Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost functions

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. While it is possible to construct the full Hessian approximation for small tensors, the exact GGN approach becomes too expensive for tensors with larger dimensions, as found in typical applications. In this paper, we therefore propose to use an inexact GGN method and provide several strategies to make this method scalable to large tensors. First, the approximation of the Hessian is only used implicitly and its multilinear structure is exploited during Hessian-vector products, 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. We also propose dedicated preconditioners for the problem. 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. 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. Finally, we show that the method can scale by using randomized block sampling.**

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, e.g., [5]—several optimization-based methods have been developed to compute the CPD of a given tensor, as described in, e.g., [6]–[8]. These methods generally assume that the rank of the CPD is known in advance. Rank estimates typically follow from domain knowledge or experimentation. Alternatively, automatic rank determination methods can find the appropriate rank [9]–[12].

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 [13]. However, these assumptions do not always hold and in specific settings better alternatives for the cost function exist. These alternative cost functions allow other statistical assumptions, resulting in new methods that can compute a CPD of a tensor with a range of non-LS cost functions. Non-LS cost functions are particularly useful when working with nonnegative data, as more meaningful results can usually be achieved with cost functions that are specifically tailored to such data. For example, the Kullback–Leibler (KL) and Itakura–Saito (IS) divergence cost functions—β-divergence [14] cost functions with $\beta = 1$ and $\beta = 0$, respectively—have been shown to give the maximum likelihood estimation under the assumption of Poisson-distributed entries or multiplicative Gamma noise, respectively [15]. The former typically arises when analyzing count data [16], while the latter occurs in audio [15] and image analysis [17],

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) and by Artificiële intelligentie (AI) Vlaanderen (3E190661); (2) KU Leuven Internal Funds C16/15/059 and ID-N project no 3E190402; (3) Leuven Institute for Artificial Intelligence (Leuven.ai).

*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.

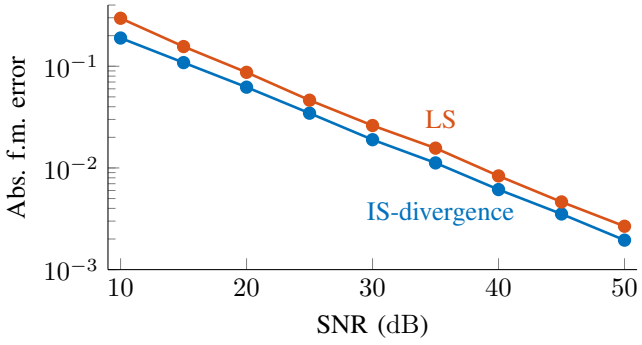

Fig. 1. When a tensor is perturbed by multiplicative Gamma noise, fitting a CPD with the IS-divergence cost function consistently leads to a better solution than when the LS cost function is used. Median absolute factor matrix errors are shown over 50 trials after matching and normalizing columns. The tensors have dimensions $20 \times 20 \times 20$ and the unperturbed tensors and the CPDs both have rank 5.

for instance in synthetic aperture radar images perturbed by speckle noise [18]. In Fig. 1, it is demonstrated that for tensors perturbed by multiplicative Gamma noise, the CPD obtained with the IS-divergence cost function consistently provides a better fit than the CPD obtained with the LS cost function. As another example, when fitting Rician distributed data, the maximum likelihood estimate of the underlying parameters is obtained using a particular non-LS cost function [19]. Rician distributed data arises, for instance, when the real and imaginary parts of a complex signal are captured independently and equal-variance additive noise is present on both parts, which is typical in medical applications such as MRI analysis [20].

In [21]–[28] first-order, gradient-based methods to compute a CPD with general cost functions are given. To improve the convergence speed of the algorithms, second-order information can be used as well [29]–[31]. Second-order methods find the next step direction $\mathbf{p}$ by solving the system $\mathbf{Hp} = -\mathbf{g}$, wherein $\mathbf{H}$ is (a good approximation of) the Hessian matrix and $\mathbf{g}$ is the gradient. 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 their lower number of iterations. For non-LS cost functions, cost function and gradient evaluations are generally much more expensive in a relative sense than for the LS cost function, which makes it easier for second-order methods to be competitive. In [32], a second-order method is proposed in the form of a generalized Gauss–Newton (GGN) method. Instead of the Gauss–Newton (GN) Hessian approximation $\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}}\mathbf{J}$, which only depends on the Jacobian matrix $\mathbf{J}$, an approximation of the form $\mathbf{J}^{\mathrm{T}}\mathbf{ZJ}$ is used, wherein the diagonal matrix $\mathbf{Z}$ depends on the specific cost function that is used. For the CPD, the GGN Hessian approximation lies very close to the true Hessian. At the same time, it is highly structured, which can, and should, be exploited to minimize the per-iteration cost. Consequently, the GGN method often outperforms first-order methods with respect to computation time [32]. Unfortunately, this method requires the construction and inversion of a full approximate Hessian, which take $\mathcal{O}(I^3R^2)$ and $\mathcal{O}(I^3R^3)$ flop, respectively, for an $I \times I \times I$ tensor of rank R. Although this cost is manageable for small to medium sized tensors, this is no longer the case for larger tensors.

To improve the scalability of second-order methods for non-LS CPD fitting, we propose to use an inexact GGN method. Instead of constructing a full Hessian approximation, its information is accessed implicitly and an iterative solver such as (preconditioned) conjugate gradient (CG) [8], [33] or MINRES [34] is used to find the next step direction $\mathbf{p}$. As a result, the computational cost of the GGN iterations is significantly reduced, which makes the method applicable for larger tensors than those that can be handled by the exact GGN method. Still, a naive implementation of the inexact GGN method is too costly for larger tensors. To make the inexact GGN approach scalable, we propose several strategies to reduce both the per-iteration cost and the number of CG/MINRES iterations. The properties of the proposed method are illustrated in detail for a non-LS cost function which is specifically tailored for Rician distributed data [19].

The main contributions in this paper are threefold. First, to limit the computational cost of solving the system $\mathbf{J}^{\mathrm{T}}\mathbf{ZJp} = -\mathbf{g}$ iteratively via CG/MINRES iterations, we carefully exploit the structure of $\mathbf{J}$ to significantly lower the cost of products of the form $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$. Second, we reduce the cost of the CG/MINRES iterations even further by approximating the second-order information with a low multilinear rank approximation (LMLRA) [35]. Note that the LMLRA is applied in a way that is fundamentally different from the standard multilinear compression CANDELINC [36]. While methods exist that approximate the entire Hessian approximation matrix $\hat{\mathbf{H}}$ [31], which in our case is equal to $\mathbf{J}^{\mathrm{T}}\mathbf{ZJ}$, we are—at least to our knowledge—the first to approximate only the $\mathbf{Z}$-matrix holding the second-order information. In this way, the cost of each CG/MINRES iteration is further decreased, while all first-order information in $\mathbf{J}$ is maintained and its structure can still be leveraged in the CG/MINRES computations. Finally, we tailor general-purpose preconditioners into dedicated preconditioners for this problem. Due to the exploitation of the specific structure of these preconditioners, they are much cheaper to apply than generic preconditioners, while they can significantly decrease the number of CG/MINRES iterations that are required to solve $\mathbf{J}^{\mathrm{T}}\mathbf{ZJp} = -\mathbf{g}$. Together, these contributions make the inexact GGN approach applicable to larger tensors: they allow fast non-stochastic computation of the CPD with arbitrary twice-differentiable cost functions when the construction of the full Hessian approximation is prohibitively expensive. As the proposed techniques are independent of the specific solver that is used, they can be applied in other second-order algorithms to increase their scalability as well. We illustrate this by applying the techniques to a randomized block sampling (RBS) [37] CPD algorithm. This method finds the CPD in a stochastic way by repeatedly sampling small blocks from the tensor. Combining RBS with our proposed techniques enables it to work with non-LS cost functions as well, opening up a whole new class of applications.

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; see supplemental material.

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 others 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, 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 $M \times M$-identity matrix is written as $\mathbf{I}_M$ and the Moore–Penrose pseudoinverse of a matrix $\mathbf{X}$ is written as $\mathbf{X}^{\dagger}$. 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 $\mathbf{\Pi}^{(1)} = \mathbf{I}_{IJK}$ 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 $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{I\times J}$ 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

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

respectively. Finally, an LMLRA [1], [2], [35] of a tensor $\mathcal{T} \in \mathbb{R}^{I_1\times I_2\times I_3}$ is defined as

$$\mathcal{T} \approx \mathcal{S} \cdot_1 \mathbf{U} \cdot_2 \mathbf{V} \cdot_3 \mathbf{W} =: [\![\mathcal{S}; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!],$$

with $\mathcal{S} \in \mathbb{R}^{R_1\times R_2\times R_3}$, $\mathbf{U} \in \mathbb{R}^{I_1\times R_1}$, $\mathbf{V} \in \mathbb{R}^{I_2\times R_2}$, $\mathbf{W} \in \mathbb{R}^{I_3\times R_3}$ and $R_1 \leq I_1$, $R_2 \leq I_2$, $R_3 \leq I_3$. 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 TENSORS WITH GENERAL COST FUNCTIONS

Optimization methods that use second-order information generally perform well at fitting the CPD of a tensor [1], [2], [7], [8], [38]–[41]. Although the per-iteration cost of second-order methods is higher than that of first-order methods, the asymptotic convergence behavior of the former is superlinear (or even quadratic), compared to linear for the latter [29]. In practice, one generally observes a significant reduction in the number of iterations required to obtain the CPD compared to first-order methods, even before the asymptotic behavior kicks in [8], [32]. Unfortunately, the per-iteration computational cost of second-order methods can become prohibitively high for larger tensors. Consequently, strategies have been developed to decrease this cost. For example, for the GN algorithm, exploiting the multilinear structure of the CPD can drastically reduce the per-iteration cost [7]. As a result, the method can leverage its lower number of iterations compared to first-order methods to obtain the solution faster [7], [8].

Reducing the number of iterations is even more important for non-LS cost functions, as their cost function and gradient evaluations are relatively more costly [32]. For example, for β-divergences, the entry-wise cost functions require the evaluation of logarithms and non-integer powers, while for the Rician cost function, they require the evaluation of modified Bessel functions. Both are much more expensive than the quadratic error of the LS cost function. Additionally, the standard tricks to avoid computing all tensor entries that are used in the LS case no longer work for non-LS cost functions. For example, first computing a multilinear rank-(R,R,R) LMLRA of a tensor, then computing a rank-R CPD of the $R \times R \times R$ core and finally expanding the factors to the original tensor dimensions [42] only works for the LS distance. Consequently, compared to the LS cost function, even first-order methods are much more expensive for large tensors.

The CPD GN algorithm for the LS cost function has been generalized to arbitrary twice differentiable cost functions in [32]. This GGN method requires 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 approximate Hessian 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 given 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 $\mathbf{z} = [\text{vec}(\mathbf{A}); \text{vec}(\mathbf{B}); \text{vec}(\mathbf{C})] \in \mathbb{R}^{(I+J+K)R}$ holding all optimization variables, the CPD can thus be obtained by solving $\min_{\mathbf{z}} f(\mathbf{z})$, with

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

For convenience, we replace the triple sum over all tensor entries by the single sum $\sum_{n=1}^{IJK}$. As discussed in [32], the LS cost function for the CPD can be generalized to

$$f(\mathbf{z}) = \sum_{n=1}^{IJK} f_n(m_n(\mathbf{z})), \tag{2}$$

with $m_n(\mathbf{z})$ mapping the optimization variables in $\mathbf{z}$ to the tensor elements in the current CPD estimate $\mathcal{M}$ of the optimization algorithm. Each f_n is a twice differentiable fit function for the corresponding model entry m_n. Aside from the LS fit function $f_n = (m_n - t_n)^2$, other examples of cost functions are given by β-divergences [14]:

$$f_n = \frac{1}{\beta(\beta-1)}\left(t_n^{\beta} + (\beta-1)m_n^{\beta} - \beta\left(t_n m_n^{(\beta-1)}\right)\right),$$

with $\beta = 1$ (KL-divergence) and $\beta = 0$ (IS-divergence) as special cases:

$$f_n = t_n \log\left(\frac{t_n}{m_n}\right) - t_n + m_n, \quad \beta = 1,$$
$$f_n = \left[\frac{t_n}{m_n} - \log\left(\frac{t_n}{m_n}\right) - 1\right], \quad \beta = 0.$$

The correntropy function [43] is another alternative, as is the maximum likelihood estimator for data that follows a Rician distribution [19]; see (9) in Sec. III. For a more extensive overview of non-LS cost functions, see, for instance, [21].

Gradient descent methods use the direction of the negative gradient to obtain a new intermediate solution $\mathbf{z}_{k+1} = \mathbf{z}_k + \alpha\mathbf{p}$. They thus require only the gradient $\mathbf{g} = \nabla_{\mathbf{z}} f$ of (2) to be computed. In contrast, second-order optimization methods find the next model estimate by solving the system $\mathbf{H}\mathbf{p} = -\mathbf{g}$ for the step direction $\mathbf{p}$, where $\mathbf{H} = \nabla^2 f \in \mathbb{R}^{(I+J+K)R\times(I+J+K)R}$ is the Hessian matrix of f. By also using information from the Hessian, one typically obtains a better local approximation of the cost function and therefore better steps are taken and fewer iterations are required to reach the solution. As the computation of the full Hessian is usually prohibitively expensive, an approximate Hessian $\hat{\mathbf{H}}$ is often used in practice. The new step direction $\mathbf{p}$ is then obtained from the approximate 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 [29] to guarantee global convergence of the method to a stationary point. With the trust region, 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 Hessian are derived. The gradient $\mathbf{g}$ can be split into terms according to the factor matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$: $\mathbf{g} = [\mathbf{g}_{\mathbf{A}}; \mathbf{g}_{\mathbf{B}}; \mathbf{g}_{\mathbf{C}}]$. By exploiting the multilinear structure, the subgradients $\mathbf{g}_{\mathbf{A}}$, $\mathbf{g}_{\mathbf{B}}$ and $\mathbf{g}_{\mathbf{C}}$ can be computed as

$$\mathbf{g}_{\mathbf{A}} = \text{vec}\left(\mathbf{R}_{(1)}(\mathbf{C}\odot\mathbf{B})\right),$$
$$\mathbf{g}_{\mathbf{B}} = \text{vec}\left(\mathbf{R}_{(2)}(\mathbf{C}\odot\mathbf{A})\right),$$
$$\mathbf{g}_{\mathbf{C}} = \text{vec}\left(\mathbf{R}_{(3)}(\mathbf{B}\odot\mathbf{A})\right),$$

where the entries of the tensor $\mathcal{R} \in \mathbb{R}^{I\times J\times K}$ are defined as:

$$r_n = \frac{\partial f_n}{\partial m_n},$$

for $1 \le n \le IJK$. For example, for the LS cost function, $\mathcal{R}$ is simply the residual tensor $\mathcal{M} - \mathcal{T}$. The subgradients $\mathbf{g}_{\mathbf{A}}$, $\mathbf{g}_{\mathbf{B}}$ and $\mathbf{g}_{\mathbf{C}}$ all have the same structure: they are products of an unfolded tensor and a Khatri–Rao product of matrices. This matricized tensor times Khatri–Rao product (`MTKRPROD`) is well-studied and can be computed efficiently using specialized routines, either sequentially or in parallel [8], [44]–[46].

For the CPD fitting problem with general cost functions (2), the Hessian matrix $\mathbf{H}$ is defined as

$$\mathbf{H} = \sum_{n=1}^{IJK} \mathbf{J}_{n:}^{\mathrm{T}} \frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2} \mathbf{J}_{n:} + \frac{\mathrm{d}f_n}{\mathrm{d}m_n}\left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2}\right). \tag{3}$$

The first term in (3) involves the Jacobian matrix $\mathbf{J} \in \mathbb{R}^{IJK\times(I+J+K)R}$. The nth row of $\mathbf{J}$ holds all first order partial derivatives of $f_n(\mathbf{z})$. Using the permutation matrices $\mathbf{\Pi}^{(i)}$, as defined in Sec. I-A, $\mathbf{J}$ is written as $[\mathbf{J}_{\mathbf{A}}; \mathbf{J}_{\mathbf{B}}; \mathbf{J}_{\mathbf{C}}]$, with

$$\mathbf{J}_{\mathbf{A}} = (\mathbf{C}\odot\mathbf{B})\otimes\mathbf{I}_I,$$
$$\mathbf{J}_{\mathbf{B}} = \mathbf{\Pi}^{(2)^{\mathrm{T}}}\left((\mathbf{C}\odot\mathbf{A})\otimes\mathbf{I}_J\right),$$
$$\mathbf{J}_{\mathbf{C}} = \mathbf{\Pi}^{(3)^{\mathrm{T}}}\left((\mathbf{B}\odot\mathbf{A})\otimes\mathbf{I}_K\right).$$

The second term in (3) consists of two factors. The factor $\frac{\mathrm{d}f_n}{\mathrm{d}m_n}$, i.e., the change in cost function, becomes very small as we move closer to the (global) optimum. The other factor, $\left(\frac{\partial^2 m_n}{\partial \mathbf{z}^2}\right)$, is sparse for the CPD, because each tensor entry only depends on a few model entries [7]. The generalized GGN method, which approximates the Hessian by only its first term

$$\hat{\mathbf{H}} = \sum_{n=1}^{IJK} \mathbf{J}_{n:}^{\mathrm{T}} \frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2} \mathbf{J}_{n:},$$

can thus be expected to perform well for fitting a CPD [8], [47], [48]. The tensor $\mathcal{D} \in \mathbb{R}^{I\times J\times K}$, with

$$d_n = \frac{\mathrm{d}^2 f_n}{\mathrm{d}{m_n}^2},$$

can be constructed to write $\hat{\mathbf{H}}$ as follows:

$$\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}}\text{Diag}\left(\text{vec}(\mathcal{D})\right)\mathbf{J}.$$

Finally, by defining

$$\mathbf{Z} = \text{Diag}\left(\text{vec}(\mathcal{D})\right) \in \mathbb{R}^{IJK\times IJK},$$

the GGN Hessian approximation can be written more compactly as:

$$\hat{\mathbf{H}} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}.$$

For the LS cost function, $\mathbf{Z} = \mathbf{I}_{IJK}$ and the Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ 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], [32].

Computing the full Hessian approximation $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ can become expensive as the tensor dimensions are increased, as $\mathbf{J}$ is of dimensions $IJK \times (I + J + K)R$ and $\mathbf{Z}$ holds IJK non-zero entries on its diagonal. Fortunately, its explicit computation 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) CG [8], [33] or MINRES [34] 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 need to efficiently evaluate the product of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ with an arbitrary vector $\mathbf{x}$, where

$$\mathbf{x} = [\mathrm{vec}(\mathbf{X_A}); \mathrm{vec}(\mathbf{X_B}); \mathrm{vec}(\mathbf{X_C})] = [\mathbf{x_A}; \mathbf{x_B}; \mathbf{x_C}]$$

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 the computational complexity of CG/MINRES is dominated by the complexity of the $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$-products, 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 is replaced by a cost of $\mathcal{O}(kl)$ flop. Herein, k is the number of iterations of the inexact solver (50 to 100 typically) and l is its per-iteration cost, i.e., the cost of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$. Below, we provide three strategies to reduce both k and l. Together, these strategies make the inexact GGN method applicable to much larger tensors than when a naive implementation is used. In Sec. II-B, we explain how the structure of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ can be exploited to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ efficiently using the CG/MINRES algorithm. Next, we approximate the tensor $\mathcal{D}$ with an LMLRA in Sec. II-C to further reduce the cost of each CG/MINRES iteration without sacrificing the structure of $\mathbf{J}$. Finally, in Sec. II-D, we provide dedicated preconditioners for the computation of a CPD with general cost functions. These preconditioners reduce the number of CG/MINRES iterations that are necessary to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{p} = -\mathbf{g}$ up to a certain accuracy and by leveraging the structure of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$, they can be applied cheaply.

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

$$\begin{aligned}\mathbf{y_A} &= \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_A}\mathbf{x_A} + \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_B}\mathbf{x_B} + \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\mathbf{J_C}\mathbf{x_C} \\ &= \mathbf{J_A}^{\mathrm{T}}\mathbf{Z}\left(\mathbf{J_A}\mathbf{x_A} + \mathbf{J_B}\mathbf{x_B} + \mathbf{J_B}\mathbf{x_C}\right), \end{aligned} \tag{4}$$

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

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

Herein, the identity

$$(\mathbf{U}\otimes\mathbf{V})\mathrm{vec}(\mathbf{W}) = \mathrm{vec}\left(\mathbf{V}\mathbf{W}\mathbf{U}^{\mathrm{T}}\right) \tag{5}$$

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 (4) 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 $\mathbf{y_A}$:

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

Writing $\mathcal{Q} = \mathcal{D} * \mathcal{E}$ and applying (5) one more time, yields

$$\mathbf{y_A} = \mathrm{vec}\left(\mathbf{Q}_{(1)}(\mathbf{C}\odot\mathbf{B})\right). \tag{6}$$

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

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

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 $\mathcal{D}$

Within a single GGN iteration, the new 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 $\mathbf{y} = \mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ using CG/MINRES. Only the vector $\mathbf{x}$ varies between the different CG/MINRES evaluations. Consequently, only the tensor $\mathcal{E}$ changes when $\mathbf{y}$ is computed as in (6), 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 $\mathcal{D}$ and $\mathcal{E}$. As the same $\mathcal{D}$ is used throughout the execution of the full CG or MINRES algorithm, 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

$$\mathbf{y_A} = \mathrm{vec}\left(\hat{\mathbf{Q}}_{(1)}(\mathbf{C}\odot\mathbf{B})\right),$$

can also be computed much more efficiently by exploiting the structure of $\hat{\mathcal{Q}}$, as explained in the remainder of this section.

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 $\hat{\mathcal{Q}} = \hat{\mathcal{D}} * [\![\mathbf{E^A}, \mathbf{E^B}, \mathbf{E^C}]\!]$ is structured as well:

$$\begin{aligned}\hat{\mathcal{Q}} &= [\![S; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!] * [\![\mathbf{E^A}, \mathbf{E^B}, \mathbf{E^C}]\!] \\ &= \sum_{s=1}^{3R} [\![S; \mathbf{U}, \mathbf{V}, \mathbf{W}]\!] * [\![\mathbf{e}^{\mathbf{A}}_{:s}, \mathbf{e}^{\mathbf{B}}_{:s}, \mathbf{e}^{\mathbf{C}}_{:s}]\!] \\ &= \sum_{s=1}^{3R} [\![\mathcal{S}; \mathbf{E}^{\mathbf{A}}_s \mathbf{U}, \mathbf{E}^{\mathbf{B}}_s \mathbf{V}, \mathbf{E}^{\mathbf{C}}_s \mathbf{W}]\!], \end{aligned} \tag{7}$$

wherein $\mathbf{E}^{\mathbf{A}}_s = \text{Diag}\left(\mathbf{e}^{\mathbf{A}}_{:s}\right)$, $\mathbf{E}^{\mathbf{B}}_s = \text{Diag}\left(\mathbf{e}^{\mathbf{B}}_{:s}\right)$ and $\mathbf{E}^{\mathbf{C}}_s = \text{Diag}\left(\mathbf{e}^{\mathbf{C}}_{:s}\right)$. Because of the structure of $\mathbf{E^A}$, some matrices $\mathbf{E}^{\mathbf{A}}_s \mathbf{U}$ are duplicates. Naturally, these have to be computed only once. The same holds for $\mathbf{E}^{\mathbf{B}}_s \mathbf{V}$ and $\mathbf{E}^{\mathbf{C}}_s \mathbf{W}$. Using (7), $\mathbf{Y_A} = \text{unvec}\left(\mathbf{y_A}\right) = \text{unvec}\left(\hat{\mathbf{Q}}_{(1)} \left(\mathbf{C} \odot \mathbf{B}\right)\right)$ can be efficiently computed as

$$\begin{aligned}\mathbf{Y_A} &= \sum_{s=1}^{3R} (\mathbf{E}^{\mathbf{A}}_s \mathbf{U}) \mathbf{S}_{(1)} \left((\mathbf{E}^{\mathbf{C}}_s \mathbf{W}) \otimes (\mathbf{E}^{\mathbf{B}}_s \mathbf{V})\right)^{\mathrm{T}} (\mathbf{C} \odot \mathbf{B}) \\ &= \sum_{s=1}^{3R} (\mathbf{E}^{\mathbf{A}}_s \mathbf{U}) \mathbf{S}_{(1)} \left(\left[(\mathbf{E}^{\mathbf{C}}_s \mathbf{W})^{\mathrm{T}} \mathbf{C}\right] \odot \left[(\mathbf{E}^{\mathbf{B}}_s \mathbf{V})^{\mathrm{T}} \mathbf{B}\right]\right), \end{aligned} \tag{8}$$

with $\mathbf{E}^{\mathbf{A}}_s \mathbf{U} \in \mathbb{R}^{I \times R_1}$, $(\mathbf{E}^{\mathbf{C}}_s \mathbf{W})^{\mathrm{T}} \mathbf{C} \in \mathbb{R}^{R_3 \times R}$ and $(\mathbf{E}^{\mathbf{B}}_s \mathbf{V})^{\mathrm{T}} \mathbf{B} \in \mathbb{R}^{R_2 \times R}$. The vector $\mathbf{y_A}$ is obtained by vectorization of $\mathbf{Y_A}$. Compared to (6), (8) is again a `MTKRPROD`, but now involving smaller tensors and matrices. The same strategy can be applied in the other two modes to obtain $\mathbf{y_B} = \hat{\mathbf{Q}}_{(2)}(\mathbf{C} \odot \mathbf{A})$ and $\mathbf{y_C} = \hat{\mathbf{Q}}_{(3)}(\mathbf{B} \odot \mathbf{A})$, wherein terms of the form $\mathbf{E}^{\mathbf{A}}_s \mathbf{U}$ and $\left(\mathbf{E}^{\mathbf{B}}_s \mathbf{V}\right)^{\mathrm{T}} \mathbf{B}$ can be reused from the computation of $\mathbf{y_A}$.

Using specialized methods, e.g., a sequentially truncated multilinear singular value decompostion (ST-MLSVD) algorithm with randomized SVD computation [49], [50], a multilinear rank-$(R_{\text{ML}}, R_{\text{ML}}, R_{\text{ML}})$ 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 [51]–[54].

With a multilinear rank-$(R_{\text{ML}}, R_{\text{ML}}, R_{\text{ML}})$ LMLRA of $\mathcal{D}$, the complexity of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{x}$ 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 $\mathbf{y_A} = (\mathbf{E}^{\mathbf{A}}_r \mathbf{U})\mathbf{M}$ from (8). Again assuming $I = J = K$, these have a complexity of $\mathcal{O}(RR^3_{\text{ML}}))$ flop and $\mathcal{O}(RR_{\text{ML}}I)$ flop, respectively. This leads to a total complexity of $\mathcal{O}(R^2 R_{\text{ML}} \max(I, R^2_{\text{ML}}))$ flop for the entire $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{x}$-evaluation. Compared to the complexity of $\mathcal{O}(I^3 R)$ flop when using the full $\mathcal{D}$-tensor, we see that the approximation decreases the complexity of one evaluation of $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{x}$ whenever $RR_{\text{ML}} < I^2$ and $RR^3_{\text{ML}} < I^3$. Taking into account that in applications typically $I \gg R$, while by definition $I \geq R_{\text{ML}}$, 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 $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{x}$-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 GGN iterations, 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 $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p} = -\mathbf{g}$ is generally higher when the eigenvalues of the matrix $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ are well clustered. To improve the spectral properties of this matrix, a suitable preconditioner [55] can be used. A good preconditioner is cheap to apply and clusters the eigenvalues of $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ around one, which should reduce the required number of CG or MINRES iterations. Using the preconditioner $\mathbf{M}$, the modified system $\mathbf{M}^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{x} = -\mathbf{M}^{-1}\mathbf{g}$ is solved instead of the original system.

In iterative methods, the preconditioner $\mathbf{M} \in \mathbb{R}^{(I+J+K)R \times (I+J+K)R}$ is applied implicitly in every iteration. Instead of the multiplication of $\mathbf{M}^{-1}$ with the full $\mathbf{J}^{\mathrm{T}}\mathbf{J}$-matrix, only systems of the form $\mathbf{M}\mathbf{s} = \mathbf{t}$ need to be solved for $\mathbf{s} \in \mathbb{R}^{(I+J+K)R}$, with $\mathbf{t} \in \mathbb{R}^{(I+J+K)R}$ an arbitrary vector. A good preconditioner clusters the eigenvalues of $\mathbf{J}^{\mathrm{T}}\mathbf{J}$, which results in a reduction of the number of CG/MINRES iterations that are required to solve the system $\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{p} = -\mathbf{g}$ [29]. If the preconditioner is sufficiently cheap to apply, this also reduces the required time to solve the system. In order obtain a preconditioner $\mathbf{M}$ that can be applied efficiently, $\mathbf{M}$ 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. Both preconditioners are sparse and have exploitable higher-order structure which makes them much cheaper to apply than generic preconditioners. 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 $\mathbf{G}^{(n,n)} = \mathbf{J}_n^{\mathrm{T}}\mathbf{J}_n$ as its blocks, where $\mathbf{G}^{(1,1)} \in \mathbb{R}^{IR \times IR}$, $\mathbf{G}^{(2,2)} \in \mathbb{R}^{JR \times JR}$ and $\mathbf{G}^{(3,3)} \in \mathbb{R}^{KR \times KR}$. To solve $\mathbf{M}\mathbf{s} = \mathbf{t}$, one thus needs an efficient method to solve $\mathbf{G}^{(n,n)}\mathbf{s}^{(n)} = \mathbf{t}^{(n)}$. The method from [32] can be applied to obtain the highly structured diagonal blocks $\mathbf{G}^{(n,n)}$. Matrices $\mathbf{G}^{(n,n)}_{\text{nz}}$ holding only the nonzero entries of the $\mathbf{G}^{(n,n)}$-matrices are efficiently computed as

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

and the zeros are added afterwards. The construction of one diagonal block requires $\mathcal{O}(I^3 R^2)$ flop. If a multilinear 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 blocks to $\mathcal{O}(\max(I, R^2_{\text{ML}})R_{\text{ML}}R^2)$ flop.

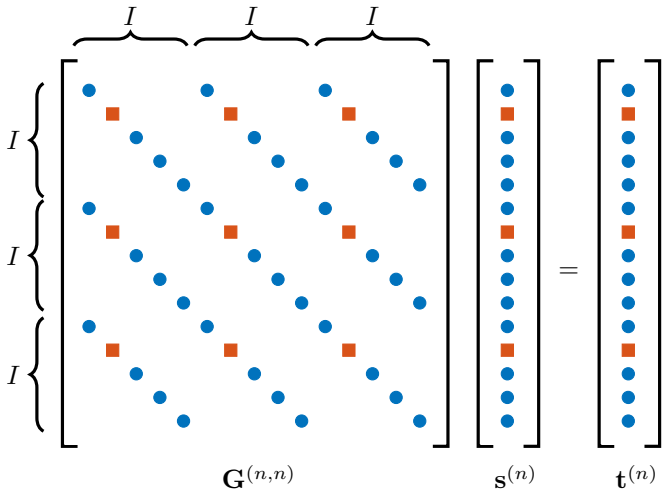

Fig. 2. Thanks to the structure of each $\mathbf{G}^{(n,n)}$-block, applying the block-Jacobi preconditioner involves solving I independent linear systems. To achieve an even lower computational cost, I independent LDL factorizations are computed such that only I back and forward substitution steps are required to apply the preconditioner. Entries corresponding to the second factorization are highlighted with red squares in the spy plot for $I = 5$ and $R = 3$.

After forming the diagonal blocks, applying the preconditioner, i.e., solving $\mathbf{Ms} = \mathbf{t}$, can be done efficiently by exploiting the particular structured sparsity of the blocks. This is done by first computing an LDL factorization [56] of $\mathbf{M}$, wherein the $\mathbf{L}$-matrix is lower triangular and the $\mathbf{D}$-matrix is diagonal, and then performing back substitution. For the proposed block-Jacobi preconditioner, an LDL factorization of $\mathbf{M}$ can be obtained by straightforwardly combining the LDL factorizations of the different $\mathbf{G}^{(n,n)}$-blocks, so we can focus on each block independently. The LDL factorization of a $\mathbf{G}^{(n,n)}$-block can be computed through I independent LDL factorizations of $R \times R$-matrices. These I independent LDL factorizations can then be combined into an LDL factorization of the entire $\mathbf{G}^{(n,n)}$-block. As an illustration, in Fig. 2, the nonzero stucture of a $\mathbf{G}^{(n,n)}$-block is shown for $I = 5$ and $R = 3$ and the entries handled by the second LDL factorization are highlighted. This dedicated approach for the CPD problem requires only $\mathcal{O}(IR^3)$ operations per diagonal block, which is much cheaper than sparse LDL factorization methods that ignore the structure and require at least $\mathcal{O}(I^2R^4)$ operations per block [57]. Additionally, there are no issues with unwanted fill-in—entries that are zero and should remain zero obtain a nonzero value due to unrelated operations and need to be restored—from which sparse LDL algorithms suffer. The back substitution then involves only R iterations per block, working on length-I vectors. The LDL factorization has to be computed only once and afterwards, the computed decomposition of $\mathbf{M}$ can be reused in every CG/MINRES iteration to solve $\mathbf{G}^{(n,n)}\mathbf{s}^{(n)} = \mathbf{t}^{(n)}$ via back substitution in $\mathcal{O}(IR^2)$ flop per block. In contrast, the $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$ product itself already requires $\mathcal{O}(I^3R)$ flop.

The block-Jacobi preconditioner reduces the computational cost if the lower 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 $\mathbf{J}^{\mathrm{T}}\mathbf{ZJ}$. 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 $\mathbf{M}$:

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

The system $\mathbf{Ms} = \mathbf{t}$ can then be solved by computing $\mathbf{s} = \frac{\mathbf{t}}{\mathbf{m}_{\text{diag}}}$. 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 $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$. 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 $\mathbf{J}^{\mathrm{T}}\mathbf{ZJx}$ evaluation to reduce the computation time. If a multilinear rank-$(R_{\mathrm{ML}}, R_{\mathrm{ML}}, R_{\mathrm{ML}})$ LMLRA of $\mathcal{D}$ is available, this reduces the complexity of computing $\mathbf{m}_{\text{diag}}$ to $\mathcal{O}(R_{\mathrm{ML}}R\max(I, R_{\mathrm{ML}}^2))$ 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 [32].

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 [58], 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:

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

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

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

Some methods to find the MLE and alternative estimators are given in [19], [59], [60]. The value of σ^2 can typically be obtained by averaging the squared magnitude over a region where the unperturbed measurement is zero [61]. For a tensor $\mathcal{T}$ with Rician distributed data and its CPD $\mathcal{M} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, one thus obtains the entry-wise cost function

$$f_n = \sum_{n=1}^{IJK} \frac{m_n^2}{2\sigma^2} - \log\left(I_0\left(\frac{m_n t_n}{\sigma^2}\right)\right). \tag{9}$$

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 $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}$ 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) [58]. 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 [58]. 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

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 standard first-order methods from the literature. We also apply our proposed techniques in the RBS framework to decompose a large tensor. 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 [62]. 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 $\bar{\mathcal{T}}$ as the perturbed version of the tensor $\mathcal{T}$, we can define the perturbation tensor itself as $\mathcal{N} = \bar{\mathcal{T}} - \mathcal{T}$. The signal to noise ratio (SNR) is then $20\log_{10}\left(\frac{\|\mathcal{T}\|_{\mathrm{F}}}{\|\mathcal{N}\|_{\mathrm{F}}}\right)$. 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 $10\log_{10}(\alpha)$.

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. 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 number of 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
$I = 50$, $R = 5$, 20 dB noise			
Median number of 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 number of 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
$I = 30$, $R = 3$, 20 dB noise			
Median number of 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

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.

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. 3, 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. 3, we can see that the computation of the product $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ becomes very costly for the larger 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. 3, 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 as the tensor dimensions are increased. 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 by multiplicative Gamma noise with an SNR of $40\,\mathrm{dB}$. 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 $S \times S \times S$. The algorithm stops when either a relative step size $\frac{\|\mathbf{p}\|}{\|\mathbf{z}\|}$ or a relative cost function change compared to the initial cost function value below 10^{-15} is encountered. Only successful runs are taken into account, where the algorithm converged to a solution with a function value $f < 10^{-4}I^3$ 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} and the Jacobi preconditioner is used. 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 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 perform the experiment from Table II for $I = 100$ with an adaptive compression factor. A compression factor of 25 is chosen and, if necessary, the multilinear rank is gradually increased by adding multilinear singular vectors in each mode until the sum of the squared multilinear singular values of $\hat{\mathcal{D}}$ sums to at least 99.9% of $\|\mathcal{D}\|_{\mathrm{F}}^2$. To have access to the extra multilinear singular vectors, an LMLRA of slightly higher multilinear rank is computed and the rank $(I/25 \times I/25 \times I/25)$-LMLRA is extracted from it. This leads to a mean time of $10\,\mathrm{s}$ and a mean number of GGN iterations of 43, while all 20 runs converge to the optimal solution (given the noise level). Strategically increasing the multilinear rank of the compression can thus improve the convergence behavior compared to fixing the compression factor at 25. In this experiment, the adaptive method performs even better than the best method from Table II, the one using a fixed compression factor of 5, without needing 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 GN 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 the largest tensors, the number of GGN iterations was only moderately affected by the compression and for $\beta = 1.5$ 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 $\beta = 0.5$, 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 $\beta = 0.5$.

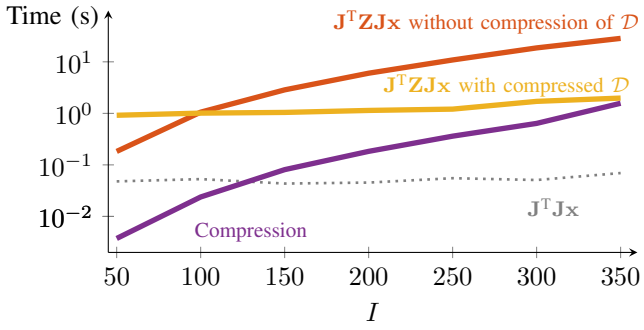

Fig. 3. When $\mathcal{D}$ is compressed, the computation time for an evaluation of $\mathbf{J}^T\mathbf{ZJx}$ (without the compression time) remains relatively constant for increasing I, while it increases exponentially without compression. Mean time (over 20 trials) for 100 evaluations of $\mathbf{J}^T\mathbf{ZJx}$ with the same $\mathbf{J}^T\mathbf{ZJ}$-matrix (a typical value using CG) for the β-divergence cost function ($\beta = 1.5$) in function of the tensor size. The tensors have dimensions $I \times I \times I$ and rank 10. The CPD model also has rank 10. A multilinear rank-$(I/25, I/25, I/25)$ approximation was used for $\mathcal{D}$. Aside from the largest dimensions, the computation time of the LMLRA is negligible compared to that of the $\mathbf{J}^T\mathbf{ZJx}$ products. The computation time for 100 evaluations of $\mathbf{J}^T\mathbf{Jx}$ for the LS distance is added for reference.

TABLE II

COMPRESSING $\mathcal{D}$ DECREASES THE COMPUTATION TIME AND NUMBER OF ITERATIONS FOR SUFFICIENTLY 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 MULTILINEAR RANK-(S, S, S) LMLRA.. THE NOISE IS MULTIPLICATIVE GAMMA NOISE WITH AN SNR OF 40 dB.

	Compression factor I/S			
	1	5	10	25
$I = 50$				
Optimal 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
$I = 100$				
Optimal 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
$I = 200$				
Optimal 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

The system $\mathbf{J}^T\mathbf{ZJx} = -\mathbf{g}$ 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.

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. The stopping criteria for the algorithms are again a relative step size below 10^{-15} or a relative cost function change below 10^{-15}. First, a comparison with the exact GGN method in function of the tensor dimensions can be seen in Fig. 4. Rank-10 tensors of dimensions $I \times I \times I$ are generated with factor matrix entries sampled from $U(0, 1)$ and the tensor $\mathcal{D}$

TABLE III

PROPER COMPRESSION OF $\mathcal{D}$ DECREASES THE COMPUTATION TIME FOR TENSORS OF LARGE ENOUGH DIMENSIONS. 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 $\mathbf{J}^T\mathbf{ZJx} = -\mathbf{g}$ 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 WITH AN SNR OF 40 dB.

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

is computed. For the exact GGN method, the mean time (over 20 trials) to compute $\hat{\mathbf{H}} = \mathbf{J}^T\mathbf{ZJ}$ 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 multilinear rank-$(10, 10, 10)$ tensor, next to the mean computation time of 100 products $\mathbf{J}^T\mathbf{ZJx}$ using this compressed version of $\mathcal{D}$, which in this case is plentiful to solve the system $\mathbf{J}^T\mathbf{ZJp} = -\mathbf{g}$ up to a precision of 10^{-6}. From Fig. 4, it can be seen that 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 larger tensors. Consequently, for such tensors, it is necessary to switch to the inexact GGN method, wherein $\mathbf{J}^T\mathbf{ZJp} = -\mathbf{g}$ 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. 4 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 $\mathbf{J}^T\mathbf{ZJx}$ 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 larger tensors, the CG/MINRES iterations can be more efficiently performed by using the GGN Hessian approximation implicitly.

For a CPD fitting problem, the inexact GGN method (with Jacobi preconditioning) is compared to the exact GGN method and two first-order methods, applied in the generalized CPD framework [21]. The first-order methods are the L-BFGS-B method [63], which is a bound-constrained quasi-Newton method and the Adam method [64], [65], which is a stochastic gradient descent method that uses both the first and second moments of the gradient to adjust the learning rate in every iteration. The first moment is used to maintain momentum in the direction of the solution and to overcome local minima, while the second moment is used to focus the update on variables that have not been changed significantly in recent iterations. The second-moment approach is called RMSProp [66], which was designed to improve on the speed of AdaGrad [67]. Adam performs a number of epochs until its stopping criterion is reached. The cost function is evaluated once per epoch, but the gradient is updated many times in every epoch, always using a different sample of tensor entries. In the experiment, the cost function is approximated in every epoch by uniformly sampling $\min\left(10^6, I^3/10\right)$ tensor entries, while 1000 gradient updates are applied per update, with 1000 entries per gradient update, following the default values from [65], [68]. Adam is used with the standard parameter values [64] and its stopping criterion is the second failed epoch, i.e., the second epoch in which the cost function value increases instead of decreases [65].

Rank-5 tensors with dimensions $I \times I \times I$ are generated with factor matrix entries sampled from $U(0,1)$ and the tensors are perturbed by multiplicative Gamma noise with an SNR of 20 dB. All methods are initialized by sampling random factor matrices with entries in $U(0,1)$ and performing five iterations of the GN LS algorithm from Tensorlab [62] (`cpd_gndl`). For the inexact GGN method, the $\mathcal{D}$-tensor is compressed to a multilinear rank-$(I/5, I/5, I/5)$ tensor. The cases $I \geq 200$ are omitted for the exact GGN method, as the use of the full $\mathbf{J}^{\mathrm{T}}\mathbf{J}$-matrix caused out-of-memory problems on the available hardware. The results can be found in Fig. 5. In the center plot it is shown that, regardless of the tensor dimensions, the GGN and L-BFGS-B methods find the same solution for the CPD, while the solution found by Adam is worse by a factor of about 2. From the left plot of Fig. 5, one can see that the computational costs of the L-BFGS-B and Adam methods grow faster with increasing tensor dimensions than the inexact GGN method, especially for $80 \leq I \leq 160$. Consequentially, for larger tensors, the inexact GGN method is significantly more efficient. For these 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. 5, one can see that the GGN methods require much fewer iterations than the L-BFGS-B method. The Adam method cannot be directly compared with the other methods, as it uses epochs instead of iterations. Adam generally requires about 10 epochs to converge in our experiments, while in every epoch 1000 iterations are performed on a part of the data. Combined with the faster iterations of the inexact GGN method compared to the exact GGN method, its lower number of iterations explains why the inexact GGN method scales much better than the other methods.

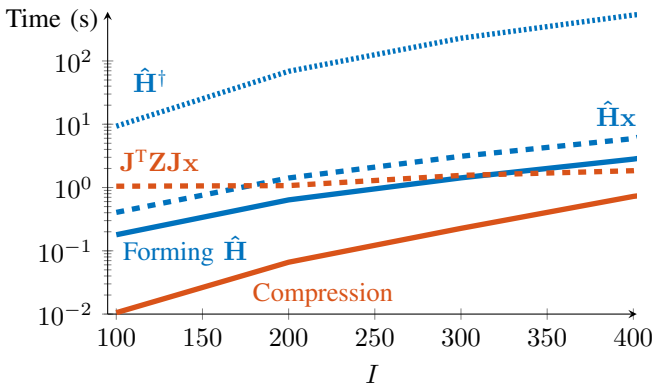

Fig. 4. The inexact GGN method with compression is faster than the exact GGN method for sufficiently large tensors. Mean time (over 20 trials) for the compression of $\mathcal{D}$ and for 100 evaluations of $\mathbf{J}^{\mathrm{T}}\mathbf{Z}\mathbf{J}\mathbf{x}$ after compression. For comparison, the 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 multilinear rank-$(I/10, I/10, I/10)$ approximation is used for $\mathcal{D}$.

In Fig. 6, the same experiment is repeated, but now the rank R is systematically increased and $I = 120$ is kept constant. From the left plot of Fig. 6, we can see that the computation time of the exact GGN method grows very quickly with increasing rank, so only ranks $R \leq 8$ were computed for this method. In contrast, the inexact GGN method has a much slower increase in computation time and can easily handle higher ranks. Still, for $R \geq 16$, the L-BFGS-B method becomes more computationally efficient than the GGN method. The GGN method is thus particularly effective when I is high and the rank R is low. Fortunately, these conditions frequently occur in practice. For example, in signal processing applications, it is often important to recover a small number of latent components from a large dataset with high precision [2]. For such cases, the GGN method can thus be significantly faster than first-order methods.

D. Large-scale tensors

For tensors of dimensions $I \times I \times I$, with $I > 300$, the inexact GGN method is still too computationally expensive to obtain a solution in a reasonable amount of time. Fortunately, the proposed techniques are independent of the specific solver that is used. Consequentially, they can also be applied in other second-order methods such as stochastic algorithms to tackle large-scale problems. Below, we apply the techniques from this paper to inexact GGN within the RBS framework [37]. Instead of using all tensor entries in every iteration of the algorithm, the RBS method samples a subtensor and only updates the

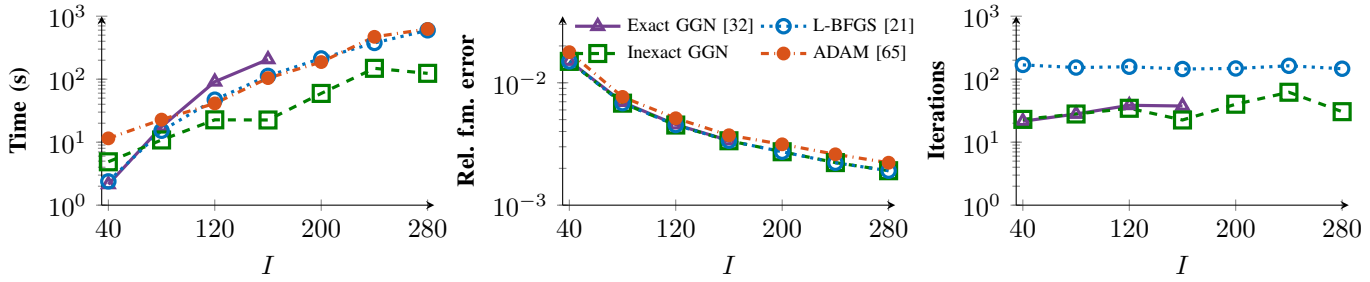

Fig. 5. The inexact GGN method with compression scales much better than the exact GGN method and the L-BFGS and Adam methods for increasing tensor dimensions, while the solution accuracy is at least as good. Median (over 20 trials) time (left), relative factor matrix error (center) and number of iterations (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 multilinear rank-$(I/5, I/5, I/5)$ approximation was used for $\hat{\mathcal{D}}$.

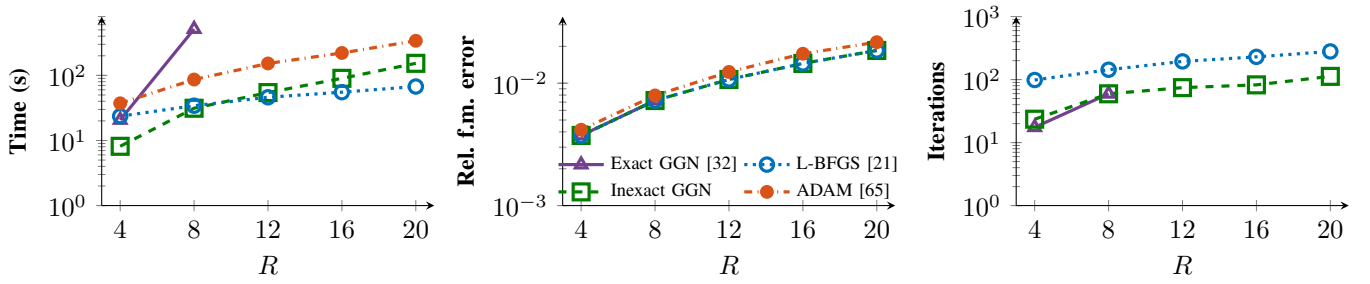

Fig. 6. The inexact GGN method with compression scales much better than the exact GGN method for increasing rank R. Still, for $R \geq 16$, the first-order method L-BFGS is more efficient. The Adam method is less accurate than the other methods. Median (over 20 trials) time (left), relative factor matrix 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 $120 \times 120 \times 120$ and rank R. The CPD model has rank R as well. A multilinear rank-$(20, 20, 20)$ approximation was used for $\hat{\mathcal{D}}$.

model with information from that subtensor. As a result, the RBS method can quickly find the CPD of a large-scale tensor with high accuracy. For this experiment, we have extended the RBS strategy so that it can handle non-LS cost functions as well: in every iteration, instead of the GN method, the GGN method is used to update the tensor with information from the sampled subtensor. In this way, it becomes feasible to apply the GGN method for the decomposition of large-scale tensors.

We tested the GGN RBS method on a dense $500 \times 500 \times 500$ tensor of rank 10 with factor matrix entries sampled from $U(0, 1)$. Multiplicative Gamma noise is added with an SNR of 20 dB. Three methods are compared: the GGN RBS method with a β-divergence cost function with $\beta = 1.5$, the first-order stochastic Adam method [65] within the generalized CPD framework [21] with the same cost function and the GN RBS method which uses the LS cost function. The RBS methods use a block size of $80 \times 80 \times 80$ and stop when the relative step becomes smaller than 10^{-5}. They use a maximal step size $\|\mathbf{p}\|$ of 10 for the first 350 iterations and thereafter, a maximal step size of $0.005 \cdot 0.99^{(k-350)}$ is used, with $k > 350$ the iteration number. The Adam method samples 1 000 000 values when evaluating the cost function and in each of the 1000 iterations per epoch, it samples 100 000 values for the gradient. The method stops when a second failed epoch occurs.

In Table IV, the results can be seen for the three methods. From the right column, one can see that GGN RBS obtains the best solution. The solution of the GN RBS method is worse because it uses a less suitable cost function, while the solution of Adam is simply less accurate. From the left column, it can

TABLE IV
USING THE RBS FRAMEWORK, THE GGN METHOD CAN BE APPLIED TO OBTAIN CPDS OF LARGE TENSORS WITH NON-LS COST FUNCTIONS. COMPARED TO THE FIRST-ORDER OPTIMIZATION METHOD ADAM, ITS SOLUTION IS BOTH MORE ACCURATE AND OBTAINED IN A SHORTER AMOUNT OF TIME. ALL RESULTS ARE MEDIANS OVER 10 TRIALS.

	Time (s)	Rel. f.m. error
GGN RBS	268.29	$2.68 \cdot 10^{-3}$
GCPD with Adam	649.02	$4.22 \cdot 10^{-3}$
GN RBS	25.93	$2.86 \cdot 10^{-3}$

be seen that the GGN RBS method is slower than GN RBS, which is to expected, as the latter uses a cost function that is much less expensive to evaluate. More importantly, the GGN RBS method is significantly faster than the first-order Adam method, while obtaining a more accurate solution.

The GGN method can thus be applied in a stochastic context to obtain an accurate solution for the CPD fitting problem with a non-LS cost function in a reasonable amount of time. Compared to an existing first-order stochastic method, it is both more accurate and faster.

E. Separating fMRI sources

In neuro-imaging, fMRI is a noninvasive technique that indirectly studies brain activity by measuring fluctuations of the Blood Oxygenation Level Dependent (BOLD) signal [69]. The localization of the activated brain areas from fMRI data is a challenging blind source separation (BSS) problem. The

sources consist of a combination of spatial maps (activated areas) and time courses (activation times). Following the methodology of [70], we use a CPD to perform blind fMRI source unmixing to recover eight simulated spatio-temporal sources. The spatial and temporal modes of these sources are shown in Fig. 7. They are generated based on knowledge of the statistical characteristics of the underlying neuronal sources. The spatial signatures have dimensions 60×60 and are vectorized into vectors of length 3600. Over time, 100 different samples are collected. For fMRI data, the phase difference between activated and resting states of a voxel induced by the task activation, is usually less than $\pi/9$ [71], so the phases of each pixel are sampled from $U(\pi/18, \pi/18)$. 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 are independently 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. The cost function from Sec. III is thus expected to perform better 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 $25\,\mathrm{dB}$. 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 Figs. 8 and 9, 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 V, the peak SNR (PSNR) and structural similarity index (SSI) [72] 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 [73]. 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 V that the results for the Rician-dedicated cost function are indeed somewhat better than those for the LS cost function.

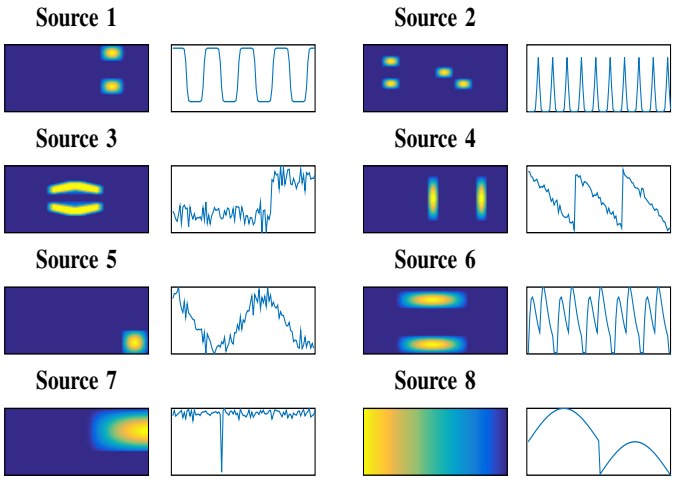

Fig. 7. Spatial (left) and temporal (right) modes of eight simulated fMRI sources.

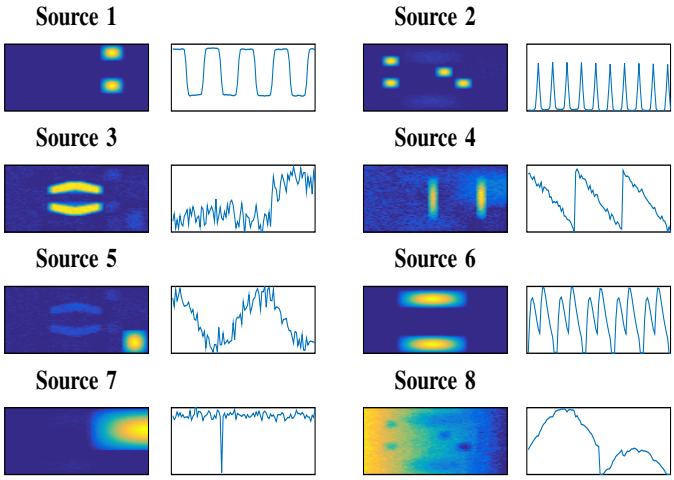

Fig. 8. Unmixed fMRI sources when a CPD with a standard LS cost function is used.

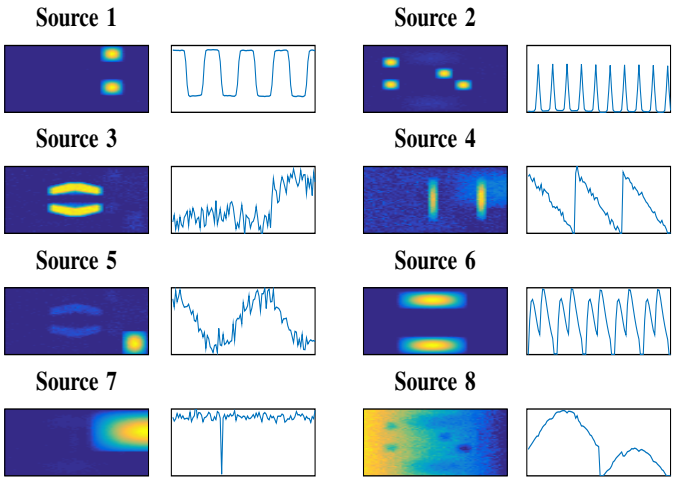

Fig. 9. Unmixed fMRI sources when a CPD with a Rician-dedicated cost function is used.

TABLE V
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 exploiting the multilinear structure of the Hessian approximation, by compressing the second-order information with an LMLRA and by applying a specialized preconditioner. 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 proposed techniques can be applied in other second-order methods as well, as shown for a large-scale stochastic algorithm. The performance of the proposed 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.), Elsevier, 2018.

[9] R. Bro and H. Kiers, "A new efficient method for determining the number of components in PARAFAC models," *J. Chemom*., vol. 17, pp. 274–286, 2003.

[10] M. Araujo, S. Papadimitriou, S. Günnemann, C. Faloutsos, P. Basu, A. Swami, E. Papalexakis, and D. Koutra, "Com2: Fast automatic discovery of temporal ('comet') communities," in *In Advances in Knowledge Discovery and Data Mining* (V. Tseng, T. Ho, Z.-H. Zhou, A. Chen, and H.-Y. Kao, eds.), pp. 271–283, Springer, 2014.

[11] M. Mørup and L. K. Hansen, "Automatic relevance determination for multi-way models," *J. Chemom*., vol. 23, no. 7–8, pp. 352–363, 2009.

[12] L. Cheng, X. Tong, S. Wang, Y.-C. Wu, and H. V. Poor, "Learning nonnegative factors from tensor data: Probabilistic modeling and inference algorithm," *IEEE Trans. Signal Process*., vol. 68, pp. 1792–1806, 2020.

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

[14] 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.

[15] 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.

[16] S. Coxe, S. G. West, and L. S. Aiken, "The analysis of count data: A gentle introduction to Poisson regression and its alternatives," *J. Pers. Assess*., vol. 91, no. 2, pp. 121–136, 2009.

[17] G. Steidl and T. Teuber, "Removing multiplicative noise by Douglas–Rachford splitting methods," *J. Math. Imaging Vis*., vol. 36, no. 2, pp. 168–184, 2010.

[18] E. Bratsolis and M. Sigelle, "Fast SAR image restoration, segmentation, and detection of high-reflectance regions," *IEEE Trans. Geosci. Remote. Sens*., vol. 41, no. 12, pp. 2890–2899, 2003.

[19] 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.

[20] H. Gudbjartsson and S. Patz, "The Rician distribution of noisy MRI data," *Magn. Reson. Medicine*, vol. 34, no. 6, pp. 910–914, 1995.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] Y. Xu and W. Yin, "A block coordinate descent method for regularized multi-convex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imaging Sci*., vol. 6, no. 3, pp. 1758–1789, 2013.

[28] X. Fu, S. Ibrahim, H.-T. Wai, C. Gao, and K. Huang, "Block-randomized stochastic proximal gradient for low-rank tensor factorization," *IEEE Trans. Signal Process*., vol. 68, pp. 2170–2185, 2020.

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

[30] A. Milzarek, X. Xiao, S. Cen, Z. Wen, and M. Ulbrich, "A stochastic semismooth Newton method for nonsmooth nonconvex optimization," *SIAM J. Optim*., vol. 29, no. 4, pp. 2916–2948, 2018.

[31] T. George, C. Laurent, X. Bouthillier, N. Ballas, and P. Vincent, "Fast approximate natural gradient descent in a Kronecker factored eigenbasis," in *Adv. Neural. Inf. Process. Syst.*, pp. 9550–9560, 2018.

[32] M. Vandecappelle, N. Vervliet, and L. De Lathauwer, "A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost," *IEEE Trans. Signal Process*., vol. 68, pp. 4454–4465, 2020.

[33] 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.

[34] 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.

[35] L. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychom*., vol. 31, pp. 279–311, 1966.

[36] J. D. Carroll, S. Pruzansky, and J. B. Kruskal, "CANDELINC: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters," *Psychom*., vol. 45, no. 1, pp. 3–24, 1980.

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

[38] G. Tomasi and R. Bro, "A comparison of algorithms for fitting the PARAFAC model," *Comput. Stat. Data Anal*., vol. 50, pp. 1700–1734, 4 2006.

[39] P. Paatero, "A weighted non-negative least squares algorithm for three-way "PARAFAC" factor analysis," *Chemom. Intell. Lab. Syst*., vol. 38, pp. 223–242, 10 1997.

[40] A.-H. Phan, P. Tichavský, and A. Cichocki, "Low complexity damped Gauss–Newton algorithms for CANDECOMP/PARAFAC," *SIAM J. Matrix Anal. Appl*., vol. 34, pp. 126–147, 1 2013.

[41] P. Tichavský, A. Phan, and A. Cichocki, "Weighted Krylov–Levenberg-Marquardt method for canonical polyadic tensor decomposition," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*, (Barcelona, Spain), pp. 3917–3921, 2020.

[42] R. Bro and C. A. Andersson, "Improving the speed of multiway algorithms: Part II: Compression," *Chemom. Intell. Lab. Syst*., vol. 42, no. 1, pp. 105–113, 1998.

[43] 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.

[44] 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.

[45] 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.

[46] K. Hayashi, G. Ballard, Y. Jiang, and M. J. Tobia, "Shared-memory parallelization of MTTKRP for dense tensors," in *Proc. 23rd ACM SIGPLAN Symp. Princ. and Pract. Parallel Programming*, (Los Angeles, CA, USA), pp. 393–394, 2018.

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

[48] 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. Decis. Control (CDC 2016)*, (Las Vegas, NV, USA), pp. 1099–1104, 2016.

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

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

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

[52] 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.

[53] M. Sørensen and L. 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.

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

[55] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997.

[56] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU press, 2012.

[57] T. Davis, S. Rajamanickam, and W. Sid-Lakhdar, "A survey of direct methods for sparse linear systems," *Acta Numer*., vol. 25, pp. 383–566, 2016.

[58] 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.

[59] 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.

[60] 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.

[61] J. Sijbers, A. den Dekker, D. Dyck, and E. Raman, "Estimation of signal and noise from Rician distributed data," in *Proc. Int. Conf. Signal Process. Commun.*, (Gran Canaria, Canary Islands, Spain), pp. 140–142, 1198.

[62] 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.

[63] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization," *ACM Trans. Math. Softw*., vol. 23, no. 4, pp. 550–560, 1997.

[64] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations (ICLR 2015)*, (San Diego, CA, USA), pp. 9550–9560, 2015.

[65] T. G. Kolda and D. Hong, "Stochastic gradients for large-scale tensor decomposition," *SIAM J. Math. Data Sci*., 2020. accepted for publication.

[66] T. Tieleman and G. Hinton, "Lecture 6.5-RMSProp, Coursera: Neural networks for machine learning," *Univ. Toronto, Tech. Rep*., 2012.

[67] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res*., vol. 12, pp. 2121–2159, 2011.

[68] B. W. Bader, T. G. Kolda, *et al.*, "MATLAB Tensor Toolbox version 3.1." Available online at http://www.tensortoolbox.org, June 2019.

[69] M. A. Lindquist *et al.*, "The statistical analysis of fMRI data," *Stat. science*, vol. 23, no. 4, pp. 439–464, 2008.

[70] 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), 2019.

[71] F. G. C. Hoogenraad, J. R. Reichenbach, E. M. Haacke, S. Lai, K. Kuppusamy, and M. Sprenger, "In vivo measurement of changes in venous blood-oxygenation with high resolution functional MRI at 0.95 Tesla by measuring changes in susceptibility and velocity," *Magn. Reson. Medicine*, vol. 39, no. 1, pp. 97–107, 1998.

[72] Z. 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.

[73] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *Proc. 37th Asilomar Conf. Signals, Systems Computers (ASILOMAR 2003)*, vol. 2, (Pacific Grove, CA, USA), pp. 1398–1402, 2003.