Numerical optimization based algorithms for data fusion

N. Vervliet, L. De Lathauwer

Abstract

Combining various sources of information to discover hidden patterns is key in data analysis. These sources can often be represented as matrices and/or multiway arrays, or tensors, which can be factorized jointly, e.g. as sums of simple terms, to gain insight into the data. In this chapter, an overview of (the rationale behind) numerically well-founded optimization techniques based on a Gauss–Newton framework is given, which has superior convergence properties and allows all multilinear structure to be exploited. Prior knowledge in the form of parametric, box or soft constraints as well as regularization can be incorporated easily. We show how matrices and/or tensors can be coupled through (partially) shared factors or through common underlying variables. The framework is further extended to more general divergences allowing more suitable statistical assumptions. Finally, as tensor problems become large-scale quickly, due to the curse of dimensionality, techniques used to alleviate or overcome this curse are discussed.

Keywords: tensor decomposition, canonical polyadic decomposition, numerical optimization, Gauss–Newton, constrained decomposition, data fusion, coupled decomposition, algorithms

1. Introduction

Consider the two views of two point clouds in a 3D space in Figure 1. When looking at either one of the views separately, it is impossible to distinguish the cloud of triangles from the cloud of circles. However, when the information from both views is combined, a 3D view can be constructed (Figure 1, right) and it becomes clear that both point clouds can be separated by a plane. In this simple example two views are *fused*, which fits in the larger framework of jointly analyzing multiple datasets. This is prevalent in data analysis as cheap measurement hardware and an enormous increase in computational power fuel the current information age, and have led to gargantuan amounts of data from very heterogeneous sources. To discover useful insights, combining information from all these sources is key as shown in chemometrics [1], neuroscience [2, 3], link prediction [4, 5], multidimensional harmonic retrieval [6, 7], multirate sampling [7], array processing [8], and so on.

The number of dimensions or *order* of the data are an important source of variation: while one and two-way data, which can be represented naturally as vectors and matrices,

Email addresses: Nico.Vervliet@kuleuven.be (N. Vervliet), Lieven.DeLathauwer@kuleuven.be (L. De Lathauwer)

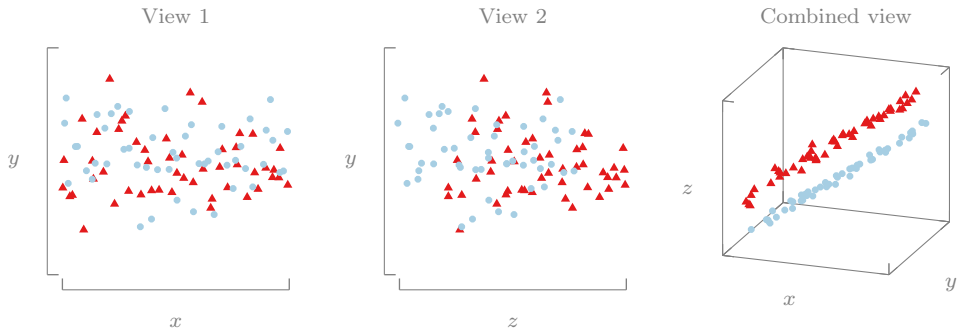


Figure 1: Clustering the points from a 3D space is impossible when only one of the left two views of the data is given. However, if the views are analyzed together, it becomes clear that both datasets can be separated by a plane as shown in the combined view.

are commonly used, many data sources are in fact multiway and can be represented by multiway arrays or *higher-order tensors* [9, 10]. To discover latent factors, these tensors can be factorized into simple terms such as a rank-1 term or a low multilinear rank terms, leading to decompositions such as the canonical polyadic decomposition (CPD) [11, 12], the decomposition in multilinear rank-$(L_r, L_r, 1)$ terms (LL1) [13, 14], the multilinear singular value decomposition (MLSVD) [15, 16] or the block term decomposition (BTD) [13]. Many of these tensor decompositions are especially attractive because of their mild uniqueness conditions [9, 10]. This has, for example, led to tensorization techniques which transform vector and matrix data to higher-order tensors [9, 17]. To improve interpretability and to further relax uniqueness conditions, additional constraints such as nonnegativity, orthogonality or symmetry can be added [9, 10, 18, 19, 20, 21]. These constraints can be seen as a form of prior knowledge. Similarly, by jointly analyzing multiple matrices and tensors, milder uniqueness conditions can be derived. (Pointers to such uniqueness results are given in section 5.)

While algebraic or semi-algebraic methods exist for (coupled and constrained) CPDs [22, 23, 20, 19, 24, 25], most methods are based on numerical optimization of a nonlinear least squares (NLS) objective function [26, 11, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36]. Rather than using an alternating least squares (ALS) approach, which is popular thanks to its simplicity of implementation and its speed for simple problems, we focus on Gauss–Newton (GN) type algorithms. These GN algorithms have several advantages over ALS. First, GN algorithms using a trust region converge to a (local) minimum for any starting point under mild conditions, while such global convergence is not guaranteed for ALS [37], even though ALS usually converges in practice. If the algorithm converges, GN converges often quadratically, while ALS converges only linearly. In practice, GN is often more robust, meaning that for many problems fewer initializations are required [38, 28, 29], and GN is less susceptible for swamps, i.e., long periods of little improvement [39, 40], thanks to the use of (approximate) curvature information [38]. For the unconstrained CPD, inexact GN algorithms have the same asymptotic per-iteration complexity as ALS [38], but GN requires many fewer iterations. In contrast to ALS, which breaks the multilinear structure up in N substeps, GN type algorithms exploit all structure in every iteration.

Finally, the GN framework allows constraints, symmetry, coupling and regularization to be implemented easily [32], as illustrated in this chapter.

As the NLS objective implicitly assumes normally distributed residuals, objective functions based on other divergences may be more suitable if other statistical assumptions are more appropriate. For example, in the case of count data, a Poisson distribution may be more appropriate and a Kullback–Leibler (KL) divergence can be used instead [41, 42]. The KL divergence can also be used to improve music reconstruction [43]. More general distributions such as Tweedie distribution are discussed in combination with coupled factorizations in [44, 45, 43]. For nonnegative tensor factorization, algorithms using alpha and beta divergences are discussed in [46, 47, 48].

Apart from a large variety of datasets, one often has to deal with a large volume as well. Especially in the case of higher-order tensors, the cost of constructing, storing and performing computations with large-scale tensors can be daunting. This is due to the curse of dimensionality: the number of entries increases exponentially with the order of the tensor. To deal with this curse, various techniques have emerged going from sampling, exploitation of structure such as sparsity or structure that results from implicit tensorization, randomization and parallelization. An overview of such techniques is given in section 6.

1.1. Outline

After a discussion of the notation and some definitions in the remainder of this section, the most important optimization concepts are discussed in section 2, including a derivation of quasi-Newton and Gauss–Newton algorithms with line search or a trust region, and inexact variants which are important for large-scale implementations. The GN algorithm is then specialized for the computation of the unconstrained CPD in section 3 and is compared to ALS. In subsection 3.4, an extension to more general divergences including Kullback–Leibler and Itakura–Saito is made. Section 4 discusses how parametric constraints and box constraints can be incorporated easily in the framework. The use of regularization to implement soft constraints is discussed, as well as imposing symmetry. In section 5 some examples of coupling are discussed and a brief overview of uniqueness results is given. We also show how both hard and approximate coupling can be implemented in the GN framework. As many tensor problems are large-scale, an overview of techniques to handle these large-scale tensors is given in section 6.

1.2. Notation and definitions

To denote scalars, (column) vectors, matrices and tensors, the notations a, $\mathbf{a}$, $\mathbf{A}$ and $\mathcal{T}$ are used, respectively. For example, the ith entry of a vector $\mathbf{a}$ is denoted as a_i, and the rth column of a matrix $\mathbf{A}$ by $\mathbf{a}_r$. For simplicity of notation, only real, third-order tensors with dimensions $I \times J \times K$ are used in this chapter, i.e., $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$. A polyadic decomposition (PD) of $\mathcal{T}$ with factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$ and $\mathbf{C} \in \mathbb{R}^{K \times R}$ is denoted as $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, which is a shorthand notation for

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r,$$

with $\otimes$ the outer product. If R is minimal, the decomposition is called canonical (CPD) and R is the rank of the tensor. In practice, a rank-R approximation is often computed. The results in this chapter can be extended easily to complex and/or higher-order tensors; see [38, 32]. A mode-n vector is the generalization of a column ($n = 1$) or a row ($n = 2$) and is defined by fixing all but one index in $\mathcal{T}$, e.g. the mode-3 vectors are defined as $\mathcal{T}(i, j, :)$ using Matlab style notation. A mode-n unfolding of a tensor $\mathcal{T}$ is defined as the matrix $\mathbf{T}_{(n)}$ collecting all mode-n vectors as its columns, ordered such that the first index not equal to n runs faster than the second, e.g., $\mathbf{T}_{(1)}(:, (k-1)K + j) = \mathcal{T}(:, j, k)$ and $\mathbf{T}_{(2)}(:, (k-1)K + i) = \mathcal{T}(i, :, k)$. The vectorization operator $\mathrm{vec}\,(\mathcal{T})$ stacks all mode-1 (column) vectors into a column vector. The reverse operation $\mathrm{unvec}\,(\mathbf{t})$ reshapes $\mathbf{t}$ into a tensor $\mathcal{T}$ with dimensions $I \times J \times K$. The following products are required. The tensor matrix product in mode n is denoted by $\mathcal{T} \cdot_n \mathbf{A}$ and is defined in terms of the mode-n unfolding as $(\mathcal{T} \cdot_n \mathbf{A})_{(n)} = \mathbf{A}\mathbf{T}_{(n)}$. The Kronecker product and Khatri–Rao product are denoted by $\otimes$ and $\odot$, respectively, and are defined for matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{p \times q}$ and $\mathbf{C} \in \mathbb{R}^{l \times n}$ as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}, \quad \mathbf{A} \odot \mathbf{C} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{c}_1 & \mathbf{a}_2 \otimes \mathbf{c}_2 & \cdots & \mathbf{a}_n \otimes \mathbf{c}_n \end{bmatrix}.$$

The Hadamard, or element-wise, product is denoted by $*$. The transpose, pseudoinverse and Frobenius norm are denoted by $\cdot^{\mathrm{T}}$, $\cdot^{\dagger}$ and $||\cdot||_{\mathrm{F}}$, respectively. $\mathbf{I}_n$ denotes the $n \times n$ identity matrix and $\mathbf{1}_n$ is a length-n column vector with ones. The column-wise concatenation of two vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted by $\begin{bmatrix}\mathbf{a}; \mathbf{b}\end{bmatrix}$ and is a shorthand for $\begin{bmatrix}\mathbf{a}^{\mathrm{T}} & \mathbf{b}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$.

2. Numerical optimization for tensor decompositions

Matrix decompositions and tensor decompositions such as the MLSVD [15] or a tensor train (TT) approximation [49] are usually computed via algebraic algorithms such as the singular value decomposition (SVD). Similarly, a CPD can be computed using algebraic methods such as the generalized eigenvalue decomposition (GEVD) [22, 23]. The performance of these algebraic methods depends on the tensor to be decomposed and on which slices are used. In practice, an optimization approach is often taken as it has several advantages: it allows all multilinear information to be exploited, it is highly efficient and it is more robust to noise. The result of the algebraic algorithm can still be used as an initialization, though. Due to its importance for tensor decompositions into rank-1 terms, a high level overview of basic optimization concepts is given in this section; see, e.g., [50, 51] for a more in-depth discussion and [52, 26] for extensions to complex tensors.

Among the optimization techniques, a nonlinear least squares (NLS) method is commonly used in a tensor context. The objective of an NLS problem is to find $\mathbf{z}^* \in \mathbb{R}^N$ such that the squared error between a data vector $\mathbf{t}$ and a nonlinear model $m(\mathbf{z})$ is minimized. Mathematically, this can be written as

$$\mathbf{z}^* = \arg\min_{\mathbf{z}} f(\mathbf{z}) \qquad \text{with} \qquad f(\mathbf{z}) = ||m(\mathbf{z}) - \mathbf{t}||_{\mathrm{F}}^2, \tag{1}$$

in which $f(\mathbf{z})$ is called the objective or loss function.

> **Example 1:** When computing a rank-R CPD of a tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ with $\mathbf{t} = \text{vec}(\mathcal{T})$, the $N = (I + J + K)R$ variables are the factor matrices, i.e., $\mathbf{z} = \big[\text{vec}(\mathbf{A}); \text{vec}(\mathbf{B}); \text{vec}(\mathbf{C})\big]$. The model is then given by $m(\mathbf{z}) = \text{vec}(\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket)$.

To find a (possibly local) optimizer $\mathbf{z}^*$ for the objective function (1), an initial guess $\mathbf{z}_0$ is iteratively refined by taking a step of length α_k in direction $\mathbf{p}_k$:

$$\mathbf{z}_k = \mathbf{z}_{k-1} + \alpha_k \mathbf{p}_k.$$

This is repeated until some stopping criterion is satisfied. Two important approaches for choosing the step length and direction are discussed in subsection 2.1. Both approaches rely on a local linear or quadratic approximation $\tilde{f}$ of the objective function to determine the step direction, as described in subsection 2.2. This approximation leads to a linear system which can be solved using direct or iterative techniques. The latter are critical to implement highly efficient algorithms for larger tensor problems; see subsection 2.3.

2.1. Line search and trust region

Line search and trust region algorithms are the two main approaches to update the variables such that $\mathbf{z}_k$ is closer to a (possibly local) optimum for the objective (1). In the case of line search, an update is computed as

$$\mathbf{z}_k = \mathbf{z}_{k-1} + \alpha_k \mathbf{p}_k, \tag{2}$$

in which $\mathbf{p}_k$ is the step direction determined by solving an easier subproblem $\tilde{f}(\mathbf{p}_k)$ and α_k is the step length along the direction $\mathbf{p}_k$. After finding $\mathbf{p}_k$, the optimal α_k along the line determined by $\mathbf{p}_k$ is found by solving the following optimization problem in a single variable:

$$\alpha_k = \arg\min_{\alpha > 0} f(\mathbf{z}_{k-1} + \alpha \mathbf{p}_k).$$

Usually, it is not necessary to find the optimal α as long as the objective function is decreased sufficiently. See [51] for more details on conditions for sufficient decrease.

If a trust region approach is used, the step direction and length are determined simultaneously by solving the constrained subproblem

$$\mathbf{p}_k = \arg\min_{\mathbf{p}} \tilde{f}(\mathbf{p}) \quad \text{subject to} \quad ||\mathbf{p}||_{\text{F}} \leq \Delta, \tag{3}$$

in which $\tilde{f}$ again is a local approximation to the objective function. The variables are updated as

$$\mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{p}_k,$$

hence $\alpha_k = 1$. While the constrained optimization problem (3) can be solved exactly, other approaches such as the dogleg method and plane search are often used as well; see, e.g., [51].

2.2. Determining step direction $\mathbf{p}_k$

Compared to nonlinear problems, linear problems are usually easy to solve. Therefore, the nonlinear objective function $f(\mathbf{z})$ is locally approximated by a function $\tilde{f}$ such that $\mathbf{p}$ can be found from a linear system. More mathematically, the function $f(\mathbf{z})$ is locally approximated by a Taylor series at the current guess $\mathbf{z}_k$:

$$f(\mathbf{z}_k + \mathbf{p}) \approx f(\mathbf{z}_k) + \mathbf{p}^{\mathrm{T}} \cdot \underbrace{\nabla_{\mathbf{z}} f(\mathbf{z}_k)}_{\mathbf{g}_k} + \frac{1}{2}\mathbf{p}^{\mathrm{T}} \cdot \underbrace{\nabla^2_{\mathbf{z}} f(\mathbf{z}_k)}_{\mathbf{H}_k} \cdot \mathbf{p} + \dots \tag{4}$$

in which $\mathbf{p} = \mathbf{z} - \mathbf{z}_k$. The derivatives $\mathbf{g}_k$ and $\mathbf{H}_k$ are the gradient and Hessian, respectively, both evaluated at $\mathbf{z}_k$:

$$\mathbf{g}_k = \begin{bmatrix} \frac{\partial f}{\partial z_1} & \frac{\partial f}{\partial z_2} & \cdots & \frac{\partial f}{\partial z_N} \end{bmatrix}^{\mathrm{T}},$$

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial^2 f}{\partial z_1^2} & \frac{\partial^2 f}{\partial z_1 \partial z_2} & \cdots & \frac{\partial^2 f}{\partial z_1 \partial z_N} \\ \frac{\partial^2 f}{\partial z_2 \partial z_1} & \frac{\partial^2 f}{\partial z_2^2} & \cdots & \frac{\partial^2 f}{\partial z_2 \partial z_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial z_N \partial z_1} & \frac{\partial^2 f}{\partial z_N \partial z_2} & \cdots & \frac{\partial^2 f}{\partial z_N^2} \end{bmatrix}.$$

A quadratic model in $\mathbf{p}$ is obtained by limiting the Taylor series in (4) to the first three terms. The optimal value $\mathbf{p}_k$ for $\tilde{f}$ is then given by

$$\mathbf{p}_k = \arg\min_{\mathbf{p}} \tilde{f}(\mathbf{p}) \qquad \text{with} \qquad \tilde{f}(\mathbf{p}) = f(\mathbf{z}_k) + \mathbf{p}^{\mathrm{T}}\mathbf{g}_k + \frac{1}{2}\mathbf{p}^{\mathrm{T}}\mathbf{H}_k\mathbf{p}.$$

To compute $\mathbf{p}_k$ the gradient of $\tilde{f}$, i.e., $\nabla_{\mathbf{p}}\tilde{f} = \mathbf{g}_k + \mathbf{H}\mathbf{p}$, is set to zero, which results in the linear system

$$\mathbf{H}_k\mathbf{p}_k = -\mathbf{g}_k. \tag{5}$$

The computed step $\mathbf{p}_k$ is called the Newton step.

Unfortunately, the Hessian $\mathbf{H}_k$ is often difficult or expensive to compute explicitly. Therefore, $\mathbf{H}_k$ is often approximated which results in quasi-Newton (qN) type algorithms such as nonlinear conjugate gradients (NCG) and BFGS, and Gauss–Newton (GN) type algorithms. Some examples of such approximations are:

- For gradient or steepest descent, $\mathbf{H}_k = \mathbf{I}$ and the step direction is simply the direction of the steepest slope.
- For NCG, $\mathbf{H}_k = \mathbf{I} - \gamma\mathbf{p}_{k-1}\boldsymbol{\delta}^{\mathrm{T}}$, i.e., the identity plus a rank-1 correction. The values γ and $\boldsymbol{\delta}$ depend on the current and previous gradient, i.e., $\mathbf{g}_{k-1}$ and $\mathbf{g}_k$, and the previous step $\mathbf{p}_{k-1}$.
- For BFGS, $\mathbf{H}_k = \mathbf{H}_{k-1} + \mathbf{U}_{k-1} + \mathbf{V}_{k-1}$ in which $\mathbf{U}_{k-1}$ and $\mathbf{V}_{k-1}$ are symmetric rank-1 matrices constructed using the previous values for the gradient, for the approximation $\mathbf{H}_{k-1}$, and for the update of $\mathbf{z}$.

- For GN, $\mathbf{H}_k = \mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k$ with $\mathbf{J}_k$ the Jacobian matrix, and $\mathbf{H}_k$ is then called the Gramian (of the Jacobian).
- For Levenberg–Marquardt (LM), $\mathbf{H}_k = \mathbf{J}_k^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{I}$ with $\mathbf{J}_k$ the Jacobian matrix and $\lambda \geq 0$ a chosen constant.

Using the Newton step, quadratic convergence can be achieved near local optima under certain conditions, while the convergence is merely linear for gradient descent. In the case of qN algorithms, the convergence improves to superlinear. Quadratic convergence can again be achieved using GN and LM, as the Gramian often approximates the exact Hessian well for many optimization problems. Example 2 and example 3 illustrate the differences in convergence speed. In combination with trust region approaches, it can be shown that GN is globally convergent, meaning that the algorithm will converge to a (local) minimum from any initial value $\mathbf{z}_0$ thanks to the fact that $\mathbf{H}_k$ is positive semidefinite which ensures that $\mathbf{p}_k$ is a descent direction. (In fact, GN with a trust region is related to LM, as it sets λ indirectly through the trust region radius Δ [51].) Moreover, we show in section 3 that GN allows the multilinear structure in CPD problems to be exploited elegantly. Because of its good properties, the remainder of this chapter is focused on GN type algorithms.

Example 2 (linear versus quadratic convergence): Consider a tensor $\mathcal{T} = \llbracket \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)} \rrbracket$ of size $250 \times 250 \times 250$ with rank 10. The random factor matrices are chosen to create a rather difficult problem and are constructed such that each column has norm one and the inner product with the other columns is 0.8, i.e., $\|\mathbf{a}_r^{(n)}\|_{\mathrm{F}} = 1$ and $\mathbf{a}_r^{(n)^{\mathrm{T}}}\mathbf{a}_s^{(n)} = 0.8$ for $r \neq s$ and $n = 1, 2, 3$. Starting from a random initialization, GN (`cpd_nls`) and ALS (`cpd_als`) from Tensorlab [53] are used to compute the optimum. The former algorithm is able to achieve up to quadratic convergence near a local optimum, while the latter achieves up to linear convergence. A typical convergence profile can be seen in Figure 2 and in the table below: while ALS needs about 100 iterations to make the error approximately 100 times smaller, the number of correct digits doubles every iteration using GN until the machine precision limits further improvement at iteration 37.

GN		ALS	
k	f	k	f
34	$2.23 \cdot 10^{-6}$	700	$1.41 \cdot 10^{-15}$
35	$5.08 \cdot 10^{-12}$	800	$2.22 \cdot 10^{-17}$
36	$5.42 \cdot 10^{-24}$	900	$3.09 \cdot 10^{-19}$
37	$4.60 \cdot 10^{-31}$	1000	$4.89 \cdot 10^{-21}$

Example 3 (second-order information and noise): Consider a rank-10 tensor $\mathcal{T} = \llbracket \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)} \rrbracket$ of size $200 \times 200 \times 200$ in which the random factor matrices have norm one columns and inner product 0.8, i.e., $\|\mathbf{a}_r^{(n)}\|_{\mathrm{F}} = 1$ and $\mathbf{a}_r^{(n)^{\mathrm{T}}}\mathbf{a}_s^{(n)} = 0.8$ for $r \neq s$. Noise is added such that the SNR is 20 dB. Starting from a random initialization, GN (`cpd_nls`) and ALS (`cpd_als`) are again used to

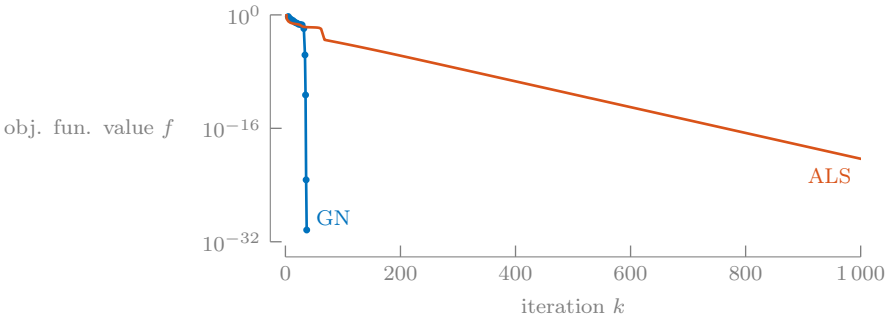


Figure 2: As the GN algorithm converges quadratically near a local minimum, only a few iterations are required to converge to the optimum up to machine precision. ALS, which converges linearly, requires many iterations to obtain a similar precision.

compute a rank-10 approximation. In Figure 3, the maximal relative error on the factor matrices, i.e.,

$$E_{\text{CPD}} = \max_n \frac{\|\mathbf{A}^{(n)} - \hat{\mathbf{A}}^{(n)}\|_{\text{F}}}{\left|\left|\mathbf{A}^{(n)}\right|\right|_{\text{F}}}$$

is reported. (Scaling and permutation indeterminacies are assumed to be resolved.) Initially, ALS improves the function value f faster compared to GN, but ALS requires many iterations with very small changes in f to reduce E_{CPD} to the same level as the GN algorithm. This result also illustrates that one should avoid stopping too early when using ALS. Fast convergence in terms of E_{CPD} is again observed for GN.

2.3. Solving $\mathbf{Hp} = -\mathbf{g}$

Each iteration of the optimization algorithm, the solution $\mathbf{p}$ of the system $\mathbf{Hp} = -\mathbf{g}$ is required. Directly solving the system by simply taking the inverse of $\mathbf{H}$ may not be a good idea for a number of numerical reasons and is not even possible if $\mathbf{H}$ is not invertible. This is for example the case when computing a CPD via GN, as $\mathbf{H} = \mathbf{J}^{\mathsf{T}}\mathbf{J}$ has $2R$ zero eigenvalues due to the scaling indeterminacy. To mitigate this, the pseudoinverse

$$\mathbf{p} = -\mathbf{H}^{\dagger}\mathbf{g}$$

or the LDL factorization [54] can be used. (The latter can only be used if $\mathbf{H}$ is symmetric, which is the case for GN.)

The direct techniques outlined above work fine if the number of variables is low (typically a few hundreds), but become expensive for large-scale problems. Therefore, rather than a direct method, an inexact, iterative solver can be used, e.g. conjugate gradients (CG). Starting from an initial guess $\mathbf{p}^{(0)}$, e.g., the Cauchy point [51], the solution of $\mathbf{Hp} = -\mathbf{g}$ is computed using only one matrix-vector product of the form $\mathbf{y}^{(l)} = \mathbf{H}\mathbf{x}^{(l-1)}$ in each CG iteration. In the lth iteration, the guess $\mathbf{p}^{(l)}$ and $\mathbf{x}^{(l)}$ are both updated using a linear combination of their respective previous value and $\mathbf{y}^{(l)}$.

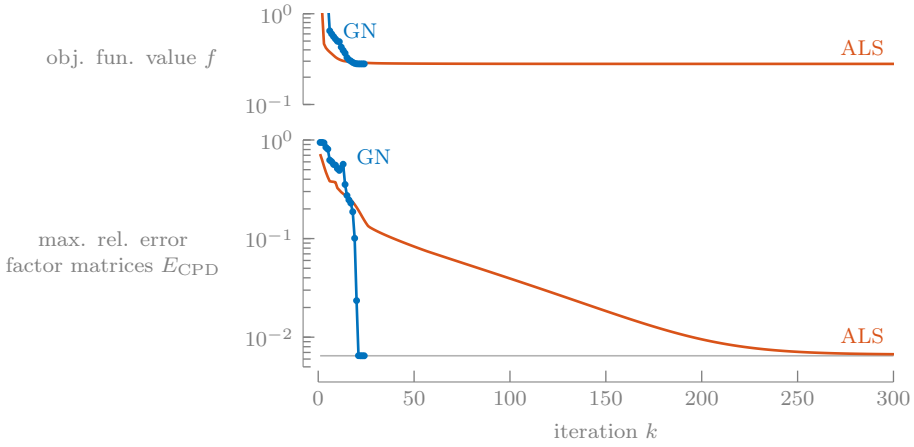


Figure 3: While the improvement in objective function value levels off after a few iterations because of the perturbations by noise (SNR is 20 dB) for both GN and ALS, the error on the factor matrices can still be improved. Thanks to the use of (approximate) second-order information in GN, an accurate solution is found quickly, while many iterations with almost no improvement in the objective function are required for ALS. The results are shown for a rank-10 tensor with correlated factor matrices in alle modes.

(See, e.g. [55] for more details on the CG algorithm). To compute the product $\mathbf{Hx}$, it is not necessary to construct $\mathbf{H}$ explicitly, allowing the structure to be exploited. As shown in section 3, the product $\mathbf{Hx}$ can be computed cheaply in the case of a CPD. The CG algorithm requires it_{CG} iterations to achieve a certain relative error, e.g., 10^{-6} or 10^{-8}. To improve the convergence and therefore reduce it_{CG} and the number of required matrix-vector products $\mathbf{Hx}$, a preconditioner $\mathbf{M}$ is often used, and the following system is solved instead:

$$\mathbf{M}^{-1}\mathbf{Hp} = -\mathbf{M}^{-1}\mathbf{g}.$$

If the eigenvalues of $\mathbf{M}^{-1}\mathbf{H}$ are more clustered than those of $\mathbf{H}$, the preconditioned CG (PCG) converges faster [55]. Ideally, $\mathbf{M}^{-1}$ is also cheap to apply.

Example 4: Consider a rank-5 tensor $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ of size $40 \times 40 \times 40$ with highly correlated factor vectors and perturbed by noise such that the SNR is 20 dB. Suppose the current guess is $\mathbf{z}_{k-1}$. The step $\mathbf{p}_k$ is computed from $\mathbf{Hp}_k = -\mathbf{g}$ using the pseudoinverse, CG and PCG with a block-Jacobi preconditioner (see subsection 3.2.2). As shown in Figure 4, fewer iterations and less time are needed when using PCG. (A stopping tolerance of 10^{-8} on the relative residual is used for the latter two algorithms.)

3. Canonical polyadic decomposition

Throughout this chapter, nearly all concepts are derived for Gauss–Newton type algorithms using a trust region. Apart from the favorable convergence properties, a big

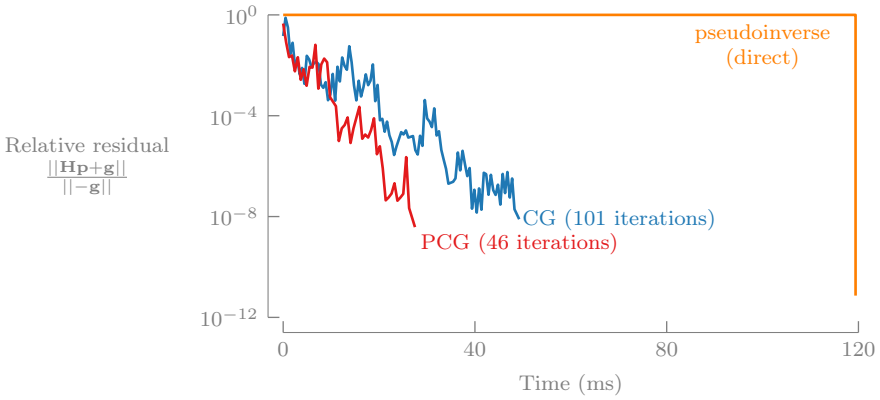


Figure 4: While the direct method finds the solution of $\mathbf{Hp} = -\mathbf{g}$ in a single step, it is outperformed by the iterative methods in terms of time. The preconditioning in PCG reduces the number of iterations needed as well as overall cost, if it is cheap to apply. The Gramian $\mathbf{H}$ and gradient $\mathbf{g}$ are computed for a random GN iteration when computing a rank-5 CPD of $40 \times 40 \times 40$ tensor with highly correlated columns. The stopping tolerance is set to 10^{-8}.

advantage is that it allows the multilinear structure of a CPD to be exploited easily. In combination with an inexact solver for the system (5), an efficient algorithm can be derived, as shown in this section. (Note that one typically uses an optimization framework, e.g. the complex optimization framework [26, 56] which is built-in into Tensorlab [53]. Hence, only relevant expressions for the objective function, gradient, Gramian-vector product and preconditioner are required.)

Concretely, the following objective function is used:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f \quad \text{with} \quad f = \frac{1}{2} \| \underbrace{[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}}_{\mathcal{R}} \|_{\mathrm{F}}^2. \tag{6}$$

This is a quadratic objective function in the (vectorized) residual $\mathbf{r} = \mathrm{vec}\,(\mathcal{R})$:

$$f = \frac{1}{2} \mathbf{r}^{\mathrm{T}} \mathbf{r},$$

with $\mathbf{r}$ a multilinear function in the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. This formulation proves useful when deriving the gradient and the Gramian of the Jacobian.

After a brief overview of important concepts in multilinear algebra and derivatives in subsection 3.1, the ingredients for a GN type CPD algorithm are derived in subsection 3.2. In subsection 3.3, we show how an ALS type algorithm fits in the optimization framework. Finally, more general objective functions are discussed in subsection 3.4.

3.1. Intermezzo: multilinear algebra

Via multilinear algebra, the gradient and Gramian-vector products required when computing the optimum of (6) using GN can be derived without resorting to element-wise expressions. A brief overview of the most important identities needed are listed here.

The following identities involving Kronecker and Khatri–Rao products are used, assuming matrices of compatible dimensions:

$$(\mathbf{A} \otimes \mathbf{B})\text{vec}\,(\mathbf{X}) = \mathbf{B}\mathbf{X}\mathbf{A}^{\mathrm{T}}, \tag{7}$$
$$(\mathbf{A} \odot \mathbf{B})^{\mathrm{T}}(\mathbf{C} \odot \mathbf{D}) = (\mathbf{A}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{D}), \tag{8}$$
$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}. \tag{9}$$

To use these identities in combination with unfolded tensors, we define the permutation matrices $\mathbf{\Pi}^{(n)} \in \{0,1\}^{IJK \times IJK}$ which permute the nth mode to the first mode:

$$\mathbf{\Pi}^{(1)} : (i,j,k) \mapsto (i,j,k)$$
$$\mathbf{\Pi}^{(2)} : (i,j,k) \mapsto (j,i,k)$$
$$\mathbf{\Pi}^{(3)} : (i,j,k) \mapsto (k,i,j).$$

The inverse operation is $\mathbf{\Pi}^{(n)^{\mathrm{T}}}$ and moves the first mode to the nth mode. Note that $\mathbf{\Pi}^{(n)}$ is a purely mathematical concept and is never constructed explicitly.

Example 5: Given $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, we have

$$\mathbf{P}^{(3)}\text{vec}\,([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) = \text{vec}\,([\![\mathbf{C}, \mathbf{A}, \mathbf{B}]\!])$$
$$\mathbf{P}^{(2)}\text{vec}\,(\mathcal{T}) = \text{vec}\left(\mathbf{T}_{(2)}\right)$$
$$\mathbf{P}^{(2)^{\mathrm{T}}}\text{vec}\,([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]) = \text{vec}\,([\![\mathbf{B}, \mathbf{A}, \mathbf{C}]\!]).$$

Example 6 (MTKRPROD): A common operation when computing a CPD is the "matricized tensor times Khatri–Rao product" (MTKRPROD or MTTKRP). For example, for the first mode, we have

$$\mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B}).$$

Using identity (7), this can be rewritten as

$$\text{vec}\left(\mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B})\right) = ((\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} \otimes \mathbf{I})\text{vec}\,(\mathcal{T}).$$

Additionally, using permutation matrices, we have for the second mode

$$\text{vec}\left(\mathbf{T}_{(2)}(\mathbf{C} \odot \mathbf{A})\right) = ((\mathbf{C} \odot \mathbf{A})^{\mathrm{T}} \otimes \mathbf{I})\mathbf{\Pi}^{(2)}\text{vec}\,(\mathcal{T}).$$

Similarly, for a CPD, we have

$$\text{vec}\left(\mathbf{A}\,(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}}\right) = \mathbf{\Pi}^{(2)^{\mathrm{T}}}((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J)\text{vec}\,(\mathbf{B}).$$

Finally, it is worthwhile to refresh some matrix derivatives. For matrices $\mathbf{A}$ and $\mathbf{B}$ with dimensions $I \times J$ and $J \times K$, respectively, we have

$$\frac{\partial \text{vec}\,(\mathbf{A}\mathbf{B})}{\partial \text{vec}\,(\mathbf{A})} = \mathbf{B}^{\mathrm{T}} \otimes \mathbf{I}_I \qquad \frac{\partial \text{vec}\,(\mathbf{A}\mathbf{B})}{\partial \text{vec}\,(\mathbf{B})} = \mathbf{I}_K \otimes \mathbf{A}. \tag{10}$$

3.2. Gauss–Newton type algorithms

An inexact GN algorithm for computing a CPD using objective function (6) is derived here. (The algorithm itself is presented in [38].) As standard trust region techniques such as the dogleg method (see [51]) are used, we focus on solving the linear system (5) using PCG, which requires the computation of the gradient and Gramian-vector products $\mathbf{Hx}$ as shown in subsection 2.3. This subsection is concluded by a brief discussion of a preconditioner.

3.2.1. Gradient

As the objective function f in (6) is a quadratic function in the residual $\mathbf{r}$, which is in turn a multilinear function of the variables $\mathbf{z}$, the chain rule can be used to derive the gradient. Recall that the variables are $\mathbf{z} = \left[\operatorname{vec}(\mathbf{A}); \operatorname{vec}(\mathbf{B}); \operatorname{vec}(\mathbf{C})\right]$. The gradient can be partitioned accordingly, i.e., $\mathbf{g} = \left[\mathbf{g}_{\mathrm{A}}; \mathbf{g}_{\mathrm{B}}; \mathbf{g}_{\mathrm{C}}\right]$. The derivative w.r.t. $\operatorname{vec}(\mathbf{A})$ is then given by

$$\mathbf{g}_{\mathrm{A}} = \frac{1}{2}\frac{\partial \mathbf{r}^{\mathrm{T}}\mathbf{r}}{\partial \operatorname{vec}(\mathbf{A})} = \left(\frac{\partial \mathbf{r}}{\partial \operatorname{vec}(\mathbf{A})}\right)^{\mathrm{T}} \mathbf{r} = \mathbf{J}_{\mathrm{A}}^{\mathrm{T}}\mathbf{r}. \tag{11}$$

The matrix $\mathbf{J}_{\mathrm{A}}$ is called the Jacobian matrix. The ith row in this Jacobian contains the derivative of the ith residual r_i, $i = 1, \ldots, IJK$, w.r.t. all entries in $\mathbf{A}$, i.e.,

$$\mathbf{J}_{\mathrm{A}} = \begin{bmatrix} \frac{\partial r_1}{\partial a_{11}} & \frac{\partial r_1}{\partial a_{21}} & \cdots & \frac{\partial r_2}{\partial a_{IR}} \\ \frac{\partial r_2}{\partial a_{11}} & \frac{\partial r_2}{\partial a_{21}} & \cdots & \frac{\partial r_2}{\partial a_{IR}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial r_{IJK}}{\partial a_{11}} & \frac{\partial r_{IJK}}{\partial a_{21}} & \cdots & \frac{\partial r_{IJK}}{\partial a_{IR}} \end{bmatrix}.$$

In the case of a CPD, the Jacobian can be computed easily using (10):

$$\mathbf{J}_{\mathrm{A}} = \frac{\partial \mathbf{r}}{\partial \operatorname{vec}(\mathbf{A})} = \frac{\partial \operatorname{vec}\left(\mathbf{A}(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} - \mathbf{T}_{(1)}\right)}{\partial \operatorname{vec}(\mathbf{A})} = (\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I. \tag{12}$$

Similarly, the Jacobians w.r.t. the other factor matrices can be computed as

$$\mathbf{J}_{\mathrm{B}} = \mathbf{\Pi}^{(2)^{\mathrm{T}}}\left((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_J\right) \tag{13}$$

$$\mathbf{J}_{\mathrm{C}} = \mathbf{\Pi}^{(3)^{\mathrm{T}}}\left((\mathbf{B} \odot \mathbf{A}) \otimes \mathbf{I}_K\right). \tag{14}$$

Finally, by combining (11) and (12)–(14) and exploiting the multilinear identities as in example 6, we find

$$\mathbf{g}_{\mathrm{A}} = \operatorname{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right), \tag{15}$$

$$\mathbf{g}_{\mathrm{B}} = \operatorname{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A})\right), \tag{16}$$

$$\mathbf{g}_{\mathrm{C}} = \operatorname{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A})\right). \tag{17}$$

These MTKRPROD can be computed efficiently without explicitly constructing the Khatri–Rao products $\mathbf{C} \odot \mathbf{B}$ and so on. Various implementations emerged that avoid permuting the tensor in memory [57, 58], that avoid communication [59], that reuse intermediate

results [58], that exploit sparsity by only computing rows of the Khatri–Rao product corresponding to nonzero entries [60, 61], or that exploit structure in the tensor such as tensor train or Hankel structure [60, 62].

3.2.2. Gramian-vector products

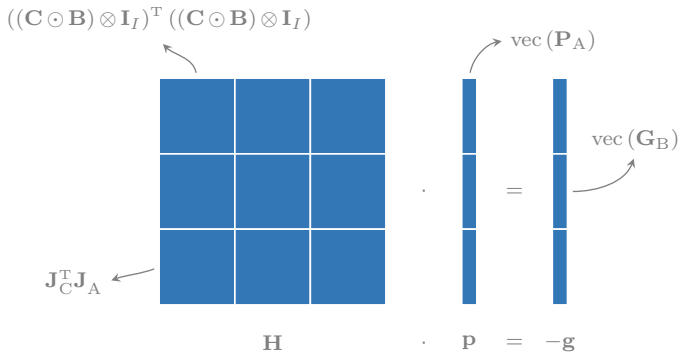


Figure 5: Grafical representation of the system $\mathbf{Hp} = -\mathbf{g}$, which is solved for $\mathbf{p}$ every iteration of the GN algorithm.

When using (preconditioned) CG, the system $\mathbf{Hp} = -\mathbf{g}$ is solved iteratively: each CG iteration a Gramian-vector product $\mathbf{y} = \mathbf{Hx}$ is computed; see subsection 2.3. The Gramian is given by the inner product of the Jacobian with itself, hence

$$\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{J} \quad \text{with} \quad \mathbf{J} = \begin{bmatrix} \mathbf{J}_{\mathrm{A}} & \mathbf{J}_{\mathrm{B}} & \mathbf{J}_{\mathrm{C}} \end{bmatrix},$$

in which $\mathbf{J}_{\mathrm{A}}$, $\mathbf{J}_{\mathrm{B}}$ and $\mathbf{J}_{\mathrm{C}}$ are defined as in (12)–(14). The block structure of $\mathbf{J}$ results in a block matrix $\mathbf{H}$ as can be seen in Figure 5, which gives a graphical overview of the system. If the vectors $\mathbf{y}$ and $\mathbf{x}$ are partitioned according to factor matrices as well, i.e.,

$$\begin{aligned} \mathbf{y} &= \begin{bmatrix} \mathbf{y}_{\mathrm{A}}; & \mathbf{y}_{\mathrm{B}}; & \mathbf{y}_{\mathrm{C}} \end{bmatrix} \\ \mathbf{x} &= \begin{bmatrix} \mathrm{vec}\,(\mathbf{X}_{\mathrm{A}})\,; & \mathrm{vec}\,(\mathbf{X}_{\mathrm{B}})\,; & \mathrm{vec}\,(\mathbf{X}_{\mathrm{C}}) \end{bmatrix}, \end{aligned}$$

the products $\mathbf{y}_{\mathrm{A}}$, $\mathbf{y}_{\mathrm{B}}$ and $\mathbf{y}_{\mathrm{C}}$ can be computed as

$$\mathbf{y}_{\mathrm{A}} = \mathbf{H}_{\mathrm{AA}}\mathrm{vec}\,(\mathbf{X}_{\mathrm{A}}) + \mathbf{H}_{\mathrm{AB}}\mathrm{vec}\,(\mathbf{X}_{\mathrm{B}}) + \mathbf{H}_{\mathrm{AC}}\mathrm{vec}\,(\mathbf{X}_{\mathrm{C}}),$$

and so on, with $\mathbf{H}_{\mathrm{AA}} = \mathbf{J}_{\mathrm{A}}^{\mathrm{T}}\mathbf{J}_{\mathrm{A}}$ etc. By exploiting the multilinear identities from subsection 3.1, efficient expressions can be derived easily, e.g.,

$$\begin{aligned} \mathbf{H}_{\mathrm{AA}}\mathrm{vec}\,(\mathbf{X}_{\mathrm{A}}) &= \mathrm{vec}\,(\mathbf{X}_{\mathrm{A}}((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B}))), \\ \mathbf{H}_{\mathrm{BC}}\mathrm{vec}\,(\mathbf{X}_{\mathrm{C}}) &= \mathrm{vec}\,(\mathbf{B}((\mathbf{X}_{\mathrm{C}}^{\mathrm{T}}\mathbf{C}) * (\mathbf{A}^{\mathrm{T}}\mathbf{A}))). \end{aligned}$$

Note that the complexity of computing $\mathbf{y}$ is low as only inner products between matrices, and matrix products with $R \times R$ matrices are needed. The complexity therefore is a function of the sum of the tensor dimensions, i.e., $\mathcal{O}(I + J + K)$, which is usually much

lower than the complexity of the gradient and function evaluation which both depend on the total number of entries in the tensor, i.e., $\mathcal{O}(IJK)$.

Example 7: The expression for $\mathbf{H}_{\mathrm{BC}}\mathrm{vec}(\mathbf{X}_{\mathrm{C}})$ can be derived by applying identities (7) and (8) and properties involving permutations matrices:

$$\begin{aligned}
\mathbf{H}_{\mathrm{BC}}\mathrm{vec}(\mathbf{X}_{\mathrm{C}}) &= \mathbf{J}_{\mathrm{B}}^{\mathrm{T}} \cdot \boldsymbol{\Pi}^{(3)^{\mathrm{T}}}((\mathbf{B} \odot \mathbf{A}) \otimes \mathbf{I}_K)\mathrm{vec}(\mathbf{X}_{\mathrm{C}}) \\
&= \mathbf{J}_{\mathrm{B}}^{\mathrm{T}} \cdot \boldsymbol{\Pi}^{(3)^{\mathrm{T}}}\mathrm{vec}(\mathbf{X}_{\mathrm{C}}(\mathbf{B} \odot \mathbf{A})^{\mathrm{T}}) \\
&= ((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J)^{\mathrm{T}}\boldsymbol{\Pi}^{(2)} \cdot \mathrm{vec}(\mathbf{A}(\mathbf{X}_{\mathrm{C}} \odot \mathbf{B})^{\mathrm{T}}) \\
&= ((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J)^{\mathrm{T}}\mathrm{vec}(\mathbf{B}(\mathbf{X}_{\mathrm{C}} \odot \mathbf{A})^{\mathrm{T}}) \\
&= \mathrm{vec}(\mathbf{B}(\mathbf{X}_{\mathrm{C}} \odot \mathbf{A})^{\mathrm{T}}(\mathbf{C} \odot \mathbf{A})) \\
&= \mathrm{vec}(\mathbf{B}((\mathbf{X}_{\mathrm{C}}^{\mathrm{T}}\mathbf{C}) * (\mathbf{A}^{\mathrm{T}}\mathbf{A}))).
\end{aligned}$$

3.2.3. Preconditioner

By using a preconditioner, the number of CG iterations, and hence the number of Gramian-vector products, can be reduced significantly. Instead of (5), the modified system

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

is solved instead, in which $\mathbf{M}^{-1}$ is chosen such that the eigenvalues of $\mathbf{M}^{-1}\mathbf{H}$ are more clustered than those of $\mathbf{H}$. As shown in [38], a block-Jacobi preconditioner is effective for the CPD: $\mathbf{M}$ is then a block diagonal approximation of $\mathbf{H}$:

$$\mathbf{M} = \begin{bmatrix} \mathbf{W}^{(1)} \otimes \mathbf{I}_I & & \\ & \mathbf{W}^{(2)} \otimes \mathbf{I}_J & \\ & & \mathbf{W}^{(3)} \otimes \mathbf{I}_K \end{bmatrix},$$

with $\mathbf{W}^{(1)} = (\mathbf{B}^{\mathrm{T}}\mathbf{B}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})$, $\mathbf{W}^{(2)} = (\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})$ and $\mathbf{W}^{(3)} = (\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})$. Applying the inverse of $\mathbf{M}$ is cheap as it only requires the inverse of typically small $R \times R$ matrices $\mathbf{W}^{(n)}$ thanks to identity (9). Note the similarities with the ALS results in subsection 3.3: the multiplication $\mathbf{M}^{-1}$ can be seen as a parallel ALS step in which all linear systems are solved simultaneously, rather than consecutively [38].

3.3. Alternating least squares

Another approach to solving (6) is to use alternating least squares (ALS), which is a popular approach thanks to its simplicity. By fixing all but one factor matrix, a linear least squares objective function is obtained. Hence, each iteration, three LS problems

are solved consecutively:

$$\begin{aligned}
\mathbf{A}_k &= \arg\min_{\mathbf{A}} \frac{1}{2} \left|\left|\mathbf{A}(\mathbf{C}_{k-1} \odot \mathbf{B}_{k-1})^{\mathrm{T}} - \mathbf{T}_{(1)}\right|\right|_{\mathrm{F}}^2 \\
\mathbf{B}_k &= \arg\min_{\mathbf{B}} \frac{1}{2} \left|\left|\mathbf{B}(\mathbf{C}_{k-1} \odot \mathbf{A}_k)^{\mathrm{T}} - \mathbf{T}_{(2)}\right|\right|_{\mathrm{F}}^2 \\
\mathbf{C}_k &= \arg\min_{\mathbf{C}} \frac{1}{2} \left|\left|\mathbf{C}(\mathbf{B}_k \odot \mathbf{A}_k)^{\mathrm{T}} - \mathbf{T}_{(3)}\right|\right|_{\mathrm{F}}^2 .
\end{aligned}$$

(Hence the updated variables from the previous LS substep are used.) It can be verified that the solution of the first LS step is given by

$$\mathbf{A} = \mathbf{T}_{(1)} \left((\mathbf{C} \odot \mathbf{B})^{\mathrm{T}}\right)^{\dagger} .$$

(We drop the subscripts $k-1$ for simplicity of notation.) As the pseudoinverse of a tall matrix $\mathbf{X}$ with full column rank can be written as $\mathbf{X}^{\dagger} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}$, the solution can be rewritten using (8) as

$$\begin{aligned}
\mathbf{A} &= \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B}) \left((\mathbf{C} \odot \mathbf{B})^{\mathrm{T}}(\mathbf{C} \odot \mathbf{B})\right)^{-1} \\
&= \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B}) \left((\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})\right)^{-1} ,
\end{aligned} \tag{18}$$

which is the classical ALS result. The matrix $\mathbf{W}^{(1)} = (\mathbf{C}^{\mathrm{T}}\mathbf{C}) * (\mathbf{B}^{\mathrm{T}}\mathbf{B})$ is an $R \times R$ matrix and can be inverted cheaply for low-rank problems.

It is fruitful to see the solution of the LS systems in the larger optimization framework, as illustrated in Figure 6. (We again drop the subscript $k-1$ for $\mathbf{B}$ and $\mathbf{C}$, and set $\mathbf{W} = \mathbf{W}^{(1)}$.) Each subproblem, a step $\mathbf{p}_k = \mathrm{vec}\,(\mathbf{P}_k)$ for updating $\mathbf{A}$ is computed from the system

$$(\mathbf{W} \otimes \mathbf{I}_I)\mathrm{vec}\,(\mathbf{P}_k) = -\mathrm{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right) ,$$

in which $\mathbf{R}_{(1)} = (\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} - \mathbf{T}_{(1)}$ is the unfolded residual. The step $\mathbf{P}_k$ is found by inverting $(\mathbf{W} \otimes \mathbf{I}_I)$, which is exactly $\mathbf{H}_{\mathrm{AA}}$ from subsection 3.2.2, using identities (7) and (9). The variables are then updated as

$$\begin{aligned}
\mathbf{A}_k &= \mathbf{A}_{k-1} + \alpha_k \mathbf{P}_k \\
&= \mathbf{A}_{k-1} - \alpha_k \left(\mathbf{A}_{k-1}(\mathbf{C} \odot \mathbf{B})^{\mathrm{T}} - \mathbf{T}_{(1)}\right)(\mathbf{C} \odot \mathbf{B})\mathbf{W}^{-1} \\
&= \mathbf{A}_{k-1} - \alpha_k \mathbf{A}_{k-1}\mathbf{W}\mathbf{W}^{-1} + \alpha_k \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B})\mathbf{W}^{-1} \\
&= (1 - \alpha_k)\mathbf{A}_{k-1} + \alpha_k \mathbf{T}_{(1)}(\mathbf{C} \odot \mathbf{B})\mathbf{W}^{-1} .
\end{aligned}$$

Setting $\alpha_k = 1$ gives the result in (18). However, to somewhat mitigate the slow convergence of ALS for ill-conditioned problems, exact or approximate line search methods selecting different values for α_k have been discussed in [11, 27, 63, 64].

3.4. More general objective functions

The Gauss–Newton algorithm is defined for the Euclidean distance, i.e., for the minimization of the least squares error. However, depending on the assumptions made for

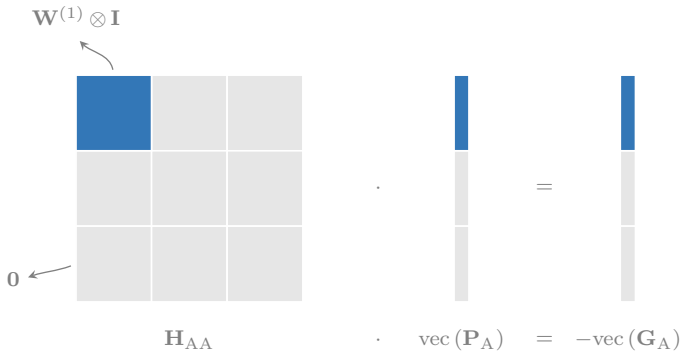


Figure 6: Every nth ALS subiteration, the system $\left(\mathbf{W}^{(n)} \otimes \mathbf{I}\right) \mathrm{vec}\,(\mathbf{P}_n) = -\mathrm{vec}\,(\mathbf{G}_n)$, with $n = 1, 2, 3$, is solved. (We define $\mathbf{P}_1 = \mathbf{P}_{\mathrm{A}}$, $\mathbf{P}_2 = \mathbf{P}_{\mathrm{B}}$ and $\mathbf{P}_3 = \mathbf{P}_{\mathrm{C}}$, and use a similar definiton for $\mathbf{G}_n$.) The subproblem for $n = 1$ is shown.

the data, other distance measures or divergences can be more appropriate. For example, in the case of count data, the Kullback–Leibler divergence can be more appropriate as its minimizer coincides with the minimizer of the negative log likelihood function for Poisson distributed data [41]. Algorithms alternating between variables and/or using multiplicative updates have been presented for generalized divergences including Tweedie and alpha and beta divergences; see, e.g., [41, 42, 43, 45, 43, 46, 47, 48]. Here, we extend the ideas of GN to more general divergences such that a positive semidefinite Hessian approximation is used and the multilinear structure is exploited. Additional constraints such as nonnegativity can be handled using the techniques from section 4.

Table 1: The (element-wise) objective function f_i and its derivatives for some divergences.

	f_i	$\frac{\mathrm{d}f_i}{\mathrm{d}m_i}$	$\frac{\mathrm{d}^2 f_i}{\mathrm{d}m_i^2}$
Euclidean	$\frac{1}{2}(m_i - t_i)^2$	$m_i - t_i$	1
Kullback–Leibler	$m_i - t_i \log m_i + t_i \log t_i - t_i$	$\frac{m_i - t_i}{m_i}$	$\frac{t_i}{m_i^2}$
Itakura–Saito	$\frac{t_i}{m_i} - \log t_i + \log m_i - 1$	$\frac{m_i - t_i}{m_i^2}$	$\frac{2t_i - m_i}{m_i^3}$

The least squares objective function (1) can be rewritten as

$$\min_{\mathbf{z}} \sum_{i=1}^{N} f_i(m_i(\mathbf{z})) \qquad \text{with} \qquad f_i = \frac{1}{2}(m_i(\mathbf{z}) - t_i)^2,$$

with N the number of data points, which is equal to the number of tensor entries in this case, i.e., $N = IJK$. If we replace the Euclidean distance with another divergence, only the function f_i changes. Some examples are given in Table 1. As explained in subsection 2.2, a second-order Taylor series expansion $\tilde{f}(\mathbf{p})$ at $\mathbf{z}_k$ can be constructed as

$$f(\mathbf{z}) \approx \tilde{f}(\mathbf{p}) = f(\mathbf{z}_k) + \mathbf{p}^{\mathrm{T}} \cdot \nabla_{\mathbf{z}} f(\mathbf{z}_k) + \frac{1}{2}\mathbf{p}^{\mathrm{T}} \cdot \nabla_{\mathbf{z}}^2 f(\mathbf{z}_k) \cdot \mathbf{p}.$$

Using the chain rule, the gradient $\mathbf{g}$ is then given by

$$\mathbf{g} = \nabla_{\mathbf{z}} f = \sum_{i=1}^{N} \frac{\partial f_i}{\partial m_i} \nabla_{\mathbf{z}} m_i, \tag{19}$$

in which $\nabla_{\mathbf{z}} m_i$ is exactly the ith (transposed) row of the Jacobian matrix $\mathbf{J}$ as derived in subsection 3.2.1. Hence, when collecting all partial derivatives w.r.t. the model m_i in a vector $\mathbf{j}_{\mathrm{m}}$, the gradient is given by

$$\mathbf{g} = \mathbf{J}^{\mathrm{T}} \mathbf{j}_{\mathrm{m}}.$$

As shown in Table 1, the vector $\mathbf{j}_{\mathrm{m}}$ is simply the vectorized residual for least squares problems. Even though the expressions are more involved for other divergences, as can be seen in Table 1, the construction of $\mathbf{j}_m$ is the only change, which means efficient implementations for the MTKRPROD can be reused.

The Hessian is computed by taking the derivative of the gradient in (19) w.r.t. the variables $\mathbf{z}$, again using the chain rule, hence

$$\nabla_{\mathbf{z}}^2 f = \sum_{i=1}^{N} \nabla_{\mathbf{z}} m_i \cdot \frac{\mathrm{d}^2 f_i}{\mathrm{d} m_i^2} \cdot (\nabla_{\mathbf{z}} m_i)^{\mathrm{T}} + \nabla_{\mathbf{z}}^2 m_i \cdot \frac{\mathrm{d} f_i}{\mathrm{d} m_i}.$$

In the case of GN, the second term is assumed to be small near the optimum and neglecting this term still gives a good approximation of the Hessian. This is also a reasonable assumption for the other divergences: $\sum_{i=1}^{N} \nabla_{\mathbf{z}}^2 m_i$ depends only on the model and is a sparse matrix as shown in [38], and $\frac{\mathrm{d} f_i}{\mathrm{d} m_i}$ becomes zero in the optimum, i.e., if $m_i = t_i$ as illustrated in Table 1. Therefore, the Hessian can be approximated as

$$\nabla_{\mathbf{z}}^2 f \approx \mathbf{H} = \mathbf{J}^{\mathrm{T}} \mathbf{D} \mathbf{J},$$

in which $\mathbf{D}$ is an $IJK \times IJK$ diagonal matrix with the derivatives $\frac{\mathrm{d}^2 f_i}{\mathrm{d} m_i^2}$ as entries on the diagonal. Compared to the least squares problem, which has $\mathbf{D} = \mathbf{I}_{IJK}$, only the entries on the diagonal of $\mathbf{D}$, which are given in Table 1, are different. As $\mathbf{H}$ is again a positive semidefinite approximation to the Hessian, favorable convergence conditions apply here as well, especially in combination with globalization strategies such as trust regions.

The multilinear structure of the CPD can again be exploited when computing the step direction from the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$. The computation of the gradient requires three MTKRPROD operations using the tensor $\tilde{\mathcal{R}} = \operatorname{unvec}(\mathbf{j}_{\mathrm{m}})$ instead of the residual tensor $\mathcal{R}$. For example, instead of (15) we have

$$\mathbf{g}_{\mathrm{A}} = \operatorname{vec}\left(\tilde{\mathbf{R}}_{(1)} (\mathbf{C} \odot \mathbf{B})\right).$$

The expressions for (16) and (17) are similar. When using PCG, the approximate

Hessian-vector products $\mathbf{y} = \mathbf{H}\mathbf{x}$ can be computed as

$$\begin{aligned}
\tilde{\mathcal{X}} &= \tilde{\mathcal{D}} * (\llbracket \mathbf{X}_\mathrm{A}, \mathbf{B}, \mathbf{C} \rrbracket + \llbracket \mathbf{A}, \mathbf{X}_\mathrm{B}, \mathbf{C} \rrbracket + \llbracket \mathbf{A}, \mathbf{B}, \mathbf{X}_\mathrm{C} \rrbracket) \\
\mathbf{y}_\mathrm{A} &= \mathrm{vec}\left(\tilde{\mathbf{X}}_{(1)}(\mathbf{C} \odot \mathbf{B})\right) \\
\mathbf{y}_\mathrm{B} &= \mathrm{vec}\left(\tilde{\mathbf{X}}_{(2)}(\mathbf{C} \odot \mathbf{A})\right) \\
\mathbf{y}_\mathrm{C} &= \mathrm{vec}\left(\tilde{\mathbf{X}}_{(3)}(\mathbf{B} \odot \mathbf{A})\right),
\end{aligned}$$

in which $\tilde{\mathcal{D}} = \mathrm{unvec}\,(\mathrm{diag}(\mathbf{D}))$ and $\tilde{\mathcal{X}}$ is an auxiliary variable. To compute the sum of the CPDs, i.e., $\tilde{\mathcal{X}}$, the intermediate results can be reused. If $\tilde{\mathcal{D}}$ is sparse, the computational complexity can be reduced by only computing the entries corresponding to nonzero entries in $\tilde{\mathcal{D}}$ and by using specialized routines for the MTKRPROD of sparse tensors; see, e.g., [60, 61].

4. Constrained decompositions

Constraints can be used to incorporate prior knowledge: are the factor vectors smooth, nonnegative, linear combinations of elements in a dictionary, or do they have a Vandermonde structure? In this section, hard and soft constraints are discussed. In the former type, the constraint always holds, while constraints can be violated in the latter type and violations are penalized. Techniques to add parametric and projection based bound, or *box*, constraints to the GN optimization problem are discussed first in subsections 4.1 and 4.2 as examples of hard constraints. Next, soft constraints are added through regularization in subsection 4.3. Finally, symmetry constraints are discussed as an example in subsection 4.4.

> **Example 8:** A rank-2 tensor $\mathcal{T}$ with smooth factor vectors is constructed as follows. The first factor vector in the first mode $\mathbf{a}_1$ is a sum of two exponentials. All other factor vectors are random polynomials with a maximal degree equal to four. (See Figure 7.) Then Gaussian, white, i.i.d. noise is added such that the SNR is $-20\,\mathrm{dB}$, which loosely means that the noise is ten times stronger than the signal. We exploit the prior knowledge that each factor vector (also $\mathbf{a}_1$) can be approximated by low-degree polynomials by imposing polynomial constraints (with a maximal degree of four). As the results in Figure 7 show, the rank-1 terms can be recovered thanks to the polynomial constraints, while the result using unconstrained CPD is heavily perturbed by noise. Albeit $\mathbf{a}_1$ is a sum of exponentials, it can be approximated well by a polynomial, as is clear from the figure.

4.1. Parametric constraints

A large number of constraints imposed on factor matrices can be modeled as a function or *transformation* of some parameter vector $\mathbf{z} = \mathrm{vec}\,(\mathbf{Z})$:

1. nonnegativity can be implemented by squaring values, i.e., $\mathbf{A} = \mathbf{Z} * \mathbf{Z}$ [65], or by taking the absolute value, i.e., $\mathbf{A} = |\mathbf{Z}|$;

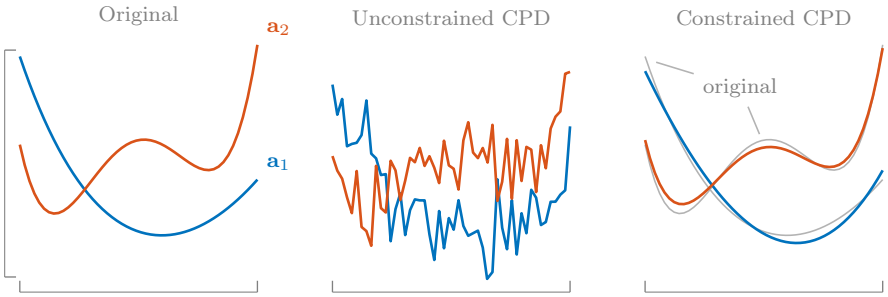


Figure 7: By imposing polynomial constraints, which we assume as prior knowledge, the smooth factor vectors can be recovered using a CPD of a noisy tensor. Without constraints, the results are heavily perturbed by the noise. The plots show the factor vectors in the first mode. The SNR is $-20\,\text{dB}$.

2. a matrix with entries in a given interval can, for example, be modeled using sigmoidal constraints, e.g. $a_{ir} = z_{ir}/\sqrt{1+z_{ir}^2}$ for $-1 \leq a_{ir} \leq 1$;
3. a matrix $\mathbf{A}$ with normalized columns can be implemented by defining each entry as $a_{ir} = z_{ir}/\sqrt{\sum_{k=1}^{I} z_{kr}^2}$;
4. a matrix $\mathbf{A}$ in which each column $\mathbf{a}_r$ is a polynomial of degree d evaluated in the given points t_i, e.g., $a_{ir} = q_{0r} + q_{1r}t_i + q_{2r}t_i^2 + \ldots + q_{dr}t_i^d$, can be modeled as a chosen basis matrix $\mathbf{M}$ times a coefficient matrix $\mathbf{Q}$, i.e., $\mathbf{A} = \mathbf{MQ}$, with, e.g. for a monomial basis

$$a_{ir} = \underbrace{\begin{bmatrix} 1 & t_i & t_i^2 & \ldots & t_i^d \end{bmatrix}}_{\mathbf{M}(i,:)} \underbrace{\begin{bmatrix} q_{0r} & q_{1r} & q_{2r} & \ldots & q_{dr} \end{bmatrix}^{\mathrm{T}}}_{\mathbf{q}_r^{\mathrm{T}}};$$

5. the decomposition in multilinear rank-$(L_r, L_r, 1)$ terms [13] or PARALIND [14] can be seen as a CPD with repeated factor vectors in the third mode, i.e., $\mathbf{C} = [\mathbf{z}_1, \ldots, \mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_2, \ldots, \mathbf{z}_R, \ldots, \mathbf{z}_R]$ in which $\mathbf{z}_r$ is repeated L_r times, which can be written as the matrix-matrix product

$$\mathbf{C} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_R \end{bmatrix} \begin{bmatrix} \mathbf{1}_{L_1}^{\mathrm{T}} & & & \\ & \mathbf{1}_{L_2}^{\mathrm{T}} & & \\ & & \ddots & \\ & & & \mathbf{1}_{L_R}^{\mathrm{T}} \end{bmatrix};$$

6. a matrix with the entries from $\mathbf{z}$ on its diagonal, i.e., $\mathbf{A} = \operatorname{diag}(\mathbf{z})$;
7. a matrix $\mathbf{A}$ with constant values on the anti-diagonals, i.e., a Hankel matrix, depends on a single generating vector $\mathbf{z}$ as $a_{ir} = z_{i+r-1}$;
8. orthogonality can be imposed using Householder reflectors $\mathbf{z}_r \in \mathbb{R}^{I-r+1}$, $r = 1, \ldots, R$, as variables [55];
9. a matrix $\mathbf{A}$ in which the rth column is a Gaussian bell curve with mean m_r and variance σ_r^2, i.e., $a_{ir} = \exp\left(\frac{(t_i - m_r)^2}{2\sigma_r^2}\right)$, uses variables $\mathbf{z} = \begin{bmatrix} \mathbf{m}; \boldsymbol{\sigma} \end{bmatrix} \in \mathbb{R}^{2R}$ and a given point set $\mathbf{t}$.

These examples represent various types of constraints that often occur. The first two examples are element-wise constraints, the third is a column-wise constraint and the fourth and fifth are matrix-matrix product type constraints. The sixth and seventh example are placement type constraints as values from $\mathbf{z}$ are put at certain positions in a matrix. For placement type constraints some entries can be constants, e.g. zero as in the sixth example. The eight example represents the most general case. Finally, while example 9 actually is a column-wise constraint and can be implemented as a single transformation, it can also be seen as chain of five simpler ones as follows

$$\begin{aligned}(\mathbf{Z}_1)_{ir} &= t_i - m_r \\ (\mathbf{z}_2)_r &= \sigma_r^2 \\ \mathbf{z}_3 &= 2\mathbf{z}_2 \\ (\mathbf{Z}_4)_{ir} &= (\mathbf{Z}_1)_{ir}/(\mathbf{z}_3)_r \\ (\mathbf{Z}_5)_{ir} &= \exp((\mathbf{Z}_4)_{ir}).\end{aligned}$$

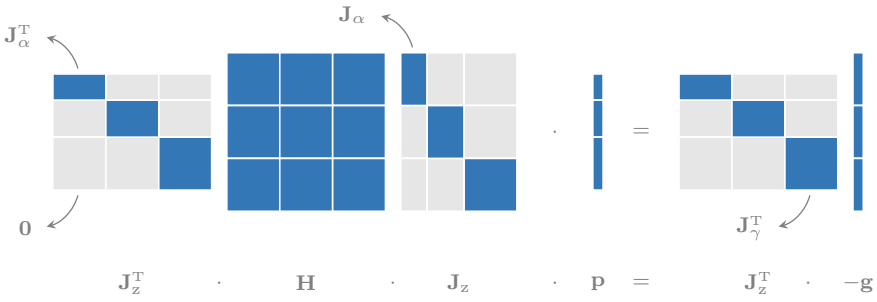


Figure 8: To impose parametric constraints on the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, which depend on the variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, respectively, an additional block diagonal matrix $\mathbf{J}_\mathrm{z}$ is introduced in the system used to find $\mathbf{p}$, i.e., the update vector for the variables.

When constraints are imposed on factor matrices, the underlying variables are updated rather than the factor matrices, hence $\mathbf{z}$ and $\mathbf{p}$ in (2) and (3) now correspond to variables instead of factor matrices. The parametric constraints can be incorporated easily in the GN framework by exploiting the chain rule for derivation as shown in [32]: the derivative w.r.t. the underlying variables $\mathbf{z}$ is given by the derivative the w.r.t. factor matrix multiplied by the derivative of the factor matrix w.r.t. the variable $\mathbf{z}$. Mathematically, the system used to compute the step $\mathbf{p}$ becomes

$$\mathbf{J}_\mathrm{z}^\mathrm{T}\mathbf{H}\mathbf{J}_\mathrm{z}\mathbf{p} = -\mathbf{J}_\mathrm{z}^\mathrm{T}\mathbf{g}, \tag{20}$$

in which $\mathbf{J}_\mathrm{z}$ is a block-diagonal matrix having the Jacobians w.r.t. the underlying variables as blocks; see Figure 8 for a schematic overview. Hence, if $\mathbf{A}$ depends on $\boldsymbol{\alpha} \in \mathbb{R}^{N_a}$, $\mathbf{B}$ on $\boldsymbol{\beta} \in \mathbb{R}^{N_b}$ and $\mathbf{C}$ on $\boldsymbol{\gamma} \in \mathbb{R}^{N_b}$, then $\mathbf{z} = \left[\boldsymbol{\alpha}; \boldsymbol{\beta}; \boldsymbol{\gamma}\right]$ and $\mathbf{J}_\mathrm{z} = \mathrm{blkdiag}(\mathbf{J}_\alpha, \mathbf{J}_\beta, \mathbf{J}_\gamma)$, in which

$\mathbf{J}_\alpha$ is the Jacobian of $\mathbf{A}$ w.r.t. $\boldsymbol{\alpha}$:

$$\mathbf{J}_\alpha = \begin{bmatrix} \frac{\partial a_{11}}{\partial \alpha_1} & \frac{\partial a_{11}}{\partial \alpha_2} & \dots & \frac{\partial a_{11}}{\partial \alpha_{N_a}} \\ \frac{\partial a_{21}}{\partial \alpha_1} & \frac{\partial a_{21}}{\partial \alpha_2} & \dots & \frac{\partial a_{21}}{\partial \alpha_{N_a}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial a_{11}}{\partial \alpha_1} & \frac{\partial a_{IR}}{\partial \alpha_2} & \dots & \frac{\partial a_{IR}}{\partial \alpha_{N_a}} \end{bmatrix}$$

and so on. The matrix $\mathbf{J}_\alpha$ depends only on the function used to transform the variables $\boldsymbol{\alpha}$ into the factor $\mathbf{A}$ and is therefore independent of the decomposition or divergence used. The structure of $\mathbf{J}_\alpha$ depends on the type of constraint, for example:

- for unconstrained factors $\mathbf{J}_\alpha = \mathbf{I}$,
- for element-wise transformations, $\mathbf{J}_\alpha$ is a diagonal matrix,
- for placement type transformations, $\mathbf{J}_\alpha$ is binary matrix such that $(\mathbf{J}_\alpha)_{ij} = 1$ if variable z_j is put in the ith position in $\mathbf{A}$, i.e., $\mathrm{vec}\,(\mathbf{A})_i = z_j$,
- for column-wise transformations, $\mathbf{J}_\alpha$ is a block diagonal matrix,
- for matrix product type constraints, $\mathbf{J}_\alpha$ is a Kronecker product; see (10).

Example 9: Consider a Toeplitz type constraint, i.e., a matrix with constant diagonals, with variables $\boldsymbol{\alpha} = [\alpha_1; \alpha_2; \alpha_3]$ and a zero upper triangular part:

$$\mathbf{A} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ \alpha_2 & \alpha_1 & 0 \\ \alpha_3 & \alpha_2 & \alpha_1 \end{bmatrix}.$$

This is a placement type constraint with constants and its binary Jacobian therefore is given by

$$\mathbf{J}_\alpha^{\mathrm{T}} = \begin{array}{c} \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{array} \begin{array}{c} \begin{array}{ccccccccc} a_{11} & a_{21} & a_{31} & a_{12} & a_{22} & a_{32} & a_{13} & a_{23} & a_{33} \end{array} \\ \begin{bmatrix} 1 & & & & 1 & & & & 1 \\ & 1 & & & & 1 & & & \\ & & 1 & & & & & & \end{bmatrix} \end{array}.$$

These structures can then be exploited when computing products with vectorized matrices: for the gradients we have

$$\mathbf{J}_{\mathbf{z}}^{\mathrm{T}} \mathbf{g} = \begin{bmatrix} \mathbf{J}_\alpha^{\mathrm{T}} \mathrm{vec}\,(\mathbf{G}_{\mathrm{A}}) \\ \mathbf{J}_\beta^{\mathrm{T}} \mathrm{vec}\,(\mathbf{G}_{\mathrm{B}}) \\ \mathbf{J}_\gamma^{\mathrm{T}} \mathrm{vec}\,(\mathbf{G}_{\mathrm{C}}) \end{bmatrix}. \tag{21}$$

To compute the Gramian-vector products required to solve system (20) using PCG, we

use the following three steps:

$$\tilde{\mathbf{x}} = \mathbf{J}_z \mathbf{x} \tag{22}$$
$$\tilde{\mathbf{y}} = \mathbf{H} \tilde{\mathbf{x}} \tag{23}$$
$$\mathbf{y} = \mathbf{J}_z \tilde{\mathbf{y}}. \tag{24}$$

The vectors $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ have the dimensions of the factor matrices, while $\mathbf{x}$ and $\mathbf{y}$ have the dimensions of the underlying variables. Hence, (22)–(24) can be seen as an expansion of the current CG iterate $\mathbf{x}$ to the factors, followed by a Gramian-vector product without constraints as discussed in subsection 3.2.2. Finally the factors are contracted again.

Example 10 (nonnegativity constraints): When implementing nonnegativity by squaring the variables $\boldsymbol{\alpha} = \operatorname{vec}(\mathbf{D})$, i.e., $\mathbf{A} = \mathbf{D} * \mathbf{D}$, the Jacobian w.r.t. $\boldsymbol{\alpha}$ is given by $\mathbf{J}_\alpha = \operatorname{diag}(2\boldsymbol{\alpha})$. (The other factors are not transformed.) Hence, the gradient w.r.t. $\boldsymbol{\alpha}$ in (21) can be computed as

$$\mathbf{J}_\alpha^{\mathrm{T}} \operatorname{vec}(\mathbf{G}_{\mathrm{A}}) = \operatorname{vec}(2\mathbf{D} * \mathbf{G}_{\mathrm{A}}).$$

For the Gramian-vector product $\mathbf{y}_\alpha = \operatorname{vec}(\mathbf{Y}_\alpha)$, we compute (22)–(24) as

$$\begin{aligned} \tilde{\mathbf{X}}_{\mathrm{A}} &= 2\mathbf{D} * \mathbf{X}_\alpha \\ \operatorname{vec}\left(\tilde{\mathbf{Y}}_{\mathrm{A}}\right) &= \mathbf{H}_{\mathrm{AA}} \operatorname{vec}\left(\tilde{\mathbf{X}}_{\mathrm{A}}\right) + \mathbf{H}_{\mathrm{AB}} \operatorname{vec}(\mathbf{X}_{\mathrm{B}}) + \mathbf{H}_{\mathrm{AC}} \operatorname{vec}(\mathbf{X}_{\mathrm{C}}) \\ \mathbf{Y}_\alpha &= 2\mathbf{D} * \tilde{\mathbf{Y}}_{\mathrm{A}}. \end{aligned}$$

Example 11 (polynomial constraints): When implementing polynomial constraints with known basis matrix $\mathbf{M}$ and unknown coefficients $\mathbf{Q}$, hence $\boldsymbol{\alpha} = \operatorname{vec}(\mathbf{Q})$ and $\mathbf{A} = \mathbf{M}\mathbf{Q}$, the Jacobian w.r.t. $\boldsymbol{\alpha}$ is given by $\mathbf{J}_\alpha = \mathbf{I}_R \otimes \mathbf{M}$. (The other factors are not transformed.) Hence, the gradient w.r.t. $\boldsymbol{\alpha}$ in (21) can be computed using (10) as

$$\mathbf{J}_\alpha^{\mathrm{T}} \operatorname{vec}(\mathbf{G}_{\mathrm{A}}) = \operatorname{vec}(\mathbf{M}^{\mathrm{T}} \mathbf{G}_{\mathrm{A}}).$$

For the Gramian-vector product $\mathbf{y}_\alpha = \operatorname{vec}(\mathbf{Y}_\alpha)$, we compute (22)–(24) as

$$\begin{aligned} \tilde{\mathbf{X}}_{\mathrm{A}} &= \mathbf{M}\mathbf{X}_\alpha \\ \operatorname{vec}\left(\tilde{\mathbf{Y}}_{\mathrm{A}}\right) &= \mathbf{H}_{\mathrm{AA}} \operatorname{vec}\left(\tilde{\mathbf{X}}_{\mathrm{A}}\right) + \mathbf{H}_{\mathrm{AB}} \operatorname{vec}(\mathbf{X}_{\mathrm{B}}) + \mathbf{H}_{\mathrm{AC}} \operatorname{vec}(\mathbf{X}_{\mathrm{C}}) \\ \mathbf{Y}_\alpha &= \mathbf{M}^{\mathrm{T}} \tilde{\mathbf{Y}}_{\mathrm{A}}. \end{aligned}$$

Example 12 (chaining constraints): If a factor matrix $\mathbf{A}$ is the result of chaining multiple transformations (see, e.g., the ninth example in the beginning of subsection 4.1), the Jacobian $\mathbf{J}_\alpha$ is the product of the Jacobians for each transformation, which follows from the chain rule. Consider a polynomial constraint with

basis matrix $\mathbf{M}$ and nonnegative coefficients, implemented by squaring variables:

$$\mathbf{A} = \mathbf{M}\mathbf{Q} \qquad \text{with} \quad \mathbf{Q} = \mathbf{D} * \mathbf{D}.$$

The Jacobian $\mathbf{J}_\alpha$ of $\operatorname{vec}(\mathbf{A})$ w.r.t. $\boldsymbol{\alpha} = \operatorname{vec}(\mathbf{D})$ is then given by

$$\mathbf{J}_\alpha = \mathbf{J}_{\mathrm{Q}}^{\mathrm{A}} \mathbf{J}_{\mathrm{D}}^{\mathrm{Q}},$$

in which $\mathbf{J}_{\mathrm{Q}}^{\mathrm{A}} = \mathbf{I}_R \otimes \mathbf{M}$ is the Jacobian of $\operatorname{vec}(\mathbf{A})$ w.r.t. $\operatorname{vec}(\mathbf{Q})$ and $\mathbf{J}_{\mathrm{D}}^{\mathrm{Q}} = \operatorname{diag}(2\boldsymbol{\alpha})$ is the Jacobian of $\operatorname{vec}(\mathbf{Q})$ w.r.t. $\boldsymbol{\alpha} = \operatorname{vec}(\mathbf{D})$; see examples 10 and 11. Hence,

$$\mathbf{J}_\alpha = (\mathbf{I}_R \otimes \mathbf{M})\operatorname{diag}(2\boldsymbol{\alpha}).$$

Therefore, the gradient w.r.t. $\boldsymbol{\alpha}$ in (21) can be computed as

$$\mathbf{J}_\alpha^{\mathrm{T}} \operatorname{vec}(\mathbf{G}_{\mathrm{A}}) = \operatorname{vec}\left(\mathbf{M}^{\mathrm{T}}(2\mathbf{D} * \mathbf{G}_{\mathrm{A}})\right).$$

The Gramian-vector products can be derived in similar way as in the previous examples.

4.2. Projection-based bound constraints

It is often useful to constrain variables between certain bounds. For example, when using the KL divergence in subsection 3.4, the model m_i should be nonnegative because of the logarithm, which can be achieved using nonnegativity constraints on the factor matrices. Instead of squaring the variables as in example 10, bound or box constraints can be used as an alternative. Mathematically, the problem

$$\min_{\mathbf{z}} \frac{1}{2} \left|\left| [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T} \right|\right|_{\mathrm{F}}^2 \quad \text{subject to} \quad \mathbf{l} \leq \mathbf{z} \leq \mathbf{u}$$

is solved, in which $\leq$ hold elementwise and $\mathbf{l} \leq \mathbf{u}$. If a certain variable z_i is unbounded below or above, the lower bound $l_i = -\infty$ or the upper bound $u_i = +\infty$ is used, respectively. To enforce these constraints, an active set method can be used. The active set $\mathcal{A}$ is defined as the set of indices corresponding to the variables for which the current estimates $\mathbf{z}$ are at the bounds, i.e.,

$$\mathcal{A} = \{i \mid l_i = z_i \text{ or } z_i = u_i\}, \tag{25}$$

and its complement is the inactive set $\mathcal{I}$. While determining active sets is not trivial in general, simple bound constraints do allow an easy method to be used [50]. At the beginning of each iteration, the active set is determined using (25). Suppose that the variables $\mathbf{z}$ are sorted such that $\mathbf{z} = \left[\mathbf{z}_{\mathcal{I}}; \mathbf{z}_{\mathcal{A}}\right]$ and that $\mathbf{p}$ and $\mathbf{g}$ are partitioned accordingly, then, instead of the system (5), the altered system

$$\left(\begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{|\mathcal{A}|} \end{bmatrix} + \begin{bmatrix} \mathbf{I}_{|\mathcal{I}|} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{H} \begin{bmatrix} \mathbf{I}_{|\mathcal{I}|} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \right) \mathbf{p} = -\mathbf{g}$$

is solved, which can be simplified as

$$\mathbf{H}\begin{bmatrix}\mathbf{p}_{\mathcal{I}}\\ \mathbf{0}\end{bmatrix} = -\begin{bmatrix}\mathbf{g}_{\mathcal{I}}\\ \mathbf{0}\end{bmatrix}$$
$$\mathbf{p}_{\mathcal{A}} = -\mathbf{g}_{\mathcal{A}}.$$

Therefore, the procedures for efficient Gramian-vector products can be reused. The only change is that every CG iteration the entries of $\mathbf{x}$ in the active set are set to zero, i.e., let $\tilde{\mathbf{x}} = \left[\mathbf{x}_{\mathcal{I}}; \mathbf{0}\right]$, after which the standard Gramian-vector product $\tilde{\mathbf{y}} = \mathbf{H}\tilde{\mathbf{x}}$ is computed as in subsection 3.2.2. The result $\mathbf{y}$ is finally constructed by concatenating the entries corresponding to the inactive set from $\tilde{\mathbf{y}}$ and the entries corresponding to the active set from $\mathbf{x}$, hence, $\mathbf{y} = \left[\tilde{\mathbf{y}}_{\mathcal{I}}; \mathbf{x}_{\mathcal{A}}\right]$. When updating the variables, instead of the computed $\mathbf{p}$, a projected step $\tilde{\mathbf{p}}$ is used to ensure that the constraints are not violated:

$$\tilde{p}_i = \begin{cases} l_i - z_i & \text{if } z_i + p_i \leq l_i \\ u_i - z_i & \text{if } z_i + p_i \geq u_i \\ p_i & \text{otherwise} \end{cases}.$$

Example 13 (nonnegative CPD): Consider a random rank-10 tensor $\mathcal{T}$ of size $50 \times 50 \times 50$, constructed using factor matrices with entries drawn from the uniform distribution $\mathcal{U}(0, 1)$ such that all factor are nonnegative. The nonnegative CPD of the noiseless tensor $\mathcal{T}$ is computed starting from random factor matrices with entries drawn from the same uniform distribution, except for the first factor matrix, which is initialized to zero. Two methods are compared: an active set method with lower bound $\mathbf{l} = \mathbf{0}$ and no upper bound ($\mathbf{u} = +\boldsymbol{\infty}$), and a method with parametric constraints, i.e., by squaring variables (see example 10). The former method is implemented as `cpd_nls` with the `nlsb_gndl` solver, while the latter uses the SDF framework through `sdf_nls` and `struct_nonneg` [53]. While the active set method recovers the original factor matrices up to machine precision, `sdf_nls` fails to perform a single step, as the gradient $\mathbf{g}$ is exactly zero for this method which means this initialization is a local minimum for the problem with parametric constraints. The inability to change variables that are initialized at zero or that become zero during the optimization process, is a common problem with nonnegativity constraints implemented using squared variables.

4.3. Regularization and soft constraints

Regularization is often used to prevent overfitting or to incorporate prior knowledge into the optimization problem. Factor matrices or variables can be regularized by adding terms to the objective function:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} \left|\left|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}\right|\right|^2 + \lambda_{\mathrm{A}} h_{\mathrm{A}}(\mathbf{A}) + \lambda_{\mathrm{B}} h_{\mathrm{B}}(\mathbf{B}) + \lambda_{\mathrm{C}} h_{\mathrm{C}}(\mathbf{C}),$$

in which λ_{A}, λ_{B} and λ_{C} are hyperparameters which have to be chosen by the user or automatically, e.g. through cross validation. Examples of possible functions h_{A}, h_{B} and h_{C} are

- L_2 norm regularization: $h(\mathbf{z}) = \frac{1}{2}\left|\left|\mathbf{z}\right|\right|_2^2$,
- L_1 norm regularization: $h(\mathbf{z}) = \left|\left|\mathbf{z}\right|\right|_1$, and
- L_0 norm regularization: $h(\mathbf{z}) = \left|\left|\mathbf{z}\right|\right|_0$.

The latter two norms are often used to achieve a sparser solution, as (small) nonzeros are penalized more heavily compared to the L_2 norm. As the derivative of a sum is the sum of the derivatives, adding regularization results in an additional term for the gradient and the Hessian approximation, as illustrated in Figure 9. Concretely, let $\tilde{\mathbf{g}}$ and $\tilde{\mathbf{H}}$ be the gradient and Gramian for the least squares term $\frac{1}{2}\left|\left|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}\right|\right|_{\mathrm{F}}^2$. Then, in the case only $\mathbf{A}$ is regularized, the gradient and Hessian approximation are altered as

$$\begin{aligned}\mathbf{g}_{\mathrm{A}} &= \tilde{\mathbf{g}}_{\mathrm{A}} + \lambda \nabla_{\mathrm{vec}(\mathbf{A})} h \\ \mathbf{H}_{\mathrm{AA}} &= \tilde{\mathbf{H}}_{\mathrm{AA}} + \lambda \nabla^2_{\mathrm{vec}(\mathbf{A})} h,\end{aligned}$$

while $\mathbf{g}_{\mathrm{B}}$ and $\mathbf{g}_{\mathrm{C}}$, and all other blocks of the Gramian remain unchanged. For example, for L_2 regularization this becomes

$$\begin{aligned}\mathbf{g}_{\mathrm{A}} &= \tilde{\mathbf{g}}_{\mathrm{A}} + \lambda \mathrm{vec}\left(\mathbf{A}\right) \\ \mathbf{H}_{\mathrm{AA}} &= \tilde{\mathbf{H}}_{\mathrm{AA}} + \lambda \mathbf{I}.\end{aligned}$$

Similar terms can be added in the case other factors are regularized as well. Instead of the exact Hessian $\nabla^2_{\mathrm{vec}(\mathbf{A})} h$, an approximation can be used; see subsection 2.2.

Using regularization, soft constraints can be imposed. In contrast to the parameter and projection based constraints, violations of the constraint are allowed, but penalized. The penalty is controlled by the hyperparameter λ and the regularization function h.

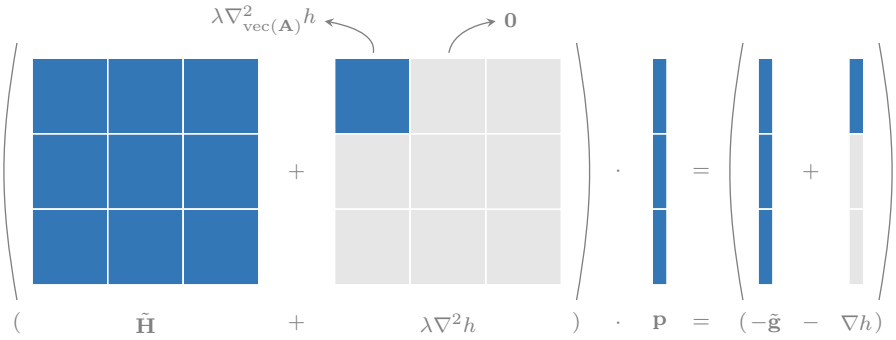


Figure 9: When adding regularization or implementing soft constraints, a block is added to the Hessian approximation and to the gradient.

Example 14 (soft orthogonality constraints): Instead of using Householder reflectors as parameters in order to enforce orthogonality, a regularization term can

be added to impose orthogonality as a soft constraint:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} ||[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!] - \mathcal{T}||_{\mathrm{F}}^2 + \frac{\lambda}{4} ||\mathbf{A}^{\mathrm{T}}\mathbf{A} - \mathbf{I}_R||_{\mathrm{F}}^2 .$$

The additional contributions to the gradient and Gramian can be derived easily by noting that $\mathbf{A}^{\mathrm{T}}\mathbf{A} = [\![\mathbf{A}^{\mathrm{T}}, \mathbf{A}^{\mathrm{T}}]\!]$ is a symmetric PD (see subsection 4.4) and that $\partial(\mathbf{A}^{\mathrm{T}}) = (\partial\mathbf{A})^{\mathrm{T}}$. Therefore, the gradient w.r.t. $\mathbf{A}$ is given by

$$\mathbf{g}_{\mathrm{A}} = \tilde{\mathbf{g}}_{\mathrm{A}} + \lambda \mathrm{vec}\left(\mathbf{A}\left(\mathbf{A}^{\mathrm{T}}\mathbf{A} - \mathbf{I}\right)\right).$$

To compute the Gramian-vector product, we find after some simplifications that

$$\mathbf{y}_{\mathrm{A}} = \tilde{\mathbf{y}}_{\mathrm{A}} + \lambda \mathrm{vec}\left(\mathbf{A}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}_{\mathrm{A}} + \mathbf{X}_{\mathrm{A}}^{\mathrm{T}}\mathbf{A}\right)\right).$$

4.4. Symmetry

In the previous subsections, we (implicitly) assumed that each factor matrix is a function of a different set of variables which results in a block-diagonal matrix $\mathbf{J}_z$. Variables can be shared across different modes, however. An extreme example is the symmetric CPD of a tensor, e.g. $\mathcal{T} = [\![\mathbf{A}, \mathbf{A}, \mathbf{A}]\!]$. Incorporating symmetry in the GN framework amounts to summing blocks of $\mathbf{H}$ and $\mathbf{g}$ as shown in Figure 10. As many blocks are identical, only some need to be computed. Note that the contributions to the gradient are only identical if the tensor $\mathcal{T}$ is symmetric as well. The idea of summing the proper blocks can be extended easily to cases in which multiple factors share the same variables by summing the blocks corresponding to the variables rather than the blocks corresponding to the factors [33].

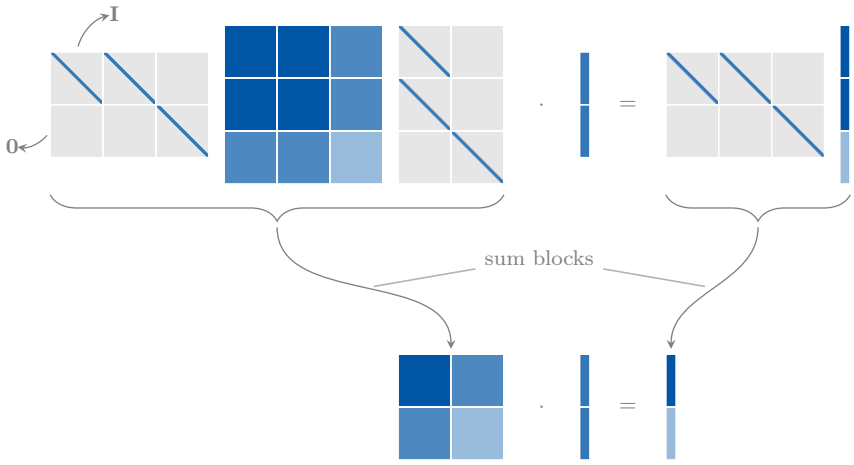


Figure 10: For symmetric tensors, e.g. $\mathcal{T} = [\![\mathbf{A}, \mathbf{A}, \mathbf{C}]\!]$, the system (5) can be solved more cheaply by summing blocks corresponding to the same variable. As many — not all — blocks in the same color are identical, the sums can be simplied; see example 15.

Example 15: Consider a tensor $\mathcal{T}$ that is approximated using a CPD with symmetry constraints in the first two modes, i.e., $\mathcal{T} \approx [\![\mathbf{A}, \mathbf{A}, \mathbf{C}]\!]$; see Figure 10. In this case, the gradient is given by

$$\begin{aligned}\mathbf{g}_{\mathrm{A}} &= \mathbf{R}_{(1)}\left(\mathbf{C} \odot \mathbf{A}\right) + \mathbf{R}_{(2)}\left(\mathbf{C} \odot \mathbf{A}\right) \\ \mathbf{g}_{\mathrm{C}} &= \mathbf{R}_{(3)}\left(\mathbf{A} \odot \mathbf{A}\right).\end{aligned}$$

Only if $\mathcal{T}$ is also symmetric in the first two modes, $\mathbf{R}_{(1)} = \mathbf{R}_{(2)}$ and $\mathbf{g}_{\mathrm{A}}$ can be computed as $\mathbf{g}_{\mathrm{A}} = 2\mathbf{R}_{(1)}\left(\mathbf{C} \odot \mathbf{A}\right)$. To compute the Gramian-vector products $\mathbf{y} = \mathbf{H}\mathbf{x}$, with $\mathbf{y} = \left[\mathbf{y}_{\mathrm{A}}; \mathbf{y}_{\mathrm{C}}\right]$ and $\mathbf{x} = \left[\mathbf{x}_{\mathrm{A}}; \mathbf{x}_{\mathrm{C}}\right]$, we can use

$$\begin{aligned}\mathbf{y}_{\mathrm{A}} &= 2\mathbf{X}_{\mathrm{A}}\left((\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})\right) + 2\mathbf{A}\left((\mathbf{X}_{\mathrm{A}}^{\mathrm{T}}\mathbf{A}) * (\mathbf{C}^{\mathrm{T}}\mathbf{C})\right) + 2\mathbf{A}\left((\mathbf{A}^{\mathrm{T}}\mathbf{A}) * (\mathbf{X}_{\mathrm{C}}^{\mathrm{T}}\mathbf{C})\right) \\ \mathbf{y}_{\mathrm{C}} &= 2\mathbf{C}\left((\mathbf{X}_{\mathrm{A}}^{\mathrm{T}}\mathbf{A}) * (\mathbf{A}^{\mathrm{T}}\mathbf{A})\right) + 2\mathbf{X}_{\mathrm{C}}\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right).\end{aligned}$$

5. Coupled decompositions

A decomposition of a third-order tensor can be seen as the joint decomposition of its matrix slices. More generally, one may consider the joint decomposition of matrices and/or tensors (possibly of different order) that (do not necessarily have the same dimensions and) only share part of their structure. For example, only a few columns of a factor matrix are shared, or a factor in one dataset is a function of a factor in another dataset. Such joint analysis is important for applications like data fusion, e.g. in multimodal data analysis in biomedical applications. For example, in [3] a tensor resulting from EEG measurements and a matrix resulting from fMRI are coupled: the time factor for the fMRI data is the convolution of the time factor in the EEG data with the hemodynamic response function. An example from recommender systems is given in example 16.

Example 16 (GPS): Predicting whether a person has attended or will attend an activity, e.g., buy diner or go shopping, at a certain location, e.g., the mall or a local grocery store, is an important part of recommender systems. Using the GPS dataset [66], a tensor with modes user $\times$ location $\times$ activity can be constructed in which entries can be missing; see Figure 11. To predict the unknown entries, a CPD of the incomplete tensor can be computed. To improve the prediction quality, additional information can be used such as features for each location, the relation between users and the similarity between activities etc. [66]. This extra information is given as a set of matrices, which are factorized jointly with the tensor by coupling the factor matrices corresponding to the same modes [32, 67]. When a new user is added, no information about visits to a location for a certain activity is yet available, hence an entire slice of the tensor is missing. However, because of (known) relations with other users or shared interests, it is still possible to make predictions using the coupled decompositions. See [32, 67] for more information.

While the obvious use of coupled decompositions is data fusion, they can also be employed in a divide-and-conquer type of approach to data analysis. The coupled decomposition acts as a sort of compound eye: each facet sees its own simple part of the data, and the coupling creates the overall view. Example 17 is a very basic illustration

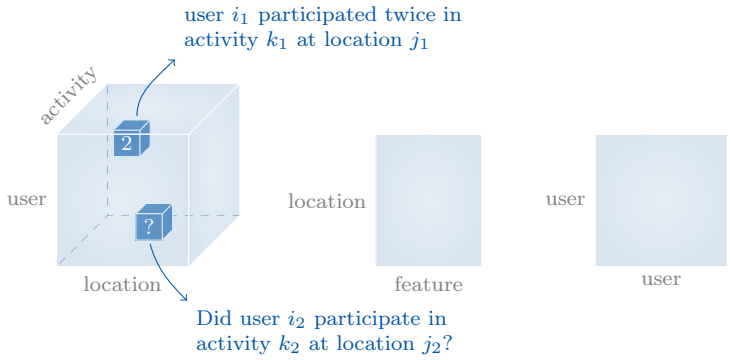


Figure 11: Did or will a user attend a certain activity at a certain location? By augmenting the GPS data tensor with information such as features for each location, relations between users and whether a user has been at a certain location, the unknown entries in the tensor can be predicted more accurately.

of a combination of two sampling rates, yielding a higher accuracy than the individual sampling rates. Multirate techniques show promise for big data applications in which classical Nyquist sampling is not feasible (the Nyquist sampling rate is twice the signal bandwidth, which may result in excessive amounts of data). In [8, 7] coupled decompositions are used for multirate harmonic retrieval (each of the decompositions takes care of one sampling rate). In [68] coupled decompositions yield algebraic uniqueness conditions and linear algebra based algorithms for tensor completion (each of the decompositions takes care of one fully observed subtensor). Noteworthy is also a connection between convolutive extensions of CPD and "instantaneous" coupled CPD [69].

A key advantage of coupling datasets in a signal separation context is that the conditions for uniqueness are usually very mild. While a matrix decomposition is not unique without requiring additional stringent and sometimes unnatural constraints such as orthogonality, the CPD, which can be seen as the joint factorization of matrix slices, is essentially unique under relatively mild conditions [70, 71, 72, 73]. When tensors and matrices are factorized jointly, the conditions can be relaxed further, and a decomposition may be recovered uniquely even if none of the decompositions of the tensors/matrices individually is unique [74, 25]. A special case is the coupled matrix tensor factorization (CMTF) for which uniqueness conditions are derived in [74]. In the case only a few columns of a factor are shared between the matrix and tensor, i.e., in the case of partial coupling, [75] discusses the remaining indeterminacies in the matrix decomposition.

While the focus in the remainder of this section lies on coupling of tensors and matrices that have a CPD structure, many other decompositions can be used. The structured data fusion (SDF) framework [32] discusses GN type algorithms to couple tensors factorized using CPDs, MLSVDs or BTDs. Other examples can be found in [43, 76, 9].

5.1. Exact coupling

Consider two tensors $\mathcal{T}_1$ and $\mathcal{T}_2$ which are both approximated by a rank-R CPD. To jointly factorize these tensors, the objective function

$$\min_{\mathbf{z}} \frac{\omega_1}{2} ||[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}_1||_{\mathrm{F}}^2 + \frac{\omega_2}{2} ||[\![\mathbf{D}, \mathbf{E}, \mathbf{F}]\!] - \mathcal{T}_2||_{\mathrm{F}}^2 \tag{26}$$

can be used with $\mathbf{z} = \left[\text{vec}\left(\mathbf{A}\right); \text{vec}\left(\mathbf{B}\right); \text{vec}\left(\mathbf{C}\right); \text{vec}\left(\mathbf{D}\right); \text{vec}\left(\mathbf{E}\right); \text{vec}\left(\mathbf{F}\right)\right]$. The weights ω_1 and ω_2 are hyperparameters and have to be chosen by the user or, e.g. through cross validation. In (26), the decompositions are uncoupled. The two decompositions are coupled exactly if one or more factors are (partially) shared, or if these factors depend on the same underlying variables. We discuss a few types of exact coupling:

- coupling factor matrices, e.g., $\mathbf{A} = \mathbf{D}$;
- partial coupling, e.g., $\mathbf{A} = \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3\end{bmatrix}$ and $\mathbf{D} = \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \mathbf{d}_3\end{bmatrix}$;
- coupling through variables $\mathbf{A} = h_1(\boldsymbol{\alpha})$ and $\mathbf{D} = h_2(\boldsymbol{\alpha})$.

For these three cases, the equality constraints can be imposed by substitution into the objective function. For example, if $\mathbf{A} = \mathbf{D}$, (26) becomes

$$\min_{\mathbf{z}} \frac{\omega_1}{2} \left|\left|[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}_1\right|\right|_{\text{F}}^2 + \frac{\omega_2}{2} \left|\left|[\![\mathbf{A}, \mathbf{E}, \mathbf{F}]\!] - \mathcal{T}_2\right|\right|_{\text{F}}^2 .$$

Hence, by eliminating $\mathbf{D}$, the variables are $\mathbf{z} = \left[\text{vec}\left(\mathbf{A}\right); \text{vec}\left(\mathbf{B}\right); \text{vec}\left(\mathbf{C}\right); \text{vec}\left(\mathbf{E}\right); \text{vec}\left(\mathbf{F}\right)\right]$. Then, because derivation is a linear operator, the system used to solve for $\mathbf{p}$ is simply

$$\left(\omega_1 \mathbf{H}_1 + \omega_2 \mathbf{H}_2\right) \mathbf{p} = -\left(\omega_1 \mathbf{g}_1 + \omega_2 \mathbf{g}_2\right)$$

in which $\mathbf{g}_i$ and $\mathbf{H}_i$ are the gradient and Gramian, respectively, of the ith term w.r.t. all variables [32, 33]. Both $\mathbf{g}_i$ and $\mathbf{H}_i$ contain zero blocks for factor matrices or variables not used in the factorization for the ith dataset, as shown in Figure 12. As explained in section 4, the gradient and the Gramian-vector products can be adapted easily. Note that for computational efficiency, $\mathbf{y} = \mathbf{H}\mathbf{x}$ is computed as $\mathbf{y} = (\omega_1 \mathbf{H}_1 \mathbf{x}) + (\omega_2 \mathbf{H}_2 \mathbf{x})$ rather than summing the blocks from $\mathbf{H}_1$ and $\mathbf{H}_2$ first [33].

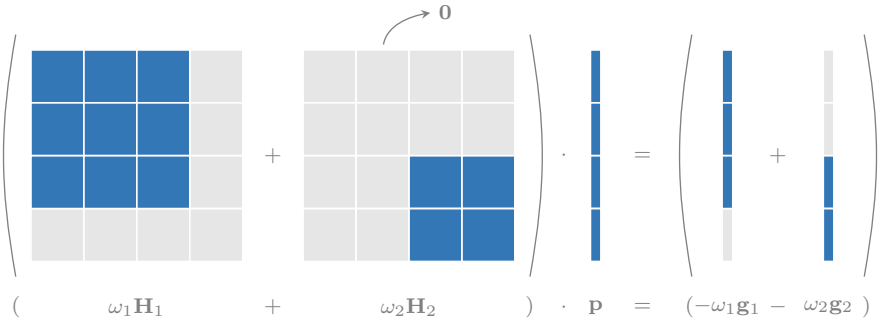


Figure 12: In the case of coupled tensor matrix factorization in which $\mathcal{T} \approx [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ and $\mathbf{M} \approx \mathbf{C}\mathbf{D}^{\text{T}}$, the system (5) is simply the sum of the two systems [33].

Example 17: Consider a function $h(x, y)$ which is sampled on two equidistant grids on $[-1, 1]$: grid 1 and grid 2 have 31 and 49 points in each dimension, respectively. Let $\mathbf{H}_1 \in \mathbb{R}^{31\times 31}$ and $\mathbf{H}_2 \in \mathbb{R}^{49\times 49}$ be two noisy measurements of $h(x, y)$ on

these grids such that the SNR is 20 dB. The goal is to approximate $h(x, y)$ as

$$h(x, y) = \sum_{r=1}^{3} a_r(x) b_r(y)$$

with $a_r(x)$ and $b_r(y)$ low degree polynomials. As explained in section 4, this is a matrix product type constraint, hence $\mathbf{H}_1$ and $\mathbf{H}_2$ are factorized as

$$\mathbf{H}_1 \approx \mathbf{A}\mathbf{B}^{\mathrm{T}} \qquad \mathbf{H}_2 \approx \mathbf{C}\mathbf{D}^{\mathrm{T}}$$

with

$$\mathbf{A} = \mathbf{M}_1\mathbf{Q}_1 \qquad \mathbf{B} = \mathbf{M}_1\mathbf{Q}_2 \tag{27}$$
$$\mathbf{C} = \mathbf{M}_2\mathbf{Q}_1 \qquad \mathbf{D} = \mathbf{M}_2\mathbf{Q}_2 \tag{28}$$

in which $\mathbf{M}_1$ and $\mathbf{M}_2$ are, in this example, Legendre bases evaluated in the 31 and 49 points corresponding to the first and second grid, respectively. Although $\mathbf{H}_1$ and $\mathbf{H}_2$ are sampled on different grids and have a different size, the underlying variables, i.e., the coefficients $\mathbf{Q}_1$ and $\mathbf{Q}_2$, are shared, hence the following coupled problem can be solved

$$\min_{\mathbf{Q}_1, \mathbf{Q}_2} \frac{\omega_1}{2 \cdot \Omega} ||\mathbf{H}_1 - \mathbf{A}\mathbf{B}^{\mathrm{T}}||_{\mathrm{F}}^2 + \frac{\omega_2}{2 \cdot \Omega} ||\mathbf{H}_2 - \mathbf{C}\mathbf{D}^{\mathrm{T}}||_{\mathrm{F}}^2 \tag{29}$$
subject to (27) and (28).

(The factor $\Omega = \omega_1 + \omega_2$ is a normalization factor, used to avoid numerical problems with large weights.) The constraints are again eliminated by substituting (27) and (28) in the objective function (29) and the coupled problem is solved using `sdf_nls` [53]. To measure the performance a third grid is used with 100 equidistant points in each direction. Let $\mathbf{H}_3$ be a noiseless measurement on this third grid, then the relative validation error

$$E = \frac{\left|\left|\mathbf{H}_3 - \hat{\mathbf{H}}_3\right|\right|_{\mathrm{F}}}{||\mathbf{H}_3||_{\mathrm{F}}}$$

is used, in which $\hat{\mathbf{H}}_3$ is the reconstruction using the computed coefficients $\mathbf{Q}_1$ and $\mathbf{Q}_2$. Figure 13 shows that the validation error E can be improved by using information from both measurements for a good choice of ω_1/ω_2.

5.2. Approximate coupling

Similar to soft constraints, two tensors can be coupled approximately by adding a regularization term. For example, if $\mathbf{A} \approx \mathbf{D}$, the objective function becomes

$$\min_{\mathbf{z}} \frac{\omega_1}{2} ||[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] - \mathcal{T}_1||_{\mathrm{F}}^2 + \frac{\omega_2}{2} ||[\![\mathbf{D}, \mathbf{E}, \mathbf{F}]\!] - \mathcal{T}_2||_{\mathrm{F}}^2 + \frac{\lambda}{2} ||\mathbf{A} - \mathbf{D}||_{\mathrm{F}}^2 .$$

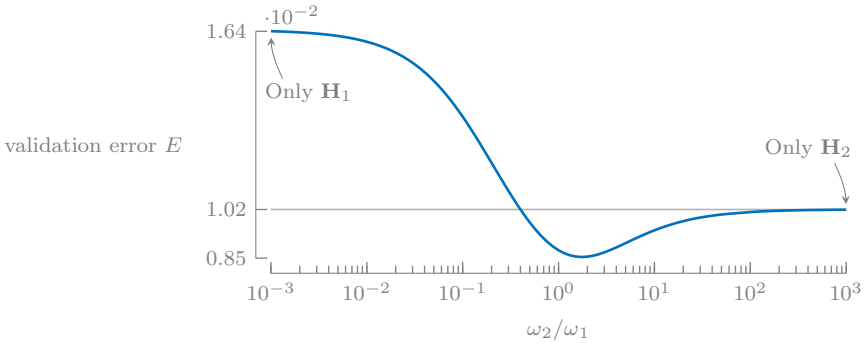


Figure 13: Over a certain range of ratios ω_1/ω_2, the validation error E is reduced when jointly factorizing both measurements $\mathbf{H}_1$ and $\mathbf{H}_2$ of $h(x, y)$, compared to using only one measurement.

Implementing this constraint can be done similarly to the approach in subsection 4.3. Note that there is an additional hyperparameter λ to be tuned. Other distance measures than the Frobenius norm can be used as well [77]. Soft coupling can also be imposed on the variables.

Example 18: The advanced coupled matrix tensor factorization (ACMTF) model [78] is an example of soft constraints. A tensor $\mathcal{T}$ and a matrix $\mathbf{M}$ share some factor vectors in the third and first mode, respectively, which can be modeled as

$$\mathcal{T} = \sum_{r=1}^{R} \sigma_r \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r \qquad \mathbf{M} = \sum_{r=1}^{R} \tau_r \mathbf{c}_r \otimes \mathbf{d}_r$$

in which σ_r is zero if the rth rank-1 term is not present in the decomposition of $\mathcal{T}$ and similarly for τ_r. Hence, if both $\sigma_r \neq 0$ and $\tau_r \neq 0$, the rth column of factor matrix $\mathbf{C}$ is shared. (The norms of the factor vectors are assumed to be equal to one.) The idea in ACMTF is that the shared columns of $\mathbf{C}$ are determined automatically by solving the following optimization problem [78]:

$$\min_{\mathbf{z}} \frac{\omega_1}{2} ||[\![\mathbf{A}, \mathbf{B}, \mathbf{C}, \boldsymbol{\sigma}^{\mathrm{T}}]\!] - \mathcal{T}||_{\mathrm{F}}^2 + \frac{\omega_2}{2} ||[\![\mathbf{C}, \mathbf{D}, \boldsymbol{\tau}^{\mathrm{T}}]\!] - \mathbf{M}||_{\mathrm{F}}^2 + \lambda \left(||\boldsymbol{\sigma}||_1 + ||\boldsymbol{\tau}||_1\right)$$
$$\text{subject to } ||\mathbf{a}_r||_{\mathrm{F}} = ||\mathbf{b}_r||_{\mathrm{F}} = ||\mathbf{c}_r||_{\mathrm{F}} = ||\mathbf{d}_r||_{\mathrm{F}} = 1, \quad r = 1, \ldots, R.$$

The L_1 norm is used to sparsify $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$. The normalization of the factor vectors can be implemented using parametric constraints (subsection 4.1) or using soft constraints [78]. Note that these normalization constraints can always be fully satisfied.

6. Large-scale computations

Compared to matrix problems, a tensor problem is more easily large-scale, even for modest-order tensors. Various techniques have been proposed to factorize large tensors, e.g., by using one or more samples of the tensor, by exploiting structure, by using ran-

domization or by distributing the problem and doing computations in parallel. In the remainder of this section, we focus on these concepts in the context of CPD computations.

Example 19 (curse of dimensionality): Physical properties such as the melting temperature, are important design parameters when designing new alloys. These properties vary among others with the fraction of each material, e.g., for stainless steel, we can have iron, carbon and chromium. If there are three components, two independent fractions can be varied to determine the property for all compositions. If the fractions are discretized as $0, 0.01, 0.02, \ldots, 0.99$, the values for all combinations can be arranged as a 100×100 matrix. When a component is added, there are three independent fractions and a tensor with 100^3 values is obtained. In general, for $N+1$ components, one needs to store and process 100^N values [79]. This exponential increase in memory and computational complexity is called the curse of dimensionality.

6.1. Compression

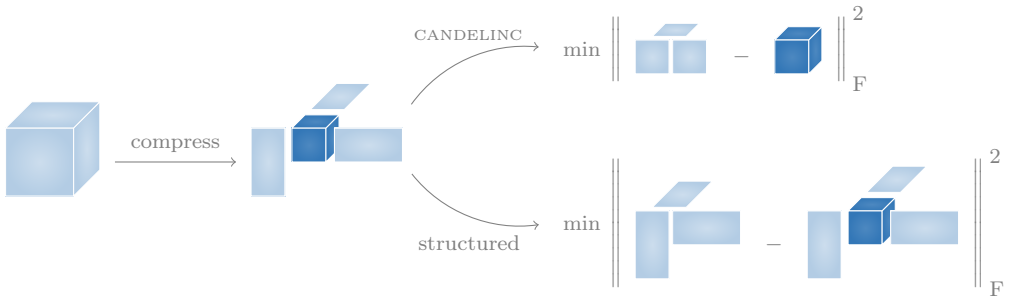


Figure 14: While the core tensor is decomposed in the CANDELINC model, the structured tensor framework replaces the tensor with its truncated MLSVD such that the original factor matrices are kept and constraints and coupling can be imposed easily.

A common strategy to reduce the complexity of computing a CPD is to compress the tensor first using an MLSVD [80]. It then suffices to decompose the core tensor, which follows from the CANDELINC model [81], as the orthogonal compression preserves norms. Hence, let $\mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W}$ be the truncated MLSVD of $\mathcal{T}$ with core size $\min(I, R) \times \min(J, R) \times \min(K, R)$. The CPD of $\mathcal{T}$ is then given by $\left[\!\left[\mathbf{U}\hat{\mathbf{A}}, \mathbf{V}\hat{\mathbf{B}}, \mathbf{W}\hat{\mathbf{C}}\right]\!\right]$ in which the factor matrices $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$, and $\hat{\mathbf{C}}$ correspond to the CPD of the small core tensor $\mathcal{S}$. For large tensors, using the SVD to compute the MLSVD can be too expensive, and it may be necessary to use randomized SVDs [82, 33] or cross approximation [83, 84, 85] instead. Coupled datasets can be compressed jointly [77, 86].

Rather than computing the MLSVD and then creating an orthogonal projection onto the dominant subspaces, PARACOMP [87] uses random compression matrices to create multiple, randomly compressed smaller tensors. The CPDs of each of these smaller tensors are computed separately, and the results are merged using anchor rows to resolve permutation and scaling ambiguity. The multiway compressed sensing method [88] uses

a single random compressed sample to recover the CPD, assuming that both the tensor and the factors are sparse.

When constraints or coupling are involved, decomposing the core tensor may no longer be possible as the constraint or coupling relation may not be preserved by the compression. Consider, for example, a nonnegativity constraint, i.e., $\mathbf{A} \geq 0$, in which $\geq$ holds entry-wise, and the compression matrix $\mathbf{U}$ such that $\mathbf{U}\hat{\mathbf{A}} = \mathbf{A}$, then the constraint $\hat{\mathbf{A}} \geq 0$ does not imply that $\mathbf{A} \geq 0$. The structured tensor decomposition framework proposed in [62] avoids this by exploiting the efficient representation of a tensor while keeping the original factor matrices, i.e., $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, in the optimization problem. Instead of working with the original $\mathcal{T}$, its truncated MLSVD $\mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W}$ is used and the multilinear structure of the MLSVD is exploited in all computations to reduce the computational complexity. Figure 14 illustrates the difference with the CANDELINC model. Efficient implementations that exploit the multilinear structure of the MLSVD are, e.g., given in [60, 62, 89]. The structured tensor framework also allows other compression types such as TT compression [62].

6.2. Sampling: incompleteness, randomization and updating

If the data is too large to be stored in memory or too expensive to measure or generate, it can be beneficial to avoid loading the entire tensor into memory at once, or even to avoid constructing all tensor entries. Here, we discuss techniques based on incomplete tensors, (randomized) block sampling and updating.

Incomplete tensors are used when some of the data is missing, e.g., due to sensor malfunction or artifact removal, or when the data is deliberately sampled in order to avoid the cost of generating all data. Two techniques are commonly used: single or repeated imputation, e.g., in an expectation maximization setting, and using a weight/sampling/observation tensor. The former technique imputes each unknown or missing entry with an estimate, thereby creating a full tensor which may not be feasible in a large-scale context; see [90, 34, 91]. The latter technique (implicitly) uses a binary weight tensor $\mathcal{W}$ in which an entry is one only if the corresponding entry in $\mathcal{T}$ is known, and solves

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \frac{1}{2} ||\mathcal{W} * (\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket - \mathcal{T})||_{\mathrm{F}}^2 ,$$

hence unknown entries are effectively ignored; see, e.g., [32, 91, 92, 93, 94].

When decomposing an incomplete tensor, only a part of the information is used. If all entries are available (explicitly or implicitly), it is also possible to repeatedly sample a number of entries and to compute an update for each sample, as is done in stochastic gradient descent [10, 76, 95], for example. In the case of a CPD, it is advantageous to sample subtensors, or *blocks*, thanks to the locality principle for a CPD: only a limited number of variables influences a block as can be seen in Figure 15 [96]. The ParCube algorithm [97] uses biased sampling to create a number of random blocks, decomposes these blocks using ALS and merges the results using anchor rows. In contrast to ParCube, the randomized block sampling (RBS) method [96] samples a block and computes only a single GN update using (5) and then samples a new block. RBS can achieve a high accuracy through step restriction strategies.

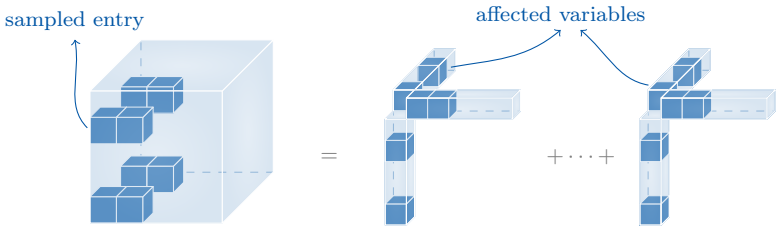


Figure 15: Thanks to the locality principle, only few variables in the rank-1 terms affect the sampled block or subtensor. The CPD RBS method [96] samples a new random block every iteration, while ParCube [97] samples a number of blocks, decomposes them and uses anchor rows to merge the results.

Updating algorithms can be used to track tensor decompositions that change over time [98, 99, 100], but can also be used to decompose large-scale tensors as follows. Rather than loading the whole tensor into memory at once, a smaller subtensor is decomposed first. The subtensor can then be discarded, and a new slice is loaded and used to update the current decomposition. This process is then repeated until all slices have been added. Hence, at any given iteration, only the factorization constructed using the previous slices and one new slice are in memory; see [100].

6.3. Exploiting structure: sparsity and implicit tensorization

Tensors are often structured and can therefore be represented by few parameters, e.g., all values in a CPD depend on the factor matrices, all entries in a sparse tensor are defined by the positions and values of nonzeros, and a Hankel tensor is determined by one generating vector. By designing specialized algorithms, the per-iteration complexity can be reduced from proportional to the number of entries to proportional to the number of parameters [62].

For sparse tensors, a large number of ALS algorithms have been devised, many of which focus on the efficient implementation of the MTKRPROD operation: only the Khatri–Rao product contributions corresponding nonzeros are computed to reduce the complexity. Instead of storing the multilinear index and the value for each nonzero [60], compressed formats such as the compressed sparse fiber (CSF) format can be used as well [61].

In the case the tensor is given by the factors of a decomposition such as a CPD, an MLSVD or a TT, the multilinear structure can be exploited when computing, e.g., the inner product, norm and MTKRPROD, without actually constructing the full tensor [60, 62, 89]. A randomized technique requiring a CPD as input tensor is presented in [101]. A similar approach can be used for a tensor that is the result of a tensorization technique such as Hankelization or Löwnerization: the necessary expressions can, e.g., be computed efficiently by exploiting properties of Hankel and Toeplitz matrices through fast Fourier transforms (FFT) [62].

6.4. Parallelization

Distributed computing and parallelization can be used to alleviate the curse of dimensionality by allocating more computational resources. (The curse is not removed, however.) The entries of the tensor are distributed over multiple computational nodes

which can be cores, processors or different computers in a cluster. Every iteration, each node computes an update for a part of the variables, after which the result is communicated to all nodes that require this update. (Not every node requires every variable due to the locality principle; see Figure 15.) Various algorithms exist that differ on the type of tensor (sparse or dense) and on how the data, and therefore the variables, are distributed across the nodes. This distribution depends on a trade-off between computational cost (computing the update), balancing computations such that every node has the same amount of work, and communication cost (distributing updates). Coarse grained distribution schedules assign a set of rows from each factor matrix to a node and distribute the data accordingly such that each node has a part of the data; see, e.g., [102, 103, 104, 105]. Fine grained schedules compute optimal distributions to balance computational load and minimize computational cost; see, e.g., [106]. While coarse grained schedules are easy to determine, they can be suboptimal, e.g. due to an imbalance of the computational load. Albeit optimal scheme are obtained with fine grained schedules, they can be (too) expensive to determine. Medium grained schedules such as [107] determine a better distribution while limiting the cost of computing this distribution. Finally, some algorithms focus on minimizing the communication cost; see, e.g., [59].

7. References

[1] I. Van Mechelen, A. K. Smilde, A generic linked-mode decomposition model for data fusion, Chemometr. Intell. Lab. 104 (1) (2010) 83–94, doi:10.1016/j.chemolab.2010.04.012.

[2] T. Adalı, Y. Levin-Schwartz, V. D. Calhoun, Multimodal data fusion using source separation: Application to medical imaging, Proc. IEEE 103 (9) (2015) 1494–1506, doi:10.1109/jproc.2015.2461601.

[3] S. Van Eyndhoven, B. Hunyadi, L. De Lathauwer, S. Van Huffel, Flexible fusion of electroencephalography and functional magnetic resonance imaging: Revealing neural-hemodynamic coupling through structured matrix-tensor factorization, 2017 25th European Signal Processing Conference (EUSIPCO) doi:10.23919/eusipco.2017.8081162.

[4] A. S. Zamzam, V. N. Ioannidis, N. D. Sidiropoulos, Coupled graph tensor factorization, 2016 50th Asilomar Conference on Signals, Systems and Computers doi:10.1109/acssc.2016.7869683.

[5] B. Ermiş, E. Acar, A. T. Cemgil, Link prediction in heterogeneous data via generalized coupled tensor factorization, Data Mining and Knowledge Discovery 29 (1) (2013) 203–236, doi:10.1007/s10618-013-0341-y.

[6] M. Sørensen, L. De Lathauwer, Multidimensional harmonic retrieval via coupled canonical polyadic decomposition — Part I: Model and identifiability, IEEE Trans. Signal Process. 65 (2) (2017) 517–527, doi:10.1109/TSP.2016.2614796.

[7] M. Sørensen, L. De Lathauwer, Multidimensional harmonic retrieval via coupled canonical polyadic decomposition — Part II: Algorithm and multirate sampling, IEEE Trans. Signal Process. 65 (2) (2017) 528–539, doi:10.1109/tsp.2016.2614797.

[8] M. Sørensen, L. De Lathauwer, Multiple invariance ESPRIT for nonuniform linear arrays: A coupled canonical polyadic decomposition approach, IEEE Trans. Signal Process. 64 (14) (2016) 3693–3704, doi:10.1109/tsp.2016.2551686.

[9] A. Cichocki, D. Mandic, A.-H. Phan, C. Caiafa, G. Zhou, Q. Zhao, L. De Lathauwer, Tensor decompositions for signal processing applications: From two-way to multiway component analysis, IEEE Signal Process. Mag. 32 (2) (2015) 145–163, doi:10.1109/msp.2013.2297439.

[10] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, C. Faloutsos, Tensor decomposition for signal processing and machine learning, IEEE Trans. Signal Process. 65 (13) (2017) 3551–3582, doi:10.1109/TSP.2017.2690524.

[11] R. Harshman, Foundations of the PARAFAC procedure: models and conditions for an "explanatory" multimodal factor analysis, UCLA Working Papers in Phonetics 16 (1970) 1–84.

[12] J. D. Carroll, J.-J. Chang, Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart–Young" decomposition, Psychometrika 35 (3) (1970) 283–319, doi:10.1007/bf02310791.

[13] L. De Lathauwer, Decompositions of a higher-order tensor in block terms — Part II: Definitions and uniqueness, SIAM J. Matrix Anal. Appl. 30 (3) (2008) 1033–1066, doi:10.1137/070690729.
[14] R. Bro, R. A. Harshman, N. D. Sidiropoulos, M. E. Lundy, Modeling multi-way data with linearly dependent loadings, J. Chemometrics 23 (7-8) (2009) 324–340, doi:10.1002/cem.1206.
[15] L. De Lathauwer, B. De Moor, J. Vandewalle, A multilinear singular value decomposition, SIAM J. Matrix Anal. Appl. 21 (4) (2000) 1253–1278, doi:10.1137/S0895479896305696.
[16] L. R. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika 31 (3) (1966) 279–311, doi:10.1007/bf02289464.
[17] O. Debals, L. De Lathauwer, The concept of tensorization, Technical Report 17–99, ESAT-STADIUS, KU Leuven, Belgium, 2017.
[18] D. Lahat, T. Adalı, C. Jutten, Multimodal data fusion: An overview of methods, challenges, and prospects, Proc. IEEE 103 (9) (2015) 1449–1477, doi:10.1109/jproc.2015.2460697.
[19] M. Sørensen, L. De Lathauwer, P. Comon, S. Icart, L. Deneire, Canonical polyadic decomposition with a columnwise orthonormal factor matrix, SIAM J. Matrix Anal. Appl. 33 (4) (2012) 1190–1213, doi:10.1137/110830034.
[20] M. Sørensen, L. De Lathauwer, Blind signal separation via tensor decomposition with Vandermonde factor: Canonical polyadic decomposition, IEEE Trans. Signal Process. 61 (22) (2013) 5507–5519, doi:10.1109/tsp.2013.2276416.
[21] A. Cichocki, R. Zdunek, A.-H. Phan, S.-I. Amari, Nonnegative matrix and tensor factorizations: Applications to exploratory multi-way data analysis and blind source separation, John Wiley, Chichester, U.K, 2009.
[22] S. E. Leurgans, R. T. Ross, R. B. Abel, A decomposition for three-way arrays, SIAM J. Matrix Anal. Appl. 14 (4) (1993) 1064–1083, doi:10.1137/0614071.
[23] I. Domanov, L. De Lathauwer, Canonical polyadic decomposition of third-order tensors: Reduction to generalized eigenvalue decomposition, SIAM J. Matrix Anal. Appl. 35 (2) (2014) 636–660, doi:10.1137/130916084.
[24] M. Sørensen, L. De Lathauwer, Tensor decompositions with block-Toeplitz structure and applications in signal processing, 2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR) doi:10.1109/acssc.2011.6190040.
[25] M. Sørensen, I. Domanov, L. De Lathauwer, Coupled canonical polyadic decompositions and (coupled) decompositions in multilinear rank-$(L_{r,n}, L_{r,n}, 1)$ terms — Part II: Algorithms, SIAM J. Matrix Anal. Appl. 36 (3) (2015) 1015–1045, doi:10.1137/140956865.
[26] L. Sorber, M. Van Barel, L. De Lathauwer, Unconstrained optimization of real functions in complex variables, SIAM J. Optim. 22 (3) (2012) 879–898, doi:10.1137/110832124.
[27] R. Bro, Multi-way analysis in the food industry: models, algorithms, and applications, Ph.D. thesis, University of Amsterdam, 1998.
[28] G. Tomasi, R. Bro, A comparison of algorithms for fitting the PARAFAC model, Comput. Stat. Data Anal. 50 (7) (2006) 1700–1734, doi:10.1016/j.csda.2004.11.013.
[29] E. Acar, D. M. Dunlavy, T. G. Kolda, A scalable optimization approach for fitting canonical tensor decompositions, J. Chemometrics 25 (2) (2011) 67–86, doi:10.1002/cem.1335.
[30] A.-H. Phan, P. Tichavský, A. Cichocki, Low complexity damped Gauss–Newton algorithms for CANDECOMP/PARAFAC, SIAM J. Matrix Anal. Appl. 34 (1) (2013) 126–147, doi:10.1137/100808034.
[31] E. Acar, T. G. Kolda, D. M. Dunlavy, All-at-once optimization for coupled matrix and tensor factorizations, arXiv:1105.3422, 2011.
[32] L. Sorber, M. Van Barel, L. De Lathauwer, Structured data fusion, IEEE J. Sel. Topics Signal Process. 9 (4) (2015) 586–600, doi:10.1109/jstsp.2015.2400415.
[33] N. Vervliet, O. Debals, L. De Lathauwer, Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization, in: 2016 50th Asilomar Conference on Signals, Systems and Computers, 1733–1738, doi:10.1109/ACSSC.2016.7869679, 2016.
[34] E. E. Papalexakis, T. M. Mitchell, N. D. Sidiropoulos, C. Faloutsos, P. P. Talukdar, B. Murphy, Turbo-SMT: Parallel coupled sparse matrix-tensor factorizations and applications, Statistical Analysis and Data Mining: The ASA Data Science Journal 9 (4) (2016) 269–290, doi:10.1002/sam.11315.
[35] P. Paatero, A weighted non-negative least squares algorithm for three-way "PARAFAC" factor analysis, Chemometr. Intell. Lab. 38 (2) (1997) 223–242, doi:10.1016/s0169-7439(97)00031-2.
[36] R. Bro, S. De Jong, A fast non-negativity-constrained least squares algorithm, J. Chemometrics 11 (5) (1997) 393–401, doi:10.1002/(sici)1099-128x(199709/10)11:5<393::aid-cem483>3.3.co;2-c.
[37] A. Uschmajew, Local convergence of the alternating least squares algorithm for canonical tensor

approximation, SIAM J. Matrix Anal. Appl. 33 (2) (2012) 639–652, doi:10.1137/110843587.
[38] L. Sorber, M. Van Barel, L. De Lathauwer, Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization, SIAM J. Optim. 23 (2) (2013) 695–720, doi:10.1137/120868323.
[39] P. Comon, X. Luciani, A. L. F. de Almeida, Tensor decompositions, alternating least squares and other tales, J. Chemometrics 23 (7-8) (2009) 393–405, doi:10.1002/cem.1236.
[40] M. J. Mohlenkamp, Musings on multilinear fitting, Linear Algebra Appl. 438 (2) (2013) 834–852, doi:10.1016/j.laa.2011.04.019.
[41] E. C. Chi, T. G. Kolda, On tensors, sparsity, and nonnegative factorizations, SIAM J. Matrix Anal. Appl. 33 (4) (2012) 1272–1299, doi:10.1137/110859063.
[42] S. Hansen, T. Plantenga, T. G. Kolda, Newton-based optimization for Kullback–Leibler nonnegative tensor factorizations, Optimization Methods and Software 30 (5) (2015) 1002–1029, doi:10.1080/10556788.2015.1009977.
[43] K. Y. Yılmaz, A. T. Cemgil, U. ŞimŞekli, Generalised coupled tensor factorisation, in: J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 24, Curran Associates, Inc., 2151–2159, 2011.
[44] J. E. Cohen, R. Cabral Farias, P. Comon, Fast decomposition of large nonnegative tensors, IEEE Signal Process. Lett. 22 (7) (2015) 862–866, doi:10.1109/lsp.2014.2374838.
[45] U. Şimşekli, A. T. Cemgil, B. Ermiş, Learning mixed divergences in coupled matrix and tensor factorization models, 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) doi:10.1109/icassp.2015.7178345.
[46] A. Cichocki, R. Zdunek, S. Choi, R. Plemmons, S.-I. Amari, Non-negative tensor factorization using alpha and beta divergences, 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07 doi:10.1109/icassp.2007.367106.
[47] Y.-D. Kim, A. Cichocki, S. Choi, Nonnegative Tucker decomposition with alpha-divergence, 2008 IEEE International Conference on Acoustics, Speech and Signal Processing doi:10.1109/icassp.2008.4517988.
[48] A.-H. Phan, A. Cichocki, Fast and efficient algorithms for nonnegative Tucker decomposition, Advances in Neural Networks - ISNN 2008 (2008) 772–782doi:10.1007/978-3-540-87734-9_88.
[49] I. V. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput. 33 (5) (2011) 2295–2317, doi:10.1137/090752286.
[50] C. Kelley, Iterative Methods for Optimization, SIAM, 1999.
[51] J. Nocedal, S. J. Wright, Numerical Optimization, Springer, New York, second edition edn., 2006.
[52] T. Adalı, P. J. Schreier, L. L. Scharf, Complex-valued signal processing: The proper way to deal with impropriety, IEEE Trans. Signal Process. 59 (11) (2011) 5101–125, doi:10.1109/tsp.2011.2162954.
[53] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, L. De Lathauwer, Tensorlab 3.0, available online at `https://www.tensorlab.net`, 2016.
[54] G. H. Golub, C. F. Van Loan, Matrix Computations, Johns Hopkins University Press, 2012.
[55] L. N. Trefethen, D. Bau, III, Numerical Linear Algebra, SIAM, 1997.
[56] L. Sorber, M. Van Barel, L. De Lathauwer, Complex optimization toolbox v1.0, available online at `https://www.esat.kuleuven.be/sista/cot/`, 2013.
[57] N. Vannieuwenhoven, K. Meerbergen, R. Vandebril, Computing the gradient in optimization algorithms for the CP decomposition in constant memory through tensor blocking, SIAM J. Sci. Comput. 37 (3) (2015) C415–C438, doi:10.1137/14097968x.
[58] A.-H. Phan, P. Tichavsky, A. Cichocki, Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations, IEEE Trans. Signal Process. 61 (19) (2013) 4834–4846, doi:10.1109/tsp.2013.2269903.
[59] G. Ballard, N. Knight, K. Rouse, Communication lower bounds for matricized tensor times Khatri–Rao product, arXiv:1708.07401, 2017.
[60] B. W. Bader, T. G. Kolda, Efficient matlab computations with sparse and factored tensors, SIAM J. Sci. Comput. 30 (1) (2008) 205–231, doi:10.1137/060676489.
[61] S. Smith, N. Ravindran, N. D. Sidiropoulos, G. Karypis, SPLATT: Efficient and parallel sparse tensor-matrix multiplication, in: 2015 IEEE International Parallel and Distributed Processing Symposium, IEEE, 61–70, doi:10.1109/ipdps.2015.27, 2015.
[62] N. Vervliet, O. Debals, L. De Lathauwer, Exploiting efficient representations in tensor decompositions, Technical Report 16–174, ESAT-STADIUS, KU Leuven, Belgium, 2016.
[63] L. Sorber, I. Domanov, M. Van Barel, L. De Lathauwer, Exact line and plane search for tensor optimization, Comput. Optim. Appl. 63 (1) (2015) 121–142, doi:10.1007/s10589-015-9761-5.

[64] M. Rajih, P. Comon, R. A. Harshman, Enhanced line search: A novel method to accelerate PARAFAC, SIAM J. Matrix Anal. Appl. 30 (3) (2008) 1128–1147, doi:10.1137/06065577.
[65] J.-P. Royer, N. Thirion-Moreau, P. Comon, Computing the polyadic decomposition of nonnegative third order tensors, Signal Processing 91 (9) (2011) 2159–2171, doi:10.1016/j.sigpro.2011.03.006.
[66] Y. Zheng, L. Zhang, X. Xie, W.-Y. Ma, Mining interesting locations and travel sequences from GPS trajectories, Proceedings of the 18th international conference on World wide web - WWW '09 doi:10.1145/1526709.1526816.
[67] O. Debals, F. Van Eeghem, N. Vervliet, L. De Lathauwer, Tensorlab demos — Release 3.0, Technical Report 16–68, ESAT-STADIUS, KU Leuven, Belgium, 2016.
[68] M. Sørensen, L. De Lathauwer, Fiber sampling approach to canonical polyadic decomposition and tensor completion, Technical Report 15-151, ESAT-STADIUS, KU Leuven, Belgium, 2015.
[69] M. Sørensen, F. Van Eeghem, L. De Lathauwer, Blind multichannel deconvolution and convolutive extensions of canonical polyadic and block term decompositions, IEEE Trans. Signal Process. 65 (15) (2017) 4132–4145, doi:10.1109/tsp.2017.2706183.
[70] J. B. Kruskal, Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics, Linear Algebra Appl. 18 (2) (1977) 95–138, doi:10.1016/0024-3795(77)90069-6.
[71] I. Domanov, L. De Lathauwer, On the uniqueness of the canonical polyadic decomposition of third-order tensors — Part II: Uniqueness of the overall decomposition, SIAM J. Matrix Anal. Appl. 34 (3) (2013) 876–903, doi:10.1137/120877258.
[72] I. Domanov, L. De Lathauwer, Canonical polyadic decomposition of third-order tensors: Relaxed uniqueness conditions and algebraic algorithm, Linear Algebra Appl. 513 (2017) 342–375, doi:10.1016/j.laa.2016.10.019.
[73] I. Domanov, L. De Lathauwer, Generic uniqueness conditions for the canonical polyadic decomposition and INDSCAL, SIAM J. Matrix Anal. Appl. 36 (4) (2015) 1567–1589, doi:10.1137/140970276.
[74] M. Sørensen, L. De Lathauwer, Coupled canonical polyadic decompositions and (coupled) decompositions in multilinear rank-$(L_{r,n}, L_{r,n}, 1)$ terms — Part I: Uniqueness, SIAM J. Matrix Anal. Appl. 36 (2) (2015) 496–522, doi:10.1137/140956853.
[75] L. De Lathauwer, E. Kofidis, Coupled matrix-tensor factorizations — The case of partially shared factors, Technical Report 17-171, ESAT-STADIUS, KU Leuven, Belgium. Accepted for publication in 2017 51th Asilomar Conference on Signals, Systems and Computers, 2017.
[76] D. Choi, J.-G. Jang, U. Kang, Fast, accurate, and scalable method for sparse coupled matrix-tensor factorization, arXiv:1708.08640, 2017.
[77] R. Cabral Farias, J. E. Cohen, P. Comon, Exploring multimodal data fusion through joint decompositions with flexible couplings, IEEE Trans. Signal Process. 64 (18) (2016) 4830–4844, doi:10.1109/tsp.2016.2576425.
[78] E. Acar, E. E. Papalexakis, G. Gürdeniz, M. A. Rasmussen, A. J. Lawaetz, M. Nilsson, R. Bro, Structure-revealing data fusion, BMC Bioinformatics 15 (1) (2014) 239, doi:10.1186/1471-2105-15-239.
[79] N. Vervliet, O. Debals, L. Sorber, L. De Lathauwer, Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis, IEEE Signal Process. Mag. 31 (5) (2014) 71–79, doi:10.1109/MSP.2014.2329429.
[80] R. Bro, C. A. Andersson, Improving the speed of multiway algorithms: Part II: Compression, Chemometr. Intell. Lab. 42 (1–2) (1998) 105–113, doi:10.1016/S0169-7439(98)00011-2.
[81] J. D. Carroll, S. Pruzansky, J. B. Kruskal, CANDELINC: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters, Psychometrika 45 (1) (1980) 3–24, doi:10.1007/bf02293596.
[82] N. Halko, P. G. Martinsson, J. A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2) (2011) 217–288, doi:10.1137/090771806.
[83] C. F. Caiafa, A. Cichocki, Generalizing the column–row matrix decomposition to multi-way arrays, Linear Algebra Appl. 433 (3) (2010) 557–573, doi:10.1016/j.laa.2010.03.020.
[84] M. Mahoney, M. Maggioni, P. Drineas, Tensor-CUR decompositions for tensor-based data, SIAM J. Matrix Anal. Appl. 30 (3) (2008) 957–987, doi:10.1137/060665336.
[85] I. V. Oseledets, D. V. Savostianov, E. E. Tyrtyshnikov, Tucker dimensionality reduction of three-dimensional arrays in linear time, SIAM J. Matrix Anal. Appl. 30 (3) (2008) 939–956, doi:10.1137/060655894.
[86] L. De Lathauwer, J. Vandewalle, Dimensionality reduction in higher-order signal processing and rank-$(R_1, R_2, \ldots, R_N)$ reduction in multilinear algebra, Linear Algebra Appl. 391 (2004) 31–55,

doi:10.1016/j.laa.2004.01.016.
[87] N. D. Sidiropoulos, E. E. Papalexakis, C. Faloutsos, Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition, IEEE Signal Process. Mag. 31 (5) (2014) 57–70, doi:10.1109/msp.2014.2329196.
[88] N. D. Sidiropoulos, A. Kyrillidis, Multi-way compressed sensing for sparse low-rank tensors, IEEE Signal Process. Lett. 19 (11) (2012) 757–760, doi:10.1109/lsp.2012.2210872.
[89] G. Zhou, A. Cichocki, Q. Zhao, S. Xie, Nonnegative matrix and tensor factorizations: An algorithmic perspective, IEEE Signal Process. Mag. 31 (3) (2014) 54–65, doi:10.1109/msp.2014.2298891.
[90] R. A. Harshman, M. E. Lundy, PARAFAC: Parallel factor analysis, Computational Statistics & Data Analysis 18 (1) (1994) 390–72, doi:10.1016/0167-9473(94)90132-5.
[91] G. Tomasi, R. Bro, Parafac and missing values, Chemometr. Intell. Lab. 75 (2) (2005) 163–180, doi:10.1016/j.chemolab.2004.07.003.
[92] E. Acar, D. M. Dunlavy, T. G. Kolda, M. Mørup, Scalable tensor factorizations for incomplete data, Chemometr. Intell. Lab. 106 (1) (2011) 41–56, doi:10.1016/j.chemolab.2010.08.004.
[93] L. Karlsson, D. Kressner, A. Uschmajew, Parallel algorithms for tensor completion in the CP format 57 (2016) 222–234, doi:10.1016/j.parco.2015.10.002.
[94] N. Vervliet, O. Debals, L. De Lathauwer, Canonical polyadic decomposition of incomplete tensors with linearly constrained factors, Technical Report 16–172, ESAT-STADIUS, KU Leuven, Belgium, 2016.
[95] R. Ge, F. Huang, C. Jin, Y. Yuan, Escaping from saddle points — Online stochastic gradient for tensor decomposition, in: Proceedings of The 28th Conference on Learning Theory, 797–842, 2015.
[96] N. Vervliet, L. De Lathauwer, A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors, IEEE J. Sel. Topics Signal Process. 10 (2) (2016) 284–295, doi:10.1109/JSTSP.2015.2503260.
[97] E. E. Papalexakis, C. Faloutsos, N. D. Sidiropoulos, ParCube: Sparse parallelizable CANDECOMP-PARAFAC tensor decomposition, ACM Trans. Knowl. Discov. Data 10 (1) (2015) 1–25, doi:10.1145/2729980.
[98] D. Nion, N. D. Sidiropoulos, Adaptive algorithms to track the PARAFAC decomposition of a third-order tensor, IEEE Trans. Signal Process. 57 (6) (2009) 2299–2310, doi:10.1109/TSP.2009.2016885.
[99] J. Sun, D. Tao, S. Papadimitriou, P. S. Yu, C. Faloutsos, Incremental tensor analysis: Theory and applications, ACM Trans. Knowl. Discov. Data 2 (3) (2008) 11:1–11:37.
[100] M. Vandecappelle, N. Vervliet, L. De Lathauwer, Nonlinear least squares updating of the canonical polyadic decomposition, in: 2017 25th European Signal Processing Conference (EUSIPCO17), 693–697, doi:10.23919/EUSIPCO.2017.8081290, 2017.
[101] M. J. Reynolds, A. Doostan, G. Beylkin, Randomized alternating least squares for canonical tensor decompositions: Application to a PDE with random data, SIAM J. Sci. Comput. 38 (5) (2016) A2634–A2664, doi:10.1137/15m1042802.
[102] J. H. Choi, S. V. N. Vishwanathan, DFacTo: Distributed factorization of tensors, in: Proceedings of the 27th International Conference on Neural Information Processing Systems, NIPS'14, MIT Press, Cambridge, MA, USA, 1296–1304, 2014.
[103] K. Shin, U. Kang, Distributed methods for high-dimensional and large-scale tensor factorization, in: Data Mining (ICDM), 2014 IEEE International Conference on, 989–994, doi:10.1109/ICDM.2014.78, 2014.
[104] U. Kang, E. E. Papalexakis, A. Harpale, 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 doi:10.1145/2339530.2339583.
[105] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, N. D. Sidiropoulos, Nesterov-based parallel algorithm for large-scale nonnegative tensor factorization, 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) doi:10.1109/icassp.2017.7953287.
[106] O. Kaya, B. Uçar, Scalable sparse tensor decompositions in distributed memory systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15, ACM, 77:1–77:11, doi:10.1145/2807591.2807624, 2015.
[107] S. Smith, G. Karypis, A medium-grained algorithm for sparse tensor factorization, in: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 902–911, doi:10.1109/ipdps.2016.113, 2016.