CANONICAL POLYADIC DECOMPOSITION OF INCOMPLETE TENSORS WITH LINEARLY CONSTRAINED FACTORS*

NICO VERVLIET†, OTTO DEBALS†, AND LIEVEN DE LATHAUWER†

Abstract. Incomplete datasets are prevalent in domains like machine learning, signal processing and scientific computing. Simple models often allow missing data to be predicted or can be used as representations for full datasets to reduce the computational cost. When data are explicitly or implicitly represented as tensors, which are multiway arrays of numerical values, a low-rank assumption is often appropriate. A nonlinear least squares algorithm to decompose an incomplete tensor into a sum of rank-1 terms is presented here. Furthermore, the algorithm allows the incorporation of prior knowledge in the form of linearly constrained factor matrices. By combining the exact Gramian and a novel statistical preconditioner, the algorithm allows faster and more accurate decompositions compared to state-of-the-art methods when few entries are known or when missing entries are structured. The accuracy is improved further while fewer known entries are required, by carefully exploiting the linear constraints. A specialized variant of the algorithm removes the data dependence from the optimization step by utilizing a compressed representation. When relatively many entries are known, computation time can be reduced significantly using this variant, as illustrated in a novel materials science application.

Key words. canonical polyadic decomposition, candecomp, parafac, CPD, CANDELINC, missing entries, incomplete tensors, big data, compressed sensing, multivariate regression

AMS subject classifications. 15A69

1. Introduction.

Tensors, or multiway arrays of numerical values, are a higher-order generalization of vectors and matrices and can be found in a variety of applications in, e.g., signal separation, object recognition, pattern detection, numerical simulations, function approximation and uncertainty quantification. Many more examples can be found in numerous overview papers [11, 15, 16, 22, 32]. As the number of entries in a tensor scales exponentially in its order, tensor problems become large-scale quickly. The computational challenges related to this exponential increase, are known as the curse of dimensionality. To extract information or to efficiently represent tensors, low rank assumptions are often appropriate. The (canonical) polyadic decomposition (CPD), which is also known as PARAFAC, CANDECOMP, tensor rank decomposition or separated representation, writes an Nth-order tensor as a (minimal) number of R rank-1 terms, each of which is the outer product of N non-zero vectors, with R the tensor rank. Mathematically, we have

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

*Submitted to the editors on April 19, 2017.

Funding: Nico Vervliet is supported by an Aspirant Grant from the Research Foundation — Flanders (FWO). Otto Debals is supported by a Ph.D. grant of the Agency for Innovation by Science and Technology (IWT). Research furthermore supported by: (1) Flemish Government: FWO: projects: G.0830.14N, G.0881.14N; (2) 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 (n° 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; (3) KU Leuven Internal Funds C16/15/059.

†KU Leuven, Dept. of Electrical Engineering ESAT/STADIUS, Kasteelpark Arenberg 10, bus 2446, B-3001 Leuven, Belgium; and Group Science, Engineering and Technology, KU Leuven - Kulak, E. Sabbelaan 53, 8500 Kortrijk, Belgium (Nico.Vervliet@kuleuven.be, Otto.Debals@kuleuven.be, Lieven.DeLathauwer@kuleuven.be).

This manuscript is for review purposes only.

where $\otimes$ denotes the outer product and $\mathbf{a}_r^{(n)}$ is the rth column of factor matrices $\mathbf{A}^{(n)}$, for $r = 1, \ldots, R$ and $n = 1, \ldots, N$. Many applications benefit from the mild uniqueness properties of a CPD, allowing components to be separated using few restrictions [11, 22, 32]. In contrast to the number of entries, the number of parameters in a CPD scales linearly in the order, allowing a compact representation of higher-order data. To handle large-scale problems, various approaches have been proposed including techniques based on compression [34], randomization [29, 42] and sparsity combined with parallel architectures [18, 35].

Measuring, computing and/or storing all entries is often expensive in terms of time, memory or cost. For example, in high-dimensional integration, the number of points easily exceeds the number of atoms in the universe [28]; in uncertainty quantification, each entry requires the solution of a PDE [23]; or in multivariate regression in many variables it may be impossible to store all possible values of a discretized function [7]. Reducing the number of entries by deliberately taking few samples is therefore important or even necessary to keep the problem manageable [43]. Also, some entries can be missing or unknown due to, e.g., broken sensors, artifacts which have been removed or physically impossible situations. Whether few entries are sampled or some entries are missing, specialized algorithms, e.g., [2, 20, 39, 40, 43], are key to decompose and analyze the resulting incomplete tensors. It is often important that the creation of the full tensor is avoided, especially for large-scale tensors.

Prior knowledge is often available, allowing the construction of more accurate and/or interpretable models. Moreover, additional constraints are important to further lower the number of required samples or to improve the accuracy. In this paper, we focus on prior knowledge that can be represented as linear constraints on the factor matrices of a CPD, i.e., $\mathbf{A}^{(n)} = \mathbf{B}^{(n)}\mathbf{C}^{(n)}$ in which $\mathbf{B}^{(n)}$ is a known matrix, while $\mathbf{C}^{(n)}$ contains the unknown coefficients. This model is also known as CANDELINC [13] and is ubiquitous although it is not always recognized as a constrained CPD. For example, in multivariate regression with polynomials or sparse grids, $\mathbf{B}^{(n)}$ can be a basis of polynomials or multilevel tent functions [7, 14]. Some applications benefit from other types of bases like Gaussian kernels or trigonometric functions [45]. Another example can be found in compressed sensing (CS) where a compressed measurement $\mathbf{b}$ is represented in a dictionary $\mathbf{D}$ by a sparse coefficient vector $\mathbf{x}$ [10, 12]. If $\mathbf{D}$ is expressed as a Kronecker product and $\mathbf{x}$ is a (possibly sparse) vectorized CPD [9, 33], the CS problem can be written as a linearly constrained CPD.

Computing a linearly constrained CPD of a full tensor is straightforward as the tensor can be compressed using the known matrices $\mathbf{B}^{(n)}$ after which the matrices $\mathbf{C}^{(n)}$ can be found from the unconstrained CPD of the compressed tensor. Because of its efficiency, compression using suitable basis matrices [4] or the Tucker decomposition/MLSVD [21, 8] is a commonly used preprocessing step. For incomplete tensors, only algorithms for unconstrained decompositions are currently available. These algorithms can be divided into imputation or expectation-maximization algorithms which replace missing entries with some value, and direct minimization techniques which take only known entries into account. Examples for the former can be found in [17, 30, 40], while CCD++ [20], CP WOPT [2], INDAFAC [40] and CPD [39] are examples of the latter. The structured data fusion (SDF) framework from [39] can be used to model linear constraints, but it does not fully exploit the inherent multilinear structure. Dedicated algorithms have been presented for specific problems, e.g., in [7] an alternating least squares (ALS) algorithm is presented for multivariate regression problems. Here, we focus on the direct minimization techniques, as the memory

This manuscript is for review purposes only.

requirements often prohibit the construction of the imputed tensor.

The efficient computation of a linearly constrained CPD of an incomplete tensor is our main focus. Two variants are presented in section 2: a data-dependent algorithm CPDLI DD which has a per-iteration complexity linear in the number of known entries and a data-independent algorithm CPDLI DI which has a per-iteration complexity depending only on the number of variables in the decomposition and not on the tensor entries. The presented algorithms are of the (inexact) Gauss–Newton type (see subsection 1.2); quasi-Newton or ALS algorithms can be derived easily. After fixing the notation and reviewing the used optimization framework in the remainder of this section, the algorithms and their complexity are derived in section 2. The unconstrained CPD of an incomplete tensor is discussed as a special case in section 3. A new statistical preconditioner is discussed in section 4. Section 5 and section 6 illustrate the performance of the presented algorithms on synthetic data and a materials science example, respectively. To keep the main text and idea clear, the derivations of the algorithms and preconditioner are discussed in the supplementary material accompanying this paper.

1.1. Notation. Scalars, vectors, matrices and tensors are denoted by lower case, e.g., a, bold lower case, e.g., $\mathbf{a}$, bold upper case, e.g., $\mathbf{A}$ and calligraphic, e.g., $\mathcal{T}$, letters, respectively. $\mathbb{K}$ denotes either $\mathbb{R}$ or $\mathbb{C}$. Sets are indexed by superscripts within parentheses, e.g., $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$. An incomplete tensor $\mathcal{T}$ has N_{ke} known entries at positions $(i_k^{(1)}, i_k^{(2)}, \ldots, i_k^{(N)})$, $k = 1, \ldots, N_{\text{ke}}$. We collect all indices of known entries in index vectors $\mathbf{i}^{(1)}, \ldots, \mathbf{i}^{(N)}$. The extended form of a matrix $\mathbf{A}^{(n)} \in \mathbb{K}^{I_n \times R}$ is denoted by $\mathbf{A}^{[n]} = \mathbf{A}^{(n)}(\mathbf{i}^{(n)}, :)$ using MATLAB style indexing. $\mathbf{A}^{[n]} \in \mathbb{K}^{N_{\text{ke}} \times R}$ contains repeated rows as $1 \leq i_k^{(n)} \leq I_n$ and $N_{\text{ke}} \geq I_n$ and is used to facilitate the derivation the algorithms. A mode-n vector is the generalization of a column (mode-1) and row (mode-2) vector and is defined by fixing all but the nth index of an Nth-order tensor. The mode-n unfolding of a tensor $\mathcal{T}$ is denoted by $\mathbf{T}_{(n)}$ and has the mode-n vectors as its columns. The vectorization operator $\text{vec}(\mathcal{T})$ stacks all mode-1 vectors into a column vector. Details on the order of the mode-n vectors in unfoldings and vectorizations can be found in [11]. A number of products are needed. The mode-n tensor-matrix product is denoted by $\mathcal{T} \cdot_n \mathbf{A}$. The Kronecker and Hadamard, or element-wise, product are denoted by $\otimes$ and $*$, respectively. The column-wise (row-wise) Khatri–Rao products between two matrices, denoted by $\odot$ ($\odot^{\mathrm{T}}$), is defined as the column-wise (row-wise) Kronecker product. To simplify expressions, the following shorthand notations are used

$$\bigotimes_{n} \equiv \bigotimes_{n=N}^{1} \qquad \bigotimes_{k \neq n} \equiv \bigotimes_{\substack{k=N \\ k \neq n}}^{1}, \tag{2}$$

where N is the order of the tensor and indices are traversed in reverse order. Similar definitions are used for $\odot$, $\odot^{\mathrm{T}}$ and $*$. To reduce the number of parentheses needed, we assume the matrix product takes precedence over the Kronecker, (row-wise) Khatri–Rao and Hadamard products, e.g.,

$$\mathbf{AB} \odot \mathbf{CD} \equiv (\mathbf{AB}) \odot (\mathbf{CD}). \tag{3}$$

This manuscript is for review purposes only.

Similarly, shorthand notations are expanded first, e.g., for $N = 3$, we have

$$\bigodot_{k\neq 2} \mathbf{A}^{(k)} \odot \mathbf{A}^{(2)} \equiv \left(\mathbf{A}^{(3)} \odot \mathbf{A}^{(1)}\right) \odot \mathbf{A}^{(2)}, \tag{4}$$

$$\bigodot_{k\neq 2} \mathbf{B}^{(k)}\mathbf{C}^{(k)} \otimes \mathbf{A}^{(2)} \equiv \left((\mathbf{B}^{(3)}\mathbf{C}^{(3)}) \odot (\mathbf{B}^{(1)}\mathbf{C}^{(1)})\right) \otimes \mathbf{A}^{(2)}. \tag{5}$$

The complex conjugate, transpose, conjugated transpose and pseudo-inverse are denoted by $\bar{\cdot}$, $\cdot^{\mathrm{T}}$, $\cdot^{\mathrm{H}}$ and $\cdot^{\dagger}$, respectively. The column-wise concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$ is written as $\mathbf{x} = [\mathbf{a}; \mathbf{b}]$ and is a shorthand for $\mathbf{x} = \begin{bmatrix}\mathbf{a}^{\mathrm{T}} & \mathbf{b}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$. The inner product between $\mathcal{A}$ and $\mathcal{B}$ is denoted by $\langle \mathcal{A}, \mathcal{B} \rangle = \mathrm{vec}\,(\mathcal{B})^{\mathrm{H}} \mathrm{vec}\,(\mathcal{A})$. The weighted inner product is defined as $\langle \mathcal{A}, \mathcal{B} \rangle_{\mathcal{S}} = \mathrm{vec}\,(\mathcal{B})^{\mathrm{H}} \mathrm{diag}(\mathrm{vec}\,(\mathcal{S}))\mathrm{vec}\,(\mathcal{A})$ where weight tensor $\mathcal{S}$ has the same dimensions as $\mathcal{A}$ and $\mathcal{B}$. The Frobenius norm is denoted by $||\cdot||$. The real part of a complex number is written as $\mathrm{Re}\,(\cdot)$ and the expectation operator as $\mathbb{E}\,[\cdot]$. The permutation matrix $\mathbf{P}^{(n)}$ maps entry $(i_1, \ldots, i_{n-1}, i_n, i_{n+1}, \ldots, i_N)$ of a vectorized tensor to entry $(i_n, i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N)$. Details on the used notation can be found in [11].

1.2. Optimization framework. We discuss the expressions required to implement quasi-Newton and nonlinear least squares (NLS) algorithms to solve the optimization problem

$$\min_{\mathbf{z}} f = \frac{1}{2} ||F(\mathbf{z})||^2 . \tag{6}$$

All presented algorithms allow complex tensors and/or factors. We use the complex optimization framework from [36, 37] which implements a large number of qN and NLS algorithms including implementations of various line search, plane search and trust region strategies. This framework requires functions computing the objective function, the gradient and, in the case of NLS algorithms, the Gramian or Gramian-vector product. In the remainder of this subsection, we elaborate on a particular NLS algorithm called Gauss–Newton. A more detailed explanation of other qN and NLS algorithms and complex optimization can be found in [3, 27, 36, 38].

Unconstrained NLS algorithms attempt to find optimal values $\mathbf{z}^* \in \mathbb{K}^M$ using objective function (13). In iteration $k+1$, the residual $F(\mathbf{z})$ is linearized using the first-order Taylor expansion at the current iterate $\mathbf{z}_k$:

$$F(\mathbf{z}) \approx F(\mathbf{z}_k) + \left.\frac{\partial F}{\partial \mathbf{z}}\right|_{\mathbf{z}=\mathbf{z}_k} (\mathbf{z} - \mathbf{z}_k) \tag{7}$$

$$= F(\mathbf{z}_k) + \mathbf{J}\mathbf{p}, \tag{8}$$

in which $\mathbf{J}$ is the Jacobian of $F(\mathbf{z})$ and $\mathbf{p}$ the step direction. Substituting the linearization in objective function (6) leads to a quadratic model

$$\min_{\mathbf{p}} \frac{1}{2} F(\mathbf{z}_k)^{\mathrm{H}} F(\mathbf{z}_k) + F(\mathbf{z}_k)^{\mathrm{H}} \mathbf{J}\mathbf{p} + \frac{1}{2} \mathbf{p}^{\mathrm{H}} \mathbf{J}^{\mathrm{H}} \mathbf{J}\mathbf{p}. \tag{9}$$

The first-order optimality criterion leads to the minimizer $\$\mathbf{p}^*$ by solving

$$\mathbf{J}^{\mathrm{H}}\mathbf{J}\mathbf{p}^* = -\mathbf{J}^{\mathrm{H}} F(\mathbf{z}_k) \tag{10}$$

$$\mathbf{H}\mathbf{p}^* = -\bar{\mathbf{g}}. \tag{11}$$

This manuscript is for review purposes only.

The matrix $\mathbf{H}$ is the Gramian (of the Jacobian) and approximates the Hessian of f. The right-hand side is the negated and conjugated gradient $\bar{\mathbf{g}} = \left(\frac{\mathrm{d}f}{\mathrm{d}\mathbf{z}}\right)^{\mathrm{H}}$. Note that (11) only requires the derivative w.r.t. $\mathbf{z}$, even for complex variables $\mathbf{z}$ while f is non-analytic. As shown in [36], the gradient actually consists of the partial derivative w.r.t. $\mathbf{z}$ and $\bar{\mathbf{z}}$, keeping $\bar{\mathbf{z}}$ and $\mathbf{z}$ constant, respectively. Both partial derivatives of f are each other's complex conjugate and the Jacobian $\partial F/\partial\bar{\mathbf{z}} = 0$, as F is analytic in $\mathbf{z}$.

When the number of variables M is large, the computational cost of computing the (pseudo) inverse of $\mathbf{H} \in \mathbb{C}^{M\times M}$ can be a bottleneck. To reduce the complexity, (preconditioned) Conjugate Gradients (CG) can be used to solve (11) [27]. CG iteratively refines an initial guess $\mathbf{p}_0$ using only Gramian-vector products $\mathbf{G}\mathbf{p}_l$, $l = 0, \ldots, M-1$. In the case of CPD computations, [38] showed that preconditioned CG can effectively reduce the computational cost for large-scale systems.

Finally, the variables $\mathbf{z}$ are updated as

$$\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha_{k+1}\mathbf{p}, \tag{12}$$

in which α_{k+1} is computed using a line search algorithm. Other variants use plane search or trust-region methods to compute $\mathbf{z}_{k+1}$. For more information on different updating methods, we refer the interested reader to, e.g., [27]. In this paper, we use a dogleg trust region (DLTR) approach [27, 38].

It is clear that an NLS based algorithm requires a way of computing the objective function, the gradient and the Gramian or Gramian-vector products. In the latter case, an effective preconditioner reduces the computational cost of the CG algorithm. In section 2, we derive these expressions for the linearly constrained CPD of incomplete tensors. An effective preconditioner is derived in section 4.

2. CPD with linearly constrained factors for incomplete tensors. The proposed CPDLI algorithms compute the rank-R CPD of an Nth-order incomplete tensor $\mathcal{T} \in \mathbb{K}^{I_1\times\cdots\times I_N}$ with linear constraints on the factor matrices using an (inexact) Gauss–Newton algorithm with dogleg trust regions. Mathematically, this results in the following optimization problem:

$$\min_{\mathbf{C}^{(1)},\ldots,\mathbf{C}^{(N)}} \quad f = \frac{1}{2}\left|\left|\mathcal{S} * \left(\left[\!\left[\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}\right]\!\right] - \mathcal{T}\right)\right|\right|^2 \tag{13}$$

$$\text{subject to} \quad \mathbf{A}^{(n)} = \mathbf{B}^{(n)}\mathbf{C}^{(n)} \quad n = 1,\ldots,N. \tag{14}$$

An entry in the binary sampling tensor $\mathcal{S}$ is one if the corresponding entry in $\mathcal{T}$ is known and zero otherwise. The factor matrices $\mathbf{A}^{(n)} \in \mathbb{K}^{I_n\times R}$ are the product of known matrices $\mathbf{B}^{(n)} \in \mathbb{K}^{I_n\times D_n}$ and unknown coefficient matrices $\mathbf{C}^{(n)} \in \mathbb{K}^{D_n\times R}$, $n = 1,\ldots,N$.

Two variants of CPDLI are derived: the data-dependent algorithm uses the data in every iteration, while the data-independent variant computes a projected representation of the data prior to the optimization and uses this representation in every iteration. The derivations of all expressions are presented in section SM1.

2.1. A data-dependent algorithm: CPDLI DD. The CPDLI DD algorithm computes the CPD with linearly constrained factor matrices of an incomplete tensor with a computational complexity linear in the number of known entries. This section summarizes the elements needed for the algorithm outlined in Algorithm 1.

This manuscript is for review purposes only.

Algorithm 1 CPDLI DD using Gauss–Newton with dogleg trust region

Input: $\mathcal{S}$, $\mathcal{T}$, $\{\mathbf{B}^{(n)}\}_{n=1}^{N}$ and $\{\mathbf{C}^{(n)}\}_{n=1}^{N}$
Output: $\{\mathbf{C}^{(n)}\}_{n=1}^{N}$
while not converged **do**
  Compute gradient $\mathbf{g}$ using (15) (or (16)).
  Compute step $\mathbf{p}$ using either
  - $\mathbf{p} = -\mathbf{H}^{\dagger}\bar{\mathbf{g}}$ with $\mathbf{H}$ from (18), or
  - PCG of $\mathbf{H}\mathbf{p} = -\bar{\mathbf{g}}$ with Gramian-vector products $\mathbf{H}\mathbf{p}$ computed using (23), (25) and (26) and preconditioner (45).

  Update $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$, using DLTR from $\mathbf{p}$, $\mathbf{g}$ and function value (13).
end while

Objective function. Computing the function value is straightforward. After constructing the factor matrices $\mathbf{A}^{(n)} = \mathbf{B}^{(n)}\mathbf{C}^{(n)}$, $n = 1, \ldots, N$, the sum of the squared entries of the residual tensor $\mathcal{F} = \mathcal{S} * \left(\left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right]\!\right] - \mathcal{T}\right)$ is computed. For sparsely sampled tensor with $N_{\text{ke}} \ll I_1 I_2 \cdots I_N$, only the nonzero entries of $\mathcal{F}$ are constructed.

Gradient. The gradient $\mathbf{g}$ of (13) w.r.t. the (vectorized) optimization variables $\mathbf{C}^{(n)}$ is partitioned as $\mathbf{g} = \left[\operatorname{vec}\left(\mathbf{G}^{(1)}\right); \ldots; \operatorname{vec}\left(\mathbf{G}^{(N)}\right)\right]$ in which each subgradient $\mathbf{G}^{(n)}$ is given by

$$\mathbf{G}^{(n)} = \frac{\partial f}{\partial \mathbf{C}^{(n)}} = \mathbf{B}^{(n)^{\mathrm{T}}} \bar{\mathbf{F}}_{(n)} \left(\bigodot_{k \neq n} \mathbf{A}^{(k)} \right) \tag{15}$$

$$= \left(\bar{\mathcal{F}} \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}} \right)_{(n)} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)} \right), \tag{16}$$

which follows from the chain rule and from multilinear identities. Which one of the expressions (15) and (16) should be used depends on the order, rank and number of known entries in the tensor and D_n. Gradient expression (15) is computed using a sparse matricized tensor times Khatri–Rao product (MTKRPROD), which can be computed in $\mathcal{O}\left(2N^2 N_{\text{ke}} R\right)$ operations for the N subgradients $\mathbf{G}^{(n)}$. Gradient expression (16) computes a projection onto basis matrices $\mathbf{B}^{(n)}$, $n = 1, \ldots, N$, which is a series of N sparse tensor-matrix products, followed by a dense MTKRPROD. The former operation can also be seen as a matricized tensor times Kronecker product (MTKRONPROD). The overall complexity of (16) is therefore $\mathcal{O}\left(2N N_{\text{ke}} \prod_n D_n\right)$ for the (sparse) MTKRONPROD and $\mathcal{O}\left(2NR \prod_n D_n\right)$ for the dense MTKRPROD. Both MTKRPROD and MTKRONPROD are standard operations in tensor computations. Efficient implementations for dense and sparse tensors are described in, e.g., [5, 19, 31, 35, 41].

Gramian of the Jacobian. Finally, we compute the Gramian $\mathbf{H} = \mathbf{J}^{\mathrm{H}}\mathbf{J}$. Similar to the gradient, the Gramian is partitioned into a grid of $N \times N$ blocks $\mathbf{H}^{(m,n)}$, $m, n = 1, \ldots, N$, corresponding to the approximation of the second-order derivative w.r.t. $\mathbf{C}^{(m)}$ and $\mathbf{C}^{(n)}$. Each block is computed from the explicitly constructed Jacobians $\mathbf{J}^{(n)}$:

$$\mathbf{J}^{(n)} = \frac{\partial}{\partial \operatorname{vec}\left(\mathbf{C}^{(n)}\right)} \operatorname{vec}(\mathcal{F}) = \mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]} \qquad \text{in which} \qquad \mathbf{V}^{[n]} = \mathop{\ast}_{k \neq n} \mathbf{A}^{[n]}. \tag{17}$$

This manuscript is for review purposes only.

The Jacobian is therefore a dense matrix of size $N_{\text{ke}} \times D_n R$ and the Gramian blocks $\mathbf{H}^{(m,n)}$ of size $D_m R \times D_n R$ are given by

$$\mathbf{H}^{(m,n)} = \left(\mathbf{V}^{[m]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)^{\mathrm{H}} \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right). \tag{18}$$

The data-independent algorithm CPDLI DI exploits the remaining structure in $\mathbf{V}^{[n]}$.

To reduce the cost of computing the pseudoinverse of $\mathbf{H}$, CG is used when dealing with many variables. Each CG iteration requires the matrix-vector product $\mathbf{y} = \mathbf{H}\mathbf{x}$. A partitioning similar to the gradient and Gramian is used for the vectors $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{x} = \left[\operatorname{vec}\left(\mathbf{X}^{(1)}\right); \ldots; \operatorname{vec}\left(\mathbf{X}^{(N)}\right)\right], \tag{19}$$

$$\mathbf{y} = \left[\operatorname{vec}\left(\mathbf{Y}^{(1)}\right); \ldots; \operatorname{vec}\left(\mathbf{Y}^{(N)}\right)\right]. \tag{20}$$

The part of the product $\mathbf{y} = \mathbf{H}\mathbf{x}$ corresponding to $\mathbf{Y}^{(m)}$ is then computed as

$$\operatorname{vec}\left(\mathbf{Y}^{(m)}\right) = \mathbf{J}^{(m)^{\mathrm{H}}} \begin{bmatrix}\mathbf{J}^{(1)} & \cdots & \mathbf{J}^{(N)}\end{bmatrix} \left[\operatorname{vec}\left(\mathbf{X}^{(1)}\right); \ldots; \operatorname{vec}\left(\mathbf{X}^{(N)}\right)\right] \tag{21}$$

$$= \mathbf{J}^{(m)^{\mathrm{H}}} \sum_{n=1}^{N} \mathbf{J}^{(n)} \operatorname{vec}\left(\mathbf{X}^{(n)}\right). \tag{22}$$

Using multilinear identities, the definition of the Jacobian (17) and the auxiliary vector $\mathbf{z}^{(n)}$, we then have

$$\mathbf{z}^{(n)} = \mathbf{J}^{(n)} \operatorname{vec}\left(\mathbf{X}^{(n)}\right) = \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]} \mathbf{X}^{(n)}\right) \mathbf{1} \tag{23}$$

$$\operatorname{vec}\left(\mathbf{Y}^{(m)}\right) = \left(\mathbf{V}^{[m]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)^{\mathrm{H}} \sum_{n=1}^{N} \mathbf{z}^{(n)}. \tag{24}$$

The matrix $\mathbf{B}^{[n]}\mathbf{X}^{(n)}$ can contain many identical rows as $\mathbf{B}^{[n]}$ is the extended form of $\mathbf{B}^{(n)}$. The computational cost can therefore be reduced by extending the product $\mathbf{B}^{(n)}\mathbf{X}^{(n)}$ instead of $\mathbf{B}^{(n)}$. The product $\left(\mathbf{V}^{[m]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)^{\mathrm{H}} \sum_{n=1}^{N} \mathbf{z}^{(n)}$ can be computed efficiently using accumulation. Let $\mathbf{z} = \sum_{n=1}^{N} \mathbf{z}^{(n)}$ and define the ith row of $\mathbf{Q}^{(m)} \in \mathbb{K}^{I_m \times R}$ by

$$\mathbf{Q}^{(m)}(i,:) = \sum_{k \in \pi^{(m)}} z_k \bar{\mathbf{V}}^{[m]}(k,:) \quad \text{with} \quad \pi^{(m)} = \left\{ k \,|\, \mathbf{i}^{(m)}(k) = i \right\}. \tag{25}$$

The cost of the row-wise Khatri–Rao product in (24) is avoided by computing

$$\mathbf{Y}^{(m)} = \mathbf{B}^{(m)^{\mathrm{H}}} \mathbf{Q}^{(m)}. \tag{26}$$

2.2. Removing data dependence: CPDLI DI. Each iteration, the CPDLI DD algorithm has a complexity dependent on the number of known entries N_{ke}. In this section, we show how the structure resulting from the linear constraints allows the per-iteration complexity to be independent of the number of known entries by computing a data-dependent matrix $\mathbf{D}$ prior to the optimization process. This matrix $\mathbf{D}$ is defined as the inner product

$$\mathbf{D} = \left(\bigodot_{n}^{\mathrm{T}} \mathbf{B}^{[n]}\right)^{\mathrm{H}} \left(\bigodot_{n}^{\mathrm{T}} \mathbf{B}^{[n]}\right), \tag{27}$$

This manuscript is for review purposes only.

which only depends on the positions of the known entries and the given matrices $\mathbf{B}^{(n)}$. Blocking is used to lower the intermediate memory requirements for expanding $\odot_n^{\mathrm{T}} \mathbf{B}^{[n]}$. As the size of $\mathbf{D}$ is $\prod_n D_n \times \prod_n D_n$, it is clear that this approach is limited to tensors of low order N and small D_n as the number of entries in $\mathbf{D}$ scales as D^{2N} assuming $D_n = D$ for $n = 1, \ldots, N$.

In the remainder of this section, we show how $\mathbf{D}$ can be used to reduce the computational cost of the ingredients in the CPDLI algorithm. Algorithm 2 gives an overview of CPDLI DI.

Algorithm 2 CPDLI DI using Gauss–Newton with dogleg trust region

1: **Input:** $\mathcal{S}$, $\mathcal{T}$, $\left\{\mathbf{B}^{(n)}\right\}_{n=1}^{N}$ and $\left\{\mathbf{C}^{(n)}\right\}_{n=1}^{N}$
2: **Output:** $\left\{\mathbf{C}^{(n)}\right\}_{n=1}^{N}$
3: Compute $\mathbf{D}$ from (27), $\mathcal{T}_B$ from (30), and $||\mathcal{S} * \mathcal{T}||^2$.
4: **while** not converged **do**
5: Compute gradient $\mathbf{g}$ using (36) and (34).
6: Compute step $\mathbf{p}$ using either
- $\mathbf{p} = -\mathbf{H}^{\dagger}\bar{\mathbf{g}}$ with $\mathbf{H}$ from (37), or
- PCG of $\mathbf{Hp} = -\bar{\mathbf{g}}$ with Gramian-vector products $\mathbf{Hp}$ computed using (40) and (41), and preconditioner (45).

7: Update $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$, using DLTR from $\mathbf{p}$, $\mathbf{g}$ and function value (29).
8: **end while**

Objective function. The objective function (13) can be expanded as follows

(28)
$$f = \frac{1}{2}\left|\left|\mathcal{S} * \mathcal{T}\right|\right|^2 - \mathrm{Re}\left(\left\langle \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right]\!\right], \mathcal{T}\right\rangle_{\mathcal{S}}\right) + \frac{1}{2}\left|\left|\mathcal{S} * \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right]\!\right]\right|\right|^2.$$

Let $\hat{\mathbf{t}}$ be the vector containing all values of known entries of $\mathcal{T}$. The norm $||\mathcal{S} * \mathcal{T}||^2$ is then computed prior to the optimization process as $\hat{\mathbf{t}}^{\mathrm{H}}\hat{\mathbf{t}}$. In subsection SM1.3 we show that f can be computed as

(29)
$$f = \frac{1}{2}\hat{\mathbf{t}}^{\mathrm{H}}\hat{\mathbf{t}} + \mathrm{Re}\left(\left(\frac{1}{2}\mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right)^{\mathrm{H}}\mathbf{D} - \hat{\mathbf{t}}_B^{\mathrm{H}}\right)\mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right)\right).$$

The vector $\hat{\mathbf{t}}_B$ is computed a priori and is defined as

(30)
$$\hat{\mathbf{t}}_B = \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]}\right)^{\mathrm{H}} \hat{\mathbf{t}}.$$

While using (29) is computationally more efficient, it is numerically less accurate. Consider only the known entries $\hat{\mathbf{t}}$ and assume an approximation up to an error vector $\boldsymbol{\delta}$ with $||\boldsymbol{\delta}|| = \epsilon\left|\left|\hat{\mathbf{t}}\right|\right|$ has been found. The data-dependent objective function is then equivalent to

(31)
$$2f = \left|\left|(\hat{\mathbf{t}} + \boldsymbol{\delta}) - \hat{\mathbf{t}}\right|\right|^2 = ||\boldsymbol{\delta}||^2 = \mathcal{O}\left(\epsilon^2\right).$$

For the data-independent objective function we have

(32)
$$2f = \left|\left|(\hat{\mathbf{t}} + \boldsymbol{\delta})\right|\right|^2 - 2\mathrm{Re}\left((\hat{\mathbf{t}} + \boldsymbol{\delta})^{\mathrm{H}}\hat{\mathbf{t}}\right) + \left|\left|\hat{\mathbf{t}}\right|\right|^2$$

(33)
$$= \mathrm{Re}\left(\left(\left|\left|\hat{\mathbf{t}}\right|\right|^2 + 2\hat{\mathbf{t}}^{\mathrm{H}}\boldsymbol{\delta} + ||\boldsymbol{\delta}||^2\right) - 2\left(\left|\left|\hat{\mathbf{t}}\right|\right|^2 + \hat{\mathbf{t}}^{\mathrm{H}}\boldsymbol{\delta}\right) + \left|\left|\hat{\mathbf{t}}\right|\right|^2\right).$$

This manuscript is for review purposes only.

If $\epsilon < \sqrt{\epsilon_{\text{mach}}}$ with ϵ_{mach} the machine precision, $||\boldsymbol{\delta}||^2$ is numerically zero, and $2f = 0$ or $2f = \pm\mathcal{O}(\epsilon_{\text{mach}})$ instead of $\mathcal{O}(\epsilon_{\text{mach}}^2)$. This limits the accuracy to $\mathcal{O}(\sqrt{\epsilon_{\text{mach}}})$. In applications, however, other factors often put more stringent restrictions to the accuracy. Due to the signal-to-noise ratio, for example, only a few accurate digits are required in signal processing, or, in machine learning and data analysis, a limited precision is expected because of model errors and regularization terms. In both examples, the limited accuracy caused by using (29) is therefore not a problem. If, in a specific application, a solution accurate up to machine precision is required, the solution found by the data-independent algorithm can be refined using a few iterations of the data-dependent algorithm.

Gradient. Similar to the derivation of the objective function, the residual tensor $\mathcal{F}$ in (16) is separated in a data part and a decomposition part. As shown in subsection SM1.3, the gradient can be written as

$$\mathbf{G}^{(n)} = \left(\bar{\mathcal{Z}} - \bar{\mathcal{T}}_B\right)_{(n)} \left(\bigodot_{k\neq n} \mathbf{C}^{(k)}\right), \tag{34}$$

which is an MTKRPROD involving the following compressed, dense tensors. The values of compressed data representation $\bar{\mathcal{T}}_B = \left(\mathcal{S} * \bar{\mathcal{T}}\right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}}$ are $\hat{\mathbf{t}}_B$ from (30). The conjugated auxiliary tensor $\bar{\mathcal{Z}}$ has size $D_1 \times \cdots \times D_N$ and is computed as

$$\bar{\mathcal{Z}} = \left(\mathcal{S} * \left[\!\left[\bar{\mathbf{A}}^{(1)}, \ldots, \bar{\mathbf{A}}^{(N)}\right]\!\right]\right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}} \tag{35}$$

$$= \text{reshape}\left(\bar{\mathbf{D}}\text{vec}\left(\left[\!\left[\bar{\mathbf{C}}^{(1)}, \ldots, \bar{\mathbf{C}}^{(N)}\right]\!\right]\right), [D_1, \ldots, D_N]\right). \tag{36}$$

Gramian of the Jacobian. The complexity of computing the data-dependent Gramian in (18) can be reduced by removing the dependency on $\mathbf{B}^{[n]}$. In subsection SM1.3, we show that

$$\mathbf{H}^{(m,n)} = \left(\bigodot_{k\neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)} \mathbf{D} \mathbf{P}^{(n)^{\mathrm{T}}} \left(\bigodot_{k\neq n} \mathbf{C}^{(k)} \otimes \mathbf{I}\right). \tag{37}$$

This is a variation on the MTKRPROD called the Khatri–Rao times matricized tensor times Khatri–Rao product (KRMTKRPROD). While an MTKRPROD can be computed in memory and cache efficient way, i.e., without permuting entries of the tensor [31, 41], permutations are required for KRMTKRPROD. Using similar ideas as in [31, 41], the performance penalty endured by permuting the tensor can be minimized using appropriate left-to-right and right-to-left contractions. The exact description of this algorithm is out of the scope of this paper. An implementation of KRMTKRPROD will be made available in Tensorlab [44].

Similar to the data-dependent algorithm, CG can be used for large-scale problems, hence, requiring a Gramian-vector product. From (37) we have for a single Gramian block $\mathbf{H}^{(m,n)}$

$$\mathbf{H}^{(m,n)}\text{vec}\left(\mathbf{X}^{(n)}\right) = \left(\bigodot_{k\neq m} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)}\text{vec}\left(\mathcal{Z}^{(n)}\right) \tag{38}$$

in which the vectorized tensor $\mathcal{Z}^{(n)}$ of size $D_1 \times \cdots \times D_N$ is given by

$$\text{vec}\left(\mathcal{Z}^{(n)}\right) = \mathbf{D}\text{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{X}^{(n)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right) \tag{39}$$

$$\text{vec}\left(\mathcal{Z}\right) = \sum_{n=1}^{N} \text{vec}\left(\mathcal{Z}^{(n)}\right) = \mathbf{D} \sum_{n=1}^{N} \text{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{X}^{(n)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right). \tag{40}$$

This manuscript is for review purposes only.

The matricized result $\mathbf{Y}^{(m)}$ is then computed using an MTKRPROD:

$$\mathbf{Y}^{(m)} = \sum_{n=1}^{N} \mathbf{H}^{(m,n)} \text{vec}\left(\mathbf{X}^{(n)}\right) = \mathbf{Z}_{(m)} \left(\bigodot_{k \neq m} \bar{\mathbf{C}}^{(k)} \right). \tag{41}$$

2.3. Complexity. To conclude this section, the per-iteration complexity of both variants is investigated and compared. For simplicity of notation, assume $\mathcal{T}$ is a cubical tensor, i.e., $I_n = I$ and matrices $\mathbf{B}^{(n)}$ have the same number of columns, i.e., $D_n = D$, for $n = 1, \ldots, N$. A summary of the per-iteration complexity for CPDLI DD and CPDLI DI is given in Table 1 and Table 2, respectively. For large-scale problems, i.e., problems with a large number of variables, the choice between CPDLI DD and CPDLI DI depends on N_{ke} and D: the former's overall complexity per iteration scales with $\mathcal{O}\left(N^2 R D N_{\text{ke}}\right)$ compared to $\mathcal{O}\left(D^{2N}\right)$ for the latter. Therefore, the CPDLI DI algorithm is favored for low-order tensors with D small and relatively many known entries.

TABLE 1

The per-iteration complexity of the CPDLI DD algorithm is dominated by Gramian operations. The overall complexity depends linearly on the number of known entries N_{ke}. A trust region method is used to determine the update of the variables, requiring it_{TR} additional function evaluations.

	Calls per iteration	Complexity
Function value	$1 + \text{it}_{\text{TR}}$	$\mathcal{O}\left(N_{\text{ke}} N R\right)$
Gradient (sparse)	1	$\mathcal{O}\left(2N^2 R N_{\text{ke}}\right)$
Gradient (dense)	1	$\mathcal{O}\left(2N D^N (R + N_{\text{ke}})\right)$
Gramian	1	$\mathcal{O}\left(N^2 D^2 R^2 N_{\text{ke}} + \frac{8}{3} N^3 D^3 R^3\right)$
Gramian-vector	it_{CG}	$\mathcal{O}\left(2N^2 D R N_{\text{ke}}\right)$
Total	Small-scale	$\mathcal{O}\left(N^2 D^2 R^2 N_{\text{ke}} + \frac{8}{3} N^3 D^3 R^3\right)$
	Large-scale	$\mathcal{O}\left(2\text{it}_{\text{CG}} N_{\text{ke}} N^2 R D\right)$

TABLE 2

The per-iteration complexity of the CPDLI DI algorithm is dominated by Gramian operations involving $\mathbf{D}$. The number of known entries N_{ke} appears only in the preparation complexity which is independent of the number of iterations. A trust region method is used to determine the update of the variables, requiring it_{TR} additional function evaluations.

	Calls per iteration	Complexity
Preparation		$\mathcal{O}\left(2N D^{2N} N_{\text{ke}}\right)$
Function value	$1 + \text{it}_{\text{TR}}$	$\mathcal{O}\left(2D^{2N}\right)$
Gradient	1	$\mathcal{O}\left(2D^{2N} + 3D^N R N\right)$
Gramian	1	$\mathcal{O}\left(N^2 R D^{2N} + \frac{8}{3} N^3 D^3 R^3\right)$
Gramian-vector	it_{CG}	$\mathcal{O}\left(2D^{2N} + N^2 D^N R\right)$
Total	Small-scale	$\mathcal{O}\left(N^2 R D^{2N} + \frac{8}{3} N^3 D^3 R^3\right)$
	Large-scale	$\mathcal{O}\left(2(\text{it}_{\text{CG}} + 2) D^{2N}\right)$

3. Unconstrained CPD for incomplete tensors. How to deal with missing entries when computing a CPD has been an important topic for many years and many algorithms have been proposed. We present a new algorithm called CPDI, which is a

This manuscript is for review purposes only.

special case of CPDLI DD. In contrast to imputation based methods [17, 30, 40], the computational per-iteration complexity is linear in N_{ke}, and in contrast to other direct minimization techniques [2, 20, 39], second-order convergence can be achieved by using the exact Gramian. By combining fast convergence with efficient Gramian-vector products, CPDI outperforms state-of-the-art methods when few entries are known or the position of the missing entries follows a pattern, as shown in the experiments in subsection 5.1. These cases are particularly interesting when dealing with large-scale problems as illustrated in [43].

The unconstrained CPD of an incomplete tensor can be seen as a special case of the CPDLI algorithm in which the known matrices are identity matrices, i.e., $\mathbf{B}^{(n)} = \mathbf{I}_{I_n}$, for $n = 1, \ldots, N$. As $D_n = I_n$, the data independent variant CPDLI DI is usually not attractive as I_n is often large. Therefore, the CPDI algorithm is based on CPDLI DD. The expressions for the function value and the gradient can be derived trivially from CPDLI DI and result in the same expressions as in [2] which uses a quasi-Newton approach and [39, 40] which use a Levenberg–Marquardt or Gauss–Newton approach (as in this paper). The main difference with [40] is that an inexact algorithm is used, meaning CG is used instead of inverting the Gramian. While the CPDI algorithm proposed here uses the exact Gramian, [39] computes the Gramian for a full tensor and then scales the result by the fraction of known entries. As this scaling is countered by a longer step length in the line search or trust region step, the incompleteness is in fact ignored.

In the experiments in subsection 5.1, we illustrate that the use of the exact Gramian is beneficial if the known entries are not scattered uniformly at random across the tensor, or if high accuracy is required while few entries are available. As $\mathbf{B}^{(n)}$ is an identity matrix in the case of CPDI, the Jacobian (Equation (17)) is sparse. Exploiting this sparsity using sparse matrix-matrix or matrix-vector products reduces the complexity to $\mathcal{O}\left(N^2R^2N_{\text{ke}} + \frac{8}{3}N^3D^3R^3\right)$ for the small-scale version (first term is a factor D^2 smaller compared to CPDLI DD) and to $\mathcal{O}\left(2\text{it}_{\text{CG}}N^2RN_{\text{ke}}\right)$ for the large-scale version (a factor D smaller).

4. Preconditioner. For relatively large-scale tensors, the cost of inverting the Gramian matrix in (11) can be alleviated by using preconditioned CG which only requires matrix-vector products [27]. In PCG, the system

$$\mathbf{M}^{-1}\mathbf{H}\mathbf{p} = -\mathbf{M}^{-1}\bar{\mathbf{g}} \tag{42}$$

is solved instead, in which the preconditioner $\mathbf{M}$ is a symmetric positive definite matrix. As the convergence of CG depends on the clustering of the eigenvalues of $\mathbf{H}$, the number of CG iterations can be reduced if the eigenvalues of $\mathbf{M}^{-1}\mathbf{H}$ are more clustered. At the same time, applying the inverse of the preconditioner should be computationally cheap.

For CPD algorithms, [38] proposes a block-Jacobi type preconditioner

$$\mathbf{M}_{\text{FULL}} = \text{blkdiag}\left(\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(N)}\right), \tag{43}$$

where $\mathbf{M}^{(n)} = \mathbf{W}^{(n)} \otimes \mathbf{I}$ with $\mathbf{W}^{(n)} = {\mathbf{V}^{(n)}}^{\text{H}}\mathbf{V}^{(n)}$ which is computed efficiently as $*_{k\neq n}\,{\mathbf{A}^{(n)}}^{\text{H}}\mathbf{A}^{(n)}$. In [39], an extension to incomplete tensors uses a scaled preconditioner

$$\mathbf{M}_{\text{INC,SCALED}} = \rho\mathbf{M}_{\text{FULL}}. \tag{44}$$

This manuscript is for review purposes only.

The fraction $\rho = N_{\text{ke}} / \prod_n I_n$ is simply the fraction of known entries. This is a reasonable approximation for tensors with known entries scattered randomly across the tensor and with entries of similar magnitudes.

Here, we propose a similar block-Jacobi type preconditioner based on the expected value of the diagonal Gramian blocks $\mathbf{H}^{(n,n)}$. First, we derive a preconditioner for the CPDI algorithm, which we then extend to the CPDLI algorithm by incorporating linear constraints.

To precondition the linear systems arising when decomposing incomplete tensors using CPDI, we propose the block-Jacobi preconditioner $\mathbf{M}_{\text{INC,STAT}}$ with diagonal blocks $\mathbf{M}^{(n)}_{\text{INC,STAT}}$. The exact expression for the block diagonal of $\mathbf{H}^{(n)} = \mathbf{J}^{(n)\text{H}}\mathbf{J}^{(n)}$ would require the inversion of a $I_n R \times I_n R$ matrix, which is often too expensive for large-scale problems. Instead, the expected value of the block $\mathbf{H}^{(n,n)}$ is used. More concretely, consider the i_nth slice of order $N-1$ and assume there are $Q^{(n)}_{i_n}$ known entries in this slice. Instead of taking the exact positions, or distribution, of the known entries into account, we assume that all possible distributions of exactly $Q^{(n)}_{i_n}$ known entries in this slice are equally likely. By taking the expected value over all distributions, only the fraction of known entries $f^{(n)}_{i_n} = Q^{(n)}_{i_n} / \prod_{k \neq n} I_k$ is required to approximate the blocks on the diagonal by

$$\mathbf{M}^{(n)}_{\text{INC,STAT}} = \mathbb{E}\left[\mathbf{H}^{(n,n)}\right] = \mathbf{W}^{(n)} \otimes \operatorname{diag}\left(\mathbf{f}^{(n)}\right) \tag{45}$$

as shown in section SM2. This expression reduces to the result from [39] if every slice contains the same number of known entries.

For both CPDLI algorithms, the preconditioner is extended to the case with linear constraints on the factor matrices, resulting in the expression

$$\mathbf{M}^{(n)}_{\text{LINC,STAT}} = \mathbb{E}\left[\mathbf{H}^{(n,n)}\right] = \mathbf{W}^{(n)} \otimes \left(\mathbf{B}^{(n)\text{H}} \operatorname{diag}\left(\mathbf{f}^{(n)}\right) \mathbf{B}^{(n)}\right). \tag{46}$$

Even though this preconditioner only approximates the exact Gramian blocks, which take the exact position of the known entries into account, the experiments in subsection 5.2 show that the number of CG iterations is effectively reduced.

5. Experiments. The performance of the CPDI method and the effectiveness of the proposed preconditioner are shown with two experiments in this section. Section 6 illustrates the benefits of linear constraints and second-order information on a materials science dataset using CPDLI. Details about the parameters and implementation used for the different algorithms are reported in Appendix A to allow reproducibility.

5.1. CPD of incomplete tensors. The effectiveness of using second-order information to reduce the computational cost of incomplete tensor decompositions is illustrated for two scenarios: randomly missing entries and structured missing entries. The proposed CPDI algorithm is compared to another Gauss–Newton type algorithm CPD [38, 39], and to two quasi-Newton algorithms based on nonlinear conjugate gradients (NCG): CPD MINF [38, 39] and CP WOPT [2]. The first three algorithms are implemented in Tensorlab [44]. The Tensor Toolbox [6] implementation is used for CP WOPT. For all algorithms, we compute a low accuracy solution ($E_{\text{CPD}} \approx 10^{-5}$) and a high accuracy solution ($E_{\text{CPD}} \leq 10^{-12}$), in which

$$E_{\text{CPD}} = \max_{n=1,\ldots,N} \frac{\left|\left|\mathbf{A}^{(n)} - \hat{\mathbf{A}}^{(n)}\right|\right|}{\left|\left|\mathbf{A}^{(n)}\right|\right|} \tag{47}$$

This manuscript is for review purposes only.

with $\mathbf{A}^{(n)}$ the (known) exact factor matrices and $\hat{\mathbf{A}}^{(n)}$ the estimated factor matrices after resolving the scaling and permutation indeterminacies. All experiments are repeated 25 times using different tensors and different initial guesses.

First, let $\mathcal{T} \in \mathbb{R}^{1000\times1000\times1000}$ be a rank-10 tensor generated using random factor matrices $\mathbf{A}^{(n)}$ drawn from a uniform distribution $\mathcal{U}(0,1)$. From this tensor, entries are sampled uniformly and randomly. Figure 1 shows the computation time when 300 000 entries and 90 000 entries are sampled, corresponding to 0.03% and 0.009% of all entries, respectively, or alternatively, 10 and 3 entries on average per variable, respectively. Both in time and number of iterations, the Gauss–Newton based methods CPDI and CPD outperform the NCG based methods. Hence the higher per-iteration complexity is countered by the reduction in number of entries. While CPD outperforms CPDI when many entries are known or when a low accuracy is needed, the number of iterations is higher for CPD compared to CPDI. For high accuracy and few known entries, CPDI benefits from its improved convergence properties. CP WOPT fails to find a high-accuracy solution due to its implementation of the objective function.

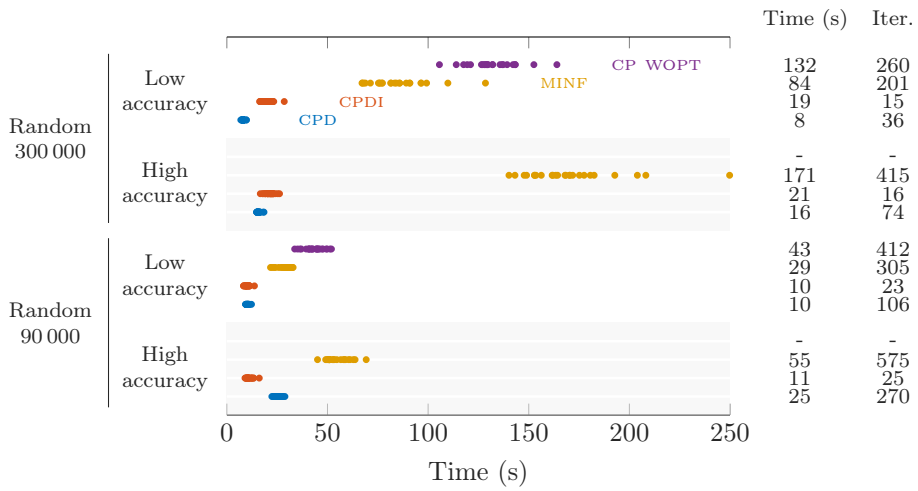


FIG. 1. *The Gauss–Newton type algorithms* CPD *and* CPDI *outperform the first-order NCG type algorithms as the higher per-iteration cost is countered by a significantly lower number of required iterations. When many entries are available, the lower complexity of* CPD *compared to* CPDI *is an advantage. For higher accuracies or fewer entries,* CPDI *is competitive because of its faster convergence rate.* CP WOPT *fails to find a highly accurate solution. The reported timings and number of iterations are averages over 25 experiments.*

The second scenario starts from a dense rank-10 tensor $\mathcal{T} \in \mathbb{R}^{100\times100\times100}$ constructed using random factor matrices with entries drawn from $\mathcal{U}(0,1)$. A structured missing data pattern is constructed as follows. From $\mathcal{T}$, 95 mode-1 slices are selected uniformly at random and from each of the selected slices 99% of the entries are discarded at random. The remaining 5 mode-1 slices are dense. While the fraction of known entries is relatively high (5.95%), the pattern prevents CPD from finding a solution. Figure 2 shows the computational cost is again reduced significantly because of the improved convergence properties for CPDI.

5.2. Preconditioner. CPDI and CPDLI have a relatively high per-iteration complexity while only a few iterations are necessary. Reducing the complexity or the number of iterations has therefore a large impact on the overall computation time.

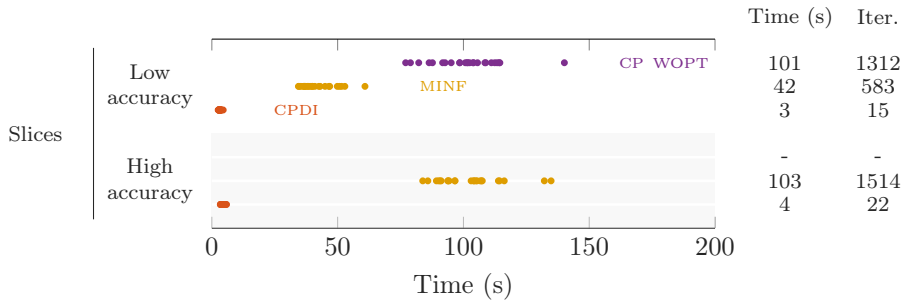


FIG. 2. *When missing entries follow a structured pattern,* CPDI *needs only few iterations to find a solution.* CPD *fails to find a solution in a reasonable amount of time for both accuracy levels and is therefore omitted.* CP WOPT *fails to find a highly accurate solution. The reported timings and number of iterations are averages over 25 experiments.*

Here, we show the effectiveness of the preconditioners proposed in section 4 in reducing the number of CG iterations and therefore in reducing the per-iteration complexity of the algorithm.

Consider the following experiment. A random tensor $\mathcal{T} \in \mathbb{R}^{100\times100\times100}$ is constructed using linearly constrained factor matrices with random basis matrices $\mathbf{B}^{(n)} \in \mathbb{R}^{100\times10}$ and coefficient matrices $\mathbf{C}^{(n)} \in \mathbb{R}^{10\times5}$. The entries of both $\mathbf{B}^{(n)}$ and $\mathbf{C}^{(n)}$ are drawn from a uniform distribution $\mathcal{U}(0,1)$ with $n = 1, 2, 3$. To simulate a typical iteration, the Gramian $\mathbf{H}$ and the gradient $\mathbf{g}$ are computed for random coefficients $\hat{\mathbf{C}}^{(n)}$ and the system $\mathbf{Hx} = -\bar{\mathbf{g}}$ is solved using PCG until convergence, i.e., until the relative residual is smaller than 10^{-6}. The number of CG iterations is monitored when using no preconditioner, the incompleteness agnostic preconditioner (44) from [39] and the proposed preconditioner (46). As in subsection 5.1, two scenarios are considered: 0.1% randomly sampled entries, and structured missing entries constructed by removing 99% of the entries of 95 randomly selected mode-1 slices. Table 3 shows the results averaged over 25 tensors and 20 random coefficient matrices $\hat{\mathbf{C}}^{(n)}$. While both preconditioners have a similar performance for randomly missing entries, the proposed preconditioner outperforms the scaled preconditioner for structured missing entries. Both preconditioners reduce the number of CG iterations significantly compared to the non-preconditioned case.

TABLE 3

The proposed PC and the incompleteness agnostic PC from [39] reduce the number of CG iterations significantly compared to the non-preconditioned case. When missing entries are scattered randomly across the tensor, both preconditioners have a similar performance, as expected. When the missing entries are structured, the proposed PC reduces the number of iterations by 14 on average. The average (standard deviation) is computed over 25 tensors for 20 sets of variables, each.

Scenario	No PC	Agnostic PC (44)	Proposed PC (46)
Uniformly at random	115.8 (19.7)	45.5 (9.3)	45.1 (9.2)
Structured missing	124.6 (18.4)	53.3 (10.0)	39.3 (7.3)

6. Materials science application. Designing new materials often requires extensive numerical simulations depending on material parameters. For example, the melting temperature of an alloy depends, among others, on the concentrations of its

constituent materials [43]. As a material property can depend on a large number of variables, e.g., concentrations, temperature and pressure, the corresponding tensors with measured or computed values often have a high order. The curse of dimensionality therefore prohibits measuring all possible combinations. In [43], the CPD of an incomplete tensor allowed modeling a ninth-order tensor using only $10^{-11}\%$ of the data. Here, we show that using incomplete tensors and a CPD with linearly constrained factors allows the Gibbs free energy to be modeled using very few data points, while improving the accuracy w.r.t. the unconstrained problem.

Concretely, the Gibbs free energy for an alloy with silver, copper, nickel and tin as constituent materials is modeled in its liquid phase, at a temperature $T = 1000\,\mathrm{K}$. All parameters other than the concentrations (expressed in mole fraction) of silver (c_1), copper (c_2) and nickel (c_3) are kept constant. The concentration of tin (c_4) is a dependent variable as all concentrations must sum to 1, hence $c_4 = 1 - c_1 - c_2 - c_3$. Following the Calphad model [24, 26], the Gibbs free energy can be modeled as

$$G(c_1, c_2, c_3) = G_p(c_1, c_2, c_3) + G_{\log}(c_1, c_2, c_3) \tag{48}$$

$$G_{\log}(c_1, c_2, c_3) = \sum_{n=1}^{4} UTc_n \log_{10} c_n, \tag{49}$$

in which U is the universal gas constant. Thermo-Calc software [1] with the COST531-v3 database [25] computes G in chosen concentrations c_1, c_2 and c_3. By discretizing the concentrations, an incomplete third-order tensor $\mathcal{G}$ is obtained with the concentrations as its modes. As the logarithmic terms in $\mathcal{G}_{\log}$ depend solely on known values, we subtract these terms from $\mathcal{G}$ and model only $\mathcal{G}_p$ as a multivariate polynomial. Note that $\mathcal{G}$ is necessarily incomplete: if sampled densely, i.e., if all feasible entries are sampled on the chosen discretized grid of concentrations, all entries lie inside a pyramid (Figure 3) as $c_1 + c_2 + c_3 > 1$ is physically unfeasible. Consequently, variables at the vertices of the pyramid are estimated using only a single data point.

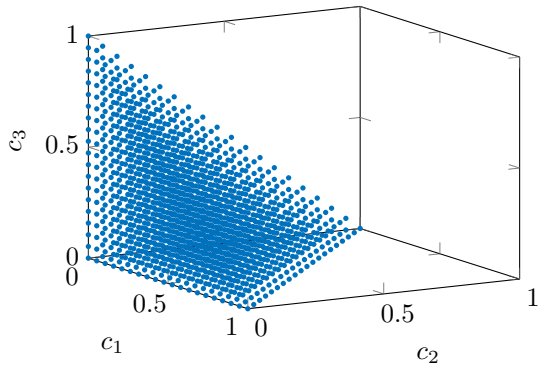


FIG. 3. *A 'dense' sampling scheme for $\mathcal{G}$ results in pyramidal structure as $c_1 + c_2 + c_3 \leq 1$.*

Five algorithms are compared in this section:

- the data-dependent (CPDLI DD) and data-independent (CPDLI DI) algorithms proposed in this paper,
- multivariate regression using alternating least squares (MVR ALS) [7],
- structured data fusion (SDF) using `struct_matvec` to implement the constraints [39] and the `cpdi` factorization which implements CPDI as proposed here [44], and

This manuscript is for review purposes only.

- CPDI, i.e., without linear constraints.

The precise parameters are outlined in Appendix A. All tested algorithms behave differently and achieve a different accuracy, defined as the median relative error over all feasible entries of the discretized Gibbs free energy tensor:

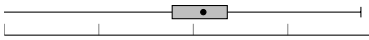
$$E = \text{median} \left| \frac{\mathcal{G}_p(i,j,k) - \sum_{r=1}^{R} a_{ir}^{(1)} a_{jr}^{(2)} a_{kr}^{(3)}}{\mathcal{G}_p(i,j,k)} \right|. \tag{50}$$

As the accuracy of the final solutions computed by the different algorithms can differ, accuracy thresholds are used to allow for a fair comparison of timing results. Five levels with accuracy thresholds are set: $10^{-3}, 5 \cdot 10^{-4}, 10^{-4}, 5 \cdot 10^{-5}, 10^{-5}$. The algorithm proceeds to the next level when the median relative error E is below the current level's threshold and the (cumulative) time for each level is recorded. A maximum computation time of three minutes is set for each level. This is repeated for 50 initial guesses for $\mathbf{C}^{(n)}$, $n = 1, 2, 3$.

In the first experiment, the entries of $\mathcal{G}_p \in \mathbb{R}^{99 \times 99 \times 99}$ are sampled according to a dense, pyramidal grid with $c_n = 0.005, 0.015, 0.025, \ldots, 0.985$ for $n = 1, 2, 3$ which amounts to 166 338 known entries (17% of all entries). A CPD with $R = 5$ terms is computed from $\mathcal{G}_p$ with polynomial constraints on the factor matrices, i.e., each factor vector is a polynomial of maximal degree $d = 4$, hence $D_n = 5$, $n = 1, 2, 3$. A monomial basis is used. Figure 4 shows that both CPDLI algorithms find a high accuracy approximation quickly. The computation time for the data-independent algorithm CPDLI DI is almost a factor ten lower, as the number of known entries is high compared to the number of variables. The accuracy of the solution achieved by the other algorithms is lower, while the time needed to find a comparable accuracy is higher. Table 4 shows that the CPDLI algorithms result in a more accurate solution proving the usefulness of second-order information. Compared to an unconstrained model, computed by CPDI, the linear constraints improve accuracy more than an order of magnitude. Note that we did not report values for the CPD method [39], as it failed to reach the first accuracy level, illustrating again that its Gramian approximation does not suffice for structured missing entries.

TABLE 4

The results computed by CPDLI algorithms are almost an order of magnitude more accurate than those computed by MVR ALS and SDF in terms of median error and maximal error. Using linear constraints improves the achievable accuracy as the CPDI algorithm consistently returns the least accurate solution. The point-wise relative error on the training set is reported and is the median over 50 experiments.

	Rel. Error	Median rel. error E
CPDLI DI		$6.91 \cdot 10^{-6}$
CPDLI DD		$7.03 \cdot 10^{-6}$
MVR ALS		$5.29 \cdot 10^{-5}$
SDF		$3.49 \cdot 10^{-5}$
CPDI		$1.28 \cdot 10^{-4}$
	10^{-6} 10^{-5} 10^{-4} 10^{-3} 10^{-2}	

In the second experiment, we sample 200 entries at random from $\mathcal{G}_p$ and test CPDLI DD and the MVR ALS algorithm using the same parameters as in the first experiment. All errors are computed for all feasible entries, i.e., E can be seen as the

This manuscript is for review purposes only.

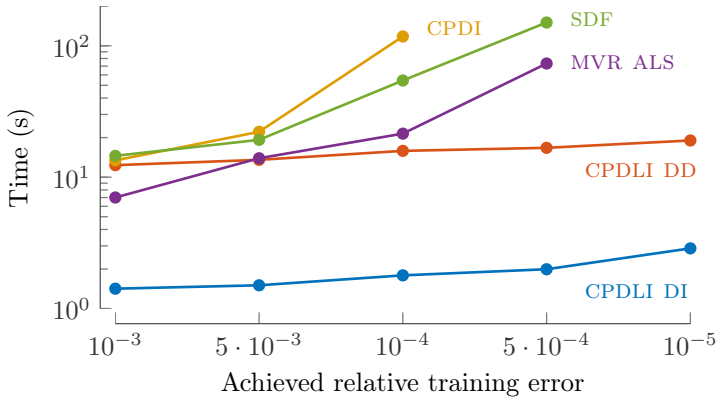


FIG. 4. *Comparison of the time needed to achieve a given accuracy or training error. Because of the second-order information,* CPDLI *requires few iterations to improve accuracy once a reasonable accurate solution is found which results in a low computation time. As many samples are known, the data-independent algorithm* CPDLI DI *reduces the computation time by a factor 10 compared to* CPDLI DD. MVR ALS *quickly finds a solution, but requires a large number of iterations to improve this solution.* SDF *is a general purpose solver which does not exploit all available structure.* CPDI *does not find a high accuracy solution as it ignores the linear constraints. The median time over 50 experiments is reported.*

validation error. Again, the CPDLI method achieves a high accuracy as can be seen in Table 5. While being slower for low accuracy solutions, CPDLI is more than an order of magnitude faster for high accuracies as can be seen in Figure 5. The speedup is caused mainly by the low number of iterations needed.

TABLE 5

CPDLI *is more than an order of magnitude more accurate than* MVR ALS *in terms of relative error on the validation data, when 200 samples are used for training (experiment 2). The median of the relative error reported here are computed for a rank $R = 6$ and degree $d = 4$ multivariate polynomial model ($D_n = 5$). As a baseline, the accuracy when the mean value is used as predictor is shown as well.*

	Rel. Error	Median rel. error E
CPDLI DD		$6.33 \cdot 10^{-6}$
MVR ALS		$2.69 \cdot 10^{-5}$
Mean		$1.84 \cdot 10^{-1}$
	10^{-6} 10^{-4} 10^{-2} 10^{0}	

7. Conclusion. Prior knowledge can be incorporated in a CPD using linear constraints, and enables significant reductions in computation time while improving accuracy, as illustrated using experiments. In the case of incomplete tensors, careful exploitation of Kronecker products allows efficient implementations of nonlinear least squares type algorithms CPDLI DD and CPDLI DI. For a large number of known entries and a relatively low number of variables, a projected version of the data removes the dependency on the number of known entries from the per-iteration computational complexity. For large-scale problems, the overall computational cost is reduced significantly using a cheap yet effective statistical block-Jacobi type preconditioner which uses the expected value of the Gramian. As a special case, the CPDI algorithm for

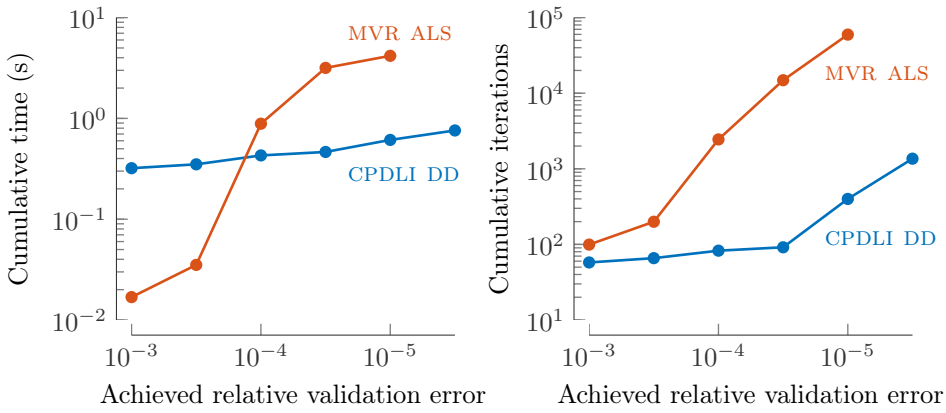


FIG. 5. CPDLI *finds a high accuracy solution quickly even though* MVR ALS *is faster in achieving a low accuracy solution. To find a high accuracy solution, the* CPDLI *algorithm benefits from its better convergence properties which translates into a low number of required iterations. The reported results are averages over 50 random initializations.*

the unconstrained CPD is discussed which benefits from better approximations to the second-order information, especially for difficult problems with structured missing entries. The effectiveness and efficiency of the presented algorithms are shown in a novel materials science application. In this application, the CPDLI algorithms improved the accuracy by almost an order of magnitude, while having a significantly lower computational cost.

Appendix A. Experiment parameters.

A.1. CPD of incomplete tensors. The `cpd_nls` implementation from Tensorlab [44] is used for the NLS based algorithms CPD and CPDI (using the `UseCPDI = true` option). In the case of CPDI, the statistical preconditioner $\mathbf{M}_{\text{INC,STAT}}$ presented here is used, as well as an optimized C implementation to construct the sparse Jacobian matrices. Both algorithms use at most `CGMaxIter = 40` CG iterations. The MINF algorithm [44] is implemented using `cpd_minf` with the `minf_ncg` solver which uses NCG with modified Hestenes–Stiefel updates and Moré–Thuente line search. The CP WOPT algorithm [2, 6] uses NCG with the same settings. The Tensorlab routines use `TolFun = 1e-24` and `TolFun = 1e-11` as stopping criterion for the high and low accuracy experiments, respectively. Other parameters are set such that they do not influence convergence. CP WOPT uses `StopTol = 1e-11` and `RelFuncTol = 1e-11` as stopping criterion.

A.2. Materials science application. Our implementation of MVR ALS from [7] uses the backslash operator to solve linear system in contrast to the CG approach suggested in [7], and computes the function value every iteration. The SDF implementation from Tensorlab [44] uses `struct_matvec` to implement the linear constraint and the `cpdi` kernel presented in this paper, but without any preconditioner. The CPDLI algorithms use the small-scale version, i.e., the Gramian $\mathbf{J}^{\text{H}}\mathbf{J}$ is constructed and inverted instead of using CG. SDF and CPDI perform at most `CGMaxIter = 150` CG iterations, while the CPDLI algorithms use at most `CGMaxIter = 15` iterations. The CPDI algorithm take advantage of the statistical block-Jacobi preconditioner $\mathbf{M}_{\text{INC,STAT}}$.

This manuscript is for review purposes only.

Acknowledgments. We would like to thank N. Moelans, Y. Yuan and B. Weltens for providing the materials science data and domain knowledge.

REFERENCES

[1] *Thermo-Calc software COST 531 database version 3.* www.thermocalc.com. Accessed Jul 2016.

[2] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, *Scalable tensor factorizations for incomplete data*, Chemometrics and Intelligent Laboratory Systems, 106 (2011), pp. 41–56.

[3] T. Adali, P. J. Schreier, and L. L. Scharf, *Complex-valued signal processing: The proper way to deal with impropriety*, IEEE Trans. Signal Process, 59 (2011), pp. 5101–125.

[4] B. K. Alsberg and O. M. Kvalheim, *Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplication*, Chemometrics and Intelligent Laboratory Systems, 24 (1994), pp. 31–42.

[5] B. W. Bader and T. G. Kolda, *Efficient matlab computations with sparse and factored tensors*, SIAM Journal on Scientific Computing, 30 (2008), pp. 205–231.

[6] B. W. Bader, T. G. Kolda, et al., *Matlab tensor toolbox version 2.6.* Available online, February 2015.

[7] G. Beylkin, J. Garcke, and M. J. Mohlenkamp, *Multivariate regression and machine learning with sums of separable functions*, SIAM Journal on Scientific Computing, 31 (2009), pp. 1840–1857.

[8] R. Bro and C. A. Andersson, *Improving the speed of multiway algorithms: Part II: Compression*, Chemometrics and Intelligent Laboratory Systems, 42 (1998), pp. 105–113.

[9] C. F. Caiafa and A. Cichocki, *Multidimensional compressed sensing and their applications*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 3 (2013), pp. 355–380.

[10] E. Candes and M. Wakin, *An introduction to compressive sampling*, IEEE Signal Processing Magazine, 25 (2008), pp. 21–30.

[11] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and A.-H. Phan, *Tensor decompositions for signal processing applications: From two-way to multiway component analysis*, IEEE Signal Processing Magazine, 32 (2015), pp. 145–163.

[12] D. Donoho, *Compressed sensing*, IEEE Transactions on Information Theory, 52 (2006), pp. 1289–1306.

[13] J. Douglas Carroll, S. Pruzansky, and J. B. Kruskal, *Candelinc: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters*, Psychometrika, 45 (1980), pp. 3–24.

[14] J. Garcke, *Sparse grids in a nutshell*, Sparse Grids and Applications, (2012), pp. 57–80.

[15] L. Grasedyck, D. Kressner, and C. Tobler, *A literature survey of low-rank tensor approximation techniques*, GAMM-Mitteilungen, 36 (2013), pp. 53–78.

[16] W. Hackbusch, *Tensor spaces and numerical tensor calculus*, Springer Series in Computational Mathematics, (2012).

[17] R. A. Harshman and M. E. Lundy, *Parafac: Parallel factor analysis*, Computational Statistics & Data Analysis, 18 (1994), pp. 390–72.

[18] I. Jeon, E. E. Papalexakis, C. Faloutsos, L. Sael, and U. Kang, *Mining billion-scale tensors: Algorithms and discoveries*, The VLDB Journal, 25 (2016), pp. 519–544.

[19] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, *Gigatensor: Scaling tensor analysis up by 100 times - algorithms and discoveries*, Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12, (2012).

[20] L. Karlsson, D. Kressner, and A. Uschmajew, *Parallel algorithms for tensor completion in the CP format*, Parallel Computing, 57 (2016), pp. 222–234.

[21] H. A. Kiers and R. A. Harshman, *Relating two proposed methods for speedup of algorithms for fitting two- and three-way principal component and related multilinear models*, Chemometrics and Intelligent Laboratory Systems, 36 (1997), pp. 31–40.

[22] T. Kolda and B. Bader, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500.

[23] K. Konakli and B. Sudret, *Polynomial meta-models with canonical low-rank approximations: Numerical insights and comparison to sparse polynomial chaos expansions*, Journal of Computational Physics, 321 (2016), pp. 1144–1169.

[24] A. Kroupa, *Modelling of phase diagrams and thermodynamic properties using Calphad method — Development of thermodynamic databases*, Computational Materials Science, 66 (2013), pp. 3–13.

This manuscript is for review purposes only.

[25] A. Kroupa, A. T. Dinsdale, A. Watson, J. Vrestal, J. Vízdal, and A. Zemanova, *The development of the COST 531 lead-free solders thermodynamic database*, JOM, 59 (2007), pp. 20–25.
[26] H. L. Lukas, S. G. Fries, and B. Sundman, *Computational thermodynamics: The Calphad method*, Cambridge university press, 2007.
[27] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, 2nd ed., 2006.
[28] I. Oseledets and E. Tyrtyshnikov, *TT-cross approximation for multidimensional arrays*, Linear Algebra and its Applications, 432 (2010), pp. 70–88.
[29] E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, *Parcube: Sparse parallelizable CANDECOMP-PARAFAC tensor decomposition*, ACM Transactions on Knowledge Discovery from Data, 10 (2015), pp. 1–25.
[30] E. E. Papalexakis, T. M. Mitchell, N. D. Sidiropoulos, C. Faloutsos, P. P. Talukdar, and B. Murphy, *Turbo-SMT: Parallel coupled sparse matrix-tensor factorizations and applications*, Statistical Analysis and Data Mining: The ASA Data Science Journal, 9 (2016), pp. 269–290.
[31] A.-H. Phan, P. Tichavský, and A. Cichocki, *Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations*, IEEE Transactions on Signal Processing, 61 (2013), pp. 4834–4846.
[32] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, *Tensor decomposition for signal processing and machine learning*. IEEE Transactions on Signal Processing (to appear), 2017.
[33] N. D. Sidiropoulos and A. Kyrillidis, *Multi-way compressed sensing for sparse low-rank tensors*, IEEE Signal Processing Letters, 19 (2012), pp. 757–760.
[34] N. D. Sidiropoulos, E. E. Papalexakis, and C. Faloutsos, *Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition*, IEEE Signal Processing Magazine, 31 (2014), pp. 57–70.
[35] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, *SPLATT: Efficient and parallel sparse tensor-matrix multiplication*, in 2015 IEEE International Parallel and Distributed Processing Symposium, May 2015, pp. 61–70.
[36] L. Sorber, M. Van Barel, and L. De Lathauwer, *Unconstrained optimization of real functions in complex variables*, SIAM Journal on Optimization, 22 (2012), pp. 879–898.
[37] L. Sorber, M. Van Barel, and L. De Lathauwer, *Complex optimization toolbox v1.0*, February 2013.
[38] 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 Journal on Optimization, 23 (2013), pp. 695–720.
[39] L. Sorber, M. Van Barel, and L. De Lathauwer, *Structured data fusion*, IEEE Journal of Selected Topics in Signal Processing, 9 (2015), pp. 586–600.
[40] G. Tomasi and R. Bro, *Parafac and missing values*, Chemometrics and Intelligent Laboratory Systems, 75 (2005), pp. 163–180.
[41] N. Vannieuwenhoven, K. Meerbergen, and R. Vandebril, *Computing the gradient in optimization algorithms for the CP decomposition in constant memory through tensor blocking*, SIAM Journal on Scientific Computing, 37 (2015), pp. C415–C438.
[42] N. Vervliet and L. De Lathauwer, *A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors*, IEEE Journal of Selected Topics in Signal Processing, 10 (2016), pp. 284–295.
[43] 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 Processing Magazine, 31 (2014), pp. 71–79.
[44] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, *Tensorlab 3.0*, Mar 2016. Available online.
[45] T. Yokota, R. Zdunek, A. Cichocki, and Y. Yamashita, *Smooth nonnegative matrix and tensor factorizations for robust multi-way data analysis*, Signal Processing, 113 (2015), pp. 234–249.

This manuscript is for review purposes only.

SUPPLEMENTARY MATERIALS: CANONICAL POLYADIC DECOMPOSITION OF INCOMPLETE TENSORS WITH LINEARLY CONSTRAINED FACTORS*

NICO VERVLIET†, OTTO DEBALS†, AND LIEVEN DE LATHAUWER†

Detailed derivations of the CPDLI DD and CPDLI DI algorithms are presented in section SM1 and the proposed preconditioners are discussed in section SM2.

SM1. Derivation of CPDLI. The derivations of CPDLI DD (subsection SM1.2) and CPDLI DI (subsection SM1.3) are given here. To keep the multilinear structure clear, element-wise expressions are avoided. An overview of useful identities regarding multilinear algebra is given in subsection SM1.1.

SM1.1. Identities. To derive expressions for the CPD of incomplete, the following identities involving Kronecker, (row-wise) Khatri–Rao and Hadamard products are useful:

$$(\mathbf{A} \odot \mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}} \odot^{\mathrm{T}} \mathbf{B}^{\mathrm{T}}, \tag{SM1}$$

$$(\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \odot \mathbf{D}) = \mathbf{AB} \odot \mathbf{CD}, \tag{SM2}$$

$$(\mathbf{A} \odot^{\mathrm{T}} \mathbf{C})(\mathbf{B} \otimes \mathbf{D}) = \mathbf{AB} \odot^{\mathrm{T}} \mathbf{CD}, \tag{SM3}$$

$$(\mathbf{A} \odot^{\mathrm{T}} \mathbf{B})(\mathbf{C} \odot \mathbf{D}) = \mathbf{AC} * \mathbf{BD}. \tag{SM4}$$

As explained in subsection 1.1, we assume matrix multiplication takes precedence over Kronecker, (row-wise) Khatri–Rao and Hadamard products to reduce the number of parentheses needed. The well-known result for the product of a Kronecker matrix with a vectorized matrix can be extended using permutation matrices $\mathbf{P}^{(n)}$:

$$(\mathbf{A} \otimes \mathbf{B})\mathrm{vec}\,(\mathbf{C}) = \mathrm{vec}\,(\mathbf{BCA}^{\mathrm{T}}), \tag{SM5}$$

$$(\mathbf{C} \odot \mathbf{B} \otimes \mathbf{I})\mathrm{vec}\,(\mathbf{A}) = \mathrm{vec}\,(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket), \tag{SM6}$$

$$\mathbf{P}^{(2)^{\mathrm{T}}}(\mathbf{C} \odot \mathbf{A} \otimes \mathbf{I})\mathrm{vec}\,(\mathbf{B}) = \mathrm{vec}\,(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket). \tag{SM7}$$

The Matrix Cookbook [SM1] provides useful rules for (vectorized) matrix derivatives:

$$\partial \mathrm{vec}\,(\mathbf{AB}) = (\mathbf{I} \otimes \mathbf{A})\partial \mathrm{vec}\,(\mathbf{B}) + (\mathbf{B}^{\mathrm{T}} \otimes \mathbf{I})\partial \mathrm{vec}\,(\mathbf{A}) \tag{SM8}$$

$$\partial \mathbf{AB} = (\partial \mathbf{A})\mathbf{B} + \mathbf{A}(\partial \mathbf{B}). \tag{SM9}$$

SM1.2. Derivation of the data-dependent algorithm. The expressions for CPDLI DD (subsection 2.1) are derived here. The objective value and Gramian-vector product are already explained in the main text and are therefore omitted.

*Submitted to the editors on April 19, 2017.

Funding: Nico Vervliet is supported by an Aspirant Grant from the Research Foundation — Flanders (FWO). Otto Debals is supported by a Ph.D. grant of the Agency for Innovation by Science and Technology (IWT). Research furthermore supported by: (1) Flemish Government: FWO: projects: G.0830.14N, G.0881.14N; (2) 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 (n° 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; (3) KU Leuven Internal Funds C16/15/059.

†KU Leuven, Dept. of Electrical Engineering ESAT/STADIUS, Kasteelpark Arenberg 10, bus 2446, B-3001 Leuven, Belgium; and Group Science, Engineering and Technology, KU Leuven - Kulak, E. Sabbelaan 53, 8500 Kortrijk, Belgium (Nico.Vervliet@kuleuven.be, Otto.Debals@kuleuven.be, Lieven.DeLathauwer@kuleuven.be).

This manuscript is for review purposes only.

Gradient. While the computation of the function value is straightforward, the gradient expression is more involved. To compute the gradient $\mathbf{G}^{(n)}$ w.r.t. $\mathbf{C}^{(n)}$, the matrix unfolding of the residual tensor

$$\mathbf{F}_{(n)} = \mathbf{S}_{(n)} * \left(\mathbf{A}^{(n)}\mathbf{V}^{(n)^{\mathrm{T}}} - \mathbf{T}_{(n)}\right) \qquad \text{where} \qquad \mathbf{V}^{(n)} = \bigodot_{k \neq n} \mathbf{A}^{(k)} \tag{SM10}$$

is used. First, the derivatives w.r.t. $\mathbf{A}^{(n)}$ are computed, after which the chain rule is used to compute the derivative w.r.t. $\mathbf{C}^{(n)}$:

$$\begin{aligned}
\frac{\partial f}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} &= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \frac{1}{2} \left|\left|\mathbf{F}_{(n)}\right|\right|_{\mathrm{F}}^2 && \text{(SM11)}\\
&= \mathrm{vec}\left(\mathbf{F}_{(n)}\right)^{\mathrm{H}} \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \mathrm{vec}\left(\mathbf{S}_{(n)} * \left(\mathbf{A}^{(n)}\mathbf{V}^{(n)^{\mathrm{T}}} - \mathbf{T}_{(n)}\right)\right) && \text{(SM12)}\\
&= \mathrm{vec}\left(\mathbf{F}_{(n)}\right)^{\mathrm{H}} \mathrm{diag}\left(\mathrm{vec}\left(\mathbf{S}_{(n)}\right)\right) \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \mathrm{vec}\left(\mathbf{A}^{(n)}\mathbf{V}^{(n)^{\mathrm{T}}}\right) && \text{(SM13)}\\
&= \mathrm{vec}\left(\mathbf{F}_{(n)}\right)^{\mathrm{H}} \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \mathrm{vec}\left(\mathbf{A}^{(n)}\mathbf{V}^{(n)^{\mathrm{T}}}\right) && \text{(SM14)}\\
&= \mathrm{vec}\left(\mathbf{F}_{(n)}\right)^{\mathrm{H}} \left(\mathbf{V}^{(n)} \otimes \mathbf{I}\right) && \text{(SM15)}\\
&= \mathrm{vec}\left(\bar{\mathbf{F}}_{(n)}\mathbf{V}^{(n)}\right)^{\mathrm{T}}. && \text{(SM16)}
\end{aligned}$$

The Hadamard product is written as a matrix multiplication to obtain (SM12). As $\mathbf{F}_{(n)}$ already contains a Hadamard product with $\mathbf{S}_{(n)}$, the diagonal matrix can be omitted in (SM13). The result is reshaped into an $I_n \times R$ matrix $\mathbf{G}^{(n)}$:

$$\mathbf{G}_A^{(n)} = \bar{\mathbf{F}}_{(n)}\mathbf{V}^{(n)}. \tag{SM17}$$

The derivative w.r.t. $\mathbf{C}^{(n)}$ is found by applying the chain rule:

$$\begin{aligned}
\mathrm{vec}\left(\mathbf{G}^{(n)}\right)^{\mathrm{T}} = \frac{\partial f}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} &= \frac{\partial f}{\partial \mathrm{vec}\left(\mathbf{A}^{(n)}\right)} \frac{\partial \mathbf{A}^{(n)}}{\partial \mathbf{C}^{(n)}} && \text{(SM18)}\\
&= \mathrm{vec}\left(\mathbf{G}_A^{(n)}\right)^{\mathrm{T}} (\mathbf{I} \otimes \mathbf{B}) && \text{(SM19)}\\
&= \mathrm{vec}\left(\mathbf{B}^{(1)^{\mathrm{T}}}\bar{\mathbf{F}}_{(n)}\mathbf{V}^{(n)}\right)^{\mathrm{T}}. && \text{(SM20)}
\end{aligned}$$

Next, the structure of $\mathbf{V}^{(n)}$ is exploited by using its definition in terms of $\mathbf{A}^{(n)}$ and identity (SM2):

$$\begin{aligned}
\mathbf{G}^{(n)} &= \mathbf{B}^{(1)^{\mathrm{T}}}\bar{\mathbf{F}}_{(n)} \left(\bigotimes_{k \neq n} \mathbf{B}^{(k)}\right) \left(\bigodot_{k \neq n} \mathbf{C}^{(k)}\right) && \text{(SM21)}\\
&= \bar{\mathbf{Z}}_{(n)} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)}\right) && \text{(SM22)}
\end{aligned}$$

in which the (conjugated) auxiliary tensor $\bar{\mathcal{Z}} = \bar{\mathcal{F}} \bullet_1 \mathbf{B}^{(1)^{\mathrm{T}}} \bullet \cdots \bullet_N \mathbf{B}^{(N)^{\mathrm{T}}}$ is used.

Gramian. The Gramian of the Jacobian $\mathbf{H}$ is computed as $\mathbf{J}^{\mathrm{H}}\mathbf{J}$ in which $\mathbf{J}$ is the Jacobian of the objective function (13). The Jacobian matrix is divided w.r.t. the factor matrices, such that $\mathbf{J} = \begin{bmatrix}\mathbf{J}^{(1)} & \cdots & \mathbf{J}^{(N)}\end{bmatrix}$. Consequently, we have $N \times N$

This manuscript is for review purposes only.

blocks in the Gramian, i.e., $\mathbf{H}^{(m,n)} = \mathbf{J}^{(m)^{\mathrm{H}}}\mathbf{J}^{(n)}$. The Jacobian $\mathbf{J}^{(n)}$ is the derivative of the vectorized residual $\mathcal{F}$ w.r.t. the vectorized factor matrix $\mathbf{C}^{(n)}$ and is given by

$$\mathbf{J}^{(n)} = \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \mathrm{vec}\left(\mathcal{F}\right) \tag{SM23}$$

$$= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \left(\mathbf{A}^{[1]} * \cdots * \mathbf{A}^{[N]}\right) \mathbf{1} \tag{SM24}$$

$$= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \left(\mathbf{V}^{[n]}\mathbf{I} * \mathbf{B}^{[n]}\mathbf{C}^{(n)}\right) \mathbf{1} \tag{SM25}$$

$$= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right) \left(\mathbf{I} \odot \mathbf{C}^{(n)}\right) \mathbf{1} \tag{SM26}$$

$$= \frac{\partial}{\partial \mathrm{vec}\left(\mathbf{C}^{(n)}\right)} \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right) \mathrm{vec}\left(\mathbf{C}^{(n)}\right) \tag{SM27}$$

$$= \mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}, \tag{SM28}$$

where we have used the definition of $\mathbf{V}^{[n]} = *_{k\neq n}\, \mathbf{A}^{[n]}$, followed by identities (SM4) and (SM7). Each Gramian block is then given by

$$\mathbf{H}^{(m,n)} = \left(\mathbf{V}^{[m]^{\mathrm{H}}} \odot \mathbf{B}^{[m]^{\mathrm{H}}}\right) \left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right). \tag{SM29}$$

This expression is further simplified in the data-independent formulation.

SM1.3. Derivation of the data-independent algorithm. The derivation of CPDLI DI (subsection 2.2) is discussed here.

Objective function. The norm in the objective function f is expanded as

$$\min f = \frac{1}{2} f_1 - \mathrm{Re}\left(f_2\right) + \frac{1}{2} f_3. \tag{SM30}$$

The first term is computed as $f_1 = ||\mathcal{S} * \mathcal{T}||^2 = \hat{\mathbf{t}}^{\mathrm{H}}\hat{\mathbf{t}}$ with $\hat{\mathbf{t}} \in \mathbb{K}^{N_{\mathrm{ke}}}$ containing the known entries of $\mathcal{T}$. The second term f_2 is given by

$$f_2 = \left\langle \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}\right]\!\right], \mathcal{T} \right\rangle_{\mathcal{S}} \tag{SM31}$$

$$= \hat{\mathbf{t}}^{\mathrm{H}} \left(\mathbf{A}^{[N]} * \cdots * \mathbf{A}^{[1]}\right) \mathbf{1} \tag{SM32}$$

$$= \hat{\mathbf{t}}^{\mathrm{H}} \left(\mathbf{B}^{[N]}\mathbf{C}^{(n)} * \cdots * \mathbf{B}^{[1]}\mathbf{C}^{(1)}\right) \mathbf{1} \tag{SM33}$$

$$= \hat{\mathbf{t}}^{\mathrm{H}} \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]}\right) \left(\bigodot_n \mathbf{C}^{(n)}\right) \mathbf{1} \tag{SM34}$$

$$= \mathrm{vec}\left(\left(\mathcal{S} * \bar{\mathcal{T}}\right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}}\right)^{\mathrm{T}} \mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right). \tag{SM35}$$

We subsequently applied the definitions of the weighted inner product and extended matrices, the definition of $\mathbf{A}^{[n]}$, identity (SM4) and finally the definition of tensor-

This manuscript is for review purposes only.

matrix products and of a vectorized CPD. Finally, the third term is derived as

(SM36)
$$f_3 = \left\langle \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right], \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] \right\rangle_{\mathcal{S}}$$

(SM37)
$$= \operatorname{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right)^{\mathrm{H}} \left[\left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right)^{\mathrm{H}} \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right) \right] \operatorname{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right)$$

(SM38)
$$= \operatorname{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right)^{\mathrm{H}} \mathbf{D} \operatorname{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(N)} \right]\!\right] \right),$$

in which we applied the definition of $\mathbf{A}^{(n)}$, identity (SM4), and the definition of $\mathbf{D}$.

Gradient. To derive the expression for the gradient in the data-independent algorithm, we expand the gradient result (SM21) using the definition of the residual tensor $\mathcal{F} = \mathcal{S} * \left(\left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] - \mathcal{T} \right)$:

(SM39)
$$\mathbf{G}^{(n)} = \mathbf{G}_1^{(n)} - \mathbf{G}_2^{(n)}.$$

To compute the first part $\mathbf{G}_1^{(n)} = \bar{\mathbf{Z}}_{(n)} \odot_n \mathbf{C}^{(n)}$ we use $\bar{\mathcal{Z}}$ which is given by

(SM40)
$$\operatorname{vec}\left(\bar{\mathcal{Z}}\right) = \operatorname{vec}\left(\left(\mathcal{S} * \left[\!\left[\bar{\mathbf{A}}^{(1)}, \ldots, \bar{\mathbf{A}}^{(N)} \right]\!\right] \right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}} \right)$$

(SM41)
$$= \left(\bigotimes_n \mathbf{B}^{(n)} \right)^{\mathrm{T}} \operatorname{diag}\left(\operatorname{vec}\left(\mathcal{S} \right) \right) \operatorname{vec}\left(\left[\!\left[\bar{\mathbf{A}}^{(1)}, \ldots, \bar{\mathbf{A}}^{(N)} \right]\!\right] \right)$$

(SM42)
$$= \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right)^{\mathrm{T}} \left(\bar{\mathbf{B}}^{[N]} \bar{\mathbf{C}}^{(n)} * \cdots * \bar{\mathbf{B}}^{[1]} \bar{\mathbf{C}}^{(1)} \right) \mathbf{1}$$

(SM43)
$$= \left(\bigodot_n^{\mathrm{T}} \mathbf{B}^{[n]} \right)^{\mathrm{T}} \left(\bigodot_n^{\mathrm{T}} \bar{\mathbf{B}}^{[n]} \right) \left(\bigodot_n \bar{\mathbf{C}}^{(n)} \right) \mathbf{1}$$

(SM44)
$$= \bar{\mathbf{D}} \operatorname{vec}\left(\left[\!\left[\bar{\mathbf{C}}^{(1)}, \ldots, \bar{\mathbf{C}}^{(N)} \right]\!\right] \right).$$

We first applied the Kronecker definition of tensor-matrix products. In (SM42) the zero rows created by $\mathcal{S}$ are ignored and the definitions of $\mathbf{A}^{(n)}$ and a vectorized CPD are used. Finally, we again apply identity (SM4) and the definitions of $\mathbf{D}$ and a vectorized CPD.

The second part $\mathbf{G}_2^{(n)}$ is computed as

(SM45)
$$\mathbf{G}_2^{(n)} = \left(\bar{\mathcal{T}}_B \right)_{(n)} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)} \right)$$

(SM46)
$$\bar{\mathcal{T}}_B = \left(\mathcal{S} * \bar{\mathcal{T}} \right) \cdot_1 \mathbf{B}^{(1)^{\mathrm{T}}} \cdot \cdots \cdot_N \mathbf{B}^{(N)^{\mathrm{T}}}.$$

Combining $\mathbf{G}_1^{(n)}$ and $\mathbf{G}_2^{(n)}$ leads to

(SM47)
$$\mathbf{G}^{(n)} = \left(\bar{\mathcal{Z}} - \bar{\mathcal{T}}_B \right)_{(n)} \left(\bigodot_{k \neq n} \mathbf{C}^{(k)} \right).$$

This manuscript is for review purposes only.

Gramian. Equation (SM29) is the starting point for the derivation of each block in the Gramian:

(SM48)
$$\mathbf{H}^{(m,n)} = \left(\mathbf{V}^{[m]^{\mathrm{H}}} \odot \mathbf{B}^{[m]^{\mathrm{H}}}\right)\left(\mathbf{V}^{[n]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)$$

(SM49)
$$= \left(\underset{k\neq m}{*} \mathbf{B}^{[k]}\mathbf{C}^{(k)} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)^{\mathrm{H}}\left(\underset{k\neq n}{*} \mathbf{B}^{[k]}\mathbf{C}^{(k)} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right)$$

(SM50)
$$= \left[\left(\underset{k\neq m}{\odot^{\mathrm{T}}} \mathbf{B}^{[k]}\right)\left(\underset{k\neq m}{\odot} \mathbf{C}^{(k)}\right) \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right]^{\mathrm{H}}\left[\left(\underset{k\neq n}{\odot^{\mathrm{T}}} \mathbf{B}^{[k]}\right)\left(\underset{k\neq n}{\odot} \mathbf{C}^{(k)}\right) \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right]$$

(SM51)
$$= \left[\left(\underset{k\neq m}{\odot^{\mathrm{T}}} \mathbf{B}^{[k]} \odot^{\mathrm{T}} \mathbf{B}^{[m]}\right)\left(\underset{k\neq m}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)\right]^{\mathrm{H}} \cdot$$
$$\left[\left(\underset{k\neq n}{\odot^{\mathrm{T}}} \mathbf{B}^{[k]} \odot^{\mathrm{T}} \mathbf{B}^{[n]}\right)\left(\underset{k\neq n}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)\right]$$

(SM52)
$$= \left(\underset{k\neq m}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)}\left(\underset{k}{\odot^{\mathrm{T}}}\mathbf{B}^{[k]}\right)^{\mathrm{H}}\left(\underset{k}{\odot^{\mathrm{T}}}\mathbf{B}^{[k]}\right)\mathbf{P}^{(n)^{\mathrm{T}}}\left(\underset{k\neq n}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)$$

(SM53)
$$= \left(\underset{k\neq m}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)}\mathbf{D}\mathbf{P}^{(n)^{\mathrm{T}}}\left(\underset{k\neq n}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)$$

First, we applied identity (SM1) and the definition of $\mathbf{V}^{[n]}$, followed by (SM4). From (SM50) to (SM51) identity (SM3) is used, after which we use the properties of permutation matrices. Finally, the definition of $\mathbf{D}$ is used to arrive at the KRMTKRPROD.

Gramian vector product. As shown in the main text, the Gramian-vector products $\mathbf{Y}^{(m)}$ can be computed by first summing over n:

(SM54)
$$\mathrm{vec}\left(\mathbf{Y}^{(m)}\right) = \sum_{n=1}^{N} \mathbf{H}^{(m,n)}\mathrm{vec}\left(\mathbf{X}^{(n)}\right).$$

After substituting the definition of $\mathbf{H}^{(m,n)}$ in (SM53) and applying identity (SM7), we find

(SM55)
$$\mathrm{vec}\left(\mathbf{Y}^{(m)}\right) = \sum_{n=1}^{N}\left(\underset{k\neq m}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)}\mathbf{D}\mathbf{P}^{(n)^{\mathrm{T}}}\left(\underset{k\neq n}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)\mathrm{vec}\left(\mathbf{X}^{(n)}\right)$$

(SM56)
$$= \left(\underset{k\neq m}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)}\mathbf{D}\sum_{n=1}^{N}\mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{X}^{(n)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right).$$

Again, an auxiliary tensor $\mathcal{Z}$

(SM57)
$$\mathrm{vec}\left(\mathcal{Z}\right) = \mathbf{D}\sum_{n=1}^{N}\mathrm{vec}\left(\left[\!\left[\mathbf{C}^{(1)}, \ldots, \mathbf{X}^{(n)}, \ldots, \mathbf{C}^{(N)}\right]\!\right]\right),$$

is used to simplify the notations:

(SM58)
$$\mathrm{vec}\left(\mathbf{Y}^{(m)}\right) = \left(\underset{k\neq m}{\odot} \mathbf{C}^{(k)} \otimes \mathbf{I}\right)^{\mathrm{H}} \mathbf{P}^{(m)}\mathrm{vec}\left(\mathcal{Z}\right)$$

(SM59)
$$\mathbf{Y}^{(m)} = \mathbf{Z}_{(m)}\left(\underset{k\neq m}{\odot} \bar{\mathbf{C}}^{(k)}\right).$$

This manuscript is for review purposes only.

SM2. Preconditioner. To prove the expressions for the preconditioner for the computation of a CPD of an incomplete tensor without (Equation (44)) and with (Equation (46)) linear constraints, we start from the block diagonal approximation of the Gramian if all entries are known. First, linear constraints are ignored. Each diagonal block is given as $\mathbf{H}^{(n,n)} = \mathbf{W} \otimes \mathbf{I}_{I_n}$ [SM2]. Element-wise, each non-zero entry is therefore given by

$$h^{(n,n)}_{(i_n,r),(i_n,s)} = \sum_{i_1=1}^{I_1} \cdots \sum_{i_{n-1}=1}^{I_{n-1}} \sum_{i_{n+1}=1}^{I_{n+1}} \cdots \sum_{i_N=1}^{I_N} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s} \tag{SM60}$$

$$= \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)\in\pi^{(n)}_{i_n}} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}, \tag{SM61}$$

in which the set $\pi^{(n)}_{i_n}$ contains all multi-indices in the i_nth mode-n slice of order $N-1$. As all entries are known $\pi^{(n)}_{i_n} = \pi^{(n)}$ for $i_n = 1, \ldots, I_n$ and

$$\pi^{(n)} = \{(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N) | i_k = 1, \ldots, I_k \text{ and } k = 1, \ldots, n-1, n+1, \ldots, N\}. \tag{SM62}$$

In the case of missing entries, the number and position of missing entries can vary in each slice. Therefore, the sets $\pi^{(n)}_{i_n}$ in (SM61) are replaced by the sets $\hat{\pi}^{(n)}_{i_n}$ containing the multi-indices of the known entries in each slice:

$$\hat{\pi}^{(n)}_{i_n} = \left\{ (i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N) \,\middle|\, (i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N) \in \pi^{(n)}; s_{i_1,\ldots,i_N} = 1 \right\}. \tag{SM63}$$

Using index sets $\hat{\pi}^{(n)}_{i_n}$, (SM61) boils down to element-wise expression for the diagonal blocks of the Gramian for incomplete tensors.

The set $\hat{\pi}^{(n)}_{i_n}$ contains $Q^{(n)}_{i_n}$ elements and corresponds to one particular distribution of entries. Let $\Pi^{(n)}_{i_n}$ be the set of all possible partitions of $Q^{(n)}_{i_n}$ known entries, i.e.,

$$\Pi^{(n)}_{i_n} = \left\{ \hat{\pi}^{(n)} \,\middle|\, |\hat{\pi}^{(n)}| = Q^{(n)}_{i_n} \right\}. \tag{SM64}$$

If all partitions $\hat{\pi}^{(n)}$ in $\Pi^{(n)}_{i_n}$ are equally likely, the expected value of $h^{(n,n)}_{(i_n,r),(i_n,s)}$ over $\Pi^{(n)}_{i_n}$ is given by

$$\mathbb{E}\left[h^{(n,n)}_{(i_n,r),(i_n,s)}\right] = \frac{1}{|\Pi^{(n)}_{i_n}|} \sum_{\hat{\pi}^{(n)}\in\Pi^{(n)}_{i_n}} \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)\in\hat{\pi}^{(n)}} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}. \tag{SM65}$$

It can be verified that each product $\prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}$ appears exactly $|\Pi^{(n)}_{i_n}| \frac{Q^{(n)}_{i_n}}{\prod_{k\neq n} I_k}$ times, hence

$$\mathbb{E}\left[h^{(n,n)}_{(i_n,r),(i_n,s)}\right] = \frac{N^{(n)}_{i_n}}{\prod_{k\neq n} I_k} \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)\in\pi^{(n)}} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s}. \tag{SM66}$$

The fraction $\frac{Q^{(n)}_{i_n}}{\prod_{k\neq n} I_k}$ is exactly the fraction of known entries $f^{(n)}_{i_n} = \mathbf{f}^{(n)}(i_n)$ in the i_nth mode-n slice. The expected value for the block can then be written as

$$\mathbb{E}\left[\mathbf{H}^{(n,n)}\right] = \mathbf{W}^{(n)} \otimes \operatorname{diag}(\mathbf{f}^{(n)}), \tag{SM67}$$

This manuscript is for review purposes only.

which concludes the derivation of (44).

In the case of linear constraints, each Gramian entry is given by

$$h^{(n,n)}_{(d_1,r),(d_2,s)} = \sum_{i_n=1}^{I_n} \sum_{(i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N)\in\pi^{(n)}} \overline{b}^{(n)}_{i_n,d_1} b^{(n)}_{i_n,d_2} \prod_{k\neq n} \overline{a}^{(k)}_{i_k,r} a^{(k)}_{i_k,s} \tag{SM68}$$

Using a similar derivation, the expected value of the block diagonal of the Gramian with respect to all possible partitions is given by

$$\mathbb{E}\left[\mathbf{H}^{(n,n)}\right] = \mathbf{W}^{(n)} \otimes \left(\mathbf{B}^{(n)^{\mathrm{H}}} \mathrm{diag}(\mathbf{f}^{(n)}) \mathbf{B}^{(n)}\right), \tag{SM69}$$

which is expression (46).

REFERENCES

[SM1] K. B. Petersen and M. S. Pedersen, *The matrix cookbook*, Nov 2012. Version 2012-11-15.

[SM2] 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 Journal on Optimization, 23 (2013), pp. 695–720.