Canonical polyadic decomposition of large tensors with non-least-squares cost functions using second-order information

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

***Abstract*—The canonical polyadic decomposition (CPD) allows one to extract compact and interpretable representations of tensors. Several optimization-based methods exist to fit the CPD of a tensor for the standard least-squares (LS) cost function. Extensions have been proposed for more general cost functions such as β-divergences as well. For these non-LS cost functions, a generalized Gauss–Newton (GGN) method has been developed. This is a second-order method that uses an approximation of the Hessian of the cost function to determine the next iterate and with this algorithm, fast convergence can be achieved close to the solution. Unfortunately, for large tensors, the exact GGN approach becomes too expensive. In this paper, we therefore propose an inexact GGN method. The approximation of the Hessian is only used implicitly, which greatly improves the scalability of the method. Next, we show that by using a compressed instance of the GGN Hessian approximation, the computation time of the inexact GGN method can be lowered even more, with only limited influence on the convergence speed. Further, the maximum likelihood estimator for Rician distributed data is examined in detail as an example of an alternative cost function. This cost function is useful for the analysis of the moduli of complex data, as in functional magnetic resonance imaging, for instance. Finally, we compare the proposed method to the existing CPD methods and demonstrate the method's speed and effectiveness on synthetic and simulated real-life data.**

I. Introduction

Many applications involve data with inherent higher-order structure, i.e., depending on more than two variables. Most of this higher-order structure is lost, however, when the information is stored in the traditional matrix format. Therefore, tensors, higher-order generalizations of vectors and matrices, have been proposed as a way to analyze such data with preservation of their higher-order structure. Tensors appear extensively within the fields of machine learning and signal processing and different kinds of tensor decompositions have been developed to enable better analysis of higher-order data [1], [2]. Even more than matrix decompositions, tensor decompositions give a compact representation of the data while maintaining their important features. This often has a denoising effect as the noise is generally not modeled by the decomposition. For large tensors, decompositions can also significantly reduce the number of parameters that are required to represent the data. One specific type of tensor decomposition is the canonical polyadic decompostion (CPD). This decomposition is a powerful tool in, for instance, signal separation applications, as it allows meaningful components to be extracted from the data. The CPD is particularly appealing due to its simplicity and mild uniqueness conditions [3], [4]. Aside from algebraic algorithms—see, for example, [5]—several optimization-based methods have been developed to compute the CPD of a given tensor, as described in, e.g., [6]–[8].

Typically, optimization-based methods use the least-squares (LS) distance as cost function to fit the CPD. The LS cost function is well suited for applications involving additive, independent and identically distributed (i.i.d.) errors from a Gaussian distribution, which is (approximately) the case in many applications [9]. However, these assumptions may not always hold and in specific settings better alternatives for the cost function exist. To allow other statistical assumptions, alternative cost functions such as, for instance, β-divergences can be used, resulting in new methods that can compute a CPD of a tensor with a range of non-LS cost functions.

In [10]–[12], different first-order, gradient-based methods are given to compute a CPD with general cost functions. To obtain faster convergence, second-order information can be used as well [13]. Although the per-iteration cost of second-order methods is usually higher than that of first-order methods, this is often more than compensated by the lower number of iterations that these methods require. In [14], a generalized Gauss–Newton (GGN) method is proposed that exploits the multilinear structure in the data to minimize the per-iteration cost. This method requires the construction of a full approximate Hessian, which takes $\mathcal{O}(I^3R^2)$ flop for an $I \times I \times I$ tensor of rank R. Although this cost is manageable for small to medium sized tensors, this is unfortunately no longer the case for large tensors,.

To improve the scalability of second-order methods for non-LS CPD fitting, we propose an inexact GGN method. Instead of constructing a full Hessian approximation, its information is accessed implicitly and efficiently. As a result, the computational cost of the GGN iterations is significantly reduced,

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 (AI Research Program).

*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.

which makes the method applicable for larger tensors. Additionally, we demonstrate that the GGN approximate Hessian can be compressed significantly by means of a low multilinear rank approximation. By exploiting the approximation's multilinear structure, the CPD computation can be sped up even more without negatively affecting the convergence speed. The properties of the proposed method are illustrated in detail for a non-LS cost function which is specifically tailored for Rician distributed data, which for instance arise as the moduli of complex data for which the real and imaginary parts are perturbed by additive Gaussian noise.

In the remainder of this section, notation is fixed and basic definitions are given. The inexact GGN algorithm is proposed in Sec. II, while in Sec. III, we examine Rician distributed data and the cost function that gives the maximum likelihood solution for such data. Lastly, numerical experiments are shown in Sec. IV. For simplicity of notation, third-order tensors are considered in this paper, but with minor modifications, the results extend to higher orders.

A. Notation

We denote scalars, vectors, matrices and tensors by lowercase (a), bold lowercase ($\mathbf{a}$), bold uppercase letters ($\mathbf{A}$) and calligraphic letters ($\mathcal{A}$), respectively. Powers, divisions and logarithms of matrices are entry-wise. 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 with $\mathbf{x}$ as its diagonal. When using subscripts to specify entries of $\mathcal{T}$, 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. As a higher-order extension of the vectorization $\mathrm{vec}(\mathbf{X}) \in \mathbb{R}^{IJ\times 1}$ of a matrix $\mathbf{X} \in \mathbb{R}^{I\times J}$, the vectorization $\mathbf{t} = \mathrm{vec}(\mathcal{T}) \in \mathbb{R}^{IJK\times 1}$ of the tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ maps t_{ijk} to t_q, with $q = i + (j-1)I + (k-1)IJ$. Its inverse (with context-implied dimensions) is denoted by $\mathrm{unvec}(\mathbf{t})$. 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 a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$. 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 $\mathbf{\Pi}^{(1)}$, is the identity 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, $\mathbf{\Pi}^{(1)^{\mathrm{T}}}$ is the identity, 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 $M \times M$-identity matrix is written as $\mathbf{I}_M$ and the Moore–Penrose pseudoinverse of a matrix $\mathbf{X}$ is written as $\mathbf{X}^{\dagger}$. 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 $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{I\times J}$ is written as $\mathbf{A} * \mathbf{B}$. The outer product of vectors is denoted by $\otimes$ and the mode-n tensor-matrix product of $\mathcal{T} \in \mathbb{R}^{I_1\times I_2\times I_3}$ and $\mathbf{X} \in \mathbb{R}^{J\times I_n}$ is written as $\mathcal{T} \cdot_n \mathbf{X}$. This product replaces every mode-n vector $\mathbf{t}$ of $\mathcal{T}$ by $\mathbf{X}\mathbf{t}$. For example, $\mathcal{T} \cdot_1 \mathbf{X} \in \mathbb{R}^{J\times I_2\times I_3}$ is obtained by replacing every mode-1 vector $\mathbf{t}_{:jk}$ of $\mathcal{T}$ by $\mathbf{X}\mathbf{t}_{:jk}$.

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. 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 $\mathbf{c}_r$ are collected as columns of factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, respectively. For a CPD $\mathcal{T} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, the mode-n unfoldings $\mathbf{T}_{(1)}$, $\mathbf{T}_{(2)}$ and $\mathbf{T}_{(3)}$ are

$$\mathbf{A}\left(\mathbf{C} \odot \mathbf{B}\right)^{\mathrm{T}},\ \mathbf{B}\left(\mathbf{C} \odot \mathbf{A}\right)^{\mathrm{T}}, \text{ and } \mathbf{C}\left(\mathbf{B} \odot \mathbf{A}\right)^{\mathrm{T}}, \tag{1}$$

respectively. Finally, a low multilinear rank approximation (LMLRA) of a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ is defined as $\mathcal{T} \approx \mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W}$, with $\mathcal{S} \in \mathbb{R}^{L\times M\times N}$, $\mathbf{U} \in \mathbb{R}^{I\times L}$, $\mathbf{V} \in \mathbb{R}^{J\times M}$, $\mathbf{W} \in \mathbb{R}^{K\times N}$ and $L \leq I$, $M \leq J$, $N \leq K$. It is also denoted by $[\![\mathcal{S}; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$. The multilinear rank of a tensor is the collection of ranks R_n of the mode-n unfoldings of the tensor and is written as (R_1, R_2, R_3).

II. CPD of large tensors with general cost functions

Fitting the CPD of a tensor is a challenging problem. The problem is nonconvex and optimization methods can end up in a suboptimal local minimum or a stationary point. Fortunately, methods that use second-order information, such as the Gauss–Newton (GN) algorithm, generally perform well for fitting the CPD. The GN method converges faster than first-order methods and, by exploiting the multilinear structure of the CPD, its computational cost can be kept low [7]. The CPD GN algorithm for the LS cost function has been generalized to arbitrary twice differentiable cost functions in [14]. This GGN method is shown to require much fewer iterations than first-order methods, with only a moderate increase of the per-iteration cost for tensors of small to moderate dimensions. Unfortunately, forming the full approximate Hessian matrix becomes prohibitively expensive for larger tensors. Therefore, we propose an inexact GGN method that does not require the full Hessian to be computed, but salvages its good convergence properties.

A. GGN for non-LS cost functions

Optimization-based algorithms typically fit a rank-R CPD $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ to a tensor $\mathcal{T} \in \mathbb{R}^{I\times J\times K}$ by minimizing the LS distance between the model and the tensor. With the vector $\mathbf{z} = [\mathrm{vec}(\mathbf{A}); \mathrm{vec}(\mathbf{B}); \mathrm{vec}(\mathbf{C})] \in \mathbb{R}^{(I+J+K)R}$ holding

all optimization variables, the CPD can thus be obtained by solving $\min_{\mathbf{z}} f(\mathbf{z})$, with

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

The triple sum over all tensor entries can be replaced by a single sum $\sum_{n=1}^{N}$, where $N = IJK$. As discussed in [14], the LS cost function for the CPD can be generalized to

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

with $m_n(\mathbf{z})$ the CPD model mapping the variables in $\mathbf{z}$ to the tensor elements in $\mathcal{M}$. Each f_n is a twice differentiable fit function for the corresponding model entry m_n. Examples of such alternative cost functions are β-divergences [15], the maximum likelihood estimator for data that follows a Rician distribution [16] and the correntropy function [17].

The GGN method requires the gradient $\mathbf{g} = \nabla_{\mathbf{z}} f$ of (2) and an approximation $\hat{\mathbf{H}}$ of the Hessian $\mathbf{H}$. The new step direction $\mathbf{p}$ is then obtained from solving the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$. This direction can be combined with the (first-order) Cauchy point within a (dogleg) trust-region approach [13] to guarantee global convergence of the method to a stationary point. When a trust region is used, 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. In [8], expressions for the gradient and GGN Hessian approximation are derived. The gradient $\mathbf{g}$ can be split into terms according to the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$: $\mathbf{g} = [\mathbf{g_A}; \mathbf{g_B}; \mathbf{g_C}]$. By exploiting the multilinear structure, the subgradients $\mathbf{g_A}$, $\mathbf{g_B}$ and $\mathbf{g_C}$ can be computed as $\mathbf{g_A} = \operatorname{vec}\left(\mathbf{R}_{(1)}(\mathbf{C}\odot\mathbf{B})\right)$, $\mathbf{g_B} = \operatorname{vec}\left(\mathbf{R}_{(2)}(\mathbf{C}\odot\mathbf{A})\right)$ and $\mathbf{g_C} = \operatorname{vec}\left(\mathbf{R}_{(3)}(\mathbf{B}\odot\mathbf{A})\right)$, where $\mathcal{R}$ 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 $\mathcal{M} - \mathcal{T}$. The subgradients $\mathbf{g_A}$, $\mathbf{g_B}$ and $\mathbf{g_C}$ can be computed efficiently using specialized routines to compute matricized tensor times Khatri–Rao products (MTKRPROD) [8], [18], [19]. For the GGN Hessian approximation $\hat{\mathbf{H}}$, the Jacobian matrix $\mathbf{J}$ is required. Using permutation matrices $\mathbf{\Pi}^{(i)}$, $\mathbf{J}$ can be written as $[\mathbf{J_A}; \mathbf{J_B}; \mathbf{J_C}] \in \mathbb{R}^{IJK\times(I+J+K)R}$ with $\mathbf{J_A} = (\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I$, $\mathbf{J_B} = \mathbf{\Pi}^{(2)^{\mathrm{T}}}((\mathbf{C}\odot\mathbf{A})\otimes\mathbf{I}_J)$, and $\mathbf{J_C} = \mathbf{\Pi}^{(3)^{\mathrm{T}}}((\mathbf{B}\odot\mathbf{A})\otimes\mathbf{I}_K)$. With the Jacobian, the Hessian approximation is defined as $\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, where $\mathbf{Z} = \operatorname{Diag}(\operatorname{vec}(\mathcal{D})) \in \mathbb{R}^{IJK\times IJK}$ and $d_n = \frac{\mathrm{d}^2 f_n}{\mathrm{d}m_n{}^2}$ for $n = 1, \ldots, N$. For the LS cost function, $\mathbf{Z} = \mathbf{I}_{IJK}$ and the Gramian $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ of the GN method is recovered. If the f_n are nonconvex, negative values can occur in $\mathbf{Z}$, especially far from the optimum. This can possibly introduce negative eigenvalues in $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ as well. In such cases, one should ensure that a descent direction is chosen and continued convergence of the optimization algorithm is guaranteed; see [8], [14].

Computing the full Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ can become expensive for large-scale problems. Fortunately, it is only required if one wants to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ exactly. Similar to inexact GN, we use an inexact GGN method to reduce the cost: an iterative solver such as (preconditioned) conjugate gradient (CG) [8], [20] or MINRES [21] is used to solve $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ approximately. CG is only applicable when $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is positive semi-definite (PSD). For convex f_n, the Hessian approximation is always PSD, but for nonconvex f_n, this is not guaranteed. In such cases, MINRES can be used instead. Both methods do not require the full matrix $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, but only the product of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ with an arbitrary vector $\mathbf{x}$, where $\mathbf{x} = [\operatorname{vec}(\mathbf{X_A}); \operatorname{vec}(\mathbf{X_B}); \operatorname{vec}(\mathbf{X_C})] = [\mathbf{x_A}; \mathbf{x_B}; \mathbf{x_C}]$ and $\mathbf{X_A}$, $\mathbf{X_B}$ and $\mathbf{X_C}$ have the same dimensions as $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, respectively. As a result, the $\mathcal{O}(I^3R^3)$ flop cost of solving the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ exactly can be reduced to $\mathcal{O}(kl)$, with k the number of iterations of the inexact solver and l its per-iteration cost. Below, we show how this per iteration cost l, computing the product $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$, can be kept low.

B. Computing $\mathbf{J}^T\mathbf{Z}\mathbf{J}\mathbf{x}$ efficiently

In the product $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$, the vector $\mathbf{y}$ can be split into three parts of the same dimensions as $\mathbf{x_A}$, $\mathbf{x_B}$ and $\mathbf{x_C}$, respectively. The parts $\mathbf{y_A}$, $\mathbf{y_B}$ and $\mathbf{y_C}$ of the product $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ can be obtained separately [8]. For the vector

$$\begin{aligned}\mathbf{y_A} &= \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_A}\mathbf{x_A} + \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_B}\mathbf{x_B} + \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_C}\mathbf{x_C}\\ &= \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\left(\mathbf{J_A}\mathbf{x_A} + \mathbf{J_B}\mathbf{x_B} + \mathbf{J_B}\mathbf{x_C}\right),\end{aligned} \tag{3}$$

the terms of the last factor can be rewritten as follows:

$$\begin{aligned}\mathbf{J_A}\mathbf{x_A} &= \left[(\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I\right]\mathbf{x_A}\\ &= \operatorname{vec}\left(\mathbf{X_A}\left(\mathbf{C}\odot\mathbf{B}\right)^{\mathrm{T}}\right),\\ \mathbf{J_B}\mathbf{x_B} &= \mathbf{\Pi}^{(2)^{\mathrm{T}}}\left[(\mathbf{C}\odot\mathbf{A})\otimes\mathbf{I}_J\right]\mathbf{x_B}\\ &= \mathbf{\Pi}^{(2)^{\mathrm{T}}}\operatorname{vec}\left(\mathbf{X_B}\left(\mathbf{C}\odot\mathbf{A}\right)^{\mathrm{T}}\right),\\ \mathbf{J_C}\mathbf{x_C} &= \mathbf{\Pi}^{(3)^{\mathrm{T}}}\left[(\mathbf{B}\odot\mathbf{A})\otimes\mathbf{I}_K\right]\mathbf{x_C}\\ &= \mathbf{\Pi}^{(3)^{\mathrm{T}}}\operatorname{vec}\left(\mathbf{X_C}\left(\mathbf{B}\odot\mathbf{A}\right)^{\mathrm{T}}\right).\end{aligned}$$

Herein, the identity

$$(\mathbf{U}\otimes\mathbf{V})\operatorname{vec}(\mathbf{W}) = \operatorname{vec}\left(\mathbf{V}\mathbf{W}\mathbf{U}^{\mathrm{T}}\right) \tag{4}$$

is used. By comparing with (1), it can be seen that each of the three terms can be seen as an unfolding of a CPD. The factor $(\mathbf{J_A}\mathbf{x_A} + \mathbf{J_B}\mathbf{x_B} + \mathbf{J_B}\mathbf{x_C})$ from (3) thus holds the sum of three unfolded CPDs. They are merged into one CPD $\mathcal{E} = \left[\!\left[\mathbf{E^A}, \mathbf{E^B}, \mathbf{E^C}\right]\!\right]$ by defining

$$\begin{aligned}\mathbf{E^A} &= \begin{bmatrix}\mathbf{X_A} & \mathbf{A} & \mathbf{A}\end{bmatrix} \in \mathbb{R}^{I\times 3R},\\ \mathbf{E^B} &= \begin{bmatrix}\mathbf{B} & \mathbf{X_B} & \mathbf{B}\end{bmatrix} \in \mathbb{R}^{J\times 3R},\\ \mathbf{E^C} &= \begin{bmatrix}\mathbf{C} & \mathbf{C} & \mathbf{X_C}\end{bmatrix} \in \mathbb{R}^{K\times 3R}.\end{aligned}$$

Exploiting the diagonal structure of $\mathbf{Z}$ then leads to the following expression for $\mathbf{y_A}$:

$$\mathbf{y_A} = \left[(\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I\right]^{\mathrm{T}}\left[\operatorname{vec}(\mathcal{D}) * \operatorname{vec}(\mathcal{E})\right].$$

Writing $\mathcal{Q} = \mathcal{D} * \mathcal{E}$ and applying (4) one more time, yields

$$\mathbf{y_A} = \operatorname{vec}\left(\mathbf{Q}_{(1)}(\mathbf{C}\odot\mathbf{B})\right). \tag{5}$$

Analogously, $\mathbf{y_B}$ and $\mathbf{y_C}$ can be found as

$$\mathbf{y_B} = \operatorname{vec}\left(\mathbf{Q}_{(2)}(\mathbf{C}\odot\mathbf{A})\right), \text{ and } \mathbf{y_C} = \operatorname{vec}\left(\mathbf{Q}_{(3)}(\mathbf{B}\odot\mathbf{A})\right).$$

The complexity of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ is dominated by the generation of the tensor $\mathcal{E}$ and the computation of the three MTKRPRODs. Assuming $I = J = K$ for simplicity, these each require $\mathcal{O}(I^3R)$ flop. In the following section, we discuss a strategy to reduce this complexity.

C. Approximating $\mathcal{D}$

When using a CG or MINRES algorithm, the step direction $\mathbf{p}$ is obtained from the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ by repeatedly evaluating products of the form $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$, wherein only the vector $\mathbf{x}$ varies between evaluations. Consequently, only the tensor $\mathcal{E}$ changes when $\mathbf{y}$ is computed as $\big[\mathrm{vec}\big(\mathbf{Q}_{(1)}(\mathbf{C}\odot\mathbf{B})\big); \mathrm{vec}\big(\mathbf{Q}_{(2)}(\mathbf{C}\odot\mathbf{A})\big); \mathrm{vec}\big(\mathbf{Q}_{(3)}(\mathbf{B}\odot\mathbf{A})\big)\big]$, with $\mathcal{Q} = \mathcal{D} * \mathcal{E}$. One of the most costly steps in the computation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ is the construction of the tensor $\mathcal{Q}$, as it requires a Hadamard product of the tensors $\mathcal{D}$ and $\mathcal{E}$. As the same $\mathcal{D}$ is reused in every iteration, it is often worthwhile to spend some time computing an LMLRA $\hat{\mathcal{D}}$ of $\mathcal{D}$. The information in $\mathcal{D}$ can be maintained up to a chosen precision, while the structure of its decomposition can be leveraged to efficiently obtain an approximation $\hat{\mathcal{Q}}$ of $\mathcal{Q}$. The second costly step, the MTKRPRODs of the form $\mathbf{y}_{\mathbf{A}} = \mathrm{vec}\left(\hat{\mathbf{Q}}_{(1)}(\mathbf{C}\odot\mathbf{B})\right)$, can then also be computed much more efficiently by exploiting the structure of $\hat{\mathcal{Q}}$.

Let $\hat{\mathcal{D}} = [\![S; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!]$ be an LMLRA of $\mathcal{D}$, with $\mathcal{S} \in \mathbb{R}^{R_1\times R_2\times R_3}$. Then, the Hadamard product $\hat{\mathcal{Q}} = \hat{\mathcal{D}} * [\![\mathbf{E}^{\mathbf{A}}, \mathbf{E}^{\mathbf{B}}, \mathbf{E}^{\mathbf{C}}]\!]$ is structured as well:

$$\begin{aligned}\hat{\mathcal{Q}} &= [\![S; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!] * [\![\mathbf{E}^{\mathbf{A}}, \mathbf{E}^{\mathbf{B}}, \mathbf{E}^{\mathbf{C}}]\!] \\ &= \sum_{r=1}^{3R} [\![S; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!] * [\![\mathbf{e}^{\mathbf{A}}_{:r}, \mathbf{e}^{\mathbf{B}}_{:r}, \mathbf{e}^{\mathbf{C}}_{:r}]\!] \\ &= \sum_{r=1}^{3R} [\![\mathcal{S}; \mathbf{E}^{\mathbf{A}}_r\mathbf{U}, \mathbf{E}^{\mathbf{B}}_r\mathbf{V}, \mathbf{E}^{\mathbf{C}}_r\mathbf{W}]\!], \end{aligned} \tag{6}$$

wherein $\mathbf{E}^{\mathbf{A}}_r = \mathrm{Diag}\left(\mathbf{e}^{\mathbf{A}}_{:r}\right)$, $\mathbf{E}^{\mathbf{B}}_r = \mathrm{Diag}\left(\mathbf{e}^{\mathbf{B}}_{:r}\right)$ and $\mathbf{E}^{\mathbf{C}}_r = \mathrm{Diag}\left(\mathbf{e}^{\mathbf{C}}_{:r}\right)$. Because of the structure of $\mathbf{E}^{\mathbf{A}}$, some matrices $\mathbf{E}^{\mathbf{A}}_r\mathbf{U}$ are duplicates. Naturally, these have to be computed only once. The same holds for $\mathbf{E}^{\mathbf{B}}_r\mathbf{V}$ and $\mathbf{E}^{\mathbf{C}}_r\mathbf{W}$. Using (6), $\mathbf{y}_{\mathbf{A}} = \hat{\mathbf{Q}}_{(1)}\left(\mathbf{C}\odot\mathbf{B}\right)$ can be efficiently computed as

$$\begin{aligned}\mathbf{y}_{\mathbf{A}} &= \sum_{r=1}^{3R} (\mathbf{E}^{\mathbf{A}}_r\mathbf{U})\mathbf{S}_{(1)}\left((\mathbf{E}^{\mathbf{C}}_r\mathbf{W})\otimes(\mathbf{E}^{\mathbf{B}}_r\mathbf{V})\right)^{\mathrm{T}}(\mathbf{C}\odot\mathbf{B}) \\ &= \sum_{r=1}^{3R} (\mathbf{E}^{\mathbf{A}}_r\mathbf{U})\mathbf{S}_{(1)}\left(\left[(\mathbf{E}^{\mathbf{C}}_r\mathbf{W})^{\mathrm{T}}\mathbf{C}\right]\odot\left[(\mathbf{E}^{\mathbf{B}}_r\mathbf{V})^{\mathrm{T}}\mathbf{B}\right]\right), \end{aligned} \tag{7}$$

with $\mathbf{E}^{\mathbf{A}}_r\mathbf{U} \in \mathbb{R}^{I\times R_1}$, $(\mathbf{E}^{\mathbf{C}}_r\mathbf{W})^{\mathrm{T}}\mathbf{C} \in \mathbb{R}^{R_3\times R}$ and $(\mathbf{E}^{\mathbf{B}}_r\mathbf{V})^{\mathrm{T}}\mathbf{B} \in \mathbb{R}^{R_2\times R}$. Compared to (5), this is again a MTKRPROD, but now involving smaller tensors and matrices. The same strategy can be applied in the other two modes to obtain $\mathbf{y}_{\mathbf{B}} = \hat{\mathbf{Q}}_{(2)}(\mathbf{C}\odot\mathbf{A})$ and $\mathbf{y}_{\mathbf{C}} = \hat{\mathbf{Q}}_{(3)}(\mathbf{B}\odot\mathbf{A})$, wherein terms of the form $\mathbf{E}^{\mathbf{A}}_r\mathbf{U}$ and $\left(\mathbf{E}^{\mathbf{B}}_r\mathbf{V}\right)^{\mathrm{T}}\mathbf{B}$ can be reused from the computation of $\mathbf{y}_{\mathbf{A}}$.

Using specialized methods, e.g., a sequentially truncated multilinear singular value decompostion (ST-MLSVD) algorithm with randomized SVD computation [22], [23], a rank-$(R_{\mathrm{ML}}, R_{\mathrm{ML}}, R_{\mathrm{ML}})$ LMLRA of an $I \times J \times K$-tensor $\mathcal{D}$ can be computed efficiently, with the approximation having good numerical properties due to the orthogonality constraints. If forming or storing the full tensor $\mathcal{D}$ is too expensive, one can sample entries or fibers from it and compute an LMLRA of this incomplete tensor instead [24]–[27].

With a rank-$(R_{\mathrm{ML}}, R_{\mathrm{ML}}, R_{\mathrm{ML}})$ LMLRA of $\mathcal{D}$, the complexity of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ is dominated by the products of the form $\mathbf{M} = \mathbf{S}_{(1)}\left(\left[(\mathbf{E}^{\mathbf{C}}_r\mathbf{W})^{\mathrm{T}}\mathbf{C}\right]\odot\left[(\mathbf{E}^{\mathbf{B}}_r\mathbf{V})^{\mathrm{T}}\mathbf{B}\right]\right)$ and $\mathbf{y}_{\mathbf{A}} = (\mathbf{E}^{\mathbf{A}}_r\mathbf{U})\mathbf{M}$ from (7). Again assuming $I = J = K$, these have a complexity of $\mathcal{O}(RR^3_{\mathrm{ML}}))$ flop and $\mathcal{O}(RR_{\mathrm{ML}}I)$ flop, respectively. This leads to a total complexity of $\mathcal{O}(R^2R_{\mathrm{ML}}\max(I, R^2_{\mathrm{ML}}))$ flop for the entire $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$-evaluation. Compared to the complexity of $\mathcal{O}(I^3R)$ flop when using the full $\mathcal{D}$-tensor, we see that the approximation decreases the complexity of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ whenever $RR_{\mathrm{ML}} < I^2$ and $RR^3_{\mathrm{ML}} < I^3$. Taking into account that in applications typically $I \gg R$, while by definition $I \geq R_{\mathrm{ML}}$, these conditions are easily satisfied for realistic values of R_{ML}, R and I. Of course, one should also take the computation time for the LMLRA itself into account. The LMLRA is thus only useful if enough $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$-evaluations are required in the CG or MINRES computations. In addition, one should be careful not to take R_{ML} so small that it yields a poor approximation, which can lead to a bad step in the optimization procedure. This can then increase the number of iterations of the optimization algorithm, negating the computational gain that replacing $\mathcal{D}$ by its LMLRA yields. In Sec. IV-B, we discuss strategies for picking an adequate value of R_{ML}.

D. Preconditioners

The convergence rate of an iterative solver for the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ is generally higher when the eigenvalues of the matrix $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ are well clustered. To improve the spectral properties of this matrix, a suitable preconditioner can be used. A good preconditioner is cheap to apply and clusters the eigenvalues of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ around one, which should reduce the required number of CG or MINRES iterations. Using the preconditioner $\mathbf{M}$, the modified system $\mathbf{M}^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x} = -\mathbf{M}^{-1}\mathbf{g}$ is solved instead of the original system.

In iterative methods, the preconditioner is applied implicitly in every iteration, i.e., only solving systems of the form $\mathbf{M}\mathbf{a} = \mathbf{b}$ is required instead of the multiplication of $\mathbf{M}^{-1}$ with the full $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$-matrix. Application of a preconditioner should be more efficient than solving the original system. In order to achieve this, $\mathbf{M}$ is typically constructed to be sparse and/or structured. This is also the case for the block-Jacobi and Jacobi preconditioners that we propose below. When showing complexities, we assume $I = J = K$, as before. In Sec. IV-A, the efficiency and effectiveness of both preconditioners is compared.

1) Block-Jacobi: A block-Jacobi preconditioner $\mathbf{M}$ for the optimization problem is a block-diagonal matrix with $\mathbf{G}^{(n,n)} = \mathbf{J}^{\mathrm{T}}_n\mathbf{Z}\mathbf{J}_n$ as its blocks. To solve $\mathbf{M}\mathbf{a} = \mathbf{b}$, one thus needs an efficient method to solve $\mathbf{G}^{(n,n)}\mathbf{a}^{(n)} = \mathbf{b}^{(n)}$. The method from [14] can be applied to obtain the highly structured diagonal blocks $\mathbf{G}^{(n,n)}$. Matrices $\mathbf{G}^{(n,n)}_{\mathrm{nz}}$ holding

only the nonzero entries of the $\mathbf{G}^{(n,n)}$-matrices are efficiently computed as

$$\mathbf{G}_{\text{nz}}^{(1,1)} = \mathbf{D}_{(1)} \left((\mathbf{C} \odot^{\text{T}} \mathbf{C}) \odot (\mathbf{B} \odot^{\text{T}} \mathbf{B}) \right),$$
$$\mathbf{G}_{\text{nz}}^{(2,2)} = \mathbf{D}_{(2)} \left((\mathbf{C} \odot^{\text{T}} \mathbf{C}) \odot (\mathbf{A} \odot^{\text{T}} \mathbf{A}) \right),$$
$$\mathbf{G}_{\text{nz}}^{(3,3)} = \mathbf{D}_{(3)} \left((\mathbf{B} \odot^{\text{T}} \mathbf{B}) \odot (\mathbf{A} \odot^{\text{T}} \mathbf{A}) \right),$$

and the zeros are added afterwards. The construction of one diagonal block requires $\mathcal{O}(I^3R^2)$ flop. If a rank-$(R_{\text{ML}}, R_{\text{ML}}, R_{\text{ML}})$ LMLRA of $\mathcal{D}$ is available, its structure can be exploited to reduce the computational load of the construction of the diagonal blocks to $\mathcal{O}(\max(I, R_{\text{ML}}^2) R_{\text{ML}} R^2)$ flop. After forming the diagonal blocks, applying the preconditioner, i.e., solving $\mathbf{Ma} = \mathbf{b}$, can be done efficiently by exploiting the sparsity in the blocks. More specifically, for the $\mathbf{G}^{(1,1)}$-block, I independent LDL factorizations [28] of $R \times R$-matrices can be computed. The back substitution steps then involve only R iterations working on length-I vectors. The $\mathbf{G}^{(2,2)}$- and $\mathbf{G}^{(3,3)}$-blocks are factored analogously. The LDL factorization, which costs $\mathcal{O}(IR^3)$ operations per block, only has to be computed once. Afterwards, the computed decomposition of $\mathbf{M}$ can be reused in every CG or MINRES iteration to solve $\mathbf{G}^{(n,n)}\mathbf{a}^{(n)} = \mathbf{b}^{(n)}$ via back substitution in $\mathcal{O}(IR^2)$ flop per block. In contrast, the $\mathbf{J}^{\text{T}}\mathbf{ZJx}$ product itself requires $\mathcal{O}(I^3R)$ flop.

The block-Jacobi preconditioner reduces the computational cost if the reduction in the number of CG or MINRES iterations compensates for the overhead of computing and decomposing the $\mathbf{G}^{(n,n)}$-matrices. In Sec. IV-A, we show that, for typical values of I, R and R_{ML}, a significant reduction is achieved.

2) Jacobi: For a Jacobi preconditioner, $\mathbf{M}$ is the diagonal of $\mathbf{J}^{\text{T}}\mathbf{ZJ}$. To compute $\mathbf{M}$, the inner product of each row of $\mathbf{J}$ with itself has to be computed, whereby each entry has to be weighted with its corresponding entry of $\mathcal{D}$. This leads to the following expression for the diagonal $\mathbf{m}_{\text{diag}}$ of $\mathbf{M}$:

$$\mathbf{m}_{\text{diag}} = \begin{bmatrix} \text{vec}\left(\mathbf{D}_{(1)}\left(\mathbf{C}^{\bullet 2} \odot \mathbf{B}^{\bullet 2}\right)\right) \\ \text{vec}\left(\mathbf{D}_{(2)}\left(\mathbf{C}^{\bullet 2} \odot \mathbf{A}^{\bullet 2}\right)\right) \\ \text{vec}\left(\mathbf{D}_{(3)}\left(\mathbf{B}^{\bullet 2} \odot \mathbf{A}^{\bullet 2}\right)\right) \end{bmatrix}.$$

The system $\mathbf{Ma} = \mathbf{b}$ can then be solved by computing $\mathbf{a} = \frac{\mathbf{b}}{\mathbf{m}_{\text{diag}}}$. Forming $\mathbf{m}_{\text{diag}}$ requires $\mathcal{O}(I^3R)$ flop. As before, the same $\mathbf{m}_{\text{diag}}$ can be used for all evaluations of $\mathbf{J}^{\text{T}}\mathbf{ZJx}$. Only $\mathbf{x} = \frac{\mathbf{y}}{\mathbf{m}_{\text{diag}}}$ (costing $\mathcal{O}(IR)$ flop) has to be computed in every evaluation, so the preconditioner needs to save only one $\mathbf{J}^{\text{T}}\mathbf{ZJx}$ evaluation to reduce the computation time. If a rank-$(R_{\text{ML}}, R_{\text{ML}}, R_{\text{ML}})$ LMLRA of $\mathcal{D}$ is available, this reduces the complexity of computing $\mathbf{m}_{\text{diag}}$ to $\mathcal{O}(R_{\text{ML}} R \max(I, R_{\text{ML}}^2))$ flop, making the preconditioner even more efficient.

III. Alternative cost functions

As mentioned in Sec. II, many alternatives exist for the standard LS cost function. Depending on the application and the distribution of the noise, they will give better or worse results. β-divergences are a popular choice of alternative cost function. Their behavior depends on the value of β: for small values of β, smaller values are given more importance compared to the LS cost function, while the converse is true for large values of β. Additionally, they interpolate continuously between three popular cost functions: the Itakura–Saito divergence ($\beta = 0$), the Kullback–Leibler divergence ($\beta = 1$) and the LS distance ($\beta = 2$). The β-divergence is discussed extensively in the context of CPD fitting in [14].

In this section, we consider another non-LS cost function for the CPD. It is particularly useful when the tensor entries follow a Rician distribution. In such cases, modeling with the Gaussian distribution introduces a bias [29], especially when the variance of the noise is large. Consider a complex measurement $z = N(\mu_1, \sigma^2) + jN(\mu_2, \sigma^2)$, with $N(\mu, \sigma^2)$ the Gaussian (or normal) distribution with mean μ and standard deviation σ. Herein, $\mu_1 + j\mu_2$ is the underlying unperturbed measurement and σ is the standard deviation of the zero-mean additive Gaussian noise on both the real and imaginary parts. Consequently $A = \sqrt{\mu_1^2 + \mu_2^2}$ corresponds to the unperturbed magnitude of the measurement. The distribution of its modulus $x = |z|$ then has a Rician probability density function:

$$p(x|A, \sigma) = \frac{x}{\sigma^2} \exp - \frac{x^2 + A^2}{2\sigma^2} I_0\left(\frac{Ax}{\sigma^2}\right),$$

with I_k the modified Bessel function of the first kind with order k. For $A = 0$, i.e., when the unperturbed measurement is zero, the Rician distribution is equal to the Rayleigh distribution. The optimum of the LS cost function does not coincide with the MLE of Rician distributed data. Instead, the MLE for a variable x following a Rician distribution with noise variance σ^2, is the solution of

$$\min_y \frac{y^2}{2\sigma^2} - \log(I_0(yx\sigma^2)).$$

Some methods to find the MLE and alternative estimators are given in [16], [30], [31]. For a tensor $\mathcal{T}$ with Rician distributed data and its CPD $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, one finds $\min_{\mathbf{z}} f(\mathbf{z})$, with

$$f(\mathbf{z}) = \sum_{n=1}^{N} \frac{m_n^2}{2\sigma^2} - \log\left(I_0\left(\frac{m_n t_n}{\sigma^2}\right)\right),$$

wherein the Bessel function is applied entry-wise. As the t_n are moduli, we restrict $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ to be nonnegative. For this cost function, the tensors $\mathcal{R}$ and $\mathcal{D}$, which are defined in Sec. II-A and used in the computation of the gradient and the Hessian approximation, respectively, satisfy:

$$\mathcal{R} = \frac{\mathcal{M}}{\sigma^2} - \frac{\mathcal{T}}{\sigma^2} * \frac{I_1(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})}{I_0(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})} \quad \text{and}$$

$$\mathcal{D} = \frac{1}{\sigma^2} - \left(\frac{\mathcal{T}^{\bullet 2}}{\sigma^4} - \frac{\mathcal{T}}{\sigma^2 \mathcal{M}} * \frac{I_1(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})}{I_0(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})} - \frac{\mathcal{T}^{\bullet 2}}{\sigma^4} * \left(\frac{I_1(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})}{I_0(\frac{\mathcal{M}*\mathcal{T}}{\sigma^2})} \right)^{\bullet 2} \right).$$

The f_n are not necessarily convex, so $\mathcal{D}$ may have negative values, especially when $\frac{t_n}{m_n}$ is large. As explained in Sec. II-A, this can lead to an indefinite Hessian approximation $\mathbf{J}^{\text{T}}\mathbf{ZJ}$ which impedes fast convergence of the algorithm. Fortunately, as for β-divergence cost functions, the LS solution often offers a decent initialization of the algorithm, which ideally implies the nonnegativity of $\mathcal{D}$.

Rician distributed data occurs for instance in medical applications where analysis focuses on the magnitude of a complex signal, such as functional magnetic resonance

imaging (fMRI) [29]. The real and imaginary parts of the complex-valued Fourier spectra can often both be modeled as Gaussians with different means but the same variance. To avoid problems due to incidental phase variations, magnitude images are predominantly used in fMRI analysis instead of the complex-valued Fourier spectra themselves [29]. As the magnitude of each measurement is not Gaussian distributed, but Rician, using the LS cost function introduces a bias which can influence the interpretability of the results. For such fMRI images, the Rician-dedicated cost function is more suitable.

IV. Experiments

Four experiments are performed on synthetic and simulated real-life data. First, we compare the performance of different preconditioners on the computation time and number of iterations to convergence that are required. Second, the influence on the computational cost of using an approximation of the tensor $\mathcal{D}$ is studied. Next, we show how the method performs compared to the small-scale version and a standard method from the literature and finally, we use our method to separate simulated fMRI sources. 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 will be sampled independently from the uniform distribution on the interval (a, b), denoted by $U(a, b)$. Setting $\bar{\mathcal{T}}$ and $\mathcal{T}$ as the perturbed and unperturbed tensors, respectively, the SNR is defined as the ratio of the summed squared magnitude of the signal to the summed squared magnitude of the noise: $20\log_{10}\left(\frac{\|\mathcal{T}\|_{\mathrm{F}}}{\|\mathcal{N}\|_{\mathrm{F}}}\right)$, where $\mathcal{N} = \bar{\mathcal{T}} - \mathcal{T}$.

A. Preconditioning

We have proposed two different preconditioners for the GGN method: a diagonal Jacobi preconditioner and a block-diagonal block-Jacobi preconditioner. In Table I, the median number of CG iterations required to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ up to a tolerance of 10^{-6} is compared for the GGN method with and without preconditioning. Preconditioning clearly decreases the number of CG iterations that are required to find the next iterate of the GGN method. With the block-Jacobi preconditioner, even fewer CG steps are needed than with the Jacobi preconditioner. However, the higher computation time of the former makes it less attractive than the latter for smaller tensors. For all problems in Table I, both preconditioners are effective in decreasing the computation time.

Table I
Preconditioners can reduce the number of CG or MINRES steps in the GGN method. With the block-Jacobi preconditioner, fewer steps are required than with the Jacobi preconditioner, but the latter can be computed much more efficiently, leading to overall faster iterations for smaller tensors. A β-divergence cost function with $\beta = 1.5$ is used to decompose the $I \times I \times I$-tensors and all noise is multiplicative Gamma noise using unit shape and scale. The preconditioning time is included in the GGN iteration time. Results are medians over 20 trials.

	Preconditioner		
Problem	None	Jacobi	Block-J.
$I = 50$, $R = 5$, no noise			
Median CG steps	110	73.5	44
Time per GGN iteration (s)	0.4427	0.3346	0.2526
Prec. time per GGN iteration (s)	0	0.0053	0.0321
$I = 50$, $R = 5$, 20dB noise			
Median CG steps	88	65	41
Time per GGN iteration (s)	0.4461	0.3274	0.2446
Prec. time per GGN iteration (s)	0	0.0053	0.0321
$I = 30$, $R = 3$, no noise			
Median CG steps	75	33	23
Time per GGN iteration (s)	0.1813	0.0772	0.1069
Prec. time per GGN iteration (s)	0	0.0025	0.0133
$I = 30$, $R = 3$, 20dB noise			
Median CG steps	49	33	21
Time per GGN iteration (s)	0.1218	0.0743	0.0765
Prec. time per GGN iteration (s)	0	0.0025	0.0133

B. Approximating $\mathcal{D}$

Using non-LS cost functions adds complexity to the CPD algorithm, which naturally results in a larger computational cost of the method. In Fig. 1, the computation time of 100 $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ evaluations is shown in function of the tensor size. In our experiments, around 100 $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ evaluations are generally required when using the CG or MINRES algorithms to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$, although with preconditioning, around 50 can often be achieved. From Fig. 1, we can see that the computation of the product $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ becomes very costly for large tensors. For such tensors, the approach of Sec. II-C can be used to approximate the tensor $\mathcal{D}$ with an LMLRA. This decomposition is then used instead of the full $\mathcal{D}$ in the $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ evaluations of the CG or MINRES algorithm. Exploiting the LMLRA structure speeds up these evaluations significantly. In Fig. 1, one can also see the computation time of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ with compression of $\mathcal{D}$. For small tensors, using a compressed $\mathcal{D}$ actually increases the computation time, because of the higher number of `MTKRPROD`s that are needed compared to the uncompressed variant ($9R$ compared to 3). This makes the compressed format more expensive to compute than simply forming all entries of $\mathcal{Q} = \mathcal{D} * \mathcal{E}$. However, the computation time for $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ grows exponentially with increasing tensor dimensions when $\mathcal{D}$ is not compressed. In contrast, the computation time stays about the same when $\mathcal{D}$ is compressed (and R_{ML} is kept constant), leading to considerable efficiency gains for large tensors. Computing the compression takes time by itself, so it is mostly useful when many $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ products have to be computed with the same $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$-matrix, as is typically the case.

To assess the effect of the compression on the convergence speed, results for different compression factors are compared in Table II for rank-4 $I \times I \times I$ tensors with $I = 50$, $I = 100$ and $I = 200$, perturbed with multiplicative Gamma noise using unit shape and scale and an SNR of 40 dB. A β-divergence cost function is used with $\beta = 1.5$ and the factor matrices are initialized randomly with entries sampled from $U(0, 1)$. The mean number of GGN iterations and computation time until convergence are shown when $\mathcal{D}$ is approximated by an LMLRA with core dimensions $S \times S \times S$. Only successful runs are taken into account, where the algorithm converged to the correct solution within 500 GGN iterations.

The compression factors I/S that are considered are 1, 5, 10 and 25. The tolerance of the CG/MINRES algorithm is 10^{-8}. First, it can be seen that for $I = 100$ and $I = 200$, the computation time is significantly reduced by compressing $\mathcal{D}$. For a compression factor of 5, the number of iterations until convergence is actually lower than when the full $\mathcal{D}$-tensor is used, which evidently reduces the computation time as well. For the large compression factors 10 and 25, the lower quality of the approximation sometimes leads to an increase of the number of GGN iterations, but the compression still offers a time gain compared to using the full $\mathcal{D}$-tensor. For smaller tensors with $I = 50$, the approximation of $\mathcal{D}$ increases the computation time and number of iterations, so no computational benefit is achieved.

To ensure that the convergence speed is not hindered by the use of an inadequate approximation rank, an adaptive compression factor can be used. By trimming only multilinear singular values under a certain threshold, a sufficient accuracy of the approximation can be guaranteed. As an example, we performed the experiment from Table II for $I = 100$ with an adaptive compression factor. We chose a compression factor of 25 and, if necessary, increased the multilinear rank of the core tensor until the sum of the squared multilinear singular values of $\hat{\mathcal{D}}$ summed to at least 99.9% of $\|\mathcal{D}\|_{\mathrm{F}}^2$. This led to a mean time of 10s and a mean number of GGN iterations of 43, while all 20 runs converged to the correct solution. Strategically increasing the multilinear rank of the compression thus improved the convergence behavior compared to a fixed compression factor of 25. The adaptive method performed even better than the best method from Table II, the one with a fixed compression factor of 5, without the need to specify the compression factor a priori.

In practice, the LS solution, which can be obtained with a cheaper method, is often a good initialization for the non-LS GGN method. Therefore, we can focus on the convergence behavior of the GGN method close to the solution. To analyze this behavior, the same experiment as above is performed, but now using the LS solution as initialization. This initialization is obtained by running the LS method until $\|\mathcal{M} - \mathcal{T}\|_{\mathrm{F}}^2$ drops below a value of $I^3 \cdot 10^{-4}$. In Table III, the results can be seen for a β-divergence cost function with $\beta = 1.5$ and $\beta = 0.5$. Again, the number of GGN iterations and the computation time until convergence are shown when $\mathcal{D}$ is approximated by an LMLRA with core dimensions $S \times S \times S$. The mean number of CG (for $\beta = 1.5$) or MINRES (for $\beta = 0.5$) iterations used to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x} = -\mathbf{g}$ is also shown. Comparing the column for compression factor 5 to the column for compression factor 1 (no compression), shows that compressing $\mathcal{D}$ can significantly reduce the runtime of the algorithm. For large tensors, the number of GGN iterations was only moderately affected by the compression and for $\beta = 1.5$ even compression factors up to 25 were viable.

There is a limit on how much time can be gained by compressing $\mathcal{D}$. A first limiting factor is that the cost function and gradient evaluations already take a significant amount of time for β-divergences. Even if the time required for the $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ evaluations could be eliminated entirely, this cost would remain. Second, from the experiments in Table III with

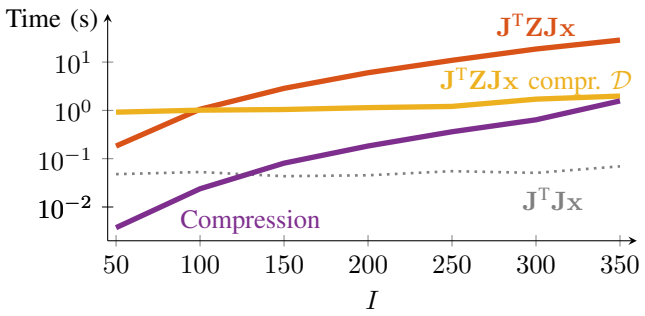


Fig. 1. When $\mathcal{D}$ is compressed, the computation time for an evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ remains constant for increasing I, while it increases exponentially without compression. Mean time (over 20 trials) for 100 evaluations of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ with the same $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$-matrix (a typical value using CG) for the β-divergence cost function ($\beta = 1.5$) in function of the tensor size. Results are shown with and without compression of $\mathcal{D}$. The tensors have dimensions $I \times I \times I$ and rank 10. The CPD model also has rank 10. A rank-$(I/25, I/25, I/25)$ approximation was used for $\mathcal{D}$. The compression time is shown as well. The computation time for 100 evaluations of $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{x}$ for the LS distance is added for reference.

Table II
COMPRESSING $\mathcal{D}$ DECREASES THE COMPUTATION TIME AND NUMBER OF ITERATIONS FOR LARGE TENSORS. MEAN (OVER 20 TRIALS) COMPUTATION TIME AND NUMBER OF GGN ITERATIONS UNTIL CONVERGENCE USING A β-DIVERGENCE COST FUNCTION WITH $\beta = 1.5$ WHEN $\hat{\mathcal{D}}$ IS A RANK-(S, S, S) LMLRA. DIFFERENT COMPRESSION FACTORS I/S ARE COMPARED FOR AN $I \times I \times I$-TENSOR OF RANK 4. THE NOISE IS MULTIPLICATIVE GAMMA NOISE USING UNIT SHAPE AND SCALE AND AN SNR OF 40 dB.

	Compression factor I/S			
	1	5	10	25
$I = 50$				
Correct solution /20	20	20	20	20
# GGN it.	26.1	29.2	27.9	34.2
Time (s)	4.21	5.06	4.47	5.43
$I = 100$				
Correct solution /20	19	19	19	19
# GGN it.	34.7	27.7	40.3	50.0
Time (s)	23.34	12.08	15.22	12.84
$I = 200$				
Correct solution /20	16	18	18	18
# GGN it.	66.9	64.7	66.8	57.11
Time (s)	138.94	85.06	64.66	48.55

$\beta = 0.5$, for which the f_n are nonconvex, it can be seen that the approximation becomes progressively worse if the compression factor is increased. If the approximation becomes too bad, for example, when $\beta = 0.5$, $I = 100$ and $I/S = 10$, this increases the number of iterations that are needed to reach the solution which in turn increases the runtime of the algorithm. Also, while $\mathcal{D}$ has all entries equal to one when $\beta = 2$, the entries change proportionally with the distance $|\beta - 2|$. As a result, a higher multilinear rank and thus a larger core tensor is required to approximate this tensor well with an LMLRA when $|\beta - 2|$ is large, which can be seen by comparing the results for $\beta = 1.5$ and $\beta = 0.5$.

The system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x} = -\mathbf{g}$ is altered when $\mathcal{D}$ is approximated. Even if this approximated system is solved exactly, its solution will be (slightly) different from the solution of the true system. The approximation could also influence the number of CG or MINRES iterations that are needed to obtain a solution with the requested accuracy. In this experiment, however, this number is largely unaffected by the compression.

Table III
PROPER COMPRESSION OF $\mathcal{D}$ DECREASES THE COMPUTATION TIME FOR LARGE TENSORS. MEAN (OVER 20 TRIALS) COMPUTATION TIME AND NUMBER OF GGN ITERATIONS UNTIL CONVERGENCE STARTING CLOSE TO THE LS SOLUTION. A β-DIVERGENCE COST FUNCTION IS USED WITH $\beta \in \{0.5, 1.5\}$. $\hat{\mathcal{D}}$ HAS MULTILINEAR RANK (S, S, S). THE MEAN (OVER ALL GGN ITERATIONS) NUMBER OF CG (FOR THE CONVEX f_n OF $\beta = 1.5$) AND MINRES (FOR THE NONCONVEX f_n OF $\beta = 0.5$) ITERATIONS TO SOLVE $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x} = -\mathbf{g}$ IS ALSO LISTED. COMPRESSION FACTORS I/S ARE COMPARED FOR AN $I \times I \times I$-TENSOR OF RANK 4. THE NOISE IS MULTIPLICATIVE GAMMA NOISE USING UNIT SHAPE AND SCALE AND AN SNR OF 40 dB.

	Compression factor I/S			
Criterion ($\beta = 1.5$)	1	5	10	25
$I = 50$				
# GGN it.	3.2	3.3	4.2	5.4
Mean # CG it.	56.3	56.7	56.9	56.8
Time (s)	0.34	0.59	0.70	0.89
$I = 100$				
# GGN it.	3.0	3.0	3.1	4.2
Mean # CG it.	77.3	77.3	77.3	78.0
Time (s)	2.07	1.60	1.49	1.42
$I = 200$				
# GGN it.	3.0	3.0	3.0	3.3
Mean # CG it.	76.5	76.7	76.7	77.1
Time (s)	13.36	5.62	4.55	4.54
Criterion ($\beta = 0.5$)				
$I = 50$				
# GGN it.	3.9	7.7	12.2	12.3
Mean # MINRES it.	51.9	53.3	57.1	53.8
Time (s)	0.37	1.43	4.72	3.73
$I = 100$				
# GGN it.	3.8	4.1	9.3	9.6
Mean # MINRES it.	121.1	123.5	132.6	156.9
Time (s)	3.71	2.95	8.96	11.15
$I = 200$				
# GGN it.	3.8	3.9	4.5	10.3
Mean # MINRES it.	135.1	137.0	139.7	164.7
Time (s)	26.62	8.17	7.54	25.99

C. Other β-divergence CPD algorithms

In this subsection, we compare the inexact GGN method to other methods to compute CPDs with general cost functions. First, a comparison with the exact GGN method in function of the tensor dimensions can be seen in Fig. 2. Rank-10 tensors of dimensions $I \times I \times I$ are generated with factor matrix entries sampled from $U(0, 1)$ and the tensor $\mathcal{D}$ is computed. For the exact GGN method, the mean time (over 20 trials) to compute $\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ is shown, alongside the mean time to compute $\hat{\mathbf{H}}^{\dagger}$. For the inexact GGN method, the mean time is displayed to compress the tensor $\mathcal{D}$ to a rank-$(10, 10, 10)$ tensor, next to the mean computation time of 100 products $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ using this compressed version of $\mathcal{D}$, which in this case is plentiful to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ up to a precision of 10^{-6}. From Fig. 2, it can be seen that for large tensors, the time required for the construction of the GGN Hessian approximation $\hat{\mathbf{H}}$ grows at a manageable pace for increasing tensor dimensions. However, the computation time of the pseudoinverse $\hat{\mathbf{H}}^{\dagger}$ grows quickly when the tensor dimensions are increased, making the exact GGN method infeasible for large tensors. Consequently, for such tensors, it is necessary to switch to the inexact GGN method, wherein the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ is solved using the CG/MINRES method.

The full Hessian approximation $\hat{\mathbf{H}}$ could be used to perform products of the form $\hat{\mathbf{H}}\mathbf{x}$ during the CG/MINRES iterations. The mean computation time of 100 such products is also shown in Fig. 2 for comparison. It can be seen that the computation time of $\hat{\mathbf{H}}\mathbf{x}$ increases steadily when the tensor dimensions are increased. In contrast, the computation time of the product $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ stays relatively constant when the Hessian approximation is not formed in full, using the approach from Sec. II. Evidently, the one-time costs for the formation of $\hat{\mathbf{H}}$ and for the compression of $\mathcal{D}$ should also be taken into account, with the latter remaining comparatively lower as the tensor dimensions are increased. Thus, for large tensors, the CG/MINRES iterations can be more efficiently performed by using the GGN Hessian approximation implicitly.

For a full CPD fitting problem, the proposed inexact GGN method is compared to the exact GGN method and the generalized CPD (GCPD) method from [10] in Fig. 3. Rank-5 tensors with dimensions $I \times I \times I$ are generated with factor matrix entries sampled in $U(0, 1)$ and the tensors are perturbed with multiplicative Gamma noise using unit shape and scale and an SNR of $40\,\mathrm{dB}$. Both methods are initialized by sampling random factor matrices with entries in $U(0, 1)$ and performing five iterations of the GN LS algorithm. For the inexact GGN method, the $\mathcal{D}$-tensor is compressed to a rank-$(I/10, I/10, I/10)$ tensor. The case $I = 200$ is omitted for the exact GGN method, as the use of the full $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$-matrix caused out-of-memory problems on the available hardware. In the center plot of Fig. 3, it is shown that, regardless of the tensor dimensions, all methods find the same solution for the CPD. From the left plot of Fig. 3, one can see that the computational costs of the GCPD and exact GGN methods grow quickly with increasing tensor dimensions. In contrast, the computation time of the inexact GGN method grows much more slowly, so that, for large tensors, this method is much more computationally efficient. For these large tensors, objective function and gradient evaluations are relatively more expensive than for smaller tensors. As a result, it is critical to limit the number of iterations of the optimization algorithm. From the right plot of Fig. 3, one can see that the GGN methods require much fewer iterations than the GCPD method. Combined with the faster iterations of the inexact GGN method compared to the exact GGN method, this explains why the inexact GGN method scales much better than the other two methods.

D. Separating fMRI sources

Following the methodology of [33], we use a CPD to perform blind fMRI source unmixing to recover nine simulated spatio-temporal sources. The spatial and temporal modes of these sources are shown in Fig. 4. The spatial signatures have dimensions 60×60 and are vectorized into vectors of length 3600. Over time, 100 different samples are collected. Four subjects are considered that each generate the same sources, but with different amplitudes depending on the person. After mixing, a $3600 \times 100 \times 4$-tensor $\mathcal{T}$ is obtained.

In practice, the real and imaginary parts of fMRI data are generally obtained separately and both are independently

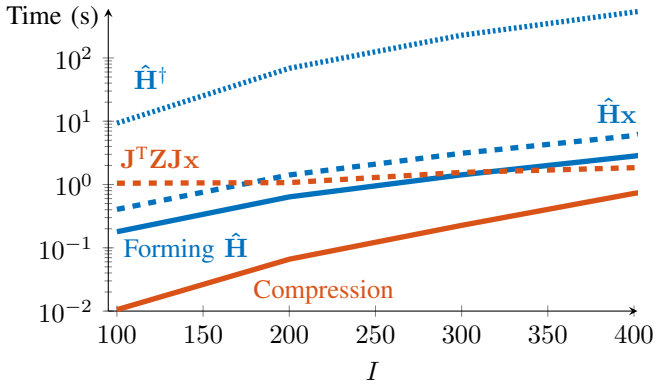


Fig. 2. The inexact GGN method with compression is faster than the exact GGN method for large tensors. Mean time (over 20 trials) for the compression of $\mathcal{D}$ and for 100 evaluations of $\mathbf{J}^\mathrm{T}\mathbf{Z}\mathbf{J}\mathbf{x}$ after compression. For comparison, the mean time for the construction of the Hessian approximation $\hat{\mathbf{H}} = \mathbf{J}^\mathrm{T}\mathbf{Z}\mathbf{J}$, for the computation of the pseudoinverse $\hat{\mathbf{H}}^\dagger$ and for 100 products $\hat{\mathbf{H}}\mathbf{x}$ are also shown for the exact GGN method. The tensors have dimensions $I \times I \times I$ and rank 10. The CPD model has rank 10 as well. A rank-$(I/10, I/10, I/10)$ approximation is used for $\mathcal{D}$.

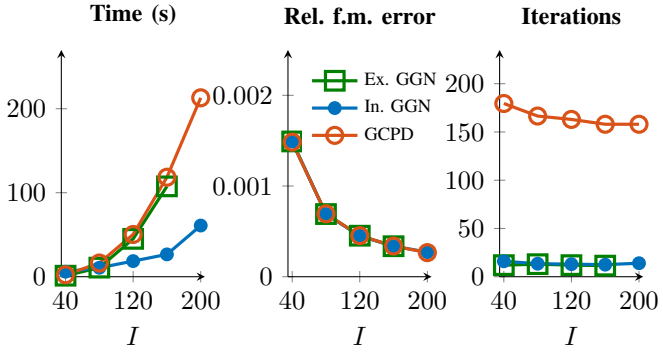


Fig. 3. The inexact GGN method with compression scales much better than the exact GGN method and the GCPD method for increasing tensor dimensions, while finding the same solution. Median (over 20 trials) time (left), relative factor matrix error (center) and number of iterations (right) for the computation of a CPD with a β-divergence cost function with $\beta = 1.5$. The tensors have dimensions $I \times I \times I$ and rank 5. The CPD model has rank 5 as well. A rank-$(I/10, I/10, I/10)$ approximation was used for $\hat{\mathcal{D}}$.

perturbed by additive Gaussian noise with equal variance during data collection. As explained in Sec. III, the magnitude of the fMRI data, which is used for analysis, follows a Rician distribution. One can thus expect that the use of the cost function from Sec. III gives better results than the LS cost function for this case. Hereafter, this cost function will be called the Rician-dedicated cost function.

Additive Gaussian noise with variance σ is added to both the real and imaginary parts of the tensor $\mathcal{T}$ such that the SNR of its real part is 25 dB. CPDs are then found for the tensor of moduli $|\mathcal{T}|$ using two different cost functions: the LS cost function, and the Rician-dedicated cost function. In Figures 5 and 6, the unmixed sources are shown for these two cost functions, respectively. They have been reordered to match the ordering of the original sources. Aside from some small artifacts leaking through from other sources, most sources are found with relatively high accuracy by both methods. Visually, the results from both methods cannot be distinguished. Comparing the mean correlations of the unmixed sources with the true sources for the two cost functions, one also finds both to be very similar: 0.9873 for the LS cost function, and 0.9875 for the Rician-dedicated cost function. In Table IV, the peak

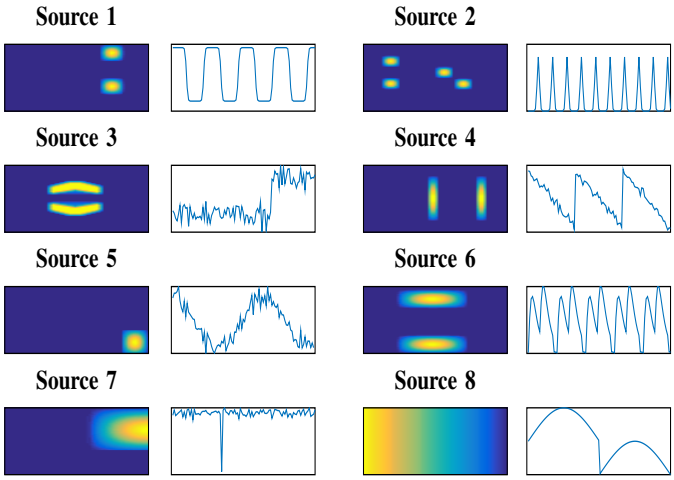


Fig. 4. Spatial and temporal modes of nine simulated fMRI sources.

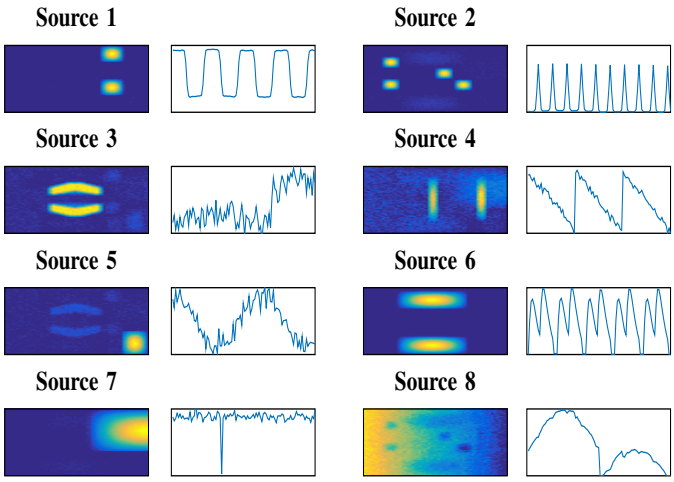


Fig. 5. Unmixed fMRI sources using a CPD with LS cost function.

SNR (PSNR) and structural similarity index (SSI) [34] of the spatial components of the 8 recovered sources are displayed. The PSNR of two images $\mathbf{X}, \mathbf{Y} \in [0,1]^{I \times J}$ is defined as

$$\mathrm{PSNR}(\mathbf{X}, \mathbf{Y}) = 10 \log_{10} \left(\frac{\rho^2 IJ}{\|\mathbf{X} - \mathbf{Y}\|_\mathrm{F}^2} \right),$$

with ρ the maximal value of the image. The SSI of $\mathbf{X}$ and $\mathbf{Y}$ is defined as

$$\mathrm{SSI}(\mathbf{X}, \mathbf{Y}) = \frac{(2\mu_\mathbf{X}\mu_\mathbf{Y} + C_1)(2\sigma_{\mathbf{XY}} + C_2)}{(\mu_\mathbf{X}^2 + \mu_\mathbf{Y}^2 + C_1)(\sigma_\mathbf{X}^2 + \sigma_\mathbf{Y}^2 + C_2)},$$

wherein $\mu_\mathbf{X}$, $\mu_\mathbf{Y}$, $\sigma_\mathbf{X}$, $\sigma_\mathbf{Y}$, and $\sigma_{\mathbf{XY}}$ are the local means, standard deviations, and cross-covariance, respectively. The constants C_1 and C_2 are set to the default values 0.0001 and 0.0009, respectively. The SSI takes the differences in luminance, contrast and structure of both images into account. It lies between 0 and 1, with 1 indicating a perfect match. For both metrics, it can be seen in Table IV that the results for the Rician-dedicated cost function are indeed somewhat better than those for the LS cost function.

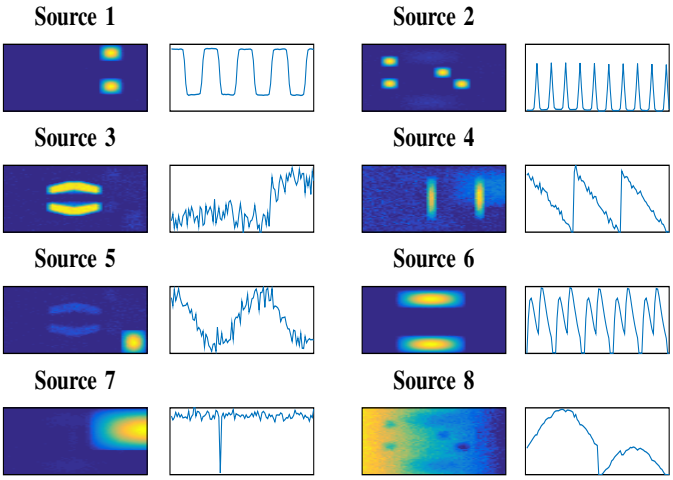


Fig. 6. Unmixed fMRI sources using a CPD with Rician-dedicated cost function.

Table IV
THE RECOVERED SPATIAL SOURCES ARE SLIGHTLY MORE ACCURATE WITH THE RICIAN-DEDICATED COST FUNCTION COMPARED TO THE LS COST FUNCTION. PSNRS AND SSIS OF THE EIGHT SPATIAL SOURCES RECOVERED USING A CPD WITH AN LS AND WITH A RICIAN-DEDICATED COST FUNCTION, TOGETHER WITH THEIR MEAN VALUES.

	Source								
PSNR	1	2	3	4	5	6	7	8	Mean
LS	64.82	53.07	54.64	42.00	48.78	74.14	66.65	53.32	57.18
Rice	67.26	54.36	59.10	42.14	50.19	74.40	64.81	53.22	58.19
SSI									
LS	0.9982	0.9651	0.9777	0.7749	0.9306	0.9998	0.9985	0.9919	0.9545
Rice	0.9992	0.9737	0.9940	0.8009	0.9530	0.9999	0.9975	0.9906	0.9636

V. CONCLUSION

An inexact GGN method was proposed to fit the CPD of a tensor for non-LS entry-wise cost functions. Rician distributed data is used as an example where the application of an alternative cost function can lead to more meaningful results. The proposed inexact GGN method obtains the new step direction $\mathbf{p}$ by solving the system $\hat{\mathbf{H}}\mathbf{p} = -\mathbf{g}$ iteratively. A considerable speedup is achieved by applying a specialized preconditioner and by compressing the information in the Hessian approximation $\hat{\mathbf{H}}$ with an LMLRA. The inexact GGN method scales much better than the exact GGN method and other existing methods without compromising the fast convergence that the use of second-order information yields. The performance of the method is demonstrated on synthetic data and on a simulated biomedical experiment.

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

[4] 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 Anal. Appl.*, vol. 34, no. 3, pp. 876–903, 2013.

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

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

[7] 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.

[8] N. Vervliet and L. De Lathauwer, "Numerical optimization based algorithms for data fusion," in *Data Fusion Methodology and Applications* (M. Cocchi, ed.), vol. 31, pp. 81–128, Elsevier, 2019.

[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] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[14] M. Vandecappelle, N. Vervliet, and Lieven De Lathauwer, "A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost." Tech. report 19–5, ESAT-STADIUS, KU Leuven (Leuven, Belgium), 11 pages, submitted to IEEE Transactions on Signal Processing (T-SP-26191-2020, minor revision), 2019.

[15] 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.

[16] 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, May 2009.

[17] W. Liu, P. P. Pokharel, and J. C. Principe, "Correntropy: Properties and applications in non-Gaussian signal processing," *IEEE Trans. Signal Process.*, vol. 55, pp. 5286–5298, Nov 2007.

[18] 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.

[19] 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.

[20] T. Steihaug, "The conjugate gradient method and trust regions in large scale optimization," *SIAM J. Numer. Anal.*, vol. 20, no. 3, pp. 626–637, 1983.

[21] C. Paige and M. Saunders, "Solution of sparse indefinite systems of linear equations," *SIAM J. Numer. Anal.*, vol. 12, no. 4, pp. 617–629, 1975.

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

[23] N. Vervliet, O. Debals, and L. De Lathauwer, "Tensorlab 3.0 — Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization," in *Proceedings of the 50th Asilomar Conference on Signals, Systems and Computers (ASILOMAR 2016)*, (Pacific Grove, CA, USA), pp. 1733–1738, Nov. 2016.

[24] Y. Xu, "Fast algorithms for higher-order singular value decomposition from incomplete data," *J. Comput. Math.*, vol. 35, no. 4, pp. 397–422, 2017.

[25] C. F. Caiafa and A. Cichocki, "Generalizing the column–row matrix decomposition to multi-way arrays," *Linear Algebr. Appl.*, vol. 433, no. 3, pp. 557–573, 2010.

[26] M. Sørensen and Lieven De Lathauwer, "A multilinear singular value decomposition method for a tensor that has missing fibers." Tech. report 20–53, ESAT-STADIUS, KU Leuven (Leuven, Belgium), 2020.

[27] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, "Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis," *IEEE Signal Process. Mag.*, vol. 31, pp. 71–79, 9 2014.
[28] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU press, 2012.
[29] A. den Dekker and J. Sijbers, "Implications of the Rician distribution for fMRI generalized likelihood ratio tests," *Magn. Reson. Imaging*, vol. 23, no. 9, pp. 953 – 959, 2005.
[30] K. K. Talukdar and W. D. Lawing, "Estimation of the parameters of the Rice distribution," *J. Acoust. Soc. Am.*, vol. 89, no. 3, pp. 1193–1197, 1991.
[31] J. Sijbers, A. den Dekker, P. Scheunders, and D. Van Dyck, "Maximum-likelihood estimation of Rician distribution parameters," *IEEE Trans. Med. Imaging*, vol. 17, no. 3, pp. 357–361, 1998.
[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] C. Chatzichristos, M. Vandecappelle, E. Kofidis, S. Theodoridis, L. De Lathauwer, and S. Van Huffel, "Tensor-based blind fMRI source separation without the Gaussian noise assumption—a β-divergence approach," in *Proc. GlobalSIP 2019*, (Ottawa, Canada), Nov 2019.
[34] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, 2004.