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, as well as 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

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, 16, 29, 50, 52, 53]. The two main tensor generalizations of the singular value decomposition (SVD) are, on one hand, the Tucker decomposition or multilinear SVD (MLSVD) [18, 60, 61] and, on the other hand, the canonical polyadic decomposition (CPD) [7, 23]. 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, 19] as a framework that unifies the MLSVD and CPD. Of particular interest is the rank-$(L_r, L_r, 1)$ BTD, which has recently proven to be useful in blind source separation [14], and particularly in telecommunication applications [17, 37, 51].

In this article, we propose several optimization-based algorithms to compute the 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

[†]Department of Computer Science, KU Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium. *Email:* {Laurent.Sorber,Marc.VanBarel}@cs.kuleuven.be.

[‡]Group Science, Engineering and Technology, KU Leuven Kulak, E. Sabbelaan 53, B-8500 Kortrijk, Belgium. *Email:* Lieven.DeLathauwer@kuleuven-kulak.be.

[§]Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium. *Email:* Lieven.DeLathauwer@esat.kuleuven.be.

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 far less sensitive to the type of initialization and are often not only more efficient and less prone to so-called swamps, but also much more likely to converge to the global minimum, given a unique decomposition.

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., $\mathcal{A}$. The ith entry of a vector $\boldsymbol{a}$ is denoted by a_i, element (i, j) of a matrix A by a_{ij} and element (i, j, k) of a third-order tensor $\mathcal{A}$ by a_{ijk}. Indices typically range from one to their capital version, e.g., $i = 1, \ldots, I$. A colon is used to indicate all elements of a mode. Thus, $\boldsymbol{a}_{:j}$ corresponds to the jth column of a matrix A, which we also denote more compactly as $\boldsymbol{a}_j$. Mode-n vectors are the higher-order analogue of matrix rows and columns. A mode-n vector is defined by fixing every index but one. Third-order tensors have mode-1, mode-2 and mode-3 vectors, denoted by $\boldsymbol{t}_{:jk}$, $\boldsymbol{t}_{i:k}$ and $\boldsymbol{t}_{ij:}$, respectively (cf. Figure 2.1).

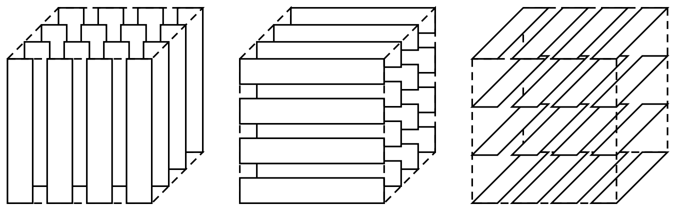


FIG. 2.1. *Mode-n vectors of a third-order tensor.*

The nth element in a sequence is denoted by a superscript in parentheses, e.g., $A^{(n)}$ denotes the nth matrix in a sequence. 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 pseudo-inverse, 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 $\begin{bmatrix}\boldsymbol{a}^{\mathrm{T}} & \boldsymbol{b}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$. 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.

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*

$$\langle \mathcal{T}, \mathcal{U}\rangle = \sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} \bar{t}_{i_1\cdots i_N} u_{i_1\cdots i_N}.$$

Definition 2.2. *The* (Frobenius) norm *of a tensor* $\mathcal{T} \in \mathbb{C}^{I_1\times\cdots\times I_N}$ *is defined as* $\|\mathcal{T}\| = \sqrt{\langle \mathcal{T}, \mathcal{T}\rangle}$.

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 $\operatorname{vec}(A)$ *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, 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*

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

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* $R_n = \operatorname{rank}_n(\mathcal{T})$, *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* $(\mathcal{T} \times_n A)_{(n)} = A \cdot T_{(n)}$.

Definition 2.9. *The* Kronecker product *of two matrices* $A \in \mathbb{C}^{I\times J}$ *and* $B \in \mathbb{C}^{K\times L}$ *is defined as*

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1J}B \\ \vdots & \ddots & \vdots \\ a_{I1}B & \cdots & a_{IJ}B \end{bmatrix}.$$

Definition 2.10. *The* Khatri–Rao product [27] *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 2.11. *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 canonical polyadic decomposition (CPD) approximates a tensor with a sum of R rank-one tensors. It was introduced by Hitchcock in 1927 [25,26], and was later referred to as the canonical decomposition (CANDECOMP) [7] and parallel factor decomposition (PARAFAC) [23] 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 (cf. Figure 3.1). 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 contrary to the matrix case, often unique. Let $A^{(n)} = \begin{bmatrix} \boldsymbol{a}_1^{(n)} & \cdots & \boldsymbol{a}_R^{(n)} \end{bmatrix}$ 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, $V^{\{n\}} = A^{(N)} \odot \cdots \odot A^{(n+1)} \odot A^{(n-1)} \odot \cdots \odot A^{(1)}$.

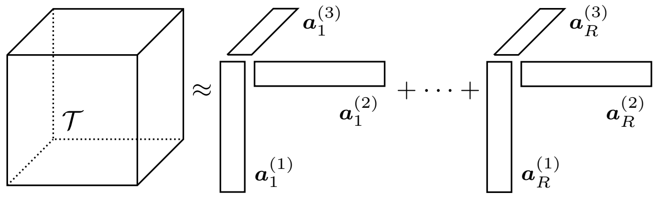


FIG. 3.1. *Canonical polyadic decomposition of a third-order tensor.*

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 [30, 31] 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 [30]. Kruskal's sufficient condition for the unicity of a rank decomposition, generalized to Nth order tensors by Sidiropoulos and Bro [49], 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].

3.2. The rank-$(L_r, L_r, 1)$ block term decomposition. The rank-$(L_r, L_r, 1)$ block term decomposition (BTD) [12, 13, 19] 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 let $\boldsymbol{c}_r \in \mathbb{C}^{I_3}$, $\boldsymbol{c}_r$ 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 T rank-$(L_r, L_r, 1)$ terms (cf. Figure 3.2). In addition to the permutation and scaling indeterminacies inherited from the CPD, the factors A_t 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 $\begin{bmatrix} A_1 & \cdots & A_R \end{bmatrix}$ 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]. Even more relaxed conditions are expected to exist, and are a topic for future research. For example, a new sufficient condition was derived in [14]. The CPD and rank-$(L_r, L_r, 1)$ BTD have both been applied in telecommunication applications [17, 37, 50–52]. 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]. The rank-$(L_r, L_r, 1)$ BTD generalizes the CPD for third-order

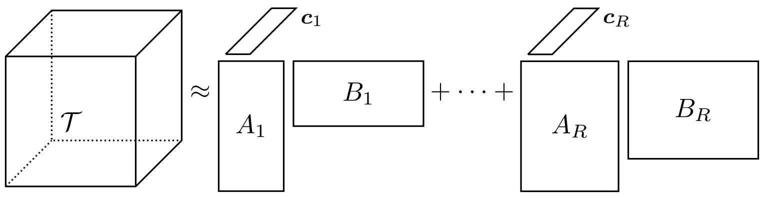


FIG. 3.2. *Rank-$(L_r, L_r, 1)$ block term decomposition of a third-order tensor.*

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 $\circ$ rank-1) block term decomposition. The (rank-L_r $\circ$ 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-$L_r \circ$ rank-one) terms (cf. Figure 3.3). 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 $Q = 1$.

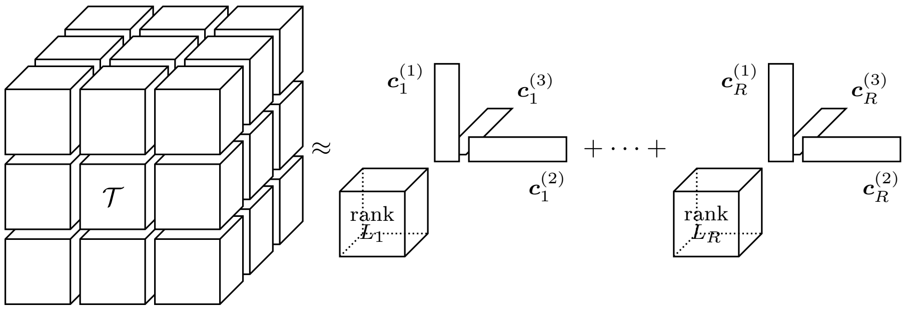


FIG. 3.3. *A (rank-L_r $\circ$ rank-1) block term decomposition of a sixth-order tensor.*

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

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

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

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

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

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

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 [54]. 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 [1,62].

4. Algorithms for the general decomposition.

4.1. General unconstrained optimization & 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 [38]. The complex counterparts of these methods [54] are built on the second-order model

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

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, e.g., $\overset{c}{\boldsymbol{z}} \equiv (\boldsymbol{z}, \overline{\boldsymbol{z}})$, and B_k is a Hermitian positive definite matrix that is updated 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 [46]. 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*

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

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

Proposition 4.2 (see e.g., [5, 45]). *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 $W = \mathop{\ast}\limits_{n=1}^{N} {A^{(n)}}^{\mathrm{H}} A^{(n)}$ be the Hadamard product of all matrices ${A^{(n)}}^{\mathrm{H}} A^{(n)}$. Then $V^{\mathrm{H}} V = W$.*

Corollary 4.3. *Define $W^{\sigma} \triangleq \mathop{\ast}\limits_{\substack{n=1 \\ n \notin \sigma}}^{N} {A^{(n)}}^{\mathrm{H}} A^{(n)}$, then $(V^{\sigma})^{\mathrm{H}} V^{\sigma} = W^{\sigma}$.*

Theorem 4.4. *Let $\boldsymbol{z}$ be the vector of unknowns* $(\operatorname{vec}(A^{(1)}), \ldots, \operatorname{vec}(A^{(P)}),$ $\operatorname{vec}(C^{(1)}), \ldots, \operatorname{vec}(C^{(Q)}))$*, 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

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

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

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

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

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

where the last equality follows from Corollary 4.3. Similarly, it can be shown that $\frac{\partial f^{(2)}_{\mathrm{BTD}}}{\partial A^{(n)}} = \overline{T}_{(n)} \cdot V^{\{n\}}$, and trivially that $\frac{\partial \overline{f}^{(2)}_{\mathrm{BTD}}}{\partial A^{(n)}} = \frac{\partial f^{(3)}_{\mathrm{BTD}}}{\partial A^{(n)}} = 0$. 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 [54].

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

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.

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 [38,63]. 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 Moré and Thuente [35] and Hager and Zhang [22].

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 $m_k^f(\mathbf{0}) - m_k^f(\overset{c}{\boldsymbol{p}}_k)$. The dogleg method [42, 43], double-dogleg method [21] 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 alternating least squares (ALS) algorithm [7, 19, 23], 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 pseudo-inverse of the matrix $V^{\{p\}\mathrm{T}}$ 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 floating point operations considering 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.

```
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
```

ALGORITHM 4.1. *Alternating least squares with fast updates.*

Although conceptually simple and often effective, ALS is not guaranteed to converge to 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 approximation [20]. 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 $R < R^*$ [20, 41]. For such sequences, two or more terms grow without bound yet nearly cancel each other out. This behaviour is referred to as degeneracy and has been linked with swamps [32, 34], which are observed when either the best rank-R approximation does not exist (strong degeneracy [28, 55, 56]), 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 [34]). Several approaches to mitigate swamp-like behaviour in ALS have been proposed in the literature, among which (exact) line search [5, 23, 37, 44] and iterated Tikhonov regularization [36]. 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 $\mathcal{F}_{\text{BTD}}$ is approximated by the linear model

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

where the partial derivative is called the complex Jacobian at $\overset{c}{\boldsymbol{z}}_k$ and is a straightforward generalization of the complex gradient [54]. The model $m_k^{\mathcal{F}}$ is then used in the modified quadratic model of the objective function

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

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

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

where J_k is the Jacobian $\frac{\partial \operatorname{vec}(\mathcal{F}_{\text{BTD}})}{\partial \boldsymbol{z}^{\text{T}}}$ 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 compute the Moore–Penrose pseudo-inverse $J_k^\dagger$. 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) [39, 59]. 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 $\mathcal{F}_{\mathrm{BTD}}$ [54], associated with (4.8) and looking for an expression for the Gramian $J^{\mathrm{H}}J$ [58]. In the real case, the latter matrix represents an approximation of the objective function's Hessian. The error of the approximation is the sum [38]

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

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 quite small in comparison to the elements of $J^{\mathrm{H}}J$. It may therefore be expected that, as the residuals $(\mathcal{F}_{\mathrm{BTD}})_{i_1\ldots i_N}$ decrease, the Gramian $J^{\mathrm{H}}J$ rapidly becomes a good approximation to the real Hessian. In Theorem 4.5 we show that $J^{\mathrm{H}}J$ is a block matrix with diagonal and rank-one blocks. This structure can be exploited to significantly reduce both its storage cost and the computational complexity of its matrix-vector product.

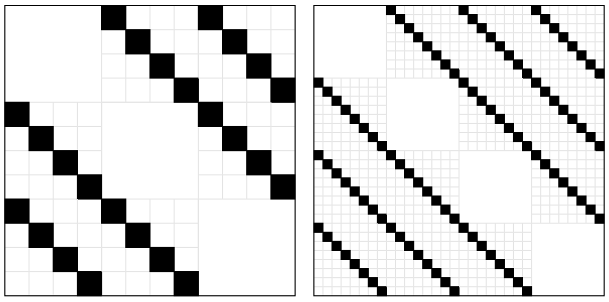

(a) $N = 3$, $R = 4$ (b) $N = 4$, $R = 8$

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

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

$$\Sigma \triangleq \operatorname{diag}\left(\mathbb{I}_{I_1 R'}, \dots, \mathbb{I}_{I_P R'}, F^{(1)}, \dots, F^{(Q)}\right) \tag{4.11}$$

and

$$\begin{aligned} \Pi^{(n_1,n_2)} &\triangleq \left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \operatorname{vec}(A^{(n_1)})^{\mathrm{T}}}\right)^{\mathrm{H}} \left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \operatorname{vec}(A^{(n_2)})^{\mathrm{T}}}\right) \\ &= \begin{cases} W^{\{n\}} \otimes \mathbb{I}_{I_n} & n = n_1 = n_2 \\ \begin{bmatrix} w_{1,1}^{\{n_1,n_2\}} \boldsymbol{a}_1^{(n_1)} {\boldsymbol{a}_1^{(n_2)}}^{\mathrm{H}} & \cdots & w_{1,R'}^{\{n_1,n_2\}} \boldsymbol{a}_{R'}^{(n_1)} {\boldsymbol{a}_1^{(n_2)}}^{\mathrm{H}} \\ \vdots & \ddots & \vdots \\ w_{R',1}^{\{n_1,n_2\}} \boldsymbol{a}_1^{(n_1)} {\boldsymbol{a}_{R'}^{(n_2)}}^{\mathrm{H}} & \cdots & w_{R',R'}^{\{n_1,n_2\}} \boldsymbol{a}_{R'}^{(n_1)} {\boldsymbol{a}_{R'}^{(n_2)}}^{\mathrm{H}} \end{bmatrix} & \textit{otherwise.} \end{cases} \end{aligned} \tag{4.12}$$

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

$$\left\langle \frac{\partial \mathcal{F}_{\mathrm{BTD}}}{\partial a_{ir_1}^{(n)}}, \frac{\partial \mathcal{F}_{\mathrm{BTD}}}{\partial a_{jr_2}^{(n)}} \right\rangle = \begin{cases} w_{r_1 r_2}^{\{n\}} & i = j \\ 0 & \text{otherwise} \end{cases}$$

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

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

and so

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

The $(p, P+q)$th block in $J^{\mathrm{H}}J$ is defined as

$$\left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \operatorname{vec}(A^{(p)})^{\mathrm{T}}}\right)^{\mathrm{H}} \left(\frac{\partial \operatorname{vec}(\mathcal{F}_{\mathrm{BTD}})}{\partial \operatorname{vec}(C^{(q)})^{\mathrm{T}}}\right)$$

and can be computed from $\Pi^{(p,P+q)}$ using (4.3b) as $\Pi^{(p,P+q)} \cdot F^{(q)\mathrm{T}}$. 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 $F^{(q_1)} \cdot \Pi^{(P+q_1,P+q_2)} \cdot F^{(q_2)\mathrm{T}}$, respectively. ∎

We propose to store $J^{\mathrm{H}}J$ as the collection of factor matrices $A^{(n)}$ and the factor matrices' Gramians $A^{(n)\mathrm{H}}A^{(n)}$, from which the Hadamard products $W^{\{n\}}$ and $W^{\{n_1,n_2\}}$ can easily be reconstructed. In the case of a hypercubical CPD, this adds only $\mathcal{O}(NR^2)$ to the memory complexity of the algorithm, as opposed to $\mathcal{O}(N^2R^2I^2)$ when $J^{\mathrm{H}}J$ is stored in a dense format. Storing the Jacobian's Gramian in this way is not appropriate for direct solvers, but is suited for matrix-vector products and hence admits the use of inexact adaptations of the Levenberg–Marquardt and Gauss–Newton algorithms. The latter only partially invert the Jacobian's Gramian using a limited number of (preconditioned) conjugate gradient iterations [38]. Besides the 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 $\boldsymbol{b}^{(p)} = \mathrm{vec}(B^{(p)})$, $p=1,\ldots,P$, $D^{(q)} \in \mathbb{C}^{I_{P+q}\times R}$ and $\boldsymbol{d}^{(q)} = \mathrm{vec}(D^{(q)})$, $q=1,\ldots,Q$. Furthermore, we define $B^{(P+q)} \triangleq D^{(q)} \cdot E$ and $\boldsymbol{b}^{(P+q)} \triangleq \mathrm{vec}(B^{(P+q)})$. The matrix-vector product $J^H J \cdot \boldsymbol{p}$ then follows from*

$$\Pi^{(p,p)} \cdot \boldsymbol{b}^{(p)} = \mathrm{vec}(X^{(p)}) \tag{4.13a}$$

$$\Pi^{(p_1,p_2)} \cdot \boldsymbol{b}^{(p_2)} = \mathrm{vec}(Y^{(p_1,p_2)}) \tag{4.13b}$$

$$(F^{(q)} \cdot \Pi^{(P+q,p)}) \cdot \boldsymbol{b}^{(p)} = \mathrm{vec}(Y^{(P+q,p)} \cdot E^{\mathrm{T}}) \tag{4.13c}$$

$$(\Pi^{(p,P+q)} \cdot {F^{(q)}}^{\mathrm{T}}) \cdot \boldsymbol{d}^{(q)} = \mathrm{vec}(Y^{(p,P+q)}) \tag{4.13d}$$

$$(F^{(q_1)} \cdot \Pi^{(P+q_1,P+q_2)} \cdot {F^{(q_2)}}^{\mathrm{T}}) \cdot \boldsymbol{d}^{(q_2)} = \mathrm{vec}(Y^{(P+q_1,P+q_2)} \cdot E^{\mathrm{T}}) \tag{4.13e}$$

$$(F^{(q)} \cdot \Pi^{(P+q,P+q)} \cdot {F^{(q)}}^{\mathrm{T}}) \cdot \boldsymbol{d}^{(q)} = \mathrm{vec}(X^{(P+q)} \cdot E^{\mathrm{T}}) \tag{4.13f}$$

where

$$X^{(n)} \triangleq B^{(n)} \cdot \overline{W}^{\{n\}}, \tag{4.14}$$

$$Y^{(n_1,n_2)} \triangleq A^{(n_1)} \cdot (\overline{W}^{\{n_1,n_2\}} * ({B^{(n_2)}}^{\mathrm{T}} \cdot \overline{A}^{(n_2)})) \text{ and} \tag{4.15}$$

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

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 (4.13b) follows. The special cases (4.13c–4.13f) are a direct result of (4.13a–4.13b) 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

$$M_{\mathrm{BJ}} = \Sigma \cdot \begin{bmatrix} \Pi^{(1,1)} & & \\ & \ddots & \\ & & \Pi^{(N,N)} \end{bmatrix} \cdot \Sigma^{\mathrm{T}} \tag{4.16}$$

is one such example. It is a block-diagonal approximation of (4.10) and can be inverted efficiently using (4.13a) and (4.13f). For instance, the solution of the system $\Pi^{(p,p)} \cdot \operatorname{vec}(X) = \operatorname{vec}(Y)$ 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 $\boldsymbol{p} = M_{\mathrm{BJ}}^{-1} \cdot \left(-2\frac{\partial f_{\mathrm{BTD}}}{\partial \overline{\boldsymbol{z}}}\right)$, 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 $O(NR^3)$ 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 more simple problems.

4.3. Computational complexity. In Section 5 we compare the accuracy and efficiency of the algorithms discussed in this section. To this end, we express the amount of effort required to reach a certain precision in terms of a unit of work which we define to be one evaluation of the objective function f_{BTD}. Table 4.2 gives an overview of some of these algorithms' computational complexity in flop per iteration (cf. Appendix A for the derivation). In the numerical experiments, we use this table to estimate the total amount of floating point operations and then divide by the number of floating point operations equivalent to one function evaluation to obtain an equivalent number of function evaluations. For example, alternating least squares with fast updates requires a number of floating point operations per iteration equivalent to the cost of $N + 1$ function evaluations.

Aside from the substantial reduction in floating point operations 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 with fast updates 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 every inner iteration. We also note that nonlinear conjugate gradient [2, 40] and exact Levenberg–Marquardt [24, 39, 58] have been applied to the real CPD before. For an overview and comparison of existing algorithms for the CPD, see [10] and [59], respectively.

Algorithm	**Complexity** (flop/iteration)
ALS (fast)	$\mathcal{O}(2(N+1)RI^N)$
ALS (accurate)	$\mathcal{O}(2(N+1)RI^N + 2NR^2I^{N-1} - 2R^3)$
L-BFGS-DL	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{dl}})RI^N)$
L-BFGS-MT	$\mathcal{O}(\mathrm{it}_{\mathrm{mt}}2(N+1)RI^N)$
CG-MT	$\mathcal{O}(\mathrm{it}_{\mathrm{mt}}2(N+1)RI^N)$
LM	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{lm}})RI^N + \mathrm{it}_{\mathrm{lm}}\frac{1}{3}N^3R^3I^3)$
GN-DL	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{gn}})RI^N + \frac{8}{3}N^3R^3I^3)$
LM (inexact)	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{lm}})RI^N + \mathrm{it}_{\mathrm{lm}}\mathrm{it}_{\mathrm{cg}}(\frac{5}{2}N^2R^2 + 8NR^2I + \frac{1}{3}NR^3))$
GN-DL (inexact)	$\mathcal{O}(2(N+\mathrm{it}_{\mathrm{gn}})RI^N + \mathrm{it}_{\mathrm{cg}}(\frac{5}{2}N^2R^2 + 8NR^2I + \frac{1}{3}NR^3))$

TABLE 4.2

Computational complexity of several BTD algorithms in flop/iteration in case of a CPD of an Nth-order $I \times \cdots \times I$ tensor in R rank-one terms. From top to bottom, the algorithms are alternating least squares with fast updates, alternating least squares with accurate updates, limited-memory BFGS with dogleg trust region, limited-memory BFGS with Moré–Thuente line search, nonlinear conjugate gradient with Moré–Thuente line search, Levenberg–Marquardt, Gauss–Newton with dogleg trust region, inexact Levenberg–Marquardt and inexact Gauss–Newton with dogleg trust region.

5. Numerical experiments.

We compare the algorithms in Table 4.2 in both accuracy and efficiency with a number of Monte Carlo simulations. In a first set of experiments, we are interested in the performance on real 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. All experiments were performed in MATLAB 7.13 (R2011b) on two octocore Intel Xeon X5550 CPUs with 32 GB RAM.

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

5.1.1. Without noise.

A simple problem. First, fifty sets of factor matrices are generated, the entries of which are pseudorandom numbers drawn from the standard normal distribution. 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 of factor matrices, 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 unique since the factor matrices are generically full column rank [13]. For each tensor, we generate another fifty sets of orthogonalized pseudorandom factor matrices with the same dimensions as the original factor matrices to use as initializations for the algorithms. The convergence criteria are identical for all algorithms and are such that the algorithms stop if the difference in objective function value or the difference of the solution between two successive iterates is less than machine precision. If neither of these conditions are satisfied, the algorithms terminate after a maximum of 1000 iterations. In other words, the algorithms continue to iterate until no further improvement in the objective function can be made, or the solution does not change, or the maximum number of iterations is reached.

Each algorithm then attempts to decompose the fifty tensors using the given initializations, and for each attempt the attained accuracy, the number of iterations and equivalent number of function evaluations are recorded. The accuracy is computed as $\sqrt{2f_{\text{BTD}}}$, which is equal to both the absolute and relative error of the approximation in Frobenius norm. As noted in Section 4.3, we derive an equivalent number of function evaluations from the algorithm's computational complexity in Table 4.2. For many algorithms, this is just a multiple of the number of iterations. The result of each attempted decomposition is divided into one of two classes: if the attained accuracy is less than 10^{-12} and the number of iterations is less than the maximal amount, the attempt is logged as successful, otherwise it is logged as unsuccessful. For both classes, we draw a box plot of the accuracy and equivalent number of function evaluations. Next to each box plot the relative number of decompositions belonging to that class is printed with a green or red background, for the successful and unsuccessful class, respectively.

We compare a total of seven algorithms: ALS with fast updates (ALS fast), ALS with fast updates followed by ALS with accurate updates (ALS fast-acc), limited memory BFGS with dogleg trust-region (L-BFGS-DL), limited memory BFGS with Moré–Thuente line search (L-BFGS-MT), nonlinear conjugate gradient with Moré–Thuente line search (CG-MT), inexact Gauss–Newton with dogleg trust-region (GN-DL) and inexact Levenberg–Marquardt (LM). For ALS fast-acc, the result of ALS fast is used to initialize ALS with accurate updates. As a consequence, the second algorithm is the only algorithm which has an effective maximal amount of iterations equal to 2000. An additional requirement for its decompositions to be labeled as successful is that the ALS fast initialization must have also completed within 1000 iterations. For the inexact nonlinear least squares methods, we have opted to truncate the CG iterations after a maximum of 20 iterations or when a relative residual of 10^{-6} is obtained, although there exist more advanced truncation strategies which take the trust-region radius into account [38, 57]. The complex optimization algorithms were validated independently on different optimization problems and then adapted for the (rank-$L_r \circ$ rank-1) BTD.

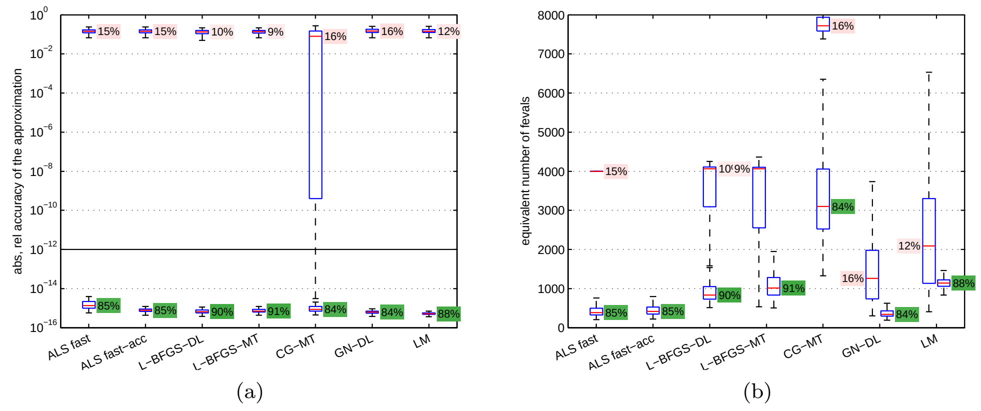


FIG. 5.1. *Accuracy and effort for rank-$(L_r, L_r, 1)$ block term decompositions generated with normally distributed pseudorandom factor matrices, where $L_1 = L_2 = L_3 = 3$. The algorithms are initialized with orthogonalized pseudorandom matrices.*

The results of the first experiment are shown in Figure 5.1. All algorithms have similar success ratios and when they are successful, they attain accuracies near machine precision with little spread. As expected, ALS fast-acc improves the accuracy of ALS fast slightly for hardly any added cost. The most efficient algorithm in terms of an equivalent number of function evaluations is GN-DL, followed closely by ALS. When the decompositions are unsuccessful, ALS suffers from weak degeneracy and uses the maximum number of iterations. In comparison, the nonlinear least squares methods at least converge to an undesired solution with relatively little effort.

Uniformly distributed pseudorandom factor matrices. This experiment is identical in setup to the first experiment, except that the tensors are now generated using uniformly distributed pseudorandom factor matrices, the entries of which lie in the open interval $(0, 1)$. Judging from Figure 5.2, this type of decomposition seems to be a much more difficult problem. Only the nonlinear least squares methods are able to find a correct decomposition sufficiently often. Again, the latter methods converge quite fast in comparison to the other algorithms, regardless if they were successful or not. The large spread on the unsuccessful cases of the other algorithms suggests they were converging to a correct solution, albeit very slowly.

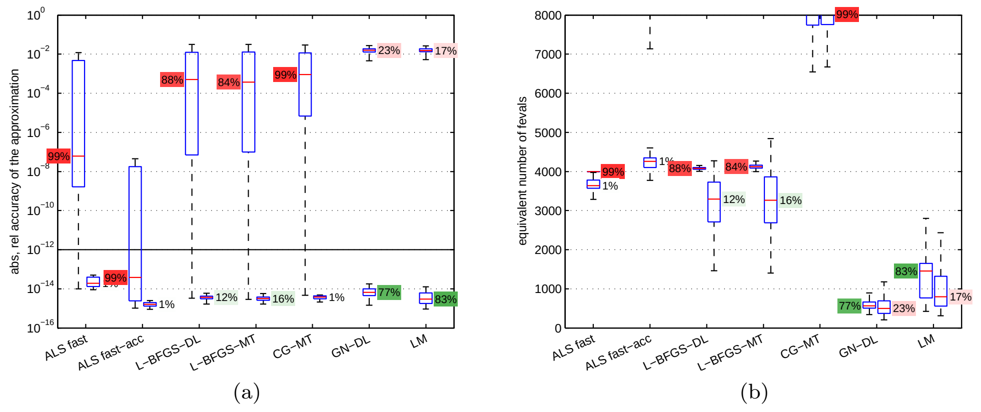


FIG. 5.2. *Accuracy and effort for rank-$(L_r, L_r, 1)$ block term decompositions generated with uniformly distributed pseudorandom factor matrices, where $L_1 = L_2 = L_3 = 3$. The algorithms are initialized with orthogonalized pseudorandom matrices.*

By using uniformly distributed pseudorandom factor matrices to initialize the algorithms, many algorithms perform much better. Figure 5.3 shows that the general unconstrained optimization techniques have drastically improved success ratios, although their efficiency still leaves a little to be desired. The nonlinear conjugate gradient method seems to have a little more trouble than limited-memory BFGS. This is likely due to its limited ability of approximating the objective function's Hessian. ALS is the only algorithm that still has trouble converging to the desired accuracy.

Different ranks. Now we examine what happens when the terms have different ranks. The factor matrices are generated with normally distributed pseudorandom numbers and correspond to a rank-$(L_r, L_r, 1)$ BTD in two terms where $L_1 = 4$ and $L_2 = 5$ in Figure 5.4 and in four terms where $L_1 = 1$, $L_2 = 2$, $L_3 = 3$ and $L_4 = 4$ in Figure 5.5. Comparing Figure 5.1 with 5.4 and 5.5 shows that the success ratios decrease significantly with an increasing number of terms of different rank. Two terms of different rank can create additional local minima. For example, in Figure

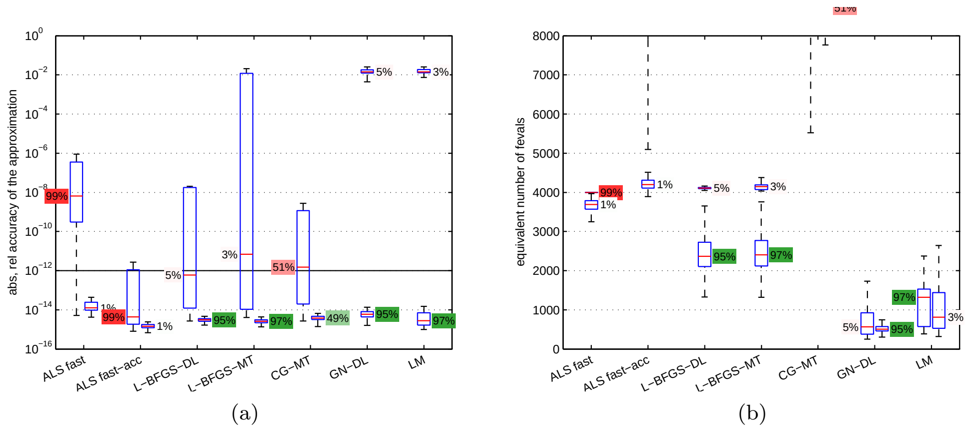


FIG. 5.3. *Accuracy and effort for rank-$(L_r, L_r, 1)$ block term decompositions generated with uniformly distributed pseudorandom factor matrices, where $L_1 = L_2 = L_3 = 3$. The algorithms are initialized with uniformly distributed pseudorandom matrices.*

5.4, 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. In general, we observe that the probability that all T terms in the optimization process converge to their corresponding term in the decomposition is about $\frac{1}{T!}$ for terms of distinct ranks, if these ranks are close to each other, relative to the smallest rank and the terms have similar norm. This rule of thumb coincides with a uniform probability of any permutation of the T terms being picked, where only one is correct.

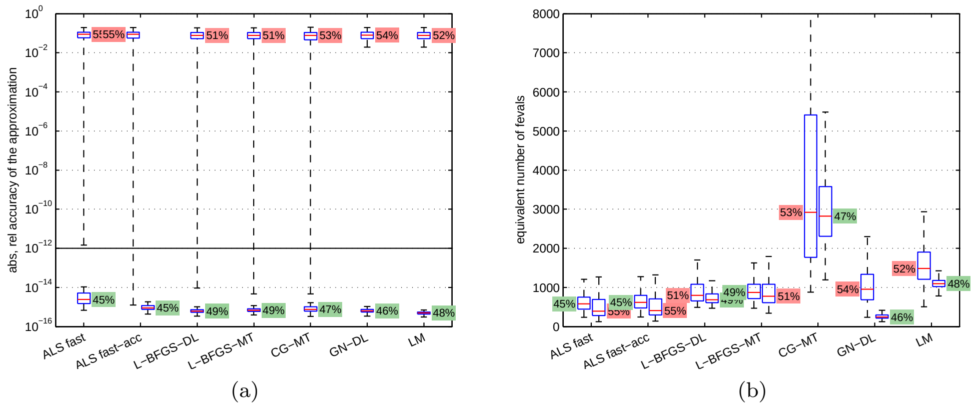
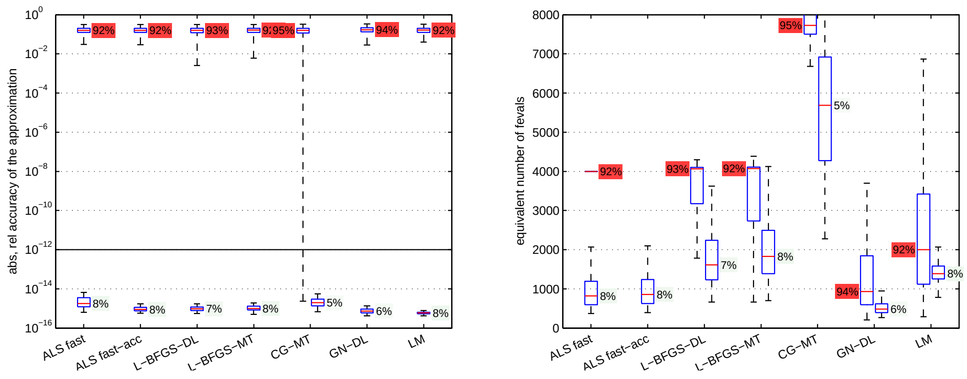


FIG. 5.4. *Accuracy and effort for rank-$(L_r, L_r, 1)$ block term decompositions generated with normally distributed pseudorandom factor matrices, where $L_1 = 4$ and $L_2 = 5$. The algorithms are initialized with orthogonalized pseudorandom matrices.*



(a) (b)

FIG. 5.5. *Accuracy and effort for rank-$(L_r, L_r, 1)$ block term decompositions generated with normally distributed pseudorandom factor matrices, where $L_1 = 1$, $L_2 = 2$, $L_3 = 3$ and $L_4 = 4$. The algorithms are initialized with orthogonalized pseudorandom matrices.*

5.1.2. With noise. In our next experiment, we add different levels of Gaussian noise to the tensors before decomposing them. We generate twenty sets of factor matrices using uniformly distributed random numbers, corresponding to rank-$(L_r, L_r, 1)$ block term decompositions in two terms where $L_1 = 4$ and $L_2 = 5$. We then generate their associated full $10 \times 11 \times 12$ tensors. After normalizing these tensors, we add white Gaussian noise so that we obtain ten noise levels for each of the twenty tensors. The noise levels correspond to the signal-to-noise ratios (SNR) of 5, 10, 15, 20, 30, 40, 50, 100, 200 and 300 dB, where the SNR is computed as $20\log_{10}(\|\mathcal{T}\|\|\mathcal{E}\|^{-1}) = -20\log_{10}(\|\mathcal{E}\|)$, in which $\mathcal{T}$ is the data tensor and $\mathcal{E}$ is the noise term. For each tensor and noise level, the algorithms are initialized using fifty orthogonalized pseudorandom sets of factor matrices of the same dimensions as in the tensor decomposition. A decomposition is now labeled as successful if the approximation is as accurate or more accurate than 85% of the SNR of the tensor being decomposed. For example, if the error $\hat{\mathcal{E}} = \mathcal{T} - \hat{\mathcal{T}}$ of the rank-$(L_r, L_r, 1)$ BTD approximation $\hat{\mathcal{T}}$ of a tensor $\mathcal{T}$ with 300 dB SNR is viewed as a noise term, then $-20\log_{10}(\|\hat{\mathcal{E}}\|)$ must be at least 255 dB for the decomposition to be labeled as successful. This threshold corresponds to an approximation error of $10^{-12.75}$ or less. Figure 5.6 shows the accuracy and effort of the decompositions labeled as successful and which percentage was successful for 10, 30, 50, 100, 200 and 300 dB SNR, while Figure 5.7 shows those for 5, 10, 15, 20, 30, 40 and 50 dB SNR. The thresholds for a decomposition to be labeled as successful are drawn as solid black lines in the figures showing the attained accuracy. The y-axis ranges from 0 to 5000 for each noise level in the figures showing the required effort, expressed as an equivalent number of function evaluations. For each of the latter plots, the corresponding noise level is displayed on the right-hand side.

In Figure 5.6, the success ratios surprisingly seem to increase as the SNR decreases. In fact, the nonlinear least squares methods reach a success ratio of 100% near 30 dB SNR. Looking at the range of 5 to 50 dB SNR in Figure 5.7, the interval in which the nonlinear least squares methods perform so well seems to be quite large: between 5 and 40 dB SNR. The general unconstrained optimization methods also benefit from a lower SNR. For very low SNRs around 5 to 10 dB, even ALS seems

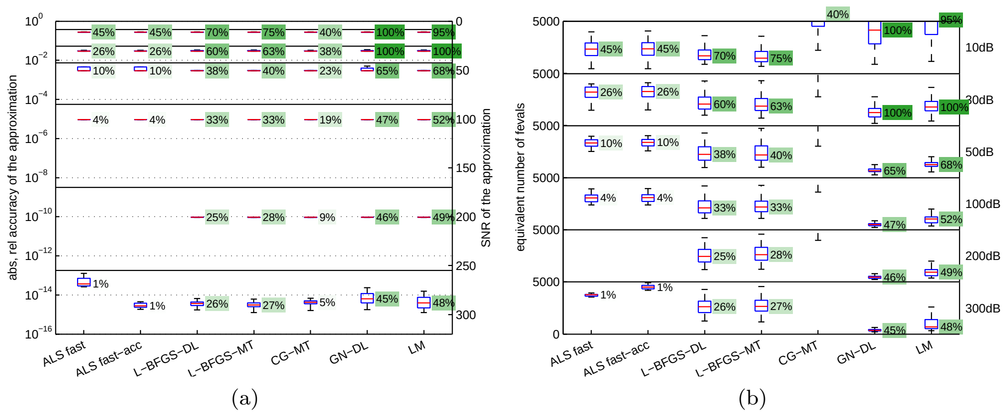


FIG. 5.6. *Accuracy and effort for rank-$(L_r, L_r, 1)$ block term decompositions generated with uniformly distributed pseudorandom factor matrices, where $L_1 = 4$ and $L_2 = 5$ and different noise levels were added to the tensors. The algorithms are initialized with orthogonalized pseudorandom matrices.*

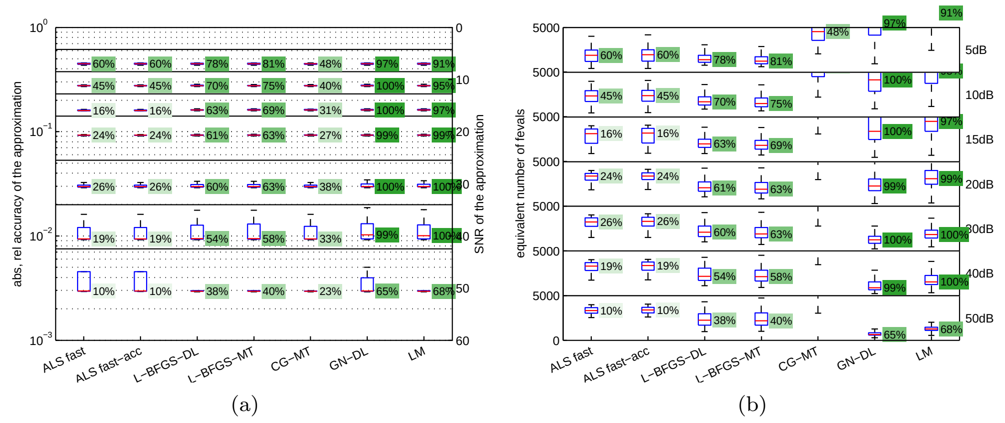
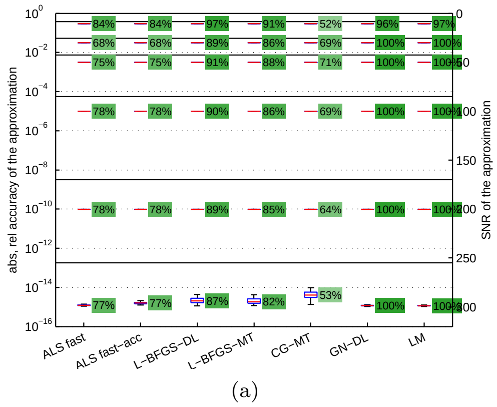


FIG. 5.7. *Accuracy and effort for rank-$(L_r, L_r, 1)$ block term decompositions generated with uniformly distributed pseudorandom factor matrices, where $L_1 = 4$ and $L_2 = 5$ and different noise levels were added to the tensors. The algorithms are initialized with orthogonalized pseudorandom matrices.*

to achieve a moderate success ratio. This may be misleading however, since at those noise levels the condition for a decomposition to be successful is relatively easily obtained by fitting the noise term. A possible explanation for the marked increase in success ratios of the other algorithms is that the noise term eliminates many local minima by providing additional descent directions in which the model can describe the noise instead of the data. Such directions of descent may only exist if the noise term is large enough and eventually allow the terms of the current iterate to 'cross' each other so that they converge to the correct terms in the decomposition. Future research might investigate other methods of introducing new descent directions, 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. For our last experiment, we generate tensors of dimensions $7 \times 8 \times 9 \times 10$ as rank-4 canonical polyadic decompositions. The first rank-one term in the decomposition is an outer product of four vectors generated using pseudorandom uniformly distributed numbers in the open interval (0,1). The next three rank-one terms are outer products of four complex vectors generated using a standard normal distribution for both their real and imaginary part. The tensors are then normalized to have unit norm and for each tensor, different levels of complex white Gaussian noise are added so that the resulting tensors' signal-to-noise ratios are 5, 10, 15, 20, 30, 40, 50, 100, 200 and 300 dB. We apply two types of initialization. For the first type, we generate fifty tensors as described above and for each tensor, we generate fifty initializations as orthogonalized complex pseudorandom sets of factor matrices. The second intialization is based on the generalized eigenvalue decomposition (GEVD) [33, 47, 48] and would in the noiseless case compute the exact decomposition. Since the latter initialization is deterministic, we generate 1000 tensors in the same way as before and only one such initialization per tensor. We use the same criteria for a decomposition to be labeled as successful as in the previous experiment. The accuracy and effort of the decompositions labeled as successful and which percentage was successful are shown in Figure 5.8 and 5.9 for the orthogonalized pseudorandom initializations and in Figure 5.10 for the GEVD-type initialization.



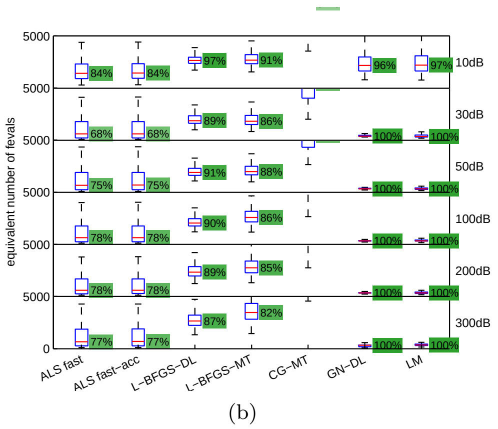


FIG. 5.8. *Accuracy and effort for complex rank-4 canonical polyadic decompositions, where different noise levels were added to the tensors. The algorithms are initialized with orthogonalized pseudorandom matrices.*

With random orthogonal initializations, the quasi-Newton methods have higher success ratios than ALS, but are also about two to four times more expensive in floating point operations. The nonlinear least squares methods are clearly the best choice here; their success ratios are close to unity, their equivalent cost in terms of number of function evaluations is the lowest among the algorithms, and the spread on the attained accuracy and required effort is very small. However, when a GEVD-type initialization is used, ALS shows markedly improved success ratios and is also very efficient. The nonlinear least squares methods are not trailing far behind in efficiency and still offer slightly higher success ratios than ALS, but the general unconstrained methods perform very poorly in combination with low to medium SNR.

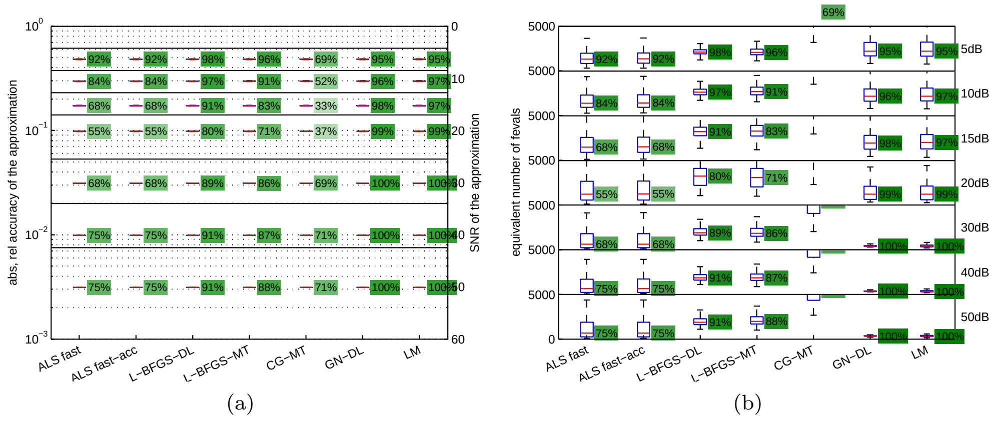


FIG. 5.9. *Accuracy and effort for complex rank-4 canonical polyadic decompositions, where different noise levels were added to the tensors. The algorithms are initialized with orthogonalized pseudorandom matrices.*

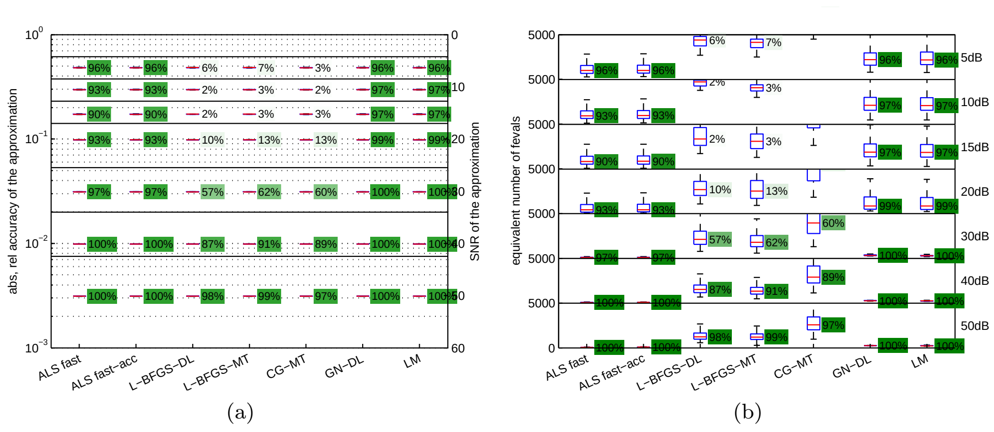


FIG. 5.10. *Accuracy and effort for complex rank-4 canonical polyadic decompositions, where different noise levels were added to the tensors. The algorithms are initialized with a GEVD-type initialization.*

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 alternating least-squares 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 reduce the number of conjugate gradient iterations with an effective 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 among the most robust; they converge to the global optimum significantly more often than competing methods and are also much less sensitive to the type of initialization. The algorithms discussed in this article were implemented in MATLAB and are available upon request.

Acknowledgments. Laurent Sorber is supported by a Flanders Institute of Science and Technology (IWT) doctoral scholarship. Marc Van Barel's research is supported by (1) the Research Council K.U.Leuven: (a) project OT/10/038 (Multi-parameter model order reduction and its applications), (b) CoE EF/05/006 Optimization in Engineering (OPTEC), and by (2) the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, Belgian Network DYSCO (Dynamical Systems, Control, and Optimization). Lieven De Lathauwer's research is supported by (1) the Research Council K.U.Leuven: (a) GOA-MaNet, (b) CoE EF/05/006 Optimization in Engineering (OPTEC), (c) CIF1, (d) STRT1/08/023, (2) F.W.O.: (a) project G.0427.10N, (b) Research Communities ICCoS, ANMMM and MLDM, (3) the Belgian Federal Science Policy Office: IUAP P6/04 (DYSCO, "Dynamical systems, control and optimization", 2007-2011), and by (4) EU: ERNSI. The scientific responsibility rests with the authors.

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. Sig. Proc., 39 (1991), pp. 1070–1087.

[2] E. ACAR, T. G. KOLDA, AND D. M. DUNLAVY, *An optimization approach for fitting canonical tensor decompositions*, tech. 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*, PhD thesis, University of Amsterdam, 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, 2009.

[9] P. COMON AND CHR. JUTTEN, *Handbook of Blind Source Separation: Independent component analysis and applications*, Academic Press, 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] ———, *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] ———, *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] ———, *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] ———, *A short introduction to tensor-based methods for factor analysis and blind source separation*, in Proc. of the 7th International Symposium on image and signal processing and analysis (ISPA 2011), Sept. 2011, pp. 558–563.

[16] 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.

[17] 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. Sig. Proc., 56 (2008), pp. 1562–1571.

[18] 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.

[19] 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.

[20] 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.

[21] J. E. Dennis, Jr. and H. H. W. Mei, *Two new unconstrained optimization algorithms which use function and gradient values*, J. Opt. Th. Appl., 28 (1979), pp. 453–482.

[22] 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.

[23] 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.

[24] C. Hayashi and F. Hayashi, *A new algorithm to solve PARAFAC-model*, Behaviormetrika, 9 (1982), pp. 49–60.

[25] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Phys., 6 (1927), pp. 164–189.

[26] ———, *Multiple invariants and generalized rank of a p-way matrix or tensor*, J. Math. Phys., 7 (1927), pp. 39–79.

[27] C. G. Khatri and C. R. Rao, *Solutions to some functional equations and their applications to characterization of probability distributions*, Sankhya: Indian J. of Stat., 30 (1968), pp. 167–180.

[28] 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.

[29] P. M. Kroonenberg, *Applied Multiway Data Analysis*, Wiley, 2008.

[30] 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.

[31] ———, *Rank, decomposition, and uniqueness for 3-way and N-way arrays*, Elsevier Science Publishers B.V., 1989, pp. 7–18.

[32] J. B. Kruskal, R. A. Harshman, and M. E. Lundy, *How 3-MFA data can cause degenerate PARAFAC solutions, among other relationships*, Elsevier Science Publishers B.V. (North-Holland), 1989, pp. 115–121.

[33] S. E. Leurgans, R. T. Ross, and R. B. Abel, *A decomposition for three-way arrays*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 1064–1083.

[34] B. C. Mitchell and D. S. Burdick, *Slowly converging PARAFAC sequences: swamps and two-factor degeneracies*, J. Chem., 8 (1994), pp. 155–168.

[35] J. J. Moré and D. J. Thuente, *Line search algorithms with guaranteed sufficient decrease*, ACM Trans. Math. Softw., 20 (1994), pp. 286–307.

[36] 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), 2008.

[37] D. Nion and L. De Lathauwer, *An enhanced line search scheme for complex-valued tensor decompositions. Application in DS-CDMA*, Signal Processing, 88 (2008), pp. 749–755.

[38] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer, second ed., 2006.

[39] P. Paatero, *A weighted non-negative least squares algorithm for three-way 'PARAFAC' factor analysis*, Chemometr. Intell. Lab. Syst., 38 (1997), pp. 223–242.

[40] ———, *The multilinear engine: A table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model*, J. Comp. Graph. Stat., 8

(1999), pp. 854–888.

[41] ———, *Construction and analysis of degenerate PARAFAC models*, J. Chem., 14 (2000), pp. 285–299.

[42] 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.

[43] ———, *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.

[44] 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.

[45] C. R. Rao and M. B. Rao, *Matrix algebra and its applications to statistics and econometrics*, World Scientific, 1998.

[46] R. Remmert, *Theory of Complex Functions*, Springer-Verlag, 1991.

[47] E. Sanchez and B. R. Kowalski, *Tensorial resolution: A direct trilinear decomposition*, J. Chem., 4 (1990), pp. 29–45.

[48] R. Sands and F. Young, *Component models for three-way data: An alternating least squares algorithm with optimal scaling features*, Psychometrika, 45 (1980), pp. 39–67.

[49] N. D. Sidiropoulos and R. Bro, *On the uniqueness of multilinear decomposition of N-way arrays*, J. Chem., 14 (2000), pp. 229–239.

[50] N. D. Sidiropoulos, R. Bro, and G. B. Giannakis, *Parallel factor analysis in sensor array processing*, IEEE Trans. Sig. Proc., 48 (2000), pp. 2377–2388.

[51] N. D. Sidiropoulos and G. Z. Dimić, *Blind multiuser detection in W-CDMA systems with large delay spread*, IEEE Sig. Proc. Letters, 8 (2001), pp. 87–89.

[52] N. D. Sidiropoulos, G. B. Giannakis, and R. Bro, *Blind PARAFAC receivers for DS-CDMA systems*, IEEE Trans. Sig. Proc., 48 (2000), pp. 810–823.

[53] A. Smilde, R. Bro, and P. Geladi, *Multi-Way Analysis with Applications in the Chemical Sciences*, Wiley, 2004.

[54] L. Sorber, M. Van Barel, and L. De Lathauwer, *Unconstrained optimization of real functions in complex variables.* Accepted for publication in SIAM J. on Optimization.

[55] 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.

[56] ———, *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.

[57] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM J. Numer. Anal., 20 (1983), pp. 626–637.

[58] G. Tomasi, *Practical and computational aspects in chemometric data analysis*, PhD thesis, The Royal Veterinary and Agricultural University, May 2006.

[59] G. Tomasi and R. Bro, *A comparison of algorithms for fitting the PARAFAC model*, Comp. Stat. Data An., 50 (2006), pp. 1700–1734.

[60] 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.

[61] ———, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.

[62] A. van den Bos, *Complex gradient and hessian*, IEE Proc. Vis., Image & Signal Process., 141 (1994), pp. 380–383.

[63] Ph. Wolfe, *Convergence conditions for ascent methods*, SIAM Rev., 11 (1969), pp. 226–235.

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. When this is not possible, we use the parameters of the experiments to convert their total effort expressed in floating point operations (flop) into an equivalent 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 floating point operations by a factor of four. A function evaluation of f_{BTD} (or $\mathcal{F}_{\mathrm{BTD}}$) then costs

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

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

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

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 $W^{\{n\}} = {V^{\{n\}}}^{\mathrm{H}} V^{\{n\}}$ 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

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

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. Limited memory BFGS with a dogleg trust-region strategy (L-BFGS-DL) requires one cogradient evaluation every iteration and one function evaluation every inner iteration. Let $\mathrm{it}_{\mathrm{dl}}$ be the number of dogleg iterations in each outer iteration, then the computational complexity is $\mathcal{O}(2(N + \mathrm{it}_{\mathrm{dl}})R\prod_{n=1}^N I_n)$ 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.

A.3. Nonlinear least squares. Let us first consider the exact nonlinear least squares methods. The matrix $J^{\text{H}}J$ requires $O(\frac{1}{2}R^2(\sum_{n=1}^{N} I_n)^2)$ memory cells. Precomputing the Gramians ${A^{(n)}}^{\text{H}}A^{(n)}$ 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 $\mathcal{O}(\sum_{n=1}^{N}\sum_{m=n+1}^{N}(\frac{1}{2}R^2 + 3R^2 I_n I_m))$ flop. In the hypercubical case $I = I_1 = \cdots = I_N$, the cost of evaluating $J^{\text{H}}J$ is then

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

In each iteration of the exact Levenberg–Marquardt algorithm (LM), a linear system involving the Gramian $J^{\text{H}}J$ 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 it_{lm}, the cost of the exact Levenberg–Marquardt algorithm is given by

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

The Gauss–Newton method with dogleg trust-region (GN-DL) 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 $J^{\text{H}}J$ 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 it_{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

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

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 product $J^{\mathrm{H}}J\cdot\boldsymbol{p}$, 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

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

and

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

respectively.