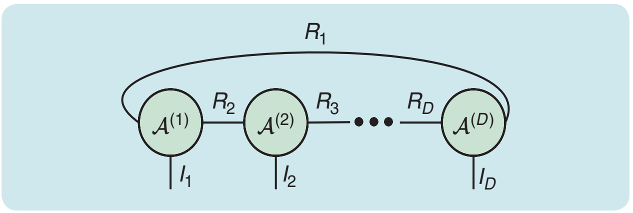


FIGURE 1 A tensor diagram of a tensor train. A tensor train is a linear sequence of D three-way tensors. The edge that corresponds with the index r_1 connects the first core tensor with the last and is of unit size.

$$\tilde{\mathcal{A}}^{(d-1)} = \mathcal{A}^{(d-1)} \times_3 \boldsymbol{T}^T,$$
$$\tilde{\mathcal{A}}^{(d)} = \mathcal{A}^{(d)} \times_1 \boldsymbol{T}^{-1},$$

without changing the underlying tensor. Similar to the Tucker decomposition, the notion of orthogonality can be used to "fix" a certain representation of the tensor train [31].

The tensor decompositions discussed so far have always been in the context of decomposing a given tensor. Through the reshape operation, however, it is also possible to use tensor decompositions for the representation/approximation of vectors and matrices. See "Blessing of Dimensionality" for more details. A particularly useful decomposition is the tensor train matrix, which is an extension of a tensor train to approximate a matrix [32]. Tensor train matrices are also called matrix product operators. Consider a matrix $\boldsymbol{A} \in \mathbb{R}^{I^D \times J^D}$ of exponentially large dimensions. The row index i and column index j can each be split into D indices $i_1,\ldots,i_D$ and $j_1,\ldots,j_D$ through a reshape operation. The resulting 2 D-way tensor can then be decomposed as a tensor train matrix that consists of D four-way tensors $\mathcal{A}^{(d)} \in \mathbb{R}^{R_d \times i_d \times j_d \times R_{d+1}}$, such that the matrix entry $A([i_1 i_2 \cdots i_D],[j_1 j_2 \cdots j_D])$ can be computed from

$$\sum_{r_1=1}^{R_1}\sum_{r_2=1}^{R_2}\cdots\sum_{r_D=1}^{R_D} \mathcal{A}^{(1)}(r_1,i_1,j_1,r_2)\mathcal{A}^{(2)}(r_2,i_2,j_2,r_3)\cdots\mathcal{A}^{(D)}(r_D,i_D,j_D,r_1). \tag{9}$$

The tensor diagram of a tensor train matrix is very similar to the diagram of a tensor train. The only difference is the addition of an extra edge to each tensor core, corresponding to the extra j_d index. A useful feature that distinguishes tensor trains and tensor train matrices from the CPD and Tucker decomposition is that numerous linear algebra operations can be computed directly in the decomposed form. An overview of such linear algebra operations is given in "Linear Algebra Operations With Tensor Trains." This property is the main motivation for the exclusive usage of tensor trains in the applications described in the following.

APPLICATION 1: MULTIPLE-INPUT, MULTIPLE-OUTPUT VOLTERRA SYSTEM IDENTIFICATION

The output $y(n)$ of a causal discrete-time FIR system of order M is described by the linear combination of lagged input values $u(n)$:

$$y(n) = h_0 + \sum_{m_1=0}^{M} h_1(m_1)u(n-m_j). \tag{10}$$

The parameter h_0 is a constant offset, also called the dc offset. The parameter M will be called, from now on, the memory of the system, as it is the maximal lag in the input. A discrete-time, truncated Volterra model generalizes a FIR model to higher orders of nonlinearity. This generalization is obtained by adding homogeneous polynomials of varying degrees d in the lagged input values to (10). These higher-order polynomials can be interpreted as finite higher-order impulse responses. For example, with memory

Blessing of Dimensionality

Tensor decompositions are useful not only in the context of higher-order tensors. Vectors and matrices can benefit from low-rank tensor decompositions that compress them by exploiting correlations between entries. This phenomenon is called the blessing of dimensionality. By increasing the order of a tensor via the reshape operator, the resulting tensor decomposition is described by more rank parameters. Therefore, the decomposition of this tensor receives more degrees of freedom, resulting in additional flexibility. An example of this blessing is described in "Low-Rank Motivation." The blessing of dimensionality for vectors is illustrated in Figure S6. A vector $\boldsymbol{a}$ of length I^D can be reshaped into a D-way tensor $\mathcal{A}$. Assuming that the vector entries are not completely unstructured [7], [8], a low-rank tensor approximation can be computed from $\mathcal{A}$, resulting in a storage-efficient representation of the original vector $\boldsymbol{a}$.

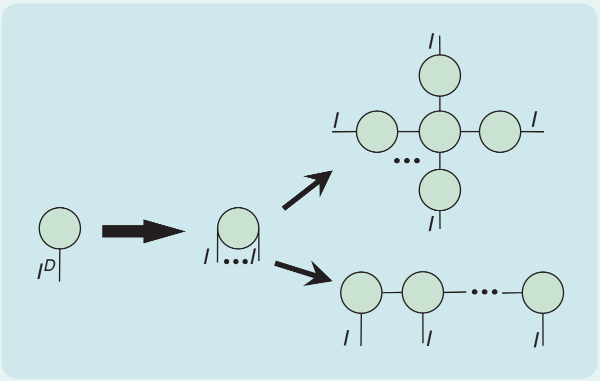


FIGURE S6 A vector of exponential size I^D is first reshaped into a D-way tensor. This tensor can then be represented by any tensor decomposition, resulting in a storage-efficient representation of the original vector.

The decomposition to use depends on the specifics of the application. If the uniqueness of the decomposition is important (for example, to be able to provide some physical meaning to the different factor matrices), then canonical polyadic decomposition should be used. Alternatively, if only the subspaces associated with the different dimensions are important, then the Tucker decomposition suffices. When certain linear algebra operations must be performed on the original vector, then the tensor train decomposition is highly recommended, as explained in "Linear Algebra Operations With Tensor Trains." The same idea of "tensorization" and replacement by a low-rank tensor decomposition can also be applied to matrices of exponential size.

$M=1$ and maximal degree $D=2$, the output of the Volterra model is

$$y(n)=h_0+h_1(0)u(n)+h_1(1)u(n-1)+h_2(0,0)u(n)^2 \\ +h_2(1,0)u(n)u(n-1)+h_2(0,1)u(n)u(n-1) \\ +h_2(1,1)u(n-1)^2.$$

The parameters $h_2(0,0)$, $h_2(1,0)$, $h_2(0,1)$, and $h_2(1,1)$ are called the second-order Volterra kernel coefficients. In general, the output $y(n)$ of a degree D, single-input, single-output (SISO), discrete-time Volterra system is described by

$$y(n)=h_0+\sum_{d=1}^{D}\sum_{m_1=0}^{M}\cdots\sum_{m_d=0}^{M}h_d(m_1,\ldots,m_d)\prod_{j=1}^{d}u(n-m_j). \quad (11)$$

The number of kernel coefficients grows exponentially with the degree of the polynomials, as the dth-order Volterra kernel $h_d(m_1,\ldots,m_d)$ consists of $(M+1)^d$ numbers. In this section, we consider the following Volterra system identification problem: for a given degree D and memory M and a set of N measured input and output values $\{(u(n),y(n))\}_{n=1}^{N}$, estimate all Volterra kernel coefficients $h_0,\ldots,h_D(M,\ldots,M)$. Since the kernel coefficients appear linear in (11), the identification problem can be written as an ordinary least-squares problem

$$\boldsymbol{y}=\boldsymbol{U}\boldsymbol{h}, \quad (12)$$

where the vector $\boldsymbol{y}\in\mathbb{R}^N$ contains the measured outputs, the vector $\boldsymbol{h}\in\mathbb{R}^H$ is the unknown kernel coefficients, and the matrix $\boldsymbol{U}\in\mathbb{R}^{N\times H}$ consists of all the monomials of lagged inputs.

The difficulty in solving (12) lies in the exponential number of kernel coefficients $H=\Sigma_{d=0}^{D}(M+1)^d$. The paradigm to break this curse is to trade storage for computation. All Volterra coefficients are replaced by a tensor decomposition from which the Volterra coefficients can be computed. This idea is not new. For example, Volterra kernels have been expanded on orthonormal basis functions to reduce their complexity [33], [34]. The notion that the kernel coefficients are not completely random but are instead evaluations of smoothly decaying functions can be encoded through the use of low-rank tensor decompositions [7], [8]. Tensors are therefore also suitable candidates for this purpose via the blessing of dimensionality. In [35], both the canonical polyadic and Tucker tensor decompositions are used to approximate each of the Volterra kernels h_d. The approach described in [36] will be briefly discussed in this section. The main differences compared to [35] are threefold: all Volterra kernels are combined into one tensor, a low-rank tensor train matrix decomposition is used, and multiple-input, multiple-output (MIMO) systems are supported.

Tensor Formulation of Volterra Systems

The first step in formulating the MIMO Volterra system lies in rewriting the least-squares problem (12). Suppose there are L outputs and P inputs, which implies that $\boldsymbol{y}(n)\in\mathbb{R}^L$ and $\boldsymbol{u}(n)\in\mathbb{R}^P$. For a given memory M and degree D, the vector $\boldsymbol{u}_n$ is defined as

$$\boldsymbol{u}_n=(1\ \boldsymbol{u}(n)^T\ \cdots\ \boldsymbol{u}(n-M)^T)^T\in\mathbb{R}^{(1+(M+1)P)}.$$

For notational convenience, let $I=(1+(M+1)P)$, and define the vector

$$\boldsymbol{u}_n^D:=\overbrace{\boldsymbol{u}_n\otimes\boldsymbol{u}_n\otimes\cdots\otimes\boldsymbol{u}_n}^{D\text{ times}}\in\mathbb{R}^{I^D}. \quad (13)$$

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

$$\boldsymbol{y}(n)=\boldsymbol{H}\,\boldsymbol{u}_n^D, \quad (14)$$

where each row of the matrix $\boldsymbol{H}\in\mathbb{R}^{L\times I^D}$ contains all the Volterra kernel coefficients responsible for one of the L outputs. The concatenation of (14) over all measured samples $n=1,\ldots,N$ results in the rewritten least-squares problem

$$\underbrace{\begin{pmatrix}\boldsymbol{y}(1)^T\\ \boldsymbol{y}(2)^T\\ \vdots\\ \boldsymbol{y}(N)^T\end{pmatrix}}_{N\times L}=\underbrace{(\boldsymbol{u}_1^D\ \ \boldsymbol{u}_2^D\ \ \cdots\ \ \boldsymbol{u}_N^D)^T}_{N\times I^D}\boldsymbol{H}^T. \quad (15)$$

Also, for this linear system, the exponential dimension I^D creates a problem. One possible solution is to replace the unknown $\boldsymbol{H}^T$ matrix with the tensor train matrix

$$\mathcal{H}^{(1)}\in\mathbb{R}^{1\times L\times I\times R_2}, \\ \mathcal{H}^{(2)}\in\mathbb{R}^{R_2\times 1\times I\times R_3}, \\ \vdots \\ \mathcal{H}^{(D)}\in\mathbb{R}^{R_D\times 1\times I\times 1}.$$

By convention, the row index l of $\boldsymbol{H}$ is "split" into D indices, where the first index has dimension L and all others have unit dimensions. Equation (14) can now be rewritten as

$$\underbrace{\boldsymbol{y}(n)}_{L\times 1}=\underbrace{(\mathcal{H}^{(1)}\times_3\boldsymbol{u}_n)}_{L\times R_2}\underbrace{(\mathcal{H}^{(2)}\times_3\boldsymbol{u}_n)}_{R_2\times R_3}\cdots\underbrace{(\mathcal{H}^{(D)}\times_3\boldsymbol{u}_n)}_{R_D\times 1}, \quad (16)$$

which is illustrated with a tensor diagram in Figure 2. The diagram has one "dangling" edge, indicating that the tensor obtained after all contractions is a vector. All factors $\{(\mathcal{H}^{(d)}\times_3\boldsymbol{u}_n)\}_{d=1}^{D-1}$ are matrices, and $(\mathcal{H}^{(D)}\times_3\boldsymbol{u}_n)$ is a vector.

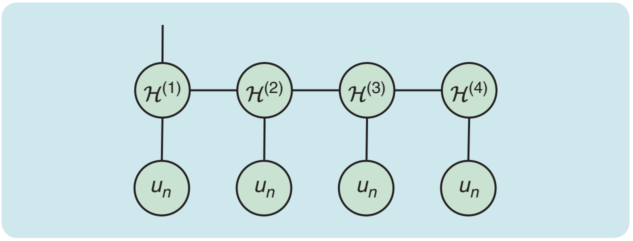


FIGURE 2 A tensor diagram of (16) for a four-way tensor. Each factor $\mathcal{H}^{(d)}\times_3\boldsymbol{u}_n$ in (16) is represented by a tensor train matrix core $\mathcal{H}^{(d)}$ that is connected to a vector $\boldsymbol{u}_n$. The whole diagram has one "dangling" edge, which means that the whole diagram represents a vector.

Equation (16) is therefore a sequence of matrix multiplications that ends with a matrix vector product, resulting in the desired vector of outputs $\boldsymbol{y}(n)$.

With this new tensor train matrix formulation, the Volterra system identification problem can be restated as: for a given degree D and memory M and a set of N measured

Linear Algebra Operations With Tensor Trains

A convenient feature of tensor trains and tensor train matrices is that many linear algebra operations on the underlying vectors and matrices can be directly performed on these decompositions. A brief overview of some useful linear algebra operations is provided in this sidebar. Some operations lead to tensor train ranks of increased size. A rounding procedure [27] can then be used to truncate the ranks for a given error tolerance.

ADDITION

The addition of two tensors $\mathcal{A},\mathcal{B} \in \mathbb{R}^{I\times\cdots\times I}$ in tensor train form can be computed directly with the tensor train cores. If the tensor train ranks of $\mathcal{A}$ and $\mathcal{B}$ are denoted by $R_1,\ldots,R_D$ and $S_1,\ldots,S_D$, respectively, then the tensor train cores of $\mathcal{C}=\mathcal{A}+\mathcal{B}$ are determined by

$$\mathcal{C}^{(1)}(1,:,:) = \left(\mathcal{A}^{(1)}(1,:,:) \;\; \mathcal{B}^{(1)}(1,:,:)\right) \in \mathbb{R}^{I\times(R_2+S_2)},$$
$$\mathcal{C}^{(d)}(:,i_d,:) = \begin{pmatrix} \mathcal{A}^{(d)}(:,i_d,:) & 0 \\ 0 & \mathcal{B}^{(d)}(:,i_d,:) \end{pmatrix} \in \mathbb{R}^{(R_d+S_d)\times(R_{d+1}+S_{d+1})} \;\; (2 \le d \le D-1),$$
$$\mathcal{C}^{(D)}(:,:,1) = \begin{pmatrix} \mathcal{A}^{(D)}(:,:,1) \\ \mathcal{A}^{(D)}(:,:,1) \end{pmatrix} \in \mathbb{R}^{(R_D+S_D)\times I}.$$

The tensor train ranks of the sum $\mathcal{C}=\mathcal{A}+\mathcal{B}$ are the sum of the respective tensor train ranks.

INNER PRODUCT OF TENSORS

The inner product of two tensors $\mathcal{A}$ and $\mathcal{B}$ involves a summation over all indices,

$$\sum_{i_1=1}^{I_1}\cdots\sum_{i_D=1}^{I_D} \mathcal{A}(i_1,\ldots,i_D)\,\mathcal{B}(i_1,\ldots,i_D),$$

resulting in a scalar. A tensor diagram of the inner product of two four-way tensors in tensor train form is in Figure S7. Each row of the diagram is a tensor train. The connecting edges between the two tensor trains are the summations over the indices i_1, i_2, i_3, and i_4. The absence of any remaining "dangling" edges indicates that the whole diagram represents a scalar.

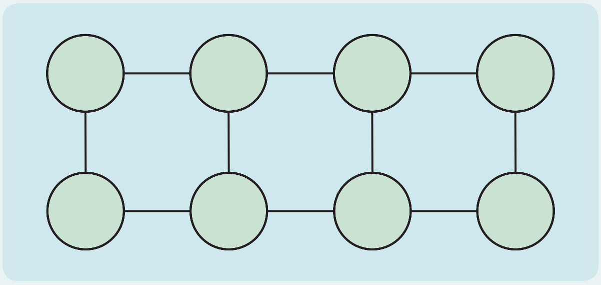


FIGURE S7 A tensor diagram of the inner product between two four-way tensors in tensor train form. The absence of any "dangling" edges implies that the resulting diagram is a scalar.

OUTER PRODUCT OF VECTORS

The outer product of two vectors can be written as an index contraction through the insertion of a dummy index of unit dimension

$$\boldsymbol{C} = \boldsymbol{a} \circ \boldsymbol{b} \Rightarrow \boldsymbol{C}(i,j) = \sum_{r=1}^{1} \boldsymbol{a}(i,r)\,\boldsymbol{b}(r,j).$$

The same idea can be applied in tensor train form. The tensor diagram in Figure S8 illustrates how the two tensor trains of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ can be combined to form the tensor train matrix of $\boldsymbol{C}$. By summing over the unit size dummy indices, tensor train cores are "merged" together. In this way, a tensor

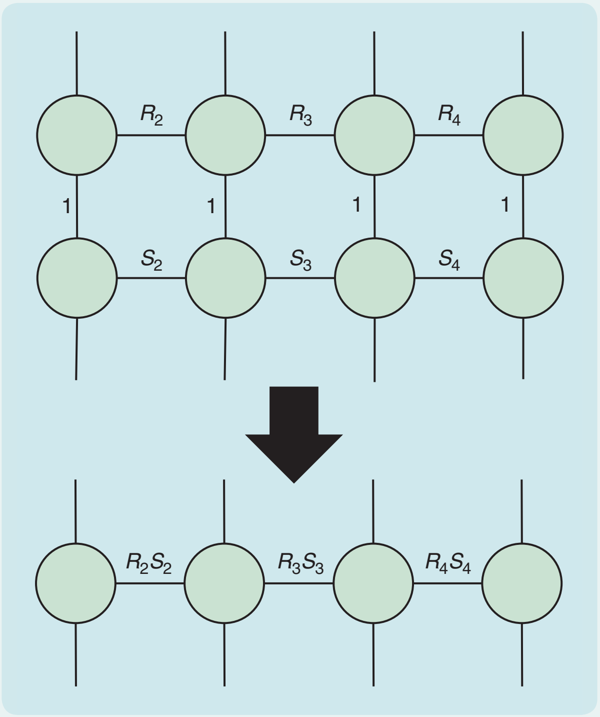


FIGURE S8 A tensor diagram of the outer product between two vectors in tensor train form. Two vectors are shown as tensor trains with ranks R_2, R_3, and R_4 and S_2, S_3, and S_4, respectively. The unit size index contractions between the two tensor trains form the "glue" connecting the two tensor trains together. After summing over all unit size edges, a tensor train matrix is obtained. The resulting tensor train ranks are R_2S_2, R_3S_3, and R_4S_4. All edges pointing downward constitute the row index of the resulting rank 1 matrix, and all upward-pointing edges constitute the column index.

input and output values $\{(u(n), y(n))\}_{n=1}^{N}$, estimate all tensor train matrix cores $\mathcal{H}^{(1)}, \ldots, \mathcal{H}^{(D)}$. Additional parameters are the tensor train matrix ranks $R_2, \ldots, R_D$, which can either be set in advance by or determined from the measured data, at an additional computational cost. In the interest of space, only the algorithm for fixed ranks will be

train matrix is obtained with ranks that are the product of the corresponding ranks of the two tensor trains.

MATRIX MULTIPLICATION

The multiplication of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ written in terms of indices is

$$\boldsymbol{C}(i,j) = \sum_{r=1}^{R} \boldsymbol{A}(i,r)\,\boldsymbol{B}(r,j).$$

In tensor train matrix form, each of the indices i, j, and r is split into D separate indices. The single summation over r will therefore also be split into D summations. The tensor diagram of a matrix multiplication in tensor train matrix form is almost identitical to the outer product in Figure S8. Again, corresponding cores of the tensor train matrices are merged through the summation over the column index of the underlying matrix. Also in this case, corresponding ranks of the tensor train matrices are multiplied. The only difference between the two diagrams is the dimensions of the contracted indices.

THIN QR FACTORIZATION

The thin QR factorization of a matrix $\boldsymbol{A} \in \mathbb{R}^{I^D \times R}$ is

$$\boldsymbol{A} = \boldsymbol{Q}\,\boldsymbol{R},$$

with $\boldsymbol{Q} \in \mathbb{R}^{I^D \times R}$ a matrix with orthonormal columns and $\boldsymbol{R} \in \mathbb{R}^{R \times R}$ an upper triangular matrix. Assume that $R \ll I^D$. If $\boldsymbol{A}$ is given in tensor train matrix form, then it is possible to compute the $\boldsymbol{Q}$ matrix in tensor train matrix form. The algorithm is summarized in Algorithm S1. The assumption is made that the last core of the tensor train matrix of $\boldsymbol{A}$ contains the column index of dimension R. The algorithm consists of a sequence of thin QR factorizations, whereby the $\boldsymbol{Q}_d$ factor is retained as the core for the tensor train of $\boldsymbol{Q}$ and the $\boldsymbol{R}_d$ factor is "absorbed" by the next core of $\boldsymbol{A}$. The last computed $\boldsymbol{R}$ factor is the desired upper triangular matrix. The sequence of orthogonal tensor train matrix cores for $\boldsymbol{Q}$ ensures that $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_R$. A tensor diagram of thin QR factorization in tensor train matrix form appears in Figure S9. The thin QR factorization in tensor train matrix form allows the computation of the Moore–Penrose inverse of a matrix $\boldsymbol{A}$ also directly in tensor train matrix form as

$$\boldsymbol{A}^{-1} = \boldsymbol{R}^{-1}\,\boldsymbol{Q}^T.$$

The permutation of the $\boldsymbol{Q}$ matrix in tensor train matrix form is obtained by a permutation of the row index with the column index of each tensor train matrix core tensor.

ALGORITHM S1 Thin QR factorization of a matrix in tensor train matrix form.

Input: tensor train matrix of $\mathcal{A} \in \mathbb{R}^{I^D \times R}$ with $\mathcal{A}^{(D)} \in \mathbb{R}^{R_D \times I \times R \times 1}$ and $R_D I \geq R$.
Output: tensor train matrix of $\boldsymbol{Q}$ and $\boldsymbol{R}$.

```
1:  for d = 1:D − 1 do
2:      $\mathcal{A}_d \leftarrow \text{reshape}(\mathcal{A}^{(d)}, [R_d I, R_{d+1}])$            % $R_1 = 1$
3:      $\mathcal{Q}_d, \mathcal{R}_d \leftarrow \text{QR}(\boldsymbol{A}_d)$                              % thin QR
4:      $\mathcal{Q}^{(d)} \leftarrow \text{reshape}(\mathcal{Q}_d, [R_d, I, R_{d+1}])$
5:      $\mathcal{A}_{d+1} \leftarrow \boldsymbol{R}_d\,\text{reshape}(\mathcal{A}^{(d+1)}, [R_{d+1}, IR_{d+2}])$   % matrix product
6:      $\mathcal{A}^{(d+1)} \leftarrow \text{reshape}(\boldsymbol{A}_{d+1}, [R_{d+1}, I, R_{d+2}])$
7:  end for
8:  $\boldsymbol{A}_D \leftarrow \text{reshape}(\mathcal{A}^{(D)}, [R_D I, R])$
9:  $\boldsymbol{Q}_D, \boldsymbol{R} \leftarrow \text{QR}(\boldsymbol{A}_D)$                                     % thin QR
10: $\mathcal{Q}^{(D)} \leftarrow \text{reshape}(\boldsymbol{Q}_D, [R_D, I, R, 1])$
```

THIN SINGULAR VALUE DECOMPOSITION

Algorithm S1 can be adjusted to compute a thin singular value decomposition (SVD)

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

with $\boldsymbol{U} \in \mathbb{R}^{I^D \times R}$ a matrix with orthonormal columns, $\boldsymbol{S} \in \mathbb{R}^{R \times R}$ a diagonal matrix, and $\boldsymbol{V} \in \mathbb{R}^{R \times R}$ orthogonal. All orthogonal tensors of the tensor train matrix are renamed $\mathcal{U}^{(d)}$, and line 9 is replaced with

$$\boldsymbol{U}_D, \boldsymbol{S}, \boldsymbol{V} \leftarrow \text{SVD}(\boldsymbol{A}_D).$$

The tensor diagram of the thin SVD is almost identical to the diagram of the thin QR factorization. The $\boldsymbol{R}$ matrix in the diagram is replaced by the matrix product $\boldsymbol{S}\boldsymbol{V}^T$. The Moore–Penrose inverse of a matrix $\boldsymbol{A}$ can then be computed directly in tensor train matrix form as

$$\boldsymbol{A}^{-1} = \boldsymbol{V}\,\boldsymbol{S}^{-1}\,\boldsymbol{U}^T.$$

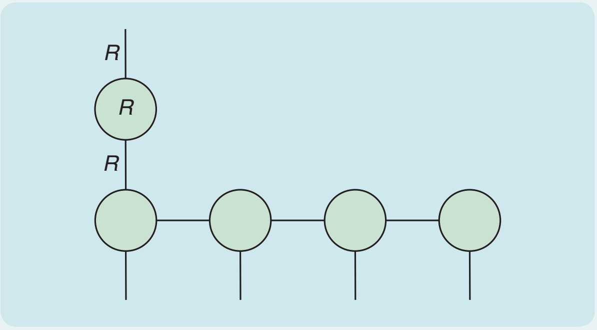


FIGURE S9 A tensor diagram of the thin QR factorization of a matrix $\boldsymbol{A} \in \mathbb{R}^{I^D \times R}$. The bottom row of tensors consists of reshaped orthogonal matrices and represents the factor matrix $\boldsymbol{Q} \in \mathbb{R}^{I^D \times R}$. The $\boldsymbol{R}$ matrix is upper triangular.

discussed. A detailed description of how the ranks can be automatically determined from the measured data can be found in [36].

Alternating Linear Scheme Volterra Identification Algorithm

The alternating linear scheme (ALS) is the workhorse algorithm concerning the computation of tensor decompositions [9], [37], and it can be readily applied to the Volterra system identification problem. The core idea of the ALS algorithm is to initialize each of the tensor train matrix cores, then update each core separately while keeping all other cores fixed. This procedure of updating will lead to a stable, efficient, and reliable convergence to a stationary point of the problem, which is not guaranteed to be the global optimum. The local linear convergence of the ALS algorithm in the case of tensor trains has been studied in [38]. A key ingredient in the formulation of the ALS is (14), rewritten in terms of one tensor train matrix core. Figure 3 illustrates how this rewriting can be done for the case where the dth core is updated. By defining a new matrix and vector

$$\boldsymbol{U}_{<d} = (\boldsymbol{\mathcal{H}}^{(1)} \times_2 \boldsymbol{u}_n) \cdots (\boldsymbol{\mathcal{H}}^{(d-1)} \times_2 \boldsymbol{u}_n) \in \mathbb{R}^{L \times R_d},$$
$$\boldsymbol{u}_{>d} = (\boldsymbol{\mathcal{H}}^{(d+1)} \times_2 \boldsymbol{u}_n) \cdots (\boldsymbol{\mathcal{H}}^{(D)} \times_2 \boldsymbol{u}_n) \in \mathbb{R}^{R_{d+1} \times 1},$$

then (14) can be rewritten as

$$\underbrace{y(n)}_{L\times 1} = \underbrace{\left(u_{>d}^T \otimes u_n^T \otimes U_{<d}\right)}_{L \times R_d I R_{d+1}} \underbrace{\mathrm{vec}\left(H^{(d)}\right)}_{R_d I R_{d+1} \times 1}. \qquad (17)$$

The two Kronecker products are indicated in Figure 3 by the two unit size edges. The least-squares problem to update $\boldsymbol{\mathcal{H}}^{(d)}$ is then obtained by considering all N samples:

$$\begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix} = \begin{pmatrix} \boldsymbol{U}_{>d}^T \otimes \boldsymbol{u}_1^T \otimes \boldsymbol{U}_{<d} \\ \boldsymbol{U}_{>d}^T \otimes \boldsymbol{u}_2^T \otimes \boldsymbol{U}_{<d} \\ \vdots \\ \boldsymbol{U}_{>d}^T \otimes \boldsymbol{u}_N^T \otimes \boldsymbol{U}_{<d} \end{pmatrix} \mathrm{vec}(\boldsymbol{\mathcal{H}}^{(d)}). \qquad (18)$$

Solving the linear system (18) has a computational complexity of $O(N(R_d I R_{d+1})^2)$, which is an improvement over the exponential complexity of the original system. Small values for R_d and R_{d+1}, and hence a low-rank representation of the matrix $\boldsymbol{H}$, are an essential ingredient in making high-order Volterra system identification feasible. Algorithm 1 summarizes the whole identification algorithm, where each core of the tensor train matrix is updated one after another. The initial guess for the tensor train matrix cores is usually chosen randomly. The algorithm can be run for a fixed number of iterations, or a threshold can be set on the relative error of the residuals of (18). An additional orthogonalization step can be introduced for more numerical stability [36]. The ALS algorithm can also be modified such that it learns the required tensor train ranks from the measured data [36, p. 32]. In the presence of measurement noise, however, the obtained ranks will be larger than when no measurement noise is present. The low-rank structure in the decomposition can thus be interpreted as a kind of regularization to prevent overfitting. A similar ALS algorithm applied to nonlinear identification with B-splines basis functions, together with an additional smoothness-inducing regularization term, is described in [39].

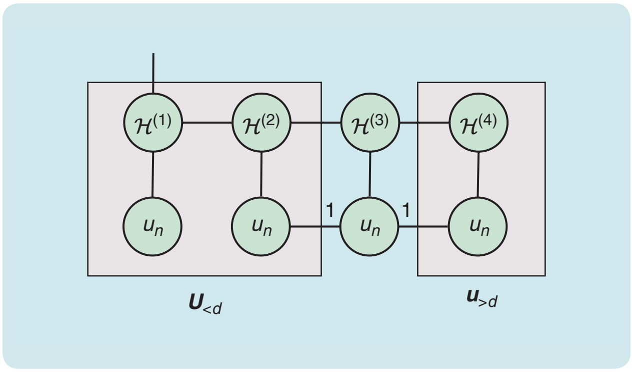


FIGURE 3 The definitions of the matrix $\boldsymbol{U}_{<d}$ and vector $\boldsymbol{u}_{>d}$ with a tensor diagram. The two unit size connections in the diagram correspond to the two Kronecker products in (17).

ALGORITHM 1 The alternating linear scheme Volterra identification.

Input: measurements $\{(\boldsymbol{u}(n), \boldsymbol{y}(n))\}_{n=1}^N$, initial tensor train matrix cores $\mathcal{H}^{(d)}$.

Output: tensor train matrix cores that solve (16).

```
1: while stopping criterion not true do
2:   for $d = 1 : D-1$ do
3:     $\mathcal{H}^{(d)} \leftarrow$ Solve (18)
4:   end for
5:   for $d = D : -1 : 2$ do
6:     $\mathcal{H}^{(d)} \leftarrow$ Solve (18)
7:   end for
8: end while
```

Double-Balanced Mixer Identification

This experiment considers a double-balanced mixer used for upconversion [40]. The reason this particular example was chosen is because it is not a weakly nonlinear system and therefore cannot be reasonably approximated by a low-order Volterra system. The output radio-frequency (RF) signal is determined by a 100-Hz, sine, low-frequency (LO) signal and a 300-Hz, square-wave, intermediate-frequency (IF) signal. A phase difference of $\pi/8$ is present between the LO and IF signals. All time series were sampled at 5 kHz for 1 s. We investigate the effect of additive output noise on the identified models. Define five different noise levels, which are added to the measured RF output, that generate signals with signal-to-noise ratios (SNRs) ranging from 11 to 25 dB. The first 700 samples of the inputs and the noisy output are then used to identify an $M = 2$, $D = 11$, two-input, one-output Volterra system using Algorithm 1. The $\boldsymbol{H}$ matrix consists of 48,828,125 entries. The tensor train ranks are all set to a fixed value of five.

The identified models were then used to simulate the remaining 4300 samples of the output. The SNR of the simulated output was computed by comparing the simulated output with the original noiseless output. Table 3 lists the SNR of the signals used in the identification, the relative residual of the simulated output, the runtime of the identification, and the SNR of the simulated signal. The SNR of the simulated output is defined as

$$10\log_{10}\left(\frac{\sum_n y(n)^2}{\sum_i (y(n)-\hat{y}(n))^2}\right),$$

where $y(n)$ is the output signal uncorrupted by noise and $\hat{y}(n)$ is the simulated output. As expected, a gradual improvement of the relative residual can be seen as the SNR of the signals used for identification increases. Although the residual remains high throughout the different SNR levels, the SNR of the simulated output is much better, with a consistent increase of 11 dB. The runtime varies between 2 and 6 s. Figure 4 provides the simulated output on the validation data for three Volterra models identified under three different SNR levels (11, 16, and 25 dB).

APPLICATION 2: KALMAN FILTERING FOR RECURSIVE VOLTERRA SYSTEM IDENFICATION

An alternative to the ALS algorithm for Volterra identification is a Bayesian filtering approach [41]. Such a recursive filtering method comes with the additional benefit of having a measure of uncertainty on the estimated model parameters. This uncertainty can then be considered when making predictions of future output values. The derivation summarized here follows the Kalman filter technique in [42] rather than the approach in [43] and [44] for two reasons. First, the Kalman filter equations do not have to be written as tensor equations, which results in a simpler derivation compared to [43] and [44]. Second, correlations between the L Volterra outputs cannot be considered using the approach in [43] and [44]. The starting point of the derivation is not the linear system (18) but the state-space model

$$\begin{aligned} \boldsymbol{h}(n+1) &= \boldsymbol{h}(n) + \boldsymbol{w}(n), \\ \boldsymbol{y}(n) &= \boldsymbol{C}(n)\boldsymbol{h}(n) + \boldsymbol{e}(n). \end{aligned} \tag{19}$$

The state vector $\boldsymbol{h} \in \mathbb{R}^{LI^D}$ is defined as

$$\boldsymbol{h} = \mathrm{vec}(\boldsymbol{H}),$$

where $\boldsymbol{H}$ is the $L \times I^D$ matrix of Volterra kernel coefficients in (14). Both the process noise $\boldsymbol{w}(n)$ and measurement noise $\boldsymbol{e}(n)$ are assumed to be stationary Gaussian white noise processes:

$$\mathbb{E}\left(\begin{pmatrix} \boldsymbol{w}(n) \\ \boldsymbol{e}(n) \end{pmatrix}\left(\boldsymbol{w}(m)^T\ \boldsymbol{e}(m)^T\right)\right) = \begin{pmatrix} \boldsymbol{R}_w & 0 \\ 0 & \boldsymbol{R}_e \end{pmatrix}\delta_{nm}.$$

In this framework, the state vector $\boldsymbol{h}(n)$ is time varying due to the process noise $\boldsymbol{w}(n)$. This process noise term provides additional flexibility in that it allows the addition of a forgetting factor λ that weighs past observations $\boldsymbol{y}(n)$ during the recursive identification. Assuming this particular definition of the $\boldsymbol{h}$ vector, it can then be shown that the corresponding time-varying measurement matrix $\boldsymbol{C}(n)$ is

$$\boldsymbol{C}(n) = (\boldsymbol{u}_n^D)^T \otimes \boldsymbol{I}_L. \tag{20}$$

By modeling the initial state $\boldsymbol{h}(0)$ as a Gaussian distribution with mean vector $\boldsymbol{m}(0)$ and covariance matrix $\boldsymbol{P}(0)$, the MIMO Volterra identification problem can be reformulated as a linear state estimation problem: given the input–output measurements $\{(\boldsymbol{u}(n), \boldsymbol{y}(n)\}_{n=1}^N$, state-space model

TABLE 3 A comparison of the results of Algorithm 1 in the identification of 5^{11} Volterra kernel coefficients for five different signal-to-noise ratios (SNRs). The relative validation error decreases as the SNR increases, and the SNR computed from the simulated signals sees a consistent increase of approximately 11 dB. The total runtime for the algorithm does not depend on the SNR of the training signals and varies between 2 and 6 s.

Identification SNR	11 dB	13 dB	16 dB	19 dB	25 dB
$\frac{\Vert \boldsymbol{y}-\hat{\boldsymbol{y}}\Vert_2}{\Vert \boldsymbol{y}\Vert_2}$	0.255	0.208	0.151	0.105	0.052
Runtime	2.3 s	5.3 s	6.4 s	3.2 s	2.2 s
Simulated signal SNR	22 dB	24 dB	27 dB	30 dB	37 dB

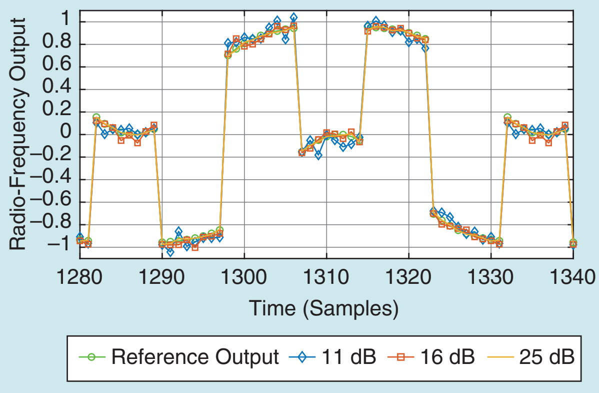


FIGURE 4 The true and simulated output from the identified models using Algorithm 1 under three different signal-to-noise ratios. As expected, higher noise levels on the output during identification results in worse simulations with the identified model.

(19), and measurement matrix (20), find the state sequence $\boldsymbol{h}(n)$ of Volterra kernel coefficients. The linear state-space model [in combination with the Gaussian assumptions on the noise sources and initial guess $\boldsymbol{h}(0)$] results in the Kalman filter providing an unbiased minimum variance state estimate of $\boldsymbol{h}(n)$. Algorithm 2 summarizes the celebrated Kalman filter equations. The first two lines of the algorithm constitute the prediction step, where predictions of the next state mean vector and covariance matrix are made, while lines 7 and 8 update these predictions by incorporating the measurement $\boldsymbol{y}(n)$ through the Kalman gain $\boldsymbol{K}(n)$.

The implementation of Algorithm 2 is not straightforward, as almost all vector and matrix quantities have exponential dimensions. For example, the size of the covariance matrix $\boldsymbol{P}(n)$ is $LI^D \times LI^D$, which can quickly lead to storage problems. Using low-rank tensor decompositions to represent all these vector and matrix quantities is again key in resolving the curse of dimensionality. The state mean $\boldsymbol{m}(n)$ contains the smoothly decaying Volterra kernels, which, once more, motivates the use of a low-rank tensor representation. Two ingredients are required for the implementation of Algorithm 2. The first ingredient is the specific low-rank tensor representations of $\boldsymbol{m}(0)$, $\boldsymbol{P}(0)$, $\boldsymbol{R}_w(n)$, and $\boldsymbol{C}(n)$ so that the filter can be initialized. The second required ingredient is algorithms to compute linear algebra operations in the tensor decomposition form.

Tensor Decompositions to Initialize the Kalman Filter

The tensor representations of $\boldsymbol{m}(0)$, $\boldsymbol{P}(0)$, $\boldsymbol{R}_w(n)$, and $\boldsymbol{C}(n)$ can be constructed such that they are all rank 1. If it is assumed that $\boldsymbol{m}(0) = m\boldsymbol{1}_{LI^D}$, where $m \in \mathbb{R}$, then the corresponding tensor train is unit rank and consists of the tensors

ALGORITHM 2 The Kalman filter state estimation algorithm for system (19).

Input: Measurements $\{(\boldsymbol{u}(n), \boldsymbol{y}(n))\}_{n=1}^N$, initial state mean $\boldsymbol{m}(0)$, initial state covariance $\boldsymbol{P}(0)$.

Output: state mean $\boldsymbol{m}(n)$, state covariance $\boldsymbol{P}(n)(n > 0)$.

```
1: for $n = 1{:}N$ do
2:     $\boldsymbol{m}(n) \leftarrow \boldsymbol{m}(n-1)$                                  % predicted mean
3:     $\boldsymbol{P}(n) \leftarrow \boldsymbol{P}(n-1) + \boldsymbol{R}_w(n)$                    % predicted covariance matrix
4:     $\boldsymbol{v}(n) \leftarrow \boldsymbol{y}(n) - \boldsymbol{C}(n)\boldsymbol{m}(n)$
5:     $\boldsymbol{S}(n) \leftarrow \boldsymbol{C}(n)\boldsymbol{P}(n)\boldsymbol{C}(n)^T + \boldsymbol{R}_e$
6:     $\boldsymbol{K}(n) \leftarrow \boldsymbol{P}(n)\boldsymbol{C}(n)^T\boldsymbol{S}(n)^{-1}$
7:     $\boldsymbol{m}(n) \leftarrow \boldsymbol{m}(n) + \boldsymbol{K}(n)\boldsymbol{v}(n)$               % measurement update of mean
8:     $\boldsymbol{P}(n) \leftarrow \boldsymbol{P}(n) - \boldsymbol{K}(n)\boldsymbol{S}(n)\boldsymbol{K}(n)^T$     % measurement update of covariance matrix
9: end for
```

$$
\begin{aligned}
\mathcal{M}^{(1)} &= m\boldsymbol{1}_I \in \mathbb{R}^{1\times I\times 1},\\
\mathcal{M}^{(2)} &= \boldsymbol{1}_I \in \mathbb{R}^{1\times I\times 1},\\
&\vdots\\
\mathcal{M}^{(D)} &= \boldsymbol{1}_I \in \mathbb{R}^{1\times I\times 1},\\
\mathcal{M}^{(D+1)} &= \boldsymbol{1}_L \in \mathbb{R}^{1\times L\times 1}.
\end{aligned}
$$

The scalar factor m can be freely moved to any of the $(D+1)$ cores. A more general $\boldsymbol{m}(0)$ can be obtained in tensor train form by, for example, explicitly constructing the $\boldsymbol{m}(0)$ vector and computing its corresponding tensor train through the tensor train SVD algorithm [27, p. 2301]. If the L outputs are assumed to be uncorrelated, then the covariance matrix can be modeled as a diagonal matrix $\boldsymbol{P}(0) = p\boldsymbol{I}_{LI^D}$, where $p > 0$ is a real scalar. The corresponding tensor train matrix is unit rank and consists of the tensors

$$
\begin{aligned}
\mathcal{P}^{(1)} &= p\boldsymbol{I}_I \in \mathbb{R}^{1\times I\times I\times 1},\\
\mathcal{P}^{(2)} &= \boldsymbol{I}_I \in \mathbb{R}^{1\times I\times I\times 1},\\
&\vdots\\
\mathcal{P}^{(D)} &= \boldsymbol{I}_I \in \mathbb{R}^{1\times I\times I\times 1},\\
\mathcal{P}^{(D+1)} &= \boldsymbol{I}_L \in \mathbb{R}^{1\times L\times L\times 1}.
\end{aligned}
$$

The scalar p can also be moved to any of the $(D+1)$ core tensors. A diagonal covariance matrix, however, does not model possible correlations between the parameters $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L$ of the L multiple-input, single-output Volterra models. These correlations can be captured in a covariance matrix:

$$
\boldsymbol{P}_L = \begin{pmatrix} p_{11} & \cdots & p_{1L} \\ \vdots & \ddots & \vdots \\ p_{1L} & \cdots & p_{LL} \end{pmatrix},
$$

where p_{ij} denotes the covariance of all kernel coefficients between Volterra models i and j. The corresponding tensor train matrix for $\boldsymbol{P}(0)$ is then obtained by substituting $p\boldsymbol{I}_I$ with $\boldsymbol{P}_L$. By choosing the covariance matrix $\boldsymbol{R}_w(n)$ of the process noise $\boldsymbol{w}(n)$ as

$$
\boldsymbol{R}_w(n) = \left(\frac{1}{\lambda} - 1\right)\boldsymbol{P}(n),
$$

a forgetting factor $\lambda \in [0,1]$ is introduced. This forgetting factor induces an approximate, exponentially decaying weighting on past data. A small λ implies that previous outputs $\boldsymbol{y}(n)$ will be weighted less in the update, while in the extremal case $\lambda = 1$, all past outputs will be taken equally into account. Finally, the particular structure of the measurement matrix $\boldsymbol{C}(n)$, as in (20), is exactly a rank 1 tensor train matrix:

$$
\begin{aligned}
\mathcal{C}^{(1)} &= \boldsymbol{I}_L \in \mathbb{R}^{1\times L\times L\times 1},\\
\mathcal{C}^{(2)} &= \boldsymbol{u}_n \in \mathbb{R}^{1\times 1\times I\times 1},\\
&\vdots\\
\mathcal{C}^{(D)} &= \boldsymbol{u}_n \in \mathbb{R}^{1\times 1\times I\times 1},\\
\mathcal{C}^{(D+1)} &= \boldsymbol{u}_n \in \mathbb{R}^{1\times 1\times I\times 1}.
\end{aligned}
$$

With these tensor representations in place, each line of Algorithm 2 can now be implemented completely in tensor train form. More information about how matrix products can be computed in tensor train form is available in "Linear Algebra Operations With Tensor Trains." Each of these products, however, will result in ever-increasing tensor train ranks. To preserve the low-rank tensor representation, a tensor train rounding algorithm [27] can be used. The rounding algorithm takes a tensor train $\mathcal{A}^{(1)},\dots,\mathcal{A}^{(D)}$ with ranks $R_2,\dots,R_D$ and finds a tensor train $\mathcal{B}^{(1)},\dots,\mathcal{B}^{(D)}$ with ranks $S_2 \le R_2,\dots,S_D \le R_D$ such that $\|\mathcal{A}-\mathcal{B}\|_F / \|\mathcal{A}\|_F \le \epsilon$, where ϵ is a user-defined error tolerance. The rank truncation of D tensor train cores happens through $D-1$ consecutive SVDs of the cores, with a total computational complexity of $O(DIR^3)$, and is described in detail in [27]. It is, however, possible that no rank truncation occurs for the specified tolerance. In this case, the user can limit the ranks to grow beyond a certain specified maximal value.

One problem that is still unexplored is how the positive definiteness of the covariance matrix $\boldsymbol{P}(n)$ can be ensured during rounding. One way to ensure the positive definiteness of $\boldsymbol{P}(n)$ is to keep its tensor train matrix rank 1, although it is not yet understood why this is the case. A square-root, tensor train matrix Kalman filter implementation is still an area of research. Once the Kalman filter has iterated over all samples, a final estimate of the Volterra kernel coefficients is encoded in the Gaussian distribution with mean $\boldsymbol{m}(\boldsymbol{n})$ and covariance matrix $\boldsymbol{P}(n)$. A predicted output $\boldsymbol{y}(n+1)$ is then also described by a Gaussian distribution:

$$\boldsymbol{y}(n+1) \sim \mathcal{N}(\boldsymbol{C}(n+1)\,\boldsymbol{m}(n), \boldsymbol{C}(n+1)\,\boldsymbol{P}(n)\,\boldsymbol{C}(n+1)^T + \boldsymbol{R}_e).$$

Both the mean and covariance of the prediction $\boldsymbol{y}(n+1)$ can be computed through contractions of the corresponding tensor trains.

Double-Balanced Mixer Identification

The same double-balanced mixer setup described in the section covering the ALS Volterra identification algorithm is used for this experiment. In this experiment, all time series were sampled at 5 kHz, for a total of 6000 samples. The output signal was corrupted with Gaussian noise such that three different outputs with respective SNRs of 12, 17, and 26 dB were obtained. A Kalman filter is then used to estimate a two-input, one-output Volterra system with $d=7$ and $M=10$ by filtering the first 5900 samples for the three noisy outputs separately. The state vector containing the Volterra kernel coefficients consists of $21^7 \approx 1.801 \times 10^9$ entries, which is well beyond the reach of a standard Kalman filter. The initial mean vector $\boldsymbol{m}(0)$ is initialized as the zero vector, and the initial covariance matrix $\boldsymbol{P}(0)$ is set to $1000\boldsymbol{I}$. The rounding parameter ϵ was set to a fixed value of 10^{-1}.

The corresponding maximal tensor train ranks of $\boldsymbol{m}(n)$ when filtering the output with SNRs of 12, 17, and 26 dB were 11, 13, and 14, respectively. All tensor train matrix ranks of $\boldsymbol{P}(n)$ were equal to one. The median runtime for one filter step was 0.0068 s, and the total runtime to filter 5900 samples was approximately 40 s. The obtained mean vectors $\boldsymbol{m}(5900)$ were then used to simulate the remaining 100 samples. The simulated outputs are shown together with the reference output, which is not corrupted by noise, in Figure 5, which demonstrates that a higher SNR results in better performance of the Kalman filter. For the outputs with SNRs of 12, 17, and 26 dB, the root-mean-square errors of the simulated outputs were 0.1778, 0.097, and 0.034, respectively.

APPLICATION 3: SUBSPACE IDENTIFICATION OF POLYNOMIAL INPUT STATE-SPACE MODELS

The output of a Volterra model is completely determined by a polynomial of past input values. A large value for the memory M may be required to capture the dynamics in a satisfactory manner. One way to retain more information from the past in the model is through the addition of a state variable $\boldsymbol{x}(n) \in \mathbb{R}^K$. The following state-space model with linear state dynamics and polynomial input dependencies is then obtained:

$$\begin{aligned}\boldsymbol{x}(n+1) &= \boldsymbol{A}\,\boldsymbol{x}(n) + \boldsymbol{B}\,\boldsymbol{u}_n^D,\\ \boldsymbol{y}(n) &= \boldsymbol{C}\,\boldsymbol{x}(n) + \boldsymbol{D}\,\boldsymbol{u}_n^D.\end{aligned} \tag{21}$$

The matrices $\boldsymbol{A} \in \mathbb{R}^{K\times K}$ and $\boldsymbol{C} \in \mathbb{R}^{L\times K}$ are assumed to be sufficiently small that they require no tensor representation. The definition of the vector $\boldsymbol{u}_n$ is slightly different from the Volterra model in that no past input values are present anymore. That is,

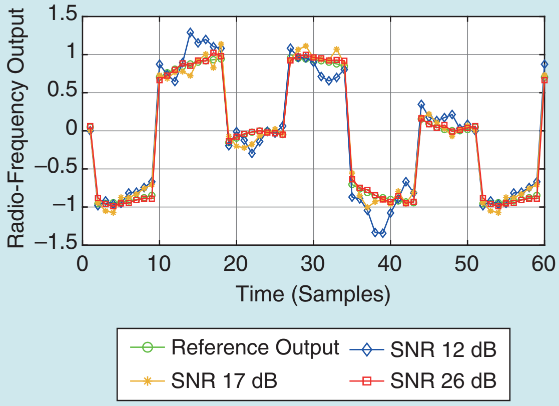


FIGURE 5 The reference and simulated output from the tensor-based Kalman filter for three different signal-to-noise ratios (SNRs). The Volterra models identified through Algorithm 2 with better SNR circumstances simulate outputs during validation that are closer to the reference output.

$$\boldsymbol{u}_n = (1 \;\; \boldsymbol{u}(n)^T)^T \in \mathbb{R}^{(1+P)},$$

where now the convention $I = 1 + P$ will be used. The matrices $\boldsymbol{B} \in \mathbb{R}^{K \times I^D}$ and $\boldsymbol{D} \in \mathbb{R}^{L \times I^D}$ have an exponential number of columns and model the polynomial input contributions to both the state dynamics and the output. The maximal total degree D of both polynomial input contributions $\boldsymbol{B}$ and $\boldsymbol{D}$ are assumed to be equal. The output $\boldsymbol{y}(n)$ of the system (21) is

$$\boldsymbol{y}(n) = \boldsymbol{C}\boldsymbol{A}^n \boldsymbol{x}(0) + \sum_{i=0}^{n-1} \boldsymbol{C}\boldsymbol{A}^{n-1-i} \boldsymbol{B}\boldsymbol{u}_i^D + \boldsymbol{D}\,\boldsymbol{u}_n^D$$

and is therefore completely determined by the initial state $\boldsymbol{x}(0)$ and all input signals $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_n$.

The system identification problem resolved in this section is: given measurements $\{(\boldsymbol{u}(n), \boldsymbol{y}(n)\}_{n=1}^{N}$ and the state-space model (21), estimate the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ up to a similarity transform. Two mainstream identification techniques for linear time-invariant systems still dominate the field today: prediction error methods and subspace methods. Prediction error methods rely on nonlinear optimization routines to find the values of all unknown matrices such that the prediction error is minimized. Subspace methods exclusively use linear algebra operations, such as projections and matrix decompositions, to find the unknown matrices. This section will focus on the use of subspace methods, as it is possible to use a conventional subspace method for the identification of (21), combined with low-rank tensor representations for the $\boldsymbol{B}$ and $\boldsymbol{D}$ matrices [45].

The key equation of subspace methods is the data equation. For the polynomial input state-space model (21), the data equation is

$$\boldsymbol{Y}_{1|S} = \boldsymbol{O}_S\,\boldsymbol{X} + \boldsymbol{P}_S\,\boldsymbol{U}_{1|S}, \qquad (22)$$

where the matrices $\boldsymbol{Y}_{1|S}$ and $\boldsymbol{U}_{1|S}$ are block Hankel data matrices:

$$\boldsymbol{Y}_{1|S} := \begin{pmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \cdots & \boldsymbol{y}_T \\ \boldsymbol{y}_2 & \boldsymbol{y}_3 & \cdots & \boldsymbol{y}_T \\ \vdots & \vdots & & \vdots \\ \boldsymbol{y}_S & \boldsymbol{y}_{S+1} & \ldots & \boldsymbol{y}_{T+S-2} \end{pmatrix} \in \mathbb{R}^{LS \times T},$$

$$\boldsymbol{U}_{1|S} := \begin{pmatrix} \boldsymbol{u}_1^D & \boldsymbol{u}_2^D & \cdots & \boldsymbol{u}_T^D \\ \boldsymbol{u}_2^D & \boldsymbol{u}_3^D & \cdots & \boldsymbol{u}_{T+1}^D \\ \vdots & \vdots & & \vdots \\ \boldsymbol{u}_S^D & \boldsymbol{u}_{S+1}^D & \cdots & \boldsymbol{u}_{T+S-2}^D \end{pmatrix} \in \mathbb{R}^{I^D S \times T}.$$

The scalar S is the window size and must be chosen such that $S > K$. The total number of measurements is $N = T + S - 2$. The matrix

$$\boldsymbol{O}_S := \begin{pmatrix} \boldsymbol{C} \\ \boldsymbol{C}\boldsymbol{A} \\ \boldsymbol{C}\boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{C}\boldsymbol{A}^{S-1} \end{pmatrix} \in \mathbb{R}^{LS \times K} \qquad (23)$$

is the well-known extended observability matrix of linear time-invariant systems, and

$$\boldsymbol{P}_S := \begin{pmatrix} \boldsymbol{D} & & \\ \boldsymbol{C}\boldsymbol{B} & \boldsymbol{D} & \\ \vdots & \ddots & \ddots \\ \boldsymbol{C}\boldsymbol{A}^{S-2}\boldsymbol{B} & \cdots & \boldsymbol{C}\boldsymbol{B} \;\; \boldsymbol{D} \end{pmatrix} \in \mathbb{R}^{LS \times I^D S} \qquad (24)$$

is a block Toeplitz matrix with an exponential number of columns. The matrix

$$\boldsymbol{X} := (\boldsymbol{x}_1 \;\; \boldsymbol{x}_2 \;\; \cdots \;\; \boldsymbol{x}_T) \in \mathbb{R}^{K \times T}$$

is the state sequence matrix.

Starting from the data equation, two commonly used subspace identification algorithms, Multivariable Output Error State Space (MOESP) and Numerical Algorithms for Subspace State-Space System Identification (N4SID), can be derived. Both algorithms start from the data matrices $\boldsymbol{U}_{1|S}$ and $\boldsymbol{Y}_{1|S}$ and are not straightforward to implement due to the exponential size of $\boldsymbol{U}_{1|S}$. The main difference between MOESP and N4SID is that MOESP uses realization theory on the observability term of the data equation to retrieve the $\boldsymbol{A}$ and $\boldsymbol{C}$ matrices, after which the $\boldsymbol{B}$ and $\boldsymbol{D}$ matrices are found separately by solving a linear system. In the N4SID approach, the state sequence $\boldsymbol{X}$ is first estimated, after which all system matrices are identified from one linear system [46, p. 165]:

$$\begin{pmatrix} \bar{\boldsymbol{X}}_{S+1} \\ \bar{\boldsymbol{Y}}_{S|S} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \bar{\boldsymbol{X}}_S \\ \bar{\boldsymbol{U}}_{S|S} \end{pmatrix}.$$

It is this last step of N4SID that is less straightforward to implement with tensor decompositions. The MOESP algorithm, summarized in Algorithm 3, has a more straightforward tensor implementation. Linear algebra operations also play a key role in this algorithm, and the use of tensor trains is again beneficial. In "Linear Algebra Operations With Tensor Trains," all required linear algebra operations using tensor trains are reviewed.

The implementation of the MOESP identification algorithm with tensor trains requires three main steps: the construction of the tensor train matrix of $\boldsymbol{U}_{1|S}$, the computation of the linear quadratic (LQ) factor matrices, and the construction of the tensor train matrix for the right-hand side of (25). What follows is a short exposition on the implementation of these steps. More details can be found in [45].

Tensor Train Matrix of a Block Hankel Data Matrix

Explicit construction of the block Hankel data matrix $\boldsymbol{U}_{1|S}$ quickly becomes infeasible due to the I^{DS} exponential number of rows. Fortunately, it is possible to construct the corresponding tensor train matrix without ever having to explicitly store the matrix $\boldsymbol{U}_{1|S}$ in memory. The block

Hankel structure in the data matrix also ensures that its tensor train matrix ranks are bounded [45]. Algorithm 4 summarizes the procedure. The main insight upon which Algorithm 4 is based is that the matrix

$$\boldsymbol{U} = \text{reshape}(\boldsymbol{U}_{1|S}, [I^D, ST]) = (\boldsymbol{u}_1^D \;\; \boldsymbol{u}_2^D \;\; \cdots \;\; \boldsymbol{u}_S^D \;\; \boldsymbol{u}_2^D \;\; \cdots \;\; \boldsymbol{u}_{T+S-2}^D)$$

can be written as a D-times-repeated Khatri–Rao product

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

of the matrix

$$\tilde{\boldsymbol{U}} = (\boldsymbol{u}_1 \;\; \boldsymbol{u}_2 \;\; \cdots \;\; \boldsymbol{u}_S \;\; \boldsymbol{u}_2 \;\; \cdots \;\; \boldsymbol{u}_{T+S-2}) \in \mathbb{R}^{I \times ST}.$$

Algorithm 4 builds up the tensor train matrix of $\boldsymbol{U}_{1|S}$ core by core, whereby the SVD operation on line 6 is used to determine the rank between two consecutive cores in line 7.

Computation of the LQ Factor Matrices

A major advantage of both the conventional N4SID and MOESP methods is that the orthogonal factors in the LQ decomposition never must be computed. The tensor train implementation of Algorithm 3, however, requires the explicit computation of the orthogonal factors $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$. In fact, the LQ decomposition in line 1 of Algorithm 3 cannot be computed in tensor train matrix form. Instead, using a modification of Algorithm S1, explained in "Linear Algebra Operations With Tensor Trains," an economical SVD of $\boldsymbol{U}_{1|S}$,

$$\boldsymbol{U}_{1|S} = \boldsymbol{W}\,\boldsymbol{T}\,\boldsymbol{Q}^T = (\boldsymbol{W}_1 \;\; \boldsymbol{W}_2)\begin{pmatrix} \boldsymbol{T}_1 & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} \boldsymbol{Q}_1^T \\ \boldsymbol{Q}_2^T \end{pmatrix}, \tag{26}$$

is computed, with $\boldsymbol{W} \in \mathbb{R}^{I^D S \times T}$ an orthogonal matrix in tensor train matrix form, $\boldsymbol{T} \in \mathbb{R}^{T \times T}$ a diagonal matrix, and $\boldsymbol{Q} \in \mathbb{R}^{T \times T}$ an orthogonal matrix. The rank of $\boldsymbol{U}_{1|S}$ is denoted R such that $\boldsymbol{T}_1 \in \mathbb{R}^{R \times R}$.

The required matrix factors $\boldsymbol{L}_{11}$, $\boldsymbol{L}_{21}$, and $\boldsymbol{L}_{22}$ of the conventional MOESP algorithm can now be computed as

$$\begin{aligned} \boldsymbol{L}_{11} &= \boldsymbol{W}_1 \boldsymbol{T}_1 \in \mathbb{R}^{I^D S \times R}, \\ \boldsymbol{L}_{21} &= \boldsymbol{Y}_{1|S} \boldsymbol{Q}_1 \in \mathbb{R}^{LS \times R}, \\ \boldsymbol{L}_{22} &= \boldsymbol{Y}_{1|S} \boldsymbol{Q}_2 \in \mathbb{R}^{LS \times LS}. \end{aligned} \tag{27}$$

The tensor train matrix of $\boldsymbol{L}_{11}$ is easily found, as the first $(D-1)$ tensors are identical to the tensors of $\boldsymbol{W}$, while the Dth tensor of $\boldsymbol{L}_{11}$ is $\mathcal{W}^{(D)} \times_3 (\boldsymbol{T}_1 \;\; \boldsymbol{0})^T$. Once the matrix factors $\boldsymbol{L}_{11}$, $\boldsymbol{L}_{21}$, and $\boldsymbol{L}_{22}$ are computed, the conventional MOESP algorithm can be used to find $\boldsymbol{A}$ and $\boldsymbol{C}$.

Solving a Least-Squares Problem With Tensor Trains

Line 7 of Algorithm 3 requires the computation and partitioning of the $(SL-K) \times SI^D$ matrix $\boldsymbol{U}_2^T \boldsymbol{L}_{21} \boldsymbol{L}_{11}^{-1}$. This requires the computation of the left inverse of $\boldsymbol{L}_{11}$ in tensor network form. Fortunately, from (27) it follows that $\boldsymbol{L}_{11}^{-1} = \boldsymbol{T}_1^{-1} \boldsymbol{W}_1^T$, as $\boldsymbol{W}_1^T \boldsymbol{W}_1 = \boldsymbol{I}_r$. The transpose of $\boldsymbol{W}_1$ as a tensor train matrix is done by a permutation of the second index with the third index of each core tensor. The tensor train matrix of $\boldsymbol{L}_{11}^{-1}$ is therefore obtained by permuting each of the tensors $\mathcal{W}^{(i)}$ into $\tilde{\mathcal{W}}^{(i)}$ and computing $\mathcal{W}^{(d)} \times_2 (\boldsymbol{T}_1^{-1} \;\; \boldsymbol{0})$, where the inverse of $\boldsymbol{T}_1$ is obtained by inverting its diagonal. Once the tensor train matrix of $\boldsymbol{L}_{11}^{-1}$ is obtained, multiplication with $\boldsymbol{U}_2^T \boldsymbol{L}_{21}$ is also performed on the Dth core tensor. In fact, the multiplication with $\boldsymbol{T}_1^{-1}$ can be combined with

ALGORITHM 3 The Multivariable Output Error State-Space subspace identification algorithm.

Input: L samples $(\boldsymbol{u}_0, \boldsymbol{y}_0), \ldots, (\boldsymbol{u}_{L-1}, \boldsymbol{y}_{L-1})$, k.
Output: matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{H}$.

```
1:  $\begin{pmatrix} U_{1|S} \\ Y_{1|S} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}\begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}$   % linear quadratic decomposition of data matrices
2:  SVD of $L_{22} = (U_1 \;\; U_2)\begin{pmatrix} S_1 & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix}$
3:  Define system order as $K := \text{rank}(L_{22})$
4:  $O_S = U_1 S_1^{1/2}$ and $C = O_S(1{:}L, :)$
5:  Compute $A$ from $O_k(1{:}SL-L, :)A = O(L+1{:}SL, :)$
6:  Partition $U_2^T := (L_1 \;\cdots\; L_S)$ into $S$ blocks of size $(SL-K) \times L$
7:  Partition $U_2^T L_{21} L_{11}^{-1} := (M_1 \;\cdots\; M_S)$ into $S$ blocks of size $(SP-K) \times I^D$
8:  Define $\bar{L}_i := (L_i \;\cdots\; L_S), i = 2, \ldots S$
9:  Compute $B, D$ from
        $\begin{pmatrix} L_1 & \bar{L}_2 O_{S-1} \\ L_2 & \bar{L}_3 O_{S-2} \\ \vdots & \vdots \\ L_{S-1} & \bar{L}_S O_1 \\ L_S & 0 \end{pmatrix}\begin{pmatrix} D \\ B \end{pmatrix} = \begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_{S-1} \\ M_S \end{pmatrix}$   (25)
```

ALGORITHM 4 Convert the repeated Khatri–Rao product matrix into a tensor train matrix.

Input: $I \times ST$ matrix $\tilde{\boldsymbol{U}}$, factor D.
Output: tensor train matrix $\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(D)}$ of $\boldsymbol{U}_{1|S}$.

```
1:  $\mathcal{U}^{(1)} \leftarrow \text{reshape}(\tilde{U}, [1, I, ST, 1])$
2:  for $d = 1, \ldots, D-1$ do
3:      $T \leftarrow \text{reshape}(\mathcal{U}^{(d)}, [R_d I, ST])$        % $R_1 = 1$
4:      $T \leftarrow T \odot \tilde{U}$
5:      $T \leftarrow \text{reshape}(T, [R_d I, IST])$
6:      $[U, S, V] \leftarrow \text{SVD}(T)$
7:      $R_{d+1} \leftarrow$ numerical rank of $T$ determined from SVD
8:      $\mathcal{U}^{(d)} \leftarrow \text{reshape}(U, [R_d, I, 1, R_{d+1}])$
9:      $\mathcal{U}^{(d+1)} \leftarrow \text{reshape}(SV^T, [R_{d+1}, I, ST, 1])$
10: end for
11: $\mathcal{U}^{(D)} \leftarrow \text{reshape}(\mathcal{U}^{(D)}, [R_D, IS, T, 1])$
```

Many resources are available to facilitate the computation of all kinds of tensor decompositions.

$\boldsymbol{U}_2^T\boldsymbol{L}_{21}$. The partitioning of $\boldsymbol{U}_2^T\boldsymbol{L}_{21}\boldsymbol{L}_{11}^{-1}$ into S blocks of size $(SL-K)\times I^d$ follows from

$$\begin{aligned}&\text{reshape}(\boldsymbol{\mathcal{M}}^{(D)}, [R_D, (SL-K), I, S, 1]),\\ &\text{permute}(\boldsymbol{\mathcal{M}}^{(D)}, [1, 2, 4, 3, 5]),\\ &\text{reshape}(\boldsymbol{\mathcal{M}}^{(D)}, [R_D, (SL-K)S, I, 1]).\end{aligned}$$

To estimate the matrices $\boldsymbol{B}$ and $\boldsymbol{D}$, the linear system (25) must be solved. The matrix on the left-hand side of (25) can be stored explicitly without the use of any tensor decomposition. If its pseudoinverse is denoted by $\boldsymbol{L}^{-1}$, then the concatenation of $\boldsymbol{D}$ with $\boldsymbol{B}$ is

$$\begin{pmatrix}\boldsymbol{D}\\ \boldsymbol{B}\end{pmatrix} = \boldsymbol{L}^{-1}\begin{pmatrix}\boldsymbol{M}_1\\ \boldsymbol{M}_2\\ \vdots\\ \boldsymbol{M}_{k-1}\\ \boldsymbol{M}_S\end{pmatrix}, \tag{28}$$

which is equivalent to $\boldsymbol{\mathcal{M}}^{(D)}\times_2\boldsymbol{L}^{-1}$. The resulting tensor train matrix represents the concatenation of the $\boldsymbol{B}$ and $\boldsymbol{D}$ matrices into a single matrix. The whole subspace identification algorithm starts from the tensor train matrix of $\boldsymbol{U}_{1|S}$ and consists only of manipulations of the Dth tensor core to compute the tensor train matrix of the $\boldsymbol{B}$ and $\boldsymbol{D}$ matrices. Since the manipulations are limited to the Dth tensor core, all ranks R_1 up to R_{D-1} of the resulting tensor train matrix will remain unchanged. The low-rank property due to the block-Hankel structure of $\boldsymbol{U}_{1|S}$ therefore ensures the low-rank tensor train matrix of the solution. Indeed, using the upper bounds on tensor train matrix ranks described in [47, p. 73], it can be shown that the rank R_D of the Dth core of the resulting tensor train matrix will be smaller than the corresponding rank of $\boldsymbol{U}_{1|S}$.

TABLE 4 A comparison of runtimes and relative validation errors for four models: a linear multivariable output error state space (MOESP), a state-space model with polynomial inputs (TNMOESP), a generic polynomial state-space model (PNLSS), and a Volterra model.

Model	Runtime (s)	Relative Validation Error
Linear (MOESP)	0.26	0.418
Tensor network MOESP	0.69	0.148
PNLSS	14,264	0.087
Volterra	1.61	0.004

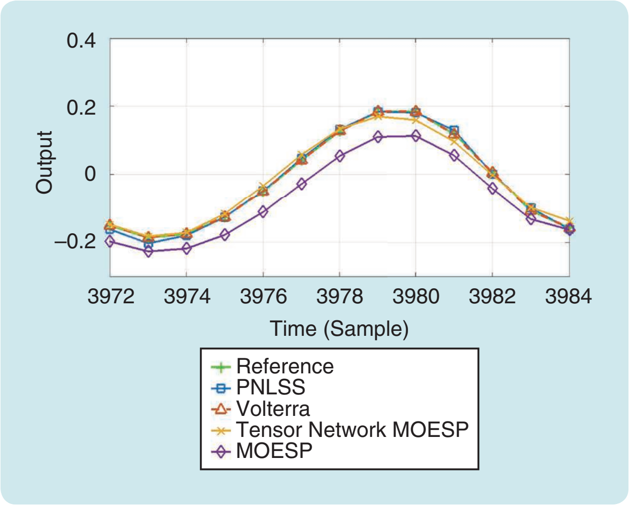


FIGURE 6 The reference and simulated amplifier output from four different models: a linear state-space model [Multivariable Output Error State Space (MOESP)], a state-space model with polynomial inputs (tensor network MOESP), a generic polynomial nonlinear state-space model (PNLSS), and a Volterra model. The linear model performs the worst on the validation data, while all three nonlinear models have acceptable performance.

High-End Valve Control Amplifier

This experiment compares the performance of Algorithm 3 with other models and methods on real-world data. The data set is from the same experiment described in [48, p. 3936], and the system under consideration is a high-end valve control amplifier (which is normally used as a preamplifier for audio signals). The amplifier is a SISO system and was fed a flat spectrum, random-phase multisine with a period of 4096 samples at 1.25 MHz. We compare four different models and methods. For each of these models/methods, only the one with the best relative validation error is reported. First, a linear state-space system was identified by Algorithm 3, with system order $K=3$, using the first 1000 samples. Then, a state-space model with polynomial inputs was identified using Algorithm 3, with $D=6$ and $K=30$, also using the first 1000 samples. In addition, a Volterra model of degree $D=2$ and memory $M=30$ was identified using Algorithm 1, also employing the first 1000 samples.

Finally, a general polynomial state space, as described in [49], was identified using the iterative methods of the polynomial nonlinear state-space (PNLSS) Matlab toolbox, with $K=15$ and where both polynomials of the state and output equations are of degree 4. To obtain good validation errors, the general polynomial state-space model needed to be estimated on two periods of the input signal. Each of the models was then used to simulate the output from the input that was not used for identification. The runtimes and relative validation errors $\|y - \hat{y}\|/\|y\|$, where y denotes the measured output and $\hat{y}$ denotes the simulated output, for each of the methods and models are listed in Table 4.

The linear state-space model can be identified very quickly but performs the worst, while using polynomial inputs improves the validation (at the cost of a slightly longer runtime). The general polynomial state-space system can improve the validation error by one order of magnitude, at the cost of a very significant runtime. The convergence of the iterative method in the PNLSS toolbox was rather slow, as it took 12,317 s for the relative validation error to drop to 0.29. The computational complexity of the PNLSS method per iteration suffers from the large number of polynomial coefficients. Surprisingly, the Volterra model is able to achieve a relative validation error that is another order of magnitude smaller than the general polynomial state-space system, which might suggest that the real-world system is better described by a Volterra model than a polynomial state-space model. Figure 6 provides 12 samples of the reference output and simulated outputs for the four different models. Due to the scale of the figure, it is not possible to distinguish the output from the Volterra model from the reference. As evident from the figure, all nonlinear models produce outputs that are closer to the real output compared to the linear model.

TABLE 5 Various tensor toolboxes and implementations for tensor-based nonlinear system identification. TT: Tensor Train; TN: Tensor Network; MVMALS: Multiple-Input, Multiple-Output Volterra Modified Alternating Linear Scheme; MOESP: Multivariable Output Error State Space; MIT: Massachusetts Institute of Technology; BSD: Berkeley Source Distribution; GPL: General Public License; LGPL: Lesser GPL; CPD: canonical polyadic decomposition.

Package	Implementation	Open Source	Description
TT Toolbox [50]	Matlab and Python	MIT license	Tensor train and tensor train matrix
Tensorlab [51]	Matlab	Closed source	CPD and Tucker decomposition
Tensor Toolbox [52]	Matlab	BSD license	CPD and Tucker decomposition
TensorBox [53]	Matlab	GPL 3.0	CPD
MVMALS [54]	Matlab	LGPL 3.0	Volterra identification with tensor trains
TNKalman [55]	Matlab	LGPL 3.0	Tensor train Kalman filter
TNMOESP [56]	Matlab	LGPL 3.0	Tensor train subspace identification

AVAILABLE RESOURCES FOR TENSOR DECOMPOSITIONS

Many resources are available to facilitate the computation of all kinds of tensor decompositions. Furthermore, the Matlab implementations of all identification methods described in this article are available as open source software. An overview is given in Table 5.

CONCLUSION

This article provided a brief introduction to tensors for systems and control. Important tensor operations and decompositions were discussed and illustrated with three applications in nonlinear system identification. Tensor-based identification algorithms were discusssed for Volterra systems and state-space models with polynomial inputs. The curse of dimensionality in each of these identification problems was lifted through the use of low-rank tensor decompositions. The low-rank property is the essential ingredient in lifting the curse of dimensionality and can be interpreted as adding the constraint that model parameters are not completely independent. By no means is the discussion on tensors in this article exhaustive. The intention was to highlight some key features of tensors and help interested practitioners more efficiently explore relevant literature.

AUTHOR INFORMATION

Kim Batselier (k.batselier@tudelft.nl) received the M.S. degree in electromechanical engineering and Ph.D. degree in applied sciences from KU Leuven in 2005 and 2013, respectively. He worked as a research engineer at BioRICS on automated performance monitoring until 2009. He is currently an assistant professor at Delft University of Technology, Delft, 2628 CD, The Netherlands. His current research interests include linear and nonlinear system theory/identification, machine learning, tensors, and numerical algorithms.

REFERENCES

[1] S. A. Billings, *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*. Hoboken, NJ: Wiley, 2013.
[2] S. A. Billings, “Identification of nonlinear systems—A survey,” *IEE Proc. D (Control Theory Appl.)*, vol. 127, no. 6, pp. 272–285, 1980.
[3] S. A. Billings and S. Fakhouri, “Identification of systems containing linear dynamic and static nonlinear elements,” *Automatica*, vol. 18, no. 1, pp. 15–26, 1982, doi: 10.1016/0005-1098(82)90022-X.
[4] J. Schoukens, J. G. Nemeth, P. Crama, Y. Rolain, and R. Pintelon, “Fast approximate identification of nonlinear systems,” *Automatica*, vol. 39, no. 7, pp. 1267–1274, 2003. doi: 10.1016/S0005-1098(03)00083-9.
[5] G. Birpoutsoukis, A. Marconato, J. Lataire, and J. Schoukens, “Regularized nonparametric Volterra kernel estimation,” *Automatica*, vol. 82, pp. 324–327, Aug. 2017, doi: 10.1016/j.automatica.2017.04.014.
[6] G. Birpoutsoukis, P. Z. Csurcsia, and J. Schoukens, “Efficient multidimensional regularization for Volterra series estimation,” *Mech. Syst. Signal Process.*, vol. 104, pp. 896–914, 2018, doi: 10.1016/j.ymssp.2017.10.007.
[7] T. Shi and A. Townsend, “On the compressibility of tensors,” *SIAM J. Matrix Anal. Appl.*, vol. 42, no. 1, pp. 275–298, 2021, doi: 0.1137/20M1316639.
[8] I. V. Oseledets, “Constructive representation of functions in low-rank tensor formats,” *Constructive Approx.*, vol. 37, no. 1, pp. 1–18, 2013, doi: 10.1007/s00365-012-9175-x.

[9] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Rev*, vol. 51, no. 3, pp. 455–500, 2009, doi: 10.1137/07070111X.
[10] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Found. Trends Mach. Learning*, vol. 9, no. 4-5, pp. 249–429, 2016, doi: 10.1561/2200000059.
[11] A. Cichocki et al., "Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives," *Found. Trends Mach. Learning*, vol. 9, no. 6, pp. 431–673, 2017, doi: 10.1561/2200000067.
[12] G. H. Golub and C. F. van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: JHU Press, 2013.
[13] V. De Silva and L.-H. Lim, "Tensor rank and the ill-posedness of the best low-rank approximation problem," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1084–1127, 2008, doi: 10.1137/06066518X.
[14] A. Cichocki et al., "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, 2015, doi: 10.1109/MSP.2013.2297439.
[15] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an 'explanatory' multi-modal factor analysis," *UCLA Work. Papers Phonetics*, vol. 16, no. 1, p. 84, 1970.
[16] J. 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, doi: 10.1007/BF02310791.
[17] N. D. Sidiropoulos and R. Bro, "On the uniqueness of multilinear decomposition of n-way arrays," *J. Chemometr.*, vol. 14, no. 3, pp. 229–239, 2000, doi: 10.1002/1099-128X(200005/06)14:3<229::AID-CEM587>3.0.CO;2-N.
[18] A. Stegeman and N. D. Sidiropoulos, "On Kruskal's uniqueness condition for the CANDECOMP/PARAFAC decomposition," *Linear Algebra Appl.*, vol. 420, nos. 2–3, pp. 540–552, 2007, doi: 10.1016/j.laa.2006.08.010.
[19] A. Smilde, R. Bro, and P. Geladi, *Multi-way Analysis: Applications in the Chemical Sciences.* Hoboken, NJ, USA: Wiley, 2005.
[20] E. Acar and B. Yener, "Unsupervised multiway data analysis: A literature survey," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 1, pp. 6–20, 2008, doi: 10.1109/TKDE.2008.112.
[21] P. Dreesen, M. Ishteva, and J. Schoukens, "Decoupling multivariate polynomials using first-order information and tensor decompositions," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 2, pp. 864–879, 2015, doi: 10.1137/140991546.
[22] J. Decuyper, P. Dreesen, J. Schoukens, M. C. Runacres, and K. Tiels, "Decoupling multivariate polynomials for nonlinear state-space models," *IEEE Control Syst. Lett.*, vol. 3, no. 3, pp. 745–750, 2019, doi: 10.1109/LCSYS.2019.2916955.
[23] C. J. Hillar and L.-H. Lim, "Most tensor problems are NP-hard," *J. ACM (JACM)*, vol. 60, no. 6, pp. 1–39, 2013, doi: 10.1145/2512329.
[24] L. R. Tucker, "Implications of factor analysis of three-way matrices for measurement of change," in *Problems Measuring Change*, C. W. Harris, Ed. Madison, WI, USA: Univ. Wis. Press, 1963, pp. 122–137.
[25] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.
[26] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal. A*, vol. 21, no. 4, pp. 1253–1278, 2000, doi: 10.1137/S0895479896305696.
[27] I. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011, doi: 10.1137/090752286.
[28] M. Espig, W. Hackbusch, S. Handschuh, and R. Schneider, "Optimization problems in contracted tensor networks," *Comput. Visualization Sci.*, vol. 14, no. 6, pp. 271–285, Aug. 2011, doi: 10.1007/s00791-012-0183-y.
[29] M. Espig, K. K. Naraparaju, and J. Schneider, "A note on tensor chain approximation," *Comput. Visualization Sci.*, vol. 15, no. 6, pp. 331–344, Dec. 2012, doi: 10.1007/s00791-014-0218-7.
[30] B. N. Khoromskij, "O(dlog N)-quantics approximation of N-d tensors in high-dimensional numerical modeling," *Construct. Approx.*, vol. 34, no. 2, pp. 257–280, Oct. 2011, doi: 10.1007/s00365-011-9131-1.
[31] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states," *Ann. Phys. (USA)*, vol. 326, no. 1, pp. 96–192, 2011, doi: 10.1016/j.aop.2010.09.012.
[32] I. Oseledets, "Approximation of $2^d \times 2^d$ matrices using tensor decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 4, pp. 2130–2145, June 2010.
[33] R. J. G. B. Campello, G. Favier, and W. C. d. Amaral, "Optimal expansions of discrete-time Volterra models using Laguerre functions," *Automatica*, vol. 40, no. 5, pp. 815–822, 2004, doi: 10.1016/j.automatica.2003.11.016.
[34] C. Diouf, M. Telescu, P. Cloastre, and N. Tanguy, "On the use of equality constraints in the identification of Volterra-Laguerre models," *IEEE Signal Process. Lett.*, vol. 19, no. 12, pp. 857–860, Dec. 2012, doi: 10.1109/LSP.2012.2223463.
[35] 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, doi: 10.1002/acs.1272.
[36] K. Batselier, Z. M. Chen, and N. Wong, "Tensor network alternating linear scheme for MIMO Volterra system identification," *Automatica*, vol. 84, pp. 26–35, Oct. 2017, doi: 10.1016/j.automatica.2017.06.033.
[37] S. Holtz, T. Rohwedder, and R. Schneider, "The alternating linear scheme for tensor optimization in the tensor train format," *SIAM J. Sci. Comput.*, vol. 34, no. 2, pp. A683–A713, 2012, doi: 10.1137/100818893.
[38] T. Rohwedder and A. Uschmajew, "On local convergence of alternating schemes for optimization of convex problems in the tensor train format," *SIAM J. Numerical Anal.*, vol. 51, no. 2, pp. 1134–1162, 2013, doi: 10.1137/110857520.
[39] R. Karagoz and K. Batselier, "Nonlinear system identification with regularized tensor network B-splines," *Automatica*, vol. 122, p. 109300, 2020, doi: 10.1016/j.automatica.2020.109300.
[40] E. Rubiola, "Tutorial on the double balanced mixer," *arXiv e-prints*, 2006.
[41] S. Särkkä, "Bayesian filtering and smoothing," in *Institute of Mathematical Statistics Textbooks*. Cambridge, U.K.: Cambridge Univ. Press, 2013.
[42] K. Batselier, C.-Y. Ko, and N. Wong, "Extended Kalman filtering with low-rank tensor networks for mimo Volterra system identification," in *Proc. IEEE 58th Conf. Decision Control (CDC)*, 2019, pp. 7148–7153, doi: 10.1109/CDC40024.2019.9028895.
[43] K. Batselier, Z. Chen, and N. Wong, "A tensor network Kalman filter with an application in recursive MIMO Volterra system identification," *Automatica*, vol. 84, pp. 17–25, Oct. 2017, doi: 10.1016/j.automatica.2017.06.019.
[44] K. Batselier and N. Wong, "Matrix output extension of the tensor network Kalman filter with an application in MIMO Volterra system identification," *Automatica*, vol. 95, pp. 413–418, 2018, doi: 10.1016/j.automatica.2018.06.015.
[45] K. Batselier, C.-Y. Ko, and N. Wong, "Tensor network subspace identification of polynomial state space models," *Automatica*, vol. 95, pp. 187–196, 2018, doi: 10.1016/j.automatica.2018.05.015.
[46] T. Katayama, "Subspace methods for system identification," *Communications and Control Engineering.* London: Springer-Verlag, 2005.
[47] I. Oseledets and E. Tyrtyshnikov, "TT-cross approximation for multidimensional arrays," *Linear Algebra Appl.*, vol. 422, no. 1, pp. 70–88, 2010.
[48] M. Schoukens, R. Pintelon, and Y. Rolain, "Parametric identification of parallel Hammerstein systems," *IEEE Trans. Instrum. Meas.*, vol. 60, no. 12, pp. 3931–3938, 2011, doi: 10.1109/TIM.2011.2138370.
[49] J. Paduart, L. Lauwers, J. Swevers, K. Smolders, J. Schoukens, and R. Pintelon, "Identification of nonlinear systems using polynomial nonlinear state space models," *Automatica*, vol. 46, no. 4, pp. 647–656, 2010, doi: 10.1016/j.automatica.2010.01.001.
[50] I. Oseledets, S. Dolgov, V. Kazeev, O. Lebedeva, and T. Mach, MATLAB TT-toolbox version 2.3 available online, June 2014. [Online]. Available: https://github.com/oseledets/TT-Toolbox
[51] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, Tensorlab 3.0, Mar. 2016. [Online]. Available: https://www.tensorlab.net
[52] B. W. Bader et al., Matlab Tensor Toolbox Version 3.1, June 2019. [Online]. Available: https://www.tensortoolbox.org
[53] A. Phan, P. Tichavsky, and A. Cichocki, "Tensorbox," GitHub, San Francisco, Mar. 2019. [Online]. Available: https://github.com/phananhhuy/TensorBox
[54] K. Batselier, "MVMALS: MIMO Volterra system identification with tensor networks," Zenodo.org, Apr. 2019, doi: 10.5281/zenodo.2644831.
[55] K. Batselier, "TNKalman: Tensor network Kalman filter for identification of time-varying MIMO Volterra systems," Zenodo.org, Apr. 2019, doi: 10.5281/zenodo.2644841.
[56] K. Batselier, "TNMOESP: Tensor network subspace identification of state space models with linear state dynamics and polynomial inputs," Zenodo.org, Apr. 2019, doi: 10.5281/zenodo.2644845.
[57] R. Penrose, "Applications of negative dimensional tensors," in *Combinatorial Mathematics and Its Applications.* D. Welsh, Ed. New York, NY, USA: Academic Press, 1971, pp. 221–244.