A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost

Michiel Vandecappelle*†, *Student member, IEEE,* Nico Vervliet*†, *Member, IEEE,* and Lieven De Lathauwer*†, *Fellow, IEEE*

***Abstract*—The canonical polyadic decomposition (CPD) can be used to extract meaningful components from a tensor. Most existing optimization methods for fitting the CPD use as cost function the least-squares distance between the tensor and its CPD. While the minimum of this cost function coincides with the maximum likelihood estimator for data with additive i.i.d. Gaussian distributed noise, for other noise distributions, better-suited cost functions exist. For such cost functions, first-order, gradient-based optimization methods have been proposed. However, (approximate) second-order methods, which additionally use information from the Hessian of the cost function to achieve faster convergence, are still largely unexplored. In this paper, we generalize the Gauss–Newton nonlinear least-squares algorithm to twice differentiable entry-wise cost functions. The low-rank structure of the problem is exploited to keep the computational cost low. As a special case, β-divergence cost functions are examined. We show that quadratic convergence can be obtained close to the solution with a reasonable extra cost in memory and computation time, making the proposed method particularly useful when high accuracy of the decomposition is desired.**

***Index Terms*—Tensors, CPD, beta-divergence.**

I. Introduction

Tensors—higher-order generalizations of matrices—are used extensively in machine learning and signal processing [1], [2]. In contrast to matrices, tensors preserve higher-order structure that is present in the data. Building on the well-established field of matrix decompositions, different types of tensor decompositions have been proposed to reveal the underlying information in the tensor in a compact way. Several algebraic and optimization-based algorithms have been developed for tensor decompositions such as the canonical polyadic decomposition (CPD) [3]–[6]. The CPD is a popular tensor decomposition due to its simplicity and mild uniqueness conditions [7], [8] and therefore, its computation has been intensively studied. Still, optimization-based algorithms for computing the CPD of a tensor have mainly considered least-squares (LS) cost functions. They minimize the Frobenius norm of the difference between the tensor and its CPD model. This is indeed optimal if the noise on the tensor data is additive, independent and identically distributed (i.i.d.) and follows a Gaussian distribution, as the LS solution is the maximum likelihood estimator (MLE) under these assumptions for the noise [9]. However, using a different criterion could be better if the noise is differently distributed [10]–[12]. While methods exist that allow the use of β-divergence [11], [12] or general [10] cost functions, these are all first-order methods that use only gradient information to determine the next optimization step. Second-order methods additionally use curvature information from the Hessian of the cost function. Consequently, every iteration is typically more costly than for first-order methods, but the use of second-order information can lead to fewer iterations and better convergence in general. In practice, the robustness with respect to initialization is often increased as well. If used naively, however, these methods quickly become prohibitively expensive in computation time and memory. We therefore propose a method for computing the CPD for arbitrary twice differentiable entry-wise cost functions that uses Hessian information and exploits the multilinear structure of the problem. The method obtains fast local convergence, while at the same time keeping storage and time requirements reasonably low.

We fix notation and give basic definitions in the remaining of this section. In Sec. II, we propose a second-order optimization algorithm to compute the CPD of a tensor for general cost functions. By exploiting the low-rank structure of the CPD, the curvature information in the Hessian can be used efficiently. We then focus on β-divergences as an example of an alternative cost function in Sec. III. Lastly, numerical experiments are shown in Sec. IV. We will consider only third-order tensors, although, with minor modifications, the results apply to higher-order tensors as well.

A. Notation

We denote scalars, vectors and matrices by lowercase (a), bold lowercase ($\mathbf{a}$) and bold uppercase letters ($\mathbf{A}$), respectively. Tensors are written in calligraphic script ($\mathcal{A}$). Powers, divisions and logarithms of matrices are entry-wise throughout the text. For matrices $\mathbf{X}$ and $\mathbf{Y}$ the former is written as $\mathbf{X}^{\bullet a}$, for the other two their scalar notation is used: $\frac{\mathbf{X}}{\mathbf{Y}}$ and $\log(\mathbf{X})$, respectively. The transpose of a matrix $\mathbf{X}$ is written as $\mathbf{X}^{\mathrm{T}}$ and $\mathrm{Diag}(\mathbf{x})$ forms a square matrix that has $\mathbf{x}$ as its diagonal. The $M \times M$-identity matrix is written as $\mathbf{I}_M$. When using subscripts to specify certain entries of a tensor

Funding: Michiel Vandecappelle is supported by an SB Grant (1S66617N) and Nico Vervliet by a Junior postdoctoral fellowship (12ZM220N) from the Research Foundation—Flanders (FWO). Research furthermore supported by: (1) Flemish Government: This work was supported by the Fonds de la Recherche Scientifique–FNRS and the Fonds Wetenschappelijk Onderzoek–Vlaanderen under EOS Project no 30468160 (SeLMA); (2) KU Leuven Internal Funds C16/15/059 and ID-N project no 3E190402. (3) This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

*KU Leuven, Dept. of Electrical Engineering ESAT/STADIUS, Kasteelpark Arenberg 10, bus 2446, B-3001 Leuven, Belgium

† KU Leuven – Kulak, Group Science, Engineering and Technology, E. Sabbelaan 53, B-8500 Kortrijk, Belgium

(Michiel.Vandecappelle, Nico.Vervliet, Lieven.DeLathauwer)@kuleuven.be.

$\mathcal{T}$, a colon refers to all entries of a specific mode, e.g. $\mathbf{t}_{1:2}$ is a vector holding all entries of the third-order tensor $\mathcal{T}$ which have 1 and 2 as their first and third index, respectively. The vectorization of the tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ maps t_{ijk} to $\mathbf{t} = \text{vec}(\mathcal{T})_q$, with $q = i + (j-1)I + (k-1)IJ$. Its inverse (with context-implied dimensions) is denoted by $\text{unvec}(\mathbf{t})$. The vectorization $\text{vec}(\mathbf{X}) \in \mathbb{R}^{IJ\times 1}$ of a matrix $\mathbf{X} \in \mathbb{R}^{I\times J}$ can be seen as a special case. A mode-n vector of a tensor is a vector obtained by fixing all indices but the nth. The mode-n unfolding $\mathbf{T}_{(n)}$ of a tensor $\mathcal{T}$ is constructed by stacking all its mode-n vectors as columns of a matrix. For example, $t_{i_1 i_2 i_3}$ is mapped to $\left(\mathbf{T}_{(2)}\right)_{i_2,q}$, with $q = i_1 + (i_3-1)I$. We also introduce the permutation matrices $\mathbf{\Pi}^{(n)} \in \{0,1\}^{IJK\times IJK}$ for vectorizations of $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$. The matrix $\mathbf{\Pi}^{(n)}$ permutes the modes of $\mathcal{T}$ directly in its vectorization by bringing the nth mode to the first position. Thus $\mathbf{\Pi}^{(1)} = \mathbf{I}_{IJK}$, and $\mathbf{\Pi}^{(2)}$ and $\mathbf{\Pi}^{(3)}$ change the mode order from $(1,2,3)$ to $(2,1,3)$ and $(3,1,2)$, respectively. The inverse permutation is denoted by $\mathbf{\Pi}^{(n)^{\mathrm{T}}}$, thus bringing the first mode to the nth position. Again, $\mathbf{\Pi}^{(1)^{\mathrm{T}}} = \mathbf{I}_{IJK}$, while $\mathbf{\Pi}^{(2)^{\mathrm{T}}}$ and $\mathbf{\Pi}^{(3)^{\mathrm{T}}}$ change the mode order from $(1,2,3)$ to $(2,1,3)$ and $(2,3,1)$, respectively.

The Kronecker product of matrices $\mathbf{A} \in \mathbb{R}^{I\times J}$ and $\mathbf{B} \in \mathbb{R}^{K\times L}$ is defined as

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{IK\times JL}.$$

For $\mathbf{A} \in \mathbb{R}^{I\times K}$ and $\mathbf{B} \in \mathbb{R}^{J\times K}$, the column-wise Khatri–Rao product is defined as $\mathbf{A}\odot\mathbf{B} = \begin{bmatrix}\mathbf{a}_{:1}\otimes\mathbf{b}_{:1} & \cdots & \mathbf{a}_{:K}\otimes\mathbf{b}_{:K}\end{bmatrix}$, while the row-wise Khatri–Rao product of $\mathbf{A} \in \mathbb{R}^{K\times I}$ and $\mathbf{B} \in \mathbb{R}^{K\times J}$ is $\mathbf{A}\odot^{\mathrm{T}}\mathbf{B} = \left(\mathbf{A}^{\mathrm{T}}\odot\mathbf{B}^{\mathrm{T}}\right)^{\mathrm{T}}$. The Hadamard, or entry-wise, product of $\mathbf{A}$ and $\mathbf{B} \in \mathbb{R}^{I\times J}$ is written as $\mathbf{A}*\mathbf{B}$. Finally, the outer product of vectors is denoted by $\otimes$. A third-order tensor is a rank-1 tensor if it is the outer product of three nonzero vectors. The minimal number of terms needed to write the tensor as a sum of rank-1 tensors, is the rank R of the tensor. Such a decomposition is called a CPD:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r\otimes\mathbf{b}_r\otimes\mathbf{c}_r =: [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!].$$

The column vectors $\mathbf{a}_r$, $\mathbf{b}_r$ and $\mathbf{c}_r$ can be collected as columns of factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, respectively.

II. General cost functions for the CPD

In this section, we show how one can use curvature information from the Hessian of the cost function without requiring an inordinate amount of computational resources. We first introduce a general cost function for fitting the CPD and derive efficient expressions for its gradient. Next, we explain that the Hessian approximation of the generalized Gauss–Newton method can be expected to perform well for the CPD and exploit the multilinear structure of the problem to compute this Hessian approximation cheaply.

Computing the CPD of a tensor is a difficult problem in general. The cost function is nonconvex and may have spurious local minima. For the LS cost function, optimization-based algorithms find a rank-R CPD $\mathcal{M} = [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$ of a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ by fitting matrices $\mathbf{A} \in \mathbb{R}^{I\times R}$, $\mathbf{B} \in \mathbb{R}^{J\times R}$, $\mathbf{C} \in \mathbb{R}^{K\times R}$ that solve $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f$, with

$$f = \min_{\mathbf{A},\mathbf{B},\mathbf{C}} \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} \frac{1}{2}\left(m_{ijk} - t_{ijk}\right)^2. \tag{1}$$

As the triple sum is simply a sum over all tensor entries, it can be replaced by a single sum $\sum_{n=1}^{N}$, where $N = IJK$. We also write $\mathbf{z} = [\text{vec}(\mathbf{A}); \text{vec}(\mathbf{B}); \text{vec}(\mathbf{C})] \in \mathbb{R}^{(I+J+K)R}$, for the vector holding all optimization variables. While the LS distance is by far the most popular cost function when fitting a CPD, its solution does not necessarily correspond to the MLE when the noise on the tensor entries is not additive i.i.d. Gaussian noise [10]. For more general settings, we may consider the following extension of (1):

$$\min_{\mathbf{z}} f = \min_{\mathbf{z}} \sum_{n=1}^{N} f_n(m_n(\mathbf{z})).$$

Note that, for the LS cost function in (1), $f_n = \frac{1}{2}\left(m_n(\mathbf{z}) - t_n\right)^2$. Here $m_n(\mathbf{z})$ is the (unconstrained) CPD model mapping the variables in $\mathbf{z}$ to the tensor elements in $\mathcal{M}$. In our discussion, the f_n may be different for different tensor entries. They can be any kind of twice differentiable fit function for the corresponding model entry m_n, although the problem may become significantly harder if the function is nonconvex. One class of alternative cost functions is given by β-divergence cost functions, with as special cases the Kullback–Leibler (KL) divergence $x\log\left(\frac{x}{y}\right) - x + y$ and the Itakura–Saito (IS) divergence $\frac{x}{y} - \log\left(\frac{x}{y}\right) - 1$, where the first one is convex and the second one is not. A second alternative cost function is the maximum likelihood estimator for data that follows a Rician distribution [13]. The entry-wise modulus of a complex-valued tensor follows such a Rician distribution when the real and imaginary parts are independently perturbed by additive Gaussian noise and they have equal noise variance. As a third alternative cost function, correntropy [14] can be considered. This function behaves as the LS distance if points are close, as the 1-norm distance if points are moderately far apart and as the 0-norm distance if they are far apart, which makes it robust against outliers. Naturally, many other entry-wise differentiable cost functions exist.

A. Gradient

The gradient $\mathbf{g} = \nabla_{\mathbf{z}} f$ can be split into terms according to the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$: $\mathbf{g} = [\mathbf{g}_{\mathbf{A}}; \mathbf{g}_{\mathbf{B}}; \mathbf{g}_{\mathbf{C}}]$. The subgradients $\mathbf{g}_{\mathbf{A}}$, $\mathbf{g}_{\mathbf{B}}$ and $\mathbf{g}_{\mathbf{C}}$ can be found by applying the chain rule. We first derive $\mathbf{g}_{\mathbf{A}}$:

$$\mathbf{g}_{\mathbf{A}} = \sum_{n=1}^{N} \frac{\mathrm{d}f_n}{\mathrm{d}m_n}\frac{\partial m_n}{\partial \text{vec}(\mathbf{A})}. \tag{2}$$

Note that all expressions of the form $\frac{\partial m_n}{\partial \text{vec}(\mathbf{A})}$ are transposed rows of the Jacobian matrix $\mathbf{J}_{\mathbf{A}} \in \mathbb{R}^{IJK\times IR}$ for the CPD:

$$\mathbf{J}_{\mathbf{A}} = \frac{\partial \text{vec}\left(\mathbf{A}(\mathbf{C}\odot\mathbf{B})^{\mathrm{T}}\right)}{\partial \text{vec}(\mathbf{A})^{\mathrm{T}}} = \begin{bmatrix} \frac{\partial m_1}{\partial \text{vec}(\mathbf{A})^{\mathrm{T}}} \\ \vdots \\ \frac{\partial m_N}{\partial \text{vec}(\mathbf{A})^{\mathrm{T}}} \end{bmatrix} = (\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I. \tag{3}$$

By substituting $\mathbf{J}_\mathbf{A} = (\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I$ into (2) and exploiting the multilinear structure, $\mathbf{g}_\mathbf{A}$ can be computed as

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

where the tensor $\mathcal{R}$ is of the same dimensions as $\mathcal{T}$ and $\mathcal{M}$ and holds the N derivatives $\frac{\partial f_n}{\partial m_n}$. For the LS cost function, $\mathcal{R}$ is simply the residual tensor $\mathcal{M} - \mathcal{T}$, which is particularly convenient computationally. For the aforementioned KL and IS divergences, the tensor $\mathcal{R}$ is equal to $\frac{\mathcal{M}-\mathcal{T}}{\mathcal{M}}$ and $\frac{\mathcal{M}-\mathcal{T}}{\mathcal{M}^{\bullet 2}}$, respectively. Using permutation matrices, the Jacobians with respect to $\mathbf{B}$ and $\mathbf{C}$ can be written as

$$\begin{aligned}\mathbf{J}_\mathbf{B} &= \mathbf{\Pi}^{(2)^\mathsf{T}}\left((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J\right), \text{ and} \\ \mathbf{J}_\mathbf{C} &= \mathbf{\Pi}^{(3)^\mathsf{T}}\left((\mathbf{B} \odot \mathbf{A}) \otimes \mathbf{I}_K\right). \end{aligned} \tag{4}$$

Using these expressions, the vectors $\mathbf{g}_\mathbf{B}$ and $\mathbf{g}_\mathbf{C}$ can be computed similarly to $\mathbf{g}_\mathbf{A}$:

$$\mathbf{g}_\mathbf{B} = \operatorname{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A})\right) \quad \text{and} \quad \mathbf{g}_\mathbf{C} = \operatorname{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A})\right).$$

The matricized tensor times Khatri–Rao products (`MTKRPROD`) in $\mathbf{g}_\mathbf{A}$, $\mathbf{g}_\mathbf{B}$ and $\mathbf{g}_\mathbf{C}$ can be computed efficiently using specialized routines [15]–[17].

B. Hessian approximation

While gradient descent methods use the direction of the negative gradient to obtain a new intermediate solution $\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha\mathbf{p}$, second-order optimization methods find the next iterate by solving the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$ for the step direction $\mathbf{p}$. They thus require, besides the gradient, also (a good approximation of) the Hessian $\mathbf{H} = \nabla^2 f \in \mathbb{R}^{(I+J+K)R \times (I+J+K)R}$. The Hessian can be computed as the Jacobian of the gradient $\mathbf{g}$, so we take the derivative of $\mathbf{g}$ with respect to the variables $\mathbf{z}$ again. The chain rule for derivatives gives:

$$\begin{aligned}\mathbf{H} &= \sum_{n=1}^{N} \frac{\partial}{\partial \mathbf{z}} \left(\frac{\mathrm{d}f_n}{\mathrm{d}m_n}\frac{\partial m_n}{\partial \mathbf{z}}\right) \\ &= \sum_{n=1}^{N} \frac{\partial}{\partial \mathbf{z}} \left(\frac{\mathrm{d}f_n}{\mathrm{d}m_n}\right)\frac{\partial m_n}{\partial \mathbf{z}} + \frac{\mathrm{d}f_n}{\mathrm{d}m_n}\frac{\partial}{\partial \mathbf{z}}\left(\frac{\partial m_n}{\partial \mathbf{z}}\right) \\ &= \sum_{n=1}^{N} \left(\frac{\partial m_n}{\partial \mathbf{z}}\right)\frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2}\left(\frac{\partial m_n}{\partial \mathbf{z}}\right)^\mathsf{T} + \frac{\mathrm{d}f_n}{\mathrm{d}m_n}\left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2}\right) \\ &= \sum_{n=1}^{N} \mathbf{J}_{n:}^\mathsf{T}\frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2}\mathbf{J}_{n:} + \frac{\mathrm{d}f_n}{\mathrm{d}m_n}\left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2}\right),\end{aligned}$$

where $\mathbf{J} = [\mathbf{J}_\mathbf{A}, \mathbf{J}_\mathbf{B}, \mathbf{J}_\mathbf{C}]$. Note that the second term of the Hessian consists of two factors. The factor $\frac{\mathrm{d}f_n}{\mathrm{d}m_n}$, i.e., the change in cost function, is expected to become small as we move closer to the (global) optimum. The factor $\left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2}\right)$ is sparse for the CPD model, as every tensor entry depends only on a limited number of model entries [4]. One can therefore argue that the generalized Gauss–Newton (GGN) method, which approximates the Hessian by only its first term $\sum_{n=1}^{N} \mathbf{J}_{n:}^\mathsf{T}\frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2}\mathbf{J}_{n:}$, will give a good approximation [17]–[19]. The values $\frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2}$ can be collected in a tensor $\mathcal{D}$ of the same dimensions as $\mathcal{T}, \mathcal{M}$ and $\mathcal{R}$. The Hessian approximation can then be written as $\mathbf{H} = \mathbf{J}^\mathsf{T}\mathbf{Z}\mathbf{J}$, where $\mathbf{Z} = \operatorname{Diag}(\operatorname{vec}(\mathcal{D})) \in \mathbb{R}^{IJK \times IJK}$. Note that, for the LS cost function, $\mathcal{D}$ has all entries equal to 1, which means that $\mathbf{Z} = \mathbf{I}_{IJK}$. For this cost function, the Hessian approximation is thus simply the Gramian of the Jacobian, and we recover the standard Gauss–Newton method for the CPD, as described in [4]. Although we would like the Hessian approximation $\mathbf{J}^\mathsf{T}\mathbf{Z}\mathbf{J}$ to be positive semidefinite (PSD), i.e., having only nonnegative eigenvalues, this does not hold in general. For nonconvex cost functions f_n, negative values can occur in $\mathbf{Z}$ when moving away from the optimum, possibly introducing negative eigenvalues in $\mathbf{J}^\mathsf{T}\mathbf{Z}\mathbf{J}$ as well. Special care should be taken when this occurs to ensure that a descent direction is chosen and continued convergence of the optimization algorithm is guaranteed. This can be done either by modifying f, by using an algorithm that can handle indefinite Hessian approximations or by modifying $\mathbf{J}^\mathsf{T}\mathbf{Z}\mathbf{J}$ to make it PSD. See Sec. III-A for an extended discussion. The step obtained from solving the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$ can be combined with the (first-order) Cauchy point within a (dogleg) trust-region approach [20] to guarantee global convergence of the method. In this case, the step is restricted to a small region around the previous iterate, wherein the local second-order approximation of the cost function can be trusted.

C. Computing $\mathbf{J}^\mathsf{T}\mathbf{Z}\mathbf{J}$

Naively computing the Hessian approximation $\mathbf{J}^\mathsf{T}\mathbf{Z}\mathbf{J}$ of a rank-R $(I \times I \times I)$-tensor requires the multiplication of the matrix $\mathbf{J}^\mathsf{T}$, the diagonal matrix $\mathbf{Z}$ and the matrix $\mathbf{J}$ and requires $\mathcal{O}(I^5R^2)$ flop, which already becomes costly for moderate values of I and R. However, by exploiting the sparsity and multilinear structure in $\mathbf{J}$, this computation can be made more efficient; see Sec. II-A. The full Hessian approximation $\mathbf{J}^\mathsf{T}\mathbf{Z}\mathbf{J}$ consists of nine blocks in a 3-by-3 grid. (For a tensor of order M, the Hessian approximation has M^2 blocks in a M-by-M grid.) The diagonal and off-diagonal blocks are of the form $\mathbf{J}_m^\mathsf{T}\mathbf{Z}\mathbf{J}_m$ and $\mathbf{J}_l^\mathsf{T}\mathbf{Z}\mathbf{J}_m$ with $m \neq l$, respectively, where $\mathbf{J}_1 = \mathbf{J}_\mathbf{A}$, $\mathbf{J}_2 = \mathbf{J}_\mathbf{B}$ and $\mathbf{J}_3 = \mathbf{J}_\mathbf{C}$. Due to symmetry, only the blocks with $m \geq l$ need to be computed. Efficient approaches to obtain the diagonal and off-diagonal blocks are given below.

1) Diagonal blocks: We show how the diagonal blocks of the GGN Hessian approximation can be computed while exploiting their specific structure and sparsity. For $m = 1$, the diagonal block $\mathbf{G}^{(1,1)}$ of $\mathbf{J}^\mathsf{T}\mathbf{Z}\mathbf{J} \in \mathbb{R}^{IR \times IR}$ is given by:

$$\mathbf{G}^{(1,1)} = \mathbf{J}_\mathbf{A}^\mathsf{T}\mathbf{Z}\mathbf{J}_\mathbf{A} = \left((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I\right)^\mathsf{T}\mathbf{Z}\left((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I\right).$$

The Kronecker product with the identity matrix $\mathbf{I}_I$ introduces many zeros into the Jacobian which in turn lead to zeros in $\mathbf{G}^{(1,1)}$. We can exploit this sparsity by only computing the nonzero values of $\mathbf{G}^{(1,1)}$. Denote by $\mathbf{G}_{\text{nz}}^{(1,1)}$ the matrix of dimensions $(I \times R^2)$ holding these nonzero values. One can obtain $\mathbf{G}_{\text{nz}}^{(1,1)}$ from the `MTKRPROD`

$$\mathbf{G}_{\text{nz}}^{(1,1)} = \mathbf{D}_{(1)}\left((\mathbf{C} \odot^\mathsf{T} \mathbf{C}) \odot (\mathbf{B} \odot^\mathsf{T} \mathbf{B})\right),$$

where the matrices $\mathbf{C} \odot^\mathsf{T} \mathbf{C} \in \mathbb{R}^{K \times R^2}$ and $\mathbf{B} \odot^\mathsf{T} \mathbf{B} \in \mathbb{R}^{J \times R^2}$ hold all pair-wise Hadamard products of columns of $\mathbf{C}$ and $\mathbf{B}$, respectively. The proof can be found in Appendix A. As $\mathbf{C} \odot^\mathsf{T} \mathbf{C}$ and $\mathbf{B} \odot^\mathsf{T} \mathbf{B}$ only have $R(R+1)/2$ unique columns

and $\mathbf{G}_{\text{nz}}^{(1,1)}$ has the same repetition pattern, the latter also has only $R(R+1)/2$ unique columns, which can be exploited. The matrix $\mathbf{G}^{(1,1)} \in \mathbb{R}^{IR \times IR}$ can then be obtained from $\mathbf{G}_{\text{nz}}^{(1,1)} \in \mathbb{R}^{I \times R^2}$ by reshaping and reintroducing the zeros. Similarly $\mathbf{G}_{\text{nz}}^{(2,2)}$ and $\mathbf{G}_{\text{nz}}^{(3,3)}$ are computed as

$$\mathbf{G}_{\text{nz}}^{(2,2)} = \mathbf{D}_{(2)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{A} \odot^{\mathrm{T}} \mathbf{A}) \right), \text{ and}$$
$$\mathbf{G}_{\text{nz}}^{(3,3)} = \mathbf{D}_{(3)} \left((\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}) \odot (\mathbf{A} \odot^{\mathrm{T}} \mathbf{A}) \right).$$

Intermediate results of the form $\mathbf{D}_{(3)}^{\mathrm{T}} \left(\mathbf{C} \odot^{\mathrm{T}} \mathbf{C} \right)$ can be reused instead of recomputing all MTKRPRODs from scratch using dimension trees [21], as $\mathbf{D}_{(1)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}) \right)$ and $\mathbf{D}_{(2)} \left((\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \odot (\mathbf{A} \odot^{\mathrm{T}} \mathbf{A}) \right)$ can be derived from it. These matrices are also needed for the off-diagonal blocks.

2) Off-diagonal blocks: The off-diagonal blocks of the Hessian approximation can be computed efficiently as well using a similar yet slightly more involved approach as for the diagonal blocks. The $(1,2)$ off-diagonal block of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ has dimensions $(IR \times JR)$ and is of the form

$$\begin{aligned}\mathbf{G}^{(1,2)} &= \mathbf{J}_{\mathbf{A}}^{\mathrm{T}} \mathbf{Z} \mathbf{J}_{\mathbf{B}} \\ &= \left((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I \right)^{\mathrm{T}} \mathbf{Z} \mathbf{\Pi}^{(2)^{\mathrm{T}}} \left((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J \right),\end{aligned}$$

where $\mathbf{J}_{\mathbf{A}}$ and $\mathbf{J}_{\mathbf{B}}$ are substituted using (3) and (4). In Appendix B, it is shown that $\mathbf{G}^{(1,2)}$ can be computed as

$$\mathbf{G}^{(1,2)} = \begin{bmatrix} \mathbf{a}_{:1}\mathbf{b}_{:1}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:1}^{\mathrm{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{:1}\mathbf{b}_{:R}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{b}_{:R}^{\mathrm{T}} \end{bmatrix} * \mathbf{W}^{(1,2)},$$

where the matrix $\mathbf{W}^{(1,2)}$ is obtained by rearranging the entries of the matrix $\mathbf{D}_{(3)}^{\mathrm{T}}(\mathbf{C} \odot^{\mathrm{T}} \mathbf{C}) \in \mathbb{R}^{IJ \times R^2}$ into an $(IR \times JR)$-matrix; see Appendix B.

For the $(1,3)$- and $(2,3)$-blocks of the Gramian, $\mathbf{G}^{(1,3)}$ and $\mathbf{G}^{(2,3)}$ are analogously computed as:

$$\mathbf{G}^{(1,3)} = \begin{bmatrix} \mathbf{a}_{:1}\mathbf{c}_{:1}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{c}_{:1}^{\mathrm{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{:1}\mathbf{c}_{:R}^{\mathrm{T}} & \cdots & \mathbf{a}_{:R}\mathbf{c}_{:R}^{\mathrm{T}} \end{bmatrix} * \mathbf{W}^{(1,3)},$$

and

$$\mathbf{G}^{(2,3)} = \begin{bmatrix} \mathbf{b}_{:1}\mathbf{c}_{:1}^{\mathrm{T}} & \cdots & \mathbf{b}_{:R}\mathbf{c}_{:1}^{\mathrm{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{b}_{:1}\mathbf{c}_{:R}^{\mathrm{T}} & \cdots & \mathbf{b}_{:R}\mathbf{c}_{:R}^{\mathrm{T}} \end{bmatrix} * \mathbf{W}^{(2,3)}.$$

Analogously to the $(1,2)$-block, the matrices $\mathbf{W}^{(1,3)}$ and $\mathbf{W}^{(2,3)}$ are now obtained by rearranging the entries of the matrices $\mathbf{D}_{(2)}^{\mathrm{T}}(\mathbf{B} \odot^{\mathrm{T}} \mathbf{B}) \in \mathbb{R}^{IK \times R^2}$ and $\mathbf{D}_{(1)}^{\mathrm{T}}(\mathbf{A} \odot^{\mathrm{T}} \mathbf{A}) \in \mathbb{R}^{JK \times R^2}$ into $(IR \times KR)$- and $(JR \times KR)$-matrices, respectively. As for the diagonal blocks, the computation of the MTKRPROD in $\mathbf{W}^{(1,2)}$ can be sped up by using dimension trees [21].

3) Complexity: The computational complexity of computing one diagonal block for an $(I \times I \times I)$-tensor is dominated by the complexity of the MTKRPROD that is used to compute $\mathbf{G}_{\text{nz}}^{(m,m)}$, which is $\mathcal{O}(I^3R^2)$ flop when computed straightforwardly. Likewise, the computation of one off-diagonal block is dominated by the MTKRPROD that is used to compute $\mathbf{W}^{(l,m)}$ and requires $\mathcal{O}(I^3R^2)$ flop. Adding the costs of the three diagonal and the three off-diagonal blocks together, the efficient computation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ requires a total of $\mathcal{O}(I^3R^2)$ flop. Thus, compared to the $\mathcal{O}(I^5R^2)$ flop for the naive computation, the exploitation of the multilinear structure offers an efficiency gain of two orders of magnitude.

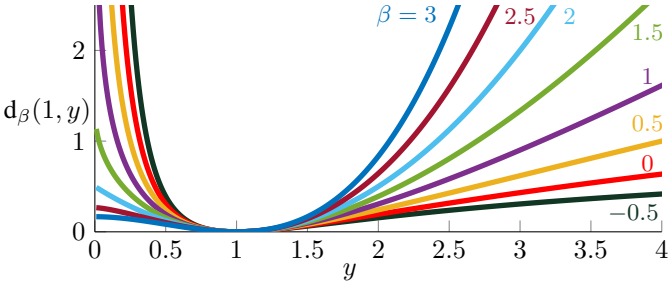

Fig. 1. Plot of $d_\beta(1, y)$ as a function of y for different values of β. These functions are only convex for $\beta \in [1, 2]$. For other values of β, they are only locally convex in a neighborhood of their minimum.

III. β-DIVERGENCES

The LS cost function does not perform optimally when the tensor is perturbed by noise that is not i.i.d. additive Gaussian. For this purpose, we consider in this section β-divergences d_β. These divergences, a special class of Bregman divergences [22], are defined for $\beta \in \mathbb{R}$ and interpolate between the Itakura–Saito (IS) divergence ($\beta = 0$), Kullback–Leibler (KL) divergence ($\beta = 1$) and the LS distance ($\beta = 2$):

$$d_\beta(x, y) = \begin{cases} \frac{x^\beta + (\beta-1)y^\beta - \beta\left(xy^{(\beta-1)}\right)}{\beta(\beta-1)} & \beta \in \mathbb{R} \setminus \{0, 1\}, \\ x \log\left(\frac{x}{y}\right) - x + y & \beta = 1, \\ \frac{x}{y} - \log\left(\frac{x}{y}\right) - 1 & \beta = 0. \end{cases}$$

Note that they are not distances, except for $\beta = 2$, as they are not symmetric and do not satisfy the triangle inequality.

Where for the LS distance, only the absolute difference between two values x and y is considered, other β-divergences also take the size of x and y themselves into account, meaning that $d_\beta(x + c, y + c) \neq d_\beta(x, y)$ for $\beta \neq 2$. More specifically, we can see from the property $d_\beta(kx, ky) = k^\beta d_\beta(x, y)$, with $k > 0$, that the parameter β controls the relative importance of the magnitude of x and y. If $\beta = 0$, the divergence is scale invariant, meaning that $d_0(x, 2x)$ is the same for any value of x. If $\beta > 0$, $d_\beta(x, 2x)$ will be larger for large values of x, while for $\beta < 0$, $d_\beta(x, 2x)$ will be larger for small values of x. See Fig. 1 for a visual representation of the β-divergence cost function for some specific values of β. Here x is the target value and y is its approximation. As can be seen from the figure, β-divergences are convex for $\beta \in [1, 2]$. Outside this interval, $d_\beta(x, y)$ is only locally convex, i.e., when $x \approx y$.

If $\beta < 2$, β-divergence cost functions penalize errors on small entries more heavily compared to the LS distance, while for values of $\beta > 2$, the converse is true. Because of this, β-divergences perform well on data with entries of different magnitudes, such as audio data. For such data, β-divergence cost functions (with $\beta < 2$) manage to capture the low intensity components of the signals better than the LS cost function. β-divergence cost functions have been used extensively in nonnegative matrix factorization (NMF) of audio spectra [23]. Where the LS cost function inherently assumes additive i.i.d.

Gaussian noise, NMFs with the KL and IS-divergence have been shown to give the maximum likelihood estimation under the assumption of Poisson-distributed entries or multiplicative Gamma noise, respectively [24]. As a generalization of NMF, β-divergence cost functions have been used to compute nonnegative CPDs of tensors [10], [12], [25]–[28]. Only the latter method, however, updates all factor matrices at the same time. Additionally, none of these methods uses Hessian information and several are only applicable for specific values of β. In contrast, all three properties are satisfied for the method from Sec. II, which we particularize for β-divergence cost functions in this section. If $R = 1$, the computations can be made particularly efficient by exploiting the multilinear structure and the specific form of the β-divergence cost function; see [29].

The β-divergence cost function for a tensor $\mathcal{T}$ and its CPD $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ is defined as $\min_{\mathbf{A},\mathbf{B},\mathbf{C}} f$ with

$$f = \frac{1}{\beta(\beta-1)} \sum_{n=1}^{N} t_n^{\beta} + (\beta-1) m_n^{\beta} - \beta \left(t_n m_n^{(\beta-1)} \right).$$

For the special cases $(\beta = 1)$ and $(\beta = 0)$, one has

$$f = \sum_{n=1}^{N} t_n \log\left(\frac{t_n}{m_n}\right) - t_n + m_n, \quad \beta = 1,$$

$$f = \sum_{n=1}^{N} \left[\frac{t_n}{m_n} - \log\left(\frac{t_n}{m_n}\right) - 1 \right], \quad \beta = 0.$$

When $\beta = 2$, the standard LS cost function is recovered. In the remainder of the paper, we constrain $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ to be nonnegative matrices as β-divergences are only defined for nonnegative values, except when $\beta \in \mathbb{N}\backslash\{1\}$. For the gradient $\mathbf{g} = [\text{vec}\left(\mathbf{R}_{(1)}(\mathbf{C} \odot \mathbf{B})\right); \text{vec}\left(\mathbf{R}_{(2)}(\mathbf{C} \odot \mathbf{A})\right); \text{vec}\left(\mathbf{R}_{(3)}(\mathbf{B} \odot \mathbf{A})\right)]$, one finds, for any value of β,

$$\mathcal{R} = \frac{\mathcal{M} - \mathcal{T}}{\mathcal{M}^{\bullet(2-\beta)}}.$$

As explained above, the Hessian can be approximated by $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ wherein $\mathbf{Z} = \text{Diag}\left(\text{vec}\left(\mathcal{D}\right)\right)$. The tensor $\mathcal{D}$ is, for any value of β, the entry-wise second-order derivative of the β-divergence cost function:

$$\mathcal{D} = \frac{(\beta-1)\mathcal{M} - (\beta-2)\mathcal{T}}{\mathcal{M}^{\bullet(3-\beta)}} = \frac{(\beta-2)\left(\mathcal{M} - \mathcal{T}\right) + \mathcal{M}}{\mathcal{M}^{\bullet(3-\beta)}}. \tag{5}$$

A. Indefiniteness of the Hessian approximation

The Gramian of the Jacobian $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ for the LS distance is always PSD. Unfortunately, this does not hold for its generalization $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$. To avoid unwelcome modifications to the algorithm, one could try to force the Hessian approximation to remain PSD during the computation of the CPD. A sufficient condition for $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ being PSD is that all entries of $\mathbf{Z}$ are nonnegative. This corresponds to the entry-wise condition

$$(\beta-1)\mathcal{M} \geq (\beta-2)\mathcal{T}. \tag{6}$$

For $\beta \in [1, 2]$, the left hand side of the equation is positive and the right hand side is negative, so (6) is always satisfied. For $\beta > 2$, the condition becomes $\mathcal{M} \geq [(\beta-2)/(\beta-1)]\mathcal{T}$, which bounds the entries of $\mathcal{M}$ from below, with the bound converging to $\mathcal{T}$ as the value of β increases. For $\beta < 1$, the condition is $\mathcal{M} \leq [(2-\beta)/(1-\beta)]\mathcal{T}$, now with the bound converging to $\mathcal{T}$ as the value of β decreases. Thus for values of β outside $[1, 2]$, the model has to be close enough to the tensor for $\mathcal{D}$ to be nonnegative, which guarantees that the Hessian approximation will be PSD. Having a good initialization is thus doubly important, as this not only decreases the number of iterations of the β-divergence algorithm, but can also avoid iterations with an indefinite Hessian approximation. We discuss four strategies to avoid indefinite Hessian approximations.

1) Modifying $\mathbf{J}^T\mathbf{Z}\mathbf{J}$: If all values of $\mathcal{D}$ are nonnegative, then $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is PSD. The easiest way to ensure this is thus by approximating $\mathcal{D}$ with a nonnegative tensor, for example by setting its negative values to zero or to a small positive value. A Levenberg–Marquardt modification of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is also possible: adding a scaled version of the identity matrix ($\lambda\mathbf{I}$) to $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ can make the Hessian approximation PSD. Of course, if λ is large, the second-order information is essentially ignored.

2) Tempering β: As noted above, the Hessian approximation is always PSD for $\beta \in [1, 2]$. One can leverage this property by first computing a solution for $\beta = 2$, using standard approaches, to get the model close to the tensor followed by a few (more expensive) iterations of the algorithm with the requested value of β. A caveat is that the LS cost function minimizes the squared error on the tensor entries, while $\mathcal{D}$ also depends on entry size (cf. (6)). This means that $\mathcal{D}$ can still be negative for the extreme (smallest or largest, depending on β) entries of the tensor when using this approach. In such cases, gradually increasing or decreasing β until the requested value is reached can be beneficial. The tempering approach is also discussed in [30].

3) Weighted least-squares: Depending on the value of β, smaller ($\beta < 2$) or larger ($\beta > 2$) tensor entries are fitted more tightly compared to the LS cost function. We can mimic such properties of cost functions by initializing with a low-rank weighted LS approach [31]. A modified problem is solved:

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

where $\mathcal{W}$ is a low-rank weight tensor of the same dimensions as $\mathcal{M}$ and $\mathcal{T}$. For β-divergences with $\beta < 2$, smaller values are given a larger weight in $\mathcal{W}$ and larger values are given a smaller weight. If $\beta > 2$, larger values are given a larger weight and smaller values are given a smaller weight. An example of a weight tensor $\mathcal{W}$ with these properties can be obtained by computing for every tensor entry t_n a second-order Taylor approximation to the cost function f around $m_n = t_n$: $f \approx f(t_n) + f'(t_n)(m_n - t_n) + \frac{f''(t_n)}{2}(m_n - t_n)^2$. Equating terms with the WLS function $w_n^2(m_n - t_n)^2$ gives:

$$w_n^2 \approx \frac{f(t_n)}{(m_n - t_n)^2} + \frac{f'(t_n)}{m_n - t_n} + \frac{f''(t_n)}{2}.$$

For the β-divergence cost function ($\beta \notin \{0, 1\}$), given as $f_\beta = \frac{1}{\beta(\beta-1)} \left(t_n^{\beta} + (\beta-1) m_n^{\beta} - \beta t_n m_n^{(\beta-1)} \right)$, one finds $f_\beta(t_n) = f'_\beta(t_n) = 0$ and $f''_\beta(t_n) = t_n^{(\beta-2)}$, leading to a weighting factor $w_n = \frac{1}{\sqrt{2}} t_n^{\frac{(\beta-2)}{2}}$. For values of $\beta > 2$, larger entries are indeed given larger weights, while the converse is

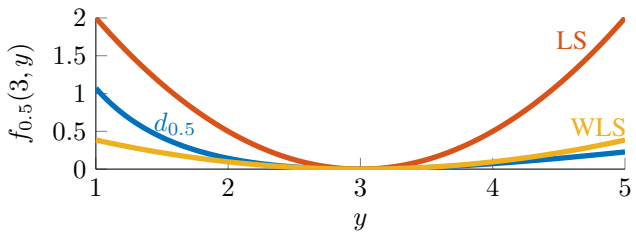

Fig. 2. The WLS cost function gives a better local approximation to $d_{0.5}(3,y)$ than the LS cost function.

true for $\beta < 2$. For $\beta = 2$, the LS cost function is recovered. In Fig. 2, it can be seen that a WLS cost function with the proposed weighting factor gives a better local quadratic approximation to the β-divergence cost function than the LS cost function. A good choice of weight tensor is, for instance, the best rank-1 approximation to $\mathcal{W} = \frac{1}{\sqrt{2}}\mathcal{T}^{\bullet\frac{(\beta-2)}{2}}$, as using a low-rank weight tensor makes the WLS CPD much less costly. As an alternative that is cheaper to compute, one can also take $\mathcal{W} = [\![\mathbf{a},\mathbf{b},\mathbf{c}]\!]$, with $\mathbf{a} = \frac{1}{\sqrt[3]{2}}\left(\frac{1}{JK}\sum_{q=1}^{JK}\mathbf{T}_{(1)_{:q}}^{\bullet\frac{(\beta-2)}{2}}\right)$, and with $\mathbf{b}$ and $\mathbf{c}$ defined analogously.

4) Shifting: A fourth way to force $\mathcal{D}$ to be nonnegative, and thus to guarantee that the Hessian approximation is PSD, is to add a constant nonnegative tensor $\mathcal{P}$ to both $\mathcal{T}$ and $\mathcal{M}$. Instead of fitting the model $\mathcal{M}$ to the tensor $\mathcal{T}$, we thus fit the model $\mathcal{M}+\mathcal{P}$ to the tensor $\mathcal{T}+\mathcal{P}$. In this way, we are essentially shifting our problem to a region where the cost function is nicer: the shifted problem ideally lies in the locally convex region of the β-divergence cost function (cf. Fig. 1). Of course, shifting introduces a bias, as the β-divergence is not shift-invariant for $\beta \neq 2$. We thus want to gradually decrease the shift back to zero during the optimization process. Revisiting (6), requiring $\mathcal{D}$ to be nonnegative introduces the following requirement for the shifting tensor $\mathcal{P}$:

$$\mathcal{P} \geq (\beta-2)\mathcal{T} - (\beta-1)\mathcal{M}.$$

In principle, every entry t_n could have its own shift p_n, but taking one common value for certain parts of the tensor or a common p for all tensor entries can reduce storage requirements. A possible shift that satisfies the bound for all entries is $p = k\max(\mathcal{P})$, with $k \geq 1$ and $\mathcal{P} = (\beta-2)\mathcal{T} - (\beta-1)\mathcal{M}$. If $p < 0$, the Hessian is already PSD, so we can set $p = 0$. In the latter case, setting $p = \delta > 0$ when p is slightly nonnegative may improve the convergence of the algorithm by increasing the size of the locally convex domain around the minimum.

The size of the shift, which depends on the residual $\mathcal{T}-\mathcal{M}$, should decrease as the model gets closer to the tensor and should eventually become zero. The objective function value, gradient and $\mathcal{D}$ are computed with the shifted tensor and model instead of the original tensor and model, but the algorithm itself remains exactly the same. The tensor $\mathcal{P}$ is computed as an intermediate result in the computation of $\mathcal{D}$ (see (5)), so the only additional computations that the shifting introduces are the sums $\mathcal{T}+\mathcal{P}$ and $\mathcal{M}+\mathcal{P}$. As $\mathcal{P}$ is a constant, the shift does not increase the dimensions of the gradient and Hessian. A nice byproduct of shifting is that it also avoids possible numerical problems due to tensor entries that are zero. Shifting works better when β is close to the interval $[1,2]$. When $\beta \ll 1$ or $\beta \gg 2$, the bound in (6) becomes tighter and thus a larger shift is needed. If the required shift is too large, it could dominate the original tensor entries. If this happens, it might not be possible to decrease the shift all the way to zero, making the algorithm converge to a suboptimal solution.

IV. Experiments

A number of experiments are performed on synthetic and real-life data. First, we motivate the use of β-divergence cost functions by comparing the CPDs for different values of β. We then analyze the performance of different strategies to handle the indefiniteness of the Hessian approximation. We show how the proposed GGN algorithm performs compared to standard methods from the literature and compare the computational cost to that of the GN method. Finally, we use our method to approximate a hyperspectral image. The experiments are performed on a computer with an Intel Core i7-6820HQ CPU at 2.70GHz and 16GB of RAM using MATLAB R2016b and Tensorlab 3.0 [32]. For experiments on synthetic data, entries of factor matrices are sampled independently from the uniform distribution on the interval (a,b), denoted by $U(a,b)$. Writing $\bar{\mathcal{T}}$ for the perturbed tensor $\mathcal{T}$, the signal to noise ratio (SNR) is defined for all noise types as $20\log_{10}\left(\frac{\|\mathcal{T}\|_F}{\|\mathcal{N}\|_F}\right)$, where $\mathcal{N} = \bar{\mathcal{T}} - \mathcal{T}$. When the entries of a tensor $\mathcal{T}$ are perturbed by multiplicative Gamma noise, we have $\bar{\mathcal{T}} = \mathcal{T} * \mathcal{N}_\gamma$. The entries of the noise tensor $\mathcal{N}_\gamma$ are sampled from a Gamma distribution with shape α and scale $\frac{1}{\alpha}$, hence, the SNR (in dB) is expected to be $10\log_{10}(\alpha)$.

A. Optimal value of β

Three random factor matrices $\mathbf{A},\mathbf{B},\mathbf{C}$ in $\mathbb{R}_+^{20\times 3}$ are generated with entries sampled from $U(1,10)$. Then, a rank-3 tensor $\mathcal{T}$ is constructed with these factor matrices, either additive Gaussian noise or multiplicative Gamma noise is added and factor matrices $\tilde{\mathbf{A}},\tilde{\mathbf{B}},\tilde{\mathbf{C}}$ are computed with a β-divergence cost function for different values of β. In Fig. 3, the mean absolute factor matrix error $\max(\|\mathbf{A}-\tilde{\mathbf{A}}\|_F, \|\mathbf{B}-\tilde{\mathbf{B}}\|_F, \|\mathbf{C}-\tilde{\mathbf{C}}\|_F)$ is shown for an SNR of 40 dB after matching columns and normalizing all factor matrices such that $\|\mathbf{a}_{:r}\| = \|\mathbf{b}_{:r}\| = \|\mathbf{c}_{:r}\|$ and $\|\tilde{\mathbf{a}}_{:r}\| = \|\tilde{\mathbf{b}}_{:r}\| = \|\tilde{\mathbf{c}}_{:r}\|$ for $1 \leq r \leq R$. For tensors that are perturbed by additive Gaussian noise, the smallest errors are obtained for $\beta = 2$, while for multiplicative Gamma noise, the lowest errors are obtained for $\beta = 0$. This is in line with the discussion at the beginning of Sec. III and motivate the use of different cost functions depending on the distribution of the noise.

β-divergence cost functions with small values of β fit small tensor entries tighter than the LS cost function. To illustrate this, a tensor $\mathcal{T} = [\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$ is generated, where $\mathbf{A},\mathbf{B},\mathbf{C} \in \mathbb{R}_+^{10\times 3}$ with entries sampled as 10^x with x sampled from $U(-2,2)$. This tensor is then perturbed with a small amount of noise to show the influence of the parameter β. For Fig. 4, the tensor is perturbed with additive Gaussian noise with an SNR of 100 dB and its CPD $\hat{\mathcal{T}}$ is computed for β-divergence cost functions with $\beta = 0$ and $\beta = 4$. A high SNR value is necessary, as the SNR value is dominated by the large tensor entries. With the large dynamic range of the entries

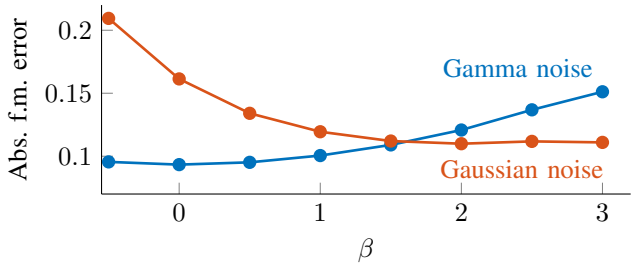

Fig. 3. The errors for tensors perturbed by Gamma noise are about 29.4% larger when using $\beta = 2$ compared to $\beta = 0$, while for Gaussian noise, using $\beta = 2$ is optimal. The mean (50 trials) absolute factor matrix error is shown for different values of β for tensors perturbed by multiplicative Gamma noise and tensors perturbed by additive Gaussian noise, both with an SNR of 40 dB.

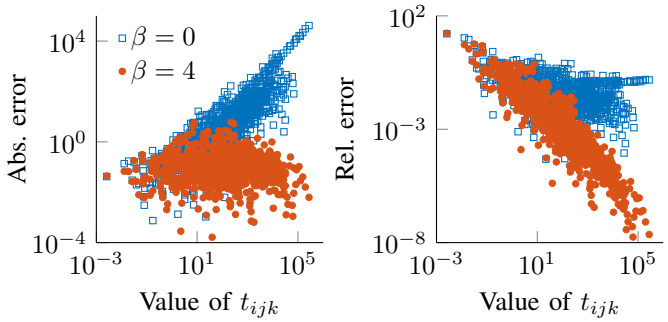

Fig. 4. The cost function with $\beta = 4$ performs better for *additive Gaussian noise* than the cost function with $\beta = 0$. Absolute (left) and relative (right) error on the tensor entries t_{ijk} when fitting a CPD of a low-rank tensor that is perturbed with *additive Gaussian noise*. Two different β-divergence cost functions are used: $\beta = 0$ and $\beta = 4$. For the former, the relative error is approximately the same for small and large tensor entries. For the latter, larger entries tend to have a smaller relative error than smaller entries.

in this experiment, the smallest entries would be completely dominated by the additive Gaussian noise for low SNRs. In Fig. 5, the same experiment is performed for multiplicative Gamma noise. In both figures, it can be seen that for the cost function with large β, the absolute error $|t_{ijk} - \hat{t}_{ijk}|$ of all tensor entries is approximately the same, independently of their size. As a result, the relative error $|t_{ijk} - \hat{t}_{ijk}|/t_{ijk}$ is smaller for larger tensor entries. For the cost function with small β, the relative error remains about the same along the range of tensor entries. Of course, this means that the absolute error is larger for larger tensor entries.

B. Handling indefiniteness

In the following experiment, four different strategies for handling indefiniteness of the Hessian approximation are compared: LS and WLS initialization (with a relative step size of 10^{-2} and a relative cost function change of 10^{-4} as stopping criteria), random initialization using shifting with one common shift for the whole tensor ($p = 2\max(\mathcal{P})$), and random initialization without shifting. These strategies are discussed in Subsections III-A3 and III-A4. Three random factor matrices are generated in $\mathbb{R}_+^{50\times 5}$ with entries sampled from $U(0,1)$ and then the associated rank-5 tensor is constructed. We perturb the tensor with multiplicative Gamma noise and an SNR of 20 dB. A relative step size of 10^{-4} and a relative cost function

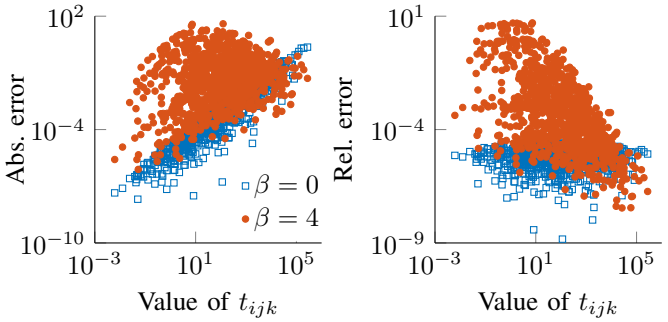

Fig. 5. For *multiplicative Gamma noise*, the cost function with $\beta = 0$ performs better. Same setting as in Fig. 4, but with *multiplicative Gamma noise* instead. For $\beta = 0$, the relative error is approximately the same for small and large tensor entries. For $\beta = 4$, larger entries tend to have a smaller relative error than smaller entries.

TABLE I

THE LS AND WLS INITIALIZATIONS ENABLE FASTER CPD COMPUTATIONS, WITH WLS PERFORMING BETTER FOR SMALL β. FOR RANDOM INITIALIZATIONS, SHIFTING DOES NOT INFLUENCE THE COMPUTATION TIME MUCH, WHILE IT DOES ASSURE CONVERGENCE. THE MEDIAN TIME (S) (OVER 20 TRIALS) IS SHOWN TO COMPUTE A CPD OF A RANK-5 TENSOR $\mathcal{T} \in \mathbb{R}_+^{50\times 50\times 50}$ PERTURBED BY MULT. GAMMA NOISE WITH AN SNR OF 20 dB WHEN USING FOUR STRATEGIES FOR HANDLING INDEFINITE HESSIANS. FOR LS AND WLS INITIALIZATION, THE INITIALIZATION TIME IS ALSO SHOWN. ONLY SUCCESSFUL TRIALS WERE USED FOR RANDOM INITIALIZATION WITHOUT SHIFTING (12 OUT OF 20).

	Time (s) for different β					
	β					
Method	-0.5	0	0.5	1	1.5	2.5
Init. with LS	0.273	0.288	0.268	0.283	0.273	0.281
Main GGN	2.704	1.590	1.296	1.043	1.072	1.081
Init. with WLS	0.229	0.234	0.271	0.284	0.315	0.388
Main GGN	1.669	1.070	1.088	0.895	0.906	1.285
Random init. + shifting	3.391	2.143	1.885	1.821	1.605	1.725
Random init.	2.984	2.276	2.058	1.735	1.545	2.020

change of 10^{-8} are used as stopping criteria. In Tables I and II, the median runtime and median number of iterations are shown for 20 tensors and for different values of β. For random initialization and without shifting, the method failed to converge to the correct solution in 8 of the 20 trials. The reported numbers do not take these failed trials into account. From the tables, it can be seen that convergence is much faster, both in time and in number of iterations after computing a LS or WLS initialization. The WLS initialization, with as weighting tensor $\mathcal{W}$ a rank-1 approximation to $\frac{1}{\sqrt{2}}\mathcal{T}^{\bullet\left(\frac{\beta-2}{2}\right)}$, performs slightly better than the LS initialization. When starting from a random initialization, shifting generally does not increase the computation time or number of iterations, while it does fix the convergence problems for random initialization.

C. Other β-divergence CPD algorithms

The proposed GGN method is compared to the generalized CPD (GCPD) method [10] for a β-divergence cost function and to the classical multiplicative update (MU) algorithm for β-divergences [12]. Random factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C} \in \mathbb{R}^{I\times R}$ are generated with entries sampled from $U(0,1)$,

TABLE II
WHEN STARTING FROM AN LS OR WLS INITIALIZATION, THE NUMBER OF REMAINING (COSTLY) GGN ITERATIONS UNTIL CONVERGENCE IS SIGNIFICANTLY LOWER THAN FOR RANDOM INITIALIZATION. SHIFTING DOES NOT CHANGE THE NUMBER OF ITERATIONS WHEN STARTING FROM A RANDOM INITIALIZATION, WHILE IT DOES ASSURE THAT THE METHOD CONVERGES. THE MEDIAN NUMBER OF ITERATIONS (OVER 20 TRIALS) IS DISPLAYED FOR THE SAME EXPERIMENT AS IN TABLE I.

	Number of iterations for different β					
	β					
Method	-0.5	0	0.5	1	1.5	2.5
Init. with LS	8.0	8.0	8.0	8.0	8.0	8.0
Main GGN	16.5	11.5	9.0	8.0	7.0	7.5
Init. with WLS	7.5	7.5	8.0	8.0	8.5	7.5
Main GGN	11.0	7.5	7.5	7.0	6.0	8.0
Random init. + shifting	22.0	15.5	13.0	14.0	11.0	12.0
Random init.	19.5	16.0	14.0	13.5	11.0	13.5

and $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$. In practice, one will often use a cheaper method to get close to the solution, after which the more expensive GGN method can leverage its fast local convergence to get to the true solution. Thus, to focus on the convergence behavior close to the solution, the methods are initialized by generating random factor matrices with entries sampled from $U(0, 1)$ and then applying five iterations of the GN algorithm for $\beta = 2$. The CPD is then computed for different values of β. In Fig. 6, the results of the different methods are shown for $I = 20$ and $R = 5$. The median computation time, relative factor matrix error and number of iterations are displayed. All methods use a relative step size of 10^{-10} as a stopping criterion. The GGN and GCPD methods both find the correct solution in all cases. The GGN method requires fewer iterations than the GCPD method, while the MU method does not converge within 1000 iterations. The median computation time of GGN is lower than GCPD, especially when $\beta \notin [1, 2]$.

Fig. 7 shows the typical evolution of the cost function value f_β close to the solution, here for $\beta = 0.5$. On the left, it can be seen that the proposed GGN method converges to the true solution after very few iterations. In contrast, the MU-algorithm requires thousands of iterations with slow improvement before converging to the true solution. GCPD requires hundreds of iterations to reach the true solution. While it converges with about the same rate as the GGN method far from the solution, its convergence is clearly slower close to the solution, where the GGN method can fully leverage its second-order information and achieves quadratic convergence.

D. GGN compared to GN

Using a non-LS cost function naturally adds complexity to the algorithm, as the Hessian approximation $\mathbf{J}^\mathrm{T}\mathbf{J}$ is replaced by the matrix $\mathbf{J}^\mathrm{T}\mathbf{Z}\mathbf{J}$, where $\mathbf{Z}$ is a diagonal matrix with $\mathrm{vec}(\mathcal{D})$ on its diagonal. As discussed in Sec. II-C, the structure of the factors of $\mathbf{J}^\mathrm{T}\mathbf{Z}\mathbf{J}$ can be exploited in the computation of this matrix, which drastically lowers its computation time. In Fig. 8, the computation time of both Hessian approximations is shown for an $(I \times I \times I)$-tensor and a rank-10 CPD. The computation time for the Hessian approximation of the general

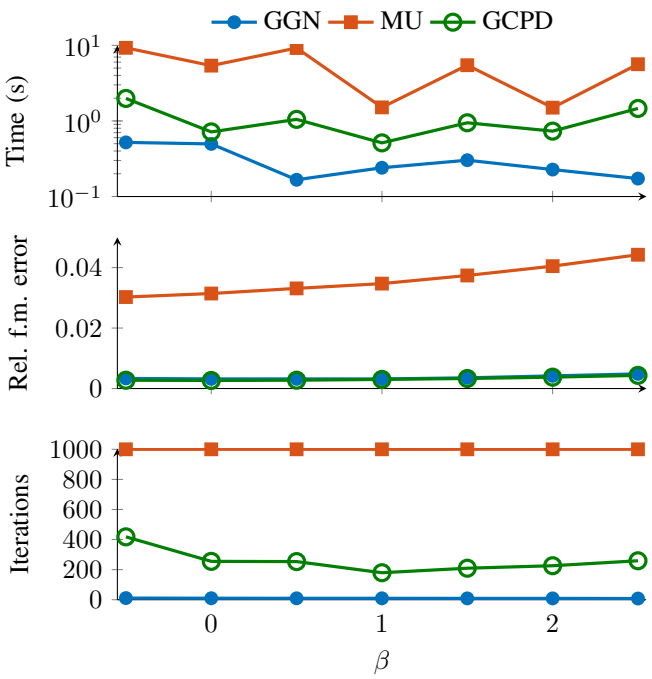

Fig. 6. The proposed GGN method requires considerably fewer iterations than the other two methods. The performance with respect to time, relative factor matrix error and number of iterations is shown for three methods to compute a rank-5 CPD of a rank-5 $(20 \times 20 \times 20)$-tensor perturbed by multiplicative Gamma noise with an SNR of 40 dB for different values of β. All methods are capped at 1000 iterations and medians over 20 trials are shown.

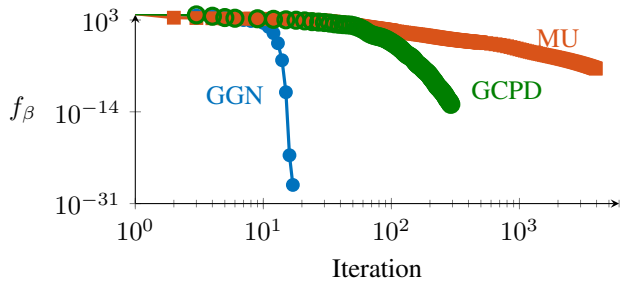

Fig. 7. The GGN method converges quadratically (until machine precision is reached) close to the solution, while the other methods converge linearly. The typical convergence behavior of a β-divergence CPD with $\beta = 0.5$ is shown for the GGN, MU and GCPD methods on a rank-5 $(40 \times 40 \times 40)$-tensor. The cost function value f_β is plotted in function of the iteration number.

cost function is larger, but only by a factor 2. Considering that the full tensor $\mathcal{D}$ has to be computed and included in the approximation, this is an acceptable price to pay. Actually, the increase in computation time is much more significant for the cost function and gradient evaluations when replacing the LS cost function by a β-divergence cost function, as can also be seen from Fig. 8. For the LS cost function, the computation time for these evaluations is almost negligible compared to the computation time for the Hessian approximation, giving first-order methods an edge compared to second-order methods. This is not the case for the β-divergence cost function, particularly when the tensor is large. As a result, even first-order β-divergence CPD methods that only evaluate the cost function and gradient will be significantly more expensive than LS CPD methods. This makes the proposed second-order method appealing for general cost functions, as computing the

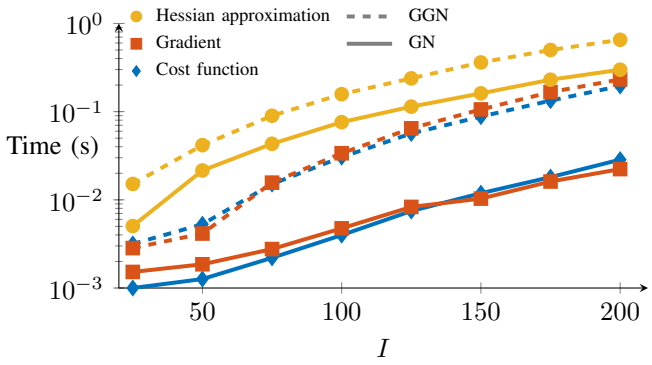

Fig. 8. The GGN method (dashed) requires about twice as much time to compute the Hessian approximation as the GN method (solid). Second, compared to LS, the β-divergence cost function and gradient evaluations are much more costly. The mean time (over 20 trials) is shown for the computation of one cost function evaluation, one gradient evaluation, and one Hessian approximation in function of the tensor size for the LS (solid) and β-divergence with $\beta = 1.5$ (dashed) cost functions. The GN approximation is used for the Hessian of the LS cost function, while the GGN approximation is used for the β-divergence cost function. The tensors have dimensions $(I \times I \times I)$ and rank 10 and are approximated by a CPD with rank 10.

Hessian approximation is a relatively smaller extra cost.

Both GN and GGN require the solution of the system $\mathbf{Hp} = -\mathbf{g}$ to obtain the step direction. The cost of solving this system is relatively lower for GGN, as the construction of its Hessian approximation is more costly. Still, for larger tensor dimensions, it can be beneficial to solve $\mathbf{Hp} = -\mathbf{g}$ approximately using inexact methods [33].

E. Hyperspectral image compression

In this example, a part of the DC Mall hyperspectral image [34] of dimensions 80×80 is considered, which contains 160 different wavelengths. The $(I\times J\times K)$-tensor $\mathcal{T}$ representing the hyperspectral image is thus of dimensions $80\times80\times160$. See Fig. 9a for an image at one wavelength. Hyperspectral images can easily become very large, thus to limit the storage requirements, a compressed image can be stored instead. Here, we use the CPD to approximate the hyperspectral image, which can also have a denoising effect on the image [35]. As Poisson noise can often be expected to be introduced during the image generation [36], this type of noise was included (see Fig. 9b) to obtain the tensor $\tilde{\mathcal{T}}$. The noise was introduced using the MATLAB command $\tilde{\mathcal{T}} = \texttt{poissrnd}(\mathcal{T}p)/p$, with p the requested noise level (higher p means less noise). This tensor was then approximated by a rank-R CPD, both with an LS and with a KL-divergence cost function, to obtain a compressed tensor $\hat{\mathcal{T}}$. The LS solution was used as an initialization for the KL-divergence CPD. In Figures 9c and 9d, the results can be seen. Although the difference is not large, the compressed image obtained with the KL-divergence cost function manages to capture the details of the image slightly better. This can also be seen when comparing quality measures of both compressed images for different noise levels and approximation ranks. The peak SNR (PSNR) and the universal image quality index (UIQI) [37] are computed for both CPDs. The former is defined as

$$\text{PSNR}(\mathcal{T}, \hat{\mathcal{T}}) = 10\log_{10}\left(\frac{\rho^2 IJK}{\|\mathcal{T}-\hat{\mathcal{T}}\|_{\text{F}}^2}\right),$$

Fig. 9a. Original image

Fig. 9b. With Poisson noise

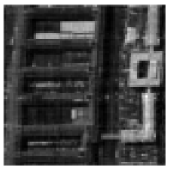

Fig. 9c. LS cost function

Fig. 9d. KL cost function

Fig. 9. Spectral band 120 of a (80×80) aerial image perturbed by Poisson noise with an SNR of 10 dB. The reconstructed images are shown when the tensor is approximated with a rank-40 CPD computed with a LS cost function and with a KL divergence cost function.

with ρ the maximal value of the image, which is simply equal to one in this case. The latter is defined as

$$\text{UIQI}(\mathcal{T}, \hat{\mathcal{T}}) = \frac{1}{K}\sum_{k=1}^{K}\frac{4\sigma^2_{\mathbf{t}_k\hat{\mathbf{t}}_k}\mu_{\mathbf{t}_k}\mu_{\hat{\mathbf{t}}_k}}{\left(\sigma^2_{\mathbf{t}_k}+\sigma^2_{\hat{\mathbf{t}}_k}\right)\left(\mu^2_{\mathbf{t}_k}+\mu^2_{\hat{\mathbf{t}}_k}\right)},$$

with $\mathbf{t}_k = \text{vec}\left(\mathbf{T}_{::k}\right)$ and $\hat{\mathbf{t}}_k = \text{vec}\left(\hat{\mathbf{T}}_{::k}\right)$. The PSNR is based on the LS fit of the CPD to the original tensor and is thus expected to be higher for the solution obtained with the LS cost function. The UIQI was designed specifically to judge the quality of compressed images and reflects the luminance and contrast distortion caused by the compression. The closer the UIQI is to 1, the better the approximation is.

The results are shown in Table III. First, it can be seen that for CPDs with relatively low rank and less noise, the LS solution indeed gives the solution with the highest PSNR. If the rank of the CPD is high, however, or the amount of noise is increased, the KL-divergence solution actually has the largest PSNR. In these cases, the change of cost function allows the optimization algorithm to escape from the suboptimal solution that was found with the LS cost function and find a better local minimum. Second, one can see that the KL-divergence cost function consistently scores slightly better than the LS cost function with respect to the UIQI. Because the tensor has been perturbed with Poisson noise, the KL-divergence cost function manages to preserve the luminance and contrast of the original image better than the LS cost function, even when the latter offers a solution with a higher PSNR.

V. Conclusion

A second-order all-at-once CPD algorithm was proposed that can handle general entry-wise cost functions. By exploiting the low-rank structure of the problem, the computational cost of the algorithm was significantly limited. The method was applied to β-divergence cost functions, which are tailored to data with non-Gaussian noise. The method was shown to outperform the widely used first-order approaches and to show fast convergence close to the solution in low-noise cases.

TABLE III
COMPARISON OF THE PSNR (LEFT) AND THE UIQI (RIGHT) FOR A HYPERSPECTRAL IMAGE APPROXIMATED BY A CPD WITH EITHER A LS OR KL-DIVERGENCE COST FUNCTION. DIFFERENT SNRS AND APPROXIMATION RANKS R ARE CONSIDERED. BEST RESULTS OF FIVE INITIALIZATIONS ARE SHOWN. THE KL-DIVERGENCE COST FUNCTION CONSISTENTLY GIVES THE SOLUTION WITH THE HIGHEST UIQI, EVEN IF THE PSNR OF THE LS SOLUTION IS HIGHER.

		PSNR		UIQI	
SNR	R	LS	KL	LS	KL
10 dB	10	25.2953	**25.3192**	0.8777	**0.8847**
	20	25.8925	**27.7919**	0.9244	**0.9402**
	40	27.1162	**30.5213**	0.9504	**0.9671**
20 dB	10	**25.4364**	25.2722	0.8833	**0.8844**
	20	**27.7218**	27.6653	0.9337	**0.9347**
	40	27.4485	**30.6091**	0.9550	**0.9672**

APPENDIX A
PROOF OF SEC. II-C1

Theorem 1: Let $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ be a CPD of $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$ and $\mathbf{Z} = \text{Diag}(\text{vec}(\mathcal{D}))$. Then the nonzero entries $\mathbf{G}_{\text{nz}}^{(1,1)}$ of the matrix $\mathbf{G}^{(1,1)} = ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\text{T}} \mathbf{Z} ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I) \in \mathbb{R}^{I \times I}$ can be computed as follows:

$$\mathbf{G}_{\text{nz}}^{(1,1)} = \mathbf{D}_{(1)} \left((\mathbf{C} \odot^{\text{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\text{T}} \mathbf{B}) \right).$$

Proof: Consider the case $R = 1$. Then $\mathbf{c} \odot \mathbf{b}$ is a vector holding all products of an entry of $\mathbf{c}$ and an entry of $\mathbf{b}$. The Kronecker product with the identity then replaces every element $c_x b_y$ of this vector with $c_x b_y$ times the $(I \times I)$-identity matrix. Thus $[(\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I]$ looks like this:

$$\begin{bmatrix} c_1 b_1 \mathbf{I}_I & \cdots & c_1 b_J \mathbf{I}_I & c_2 b_1 \mathbf{I}_I & \cdots & c_K b_J \mathbf{I}_I \end{bmatrix}^{\text{T}}.$$

The product $((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I)^{\text{T}} \mathbf{Z} ((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I) \in \mathbb{R}^{I \times I}$ can then be computed block-wise:

$$\begin{aligned} &((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I)^{\text{T}} \text{Diag}\left(\begin{bmatrix} \mathbf{d}_{:11}^{\text{T}}, \cdots, \mathbf{d}_{:JK}^{\text{T}} \end{bmatrix}^{\text{T}} \right) ((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I) \\ &= c_1^2 b_1^2 \text{Diag}\left(\mathbf{d}_{:11}\right) + \cdots + c_1^2 b_J^2 \text{Diag}\left(\mathbf{d}_{:J1}\right) \\ &\quad + c_2^2 b_1^2 \text{Diag}\left(\mathbf{d}_{:12}\right) + \cdots + c_K^2 b_J^2 \text{Diag}\left(\mathbf{d}_{:JK}\right). \end{aligned}$$

Note that this is a diagonal matrix. We can compute only the nonzero values by removing the diagonalization, which gives us the vector $\mathbf{G}_{\text{nz}}^{(1,1)} \in \mathbb{R}^{I \times 1}$:

$$\begin{aligned} \mathbf{G}_{\text{nz}}^{(1,1)} &= c_1^2 b_1^2 \mathbf{d}_{:11} + \cdots + c_1^2 b_J^2 \mathbf{d}_{:J1} \\ &\quad + c_2^2 b_1^2 \mathbf{d}_{:12} + \cdots + c_K^2 b_J^2 \mathbf{d}_{:JK} \\ &= \mathbf{D}_{(1)} \left(\mathbf{c}^{\bullet 2} \odot \mathbf{b}^{\bullet 2} \right). \end{aligned} \tag{7}$$

For the general case where $R > 1$, the matrix $((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\text{T}}$ is of the form

$$\begin{bmatrix} c_{11} b_{11} \mathbf{I}_I & \cdots & c_{11} b_{J1} \mathbf{I}_I & c_{21} b_{11} \mathbf{I}_I & \cdots & c_{K1} b_{J1} \mathbf{I}_I \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_{1R} b_{1R} \mathbf{I}_I & \cdots & c_{1R} b_{JR} \mathbf{I}_I & c_{2R} b_{1R} \mathbf{I}_I & \cdots & c_{KR} b_{JR} \mathbf{I}_I \end{bmatrix}.$$

After computing $((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\text{T}} \mathbf{Z} ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)$, one then obtains R^2 sums of the form:

$$\begin{aligned} &c_{1x} c_{1y} b_{1x} b_{1y} \mathbf{d}_{:11} + \cdots + c_{1x} c_{1y} b_{Jx} b_{Jy} \mathbf{d}_{:J1} \\ &\quad + c_{2x} c_{2y} b_{1x} b_{1y} \mathbf{d}_{:12} + \cdots + c_{Kx} c_{Ky} b_{Jx} b_{Jy} \mathbf{d}_{:JK} \end{aligned}$$

for $x, y \in \{1, \ldots, R\}$. This means that we now need all entry-wise products of two columns of $\mathbf{C}$ and of two columns of $\mathbf{B}$, as a generalization of the factors $\mathbf{c}^{\bullet 2}$ and $\mathbf{b}^{\bullet 2}$ in (7). These are obtained by computing $(\mathbf{C} \odot^{\text{T}} \mathbf{C})$ and $(\mathbf{B} \odot^{\text{T}} \mathbf{B})$, respectively. This leads us to the expression

$$\mathbf{G}_{\text{nz}}^{(1,1)} = \mathbf{D}_{(1)} \left((\mathbf{C} \odot^{\text{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\text{T}} \mathbf{B}) \right),$$

which holds all nonzero values of $((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\text{T}} \mathbf{Z} ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)$. ■

APPENDIX B
PROOF OF SEC. II-C2

Theorem 2: Let $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ be a CPD of $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$. Also, let $\mathbf{Z} = \text{Diag}(\text{vec}(\mathcal{D}))$ and $\mathbf{W}^{(1,2)}$ be the matrix $\mathbf{D}_{(3)}^{\text{T}}(\mathbf{C} \odot^{\text{T}} \mathbf{C}) \in \mathbb{R}^{IJ \times R^2}$ reshaped into an $(IR \times JR)$-matrix. Then the matrix $\mathbf{G}^{(1,2)} = ((\mathbf{C} \odot \mathbf{B}) \otimes \mathbf{I}_I)^{\text{T}} \mathbf{Z} ((\mathbf{C} \odot \mathbf{A}) \otimes \mathbf{I}_J) \in \mathbb{R}^{IR \times JR}$ can be computed as follows:

$$\mathbf{G}^{(1,2)} = \begin{bmatrix} \mathbf{a}_{:1} \mathbf{b}_{:1}^{\text{T}} & \cdots & \mathbf{a}_{:R} \mathbf{b}_{:1}^{\text{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{:1} \mathbf{b}_{:R}^{\text{T}} & \cdots & \mathbf{a}_{:R} \mathbf{b}_{:R}^{\text{T}} \end{bmatrix} * \mathbf{W}^{(1,2)}.$$

Proof: Consider the case $R = 1$, where the goal is to compute $\mathbf{G}^{(1,2)} = ((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I)^{\text{T}} \mathbf{Z} \mathbf{\Pi}^{(2)^{\text{T}}} ((\mathbf{c} \odot \mathbf{a}) \otimes \mathbf{I}_J)$. This product can again be computed block by block, where the kth block contains all rows of the Jacobians $\mathbf{J_A}$ and $\mathbf{J_B}$ that contain the factor c_k and the kth column of $\mathbf{D}_{(3)}^{\text{T}}$:

$$((\mathbf{c} \odot \mathbf{b}) \otimes \mathbf{I}_I)^{\text{T}} \text{Diag}\Big(\begin{bmatrix} \mathbf{d}_{(3)_{:1}}^{\text{T}}, \cdots, \mathbf{d}_{(3)_{:K}}^{\text{T}} \end{bmatrix}^{\text{T}} \Big) \mathbf{\Pi}^{(2)^{\text{T}}} ((\mathbf{c} \odot \mathbf{a}) \otimes \mathbf{I}_J)$$

$$\begin{aligned} &= c_1^2 \left(\mathbf{b}^{\text{T}} \otimes \mathbf{I}_I \right) \text{Diag}\left(\mathbf{d}_{(3)_{:1}}^{\text{T}} \right) (\mathbf{I}_J \otimes \mathbf{a}) \\ &\quad + \cdots + c_K^2 \left(\mathbf{b}^{\text{T}} \otimes \mathbf{I}_I \right) \text{Diag}\left(\mathbf{d}_{(3)_{:K}}^{\text{T}} \right) (\mathbf{I}_J \otimes \mathbf{a}). \end{aligned}$$

The structure of these terms can then be exploited to obtain:

$$\begin{aligned} \mathbf{G}^{(1,2)} &= c_1^2 \text{ unvec}\left(\mathbf{d}_{(3)_{:1}}^{\text{T}} \right) * \left(\mathbf{a} \mathbf{b}^{\text{T}} \right) \\ &\quad + \cdots + c_K^2 \text{ unvec}\left(\mathbf{d}_{(3)_{:K}}^{\text{T}} \right) * \left(\mathbf{a} \mathbf{b}^{\text{T}} \right). \end{aligned}$$

Computing $\mathbf{D}_{(3)}^{\text{T}} \left(\mathbf{c}^{\bullet 2} \right) \in \mathbb{R}^{IJ \times 1}$ and reshaping this matrix into the $(I \times J)$-matrix $\mathbf{W}^{(1,2)}$ then gives:

$$\mathbf{G}^{(1,2)} = \left(\mathbf{a} \mathbf{b}^{\text{T}} \right) * \mathbf{W}^{(1,2)}.$$

For the general case where $R > 1$, again, all combinations of columns can be handled separately. One now needs all entry-wise products of two columns of $\mathbf{C}$ and each of these terms should be matched with its corresponding outer product of a column of $\mathbf{A}$ and a column of $\mathbf{B}$, thus $\mathbf{c}_{:x} \mathbf{c}_{:y}$ should be matched with $\mathbf{a}_{:x} \mathbf{b}_{:y}^{\text{T}}$. The matrix $\mathbf{W}^{(1,2)} \in \mathbb{R}^{IR \times JR}$ is then obtained by reshaping $\mathbf{D}_{(3)}^{\text{T}} \left(\mathbf{C} \odot^{\text{T}} \mathbf{C} \right) \in \mathbb{R}^{IJ \times R^2}$ into an $(I \times J \times R \times R)$-tensor, permuting its modes to get an $(I \times R \times J \times R)$-tensor and finally reshaping this tensor into $\mathbf{W}^{(1,2)}$. The full $(1,2)$ off-diagonal block then becomes:

$$\mathbf{G}^{(1,2)} = \begin{bmatrix} \mathbf{a}_{:1} \mathbf{b}_{:1}^{\text{T}} & \cdots & \mathbf{a}_{:R} \mathbf{b}_{:1}^{\text{T}} \\ \vdots & \ddots & \vdots \\ \mathbf{a}_{:1} \mathbf{b}_{:R}^{\text{T}} & \cdots & \mathbf{a}_{:R} \mathbf{b}_{:R}^{\text{T}} \end{bmatrix} * \mathbf{W}^{(1,2)}.$$

■

REFERENCES

[1] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Trans. Signal Process*., vol. 65, no. 13, pp. 3551–3582, 2017.

[2] A. Cichocki, C. Mandic, A.-H. Phan, C. Caiafa, G. Zhou, Q. Zhao, and L. De Lathauwer, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag*., vol. 32, pp. 145–163, 2015.

[3] A.-H. Phan and A. Cichocki, "PARAFAC algorithms for large-scale problems," *Neurocomputing*, vol. 74, no. 11, pp. 1970–1984, 2011.

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

[5] I. Domanov and L. De Lathauwer, "Canonical polyadic decomposition of third-order tensors: Reduction to generalized eigenvalue decomposition," *SIAM J. Matrix Analysis Appl*., vol. 35, no. 2, pp. 636–660, 2014.

[6] E. Acar, D. M. Dunlavy, and T. G. Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *J. Chemom*., vol. 25, no. 2, pp. 67–86, 2011.

[7] J. B. Kruskal, "Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics," *Linear Algebra Appl*., vol. 18, no. 2, pp. 95–138, 1977.

[8] I. Domanov and 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 Analysis Appl*., vol. 34, no. 3, pp. 876–903, 2013.

[9] I. J. Myung, "Tutorial on maximum likelihood estimation," *J. Math. Psychol*., vol. 47, no. 1, pp. 90 – 100, 2003.

[10] D. Hong, T. G. Kolda, and J. A. Duersch, "Generalized canonical polyadic tensor decomposition," *SIAM Rev*., vol. 62, no. 1, pp. 133–163, 2020.

[11] C. Févotte and A. Ozerov, "Notes on nonnegative tensor factorization of the spectrogram for audio source separation: statistical insights and towards self-clustering of the spatial cues," in *Intern. Symp. Computer Music Mod. Retr. (CMMR 2010)*, (Málaga, Spain), pp. 102–115, 2010.

[12] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale nonnegative matrix and tensor factorizations," *IEICE Trans. Fundamentals Electron. Commun. Comput. Sci*., vol. 92, no. 3, pp. 708–721, 2009.

[13] L. Lauwers, K. Barbé, W. Moer, and R. Pintelon, "Estimating the parameters of a Rice distribution: A Bayesian approach," in *IEEE Instrum. Meas. Techn. Conf. (I2MTC 2009)*, (Singapore), pp. 114–117, 2009.

[14] W. Liu, P. P. Pokharel, and J. C. Principe, "Correntropy: Properties and applications in non-Gaussian signal processing," *IEEE Trans. Signal Process*., vol. 55, no. 11, pp. 5286–5298, 2007.

[15] N. Vannieuwenhoven, K. Meerbergen, and R. Vandebril, "Computing the gradient in optimization algorithms for the CP decomposition in constant memory through tensor blocking," *SIAM J. Sci. Comput*., vol. 37, no. 3, pp. C415–C438, 2015.

[16] N. Vervliet, O. Debals, and L. De Lathauwer, "Exploiting efficient representations in large-scale tensor decompositions," *SIAM J. Sci. Comput*., vol. 41, no. 2, pp. A789–A815, 2019.

[17] N. Vervliet and L. De Lathauwer, "Numerical optimization based algorithms for data fusion," in *Data Fusion Methodology and Applications* (M. Cocchi, ed.), Elsevier, 2018.

[18] N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural Comput*., vol. 14, pp. 1723–1738, 2002.

[19] R. Verschueren, N. van Duijkeren, R. Quirynen, and M. Diehl, "Exploiting convexity in direct optimal control: a sequential convex quadratic programming method," in *Proc. 2016 IEEE 55th Conf. Decision, Control (CDC 2016)*, (Las Vegas, NV, USA), pp. 1099–1104, 2016.

[20] J. Nocedal and S. Wright, *Numerical optimization*. Springer New York, 7 2006.

[21] A.-H. Phan, P. Tichavský, and A. Cichocki, "Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations," *IEEE Trans. Signal Process*., vol. 61, no. 19, pp. 4834–4846, 2013.

[22] R. Hennequin, B. David, and R. Badeau, "Beta-divergence as a subclass of Bregman divergence," *IEEE Signal Process. Lett*., vol. 18, no. 2, pp. 83–86, 2011.

[23] C. Févotte and J. Idier, "Algorithms for nonnegative matrix factorization with the β-divergence," *Neural Comput*., vol. 23, no. 9, pp. 2421–2456, 2011.

[24] C. Févotte, N. Bertin, and J.-L. Durrieu, "Nonnegative matrix factorization with the Itakura–Saito divergence: With application to music analysis," *Neural Comput*., vol. 21, no. 3, pp. 793–830, 2009.

[25] E. C. Chi and T. G. Kolda, "On tensors, sparsity, and nonnegative factorizations," *SIAM J. Matrix Analysis Appl*., vol. 33, no. 4, pp. 1272–1299, 2012.

[26] S. Hansen, T. Plantenga, and T. G. Kolda, "Newton-based optimization for Kullback–Leibler nonnegative tensor factorizations," *Optim. Methods Softw*., vol. 30, no. 5, pp. 1002–1029, 2015.

[27] D. FitzGerald, M. Cranitch, and E. Coyle, "Extended nonnegative tensor factorisation models for musical sound source separation," *Comput. Intell. Neurosci*., vol. 2008, pp. 50–64, 2008.

[28] A. Yeredor and M. Haardt, "Maximum likelihood estimation of a low-rank probability mass tensor from partial observations," *IEEE Signal Process. Lett*., vol. 26, no. 10, pp. 1551–1555, 2019.

[29] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "Rank-one tensor approximation with beta-divergence cost functions," in *Proc. 27th Eur. Signal Process. Conf. (EUSIPCO 2019)*, (A Coruña, Spain), 2019.

[30] N. Bertin, C. Févotte, and R. Badeau, "A tempering approach for Itakura–Saito non-negative matrix factorization. with application to music transcription," in *2009 IEEE Intern. Conf. Acoustics, Speech, Signal Process. (ICASSP 2009)*, (Taipei, Taiwan), pp. 1545–1548, 2009.

[31] M. Boussé and L. De Lathauwer, "Nonlinear least squares algorithm for canonical polyadic decomposition using low-rank weights," in *2017 IEEE 7th Intern. Workshop Comp. Adv. Multi-Sensor Adapt. Process. (CAMSAP 2017)*, (Curaçao, Dutch Antilles), 2017.

[32] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab 3.0." Available online at http://www.tensorlab.net, Mar. 2016.

[33] M. Vandecappelle, N. Vervliet, and Lieven De Lathauwer, "Canonical polyadic decomposition of large tensors with non-least-squares cost functions using second-order information." Internal Report 20-54, ESAT-STADIUS, KU Leuven (Leuven, Belgium), 2020.

[34] R. W. Basedow, D. C. Carmer, and M. E. Anderson, "HYDICE system: implementation and performance," in *Imaging Spectrometry* (M. R. Descour, J. M. Mooney, D. L. Perry, and L. R. Illing, eds.), vol. 2480, pp. 258 – 267, International Society for Optics and Photonics, 1995.

[35] X. Liu, S. Bourennane, and C. Fossati, "Denoising of hyperspectral images using the PARAFAC model and statistical performance analysis," *IEEE Trans. Geosci. Remote. Sens*., vol. 50, no. 10, pp. 3717–3724, 2012.

[36] Y. Qian, Y. Shen, M. Ye, and Q. Wang, "3-D nonlocal means filter with noise estimation for hyperspectral imagery denoising," in *Proc. 2012 IEEE Intern. Geosc. Remote Sens. Symp. (IGARSS 2012)*, (Munich, Germany), pp. 1345–1348, 2012.

[37] Zhou Wang and A. C. Bovik, "A universal image quality index," *IEEE Signal Process. Lett*., vol. 9, no. 3, pp. 81–84, 2002.