KU LEUVEN

STADIUS
Center for Dynamical Systems, Signal Processing and Data Analytics

| | |
|---|---|
| **Citation/Reference** | Vervliet, N., Debals, O., De Lathauwer, L. (2016), **Tensorlab 3.0 – Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization** in Proc. of the 50th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, Nov. 2016, pp. 1733–1738 |
| **Archived version** | Author manuscript: the content is identical to the content of the published paper, but without the final typesetting by the publisher |
| **Published version** | http://dx.doi.org/10.1109/ACSSC.2016.7869679 |
| **Journal homepage** | http://www.asilomarsscconf.org/ |
| **Author contact** | Nico.Vervliet@kuleuven.be<br>*+ 32 (0)16 3 20362* |
| **IR** | https://lirias.kuleuven.be/handle/123456789/577435 |

*(article begins on next page)*



# Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization

Nico Vervliet[†], Otto Debals[†*], Lieven De Lathauwer[†*]

[†]Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium;
[*]Group Science, Engineering and Technology, KU Leuven Kulak, E. Sabbelaan 53, B-8500 Kortrijk, Belgium.

***Abstract*—We give an overview of recent developments in numerical optimization-based computation of tensor decompositions that have led to the release of Tensorlab 3.0 in March 2016 (www.tensorlab.net). By careful exploitation of tensor product structure in methods such as quasi-Newton and nonlinear least squares, good convergence is combined with fast computation. A modular approach extends the computation to coupled factorizations and structured factors. Given large datasets, different compact representations (polyadic, Tucker, ...) may be obtained by stochastic optimization, randomization, compressed sensing, etc. Exploiting the representation structure allows us to scale the algorithms for constrained/coupled factorizations to large problem sizes.**

## I. Introduction

Central to multilinear algebra are tensors, or multi-way arrays of numerical values, and their many types of decompositions such as the Canonical Polyadic Decomposition (CPD), the Block Term Decomposition (BTD) or the Multilinear Singular Value Decomposition (MLSVD). Similar to their matrix counterparts, these decompositions can be used to analyze data, compress data, make predictions and much more. The multilinear structures allow more complex relations to be modeled, as has been shown in countless applications not only in signal processing [1], [2], [3], but, among others, also in data analytics and machine learning [4], [5], [6].

Tensorlab [7] is a MATLAB toolbox with as main purpose to provide user-friendly access to a variety of state-of-the-art numerical algorithms and tools for tensor computations. In March 2016, the third version of Tensorlab has been released. This paper gives a birds-eye overview of some new techniques that have been made available. The overview is by no means exhaustive: a full overview can be found at www.tensorlab.net.

This research is funded by (1) an Aspirant Grant from the Research Foundation — Flanders (FWO), (2) a Ph.D. grant of the Agency for Innovation by Science and Technology (IWT), (3) Research Council KU Leuven: C1 project c16/15/059-nD, CoE PFV/10/002 (OPTEC), (4) F.W.O.: project G.0830.14N and G.0881.14N, (5) the Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO II, Dynamical systems, control and optimization, 2012-2017), (6) EU: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Advanced Grant: BIOTENSORS (no. 339804). This paper reflects only the authors' views and the Union is not liable for any use that may be made of the contained information.

A number of demos illustrating good Tensorlab practice can be accessed at www.tensorlab.net/demos.

We continue this section by explaining the history and philosophy of Tensorlab and by fixing the notations. Section II discusses the SDF framework from Tensorlab, while Section III explains the concept of tensorization. Section IV introduces a new algorithm for coupled matrix/tensor factorizations in Tensorlab 3.0. Large-scale approaches are discussed in Section V, with a focus on compression, incompleteness, randomizations and efficient representations.

### A. History and philosophy

The first version of Tensorlab provided state-of-the-art algorithms for the computation of CPDs, BTDs or Low Multilinear Rank Approximations (LMRLA) as well as a large number of convenience methods involving tensors. These algorithms are based on the Complex Optimization Toolbox (COT) [8], [9], allowing decompositions of both real and complex datasets and/or variables. In optimization problems, real-valued functions with complex arguments are often split into the real part and the imaginary part, and both problems are solved separately. In contrast, the complex Taylor series expansion can be used to generalize standard real-valued optimization algorithms for complex arguments and data, thereby exploiting inherent structure present in derivatives which would otherwise be ignored [9], [10]. COT leverages this structure and provides generalizations of many standard optimization algorithms.

The Alternating Least Squares (ALS) algorithm is undoubtedly the most popular algorithm for tensor decompositions, mainly because of its simplicity. While it effectively exploits multilinear structures and often provides good results quickly, it is numerically not very sophisticated and it has no proven convergence [11], [12]. In Tensorlab, the main focus lies on more advanced optimization algorithms such as Nonlinear Least Squares (NLS) methods, thereby benefiting from the many good results in numerical optimization, including convergence guarantees. The number of iterations needed is often lower because of the quadratic convergence. The asymptotic cost per iteration of NLS can be reduced to the cost of ALS, although with some larger constants [11]. To achieve this low cost, the multilinear structure is exploited and a preconditioned

iterative solver is used to determine the step direction. In particular, in NLS algorithms the system

$$\mathbf{G}\mathbf{p} = -\mathbf{g} \tag{1}$$

is solved in every iteration, in which $\mathbf{G}$ is the Gramian of the Jacobian and $\mathbf{g}$ is the gradient. As inverting $\mathbf{G}$ is too expensive, the Conjugate Gradients (CG) method is used. CG requires only the matrix-vector products $\mathbf{G}\mathbf{p}$ to iteratively solve (1). In many tensor decomposition algorithms the multilinear structure can be exploited when computing these products. To reduce the number of CG iterations needed, preconditioning is used, i.e., instead of (1) the system

$$\mathbf{M}^{-1}\mathbf{G}\mathbf{p} = -\mathbf{M}^{-1}\mathbf{g} \tag{2}$$

is solved, in which the preconditioner $\mathbf{M}$ is an easily invertible matrix chosen such that (2) is easier to solve. (More technically, the eigenvalues of $\mathbf{M}^{-1}\mathbf{G}$ are more clustered than those of $\mathbf{G}$.) For tensor problems, a block-Jacobi preconditioner, i.e., a block diagonal approximation to $\mathbf{G}$, is often an effective choice [11]. The combination of low per-iteration cost with quadratic convergence of NLS type methods leads to a fast algorithm. In practice, the algorithms also seem more robust for ill-conditioned problems [11].

Since its official launch in February 2013, Tensorlab has seen two more releases. In January 2014, Tensorlab 2.0 was revealed, including the Structured Data Fusion (SDF) framework as its major feature. SDF allows structured and coupled decompositions of multiple full, sparse or incomplete matrices or tensors. This was inspired by the success of specific dedicated algorithms, each exploiting a particular type of constraint on the factor matrices. SDF allows the user to choose different decompositions, constraints and regularizations and combine these to their liking using SDF's own domain specific language [13]. By leveraging the chain rule for derivatives, parametric constraints can be handled easily: over 40 constraints are included, such as non-negativity, Toeplitz, polynomial, Kronecker, Vandermonde and matrix multiplication. Different types of regularization can be used to model soft constraints as well.

The most recent release from March 2016, Tensorlab 3.0, introduces tensorization and structured tensors, extends and improves the SDF framework while making it more user-friendly, introduces a number of large-scale algorithms and a new algorithmic family, improves coupled matrix/tensor factorizations, and much more. In the following sections, we discuss a number of these new features in more detail.

### B. Notation

An $N$th order tensor $\mathcal{T}$ can be factorized in various ways. The (Canonical) Polyadic Decomposition (CPD) writes the tensor as a (minimal) number of rank-1 terms, each of which is the outer product, denoted by $\otimes$, of $N$ non-zero vectors $\mathbf{a}_r^{(n)}$:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \otimes \cdots \otimes \mathbf{a}_r^{(N)} = \left[\!\!\left[ \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\!\right],$$

in which the factor matrix $\mathbf{A}^{(n)}$ contains the vectors $\mathbf{a}_r^{(n)}$ as its columns. The Higher-Order SVD (HOSVD) or Multilinear SVD (MLSVD) can be written as the mode-$n$ tensor-matrix product $\cdot_n$ of a core tensor $\mathcal{S}$ and $N$ factor matrices $\mathbf{U}^{(n)}$:

$$\mathcal{T} = \mathcal{S} \cdot_1 \mathbf{U}^{(1)} \cdot \cdots \cdot_N \mathbf{U}^{(N)}.$$

The Block Term Decomposition (BTD) writes a tensor as a sum of low-multilinear rank terms:

$$\mathcal{T} = \sum_{r=1}^{R} \mathcal{S}^{(r)} \cdot_1 \mathbf{U}^{(r,1)} \cdot \cdots \cdot_N \mathbf{U}^{(r,N)}.$$

A special variant of the BTD is the decomposition into a sum of multilinear rank-$(L_r, L_r, 1)$ terms (LL1):

$$\mathcal{T} = \sum_{r=1}^{R} (\mathbf{A}_r \mathbf{B}_r^{\mathrm{T}}) \otimes \mathbf{c}_r.$$

An overview of these decompositions is given in Fig. 1.

The mode-$n$ unfolding of a tensor $\mathcal{T}$ is denoted by $\mathbf{T}_{(n)}$ and concatenates the mode-$n$ vectors as columns in the matrix $\mathbf{T}_{(n)}$. The element-wise product or Hadamard product, the transpose and the Hermitian transpose are denoted by $*$, $\cdot^{\mathrm{T}}$ and $\cdot^{\mathrm{H}}$, respectively. $\langle \cdot, \cdot \rangle$ denotes the inner product.

## II. Structured Data Fusion

Structured Data Fusion (SDF) is a framework for rapid prototyping of analysis and knowledge discovery in one or more multidimensional datasets in the form of tensors. Fig. 2 gives a schematic overview. These tensors can be complex, incomplete, sparse and/or structured. Each tensor is decomposed using one of the tensor decompositions that are included in Tensorlab. The factor matrices are possibly shared between the different datasets, meaning that the tensors are coupled. They can also be equal within a tensor decomposition, indicating the presence of symmetry. Furthermore, besides the choice of factorizations, regularization terms can be added as well, based on $L_0$, $L_1$ or $L_2$ norms. Regularization can be used to prevent overfitting but also to implement soft constraints.

In a lot of applications, prior knowledge is available on the factor matrices indicating some kind of structure such as orthogonality or non-negativity. More than 40 structures are readily available in Tensorlab to constrain the factor matrices. Besides the administered structures, a user can design its own constraints as well by providing the mapping and its first-order derivative information. It is worthwhile to note that the constraints are implemented with parametric transformations of underlying optimization variables, rather than with penalty terms. For example, an orthogonal factor matrix of size $I \times R$ requires only $R(I-(R-1)/2)$ variables while a Vandermonde matrix of size $I \times R$ requires only $I$ generating variables. Hence, the solution space is reduced to a restricted search space, and the constraints are imposed exactly rather than only approximately. The chain rule is then internally used to cope with the composition of the tensor decomposition model and the various transformations/constraints, and to solve for the underlying variables.

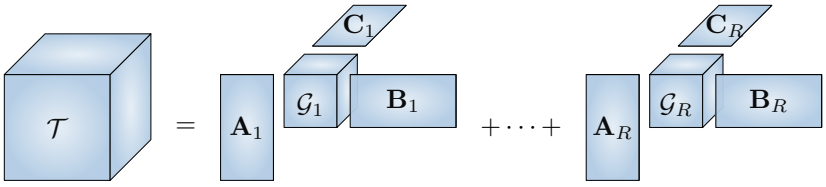



Fig. 1. Block term decomposition of a tensor $\mathcal{T}$ in terms with multilinear ranks $(L_r, M_r, N_r)$. If $R = 1$, a LMLRA is obtained. If $M_r = L_r$ and $N_r = 1$, thus if the $r$th core tensor has size $(L_r, L_r, 1)$, a BTD in multilinear rank-$(L_r, L_r, 1)$ terms is obtained. If $L_r = M_r = N_r = 1$, a CPD is obtained.

The type of tensor decomposition, the coupling and the structure imposed on the factors can all be chosen independently of the solver and its options. Two different popular classes of algorithms are available to solve SDF problems in Tensorlab: quasi-Newton (QN) methods and Nonlinear Least Squares (NLS) methods, implemented in `sdf_minf` and `sdf_nls`, respectively. Within the QN methods, both limited memory BFGS (L-BFGS, subdivided in line search and trust region approaches) and Nonlinear Conjugate Gradient (NCG) methods can be selected, while Gauss–Newton (CG-Steihaug and Dogleg trust region approaches) and Levenberg–Marquardt algorithms are implemented within the NLS class.

In Tensorlab 3.0, the SDF framework has been updated in several respects. Two new solvers for symmetric and/or coupled CPDs are introduced (as discussed in Section IV), as well as three new factorization types and various updated and new transformations. Besides a focus on content, there has also been a focus on user-friendliness. Using a new language parser (`sdf_check`), it is easier to formulate SDF models and to investigate them. It also helps finding errors in the model. Furthermore, the domain specific language has been made more lenient to allow more flexible model formulation, e.g., by automatically converting arrays to cells and adding braces, wherever necessary.

The handling of incomplete and sparse tensors has also improved from Tensorlab 3.0 on. Note that with the surge of big data applications in mind, the Tensorlab algorithms have a linear time complexity in the number of known/non-zero elements of the data tensor. The SDF features regarding incomplete tensors have shown its value in various applications before, such as in movie recommendation and user participation predictions [13] as well as in the design of alloys and in multidimensional harmonic retrieval [14]. This is further discussed in Section V-B.

## III. Tensorization

Many powerful tensor tools have been developed throughout the years for analyzing multi-way data. When no tensor data is available and only a matrix is given, tensor tools may still be used after first transforming the matrix data to tensor data. This transformation is called tensorization, and many different mappings are possible. The tensorization step is conceptually an important step by itself. Many results concerning tensorization have appeared in the literature in a disparate manner but have not been discussed as such, e.g., [15].

After the tensorization step, one often computes a tensor decomposition. This is especially the case in blind signal separation, where the first tensorization step implements assumptions on the source signals while the second decomposition step realizes the actual separation of the source signals [16].

Tensorlab 3.0 contains a number of tensorization techniques [16]. Hankelization (Hankel-based mapping) and Löwnerization (Löwner-based mapping) can be used when dealing with approximations by exponentials/sinusoids and rational functions, respectively. Segmentation and decimation are based on folding matrix data, which is, e.g., useful when dealing with large-scale data [17]. Also higher-order and lagged second-order statistics have been included.

Corresponding detensorization techniques have been included where possible. They can be useful, for example, to extract source estimates from the terms in the tensor decomposition. By providing a (noisy) Hankel matrix or tensor for example, the command `dehankelize` returns the averaged anti-diagonals or anti-diagonal slices, respectively.

Tensorization typically involves including redundant information in the higher-order tensor. The number of elements in the obtained tensor can grow quickly, in line with the curse of dimensionality which states that the number of elements in a tensor increases exponentially with the number of dimensions, and so do the computational and memory requirements. To cope with this curse, Tensorlab 3.0 can use efficient representations of the higher-order tensors resulting from the tensorization. The efficiency of these representations can then be exploited in the decomposition algorithms, as discussed in Section V-D.

## IV. Coupled matrix/tensor factorization

Joint decomposition of multiple datasets into rank-1 terms is a common problem in data analysis. Often symmetry constraints are used as well. Both coupling and symmetry, at the level of the data and the factorization, are easy to implement using SDF. In this section, we discuss how the new, specialized Coupled and Symmetric CPD (CCPD) solver improves convergence and reduces computation time compared to the standard SDF solvers by exploiting both constraints early.

The general SDF solvers `sdf_minf` and `sdf_nls` handle coupling and symmetry by first computing the Gramian of the Jacobian $\mathbf{G}$ and the gradient $\mathbf{g}$ as if no constraints were imposed (see Equation (1)). $\mathbf{G}$ and $\mathbf{g}$ are then contracted to $\mathbf{G}_c$ and $\mathbf{g}_c$ which, in this case, boils down to summing the

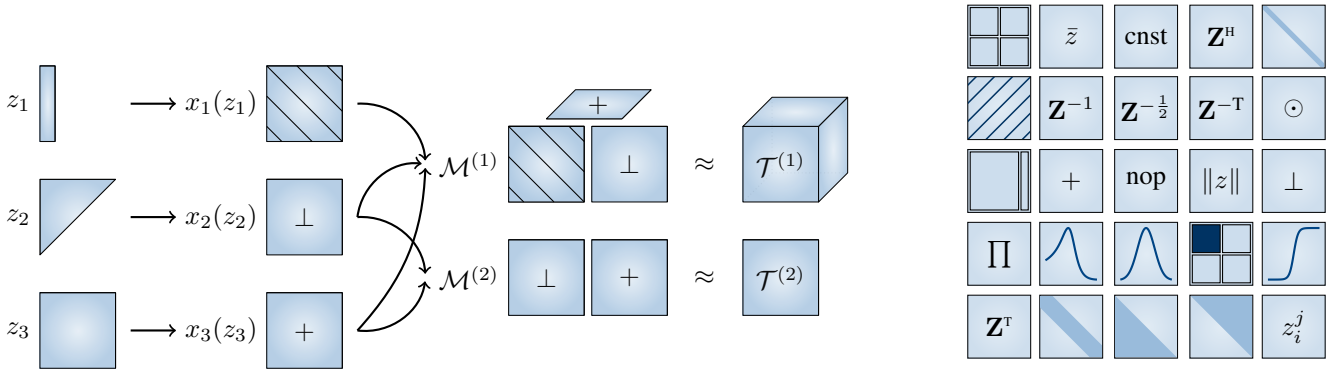



Fig. 2. On the left-hand side a schematic of structured data fusion (adapted from [13]). The vector $z_1$, upper triangular matrix $z_2$ (representing a sequence of Householder reflectors) and full matrix $z_3$ are transformed into a Toeplitz, orthogonal and nonnegative matrix, respectively. The resulting factors are then used to jointly factorize two coupled datasets $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$. More than 40 structures can be imposed on factor matrices. 25 examples are shown schematically on the right-hand side.

proper blocks, as indicated in Fig. 3. The result is a smaller system which is cheaper to solve.

Fig. 3 shows that many blocks in $\mathbf{G}$ are repeated because of symmetry. The `ccpd_nls` function takes this into account directly: each unique block is multiplied by the number of occurrences instead of summing all blocks after computing them. In the case of the gradient, symmetry in the data $\mathcal{T}$ and the decomposition, e.g., $[\![\mathbf{A}, \mathbf{A}, \mathbf{B}]\!]$, has to be considered. In the example, the decomposition is symmetric in the first two modes as the factor matrices are identical. The gradients w.r.t. the first and second mode are only identical if $\mathcal{T}$ is symmetric in the first two modes as well. If this is the case, computing the gradient w.r.t. to second mode is unnecessary. Otherwise, there is no computational gain possible. Detecting symmetry is therefore an important task in the CCPD solvers.

For a regular CPD, a block-Jacobi preconditioner has shown to be effective and efficient to reduce the cost of solving (1) because of the Kronecker structure [11]. The `ccpd_nls` algorithm uses a similar preconditioner that exploits symmetry and coupling while keeping the Kronecker structure, in contrast to the non-preconditioned `sdf_nls` algorithm.

To illustrate the performance gain of the new algorithm, consider the following coupled and symmetric problem [18]:

$$\min_{\mathbf{M},\boldsymbol{\kappa}} \left|\left|\mathbf{C}^{(2)} - \mathbf{M}\mathbf{M}^{\mathrm{T}}\right|\right|^2 + \left|\left|\mathcal{C}^{(4)} - [\![\mathbf{M}, \mathbf{M}, \mathbf{M}, \mathbf{M}, \boldsymbol{\kappa}]\!]\right|\right|^2 \quad (3)$$

in which $\mathbf{C}^{(2)}$ and $\mathcal{C}^{(4)}$ are constructed using $\mathbf{M} \in \mathbb{R}^{50\times 25}$ and $\boldsymbol{\kappa} \in \mathbb{R}^{1\times 25}$ drawn from a normal distribution. In Table I the SDF and the NLS algorithms are compared [1]. It is clear that exploiting all symmetry reduces the time per iteration. The block-Jacobi preconditioner used to solve (1) improves convergence considerably as can be seen from the reduced number of iterations. The combination of all improvements reduces the total computation time significantly.

[1]The timings for both algorithms benefited from a modified version of `mtkrprod` which is not yet released.

TABLE I
COMPARED TO SDF, CCPD REQUIRES LESS TIME AND FEWER ITERATIONS TO CONVERGE WHEN COMPUTING (3). INCREASING THE NUMBER OF CG ITERATIONS IMPROVES CONVERGENCE AND REDUCES COMPUTATION TIME. ALL NUMBERS ARE MEDIANS OVER 50 EXPERIMENTS. BOTH ALGORITHMS USE THE OPTIONS `TolX = eps` AND `TolFun = eps^2`, WITH `eps` THE MACHINE PRECISION.

| | 25 CG Iter. | | 75 CG Iter. | |
|---|---|---|---|---|
| | SDF | CCPD | SDF | CCPD |
| Time (s) | 70.3 | 6.9 | 19.4 | 6.2 |
| Iterations | 170.0 | 45.5 | 39.5 | 29.5 |
| Time/iteration (s) | 0.40 | 0.15 | 0.47 | 0.20 |

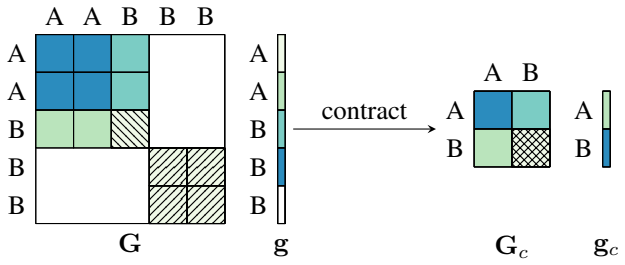



Fig. 3. Illustration of contraction in `sdf_nls` for a decomposition of $\mathcal{T} = [\![\mathbf{A}, \mathbf{A}, \mathbf{B}]\!]$ and $\mathbf{M} = \mathbf{B}\mathbf{B}^{\mathrm{T}}$. All blocks with the same shading are identical. The CCPD algorithm computes the contracted Gramian and gradient directly.

## V. LARGE-SCALE TENSOR DECOMPOSITIONS

There exist many strategies for handling large-scale tensors: parallelization of operations, parallel decompositions, incompleteness, compression, exploitation of sparsity and so on. Here, we discuss four techniques readily available in Tensorlab: MLSVD computation using randomized matrix algebra, the use of incomplete tensors and randomized block sampling for polyadic decompositions, and the use of structured tensors.

### A. Randomized compression

Using randomized matrix algebra, we derive a fast yet precise algorithm for computing an approximate multilinear

singular value decomposition of a tensor $\mathcal{T}$. The standard way to compute an MLSVD uses the matrix SVD to compute the left singular vectors $\mathbf{U}^{(n)}$, $n = 1, 2, 3$, of the different unfoldings $\mathbf{T}_{(n)}$ of the tensor, and computes the core tensor $\mathcal{S}$ as $\mathcal{T} \cdot_1 \mathbf{U}^{(1)^{\mathrm{T}}} \cdot_2 \mathbf{U}^{(2)^{\mathrm{T}}} \cdot_3 \mathbf{U}^{(3)^{\mathrm{T}}}$ [19]. In very recent literature, one has replaced the SVD by a randomized variant from [20]. Here we present a variant that combines a sequential truncation strategy [21] with randomized SVDs and $Q$ subspace iterations [20]. The full algorithm is described in Algorithm 1.

**Algorithm 1:** Computation of MLSVD using randomization and subspace iteration. (Implemented as `mlsvd_rsi`.)

**Input**: $N$th order tensor $\mathcal{T}$ of size $I_1 \times \cdots \times I_N$, compression size $R_1 \times \cdots \times R_N$, oversampling parameter $P$ and number of subspace iterations $Q$.
**Output**: Factor matrices $\mathbf{U}^{(n)}$, $n = 1, \ldots, N$ and core tensor $\mathcal{S}$ such that $\mathcal{S} \cdot_1 \mathbf{U}^{(1)} \cdot \cdots \cdot_N \mathbf{U}^{(N)} \approx \mathcal{T}$.

Set size $s_n \leftarrow I_n$, $n = 1, \ldots, N$ and $\mathcal{Y} \leftarrow \mathcal{T}$;
**for** $n = 1 \ldots, N$ **do**
  Let $\mathbf{\Omega}$ be a random matrix of size $\prod_{k \neq n} s_k \times R_n + P$;
  $\mathbf{QR} \xleftarrow{QR} \mathbf{Y}_{(n)}\mathbf{\Omega}$;
  **for** $q = 1, \ldots, Q$ **do**
    $\mathbf{QR} \xleftarrow{QR} \mathbf{Y}_{(n)}^{\mathrm{T}}\mathbf{Q}$;
    $\mathbf{QR} \xleftarrow{QR} \mathbf{Y}_{(n)}\mathbf{Q}$;
  $\mathbf{USV}^{\mathrm{T}} \xleftarrow{SVD} \mathbf{Q}^{\mathrm{T}}\mathbf{Y}_{(n)}$;
  $\mathbf{U}^{(n)} \leftarrow \mathbf{QU}(:, 1 : J_n)$;
  $s_n \leftarrow R_n + P$;
  $\mathcal{Y} \leftarrow \text{reshape}(\mathbf{SV}^{\mathrm{T}}, s_1, \ldots, s_N)$;
$\mathcal{S} \leftarrow \mathcal{Y}(1 : R_1, \ldots, 1 : R_N)$;

As example, we create 400 random third-order tensors of size $I_1 \times I_2 \times I_3$ with $I_n$ uniformly distributed in $[100; 400]$ and with multilinear ranks $(R_1, R_2, R_3)$ with $R_n$ distributed uniformly in $[10; 50]$, $n = 1, 2, 3$. The compression size is $(\tilde{R}_1, \tilde{R}_2, \tilde{R}_3)$ with $\tilde{R}_n$ distributed uniformly in $[10; 40]$. The oversampling parameter $P$ is 5 and the number of subspace iterations $Q$ is 2. The relative Frobenius norm error is maximally 4.2% higher in the case of the randomized algorithm `mlsvd_rsi` compared to the standard algorithm `mlsvd`, while the speedup is a factor 3 for small tensors and a factor 25 for larger tensors. If the used compression size is equal to or larger than the multilinear rank of the tensor, the mean relative errors are $1.3 \cdot 10^{-14}$ and $0.5 \cdot 10^{-14}$ for the standard and the randomized algorithm, respectively.

### B. Incomplete tensors

Incomplete tensors occur for two main reasons. First, one can be unable to know some entries, for example, because a sensor breaks down, or because some entries correspond to physically impossible situations, e.g., negative concentrations [14]. In the second case, all elements could be known, but computing or storing all entries is too costly, hence some elements are deliberately omitted. For example, for a rank-$R$ CPD of an $N$th order tensor $\mathcal{T}$ of size $I \times \cdots \times I$, the number of entries is $I^N$, while the number of variables is only $NIR$. Hence, the number of entries scales exponentially in the order, while the number of variables scales only linearly. This enables the use of very sparse sampling schemes [14].

Here, we restrict the discussion to the computation of a CPD of an incomplete tensor. Three main techniques can be found in literature [14]. First, unknown elements can be imputed, e.g., by replacing all unknown values with the mean value or with zero. Second, in an expectation-maximization scheme, the unknown values are imputed each iteration with the current best guess from the model. Third, the unknown elements can be ignored altogether. In this last approach, the objective function for a CPD becomes

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \left|\left| \mathcal{W} * (\mathcal{T} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) \right|\right|_{\mathrm{F}}^2, \tag{4}$$

in which $\mathcal{W}$ is a binary observation tensor. Various optimization schemes have been used to minimize objective (4) [13], [22], [23], [24].

Two NLS type algorithms are available in Tensorlab. The first technique scales the Gramian by the fraction of known values, but ignores the structure of the missing data [13]. While this approach is very fast, the result may not be accurate in some cases. If the number of known entries is extremely small, this algorithm may fail [24]. The second technique uses the exact Gramian of the Jacobian, i.e, the structure of the missing data is exploited. Second-order convergence can be achieved, but each iteration is relatively expensive. This is often compensated for, however, as the number of iterations needed for convergence is reduced significantly. As shown in [24], leveraging the exact Gramian can sometimes be crucial in order to find a reasonable solution.

### C. Randomized block sampling

A third technique involves full tensors which may not fit into memory entirely, or for which the computation cost per iteration would be excessive. In [25] a technique called Randomized Block Sampling (RBS) was presented to compute the CPD of large-scale tensors. This method combines block coordinate descent techniques with stochastic optimization as follows. Every iteration, a random subtensor or *block* is sampled from the full tensor. Using this block, one optimization step is performed. Due to the structure of a CPD, only a limited amount of variables are affected in each step. This means multiple steps from multiple blocks can be computed in parallel, as long as the affected variables do not overlap. As only small blocks are used, there is no need to load the full tensor. Blocks can also be generated on-the-fly obfuscating the need to construct a tensor beforehand. Thanks to a simple step restriction schedule, the underlying CP structure can be recovered almost as accurately as if the full tensor were decomposed, even if only a fraction of the data is used.

### D. Efficient representation of structured tensors

Tensors are not always given as a multi-way array of numbers or as a list of non-zeros or known entries. The tensor

can, for example, be given in the Tucker format as a result of randomized compression, as a Tensor Train approximation [26] to solution of a partial differential equation, or in a Hankel format after tensorization (see Section III). As discussed in [27], the efficient representation of a tensor $\mathcal{T}$ can be exploited by rewriting the objective function

$$\min \|\mathcal{T} - \hat{\mathcal{T}}\|_{\mathrm{F}}^2 \;=\; \min \|\mathcal{T}\|_{\mathrm{F}}^2 - 2\langle \mathcal{T}, \hat{\mathcal{T}} \rangle + \|\hat{\mathcal{T}}\|_{\mathrm{F}}^2, \quad (5)$$

in which $\hat{\mathcal{T}}$ can be a CPD, an LMLRA, an LL1 or a BTD. The gradients can be rewritten in a similar way. All norms and inner products at the right-hand side of (5), and all matricized tensor times Khatri–Rao or Kronecker products needed for the gradients can be computed efficiently by exploiting the structure of $\mathcal{T}$ and $\hat{\mathcal{T}}$. This can lead to speedups in many algorithms, including ALS, quasi-Newton and NLS algorithms.

Exploiting the structure of tensors by rewriting the objective function as (5) does not change the optimization variables. This has as consequence that constraints on factor matrices, symmetry or coupling can be handled trivially. Consider, for example, a non-negative CPD of a large-scale tensor. Without constraints, the tensor $\mathcal{T}$ is often compressed first to reduce the computational complexity using $\mathcal{S} = \mathcal{T} \cdot_1 \mathbf{U}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{U}^{(N)^{\mathrm{T}}}$. The compressed tensor $\mathcal{S}$ is then decomposed as $\left[\!\left[ \hat{\mathbf{A}}^{(1)}, \ldots, \hat{\mathbf{A}}^{(N)} \right]\!\right]$. The CPD $\left[\!\left[ \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right]$ of $\mathcal{T}$ can be recovered using $\mathbf{A}^{(n)} = \mathbf{U}^{(n)} \hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$. This technique cannot be used to compute the non-negative CPD, as non-negativity is not preserved by compression, i.e., $\hat{\mathbf{A}}^{(n)} \geq 0$ does not imply $\mathbf{A}^{(n)} \geq 0$ in which $\geq$ holds entry-wise. However, using the structured format, the optimization variables are the full factor matrices $\mathbf{A}^{(n)}$ instead of the compressed ones $\hat{\mathbf{A}}^{(n)}$. Hence standard non-negativity techniques can be used, while still exploiting the structure.

## VI. Conclusion

In this paper, we have elaborated on a number of features that have been introduced by the third release of Tensorlab in March 2016. First of all, new factorizations and constraints are available in the Structured Data Fusion (SDF) framework. A new tool improves the user-friendliness of SDF by finding model errors early. A number of tensorization and detensorization methods have also been added, allowing the transformation of lower-order to higher-order data, and vice versa. By carefully exploiting the symmetry and coupling structure in different stages including the preconditioning stage, new solvers for coupled matrix/tensor factorizations have been enabled. Furthermore, a number of new large-scale approaches have been discussed in this paper, such as randomized block sampling and decompositions of large structured tensors using efficient representations.

## References


[1] A. Cichocki, D. Mandic, A. H. Phan, C. Caiafa, G. Zhou, Q. Zhao, and L. De Lathauwer, "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.

[2] P. Comon, "Tensors: a brief introduction," *IEEE Signal Process. Mag.*, vol. 31, no. 3, pp. 44–53, 2014.

[3] N. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," June 2016, Technical Report 16–34, ESAT-STADIUS, KU Leuven, Belgium.

[4] A. K. Smilde, R. Bro, P. Geladi, and J. Wiley, *Multi-way analysis with applications in the chemical sciences*. Wiley Chichester, UK, 2004.

[5] E. Acar and B. Yener, "Unsupervised multiway data analysis: A literature survey," *IEEE Trans. Knowl. Data Eng*, vol. 21, no. 1, pp. 6–20, 2009.

[6] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "A survey of multilinear subspace learning for tensor data," *Pattern Recognition*, vol. 44, no. 7, pp. 1540–1551, 2011.

[7] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer. "Tensorlab 3.0." Available online. http://www.tensorlab.net.

[8] L. Sorber, M. Van Barel, and L. De Lathauwer. "Complex optimization toolbox v1.0." Available online. http://esat.kuleuven.be/stadius/cot/.

[9] ——, "Unconstrained optimization of real functions in complex variables," *SIAM J. Optim.*, vol. 22, no. 3, pp. 879–898, 2012.

[10] T. Adalı, P. J. Schreier, and L. L. Scharf, "Complex-valued signal processing: The proper way to deal with impropriety," *IEEE Trans. Signal Process*, vol. 59, no. 11, pp. 5101–125, Nov 2011.

[11] L. Sorber, M. Van Barel, and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization," *SIAM J. Optim.*, vol. 23, no. 2, pp. 695–720, 2013.

[12] A. Uschmajew, "Local convergence of the alternating least squares algorithm for canonical tensor approximation," *SIAM J. Matrix Anal. Appl.*, vol. 33, no. 2, pp. 639–652, Jan 2012.

[13] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion," *IEEE J. Sel. Topics Signal Process.*, vol. 9, no. 4, pp. 586–600, June 2015.

[14] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, "Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 71–79, Sept. 2014.

[15] A.-J. Van der Veen and A. Paulraj, "An analytical constant modulus algorithm," *IEEE Trans. Signal Process*, vol. 44, no. 5, pp. 1136–1155, 1996.

[16] O. Debals and L. De Lathauwer, "Stochastic and deterministic tensorization for blind signal separation," in *Latent Variable Analysis and Signal Separation*, ser. Lecture Notes in Computer Science, vol. 9237. Springer Berlin / Heidelberg, 2015, pp. 3–13.

[17] M. Boussé, O. Debals, and L. De Lathauwer, "A tensor-based method for large-scale blind source separation using segmentation," *IEEE Trans. Signal Process*, pp. 1–1, 2016.

[18] O. Debals, N. Vervliet, F. Van Eeghem, and L. De Lathauwer, "Using coupled tensor decompositions and incomplete tensors for independent component analysis," 2016, Technical Report 16–171, ESAT-STADIUS, KU Leuven, Belgium.

[19] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, p. 12531278, Jan 2000.

[20] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011.

[21] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition," *SIAM J. Sci. Comput.*, vol. 34, no. 2, pp. A1027–A1052, Jan 2012.

[22] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable tensor factorizations for incomplete data," *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, 2011.

[23] G. Tomasi and R. Bro, "PARAFAC and missing values," *Chemom. Intell. Lab. Syst.*, vol. 75, no. 2, pp. 163–180, 2005.

[24] N. Vervliet, O. Debals, and L. De Lathauwer, "Canonical polyadic decomposition of incomplete tensors with linearly constrained factors," 2016, Technical Report 16–172, ESAT-STADIUS, KU Leuven, Belgium.

[25] N. Vervliet and L. De Lathauwer, "A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 2, pp. 284–295, 2016.

[26] I. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011.

[27] N. Vervliet, O. Debals, and L. De Lathauwer, "Exploiting efficient data representations in tensor decompositions," 2016, Technical Report 16–174, ESAT-STADIUS, KU Leuven, Belgium.