SIAM J. OPTIM.
Vol. 23, No. 2, pp. 695–720
© 2013 Society for Industrial and Applied Mathematics

OPTIMIZATION-BASED ALGORITHMS FOR TENSOR DECOMPOSITIONS: CANONICAL POLYADIC DECOMPOSITION, DECOMPOSITION IN RANK-$(L_r, L_r, 1)$ TERMS, AND A NEW GENERALIZATION*

LAURENT SORBER†, MARC VAN BAREL†, AND LIEVEN DE LATHAUWER‡

Abstract. The canonical polyadic and rank-$(L_r, L_r, 1)$ block term decomposition (CPD and BTD, respectively) are two closely related tensor decompositions. The CPD and, recently, BTD are important tools in psychometrics, chemometrics, neuroscience, and signal processing. We present a decomposition that generalizes these two and develop algorithms for its computation. Among these algorithms are alternating least squares schemes, several general unconstrained optimization techniques, and matrix-free nonlinear least squares methods. In the latter we exploit the structure of the Jacobian's Gramian to reduce computational and memory cost. Combined with an effective preconditioner, numerical experiments confirm that these methods are among the most efficient and robust currently available for computing the CPD, rank-$(L_r, L_r, 1)$ BTD, and their generalized decomposition.

Key words. multilinear algebra, tensor decompositions, canonical polyadic decomposition, block term decomposition, optimization, algorithms

AMS subject classifications. 90-08, 90C06, 90C53, 90C90, 65K05

DOI. 10.1137/120868323

1. Introduction.

Tensor decompositions are important techniques for data mining, dimensionality reduction, pattern recognition, object detection, classification, clustering, and blind source separation [4, 8, 9, 15, 17, 55, 57]. For an introduction to tensor decompositions and their applications we refer to the review article and books [33, 35, 58]. The two main tensor generalizations of the singular value decomposition (SVD) are, on one hand, the Tucker decomposition or multilinear SVD (MLSVD) [19, 65, 66] and, on the other hand, the canonical polyadic decomposition (CPD) [7, 27]. In fact, the CPD generalizes any rank-revealing matrix decomposition. The MLSVD and CPD are connected with two different tensor generalizations of the concept of matrix rank. The former is linked with the set of mode-n ranks, which generalize column rank, row rank, etc. The latter generalizes rank in the sense of the minimal number of rank-one terms whose sum is equal to a given tensor. Block term decompositions (BTD) were introduced by De Lathauwer [12, 13, 20] as a frame-

*Received by the editors March 1, 2012; accepted for publication (in revised form) January 17, 2013; published electronically April 18, 2013. The scientific responsibility rests with the authors.
http://www.siam.org/journals/siopt/23-2/86832.html

†Department of Computer Science, KU Leuven, Celestijnenlaan 200A, BE-3001 Leuven, Belgium (Laurent.Sorber@cs.kuleuven.be, Marc.VanBarel@cs.kuleuven.be). The first author is supported by a doctoral fellowship of the Flanders agency for Innovation by Science and Technology (IWT). The second author's research is supported by the Research Council KU Leuven project OT/10/038 and CoE EF/05/006 Optimization in Engineering (OPTEC), and by the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office.

‡Group Science, Engineering and Technology, KU Leuven Kulak, BE-8500 Kortrijk, Belgium and Department of Electrical Engineering (ESAT), KU Leuven, BE-3001 Leuven, Belgium (Lieven.DeLathauwer@kuleuven-kulak.be, Lieven.DeLathauwer@esat.kuleuven.be). This author's research is supported by the Research Council KU Leuven GOA-MaNet, CoE EF/05/006 Optimization in Engineering (OPTEC), CIF1, STRT1/08/023; the Research Foundation Flanders (FWO) project G.0427.10N and Research Communities ICCoS, ANMMM, and MLDM; the Belgian Network DYSCO; and by EU: ERNSI.

work that unifies the MLSVD and CPD. Of particular interest is the rank-$(L_r, L_r, 1)$ BTD, which has recently proved to be useful in blind source separation [14, 16], and particularly in telecommunication applications [18, 42, 56].

In this article, we propose several optimization-based algorithms to compute the CPD and rank-$(L_r, L_r, 1)$ BTD. In fact, we first introduce a more general decomposition called the (rank-$L_r \circ$ rank-1) BTD. Whereas the rank-$(L_r, L_r, 1)$ BTD is a generalization of the third-order CPD, the (rank-$L_r \circ$ rank-1) BTD is a generalization of both the Nth-order CPD and the rank-$(L_r, L_r, 1)$ BTD. Consequently, algorithms designed for the more general decomposition are also applicable to the former decompositions. We develop alternating least squares (ALS) schemes, memory-efficient gradient-based methods such as nonlinear conjugate gradient and limited-memory BFGS using both line search and trust-region frameworks, and nonlinear least squares methods such as Gauss–Newton and Levenberg–Marquardt. For the latter, we derive a matrix-free implementation that exploits the structure inherent to the decomposition, reducing computational complexity significantly compared to the exact method and reducing memory cost to that of ALS. Throughout the paper, we consider the general case where the decompositions may be complex, although real decompositions are also supported through the choice of the initialization. Our numerical experiments reveal that ALS, despite its popularity, is in many cases not a suitable choice. Nonlinear least squares methods are less sensitive to the type of initialization and are often not only more efficient and less prone to so-called swamps but are also very robust for more difficult decompositions in which there is some degree of collinearity between factors.

The paper is organized as follows. In section 2 we review our notation and introduce some basic definitions. In section 3 we recall the canonical polyadic decomposition and the rank-$(L_r, L_r, 1)$ block term decomposition and also introduce the (rank-$L_r \circ$ rank-1) block term decomposition. In section 4 we derive the (co)gradient necessary for alternating least squares and gradient-based methods. We then demonstrate that the associated Gramian of the Jacobian may be intuitively expected to approximate the objective function's Hessian relatively well as the residuals decrease, depending on the tensor's order and rank. We also show that this matrix has a very specific structure, which we exploit in matrix-free nonlinear least squares algorithms. We close this section with an overview of each algorithm's computational complexity. In section 5 we evaluate the performance of the proposed algorithms with Monte Carlo simulations of both exact and noisy decompositions. We conclude the paper in section 6.

2. Notation and preliminaries. A tensor is an element of a tensor product of vector spaces. In this article, we refer to a tensor represented as a multidimensional array, given a choice of bases for each of these vector spaces. The order, or the number of modes, of a tensor is the number of indices associated with each element of that tensor. Vectors are denoted by boldface letters and are lower case, e.g., $\boldsymbol{a}$. Matrices are denoted by capital letters, e.g., A. Higher-order tensors are denoted by Euler script letters, e.g., $\mathcal{A}$. An entry of a vector $\boldsymbol{a}$, matrix A, or tensor $\mathcal{A}$ is denoted by a_i, a_{ij}, or $a_{ijk\ldots}$, depending on the number of modes. A colon is used to select all entries of a mode. For instance, $\boldsymbol{a}_{:j}$ corresponds to the jth column of a matrix A. When there is no confusion, we also use $\boldsymbol{a}_j$ to denote the jth column of the matrix A. Mode-n vectors are the generalization of matrix rows and columns to tensors. A mode-n vector is a vector in which all but one of the indices are fixed. For example, the mode-1, mode-2, and mode-3 vectors of a third-order tensor $\mathcal{T}$ are denoted by $\boldsymbol{t}_{:jk}$, $\boldsymbol{t}_{i:k}$ and $\boldsymbol{t}_{ij:}$, respectively. Sequences are denoted by a superscript in parentheses, e.g., $\{A^{(n)}\}_{n=1}^{N}$. The superscripts $\cdot^{\mathrm{T}}$, $\cdot^{\mathrm{H}}$, $\cdot^{-1}$ and $\cdot^{\dagger}$ are used for the

transpose, Hermitian conjugate, matrix inverse, and Moore–Penrose pseudoinverse, respectively. The complex conjugate is denoted by an overbar, e.g., $\overline{a}$ is the complex conjugate of the scalar a. We use parentheses to denote the concatenation of two or more vectors, e.g., $(\boldsymbol{a}, \boldsymbol{b})$ is equivalent to $[\boldsymbol{a}^{\mathrm{T}} \quad \boldsymbol{b}^{\mathrm{T}}]^{\mathrm{T}}$. The $n \times n$ identity matrix is denoted by $\mathbb{I}_n$, and the all-zero and all-one $m \times n$ matrices by $0_{m\times n}$ and $1_{m\times n}$, respectively. The Kronecker product of two matrices A and B is denoted by $A \otimes B$.

DEFINITION 2.1. *The* inner product $\langle \mathcal{T}, \mathcal{U} \rangle$ *of two tensors* $\mathcal{T}, \mathcal{U} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$ *is defined as* $\langle \mathcal{T}, \mathcal{U} \rangle = \sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} \overline{t}_{i_1 \cdots i_N} u_{i_1 \cdots i_N}$.

DEFINITION 2.2. *The* (Frobenius) norm *is denoted by* $\|\mathcal{T}\| = \sqrt{\langle \mathcal{T}, \mathcal{T} \rangle}$.

DEFINITION 2.3. *The* outer product $\mathcal{T} \circ \mathcal{U}$ *of a tensor* $\mathcal{T} \in \mathbb{C}^{I_1 \times \cdots \times I_P}$ *and a tensor* $\mathcal{U} \in \mathbb{C}^{J_1 \times \cdots \times J_Q}$ *is the tensor defined by* $(\mathcal{T} \circ \mathcal{U})_{i_1 \cdots i_P j_1 \cdots j_Q} = t_{i_1 \cdots i_P} u_{j_1 \cdots j_Q}$.

DEFINITION 2.4. *An* N*th-order tensor* $\mathcal{T}$ *is* rank-one *if it is equal to the outer product of* N *nonzero vectors* $\boldsymbol{a}^{(n)} \in \mathbb{C}^{I_n}$, *i.e.,* $\mathcal{T} = \boldsymbol{a}^{(1)} \circ \cdots \circ \boldsymbol{a}^{(N)}$.

DEFINITION 2.5. *In a* vectorization $\mathrm{vec}(A)$ *of a matrix* $A \in \mathbb{C}^{I \times J}$, *matrix element* (i, j) *is mapped to vector element* $(i + (j-1)J)$.

DEFINITION 2.6. *In a* mode-n matricization *or a* flattening *or* unfolding $T_{(n)}$ *of an* N*th-order tensor* $\mathcal{T} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$, *tensor element with indices* $(i_1, \ldots, i_N)$ *is mapped to matrix element* (i_n, j) *such that*

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k \text{ with } J_k = \begin{cases} 1 & \text{for } k = 1 \text{ or } (k = 2 \text{ and } n = 1) \\ \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m & \text{otherwise.} \end{cases}$$

In other words, the columns of the mode-n matricization $T_{(n)}$ are the mode-n vectors of $\mathcal{T}$ arranged along the natural order of the remaining modes.

DEFINITION 2.7. *The* n-rank *of an* N*th-order tensor* $\mathcal{T}$, *denoted by* $R_n = \mathrm{rank}_n(\mathcal{T})$, *is defined as the column rank of* $T_{(n)}$. *In other words, the* n*-rank is the dimension of the space spanned by the mode-*n *vectors of* $\mathcal{T}$. *A tensor characterized by its* n*-ranks* R_n, $n = 1, \ldots, N$, *is said to be of rank-*$(R_1, \ldots, R_N)$.

DEFINITION 2.8. *The* n-mode (matrix) product $\mathcal{T} \times_n A$ *of a tensor* $\mathcal{T} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$ *with a matrix* $A \in \mathbb{C}^{J \times I_n}$ *is the tensor defined by the tensor unfolding of the matrix* $(\mathcal{T} \times_n A)_{(n)} = A \cdot T_{(n)}$.

DEFINITION 2.9. *The* Khatri–Rao product [32] *of two matrices* $A \in \mathbb{C}^{I \times K}$ *and* $B \in \mathbb{C}^{J \times K}$ *is defined as* $A \odot B = \begin{bmatrix} \boldsymbol{a}_1 \otimes \boldsymbol{b}_1 & \cdots & \boldsymbol{a}_K \otimes \boldsymbol{b}_K \end{bmatrix}$.

DEFINITION 2.10. *The* Hadamard product *of two matrices* $A \in \mathbb{C}^{I \times J}$ *and* $B \in \mathbb{C}^{I \times J}$ *is the matrix defined by* $(A * B)_{ij} = a_{ij} b_{ij}$.

3. Tensor decompositions.

3.1. The canonical polyadic decomposition. The CPD approximates a tensor with a sum of R rank-one tensors. It was introduced by Hitchcock in 1927 [29, 30] and was later referred to as the canonical decomposition (CANDECOMP) [7] and parallel factor decomposition (PARAFAC) [27] in psychometrics and phonetics, respectively. Let $\mathcal{T} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$ and $\boldsymbol{a}_r^{(n)} \in \mathbb{C}^{I_n}$, $\boldsymbol{a}_r^{(n)}$ nonzero; then

$$\mathcal{T} \approx \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \cdots \circ \boldsymbol{a}_r^{(N)} \tag{3.1}$$

is a CPD of the tensor $\mathcal{T}$ in R rank-one terms. The rank of the tensor is defined as the smallest R for which (3.1) is exact. Such a decomposition is called a rank

decomposition and is, in contrast to the matrix case, often unique. Let $A^{(n)} = [\, \boldsymbol{a}_1^{(n)} \cdots \boldsymbol{a}_R^{(n)} \,]$ be the factor matrix corresponding to the nth mode. The CPD can then be written in matrix form as

$$T_{(n)} \approx A^{(n)} \cdot V^{\{n\}^{\mathrm{T}}}, \text{ where } V^{\sigma} \triangleq \bigodot_{\substack{n=0 \\ N-n \notin \sigma}}^{N-1} A^{(N-n)} \tag{3.2}$$

for any $1 \leq n \leq N$. Note that σ defines a set of factor matrices to exclude from the string of Khatri–Rao products. For example, $V^{\{n\}} = A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \cdots \odot A^{(1)}$.

To a large extent, the practical importance of the CPD stems from its uniqueness properties. It is clear that one can arbitrarily permute the different rank-one terms. Also, the factors of a single rank-one term may be arbitrarily scaled, as long as their product remains the same. We call a CPD essentially unique when it is subject only to these trivial indeterminacies. The most well-known result on uniqueness is due to Kruskal [36, 37] and depends on the concept of k-rank. The k-rank of a matrix A, denoted k_A, is defined as the maximum value k such that any k columns of A are linearly independent [36]. Kruskal's sufficient condition for the unicity of a rank decomposition, generalized to Nth-order tensors by Sidiropoulos and Bro [54], is

$$\sum_{n=1}^{N} k_{A^{(n)}} \geq 2R + (N-1). \tag{3.3}$$

This condition can be relaxed by an order of magnitude if the tensor is long in one mode [11, 31]. More recent relaxations of Kruskal's condition are available in [24, 25].

3.2. The rank-$(L_r, L_r, 1)$ block term decomposition. The rank-$(L_r, L_r, 1)$ BTD [12, 13, 20] approximates a third-order tensor by a sum of R terms, each of which is an outer product of a rank-L_r matrix and a nonzero vector. Let $\mathcal{T}$ be a third-order tensor, $A_r \in \mathbb{C}^{I_1 \times L_r}$ and $B_r \in \mathbb{C}^{I_2 \times L_r}$ be rank-L_r matrices, and $\boldsymbol{c}_r \in \mathbb{C}^{I_3}$, $\boldsymbol{c}_r$ nonzero; then

$$\mathcal{T} \approx \sum_{r=1}^{R} (A_r \cdot B_r^{\mathrm{T}}) \circ \boldsymbol{c}_r \tag{3.4}$$

is a BTD of the tensor $\mathcal{T}$ in R rank-$(L_r, L_r, 1)$ terms (cf. Figure 3.1). In addition to the permutation and scaling indeterminacies inherited from the CPD, the factors A_r may be postmultiplied by any nonsingular matrix $F_r \in \mathbb{C}^{L_r \times L_r}$, provided B_r^{T} is premultiplied by the inverse of F_r. This decomposition is also called essentially unique when it is subject only to these trivial indeterminacies. When the matrices $\begin{bmatrix} A_1 & \cdots & A_R \end{bmatrix}$

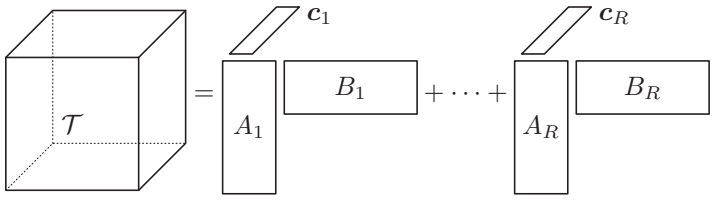


FIG. 3.1. *Rank-$(L_r, L_r, 1)$ block term decomposition of a third-order tensor.*

and $\begin{bmatrix} B_1 & \cdots & B_R \end{bmatrix}$ are full column rank and the matrix $\begin{bmatrix} \boldsymbol{c}_1 & \cdots & \boldsymbol{c}_R \end{bmatrix}$ does not contain collinear columns, the decomposition is guaranteed to be essentially unique. Furthermore, the decomposition may then be computed with a generalized eigenvalue decomposition (GEVD). These GEVD-type sufficient conditions, along with Kruskal-type conditions, were derived in [13]. A new necessary and sufficient condition was derived in [14]. The CPD and rank-$(L_r, L_r, 1)$ BTD have both been applied in telecommunication applications [18, 42, 55, 56, 57]. However, the terms of a rank-$(L_r, L_r, 1)$ BTD possess a more general low-rank structure that, together with its relatively mild conditions for unicity, make it a promising new candidate for blind source separation [14, 16]. The rank-$(L_r, L_r, 1)$ BTD generalizes the CPD for third-order tensors. We now introduce a new block term decomposition which generalizes both these decompositions. Designing algorithms for this more general decomposition requires little additional effort and will result in algorithms applicable to all of the aforementioned decompositions.

3.3. The (rank-L_r ∘ rank-1) block term decomposition. The (rank-L_r ∘ rank-1) block term decomposition approximates a tensor by a sum of R terms, each of which is an outer product of a rank-L_r tensor and a rank-one tensor. Let N, P, and Q be positive integers so that $N = P + Q$ and let $\mathcal{T}$ be an Nth-order tensor, $R' = \sum_{r=1}^{R} L_r$, $A^{(p)} = \begin{bmatrix} \boldsymbol{a}_1^{(p)} & \cdots & \boldsymbol{a}_{R'}^{(p)} \end{bmatrix}$ and $C^{(q)} = \begin{bmatrix} \boldsymbol{c}_1^{(q)} & \cdots & \boldsymbol{c}_R^{(q)} \end{bmatrix}$; then

$$\mathcal{T} \approx \sum_{r=1}^{R} \left[\left(\sum_{r'=1+\sum_{u=1}^{r-1} L_u}^{\sum_{u=1}^{r} L_u} \boldsymbol{a}_{r'}^{(1)} \circ \cdots \circ \boldsymbol{a}_{r'}^{(P)} \right) \circ \left(\boldsymbol{c}_r^{(1)} \circ \cdots \circ \boldsymbol{c}_r^{(Q)} \right) \right] \tag{3.5}$$

is a BTD of the tensor $\mathcal{T}$ in R (rank-L_r ∘ rank-one) terms (cf. Figure 3.2). By setting $L_r \equiv 1$, it is obvious that this decomposition reduces to the CPD. On the other hand, it is equivalent to the rank-$(L_r, L_r, 1)$ BTD if $P = 2$ and $Q = 1$.

Another way to interpret this decomposition is as a structured CPD in which the nth factor matrix is just $A^{(n)}$ when $1 \le n \le P$ and is defined as

$$A^{(n)} \triangleq C^{(n-P)} \cdot E \text{ when } P < n \le N. \tag{3.6}$$

Here, E is defined as the $R \times R'$ block-diagonal matrix $\operatorname{diag}(\mathbf{1}_{1\times L_1}, \ldots, \mathbf{1}_{1\times L_R})$ in which the rth block on the diagonal is the row vector $\mathbf{1}_{1\times L_r}$. We use this interpretation to formulate (3.5) as the nonlinear least squares problem

$$\min_{\substack{A^{(1)},\ldots,A^{(P)} \\ C^{(1)},\ldots,C^{(Q)}}} f_{\text{BTD}}, \text{ where } f_{\text{BTD}} \triangleq \frac{1}{2}\|\mathcal{F}_{\text{BTD}}\|^2, \tag{3.7}$$

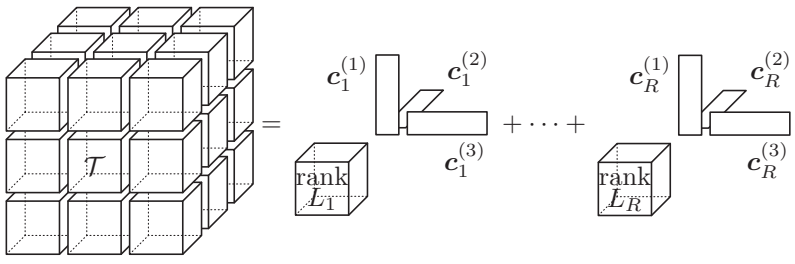


FIG. 3.2. *A (rank-L_r ∘ rank-1) block term decomposition of a sixth-order tensor.*

and the mode-n unfolding of the residual tensor $\mathcal{F}_{\text{BTD}}$ is defined as

$$(\mathcal{F}_{\text{BTD}})_{(n)} \triangleq A^{(n)} \cdot V^{\{n\}^{\text{T}}} - T_{(n)} \text{ for any } 1 \leq n \leq N. \tag{3.8}$$

Although the residual tensor $\mathcal{F}_{\text{BTD}}$ is analytic in its argument, the objective function f_{BTD} is not because of its dependency on the complex conjugate of the argument. This implies that f_{BTD} is not complex differentiable and hence that its Taylor series does not exist everywhere on its domain. In the following section, we derive gradient-based unconstrained optimization methods and nonlinear least squares methods for (3.7), built with generalized algorithms for this class of optimization problems [59]. These methods are based on the observation that if a function of complex variables is analytic in the real and imaginary part of its argument, it is also analytic in its argument and the complex conjugate of its argument as a whole [1, 67].

4. Algorithms for the general decomposition.

4.1. General unconstrained optimization and alternating least squares. Solving (3.7) not only allows us to compute the (rank-$L_r \circ$ rank-1) BTD but also the CPD and the rank-$(L_r, L_r, 1)$ BTD. To this end, we first consider gradient-based unconstrained optimization methods that attempt to minimize f_{BTD} directly. Among these methods are the nonlinear conjugate gradient method and quasi-Newton methods such as the (limited-memory) BFGS method [43]. The complex counterparts of these methods [59] are built on the second-order model

$$m_k^f(\overset{c}{\boldsymbol{p}}) \triangleq f(\overset{c}{\boldsymbol{z}}_k) + \overset{c}{\boldsymbol{p}}^{\text{T}} \frac{\partial f(\overset{c}{\boldsymbol{z}}_k)}{\partial \overset{c}{\boldsymbol{z}}} + \frac{1}{2} \overset{c}{\boldsymbol{p}}^{\text{H}} B_k \overset{c}{\boldsymbol{p}} \tag{4.1}$$

of the objective function f at the current iterate $\overset{c}{\boldsymbol{z}}_k$, where the superscript $\overset{c}{\cdot}$ denotes the concatenation of its argument with its complex conjugate, i.e., $\overset{c}{\boldsymbol{z}} \equiv (\boldsymbol{z}, \overline{\boldsymbol{z}})$, and B_k is a Hermitian positive definite matrix that is updated at every iteration. The partial derivative in (4.1) is called the complex gradient at $\overset{c}{\boldsymbol{z}}_k$ and is composed of two parts: its top half $\frac{\partial f}{\partial \boldsymbol{z}}$ is the cogradient and its bottom half $\frac{\partial f}{\partial \overline{\boldsymbol{z}}}$ is the conjugate cogradient. For real-valued f, the cogradients are each other's complex conjugates. By definition, these complex derivatives are to be interpreted as partial derivatives with respect to complex variables while treating their complex conjugates as constant. They are also known, especially in the German literature, as Wirtinger derivatives [53]. The minimizer of the convex quadratic model (4.1) can be obtained by setting the model's conjugate complex gradient equal to zero and is given by

$$\overset{c}{\boldsymbol{p}}_k^* = -B_k^{-1} \overline{\frac{\partial f(\overset{c}{\boldsymbol{z}}_k)}{\partial \overset{c}{\boldsymbol{z}}}}. \tag{4.2}$$

Storing and manipulating the full Hessian approximation B_k or its inverse B_k^{-1} can be quite expensive for functions of many variables, as is often the case for tensor decompositions. Many strategies to store the approximation to the Hessian exist. For instance, the nonlinear conjugate gradient method can be interpreted to build B_k as the sum of the identity matrix and a rank-two update. The limited-memory BFGS (L-BFGS) method generalizes this concept to the sum of a (scaled) identity matrix and m rank-two updates. In practice, L-BFGS often performs better than nonlinear conjugate gradient due to its flexibility and is regarded as superior to the latter.

In Theorem 4.4 we derive an expression for f_{BTD}'s cogradient, which enables us to update B_k and compute the search direction (4.2). The following proposition is

a useful tool that allows us to initially disregard any structure in $A^{(P+q)}$ by relating the partial derivative with respect to $C^{(q)}$ to that of $A^{(P+q)}$. Proposition 4.2 is a well-known property of the Khatri–Rao product that we will use in the derivation of the cogradient.

PROPOSITION 4.1. *Let* $F^{(q)} = E \otimes \mathbb{I}_{I_{(P+q)}}$*; then we have that* $\operatorname{vec}(C^{(q)} \cdot E)^{\mathrm{T}} = \operatorname{vec}(C^{(q)})^{\mathrm{T}} \cdot F^{(q)}$*. Furthermore, applying the chain rule leads to*

$$\frac{\partial}{\partial C^{(q)}} = \frac{\partial}{\partial A^{(P+q)}} \cdot E^{\mathrm{T}} \text{ and} \tag{4.3a}$$

$$\frac{\partial}{\partial \operatorname{vec}(C^{(q)})^{\mathrm{T}}} = \frac{\partial}{\partial \operatorname{vec}(A^{(P+q)})^{\mathrm{T}}} \cdot {F^{(q)}}^{\mathrm{T}}. \tag{4.3b}$$

PROPOSITION 4.2 (see, e.g., [5, 52]). *Let* $A^{(n)} \in \mathbb{C}^{I_n \times J}$*,* $n = 1, \ldots, N$*, let* $V = A^{(p_1)} \odot \cdots \odot A^{(p_N)}$*, where* $\boldsymbol{p}$ *is any permutation of* $\{1, \ldots, N\}$*, and let* $W = \overset{N}{\underset{n=1}{*}} {A^{(n)}}^{\mathrm{H}} A^{(n)}$ *be the Hadamard product of all matrices* ${A^{(n)}}^{\mathrm{H}} A^{(n)}$*. Then* $V^{\mathrm{H}} V = W$*.*

COROLLARY 4.3. *Define* $W^{\sigma} \triangleq \overset{N}{\underset{\substack{n=1 \\ n \notin \sigma}}{*}} {A^{(n)}}^{\mathrm{H}} A^{(n)}$*; then* $(V^{\sigma})^{\mathrm{H}} V^{\sigma} = W^{\sigma}$*.*

THEOREM 4.4. *Let* $\boldsymbol{z}$ *be the vector of unknowns* $(\operatorname{vec}(A^{(1)}), \ldots, \operatorname{vec}(A^{(P)}), \operatorname{vec}(C^{(1)}), \ldots, \operatorname{vec}(C^{(Q)}))$*, and then* f_{BTD}*'s complex cogradient is given by*

$$\begin{aligned} \frac{\partial f_{\mathrm{BTD}}}{\partial \boldsymbol{z}} = \Bigg(& \operatorname{vec}\left(\frac{\partial f_{\mathrm{BTD}}}{\partial A^{(1)}}\right), \ldots, \operatorname{vec}\left(\frac{\partial f_{\mathrm{BTD}}}{\partial A^{(P)}}\right), \\ & \operatorname{vec}\left(\frac{\partial f_{\mathrm{BTD}}}{\partial C^{(1)}}\right), \ldots, \operatorname{vec}\left(\frac{\partial f_{\mathrm{BTD}}}{\partial C^{(Q)}}\right)\Bigg), \end{aligned} \tag{4.4}$$

where

$$2\frac{\partial f_{\mathrm{BTD}}}{\partial A^{(p)}} = \overline{A}^{(p)} \cdot W^{\{p\}} - \overline{T}_{(p)} \cdot V^{\{p\}}, \tag{4.5a}$$

$$2\frac{\partial f_{\mathrm{BTD}}}{\partial C^{(q)}} = \overline{C}^{(q)} \cdot E \cdot W^{\{P+q\}} \cdot E^{\mathrm{T}} - \overline{T}_{(P+q)} \cdot V^{\{P+q\}} \cdot E^{\mathrm{T}} \tag{4.5b}$$

for $1 \leq p \leq P$ *and* $1 \leq q \leq Q$*, respectively.*

Proof. Let $f_{\mathrm{BTD}}^{(1)} = \|A^{(n)} \cdot {V^{\{n\}}}^{\mathrm{T}}\|^2$, $f_{\mathrm{BTD}}^{(2)} = \langle T_{(n)}, A^{(n)} \cdot {V^{\{n\}}}^{\mathrm{T}} \rangle$, and $f_{\mathrm{BTD}}^{(3)} = \|T_{(n)}\|^2$, so that $f_{\mathrm{BTD}} = \frac{1}{2}(f_{\mathrm{BTD}}^{(1)} - f_{\mathrm{BTD}}^{(2)} - \overline{f}_{\mathrm{BTD}}^{(2)} + f_{\mathrm{BTD}}^{(3)})$. We have that

$$\begin{aligned} \frac{\partial f_{\mathrm{BTD}}^{(1)}}{\partial A^{(n)}} &= \frac{\partial}{\partial A^{(n)}} \sum_{i=1} \left(\overline{\boldsymbol{v}}_{i:}^{\{n\}} \overline{A}^{(n)^{\mathrm{T}}}\right) \left(A^{(n)} {\boldsymbol{v}_{i:}^{\{n\}}}^{\mathrm{T}}\right) \\ &= \sum_{i=1} \overline{A}^{(n)} {\boldsymbol{v}_{i:}^{\{n\}}}^{\mathrm{H}} \boldsymbol{v}_{i:}^{\{n\}} \\ &= \overline{A}^{(n)} \cdot W^{\{n\}}, \end{aligned}$$

where the last equality follows from Corollary 4.3. Similarly, it can be shown that $\frac{\partial f_{\mathrm{BTD}}^{(2)}}{\partial A^{(n)}} = \overline{T}_{(n)} \cdot V^{\{n\}}$, and trivially that $\frac{\partial \overline{f}_{\mathrm{BTD}}^{(2)}}{\partial A^{(n)}} = \frac{\partial f_{\mathrm{BTD}}^{(3)}}{\partial A^{(n)}} = 0$. Expression (4.5b) follows from (4.5a), (3.6), and Proposition 4.1. ∎

In Table 4.1 we summarize Theorem 4.4 for the special case of a (complex) third-order CPD and rank-$(L_r, L_r, 1)$ BTD. If the decomposition is real, then it is easy to show that the real gradient is twice the expression for the complex cogradient [59].

TABLE 4.1
The (rank-$L_r \circ$ rank-1) BTD objective function's cogradient in the special case of a third-order CPD and a rank-$(L_r, L_r, 1)$ BTD.

Decomposition	Cogradient $\frac{\partial f_{\text{BTD}}}{\partial \boldsymbol{z}}$
third-order CPD	$\frac{1}{2}\begin{bmatrix} \operatorname{vec}(\overline{A}^{(1)}[(A^{(2)\text{H}}A^{(2)}) * (A^{(3)\text{H}}A^{(3)})] - \overline{T}_{(1)}(A^{(3)} \odot A^{(2)})) \\ \operatorname{vec}(\overline{A}^{(2)}[(A^{(1)\text{H}}A^{(1)}) * (A^{(3)\text{H}}A^{(3)})] - \overline{T}_{(2)}(A^{(3)} \odot A^{(1)})) \\ \operatorname{vec}(\overline{A}^{(3)}[(A^{(1)\text{H}}A^{(1)}) * (A^{(2)\text{H}}A^{(2)})] - \overline{T}_{(3)}(A^{(2)} \odot A^{(1)})) \end{bmatrix}$
rank-$(L_r, L_r, 1)$ BTD	$\frac{1}{2}\begin{bmatrix} \operatorname{vec}(\overline{A}^{(1)}[(A^{(2)\text{H}}A^{(2)}) * (E^{\text{T}}C^{(1)\text{H}}C^{(1)}E)] - \overline{T}_{(1)}((C^{(1)}E) \odot A^{(2)})) \\ \operatorname{vec}(\overline{A}^{(2)}[(A^{(1)\text{H}}A^{(1)}) * (E^{\text{T}}C^{(1)\text{H}}C^{(1)}E)] - \overline{T}_{(2)}((C^{(1)}E) \odot A^{(1)})) \\ \operatorname{vec}(\overline{C}^{(1)}E[(A^{(1)\text{H}}A^{(1)}) * (A^{(2)\text{H}}A^{(2)})]E^{\text{T}} - \overline{T}_{(3)}(A^{(2)} \odot A^{(1)})E^{\text{T}}) \end{bmatrix}$

In a line search framework, the next iterate is $\boldsymbol{z}_{k+1} = \boldsymbol{z}_k + \alpha_k \boldsymbol{p}_k$, where the real step length α_k is usually chosen to satisfy the (strong) Wolfe conditions [43, 68]. Line search algorithms are an integral part of quasi-Newton methods but can be difficult to implement. There are several good software implementations available in the public domain, such as from Moré and Thuente [40] and Hager and Zhang [26].

Another approach to select the step is using a trust-region framework, where a region around the current iterate $\boldsymbol{z}_k$ is defined in which the model m_k is trusted to be an adequate representation of the objective function. The next iterate $\boldsymbol{z}_{k+1}$ is then chosen to be the approximate minimizer of the model in this region. In effect, the direction and length of the step is chosen simultaneously. The trust-region radius Δ_k is updated every iteration based on the trustworthiness ρ_k of the model, which is defined as the ratio of the actual reduction $f(\overset{c}{\boldsymbol{z}}_k) - f(\overset{c}{\boldsymbol{z}}_k + \overset{c}{\boldsymbol{p}}_k)$ of the objective function and the predicted reduction $m_k^f(\boldsymbol{0}) - m_k^f(\overset{c}{\boldsymbol{p}}_k)$. The dogleg method [49, 50], double-dogleg method [22], and two-dimensional subspace minimization [6] all attempt to approximately minimize the trust-region subproblem by restricting the step $\boldsymbol{p}$ to (a subset of) the two-dimensional subspace spanned by the steepest descent direction $-\frac{\partial f}{\partial \overline{\boldsymbol{z}}}$ and the quasi-Newton step (4.2) when the model Hessian B_k is positive definite.

Aside from quasi-Newton methods, the cogradient also gives rise to the simple but effective ALS algorithm [7, 20, 27], which is an application of the nonlinear block Gauss–Seidel method. In the latter, the partial gradients (4.5) are alternately set to zero. Each factor matrix $A^{(p)}$ is then computed as the right Moore–Penrose pseudoinverse of the matrix $V^{\{p\}\text{T}}$ applied to the unfolding $T_{(p)}$ and is subsequently used to recompute $V^{\{p\}}$ and $W^{\{p\}}$. This can be done by explicitly solving the associated normal equations as in Algorithm 4.1 or by first decomposing $V^{\{p\}}$ with a QR factorization. The latter is numerically more stable but also more expensive in both memory and flop considering that the normal equations can be computed efficiently using Corollary 4.3. We refer to these two types of updates as fast and accurate, respectively. The difference in accuracy between the two methods often has a negligible effect on convergence speed. Hence, a practical implementation might start by using the fast updates until some convergence criterion is satisfied, after which a few more accurate updates are computed to obtain maximal precision.

Although conceptually simple and often effective, ALS is not guaranteed to converge to a stationary point and is also sensitive to so-called swamps, where convergence is very slow for many iterations. For a given tensor, its best rank-R approximation may not exist. For example, no real $2 \times 2 \times 2$ rank-3 tensor has a best rank-2 approx-

ALGORITHM 4.1. ALTERNATING LEAST SQUARES.

while (not converged) **do**
  for $p = 1, \ldots, P$ **do**
    $A^{(p)} \leftarrow T_{(p)} \cdot \overline{V}^{\{p\}} \cdot \overline{W}^{\{p\}^{-1}}$;
  end
  for $q = 1, \ldots, Q$ **do**
    $C^{(q)} \leftarrow T_{(P+q)} \cdot \overline{V}^{\{P+q\}} \cdot E^{\mathrm{T}} \cdot (E \cdot \overline{W}^{\{P+q\}} \cdot E^{\mathrm{T}})^{-1}$;
  end
end

imation [21]. This ill-posedness is a consequence of the fact that the set of rank-R tensors is not closed, i.e., there exist sequences of rank-R tensors that converge to rank-R^* tensors, where $R < R^*$ [21, 46]. For such sequences, two or more terms grow without bound yet nearly cancel each other out. This behavior is referred to as degeneracy and has been linked with swamps [38, 39], which are observed when either the best rank-R approximation does not exist (strong degeneracy [34, 60, 61]) or when the best rank-R approximation does exist, but the path between the initialization and the solution displays degenerate factors for a certain number of iterations (weak degeneracy [39]). Several approaches to mitigate swamp-like behavior in ALS have been proposed in the literature, among which are (exact) line search [5, 27, 42, 51] and iterated Tikhonov regularization [41]. In contrast, optimization-based algorithms coupled with line search or trust region globalization strategies do guarantee convergence to a stationary point and, depending on the method, spend comparatively fewer iterations in swamps.

4.2. Nonlinear least squares methods. Another approach to solve (3.7) is by means of nonlinear least squares methods such as Gauss–Newton or Levenberg–Marquardt. In these methods, the residual tensor $\mathcal{F}$ is approximated by the linear model

$$m_k^{\mathcal{F}}(\overset{\mathcal{C}}{\boldsymbol{p}}) \triangleq \operatorname{vec}(\mathcal{F}(\overset{\mathcal{C}}{\boldsymbol{z}}_k)) + \frac{\partial \operatorname{vec}(\mathcal{F}(\overset{\mathcal{C}}{\boldsymbol{z}}_k))}{\partial \overset{\mathcal{C}}{\boldsymbol{z}}^{\mathrm{T}}} \overset{\mathcal{C}}{\boldsymbol{p}}, \tag{4.6}$$

where the partial derivative is called the complex Jacobian at $\overset{\mathcal{C}}{\boldsymbol{z}}_k$ and is a straightforward generalization of the complex gradient [59]. The model $m_k^{\mathcal{F}}$ is then used in the modified quadratic model of the objective function

$$m_k^f(\overset{\mathcal{C}}{\boldsymbol{p}}) \triangleq \frac{1}{2}\|m_k^{\mathcal{F}}(\overset{\mathcal{C}}{\boldsymbol{p}})\|^2 + \frac{\lambda_k}{2}\|\boldsymbol{p}\|^2, \tag{4.7}$$

where λ_k is the Levenberg–Marquardt regularization parameter which influences both the length and direction of the step $\boldsymbol{p}$ that minimizes m_k^f. In the Gauss–Newton method, $\lambda_k = 0$ for all k, and a trust-region framework can instead be used to control the length and direction of the step.

By noting that $\mathcal{F}_{\mathrm{BTD}}$ is analytic in its argument, we need only work with half of the complex Jacobian, since the conjugate Jacobian $\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \overline{\boldsymbol{z}}^{\mathrm{T}}}$ is identically equal to zero. As a consequence, the quadratic model $m_k^{f_{\mathrm{BTD}}}$ is minimized by the Levenberg–

Marquardt step

$$\boldsymbol{p}_k^* = -\begin{bmatrix} J_k \\ \sqrt{\lambda_k}\mathbb{I} \end{bmatrix}^\dagger \begin{bmatrix} \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}}(\boldsymbol{z}_k)) \\ \mathbf{0} \end{bmatrix}, \tag{4.8}$$

where J_k is the Jacobian $\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \boldsymbol{z}^{\mathrm{T}}}$ at the kth iterate $\boldsymbol{z}_k$ and the identity matrix and zero vector are of the appropriate dimensions. Due to the scaling indeterminacy of the decomposition, the Jacobian is rank deficient and a certain number of its singular values are equal to zero. In general, each of the R terms contains a Qth-order rank-one tensor and a Pth-order rank-L_r tensor, which contribute $(Q-1)R$ and $(P-1)R'$ degrees of freedom, respectively. Furthermore, there is one more degree of freedom in the scaling between the rank-one and rank-L_r part of each term, bringing the total number of singular values equal to zero to at least $QR + (P-1)R'$. Note that this is just a lower bound and that the actual amount of singular values equal to zero depends on the unicity of the individual terms. For example, when $P = 2$, each term contains a rank-L_r matrix, the factors of which are only unique up to a linear transformation. In the Levenberg–Marquardt algorithm, the Jacobian is allowed to be singular since each of the steps are regularized by a norm constraint on the solution. However, in a Gauss–Newton trust-region algorithm, special care must be taken to invert the Jacobian in a meaningful way. One approach is to compute the Moore–Penrose pseudoinverse $J_k^\dagger$. Another is to compute an approximate solution with a truncated conjugate gradient algorithm, where the amount of regularization is controlled by the number of iterations. As we will see, the latter method allows for significant savings in both memory and computational cost.

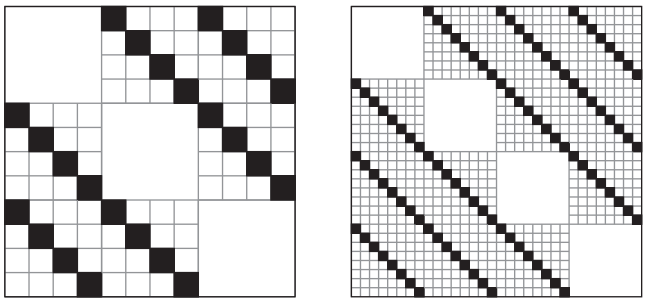
Storing the Jacobian as a dense matrix is impractical as each of its columns requires as much memory as the tensor $\mathcal{T}$ itself. A sparse representation is one way to store the Jacobian more efficiently but is likely of limited use due to a large amount of fill-in appearing when applying the QR factorization to solve (4.8) [44, 64]. At the cost of squaring the condition number of the system, the memory requirement can be reduced by solving the normal equations

$$(J_k^{\mathrm{H}} J_k + \lambda_k \mathbb{I})\boldsymbol{p}_k^* = -J_k^{\mathrm{H}} \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}}(\boldsymbol{z}_k)) = -2\frac{\partial f_{\mathrm{BTD}}}{\partial \overline{\boldsymbol{z}}}(\boldsymbol{z}_k), \tag{4.9}$$

where the last equality follows from the analyticity of $\mathcal{F}_{\mathrm{BTD}}$ [59], associated with (4.8) and looking for an expression for the Gramian $J^{\mathrm{H}}J$ [63]. In the real case, the latter matrix represents an approximation of the objective function's Hessian. The error of the approximation is the sum [43]

$$\sum_{i_1,\ldots,i_N=1}^{I_1,\ldots,I_N} (\mathcal{F}_{\mathrm{BTD}})_{i_1\ldots i_N} \frac{\partial (\mathcal{F}_{\mathrm{BTD}})_{i_1\ldots i_N}}{\partial \boldsymbol{z} \partial \boldsymbol{z}^{\mathrm{T}}}$$

and hence is small when the residuals $\mathcal{F}_{\mathrm{BTD}}$ are small, or when $\mathcal{F}_{\mathrm{BTD}}$ is nearly linear in its argument. Fortunately, the multilinear structure of $\mathcal{F}_{\mathrm{BTD}}$ ensures that the total contribution of the Hessians $\frac{\partial (\mathcal{F}_{\mathrm{BTD}})_{i_1\ldots i_N}}{\partial \boldsymbol{z} \partial \boldsymbol{z}^{\mathrm{T}}}$ is quite sparse, depending on the number of rank-one terms. In fact, one can show that the relative number of nonzero elements in the superimposed Hessian is equal to $\frac{N(N-1)RI^2}{(NRI)^2} = \frac{N-1}{NR}$ for a CPD of an Nth-order tensor of dimensions $I \times \cdots \times I$ in R rank-one terms (cf. Figure 4.1). Moreover, it can be observed that these nonzero elements are often relatively small in comparison to the elements of $J^{\mathrm{H}}J$. It may therefore be expected that, as the residuals



(a) $N = 3$, $R = 4$ (b) $N = 4$, $R = 8$

FIG. 4.1. *Sparsity pattern of the total contribution of the Hessians* $\frac{\partial (\mathcal{F}_{\mathrm{BTD}})_{i_1 \dots i_N}}{\partial \boldsymbol{z} \partial \boldsymbol{z}^{\mathrm{T}}}$ *in the case of a real CPD of a (hyper)cubical third-order tensor (left) and fourth-order tensor (right).*

$(\mathcal{F}_{\mathrm{BTD}})_{i_1 \dots i_N}$ decrease, the Gramian $J^{\mathrm{H}}J$ rapidly becomes a good approximation to the real Hessian. In Theorem 4.5 we derive the structure of $J^{\mathrm{H}}J$: it is a block matrix consisting of diagonal and low-rank blocks.

THEOREM 4.5. *Let J be the Jacobian $\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \boldsymbol{z}^{\mathrm{T}}}$ and let $F^{(q)}$ be defined as in Proposition* 4.1*; then the Gramian $J^{\mathrm{H}}J$ is given by*

$$J^{\mathrm{H}}J = \Sigma \cdot \begin{bmatrix} \Pi^{(1,1)} & \cdots & \Pi^{(1,N)} \\ \vdots & \ddots & \vdots \\ \Pi^{(N,1)} & \cdots & \Pi^{(N,N)} \end{bmatrix} \cdot \Sigma^{\mathrm{T}}, \tag{4.10}$$

where $\Sigma \triangleq \operatorname{diag}(\mathbb{I}_{I_1 R'}, \dots, \mathbb{I}_{I_P R'}, F^{(1)}, \dots, F^{(Q)})$ *and*

(4.11)

$$\Pi^{(n_1,n_2)} \triangleq \left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \operatorname{vec}(A^{(n_1)})^{\mathrm{T}}} \right)^{\mathrm{H}} \left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \operatorname{vec}(A^{(n_2)})^{\mathrm{T}}} \right)$$

$$= \begin{cases} W^{\{n\}} \otimes \mathbb{I}_{I_n} & n = n_1 = n_2 \\ \begin{bmatrix} w_{1,1}^{\{n_1,n_2\}} \boldsymbol{a}_1^{(n_1)} \boldsymbol{a}_1^{(n_2)\mathrm{H}} & \cdots & w_{1,R'}^{\{n_1,n_2\}} \boldsymbol{a}_{R'}^{(n_1)} \boldsymbol{a}_1^{(n_2)\mathrm{H}} \\ \vdots & \ddots & \vdots \\ w_{R',1}^{\{n_1,n_2\}} \boldsymbol{a}_1^{(n_1)} \boldsymbol{a}_{R'}^{(n_2)\mathrm{H}} & \cdots & w_{R',R'}^{\{n_1,n_2\}} \boldsymbol{a}_{R'}^{(n_1)} \boldsymbol{a}_{R'}^{(n_2)\mathrm{H}} \end{bmatrix} & \text{otherwise.} \end{cases}$$

Proof. Let $\boldsymbol{e}_i^{(n)}$ be the ith column of the identity matrix $\mathbb{I}_{I_n}$; then

$$\frac{\partial \mathcal{F}_{\mathrm{BTD}}}{\partial a_{i_n r}^{(n)}} = \boldsymbol{a}_r^{(1)} \circ \cdots \circ \boldsymbol{a}_r^{(n-1)} \circ \boldsymbol{e}_{i_n}^{(n)} \circ \boldsymbol{a}_r^{(n+1)} \circ \cdots \circ \boldsymbol{a}_r^{(N)}.$$

Diagonal blocks. For $n = n_1 = n_2$ we have

$$\left\langle \frac{\partial \mathcal{F}_{\mathrm{BTD}}}{\partial a_{i r_1}^{(n)}}, \frac{\partial \mathcal{F}_{\mathrm{BTD}}}{\partial a_{j r_2}^{(n)}} \right\rangle = \begin{cases} w_{r_1 r_2}^{\{n\}} & i = j, \\ 0 & \text{otherwise.} \end{cases}$$

and so

$$\left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \boldsymbol{a}_{r_1}^{(n)\mathrm{T}}} \right)^{\mathrm{H}} \left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \boldsymbol{a}_{r_2}^{(n)\mathrm{T}}} \right) = w_{r_1 r_2}^{\{n\}} \mathbb{I}_{I_n}.$$

Off-diagonal blocks. For $n_1 \neq n_2$ we have

$$\left\langle \frac{\partial \mathcal{F}_{\text{BTD}}}{\partial a^{(n_1)}_{i_{n_1} r_1}}, \frac{\partial \mathcal{F}_{\text{BTD}}}{\partial a^{(n_2)}_{i_{n_2} r_2}} \right\rangle = w^{\{n_1,n_2\}}_{r_1 r_2} a^{(n_1)}_{i_{n_1} r_2} \overline{a}^{(n_2)}_{i_{n_2} r_1},$$

and so

$$\left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\text{BTD}})}{\partial \boldsymbol{a}^{(n_1)\text{T}}_{r_1}} \right)^{\text{H}} \left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\text{BTD}})}{\partial \boldsymbol{a}^{(n_2)\text{T}}_{r_2}} \right) = w^{\{n_1,n_2\}}_{r_1 r_2} \boldsymbol{a}^{(n_1)}_{r_2} \boldsymbol{a}^{(n_2)\text{H}}_{r_1}.$$

The $(p, P+q)$th block in $J^{\text{H}}J$ is defined as $(\frac{\partial \operatorname{vec}(\mathcal{F}_{\text{BTD}})}{\partial \operatorname{vec}(A^{(p)})^{\text{T}}})^{\text{H}}(\frac{\partial \operatorname{vec}(\mathcal{F}_{\text{BTD}})}{\partial \operatorname{vec}(C^{(q)})^{\text{T}}})$ and can be computed from $\Pi^{(p,P+q)}$ using (4.3b) as $\Pi^{(p,P+q)} \cdot F^{(q)\text{T}}$. Similarly, the $(P+q, p)$th and $(P+q_1, P+q_2)$th blocks can be computed as $F^{(q)} \cdot \Pi^{(p,P+q)}$ and $F^{(q_1)} \cdot \Pi^{(P+q_1,P+q_2)} \cdot F^{(q_2)\text{T}}$, respectively. ∎

Because $\mathcal{F}_{\text{BTD}}$ is analytic in its argument, the expression for $J^{\text{H}}J$ is nearly the same as in the real case. Its structure has been noticed before for the real CPD [44], and several methods of exploiting it have been proposed in the literature, for instance, by efficient construction of the matrix [63] or by writing it as a block-diagonal matrix plus a low-rank correction [47]. We propose to exploit its structure by storing the matrix implicitly as the collection of factor matrices $A^{(n)}$ and the factor matrices' Gramians $A^{(n)\text{H}}A^{(n)}$, from which the Hadamard products $W^{\{n\}}$ and $W^{\{n_1,n_2\}}$ can easily be reconstructed. Storing the Jacobian's Gramian in this way is not appropriate for direct solvers but does admit the use of inexact adaptations of the Levenberg–Marquardt and Gauss–Newton algorithms by computing only matrix-vector products. The latter methods only partially invert the Jacobian's Gramian using a limited number of (preconditioned) conjugate gradient iterations [43]. Besides the significant reduction in memory cost, the following theorem shows that the Gramian's structure also allows for an efficient matrix-vector product. Compared to a dense matrix-vector product, Theorem 4.6 can reduce the computational complexity from $O(N^2R^2I^2)$ to $O(NR^2I)$ flop per matrix-vector product, depending on the implementation.

THEOREM 4.6. *Let* $\boldsymbol{p} = (\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(P)}, \boldsymbol{d}^{(1)}, \ldots, \boldsymbol{d}^{(Q)})$, *where* $B^{(p)} \in \mathbb{C}^{I_p \times R'}$ *and* $\boldsymbol{b}^{(p)} = \operatorname{vec}(B^{(p)})$, $p = 1, \ldots, P$, $D^{(q)} \in \mathbb{C}^{I_{P+q} \times R}$, *and* $\boldsymbol{d}^{(q)} = \operatorname{vec}(D^{(q)})$, $q = 1, \ldots, Q$. *Furthermore, we define* $B^{(P+q)} \triangleq D^{(q)} \cdot E$ *and* $\boldsymbol{b}^{(P+q)} \triangleq \operatorname{vec}(B^{(P+q)})$. *The matrix-vector product* $J^H J \cdot \boldsymbol{p}$ *then follows from*

$$\Pi^{(p,p)} \cdot \boldsymbol{b}^{(p)} = \operatorname{vec}(X^{(p)}), \tag{4.12a}$$

$$\Pi^{(p_1,p_2)} \cdot \boldsymbol{b}^{(p_2)} = \operatorname{vec}(Y^{(p_1,p_2)}), \tag{4.12b}$$

$$(F^{(q)} \cdot \Pi^{(P+q,p)}) \cdot \boldsymbol{b}^{(p)} = \operatorname{vec}(Y^{(P+q,p)} \cdot E^{\text{T}}), \tag{4.12c}$$

$$(\Pi^{(p,P+q)} \cdot F^{(q)\text{T}}) \cdot \boldsymbol{d}^{(q)} = \operatorname{vec}(Y^{(p,P+q)}), \tag{4.12d}$$

$$(F^{(q_1)} \cdot \Pi^{(P+q_1,P+q_2)} \cdot F^{(q_2)\text{T}}) \cdot \boldsymbol{d}^{(q_2)} = \operatorname{vec}(Y^{(P+q_1,P+q_2)} \cdot E^{\text{T}}), \tag{4.12e}$$

$$(F^{(q)} \cdot \Pi^{(P+q,P+q)} \cdot F^{(q)\text{T}}) \cdot \boldsymbol{d}^{(q)} = \operatorname{vec}(X^{(P+q)} \cdot E^{\text{T}}), \tag{4.12f}$$

where

$$X^{(n)} \triangleq B^{(n)} \cdot \overline{W}^{\{n\}}, \tag{4.13}$$

$$Y^{(n_1,n_2)} \triangleq A^{(n_1)} \cdot (\overline{W}^{\{n_1,n_2\}} * (B^{(n_2)\text{T}} \cdot \overline{A}^{(n_2)})), \text{ and} \tag{4.14}$$

$1 \leq p_1 \neq p_2 \leq P$, $1 \leq q_1 \neq q_2 \leq Q$, *and* $1 \leq n, n_1 \neq n_2 \leq N$.

Proof. We have that $\Pi^{(p,p)} \cdot \boldsymbol{b}^{(p)} = (W^{\{p\}} \otimes \mathbb{I}_{I_p}) \cdot \mathrm{vec}(B^{(p)}) = \mathrm{vec}(B^{(p)} \cdot W^{\{p\}\mathrm{T}}) = \mathrm{vec}(B^{(p)} \cdot \overline{W}^{\{p\}})$. A similar structure is present in $\Pi^{(p_1,p_2)} \cdot \boldsymbol{b}^{(p_2)}$. Let $Z = W^{\{p_1,p_2\}} * (A^{(p_2)\mathrm{H}} B^{(p_2)})$; then it is not hard to show that $\Pi^{(p_1,p_2)} \cdot \boldsymbol{b}^{(p_2)} = (Z \otimes \mathbb{I}_{I_{p_1}}) \cdot \mathrm{vec}(A^{(p_1)})$, from which (4.12b) follows. The special cases (4.12c)–(4.12f) are a direct result of (4.12a)–(4.12b) and Proposition 4.1. ∎

The convergence rate of the conjugate gradient algorithm depends on how well the system's eigenvalues are clustered and is consequently also influenced by its condition number. In the preconditioned conjugate gradient (PCG) algorithm, the eigenvalue distribution is improved by solving a system of the form

$$M^{-1} \cdot J^{\mathrm{H}} J \cdot \boldsymbol{p} = M^{-1} \cdot \left(-2 \frac{\partial f_{\mathrm{BTD}}}{\partial \overline{\boldsymbol{z}}} \right),$$

where M is a symmetric positive definite matrix called the preconditioner. The inverse of the preconditioner M^{-1} should be cheap to apply and is often designed so that $M^{-1} \cdot J^{\mathrm{H}} J \approx \mathbb{I}$. The block Jacobi preconditioner

$$M_{\mathrm{BJ}} = \Sigma \cdot \begin{bmatrix} \Pi^{(1,1)} & & \\ & \ddots & \\ & & \Pi^{(N,N)} \end{bmatrix} \cdot \Sigma^{\mathrm{T}} \tag{4.15}$$

is one such example. It is a block-diagonal approximation of (4.10) and can be inverted efficiently using (4.12a) and (4.12f). For instance, the solution of the system $\Pi^{(p,p)} \cdot \mathrm{vec}(X) = \mathrm{vec}(Y)$ can be computed by solving the much smaller system $X = Y \cdot \overline{W}^{\{p\}-1}$. It is interesting to note that if $M_{\mathrm{BJ}}^{-1} \cdot J^{\mathrm{H}} J = \mathbb{I}$, the computed step $\boldsymbol{p}$ amounts to a simultaneous version of the fast ALS updates of Algorithm 4.1. Indeed, we then have that $\boldsymbol{p} = M_{\mathrm{BJ}}^{-1} \cdot (-2 \frac{\partial f_{\mathrm{BTD}}}{\partial \overline{\boldsymbol{z}}})$, of which the pth component $B^{(p)}$ is given by

$$B^{(p)} = -2 \frac{\partial f_{\mathrm{BTD}}}{\partial \overline{A^{(p)}}} \cdot \overline{W}^{\{p\}^{-1}} = -A^{(p)} + T_{(p)} \cdot \overline{V}^{\{p\}} \cdot \overline{W}^{\{p\}^{-1}}.$$

The pth factor matrix of the next iterate $A^{(p)} + B^{(p)}$ is hence equal to that obtained by a fast ALS update in which the updated factor matrices are not used to recompute $V^{\{p\}}$ or $W^{\{p\}}$. This preconditioner adds only $O(\frac{1}{3}NR^3 + 2NR^2I)$ flop per conjugate gradient iteration, which is relatively cheap in comparison to the computation of the scaled conjugate cogradient that acts as the right-hand side of the linear system. In summary, the initial solution computed by PCG with a block Jacobi preconditioner is similar to that of an ALS update, and the effect of the off-diagonal blocks is subsequently taken into account in an iterative manner. In this light, these matrix-free nonlinear least squares methods can be viewed as a refinement of the alternating least squares algorithm with simultaneous updates. The numerical experiments show that this refinement pays off for difficult problems and is still competitive with ALS for simpler problems.

4.3. Computational complexity. In section 5 we compare the robustness and efficiency of ALS, quasi-Newton methods (nonlinear conjugate gradient and L-BFGS with Moré–Thuente line search, L-BFGS with dogleg trust-region (L-BFGS-DL) [49, 50]), and inexact nonlinear least squares methods (inexact Gauss–Newton with CG-Steihaug [62] or dogleg trust region, inexact Levenberg–Marquardt) on complex block term and canonical polyadic decompositions. The comparison is based on performance profiles [23] with the number of flop, expressed as an equivalent number of

TABLE 4.2

Computational complexity of several BTD algorithms in flop/iteration in case of a real CPD of an Nth-order $I_1 \times \cdots \times I_N$ tensor in R rank-one terms. From top to bottom, the algorithms are alternating least squares, limited-memory BFGS with dogleg trust region, nonlinear conjugate gradient and limited-memory BFGS with Moré–Thuente line search, Levenberg–Marquardt, Gauss–Newton with dogleg trust region, inexact Levenberg–Marquardt and inexact Gauss–Newton with CG-Steihaug, or dogleg trust region. (Note that for CG-MT, $m \equiv 1$.)

Algorithm	Complexity (flop/iteration)
ALS	$\mathcal{O}(2(N+1)R\prod I_n)$
L-BFGS-DL	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{dl}})R\prod I_n + 8mR\sum I_n)$
{CG,L-BFGS}-MT	$\mathcal{O}(\mathrm{it}_{\mathrm{mt}}2(N+1)R\prod I_n + 8mR\sum I_n)$
LM	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{lm}})R\prod I_n + \mathrm{it}_{\mathrm{lm}}\frac{1}{3}R^3(\sum I_n)^3)$
GN-DL	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{gn}})R\prod I_n + \frac{8}{3}R^3(\sum I_n)^3)$
LM (inexact)	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{lm}})R\prod I_n + \mathrm{it}_{\mathrm{lm}}\mathrm{it}_{\mathrm{cg}}(\frac{5}{2}N^2R^2 + \frac{1}{3}NR^3 + 8R^2\sum I_n))$
GN-{CGS,DL} (inexact)	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{gn}})R\prod I_n + \mathrm{it}_{\mathrm{cg}}(\frac{5}{2}N^2R^2 + \frac{1}{3}NR^3 + 8R^2\sum I_n))$

objective function evaluations, as a metric for the amount of computational resource required by solver s to solve problem p. Table 4.2 gives an overview of the amount of flop per iteration for the above algorithms (cf. Appendix A for the derivation), where it is assumed that the inexact nonlinear least squares methods use a block Jacobi preconditioner. For most algorithms, a significant part of the cost in each iteration is to compute the gradient of the objective function. This cost can be reduced by a factor N by either exploiting parallelism or, as recently proposed, by recursive computation [48]. We use Table 4.2 to estimate the total amount of flop given the iteration counts and then divide by the number of flop equivalent to one function evaluation to obtain an equivalent number of function evaluations. For example, alternating least squares requires a number of flop per iteration equivalent to the cost of $N+1$ function evaluations.

Aside from the substantial reduction in flop that the inexact nonlinear least squares methods offer, they also require significantly less memory in comparison to their exact counterparts. The exact methods need $\mathcal{O}(N^2R^2I^2)$ memory cells to store the Jacobian's Gramian, while the inexact methods only require $\mathcal{O}(NR^2)$ memory cells, the same amount of memory that the alternating least squares algorithm requires. Furthermore, it is to be expected that the Gauss–Newton with dogleg trust-region strategy is more efficient than the Levenberg–Marquardt method; the former only solves one system involving the Jacobian's Gramian per iteration, while the latter solves such a system at every inner iteration. We also note that nonlinear conjugate gradient [2, 45] and exact Levenberg–Marquardt [28, 44, 47, 63] have been applied to the real CPD before. For an overview and comparison of existing algorithms for the CPD, see [10] and [64], respectively.

5. Numerical experiments. We compare the relative performance of the algorithms in Table 4.2 using the performance profiles proposed by Dolan and Moré [23]. In a first set of experiments, we are interested in the performance on complex rank-$(L_r, L_r, 1)$ block term decompositions. In the second set of experiments, we look at the performance on complex fourth-order canonical polyadic decompositions. The set of solvers S consists of the algorithms in Table 4.2, excluding the exact nonlinear least squares methods. For each experiment, we generate 50 sets of factor matrices which are used to generate the to-be decomposed tensors. Another 50 sets of factor matrices of the same dimensions are generated to be used as initialization for the algorithms. The set of problems P includes all 2500 {tensor, initialization} pairs. The amount

of work required by solver s to solve problem p is denoted by $t_{p,s}$ and is measured as an equivalent number of objective function evaluations (cf. section 4.3). If solver s did not successfully solve problem p, we set $t_{p,s} = \infty$. The performance profile for solver s,

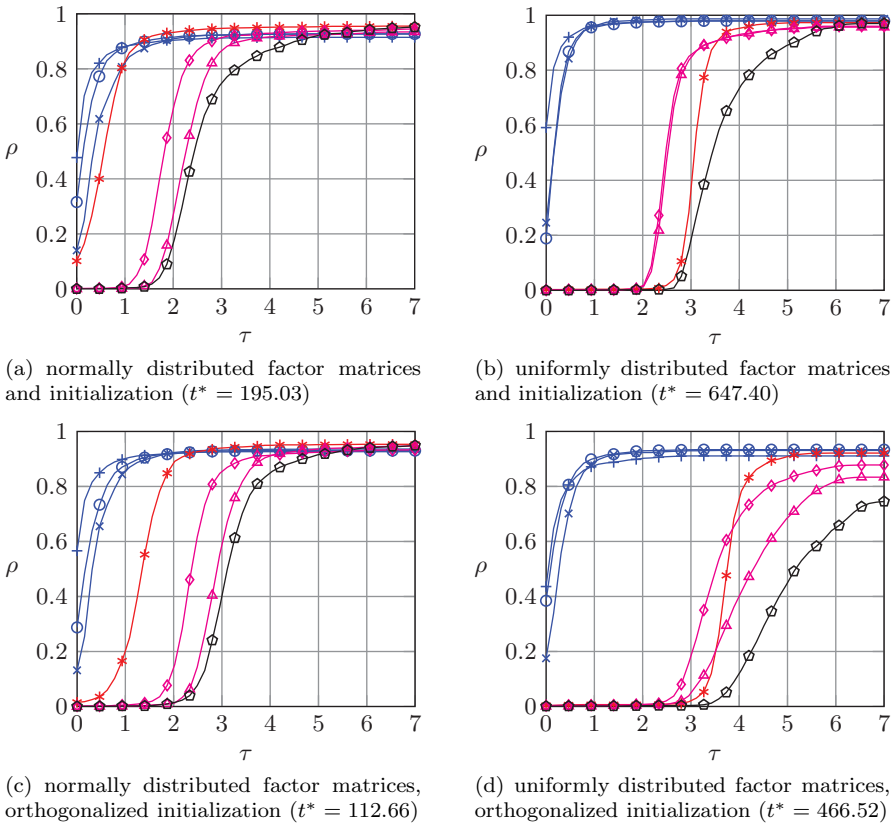
$$\rho_s(\tau) = \frac{|\{p \in P : t_{p,s} \leq 2^\tau \cdot (\min_{s \in S} t_{p,s})\}|}{|P|},$$

depicts the fraction of problems that solver s was able to solve with a required amount of work within a factor 2^τ of the most efficient solver for that problem. For the inexact nonlinear least squares methods, we use the block Jacobi preconditioner described in section 4.2 and have opted to truncate the inner CG iterations after a maximum of 10 iterations or when a relative residual of 10^{-4} is obtained. We have experimented with other preconditioners such as Jacobi and symmetric successive over-relaxation (SSOR) but found these to be less efficient than the former. We let L-BFGS use a maximum of $m = 30$ rank-two updates to represent the approximate Hessian. The maximum number of iterations is 5000 for all algorithms except the inexact nonlinear least squares methods, which are allowed a maximum of 500 iterations per problem. The stop criteria are identical for all algorithms and are such that an algorithm stops if either the relative change between two successive iterates is less than a specified tolerance ϵ_z, or if the change in objective function value relative to the norm of the tensor is less than a specified tolerance ϵ_f. Formally, these conditions are described by $\|\boldsymbol{z}_k - \boldsymbol{z}_{k-1}\| \leq \epsilon_z \|\boldsymbol{z}_{k-1}\|$ and $f_{\text{BTD}}(\boldsymbol{z}_{k-1}) - f_{\text{BTD}}(\boldsymbol{z}_k) \leq \epsilon_f^2 \frac{1}{2} \|T\|^2$, respectively. All experiments were performed in MATLAB 7.14 (R2012a) on two hexacore Intel Xeon E5645 CPUs with 48 GB RAM. The underlying complex optimization algorithms were validated independently on different optimization problems and then adapted for the (rank-$L_r \circ$ rank-1) BTD.

5.1. Computing the rank-$(L_r, L_r, 1)$ block term decomposition.

5.1.1. Without noise. For our first set of experiments, we generate 50 sets of factor matrices of which the real and imaginary part of the entries are drawn from either the standard normal distribution or from a uniform distribution in the interval $(0, 1)$. We refer to these two types of factor matrices as normally distributed and uniformly distributed factor matrices, respectively. Each set of factor matrices corresponds to a rank-$(L_r, L_r, 1)$ BTD in three terms, where $L_1 = L_2 = L_3 = 3$. For each set, we generate its associated full $10 \times 11 \times 12$ tensor, which is then scaled such that it has unit norm. Note that these decompositions are generically (essentially) unique since the factor matrices are generically full column rank [13]. For each tensor, another 50 sets of factor matrices of the same dimensions are generated using the same distribution as that of the factor matrices that generate the BTD. A second set of initializations is obtained by orthogonalizing pseudorandomly generated factor matrices with a QR factorization, which is a popular initialization method for ALS. We set $\epsilon_z = \epsilon_f = 10^{-12}$ and say that a decomposition is successful if and only if $\|\mathcal{F}_{\text{BTD}}(\boldsymbol{z}^*_{p,s})\| \leq 10^{-8}$, where $\boldsymbol{z}^*_{p,s}$ is the solution computed by solver s for problem p.

The results of the first four experiments are shown in Figure 5.1. Performance profiles are useful for judging relative difference in performance between algorithms on a set of problems but do not say anything about the difficulty of those problems. For each experiment, we use the median number of function evaluations of the best solver for each problem $t^* \triangleq \text{median}_{p \in P}(\min_{s \in S} t_{p,s})$ as a metric for the difficulty of the experiment's set of problems. Comparing Figure 5.1(b) with 5.1(d) we see



(a) normally distributed factor matrices and initialization ($t^* = 195.03$)

(b) uniformly distributed factor matrices and initialization ($t^* = 647.40$)

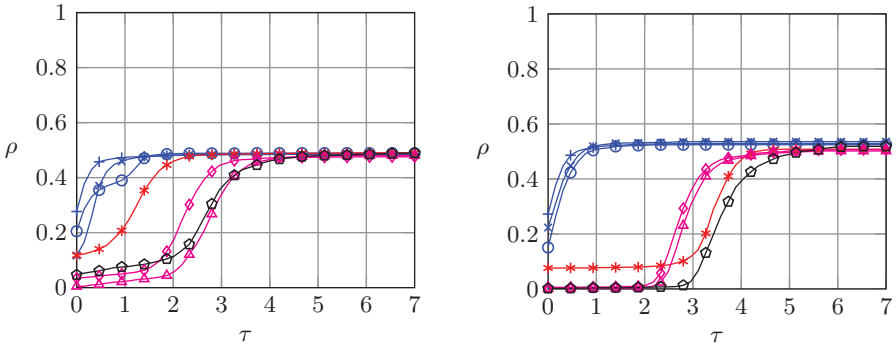
(c) normally distributed factor matrices, orthogonalized initialization ($t^* = 112.66$)

(d) uniformly distributed factor matrices, orthogonalized initialization ($t^* = 466.52$)

FIG. 5.1. *Performance profiles for complex rank-$(L_r, L_r, 1)$ block term decompositions, where $L_1 = L_2 = L_3 = 3$, of tensors of dimensions $10 \times 11 \times 12$. Legend: GN-DL (—+—), LM (—×—), GN-CGS (—○—), ALS (—✱—), L-BFGS-DL (—△—), L-BFGS-MT (—◇—), and NCG-MT (—⬠—).*

that the quasi-Newton methods can be quite sensitive to the type of initialization. Paired with a uniformly distributed initialization they can require up to 16 times more flop than the most efficient solver, while with an orthogonalized initialization it can be up to 32 times. Second, the fraction of tensors successfully decomposed by the quasi-Newton methods, visible on the far right of each figure, decreases from about 95% to less than 90%. Its relatively poor performance is likely due to its limited ability of approximating the objective function's Hessian and possibly also because it is known to generate poorly scaled search directions. Although the relative performance between ALS and the nonlinear least squares methods seems relatively robust w.r.t. the type of initialization, using orthogonalized factor matrices as initialization for tensors generated using uniformly distributed factor matrices does increase the overall required amount of work. The nonlinear least squares methods are the most efficient in each of the experiments, and they excel most for difficult problems such as in the uniformly distributed factor matrices case.

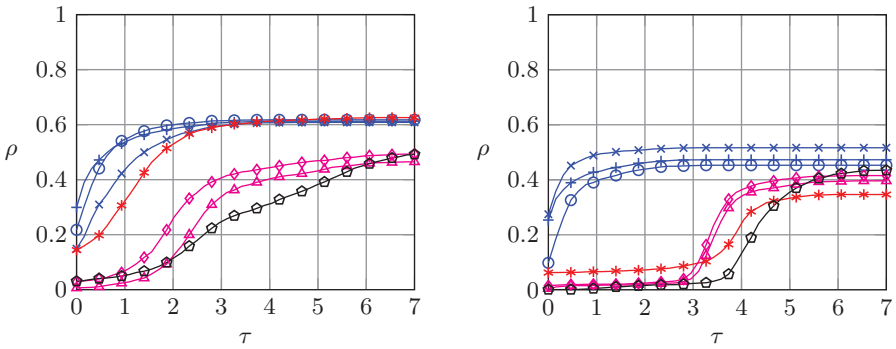
Now we examine what happens when the terms have different ranks. The following experiments are identical in setup to the previous experiments, except we generate the tensors using $L_1 = 4$ and $L_2 = 5$ (cf. Figure 5.2) and $L_1 = 1$, $L_2 = 3$, and $L_3 = 5$ (cf. Figure 5.3). We omit the orthogonalized initialization as it has an adverse effect on the general unconstrained optimization methods. Two terms of different rank can



(a) normally distributed factor matrices and initialization ($t^* = 242.72$)

(b) uniformly distributed factor matrices and initialization ($t^* = 830.36$)

FIG. 5.2. *Performance profiles for complex rank-$(L_r, L_r, 1)$ block term decompositions, where $L_1 = 4$ and $L_2 = 5$, of tensors of dimensions $10 \times 11 \times 12$. Legend: GN-DL (—+—), LM (—×—), GN-CGS (—○—), ALS (—✳—), L-BFGS-DL (—△—), L-BFGS-MT (—◇—), and NCG-MT (—⬠—).*



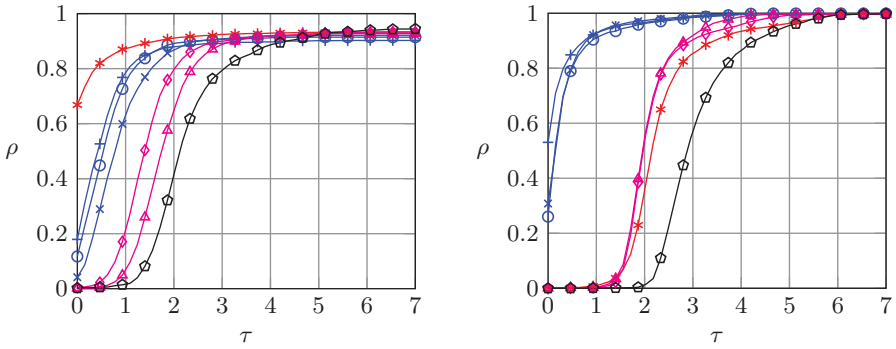
(a) normally distributed factor matrices and initialization ($t^* = 465.44$)

(b) uniformly distributed factor matrices and initialization ($t^* = 983.35$)

FIG. 5.3. *Performance profiles for complex rank-$(L_r, L_r, 1)$ block term decompositions, where $L_1 = 1$, $L_2 = 3$, and $L_3 = 5$, of tensors of dimensions $10 \times 11 \times 12$. Legend: GN-DL (—+—), LM (—×—), GN-CGS (—○—), ALS (—✳—), L-BFGS-DL (—△—), L-BFGS-MT (—◇—), and NCG-MT (—⬠—).*

create additional local minima. For example, in Figure 5.2, tensors are generated as the sum of a rank-$(4, 4, 1)$ term and a rank-$(5, 5, 1)$ term. Initially, neither of the two terms are strongly inclined to converge to a specific term of the decomposition and so we may expect that the rank-$(4, 4, 1)$ term of the initialization will start to converge to the rank-$(5, 5, 1)$ term of the decomposition and vice versa for roughly half of the initializations. Such a solution represents a local minimum because there is no way for the two terms to "cross" each other without increasing the objective function value and explains why the fraction of successful decompositions is much lower than in Figure 5.1.

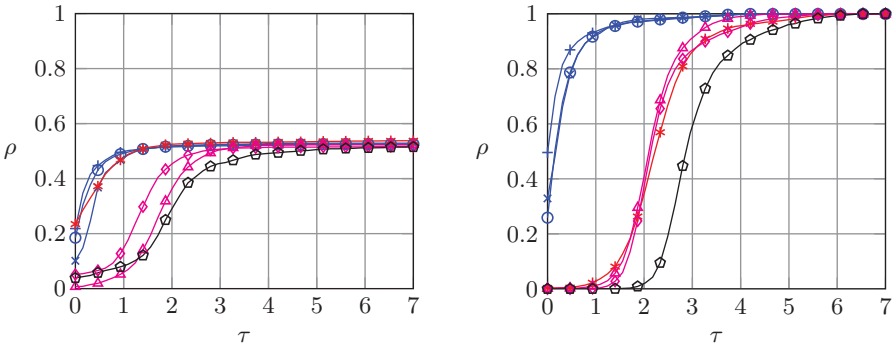
5.1.2. With noise. In the following experiments, we add complex Gaussian noise to the tensors before decomposing them. A noise term $\mathcal{N}$, of which the real and imaginary part are drawn from the standard normal distribution, is generated for each tensor. Each solver is then to decompose a tensor of the form $\mathcal{T} + \sigma\mathcal{N}$, where the scalar σ is chosen so that the signal-to-noise ratio (SNR) $20 \log_{10}(\|\mathcal{T}\| \|\mathcal{N}\|^{-1})$ is 25 dB. We



(a) normally distributed factor matrices and initialization ($t^* = 136.00$)

(b) uniformly distributed factor matrices and initialization ($t^* = 351.85$)

FIG. 5.4. *Performance profiles for complex rank-$(L_r, L_r, 1)$ block term decompositions, where $L_1 = L_2 = L_3 = 3$, of noisy tensors of dimensions $10 \times 11 \times 12$ with an SNR of 25 dB. Legend: GN-DL (—+—), LM (—×—), GN-CGS (—○—), ALS (—✱—), L-BFGS-DL (—△—), L-BFGS-MT (—◇—), and NCG-MT (—⬠—).*

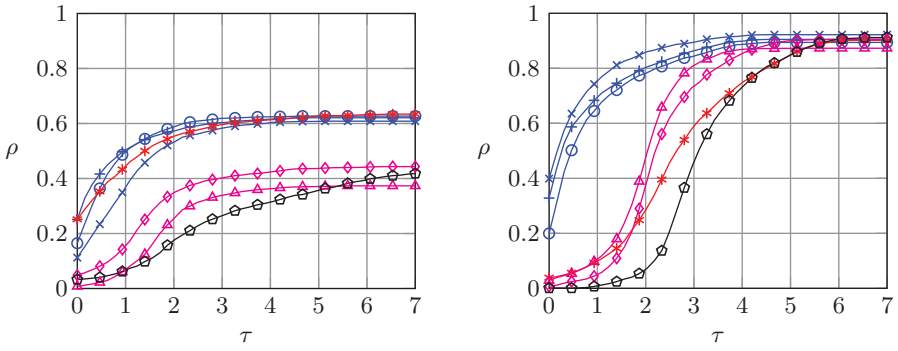


(a) normally distributed factor matrices and initialization ($t^* = 177.41$)

(b) uniformly distributed factor matrices and initialization ($t^* = 424.69$)

FIG. 5.5. *Performance profiles for complex rank-$(L_r, L_r, 1)$ block term decompositions, where $L_1 = 4$ and $L_2 = 5$, of noisy tensors of dimensions $10 \times 11 \times 12$ with an SNR of 25 dB. Legend: GN-DL (—+—), LM (—×—), GN-CGS (—○—), ALS (—✱—), L-BFGS-DL (—△—), L-BFGS-MT (—◇—), and NCG-MT (—⬠—).*

set $\epsilon_z = \epsilon_f = 10^{-5}$, which is necessary to avoid premature termination due to swamp-like behavior. We say a decomposition is successful if and only if the approximate tensor $\mathcal{T}_s$ computed by solver s satisfies $20 \log_{10}(\|\mathcal{T}\| \|\mathcal{T} - \mathcal{T}_s\|^{-1}) \geq 25\,\text{dB}$. As in the noiseless case, we examine rank-$(L_r, L_r, 1)$ block term decompositions, where $L_1 = L_2 = L_3 = 3$ (cf. Figure 5.4), $L_1 = 4$ and $L_2 = 5$ (cf. Figure 5.5), and $L_1 = 1$, $L_2 = 3$, and $L_3 = 5$ (cf. Figure 5.6). Perhaps surprisingly, the fraction of successful decompositions can be significantly higher in the noisy case. For instance, compare the fraction of successful decompositions in Figure 5.2(b), which is close to 50%, with the noisy case in Figure 5.5(b), which is 100% for all solvers. A possible explanation for this marked improvement is that a noise term may eliminate many local minima by providing additional directions of descent in which the model can describe the noise term instead of the data. Moreover, we observe this effect for several types of



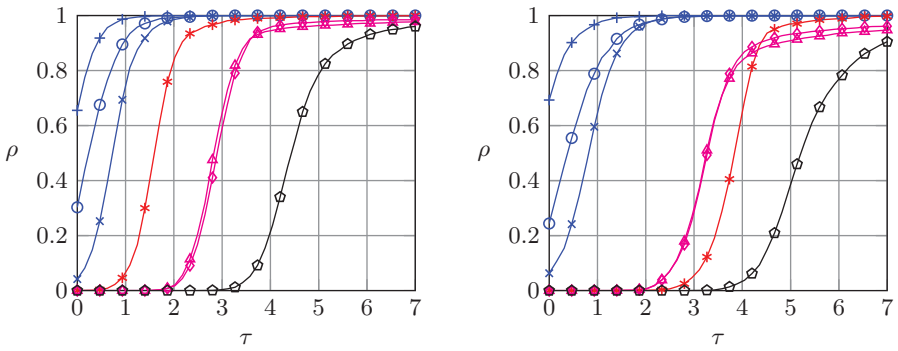
(a) normally distributed factor matrices and initialization ($t^* = 352.00$)

(b) uniformly distributed factor matrices and initialization ($t^* = 649.90$)

FIG. 5.6. *Performance profiles for complex rank-$(L_r, L_r, 1)$ block term decompositions, where $L_1 = 1$, $L_2 = 3$, and $L_3 = 5$, of noisy tensors of dimensions $10 \times 11 \times 12$ with an SNR of 25 dB. Legend: GN-DL (—+—), LM (—×—), GN-CGS (—○—), ALS (—✳—), L-BFGS-DL (—△—), L-BFGS-MT (—◇—), and NCG-MT (—⬠—).*

factor matrices so long as the noise term is sufficiently large. Future research might investigate other methods of introducing new directions of descent, perhaps by means of a "lifting" scheme [3], in which the model is modified instead of the data.

5.2. Computing the canonical polyadic decomposition. In the previous experiments, ALS generally seemed to perform quite well in the normally distributed factor matrices case and much less so in the uniformly distributed factor matrices case. ALS is known to perform poorly when the terms of the decomposition are difficult to separate [10], which is the case when they are similar to each other in some sense. One metric for similarity between terms is the cosine of the angle between them. Normally distributed factor matrices tend to generate highly uncorrelated terms, while uniformly distributed factor matrices will naturally generate terms with some degree of collinearity between them. In our last two experiments, we look at the performance on decompositions with mildly to moderately collinear terms. First, we generate 50 sets of normally distributed factor matrices. Each set of factor matrices corresponds to a complex rank-4 CPD of a fourth-order tensor of dimensions $10 \times 10 \times 10 \times 10$. For each set, we then modify the mode-n vectors of the second rank-one term so that the angle with the corresponding mode-n vectors in the first rank-one term is fixed. More specifically, we set $\boldsymbol{a}_2^{(n)} \leftarrow \alpha \boldsymbol{a}_1^{(n)} + \beta \boldsymbol{a}_2^{(n)}$, where the scalars α and β are chosen so that we obtain vectors $\boldsymbol{a}_2^{(n)}$ for which $\|\boldsymbol{a}_2^{(n)}\| = \|\boldsymbol{a}_1^{(n)}\|$ and $\boldsymbol{a}_1^{(n)\mathrm{H}} \boldsymbol{a}_2^{(n)} = \|\boldsymbol{a}_1^{(n)}\| \|\boldsymbol{a}_2^{(n)}\| \gamma$ hold for all $1 \leq n \leq N$. Here, γ is the parameter that determines the cosine of the angle between $\boldsymbol{a}_1^{(n)}$ and $\boldsymbol{a}_2^{(n)}$. We set γ equal to 0.65 and 0.85 in the first and second experiment, respectively. Effectively, this means cosine of the angle between the first two rank-one terms is $0.65^4 \approx 0.18$ and $0.85^4 \approx 0.52$, respectively. This corresponds to an angle of about 80° and 59°, respectively. For each tensor we generate another 50 sets of factor matrices of the same distribution as initialization for the solvers. We use the same stop criteria and condition for success as in the BTD experiments without noise. The performance profiles for the two experiments are plotted in Figure 5.7. It is clear that ALS is very sensitive to the amount of collinearity between terms, in contrast to the relatively robust performance of the optimization-based methods. Of the optimization-based methods, the inexact nonlinear least squares algorithms clearly



(a) normally distributed factor matrices and initialization with $\gamma = 0.65$ ($t^* = 69.36$)

(b) normally distributed factor matrices and initialization with $\gamma = 0.85$ ($t^* = 79.81$)

FIG. 5.7. *Performance profiles for complex rank-4 canonical polyadic decompositions of noisy tensors of dimensions* $10 \times 10 \times 10 \times 10$ *with some collinearity between two rank-one terms. Legend: GN-DL (—+—), LM (—×—), GN-CGS (—○—), ALS (—∗—), L-BFGS-DL (—△—), L-BFGS-MT (—◇—), and NCG-MT (—⬠—).*

outperform the others. Moreover, they are well suited for parallelization, while ALS is an inherently sequential algorithm.

6. Conclusion. The rank-$(L_r, L_r, 1)$ block term decomposition is an emerging decomposition for signal processing and blind source separation. We introduced the (rank-$L_r \circ$ rank-1) block term decomposition as a generalization of the former and the canonical polyadic decomposition. We then developed several algorithms for its computation, among which are ALS schemes, memory-efficient unconstrained gradient-based optimization methods such as nonlinear conjugate gradient and limited-memory BFGS with line search and trust-region frameworks, and matrix-free adaptations of nonlinear least squares methods such as Levenberg–Marquardt and Gauss–Newton. The resulting algorithms are all applicable to the canonical polyadic decomposition, the rank-$(L_r, L_r, 1)$ block term decomposition, and their generalized decomposition. Due to the multilinear structure of the objective function, the latter may be expected to converge close to quadratically, especially as the residuals decrease. Exploiting the structure of the Jacobian's Gramian has led to a significant decrease in computational complexity, compared to their exact counterparts, and reduced the memory cost to that of alternating least squares. The singularity of the Gramian is inherently handled by the built-in regularization of the conjugate gradient algorithm. Additionally, we increase the quality of the search directions generated by the conjugate gradient algorithm with a block Jacobi preconditioner. Numerical experiments confirm that these improvements make the inexact nonlinear least squares methods among the most efficient currently available. Furthermore, they are also robust w.r.t. the type of initialization, collinearity between terms, and the number of successful decompositions. The algorithms discussed in this article were implemented in MATLAB and are available upon request.

Appendix A. Computational complexity. In the following, we derive the computational complexity per iteration of several algorithms, which can often conveniently be expressed in terms of a number of function evaluations. To simplify the obtained expressions, we restrict the discussion to the CPD of a real Nth-order tensor of dimensions $I_1 \times \cdots \times I_N$ in R rank-one terms. In the complex case, a good rule of

thumb is to multiply the number of flop by a factor of four. A function evaluation of f_{BTD} (or $\mathcal{F}_{\text{BTD}}$) then costs

$$\begin{array}{rl} \mathcal{O}(2R\prod_{n=1}^{N} I_n) & \text{(Computing } \mathcal{F}_{\text{BTD}}\text{)} \\ \underline{\mathcal{O}(2\prod_{n=1}^{N} I_n)} & \text{(Computing } \tfrac{1}{2}\|\mathcal{F}_{\text{BTD}}\|^2\text{)} \\ \mathcal{O}(2R\prod_{n=1}^{N} I_n) & \text{flop.} \end{array}$$

A cogradient evaluation $\frac{\partial f_{\text{BTD}}}{\partial \boldsymbol{z}}$ costs the equivalent of N function evaluations, namely,

$$\begin{array}{rl} \mathcal{O}(\sum_{n=1}^{N}\prod_{\substack{m=1\\m\neq n}}^{N} I_m) & \text{(Computing all } V^{\{n\}}\text{)} \\ \mathcal{O}(2NR\prod_{n=1}^{N} I_n) & \text{(Computing all } \overline{T}_{(n)}\cdot V^{\{n\}}\text{)} \\ \underline{\mathcal{O}(\tfrac{1}{2}(N-1)R^2 + R^2\sum_{n=1}^{N} I_n)} & \text{(Computing all } W^{\{n\}} \text{ and } \overline{A}^{(n)}\cdot W^{\{n\}}\text{)} \\ \mathcal{O}(2NR\prod_{n=1}^{N} I_n) & \text{flop.} \end{array}$$

A.1. Alternating least squares. The cost of a fast ALS iteration is very similar to a gradient evaluation, with the most important difference being that the matrix-matrix product $\overline{A}^{(n)}\cdot W^{\{n\}}$ is replaced by solving a Hermitian linear system of order R. The matrices $W^{\{n\}} = {V^{\{n\}}}^{\text{H}}V^{\{n\}}$ can be factorized with a Cholesky decomposition, in which case solving all N systems costs $\mathcal{O}(\frac{1}{3}NR^3 + 2R^2\sum_{n=1}^{N} I_n)$ flop, which is often negligible compared to the cost of computing the products $\overline{T}_{(n)}\cdot V^{\{n\}}$. Every iteration, the objective function is also evaluated to check for convergence. The complexity of a fast ALS iteration is hence equal to

$$\begin{array}{rl} \mathcal{O}(2NR\prod_{n=1}^{N} I_n) & \text{(Computing all } T_{(n)}\cdot \overline{V}^{\{n\}}\text{)} \\ \mathcal{O}(\frac{1}{3}NR^3 + 2R^2\sum_{n=1}^{N} I_n) & \text{(Solving the linear systems)} \\ \underline{\mathcal{O}(2R\prod_{n=1}^{N} I_n)} & \text{(Function evaluation)} \\ \mathcal{O}(2(N+1)R\prod_{n=1}^{N} I_n) & \text{flop/iteration.} \end{array}$$

An accurate ALS iteration projects the rows of $T_{(n)}$ onto an orthogonal basis of $V^{\{n\}}$, instead of on $V^{\{n\}}$ itself as in a fast ALS iteration. An orthogonal basis of $V^{\{n\}}$ can be obtained by means of a QR decomposition. These decompositions require a total of $\mathcal{O}(-2R^3 + 2R^2\sum_{n=1}^{N}\prod_{\substack{m=1\\m\neq n}}^{N} I_m)$ flop, which is comparable in cost to a fast ALS iteration for moderate R. Hence the cost of an accurate ALS iteration is roughly double that of a fast ALS iteration.

A.2. General unconstrained optimization. L-BFGS-DL requires one cogradient evaluation every iteration and one function evaluation every inner iteration. Let it_{dl} be the number of dogleg iterations in each outer iteration; then for a small number of L-BFGS updates m the computational complexity is $\mathcal{O}(2(N+\text{it}_{\text{dl}})R\prod_{n=1}^{N} I_n)$ flop/iteration.

The general optimization methods based on a Moré–Thuente line search require one function and one gradient evaluation per line search iteration. Let it_{mt} be the number of line search iterations in each outer iteration; then L-BFGS and nonlinear conjugate gradient with Moré–Thuente line search (L-BFGS-MT and CG-MT) cost $\mathcal{O}(\text{it}_{\text{mt}}2(N+1)R\prod_{n=1}^{N} I_n)$ flop/iteration for small m.

A.3. Nonlinear least squares. Let us first consider the exact nonlinear least squares methods. The matrix $J^{\mathrm{H}}J$ requires $O(\frac{1}{2}R^2(\sum_{n=1}^{N} I_n)^2)$ memory cells. Precomputing the Gramians ${A^{(n)}}^{\mathrm{H}}A^{(n)}$ costs $\mathcal{O}(R^2\sum_{n=1}^{N} I_n)$ flop. For each diagonal block $\Pi^{(n,n)}$, the Hadamard products $W^{\{n\}}$ must be generated given these precomputed Gramians, which costs $\mathcal{O}(\frac{1}{2}NR^2)$ flop given $W^{\{\}}$. Each off-diagonal block $\Pi^{(n,m)}$ also requires a matrix $W^{\{n,m\}}$ and a further three multiplications per entry. Taking the symmetry into account, this is a total of $\mathcal{O}(\sum_{n=1}^{N}\sum_{m=n+1}^{N}(\frac{1}{2}R^2 + 3R^2I_nI_m))$ flop. In the hypercubical case $I = I_1 = \cdots = I_N$, the cost of evaluating $J^{\mathrm{H}}J$ is then

$$\begin{array}{rl}
\mathcal{O}(NR^2I) & \text{(Computing all } {A^{(n)}}^{\mathrm{H}}A^{(n)}\text{)} \\
\mathcal{O}(\frac{1}{2}NR^2) & \text{(Computing all } W^{\{n\}}\text{)} \\
\mathcal{O}(1) & \text{(Computing all } \Pi^{(n,n)}\text{)} \\
\mathcal{O}(\frac{1}{2}\frac{N(N-1)}{2}R^2) & \text{(Computing all } W^{\{n,m\}}\text{)} \\
\mathcal{O}(3\frac{N(N-1)}{2}R^2I^2) & \text{(Computing all } \Pi^{(n,m)}\text{)} \\
\hline
\mathcal{O}(\frac{3}{2}N^2R^2I^2) & \text{flop.}
\end{array}$$

In each iteration of the exact Levenberg–Marquardt algorithm, a linear system involving the Gramian $J^{\mathrm{H}}J$ and the conjugate cogradient must be solved repeatedly for different values of the regularization parameter λ until some descent criterion is satisfied, which is checked using function evaluations. If the number of such inner iterations in each outer iteration is denoted by $\mathrm{it}_{\mathrm{lm}}$, the cost of the exact Levenberg–Marquardt algorithm is given by

$$\begin{array}{rl}
\mathcal{O}(\frac{3}{2}N^2R^2I^2) & \text{(Evaluating } J^{\mathrm{H}}J\text{)} \\
\mathcal{O}(2NRI^N) & \text{(Cogradient evaluation)} \\
\mathcal{O}(\mathrm{it}_{\mathrm{lm}}(\frac{1}{3}N^3R^3I^3 + 2N^2R^2I^2)) & \text{(Solving the linear system)} \\
\mathcal{O}(\mathrm{it}_{\mathrm{lm}}2RI^N) & \text{(Function evaluation)} \\
\hline
\mathcal{O}(2(N+\mathrm{it}_{\mathrm{lm}})RI^N + \mathrm{it}_{\mathrm{lm}}\frac{1}{3}N^3R^3I^3) & \text{flop/iteration.}
\end{array}$$

The Gauss–Newton method with dogleg trust-region solves only one linear system per iteration. The trust-region subproblem is then iteratively minimized along a line connecting the resulting Gauss–Newton step (4.8) and the steepest descent direction. These inner iterations each require one function evaluation to check if the model is sufficiently accurate. Unfortunately, the system $J^{\mathrm{H}}J$ is always singular, and a filtered solution should be computed. One way of obtaining an accurate filtered solution is by means of the singular value decomposition, which in this case costs $\mathcal{O}(\frac{8}{3}N^3R^3I^3)$ flop. Let $\mathrm{it}_{\mathrm{gn}}$ be the number of inner iterations in each outer iteration; then the cost of the exact Gauss–Newton algorithm with dogleg trust-region is given by

$$\begin{array}{rl}
\mathcal{O}(\frac{3}{2}N^2R^2I^2) & \text{(Evaluating } J^{\mathrm{H}}J\text{)} \\
\mathcal{O}(2NRI^N) & \text{(Cogradient evaluation)} \\
\mathcal{O}(\frac{8}{3}N^3R^3I^3) & \text{(Solving the linear system)} \\
\mathcal{O}(\mathrm{it}_{\mathrm{gn}}2RI^N) & \text{(Function evaluation)} \\
\hline
\mathcal{O}(2(N+\mathrm{it}_{\mathrm{gn}})RI^N + \frac{8}{3}N^3R^3I^3) & \text{flop/iteration.}
\end{array}$$

In the proposed inexact adaptations of these nonlinear least squares methods, the Levenberg–Marquardt and Gauss–Newton steps are solved iteratively using a preconditioned conjugate gradient algorithm. To compute the matrix-vector prod-

uct $J^{\mathrm{H}}J \cdot \boldsymbol{p}$, where $\boldsymbol{p}$ is defined as in Theorem 4.6, the matrices ${A^{(n)}}^{\mathrm{T}}\overline{A}^{(n)}$ and ${B^{(n)}}^{\mathrm{T}}\overline{A}^{(n)}$ should first be precomputed. (The former need only be computed once.) These products cost $\mathcal{O}(R^2\sum_{n=1}^{N} I_n)$ and $\mathcal{O}(2R^2\sum_{n=1}^{N} I_n)$ flop, respectively. The contribution of the diagonal blocks $\Pi^{(n,n)} \cdot \mathrm{vec}(B^{(n)})$ is obtained by first computing $W^{\{n\}}$ from $W^{\{\}}$ and then the matrix products $B^{(n)} \cdot \overline{W}^{\{n\}}$, which costs a total of $\mathcal{O}(\sum_{n=1}^{N}(\frac{1}{2}R^2 + 2I_nR^2))$ flop. There are several ways to compute the contribution of the off-diagonal blocks. We propose to compute them row by row, which has the advantage that only one multiplication by $A^{(n)}$ is needed per row. The cost of the contribution of the off-diagonal blocks can then be shown to be $\mathcal{O}(\sum_{n=1}^{N}(\frac{1}{2}R^2 + \frac{1}{2}(N-1)R^2 + (N-1)R^2 + (N-2)R^2 + 2I_nR^2))$ flop. Applying a block-diagonal preconditioner costs a further $\mathcal{O}(\frac{1}{3}NR^3 + 2R^2\sum_{n=1}^{N} I_n)$ flop per conjugate gradient iteration. Let $\mathrm{it}_{\mathrm{cg}}$ be the number of conjugate gradient iterations required to solve the linear system to a prescribed accuracy; then the inexact Levenberg–Marquardt and Gauss–Newton algorithms cost

$$\begin{array}{rl}
\mathcal{O}(2NRI^N) & \text{(Cogradient evaluation)} \\
\mathcal{O}(\mathrm{it}_{\mathrm{lm}}\mathrm{it}_{\mathrm{cg}}(\frac{5}{2}N^2R^2 + 8NR^2I + \frac{1}{3}NR^3)) & \text{(Solving the linear system)} \\
\mathcal{O}(\mathrm{it}_{\mathrm{lm}}2RI^N) & \text{(Function evaluation)} \\
\hline
\mathcal{O}(2(N + \mathrm{it}_{\mathrm{lm}})RI^N + \mathrm{it}_{\mathrm{lm}}\mathrm{it}_{\mathrm{cg}} & \\
\quad (\frac{5}{2}N^2R^2 + 8NR^2I + \frac{1}{3}NR^3)) & \text{flop/iteration}
\end{array}$$

and

$$\begin{array}{rl}
\mathcal{O}(2NRI^N) & \text{(Cogradient evaluation)} \\
\mathcal{O}(\mathrm{it}_{\mathrm{cg}}(\frac{5}{2}N^2R^2 + 8NR^2I + \frac{1}{3}NR^3)) & \text{(Solving the linear system)} \\
\mathcal{O}(\mathrm{it}_{\mathrm{gn}}2RI^N) & \text{(Function evaluation)} \\
\hline
\mathcal{O}(2(N + \mathrm{it}_{\mathrm{gn}})RI^N + \mathrm{it}_{\mathrm{cg}} & \\
\quad (\frac{5}{2}N^2R^2 + 8NR^2I + \frac{1}{3}NR^3)) & \text{flop/iteration,}
\end{array}$$

respectively.

REFERENCES

[1] T. J. Abatzoglou, J. M. Mendel, and G. A. Harada, *The constrained total least squares technique and its applications to harmonic superresolution*, IEEE Trans. Signal Process., 39 (1991), pp. 1070–1087.

[2] E. Acar, T. G. Kolda, and D. M. Dunlavy, *An Optimization Approach for Fitting Canonical Tensor Decompositions*, Technical report, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2009.

[3] J. Albersmeyer and M. Diehl, *The lifted Newton method and its application in optimization*, SIAM J. Optim., 20 (2010), pp. 1655–1684.

[4] B. W. Bader and T. G. Kolda, *Efficient MATLAB computations with sparse and factored tensors*, SIAM J. Sci. Comput., 30 (2006), pp. 205–231.

[5] R. Bro, *Multi-Way Analysis in the Food Industry: Models, Algorithms, and Applications*, Ph.D. thesis, University of Amsterdam, Amsterdam, Netherlands, 1998.

[6] R. H. Byrd, R. B. Schnabel, and G. A. Schultz, *Approximate solution of the trust region problem by minimization over two-dimensional subspaces*, Math. Program., 40 (1988), pp. 247–263.

[7] J. D. Carroll and J.-J. Chang, *Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart–Young" decomposition*, Psychometrika, 35 (1970), pp. 283–319.

[8] A. Cichocki, R. Zdunek, A. H. Phan, and S.-I. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-Way Data Analysis and Blind Source Separation*, Wiley, New York, 2009.

[9] P. Comon and Chr. Jutten, *Handbook of Blind Source Separation: Independent Component Analysis and Applications*, Academic Press, New York, 2010.
[10] P. Comon, X. Luciani, and A. L. F. de Almeida, *Tensor decompositions, alternating least squares and other tales*, J. Chem., 23 (2009), pp. 393–405.
[11] L. De Lathauwer, *A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 642–666.
[12] L. De Lathauwer, *Decompositions of a higher-order tensor in block terms—Part* I: *Lemmas for partitioned matrices*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1022–1032.
[13] L. De Lathauwer, *Decompositions of a higher-order tensor in block terms—Part* II: *Definitions and uniqueness*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1033–1066.
[14] L. De Lathauwer, *Blind separation of exponential polynomials and the decomposition of a tensor in rank-*$(L_r, L_r, 1)$ *terms*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1451–1474.
[15] L. De Lathauwer, *A short introduction to tensor-based methods for factor analysis and blind source separation*, in Proceedings of the 7th International Symposium on Image and Signal Processing and Analysis (ISPA 2011), 2011, pp. 558–563.
[16] L. De Lathauwer, *Block component analysis, a new concept for source separation and factor analysis*, in Latent Variable Analysis and Signal Separation, Lecture Notes in Comput. Sci. 7191, Springer, Berlin, 2012, pp. 1–8.
[17] L. De Lathauwer, P. Comon, and N. Mastronardi, *Special issue on tensor decompositions and applications*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1–1279.
[18] L. De Lathauwer and A. de Baynast, *Blind deconvolution of DS-CDMA signals by means of decomposition in rank-*$(1, L, L)$ *terms*, IEEE Trans. Signal Process., 56 (2008), pp. 1562–1571.
[19] L. De Lathauwer, B. L. R. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278.
[20] L. De Lathauwer and D. Nion, *Decompositions of a higher-order tensor in block terms—Part* III: *Alternating least squares algorithms*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1067–1083.
[21] V. de Silva and L.-H. Lim, *Tensor rank and the ill-posedness of the best low-rank approximation problem*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1084–1127.
[22] J. E. Dennis, Jr. and H. H. W. Mei, *Two new unconstrained optimization algorithms which use function and gradient values*, J. Optim. Theory Appl., 28 (1979), pp. 453–482.
[23] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.
[24] I. Domanov and L. De Lathauwer, *On the Uniqueness of the Canonical Polyadic Decomposition—Part* I: *Basic Results and Uniqueness of One Factor Matrix*, Internal report 12-72, ESAT-SISTA, KU Leuven, 2012.
[25] I. Domanov and L. De Lathauwer, *On the Uniqueness of the Canonical Polyadic Decomposition—Part* II: *Overall Uniqueness*, Internal report 12-90, ESAT-SISTA, KU Leuven, 2012.
[26] W. W. Hager and H. Zhang, *A new conjugate gradient method with guaranteed descent and an efficient line search*, SIAM J. Optim., 16 (2006), pp. 170–192.
[27] R. A. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis*, UCLA Working Papers in Phonetics, 16 (1970), pp. 84–84.
[28] C. Hayashi and F. Hayashi, *A new algorithm to solve PARAFAC-model*, Behaviormetrika, 9 (1982), pp. 49–60.
[29] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Phys., 6 (1927), pp. 164–189.
[30] F. L. Hitchcock, *Multiple invariants and generalized rank of a p-way matrix or tensor*, J. Math. Phys., 7 (1927), pp. 39–79.
[31] T. Jiang and N. D. Sidiropoulos, *Kruskal's permutation lemma and the identification of CANDECOMP/PARAFAC and bilinear models with constant modulus constraints*, IEEE Trans. Signal Process., 52 (2004), pp. 2625–2636.
[32] C. G. Khatri and C. R. Rao, *Solutions to some functional equations and their applications to characterization of probability distributions*, Sankhyā Ser. A, 30 (1968), pp. 167–180.
[33] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.
[34] W. P. Krijnen, T. K. Dijkstra, and A. Stegeman, *On the non-existence of optimal solutions and the occurrence of "degeneracy" in the Candecomp/Parafac model*, Psychometrika, 73 (2008), pp. 431–439.
[35] P. M. Kroonenberg, *Applied Multiway Data Analysis*, Wiley, New York, 2008.

[36] J. B. Kruskal, *Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics*, Linear Algebra Appl., 18 (1977), pp. 95–138.
[37] J. B. Kruskal, *Rank, Decomposition, and Uniqueness for* 3*-way and N-way Arrays*, Elsevier Science Publishers B.V., Amsterdam, 1989, pp. 7–18.
[38] J. B. Kruskal, R. A. Harshman, and M. E. Lundy, *How* 3*-MFA data can cause degenerate PARAFAC solutions, among other relationships*, in Multiway Data Analysis, R. Coppi and S. Bolasco, eds., North-Holland, Amsterdam, 1989, pp. 115–121.
[39] B. C. Mitchell and D. S. Burdick, *Slowly converging PARAFAC sequences: Swamps and two-factor degeneracies*, J. Chem., 8 (1994), pp. 155–168.
[40] J. J. Moré and D. J. Thuente, *Line search algorithms with guaranteed sufficient decrease*, ACM Trans. Math. Software, 20 (1994), pp. 286–307.
[41] C. Navasca, L. De Lathauwer, and S. Kindermann, *Swamp reducing technique for tensor decomposition*, in Proceedings of the 16th European Signal Processing Conference (EUSIPCO 2008), Lausanne, Switzerland, 2008.
[42] D. Nion and L. De Lathauwer, *An enhanced line search scheme for complex-valued tensor decompositions. Application in DS-CDMA*, Signal Process., 88 (2008), pp. 749–755.
[43] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Ser. Oper. Res., 2nd ed., Springer, New York, 2006.
[44] P. Paatero, *A weighted non-negative least squares algorithm for three-way "PARAFAC" factor analysis*, Chemometr. Intell. Lab. Syst., 38 (1997), pp. 223–242.
[45] P. Paatero, *The multilinear engine: A table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model*, J. Comput. Graph. Statist., 8 (1999), pp. 854–888.
[46] P. Paatero, *Construction and analysis of degenerate PARAFAC models*, J. Chem., 14 (2000), pp. 285–299.
[47] A. H. Phan, P. Tichavský, and A. Cichocki, *Low complexity damped Gauss-Newton algorithms for CANDECOMP/PARAFAC*, Comp. Research Repos., arXiv:1205.2584, 2012.
[48] A. H. Phan, P. Tichavský, and A. Cichocki, *On fast computation of gradients for CANDECOMP/PARAFAC algorithms*, Comp. Research Repos., arXiv:1204.1586, 2012.
[49] M. J. D. Powell, *A hybrid method for nonlinear equations*, in Numerical Methods for Nonlinear Algebraic Equations, P. Robinowitz, ed., Gordon and Breach Science, London, 1970, pp. 87–144.
[50] M. J. D. Powell, *A new algorithm for unconstrained optimization*, in Nonlinear Programming, J. B. Rosen, O. L. Mangasarian, and K. Ritter, eds., Academic Press, New York, 1970, pp. 31–66.
[51] M. Rajih, P. Comon, and R. A. Harshman, *Enhanced line search: A novel method to accelerate PARAFAC*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1128–1147.
[52] C. R. Rao and M. B. Rao, *Matrix Algebra and its Applications to Statistics and Econometrics*, World Scientific, River Edge, NJ, 1998.
[53] R. Remmert, *Theory of Complex Functions*, Springer-Verlag, New York, 1991.
[54] N. D. Sidiropoulos and R. Bro, *On the uniqueness of multilinear decomposition of N-way arrays*, J. Chem., 14 (2000), pp. 229–239.
[55] N. D. Sidiropoulos, R. Bro, and G. B. Giannakis, *Parallel factor analysis in sensor array processing*, IEEE Trans. Signal Process., 48 (2000), pp. 2377–2388.
[56] N. D. Sidiropoulos and G. Z. Dimić, *Blind multiuser detection in W-CDMA systems with large delay spread*, IEEE Signal Process. Lett., 8 (2001), pp. 87–89.
[57] N. D. Sidiropoulos, G. B. Giannakis, and R. Bro, *Blind PARAFAC receivers for DS-CDMA systems*, IEEE Trans. Signal Process., 48 (2000), pp. 810–823.
[58] A. Smilde, R. Bro, and P. Geladi, *Multi-Way Analysis with Applications in the Chemical Sciences*, Wiley, New York, 2004.
[59] L. Sorber, M. Van Barel, and L. De Lathauwer, *Unconstrained optimization of real functions in complex variables*, SIAM J. Optim., 22 (2012), pp. 879–898.
[60] A. Stegeman, *Degeneracy in Candecomp/Parafac explained for* $p \times p \times 2$ *arrays of rank* $p+1$ *or higher*, Psychometrika, 71 (2006), pp. 483–501.
[61] A. Stegeman, *Low-rank approximation of generic* $p \times q \times 2$ *arrays and diverging components in the Candecomp/Parafac model*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 988–1007.
[62] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM J. Numer. Anal., 20 (1983), pp. 626–637.
[63] G. Tomasi, *Practical and Computational Aspects in Chemometric Data Analysis*, Ph.D. thesis, The Royal Veterinary and Agricultural University, Frederiksberg, Denmark, 2006.
[64] G. Tomasi and R. Bro, *A comparison of algorithms for fitting the PARAFAC model*, Comput. Statist. Data Anal., 50 (2006), pp. 1700–1734.

[65] L. R. Tucker, *The extension of factor analysis to three-dimensional matrices*, in Contributions to Mathematical Psychology, H. Gulliksen and N. Frederiksen, eds., Holt, Rinehart and Winston, New York, 1964, pp. 110–127.
[66] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.
[67] A. van den Bos, *Complex gradient and Hessian*, IEE Proc. Vis., Image & Signal Process., 141 (1994), pp. 380–383.
[68] P. Wolfe, *Convergence conditions for ascent methods*, SIAM Rev., 11 (1969), pp. 226–235.