Research Article

JOURNAL OF CHEMOMETRICS

Received: 5 April 2010, Revised: 8 June 2010, Accepted: 14 June 2010, Published online in Wiley Online Library: 27 January 2011

(wileyonlinelibrary.com) DOI: 10.1002/cem.1335

A scalable optimization approach for fitting canonical tensor decompositions

Evrim Acar[a], Daniel M. Dunlavy[b] and Tamara G. Kolda[c]*

Tensor decompositions are higher-order analogues of matrix decompositions and have proven to be powerful tools for data analysis. In particular, we are interested in the canonical tensor decomposition, otherwise known as CANDECOMP/PARAFAC (CP), which expresses a tensor as the sum of component rank-one tensors and is used in a multitude of applications such as chemometrics, signal processing, neuroscience and web analysis. The task of computing CP, however, can be difficult. The typical approach is based on alternating least-squares (ALS) optimization, but it is not accurate in the case of overfactoring. High accuracy can be obtained by using nonlinear least-squares (NLS) methods; the disadvantage is that NLS methods are much slower than ALS. In this paper, we propose the use of gradient-based optimization methods. We discuss the mathematical calculation of the derivatives and show that they can be computed efficiently, at the same cost as one iteration of ALS. Computational experiments demonstrate that the gradient-based optimization methods are more accurate than ALS and faster than NLS in terms of total computation time. Copyright © 2011 John Wiley & Sons, Ltd.

Keywords: tensor decomposition; tensor factorization; CANDECOMP; PARAFAC; optimization

1. INTRODUCTION

A tensor is a multidimensional or N-way array. The canonical tensor decomposition [1,2] is a higher-order (i.e., $N \geq 3$) generalization of the matrix singular value decomposition (SVD) and has proved useful in many applications such as chemometrics, signal processing, neuroscience and web analysis; e.g., see References [3,4] for recent surveys of work in this area. We refer to the canonical decomposition as CANDECOMP/PARAFAC (CP) to recognize the original names given to it by Carroll and Chang [1] and Harshman [2], respectively.

CP is an analogue of the matrix SVD because it decomposes a tensor into the sum of component rank-one tensors. To understand CP, suppose that $\mathcal{Z}$ is a real-valued three-way tensor of size $I \times J \times K$ and of rank R, meaning that it can be expressed as the sum of no fewer than R components. Then its CP factorization is

$$\mathcal{Z} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r,$$

where $\circ$ denotes the vector outer product, and $\mathbf{a}_r \in \mathbb{R}^I$, $\mathbf{b}_r \in \mathbb{R}^J$ and $\mathbf{c}_r \in \mathbb{R}^K$ for $r = 1, \ldots, R$. Each element in the summation, $\mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$, is a rank-one tensor because it is the outer product of vectors. The *factor matrices* of CP are defined by $\mathbf{A} = [\mathbf{a}_1 \ \mathbf{a}_2 \ \cdots \ \mathbf{a}_R]$, $\mathbf{B} = [\mathbf{b}_1 \ \mathbf{b}_2 \ \cdots \ \mathbf{b}_R]$ and $\mathbf{C} = [\mathbf{c}_1 \ \mathbf{c}_2 \ \cdots \ \mathbf{c}_R]$, and the *factors* refer to the columns of the factor matrices. Specifically, the factors in mode one refer to the columns of $\mathbf{A}$, the factors in mode two to the columns of $\mathbf{B}$, and so on. The factors are analogous to the singular vectors in the SVD; however, a major difference is that CP factors are not orthonormal in each mode. In fact, it is possible that the rank is greater than the largest dimension, i.e., $R > \max\{I, J, K\}$, meaning that the factors in each mode are necessarily linear dependent. There is a well-known example of a $2 \times 2 \times 2$ tensor with rank three (see Reference [4§3.1]); since $\mathbf{A}$ is of size 2×3, its columns are necessarily linearly dependent. Therefore, in general, there is no guarantee of linear independence of the factors in each mode. Further, we do not assume that the factors are normalized to length one, although CP can be formulated in this way; we assume for convenience that any multiplier is absorbed into the factors. Despite CP's lack of orthogonality or even linear independence of factors in each mode, Kruskal [5] and others (see Reference [4§3.2] for a survey) have shown that it does have the advantage of uniqueness, up to permutation and scaling, under mild conditions. In contrast, it is well known that uniqueness of the SVD (up to sign) is due to its orthogonality constraints and even then is not unique when multiple singular values are equal. It is perhaps because of CP's uniqueness property that it often correctly describes underlying generating phenomena in data; this is particularly true in the modeling of fluorescence excitation-emission measurements [6] commonly used in chemistry.

In terms of computing CP, the first question, not directly addressed in this paper, is the choice of R, the number of compo-

* *Correspondence to: Tamara G. Kolda, Sandia National Laboratories, MS 9159, P.O. Box 969, Livermore, CA 94551-0969, USA. E-mail: tgkolda@sandia.gov*

a *E. Acar*
National Research Institute of Electronics and Cryptology (TUBITAK-UEKAE), Gebze, Turkey

b *D. M. Dunlavy*
Computer Science & Informatics Department, Sandia National Laboratories, Albuquerque, New Mexico, USA

c *T. G. Kolda*
Informatics and Systems Assessments Department, Sandia National Laboratories, Livermore, California, USA

nent rank-one tensors. We generally do not know the tensor rank and its computation is NP-complete [4]. Since we cannot know the rank in advance, we often face the problem of *overfactoring*, i.e., computing CP when R is greater than the rank of the tensor. In practice, the user tries different values of R and picks the 'best' based on some criteria e.g., model fit and the core consistency (CORCONDIA) diagnostic [7]. However, there is a trade-off between the model fit and core consistency for noisy data sets. Core consistency should be low for values of R exceeding the necessary number of component rank-one tensors to describe the low-rank multilinear structure in the data, so increasing values of R are tested until the core consistency drops. On the other hand, as R increases, the model fit typically improves. In practice, one must balance the improvement of fit with the drop in core consistency resulting in the very real possibility of choosing a larger value of R than is needed to capture the underlying multilinear structure. As an example of using the core consistency diagnostic, see Reference [8] for further discussion in the context of extracting brain activities through the analysis of EEG (electroencephalography) signals without any prior information about the number of activities. Another issue, not addressed in this paper, is that computing CP can be difficult because some tensors may have approximations of a lower rank that are arbitrarily close in terms of fit; this leads to *degeneracy* and can cause problems in practice [9–12].

In this paper we focus on the question of how to compute the CP factor matrices for a given value of R (not necessarily equal to the rank of the tensor). The typical method for finding the CP components is alternating least-squares (ALS) optimization, as proposed in the original CP papers [1,2]. The essential idea in ALS is to start with an initial guess for the factor matrices, solve a least-squares problem for $\mathbf{A}$ while holding $\mathbf{B}$ and $\mathbf{C}$ fixed, then fix $\mathbf{C}$ and the new $\mathbf{A}$ to solve for $\mathbf{B}$, and so on. This is the method of choice because of its speed and ease of implementation. Unfortunately, ALS often fails to obtain the underlying structure in the data, especially in the case of overfactoring. Another promising alternative that we discuss is a nonlinear least-squares (NLS) formulation; the NLS approach is superior to ALS in terms of finding the underlying structure, but significantly slower. Here, we propose using a general gradient-based optimization (OPT) formulation. We present the objective function, formulate the derivatives, and discuss computational issues such as regularization. Numerical studies indicate that OPT is faster than NLS in terms of total computation time without sacrificing accuracy.

Before we continue, we direct the reader to other alternative algorithms that have been proposed over the years with a goal of improving the convergence rate of ALS and its robustness to overfactoring. Faber, Bro and Hopke [13] compared ALS with a number of competing algorithms: direct trilinear decomposition (DTLD) [14–20], alternating trilinear decomposition (ATLD) [21], self-weighted alternating trilinear decomposition (SWATLD) [22,23], pseudo-alternating least-squares (PALS) [24], alternating coupled vectors resolution (ACOVER) [25], alternating slice-wise diagonalization (ASD) [26] and alternating coupled matrices resolution (ACOMAR) [27]. It is shown that while none of the algorithms is better than ALS in terms of the quality of solution, ASD may be an alternative to ALS when the computation time is a priority. Recently, Tomasi and Bro [28] have compared ALS with some of the algorithms mentioned above (DTLD, ASD, SWATLD) as well as two NLS approaches, PMF3 [29] and dGN (damped Gauss–Newton) [30,28]. The performance results show that ASD, the best alternative in the previous study [13], is not as good as ALS in terms of the accuracy of the solution. On the other hand, current NLS-based approaches outperform ALS in terms of accuracy (specifically in the case of overfactoring) but at the expense of memory and time overhead, making NLS-based approaches intractable for large data sets. We use the experimental methodology of Reference [28] as the basis for the numerical results in this paper in which we compare ALS and NLS and an implementation of our proposed OPT methods. Tomasi [31] has also compared ALS and NLS along with both first-order (nonlinear conjugate gradient method) and second-order (Newton's method) optimization; we contrast his results with ours in the conclusion.

Other approaches have been proposed in the literature as well but not yet compared numerically in studies such as the ones mentioned above. De Lathauwer, De Moor and Vandewalle [32] cast CP as a simultaneous generalized Schur decomposition (SGSD) and this approach has been applied to overcoming the problem of degeneracy [12]. De Lathauwer [33] also developed a method based on simultaneous matrix diagonalization. Vorobyov *et al*. [34] formulated the CP problem using an objective function based on least absolute error instead of least-squares error in order to compute CP robustly in the presence of non-Gaussian noise.

The main contributions of this paper are summarized as follows:

- Exploring the formulation of CP as a general optimization problem, with a particular focus on first-order optimization methods, which promise better scalability. This formulation uses regularization in order to directly address the scaling indeterminacy.
- Directly calculating the gradient (without recourse to the Jacobian) for the general optimization formulation of CP and showing that the derivatives can be computed efficiently. The analysis can be used to obtain analogous formulations for other tensor decompositions.
- Extensively comparing the performance of several methods for computing CP on both real and synthetic third-order data. These studies indicate that the OPT approach advanced in this paper has advantages in comparison with ALS and NLS.

This paper is structured as follows. Section 2 presents the notation and basic operations used throughout the paper. The ALS method is reviewed in Section 3. The OPT method is presented in Section 4, with a focus on the formulation of the required derivatives, as well as discussion of practical issues such as regularization. We contrast the NLS approach in Section 5. We follow the experimental procedure of Tomasi and Bro [28] to compare the methods; the details of the methods that are employed and the results are presented in Section 6. Conclusions are discussed in Section 7. Detailed numerical results are available in Appendix B.

2. NOTATION

We generally use the notation of Reference [4], which was adapted from Reference [35]. Scalars are denoted by lowercase letters, e.g., a. Vectors are denoted by boldface lowercase letters, e.g., $\mathbf{a}$. Matrices are denoted by boldface capital letters, e.g., $\mathbf{A}$. Higher-order tensors are denoted by boldface Euler script letters, e.g., $\mathcal{X}$. The ith entry of a vector $\mathbf{a}$ is denoted by a_i, element (i, j) of a matrix $\mathbf{A}$ is denoted by a_{ij}, and element (i, j, k) of a third-order tensor $\mathcal{X}$ is denoted by x_{ijk}. The jth column of a matrix $\mathbf{A}$ is denoted $\mathbf{a}_j$. Indices typically range from 1 to their capital version, e.g., $i = 1, \ldots, I$. The nth element in a sequence is denoted by a

superscript in parentheses, e.g., $\mathbf{A}^{(n)}$ denotes the nth matrix in a sequence.

The *inner product* of two same-sized tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the sum of the products of their entries, i.e.,

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N} y_{i_1 i_2 \cdots i_N}.$$

The *norm* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the square root of its inner product with itself, i.e.,

$$\|\mathcal{X}\| = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle}.$$

For matrices (i.e., second-order tensors), $\|\cdot\|$ refers to the analogous Frobenius norm, and, for vectors (i.e., first-order tensors), $\|\cdot\|$ refers to the analogous two-norm.

An N-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is *rank one* if it can be written as the outer product of N vectors, i.e.,

$$\mathcal{X} = \mathbf{a}^{(1)} \circ \mathbf{a}^{(2)} \circ \cdots \circ \mathbf{a}^{(N)}.$$

The symbol '$\circ$' represents the vector outer product. This means that each element of the tensor is the product of the corresponding vector elements:

$$x_{i_1 i_2 \cdots i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} \cdots a^{(N)}_{i_N} \quad \text{for all } 1 \le i_n \le I_n.$$

The *Khatri–Rao product*† [36] is the 'matching columnwise' Kronecker product. Given matrices $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{J \times K}$, their Khatri-Rao product is denoted by $\mathbf{A} \odot \mathbf{B}$. The result is a matrix of size $(IJ) \times K$ and defined by

$$\mathbf{A} \odot \mathbf{B} = [\, \mathbf{a}_1 \otimes \mathbf{b}_1 \quad \mathbf{a}_2 \otimes \mathbf{b}_2 \quad \cdots \quad \mathbf{a}_K \otimes \mathbf{b}_K \,].$$

Recall that the Kronecker product of two vectors $\mathbf{a} \in \mathbb{R}^I$ and $\mathbf{b} \in \mathbb{R}^J$ is a vector of length IJ defined by

$$\mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_1 \mathbf{b} \\ a_2 \mathbf{b} \\ \vdots \\ a_I \mathbf{b} \end{bmatrix}.$$

The Khatri-Rao product has the following property [36]:

$$(\mathbf{A} \odot \mathbf{B})^{\mathsf{T}} (\mathbf{A} \odot \mathbf{B}) = \mathbf{A}^{\mathsf{T}} \mathbf{A} * \mathbf{B}^{\mathsf{T}} \mathbf{B},$$

where $*$ denotes the elementwise product. Furthermore, the pseudo-inverse of the Khatri–Rao product has special form [36]:

$$(\mathbf{A} \odot \mathbf{B})^{\dagger} = ((\mathbf{A}^{\mathsf{T}} \mathbf{A}) * (\mathbf{B}^{\mathsf{T}} \mathbf{B}))^{\dagger} (\mathbf{A} \odot \mathbf{B})^{\mathsf{T}},$$

where $\mathbf{A}^{\dagger}$ denotes the Moore–Penrose pseudo-inverse of $\mathbf{A}$ [37]. Recall that the pseudo-inverse of the transpose is the transpose of the pseudo-inverse, so

$$((\mathbf{A} \odot \mathbf{B})^{\mathsf{T}})^{\dagger} = (\mathbf{A} \odot \mathbf{B})((\mathbf{A}^{\mathsf{T}} \mathbf{A}) * (\mathbf{B}^{\mathsf{T}} \mathbf{B}))^{\dagger}. \quad (1)$$

Further, because the Khatri-Rao product is associative, this property can be extended to more than two matrices; see Reference [36] for further details.

Matricization, also known as *unfolding* or *flattening*, is the process of reordering the elements of an N-way tensor into a matrix. The mode-n matricization of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted by $\mathbf{X}_{(n)}$ and arranges the mode-n one-dimensional 'fibers' to be the columns of the resulting matrix. Specifically, tensor element $(i_1, i_2, \ldots, i_N)$ maps to matrix element (i_n, j) where

$$j = 1 + \sum_{\substack{k=1 \\ k \ne n}}^{N} (i_k - 1) J_k, \quad \text{with} \quad J_k = \prod_{\substack{m=1 \\ m \ne n}}^{k-1} I_m.$$

Note that $J_k = 1$ if $k = 1$ or if $k = 2$ and $n = 1$. Since matricization is just a rearrangement of the elements, clearly $\|\mathcal{X}\| = \|\mathbf{X}_{(n)}\|$ for $n = 1, \ldots, N$; see Reference [4§2.4] for further details on matricization.

The *n-mode (vector) product* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a vector $\mathbf{v} \in \mathbb{R}^{I_n}$ is denoted by $\mathcal{X} \times_n \mathbf{v}$. The result is of order $N - 1$, i.e., the size is $I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N$. Elementwise,

$$(\mathcal{X} \times_n \mathbf{v})_{i_1 \cdots i_{n-1} i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} \, v_{i_n}.$$

A tensor may be multiplied by multiple vectors at once. For example, assume $\mathbf{v}^{(n)} \in \mathbb{R}^{I_n}$ for $n = 1, \ldots, N$. Then we use the new notation $\bigtimes$ to denote multiplication in multiple modes. Multiplication in all modes results in a scalar, i.e.,

$$\begin{aligned} \mathcal{X} \bigtimes_{n=1}^{N} \mathbf{v}^{(n)} &\equiv \mathcal{X} \times_1 \mathbf{v}^{(1)} \times_2 \mathbf{v}^{(2)} \cdots \times_N \mathbf{v}^{(N)} \\ &= \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N} v^{(1)}_{i_1} v^{(2)}_{i_2} \cdots v^{(N)}_{i_N}. \end{aligned}$$

Multiplication in every mode except mode n results in a vector of length I_n, i.e.,

$$\mathcal{X} \bigtimes_{\substack{m=1 \\ m \ne n}}^{N} \mathbf{v}^{(m)} = \mathbf{X}_{(n)} \mathbf{v}^{(-n)},$$

where

$$\mathbf{v}^{(-n)} \equiv \mathbf{v}^{(N)} \otimes \cdots \otimes \mathbf{v}^{(n+1)} \otimes \mathbf{v}^{(n-1)} \otimes \cdots \otimes \mathbf{v}^{(1)}.$$

We also note that multiplication in every mode except n and p results in a matrix of size $I_n \times I_p$.

3. CP AND ALTERNATING LEAST-SQUARES

In the introduction, we presented CP for three-way tensors. Here, we present CP for general N-way tensors. Let $\mathcal{Z}$ be a real-valued

† We use the Khatri–Rao product as defined in Reference [38,36], which corresponds to columnwise Kronecker product. The Khatri–Rao product has also been defined as a block Kronecker product in the literature [39].

N-way tensor of size $I_1 \times I_2 \times \cdots \times I_N$. Given R (the number of components), our goal is to find a CP factorization [1,2] such that

$$\mathcal{Z} \approx \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)},$$

where $\mathbf{a}_r^{(n)} \in \mathbb{R}^{I_n}$ for $n = 1, \ldots, N$ and $r = 1, \ldots, R$. Note that this formulation does not have scalar weights for each component rank-one tensor; these are assumed to be absorbed into the factors. We use the 'Kruskal operator' shorthand notation of [40]:

$$[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] \equiv \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)},$$

where the *factor matrices* are defined as

$$\mathbf{A}^{(n)} = \left[\mathbf{a}_1^{(n)} \cdots \mathbf{a}_R^{(n)}\right]$$

and so are of size $I_n \times R$, for $n = 1, \ldots, N$. The columns of $\mathbf{A}^{(n)}$ are the *factors* for mode n. It is useful to note that $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ can be written in matricized form (see, e.g., Reference [40]) as

$$\left([\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\right)_{(n)} = \mathbf{A}^{(n)}(\mathbf{A}^{(-n)})^{\mathsf{T}},$$

where

$$\mathbf{A}^{(-n)} \equiv \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}.$$

The problem of computing CP is, given a tensor $\mathcal{Z}$ and a choice for R (not necessarily the rank of $\mathcal{Z}$), find the factor matrices $\mathbf{A}^{(n)}$ of size $I_n \times R$ for $n = 1, \ldots, N$. Using the 'Kruskal operator' notation defined above, we can formulate the problem of fitting CP as a least-squares optimization problem:

$$\min f(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \equiv \frac{1}{2} \left\| \mathcal{Z} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] \right\|^2. \tag{2}$$

The ALS method for CP was proposed in the original papers by Carroll and Chang [1] and Harshman [2], and still remains the primary workhorse algorithm today due to its speed and ease of implementation [28]. In this section, we derive ALS as it is typically done in the tensor factorization community; the next section includes an alternate derivation. The premise is to iteratively optimize one factor matrix at a time, rather than solving (2) for $\mathbf{A}^{(1)}$ through $\mathbf{A}^{(N)}$ simultaneously. We can think of this as a block nonlinear Gauss–Seidel approach because we are solving a nonlinear equation for a block of variables while holding all the other variables fixed. Therefore, at each inner iteration, the goal is to solve

$$\min_{\mathbf{A}^{(n)}} f(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}), \tag{3}$$

for some particular fixed n, while holding all the other factor matrices constant. We can rewrite the equation in matrix form as

$$\min_{\mathbf{A}^{(n)}} \frac{1}{2} \left\| \mathbf{Z}_{(n)} - \mathbf{A}^{(n)}(\mathbf{A}^{(-n)})^{\mathsf{T}} \right\|^2.$$

With all but one factor matrix fixed, the problem reduces to a linear least-squares problem, and the exact solution is given by

$$\mathbf{A}^{(n)} = \mathbf{Z}_{(n)} \left((\mathbf{A}^{(-n)})^{\mathsf{T}}\right)^{\dagger}.$$

Naively, this requires computing the pseudo-inverse of a matrix of size $\prod_{\substack{m=1 \\ m \neq n}}^{N} I_m \times R$. However, from (1), this can be simplified as follows. Define

$$\mathbf{\Upsilon}^{(n)} = \mathbf{A}^{(n)\mathsf{T}}\mathbf{A}^{(n)} \quad \text{for } n = 1, \ldots, N. \tag{4}$$

Then

$$\mathbf{A}^{(n)} = \mathbf{Z}_{(n)}\mathbf{A}^{(-n)}(\mathbf{\Gamma}^{(n)})^{\dagger}, \tag{5}$$

where

$$\mathbf{\Gamma}^{(n)} = \mathbf{\Upsilon}^{(1)} * \cdots * \mathbf{\Upsilon}^{(n-1)} * \mathbf{\Upsilon}^{(n+1)} * \cdots * \mathbf{\Upsilon}^{(N)}. \tag{6}$$

Note that this formulation only requires computing the pseudo-inverse of a matrix of size $R \times R$. The ALS procedure for CP is well known; see, e.g., Reference [36].

4. OPTIMIZATION FOR CP

As an alternative to the ALS approach for CP, we propose solving for all the factor matrices simultaneously using a gradient-based optimization approach. (It is of course also possible to exploit the least-squares structure of the problem via an NLS method; this is discussed in Section 5.) We can consider the CP objective function f in (2) as a mapping from the cross-product of N two-dimensional vector spaces to $\mathbb{R}$, i.e.,

$$f : \mathbb{R}^{I_1 \times R} \otimes \mathbb{R}^{I_2 \times R} \otimes \cdots \otimes \mathbb{R}^{I_N \times R} \mapsto \mathbb{R}.$$

Therefore, we have a function of

$$P = R \sum_{n=1}^{N} I_n \tag{7}$$

variables. Although f in (2) is written as a function of matrices, it can be thought of as a scalar-valued function where the parameter vector $\mathbf{x}$ comprises the vectorized and stacked matrices $\mathbf{A}^{(1)}$ through $\mathbf{A}^{(N)}$, i.e.,

$$\mathbf{x} = \begin{bmatrix} \mathbf{a}_1^{(1)} \\ \vdots \\ \mathbf{a}_R^{(1)} \\ \vdots \\ \mathbf{a}_1^{(N)} \\ \vdots \\ \mathbf{a}_R^{(N)} \end{bmatrix}. \tag{8}$$

In this view, $f : \mathbb{R}^P \mapsto \mathbb{R}$, and it is straightforward to derive the gradient, which we do in Section 4.1. We note that computation

of the Hessian of f is also straightforward (see, e.g., Reference [41] for details) and can be used for higher-order optimization methods, but we do not address this in our paper. We discuss the effect of the scaling and permutation indeterminacies of CP in Section 4.2, explaining the benefits of regularization and deriving the derivatives of the regularized objective. Once the derivatives are known, any first-order optimization method can be used. We use a generic nonlinear conjugate gradient (NCG) method and test its numerical performance in Section 6. Harshman has discussed using gradient-based optimization for computing CP, which at that time was impractical due to the limited computing resources [2]. Both Paatero [42] and Wang and Hopke [43] have previously proposed using specialized NCG methods for CP, although neither presented numerical studies. Our implementation is different, as described in Section 6.

4.1. CP gradient

We can assemble the gradient, a vector of size P, by calculating the partial derivative with respect to each $\mathbf{a}_r^{(n)}$ for $r = 1, \ldots, R$ and $n = 1, \ldots, N$. Note that the partial derivative $\frac{\partial f}{\partial \mathbf{a}_r^{(n)}}$ is a vector of length I_n. Theorem 4.1 specifies the partial derivative; the same result has appeared in Reference [44] in the context of non-negative tensor factorizations.

Theorem 4.1. *The partial derivatives of the objective function f in (2) are given by*

$$\frac{\partial f}{\partial \mathbf{a}_r^{(n)}} = -\left(\mathcal{Z} \underset{\substack{m=1\\m\neq n}}{\overset{N}{\times}} \mathbf{a}_r^{(m)}\right) + \sum_{\ell=1}^{R} \gamma_{r\ell}^{(n)} \mathbf{a}_\ell^{(n)}, \tag{9}$$

for $r = 1, \ldots, R$ and $n = 1, \ldots, N$, with $\gamma_{r\ell}^{(n)}$ defined as

$$\gamma_{r\ell}^{(n)} \equiv \prod_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)\mathsf{T}} \mathbf{a}_\ell^{(m)}. \tag{10}$$

Proof. It will prove useful to rewrite the objective function in (2) as three summands:

$$f(\mathbf{x}) = \frac{1}{2}\underbrace{\|\mathcal{Z}\|^2}_{f_1(\mathbf{x})} - \underbrace{\langle \mathcal{Z}, [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\rangle}_{f_2(\mathbf{x})} + \frac{1}{2}\underbrace{\left\|[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\right\|^2}_{f_3(\mathbf{x})}. \tag{11}$$

The first summand does not involve the variables; therefore,

$$\frac{\partial f_1}{\partial \mathbf{a}_r^{(n)}} = \mathbf{0},$$

where $\mathbf{0}$ is the zero vector of length I_n. The second summand is the inner product between the tensor $\mathcal{Z}$ and its CP approximation, given by

$$\begin{aligned} f_2(\mathbf{x}) &= \langle \mathcal{Z}, [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\rangle \\ &= \left\langle \mathcal{Z}, \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)} \right\rangle \\ &= \sum_{r=1}^{R} \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} z_{i_1 i_2 \cdots i_N} a_{i_1 r}^{(1)} a_{i_2 r}^{(2)} \cdots a_{i_N r}^{(N)} \\ &= \sum_{r=1}^{R} \left(\mathcal{Z} \underset{m=1}{\overset{N}{\times}} \mathbf{a}_r^{(m)}\right) \\ &= \sum_{r=1}^{R} \left(\mathcal{Z} \underset{\substack{m=1\\m\neq n}}{\overset{N}{\times}} \mathbf{a}_r^{(m)}\right)^{\mathsf{T}} \mathbf{a}_r^{(n)}. \end{aligned}$$

Writing f_2 this way makes it obvious that

$$\frac{\partial f_2}{\partial \mathbf{a}_r^{(n)}} = \left(\mathcal{Z} \underset{\substack{m=1\\m\neq n}}{\overset{N}{\times}} \mathbf{a}_r^{(m)}\right). \tag{12}$$

The third summand is

$$\begin{aligned} f_3(\mathbf{x}) &= \left\|[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\right\|^2 \\ &= \left\langle \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)}, \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)} \right\rangle \\ &= \sum_{k=1}^{R} \sum_{\ell=1}^{R} \prod_{m=1}^{N} \mathbf{a}_k^{(m)\mathsf{T}} \mathbf{a}_\ell^{(m)} \\ &= \prod_{m=1}^{N} \mathbf{a}_r^{(m)\mathsf{T}} \mathbf{a}_r^{(m)} + 2\sum_{\substack{\ell=1\\\ell\neq r}}^{R} \prod_{m=1}^{N} \mathbf{a}_r^{(m)\mathsf{T}} \mathbf{a}_\ell^{(m)} \\ &\quad + \sum_{\substack{k=1\\k\neq r}}^{R} \sum_{\substack{\ell=1\\\ell\neq r}}^{R} \prod_{m=1}^{N} \mathbf{a}_k^{(m)\mathsf{T}} \mathbf{a}_\ell^{(m)}. \end{aligned}$$

Therefore,

$$\begin{aligned} \frac{\partial f_3}{\partial \mathbf{a}_r^{(n)}} &= 2\left(\prod_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)\mathsf{T}} \mathbf{a}_r^{(m)}\right) \mathbf{a}_r^{(n)} \\ &\quad + 2\sum_{\substack{\ell=1\\\ell\neq r}}^{R} \left(\prod_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)\mathsf{T}} \mathbf{a}_\ell^{(m)}\right) \mathbf{a}_\ell^{(n)} \\ &= 2\sum_{\ell=1}^{R} \left(\prod_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)\mathsf{T}} \mathbf{a}_\ell^{(m)}\right) \mathbf{a}_\ell^{(n)}. \end{aligned} \tag{13}$$

Combining (12) and (13) yields the desired result. □

Observe that $\gamma_{r\ell}^{(n)}$ is indeed the (r, ℓ) entry of the matrix $\mathbf{\Gamma}^{(n)}$ defined in (6). These values can be computed as follows. Compute the following $R \times R$ matrices $\mathbf{\Upsilon}^{(n)}$ from (4) (one per mode), and then it is clear that

$$\mathbf{\Gamma}^{(n)} = \mathbf{\Upsilon}^{(1)} * \cdots * \mathbf{\Upsilon}^{(n-1)} * \mathbf{\Upsilon}^{(n+1)} * \cdots * \mathbf{\Upsilon}^{(N)} \quad \text{for } n = 1, \ldots, N. \tag{14}$$

In fact, we can rewrite the gradient in matrix form, as the following corollary shows.

Corollary 4.2. *The partial derivatives of the objective function f in* (2) *are given by*

$$\frac{\partial f}{\partial \mathbf{A}^{(n)}} = -\mathbf{Z}_{(n)}\mathbf{A}^{(-n)} + \mathbf{A}^{(n)}\mathbf{\Gamma}^{(n)}, \tag{15}$$

for $n = 1, \ldots, N$, *where* $\mathbf{\Gamma}^{(n)}$ *is defined in* (6).

Proof. Equation (9) in Theorem 4.1 can be rewritten as

$$\frac{\partial f}{\partial \mathbf{a}_r^{(n)}} = -\mathbf{Z}_{(n)}\mathbf{a}^{(-n)} + \mathbf{A}^{(n)}\gamma_r^{(n)},$$

for $r = 1, \ldots, R$. Note that this expression exploits the fact that $\mathbf{\Gamma}^{(n)}$ is symmetric. Associating each $r = 1, \ldots, R$ with the column of a matrix yields (15). □

Corollary 4.2 can also be derived starting from an NLS formulation, as has been done by Tomasi [31, Paper III, Equation (35)]. We show this derivation in the next section.

Here we also see an alternative derivation of the ALS updates. Setting the gradient of f with respect to $\mathbf{A}^{(n)}$ equal to zero yields

$$\mathbf{A}^{(n)}\mathbf{\Gamma}^{(n)} = \mathbf{Z}_{(n)}\mathbf{A}^{(-n)}.$$

The ALS equation for updating $\mathbf{A}^{(n)}$ given in (5) then follows immediately.

4.2. Regularizing the optimization formulation of CP

CP is known to be unique when it satisfies, e.g., the Kruskal conditions [5] (see Reference [4§3.2] for a survey), but only up to permutation and scaling of the factor matrices. In other words, a CP factorization is unchanged by permutation, i.e.,

$$[\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}]\!] = [\![\mathbf{A}^{(1)}\mathbf{\Pi}, \mathbf{A}^{(2)}\mathbf{\Pi}, \ldots, \mathbf{A}^{(N)}\mathbf{\Pi}]\!]$$

where $\mathbf{\Pi}$ is an $R \times R$ permutation matrix. Likewise, CP is unchanged by scaling, e.g.,

$$[\![\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}]\!] = \left[\!\!\left[2\mathbf{A}^{(1)}, \frac{1}{2}\mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}\right]\!\!\right].$$

The scaling indeterminacy means that there is a continuous manifold of equivalent solutions, which makes it difficult for optimization methods to find *the* solution because there is not just one. In fact, the Hessian of f (see Reference [41]) is singular at a solution. This lack of a locally unique solution can be corrected by modifying the objective function to include a Tikhonov regularization term:

$$\hat{f}(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \equiv \frac{1}{2}\left\|\mathcal{Z} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]\right\|^2 + \frac{\lambda}{2}\sum_{n=1}^{N}\left\|\mathbf{A}^{(n)}\right\|^2. \tag{16}$$

This is the same approach proposed by Paatero in his NLS formulation [29]. The regularization has the effect of encouraging the norms of the factor matrices to be equal, i.e.,

$$\left\|\mathbf{A}^{(1)}\right\| = \left\|\mathbf{A}^{(2)}\right\| = \cdots = \left\|\mathbf{A}^{(N)}\right\|.$$

To justify our claim of equal norms, we prove in Appendix A that if the factor matrices are fixed (e.g., a solution with perfect fit) except for scaling, then the effect of the regularization is to equalize the magnitude of the vectors within each component. Obviously the situation where the factors are not fixed is more complicated and the value of λ must be chosen carefully so that the regularization term does not negatively impact the fit. The permutation indeterminacy does lead to multiple equivalent minimizers of f, but they are isolated minimizers and so do not negatively impact the optimization.

Corollary 4.3. *The partial derivatives of the objective function* $\hat{f}$ *in* (16) *are given by*

$$\frac{\partial \hat{f}}{\partial \mathbf{A}^{(n)}} = -\mathbf{Z}_{(n)}\mathbf{A}^{(-n)} + \mathbf{A}^{(n)}\mathbf{\Gamma}^{(n)} + \lambda\mathbf{A}^{(n)}, \tag{17}$$

for $n = 1, \ldots, N$, *where* $\mathbf{\Gamma}^{(n)}$ *is defined in* (6).

The proof is straightforward and so is omitted. We compare both regularized and unregularized formulations in Section 6. We should note that regularized formulation may also mitigate the effect of degeneracy; however, it does not fully resolve the problem based on our preliminary tests. We do not address the degeneracy problem in this paper and leave the effect of regularization on degeneracy as a topic of future research. It is also worth noting that there are other regularization approaches that can be explored; for example, Navasca *et al*. [45] use regularization in the context of ALS, penalizing the change between factor matrices across iterations and varying the regularization parameter at each iteration.

5. NONLINEAR LEAST-SQUARES APPROACH FOR CP

Paatero [29] and Tomasi and Bro [31,30,28] have formulated the CP problem as an NLS problem, explicitly computing its Jacobian $\mathbf{J}$ or the normalized form $\mathbf{J}^\mathsf{T}\mathbf{J}$.

In this case, consider the CP problem as a nonlinear equation

$$F(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \equiv \mathcal{Z} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] = 0. \tag{18}$$

Thus, we have

$$F : \mathbb{R}^{I_1 \times R} \otimes \mathbb{R}^{I_2 \times R} \otimes \cdots \otimes \mathbb{R}^{I_N \times R} \mapsto \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}.$$

Clearly (18) has a solution when the data is noise-free and R is the rank of $\mathcal{Z}$. More generally, this can be solved in a least-squares sense using NLS approaches. The f from (2) is equivalent to the least-squares objective function, i.e., $f(\mathbf{x}) = \frac{1}{2}F(\mathbf{x})^\mathsf{T}F(\mathbf{x})$, with $\mathbf{x}$ as defined in (8). The derivative of F is given elementwise as follows.

Theorem 5.1. *The first partial derivative of* (18) *with respect to* $a_{jr}^{(n)}$ *is a tensor of size* $I_1 \times I_2 \times \cdots \times I_N$ *defined elementwise as*

$$\left(\frac{\partial F}{\partial a_{jr}^{(n)}}\right)_{i_1 i_2 \cdots i_N} = \begin{cases} -\prod_{\substack{m=1 \\ m \neq n}}^{N} a_{i_m r}^{(m)} & \text{if } j = i_n, \\ 0 & \text{if } j \neq i_n. \end{cases}$$

The proof is straightforward, and so it is omitted. Unfortunately, it is difficult to express this in tensor notation.

Instead, we can vectorize the input and output arguments in order to think of F as a simpler mapping:

$$F : \mathbb{R}^P \mapsto \mathbb{R}^Q,$$

where

$$Q = \prod_{n=1}^{N} I_n. \tag{19}$$

Let $\mathbf{J}$ denote the Jacobian of F. Then we can write $\mathbf{J}$ as a blocked matrix where

$$\mathbf{J} = [\mathbf{J}^{(1)} \quad \mathbf{J}^{(2)} \quad \cdots \quad \mathbf{J}^{(N)}],$$

and $\mathbf{J}^{(n)}$ is of size $Q \times RI_n$ for $n = 1, \ldots, N$. The matrix $\mathbf{J}^{(n)}$ is in turn divided into a series of R submatrices:

$$\mathbf{J}^{(n)} = [\mathbf{J}^{(n1)} \quad \mathbf{J}^{(n2)} \quad \cdots \quad \mathbf{J}^{(nR)}],$$

where

$$\mathbf{J}^{(nr)} = -\mathbf{a}_r^{(1)} \otimes \cdots \otimes \mathbf{a}_r^{(n-1)} \otimes \mathbf{I} \otimes \mathbf{a}_r^{(n+1)} \otimes \cdots \otimes \mathbf{a}_r^{(N)},$$

and $\mathbf{I}$ is the identity matrix of size $I_n \times I_n$. The blocks $\mathbf{J}^{(n)}$ can be computed efficiently as described in Reference [31].

The matrix $\mathbf{J}$ has NR structural nonzeros per row, i.e., the product of the number of modes in the tensor with the number of components in the factorization. For example, if $\mathcal{Z}$ is a tensor of size $5 \times 4 \times 3$ and there are $R = 2$ components, then the Jacobian nonzero pattern is shown in Figure 1. It has $Q = \prod_{n=1}^{3} I_n = 60$ rows, $P = R \sum_{n=1}^{3} I_n = 24$ columns, and $NRQ = 360$ nonzeros. Note that this figure is similar to Figure 1 in Reference [29] but the columns are ordered differently, corresponding to our methodology for vectorizing a set of factor matrices as in (8).

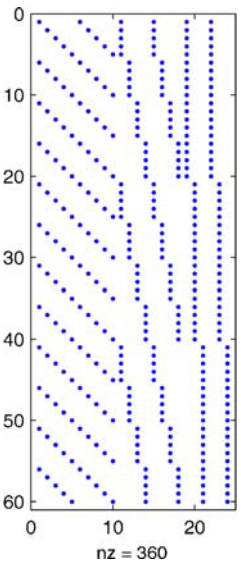

Figure 1. Jacobian nonzero pattern for a tensor of size $5 \times 4 \times 3$ with $R = 2$.

Using the Jacobian, (18) can be solved in an NLS sense via the Gauss–Newton method. This, however, requires solving a system with Jacobian matrix $\mathbf{J}$, which can be extremely large (size $Q \times P$, even though it is relatively sparse). Thus, Tomasi and Bro [30,28] have argued that it is preferable to work instead with $\mathbf{J}^\mathsf{T}\mathbf{J}$ using a Levenberg–Marquardt (LM) method. In this case, the matrix is only of size $P \times P$ and can be computed efficiently by exploiting the structure of $\mathbf{J}$ [31]. Moreover, the inclusion of a multiple of the identity matrix in the LM method serves a regularization function similar to that in (16).

As an aside, we note that the Jacobian can also be regarded as an operator. In particular, consider $\mathbf{J}^\mathsf{T}$ as a mapping from 'tensor' space to 'factor matrix' space:

$$\mathbf{J}^\mathsf{T} : \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} \mapsto \mathbb{R}^{I_1 \times R} \otimes \mathbb{R}^{I_2 \times R} \otimes \cdots \otimes \mathbb{R}^{I_N \times R}.$$

Therefore, if $\mathbf{J}^\mathsf{T}$ is applied to a $\mathcal{U}$, an N-way tensor of size $I_1 \times I_2 \times \cdots \times I_N$, then the result is a set of matrices $\mathbf{V}^{(1)}$ through $\mathbf{V}^{(N)}$ defined by

$$\mathbf{V}^{(n)} = \mathbf{U}_{(n)}\mathbf{A}^{(-n)}.$$

For example, it is well known that $\nabla f(\mathbf{x}) = -\mathbf{J}(\mathbf{x})^\mathsf{T} F(\mathbf{x})$. The 'vector' F is really a tensor of size $I_1 \times I_2 \times \cdots \times I_N$, so we will denote it by $\mathcal{F}$ to make that clear. Likewise, the 'vector' ∇f is really a set of N matrices of size $I_n \times R$ for $n = 1, \ldots, N$, so we will denote them by $\mathbf{G}^{(1)}$ through $\mathbf{G}^{(N)}$. Then we have

$$\begin{aligned}\mathbf{G}^{(n)} &= -\mathbf{F}_{(n)}\mathbf{A}^{(-n)} = \left(-\mathbf{Z}_{(n)} + \mathbf{A}^{(n)}\left(\mathbf{A}^{(-n)}\right)^\mathsf{T}\right)\mathbf{A}^{(-n)} \\ &= -\mathbf{Z}_{(n)}\mathbf{A}^{(-n)} + \mathbf{A}^{(n)}\boldsymbol{\Gamma}^{(n)}.\end{aligned}$$

This is, as expected, the gradient in (15).

6. NUMERICAL RESULTS

We compare the ALS, OPT and NLS approaches for computing CP, employing the data generation methodology of Reference [28]. The data, described in Section 6.1, is a collection of randomly generated three-way tensors with different ranks, varying levels of collinearity (defined below) between the factors, and multiple levels of homoscedastic and heteroscedastic noise. For each tensor, we compute a CP factorization with the number of components equal to the rank of the tensor and equal to one more than its rank (*overfactoring*). The details of the implementations and parameter settings are given in Section 6.2. In Section 6.3, we analyze the results. Our comparisons of ALS and NLS are consistent with Reference [28]. ALS can be remarkably fast‡ but is not always accurate, whereas NLS is accurate but is slower in our experiments. Our OPT methods are as accurate as NLS but faster. Timing comparisons are based on total computation time of each algorithm rather than iteration counts, since the expense of one iteration of each algorithm is different as discussed in detail in Section 6.2. We note that we do not consider Tucker compression (see, e.g., Reference [28]) in this paper, but compression can certainly be used with the OPT method as it has been for ALS and NLS.

‡ With a more strict stopping criterion, e.g., lower threshold for relative change in function value, ALS may get extremely slow.

6.1. Data

We test our methods by factorizing artificially generated tensors of varying size and rank. Specifically, we consider three-way cubic tensors of sizes 20, 50, 100 and 250. We let R_{true} denote the rank of the tensor (before adding noise) and use $R_{\text{true}} = 3$ and $R_{\text{true}} = 5$ in our tests. Following Reference [28], we factorize the test tensors using $R = R_{\text{true}}$ and $R = R_{\text{true}} + 1$ (*overfactoring*).

We generate test tensors following the procedures and parameters in Reference [28]. We randomly generate factor matrices, $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$ and $\mathbf{A}^{(3)}$ of appropriate size so that the *collinearity* of the factors in each mode is set to a particular value, C. This means that

$$\frac{\mathbf{a}_r^{(n)\mathsf{T}}\mathbf{a}_s^{(n)}}{\|\mathbf{a}_r^{(n)}\|\|\mathbf{a}_s^{(n)}\|} = C \tag{20}$$

for $r \neq s$, $r, s = 1, \ldots, R_{\text{true}}$, and $n = 1, \ldots, N$. We use $C = 0.5$ and $C = 0.9$ in our experiments and generate 20 sets of factors matrices for each combination of C and R_{true}; higher values of C make the problem more difficult. The goal is to recover these underlying factor matrices once they have been assembled into a tensor and noise has been added.

From each set of factor matrices, nine test tensors are created by adding different levels of homoscedastic (i.e., constant variance) and heteroscedastic (i.e., differing variance) noise as follows. We set $\ell_1 = 1, 5, 10$ and $\ell_2 = 0, 1, 5$ to be the desired noise ratios of homoscedastic and heteroscedastic noise, respectively (corresponding to the values used in Reference [28]). Let $\mathcal{N}_1, \mathcal{N}_2 \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be tensors with entries randomly chosen from a standard normal distribution. Then the test tensors are generated as follows. The original tensor is:

$$\mathcal{Z} = [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!].$$

Homoscedastic noise is added to produce:

$$\mathcal{Z}' = \mathcal{Z} + (100/\ell_1 - 1)^{-1/2}\frac{\|\mathcal{Z}\|}{\|\mathcal{N}_1\|}\mathcal{N}_1.$$

Finally, heteroscedastic noise is added and results in

$$\mathcal{Z}'' = \mathcal{Z}' + (100/\ell_2 - 1)^{-1/2}\frac{\|\mathcal{Z}'\|}{\|\mathcal{N}_2 * \mathcal{Z}'\|}\mathcal{N}_2 * \mathcal{Z}'.$$

The tensor $\mathcal{Z}'$ is used when $\ell_2 = 0$. Note that when $\ell_2 = 0$, only Gaussian noise has been added.

To summarize, we generate a total of 720 cubic three-way test tensors for sizes 20, 50, 100 and 250; and we do a total of 1440 CP calculations (using $R = R_{\text{true}}$ and $R = R_{\text{true}} + 1$). The 720 test tensors come from 2 true ranks ($R_{\text{true}} = 3, 5$), 2 collinearity levels ($C = 0.5, 0.9$), 20 sets of factor matrices per combination of C and R_{true} and 9 noise level combinations ($\ell_1 = 1, 5, 10$ and $\ell_2 = 0, 1, 5$). These parameters are summarized in Table I.

Table I. Summary of parameters used to generate data for experiments

Parameter	Variants	Values
Size ($I \times I \times I$)	4	20, 50, 100, 250
True Rank (R_{true})	2	3, 5
Collinearity (C)	2	0.5, 0.9
Replicates	20	Random
Homoscedastic Noise (ℓ_1)	3	1, 5, 10
Heteroscedastic Noise (ℓ_2)	3	0, 1, 5
Number of Components (R)	2	R_{true}, $R_{\text{true}} + 1$

6.2. Implementation details

In this section, CPALS, CPOPT(R) and CPNLS denote the *implementations* of the methods presented in Sections 3, 4 and 5, respectively. All experiments were performed using MATLAB v7.6. The details of the implementation of each method, as well as the expected computational cost, are discussed below. Initial points for all tests were generated using the n-mode singular vectors of the tensor (i.e., the `nvecs` command in the Tensor Toolbox [46]). We note that none of the optimization methods discussed here is guaranteed to find a global minimum—we can at best hope for a stationary point in the case of OPT and NLS. The convergence of ALS to a stationary point has not been proven [4].

6.2.1. CPALS

ALS is implemented in the `parafac_als` code from the Tensor Toolbox [47,46]. We used the default settings for the code except: (1) output to the screen was disabled and (2) the stopping conditions specified in Section 6.2.4 were used. Each iteration requires the computation of $\mathbf{A}^{(n)}$ for $n = 1, \ldots, N$ (see Equation 5). Since R is generally small in comparison to the size of the tensor, i.e., $R \ll I_n$ for $n = 1, \ldots, N$, it is assumed that the dominant computation in (5) is the matricized-tensor times Khatri–Rao product, i.e.,

$$\mathbf{Z}_{(n)}\mathbf{A}^{(-n)}. \tag{21}$$

There is a special function for this computation in the Tensor Toolbox called `mttkrp`. The primary cost in (21) is multiplying the matrix $\mathbf{Z}_{(n)}$ of size $I_n \times (Q/I_n)$, where Q is defined in (19), by the Khatri–Rao product of size $(Q/I_n) \times R$. Therefore, the computational cost, measured in terms of the number of operations, is $O(QR)$. Consequently, the cost of each outer iteration of ALS, which contains N computations of (21), is $O(NQR)$.

There are other versions of ALS in the literature such as the ones accelerated with different line searches or optimized implementations as suggested in Reference [39]. Much of the work on line searches for ALS has focused on either linear [48,2,31] or nonlinear [49] extrapolation of factor matrices from previous iterations aimed at speeding up the convergence of the algorithm and not at improving accuracy of the method. The ALS implementation we use is already the fastest algorithm in comparison to OPT and NLS approaches in our experiments; therefore, we have not included these possible alternatives.

6.2.2. CPNLS

For NLS, the damped Gauss–Newton (`dGN`) method for three-way arrays in PARAFAC3W [50] was used to solve CP formulated as in (18). We used the default settings for the `dGN` code except: (1) output to the screen was disabled, (2) compression was disabled and (3) the stopping conditions specified in 6.2.4 were used with all others disabled. This code implements the Levenberg and Marquardt (LM) method [51], which modifies the normal

equations, i.e.,

$$\left(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \lambda \mathbf{I}\right)\Delta \mathbf{x} = -\mathbf{J}^{\mathsf{T}}F, \tag{22}$$

updating λ each iteration. The Jacobian $\mathbf{J}$ is singular due to the scaling indeterminacy of CP [29,42,30,28]; therefore, the modified normal equations in the LM approach can be thought of as a form of regularization as discussed in Section 4.2 for the OPT approach. As noted in Section 5, the Jacobian is sparse and has a specific structure; therefore, $\mathbf{J}^{\mathsf{T}}\mathbf{J}$ can be computed efficiently but $\mathbf{J}^{\mathsf{T}}F$ still requires $O(NQR)$ operations. For three-way arrays, the primary expense at each iteration of NLS is solving the system in (22) at a cost of $O(P^3)$ operations, where P is as defined in (7). The dGN method in the PARAFAC3W code was selected because it implements LM as described in Reference [28] and is freely available.

6.2.3. CPOPT

The OPT and OPTR methods were implemented using the Tensor Toolbox. At each iteration, the function in (2) and the gradient in (15) for $n = 1, \ldots, N$ must be computed. This is similar to the computation of the ALS update in (5). The pseudo-code for the computation is shown in Figure 2. As with ALS, the matricized-tensor times Khatri-Rao product (`mttkrp`) computation given in (21) dominates the expense of the calculation in the OPT methods. Therefore, the cost per function/gradient evaluation is $O(NQR)$. Recall that the only difference in OPTR is the addition of the regularization term in the function and gradient, which has little impact on the cost per evaluation; see (16) and (17).

```
Upsilon = cell(N,1);
Gamma = cell(N,1);
G = cell(N,1);

for n = 1:N
  Upsilon{n} = A{n}' * A{n};
end

for n = 1:N
  Gamma{n} = ones(R,R);
  for m = [1:n-1,n+1:N]
    Gamma{n} = Gamma{n} .* Upsilon{m};
  end
end

U = mttkrp(Z,A,1);
V = A{1} .* U;
f2 = sum(V(:));

G{1} = -U + A{1} * Gamma{1};
for n = 2:N
  U = mttkrp(Z,A,n);
  G{n} = -U + A{n} * Gamma{n};
end

W = Gamma{1} .* Upsilon{1};
f3 = sum(W(:));

f = .5 * nrmZsqr - f2 + .5 * f3;
```

Figure 2. MATLAB code for calculating function value (`f`) and gradient (`G`) for CPOPT for a given tensor $\mathcal{Z}$ and factorization $[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$. We assume the squared norm of $\mathcal{Z}$ has been precomputed and is stored as `nrmZsqr`.

Gradient-based optimization is performed using our own implementation of the nonlinear conjugate gradient (NCG) method with Hestenes–Stiefel (HS) updates in Poblano Toolbox [52]. The Moré–Thuente line search from MINPACK[§] was used for globalization of the NCG method [53]. Exact line search methods have also been studied in computing CP using ALS [31], where $2N - 1$ function values are required. However, in our experiments, the Moré–Thuente line search averaged only 2.5 function evaluations per iteration, so we have not explored exact line searches in this work. For all experiments, the Moré–Thuente line search parameters used were as follows: 10^{-4} for the function value tolerance, 10^{-2} for the gradient norm tolerance, a starting search step length of 1 and a maximum of 20 iterations. Note that any first-order optimization could potentially be substituted for NCG. We have also employed a limited memory quasi-Newton method using BFGS updates (L-BFGS) [54] for this problem, but the differences between this method and NCG were negligible in our experiments. For NCG (and also L-BFGS in the form of the initial guess for the Hessian), preconditioning is extremely important when the variables are of different magnitudes. A treatment of preconditioning for NCG can be found in Reference [55]. In the experiments of the present work, no preconditioning was needed in the form of scaling the variables, because all values were of same order of magnitude.

We noted previously that both Paatero [42] and Wang and Hopke [43] proposed using specialized NCG, and here we make a few observations on the differences. CPOPT uses HS updates in contrast to Fletcher–Reeves (FR) updates for both of their methods. In our preliminary tests, HS outperformed FR. Both of their methods also use backtracking line searches, in contrast to CPOPT, which uses the more sophisticated Moré–Thuente line search.

It is important to observe that the cost per function/gradient evaluation in OPT is equivalent to one outer iteration of ALS; therefore, even though we cannot predict how many iterations of ALS or function evaluations in OPT we require for a given CP factorization, we might expect the overall computational costs of OPT and ALS to be on the same order of magnitude.

For CPOPTR, the regularization parameter was selected to be $\lambda = 0.02$ for experiments with $C = 0.5$ and $\lambda = 0.0001$ for experiments with $C = 0.9$. More work is needed to investigate how to best choose λ or how to best modify λ per iteration as in Reference [45].

6.2.4. Stopping conditions

In order to produce comparable results, all implementations were modified to share a common stopping criterion, the one commonly used for termination in ALS methods, which is the relative change in the function value of f in (2). Specifically, the algorithm stops when

$$\frac{|f_{\text{current}} - f_{\text{previous}}|}{f_{\text{previous}}} \leq 10^{-6},$$

where f_{current} and f_{previous} are the values of f at the current and previous iterations, respectively.

[§] Adapted for MATLAB by Dianne P. O'Leary.

Table II. Speed and accuracy comparison with collinearity $C = 0.5$

	Time (sec)			
Size	**CPALS**	**CPNLS**	**CPOPT**	**CPOPTR**
$20 \times 20 \times 20$	0.5 ± 1.0	0.3 ± 0.3	0.3 ± 0.2	0.2 ± 0.1
$50 \times 50 \times 50$	0.3 ± 0.3	2.0 ± 2.6	0.7 ± 0.5	0.5 ± 0.1
$100 \times 100 \times 100$	1.7 ± 1.1	11.5 ± 11.5	5.6 ± 3.6	4.3 ± 1.3
$250 \times 250 \times 250$	26.6 ± 9.1	143.9 ± 125.0	83.5 ± 35.2	81.9 ± 22.8
	Accuracy (%)			
Size	**CPALS**	**CPNLS**	**CPOPT**	**CPOPTR**
$20 \times 20 \times 20$	78.8	99.0	99.9	100.0
$50 \times 50 \times 50$	65.7	99.0	100.0	100.0
$100 \times 100 \times 100$	63.5	97.9	100.0	100.0
$250 \times 250 \times 250$	62.2	99.0	100.0	100.0

In addition to the relative change in the function value, we use the following stopping conditions for the individual solvers. For CPALS, the maximum number of iterations is set to 10^4. For CPNLS, the tolerance on the infinity norm of the gradient is set to 10^{-9}, and the maximum number of iterations is set to 10^3 based on the values used in Reference [28]. For CPOPT and CPOPTR, the tolerance on the two-norm of the gradient divided by the number of entries in the gradient is set to 10^{-8}, the maximum number of iterations is set to 10^3 and the maximum number of function evaluations is set to 10^4. We note that all runs stopped by satisfying the condition for the relative change in the function value, except for five runs where CPOPT reached the tolerance for the gradient.

6.3. Analysis

Detailed numerical results are provided in Appendix B. In this section, we consider the results summarized according to different experimental parameters.

All timings are reported for a Linux Workstation with a Quad-Core Intel Xeon 2.5GHz processor and 9GB RAM. Throughout, we report the time per CP calculation, and timings are written as $a \pm b$ where a is the average time and b is the sample standard deviation.

The accuracy is the percentage of runs that a given implementation is able to recover the original set of factor matrices. Specifically, we say that the factors have been recovered if the *congruence* for every component is above a threshold of 0.97 ($\approx 0.99^3$). The congruence between two rank-one tensors, $\mathcal{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ and $\mathcal{Y} = \mathbf{p} \circ \mathbf{q} \circ \mathbf{r}$, is defined as [28]:

$$\text{cong}(\mathcal{X}, \mathcal{Y}) = \frac{|\mathbf{a}^T\mathbf{p}|}{\|\mathbf{a}\|\|\mathbf{p}\|} \times \frac{|\mathbf{b}^T\mathbf{q}|}{\|\mathbf{b}\|\|\mathbf{q}\|} \times \frac{|\mathbf{c}^T\mathbf{r}|}{\|\mathbf{c}\|\|\mathbf{r}\|}. \tag{23}$$

Since there is sign ambiguity among the vectors comprising each component rank-one tensor, i.e., $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c} = (-\mathbf{a}) \circ (-\mathbf{b}) \circ \mathbf{c}$, absolute values are used in the numerators of (23). Since CP is unique only up to a permutation of the component rank-one tensors, we consider all permutations, choosing the one that results in the greatest sum of congruences.

6.3.1. *Tensor size*

Across all sizes, CPOPT is more accurate than CPALS and faster than CPNLS. Table II shows accuracy and timing results with $C = 0.5$ held constant; this means that each cell in the table corresponds to 360 test tensors and 720 factorizations.

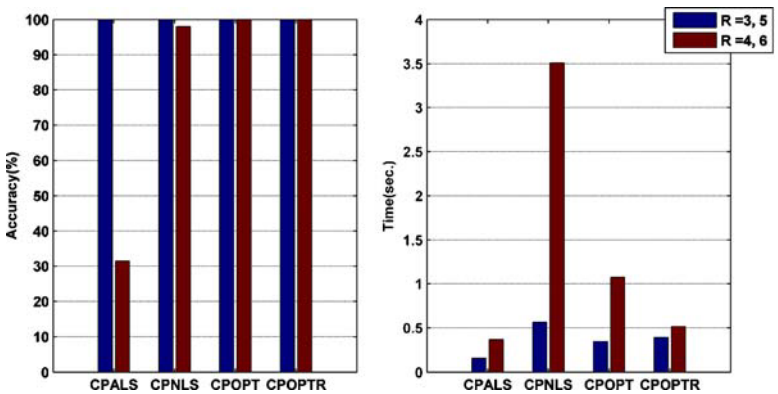

Figure 3. Speed and accuracy comparison for extracting the number of true underlying components (blue) and overfactoring (red) on tensors of size $50 \times 50 \times 50$ and collinearity $C = 0.5$.

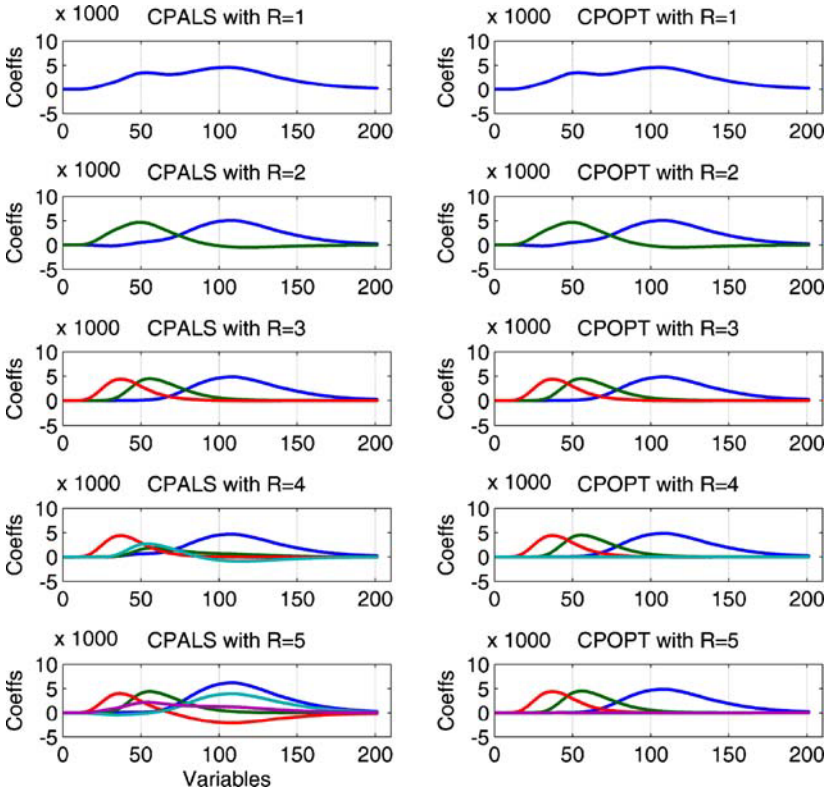

Figure 4. Factors corresponding to the emission mode of the amino acid fluorescence data set [48,56]. Plots are shown for CPALS (left) and CPOPT (right).

Recall that the cost for one iteration of CPALS is equal to that for one gradient calculation for CPOPT. Therefore, it is not surprising that the cost in time for these methods is of the same order of magnitude. In general, it seems that CPOPT is about three times slower than CPALS. Further, CPOPTR is slightly faster than CPOPT through the use of regularization. CPNLS is slower than the other methods because it must solve the modified Gauss–Newton equation at each inner iteration; moreover, it has a large standard deviation. The results in the next subsection indicate that it is slower in the case of overfactoring.

In terms of accuracy, both CPNLS and CPOPT are essentially perfect, but CPALS only obtains accuracies of 62-79%. Once again, results in the next subsection indicate that the accuracy of CPALS suffers in the case of overfactoring.

6.3.2. *Number of components in the factorization*

Many of the differences in the speed and accuracy of the methods can be attributed to their performance in the case of overfactoring ($R = R_{\text{true}} + 1$). Figure 3 illustrates the accuracy and timing results for tensors of size $50 \times 50 \times 50$ with $C = 0.5$, separating the $R = R_{\text{true}}$ (blue) and $R = R_{\text{true}} + 1$ (red) cases. Observe that the accuracy of CPALS is 100% in the case that $R = R_{\text{true}}$ but falls to 30% when $R = R_{\text{true}} + 1$. All the methods are more computationally expensive in the case of overfactoring because it takes more iterations/function evaluations to converge for each method. Regularization helps with overfactoring (for $C = 0.5$) and the increase in the number of function evaluations for CPOPTR is, on average, less than the increase for CPOPT.

Table III. Norms of the component rank-one tensors and timings in the case of correct R and overfactoring for the results shown in Figure 4

	CPALS						CPOPT					
	Component Norm ($\times 10^4$)					Time (sec)	Component Norm ($\times 10^4$)					Time (sec)
R	1	2	3	4	5		1	2	3	4	5	
3	3.3	2.3	2.1			0.5	3.3	2.3	2.1			1.5
4	3.3	1.1	2.1	1.4		6.2	3.3	2.3	2.1	10^{-4}		1.7
5	4.3	2.3	2.3	2.7	1.5	60.0	3.3	2.3	2.1	10^{-4}	10^{-4}	1.9

Table IV. Speed and accuracy comparison with collinearity $C = 0.9$

	Time (sec)			
Size	**CPALS**	**CPNLS**	**CPOPT**	**CPOPTR**
$20 \times 20 \times 20$	1.1 ± 0.8	0.5 ± 0.5	0.7 ± 0.3	0.7 ± 0.3
$50 \times 50 \times 50$	1.8 ± 0.9	3.3 ± 3.4	1.5 ± 0.7	1.5 ± 0.6
$100 \times 100 \times 100$	14.2 ± 5.5	19.2 ± 20.4	12.9 ± 5.0	12.9 ± 4.8
$250 \times 250 \times 250$	242.1 ± 99.9	198.7 ± 171.8	231.5 ± 82.8	229.3 ± 82.7
	Accuracy (%)			
Size	**CPALS**	**CPNLS**	**CPOPT**	**CPOPTR**
$20 \times 20 \times 20$	29.0	31.8	32.2	32.6
$50 \times 50 \times 50$	65.6	69.9	69.9	69.9
$100 \times 100 \times 100$	73.1	77.5	79.4	79.7
$250 \times 250 \times 250$	79.9	87.1	89.6	89.4

To further explore the overfactoring phenomena, we consider publicly available data [56] comprising five chemical samples measured by fluorescence at 61 excitation and 201 emission wavelengths forming, a third-order tensor with the following modes: samples, emission wavelengths, and excitation wavelengths. Each of the five samples contains different amounts of three amino acids [48]. It is known that the number of true underlying components in the data is three; therefore, the spectra of these chemicals in emission and excitation modes are accurately captured by computing a CP factorization with $R = 3$ components. Even though $R = 3$ corresponds to the number of true underlying components in the data, there is an artifact due to the Rayleigh scatter [48] such that an extra component may partially capture it. We compute CP factorizations with R values of 1 through 5 to show the effects of both underfactoring and overfactoring.

Figure 4 plots the emission-mode factors—i.e., the columns of a CP factor matrix corresponding to the mode containing emission wavelengths—scaled by the norm of the corresponding component. Ideally, these are the *emission spectra* of the chemical analytes in the sample. We can see that both CPALS and CPOPT extract the same emission factors for $R = 1, 2, 3$. For $R = 1$, we capture a sort of average of the first three factors. For $R = 2$, one factor is resolved, but the remaining two are still somewhat merged. The $R = 3$ case corresponds to the correct emission spectra of underlying chemical analytes. In the case of overfactoring, for $R = 4, 5$, the factors for CPALS change, especially the smallest factor. For CPOPT, however, the first three factors for $R = 4, 5$ are the same as the ones for $R = 3$ and the extra factors are close to zero.

The norms of the components and timings of the methods for $R = 3, 4, 5$ are reported in Table III. The extra components computed by CPOPT are small in magnitude. Moreover, CPOPT is actually faster than CPALS in both cases of overfactoring.

We have also tested CPOPT on data sets from Reference [57] that are considered to be more difficult. Our experiments show that CPOPT is also robust to overfactoring on these problems.

6.3.3. *Collinearity*

The performance of all four methods degrades as the collinearity of the factors in each mode, see (20), is increased. With higher collinearity, the problem is more difficult because there is more overlap in the factors. Table IV shows results with $C = 0.9$, which can be contrasted to those in Table II with $C = 0.5$. The higher collinearity problems take more time to solve and are less accurate. Furthermore, unlike the low collinearity case where CPOPTR is slightly faster than CPOPT, regularization does not help in terms of computation time in the high collinearity case. The reason for this is that a small regularization parameter ($\lambda = 0.0001$) is used at the high collinearity level in order to keep the regularization error down and the accuracy levels comparable for CPOPTR compared with the other methods.

In Table IV, we also note that accuracies of all algorithms increase as the data set size increases. The underlying reason is that for the same R value, the ratio of the data entries, i.e., $I \times I \times I$, to the number of degrees of freedom in a rank-R CP model increases as the data set size increases. This ratio can be considered as an indicator of the difficulty of the problem. Since this ratio is higher, the problem is easier for larger data sets, even in the case of higher levels of noise.

Figure 5 shows a breakdown of speed and accuracy results for tensors of size $50 \times 50 \times 50$. In comparison to Figure 3 where $C = 0.5$, the accuracies of the methods are low even if $R = R_{\text{true}}$. A comparable result is presented in Reference [28]. Accuracy actually improves for CPNLS, CPOPT and CPOPTR in the case of overfactoring (red), although the times are all more expensive. The reason for the increase in accuracy is that the extra factor captures the effects of the noise, which we discuss in more detail next.

6.3.4. *Noise level*

As noise level increases, the accuracy generally decreases. Figure 6 plots the accuracies for all combinations of homoscedastic and heteroscedastic noise for $R = R_{\text{true}}$ (left column) and $R = R_{\text{true}} + 1$ (right column). For $R = R_{\text{true}}$, all the methods perform similarly. Furthermore, we know that the accuracy is perfect for all methods at $C = 0.5$ when $R = R_{\text{true}}$ (see Figure 3), so the dropoff at higher noise levels is due to the more difficult problems with $C = 0.9$.

For $R = R_{\text{true}} + 1$, the performance of CPALS is worse. Interestingly, the performance improves for $\ell_2 = 1$ for CPALS. We hypothesize that the extra component is then modeling the noise, but for $\ell_2 = 5$ the performance degrades again.

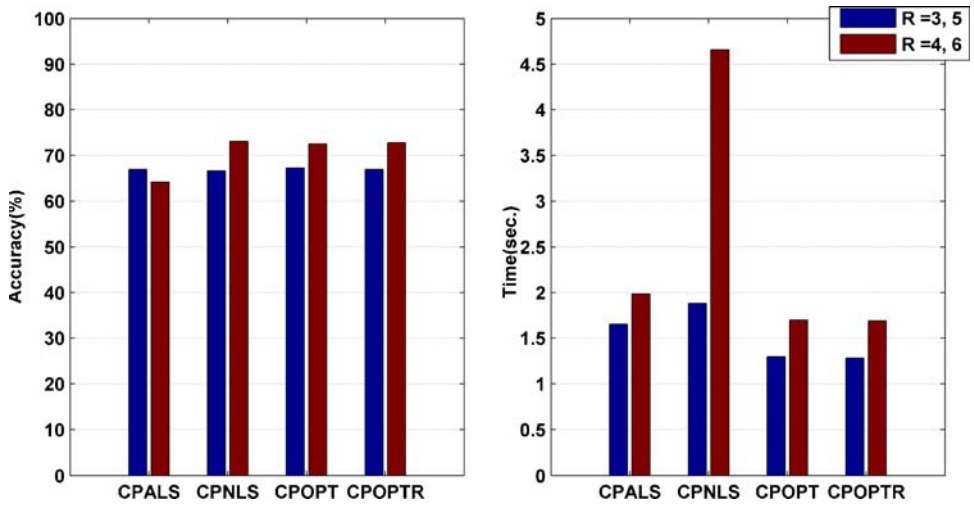

Figure 5. Speed and accuracy comparison for computing the number of true underlying components (blue) and overfactoring (red) for tensors of size $50 \times 50 \times 50$ and collinearity $C = 0.9$.

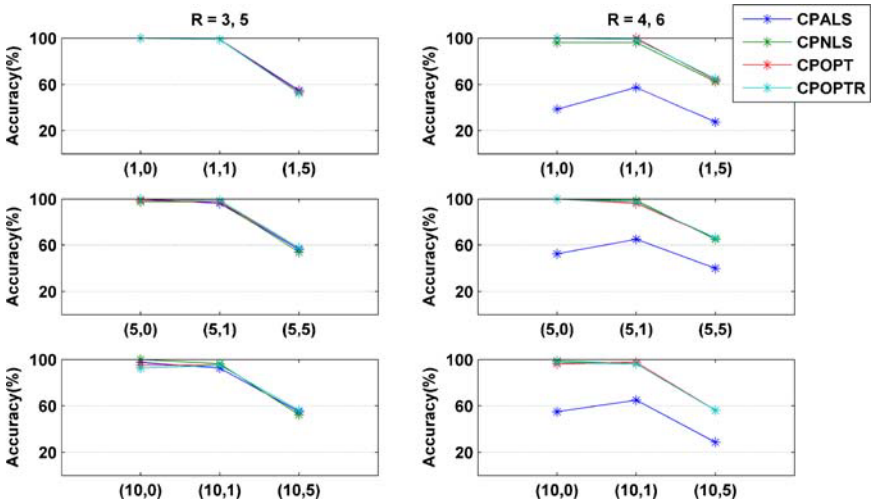

Figure 6. Accuracy of different methods for computing CP for $R = R_{\text{true}}$ (left) and $R = R_{\text{true}} + 1$ (right) at increasing levels of noise for tensors of size $50 \times 50 \times 50$ taking into consideration both collinearity levels, $C = 0.5$ and $C = 0.9$. The numbers on the horizontal axes indicate (ℓ_1, ℓ_2) pairs. Each subplot presents results where the homoscedastic noise ratio (ℓ_1) is held constant while the heteroscedastic noise ratio (ℓ_2) changes.

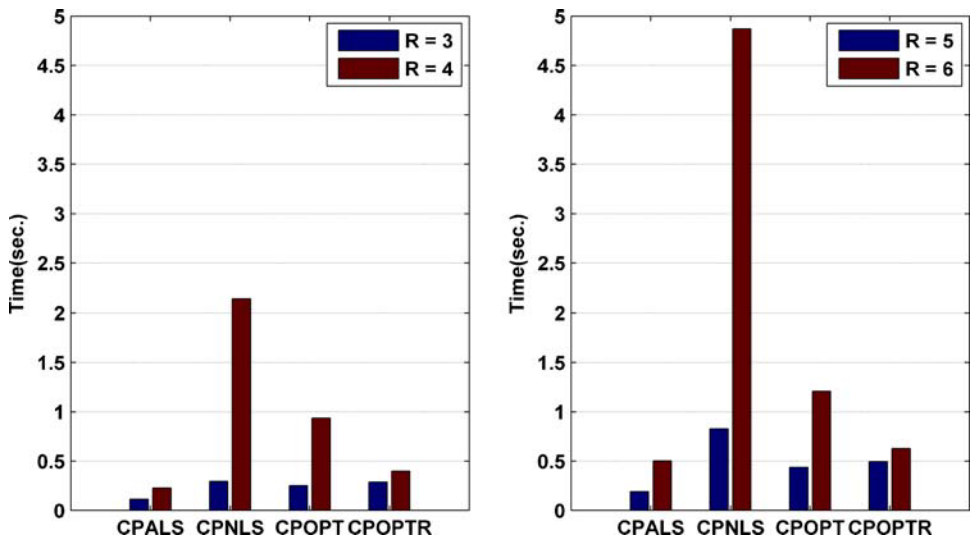

Figure 7. Speed for $R_{\text{true}} = 3$ and $R_{\text{true}} = 5$ for tensors of size $50 \times 50 \times 50$ at $C = 0.5$.

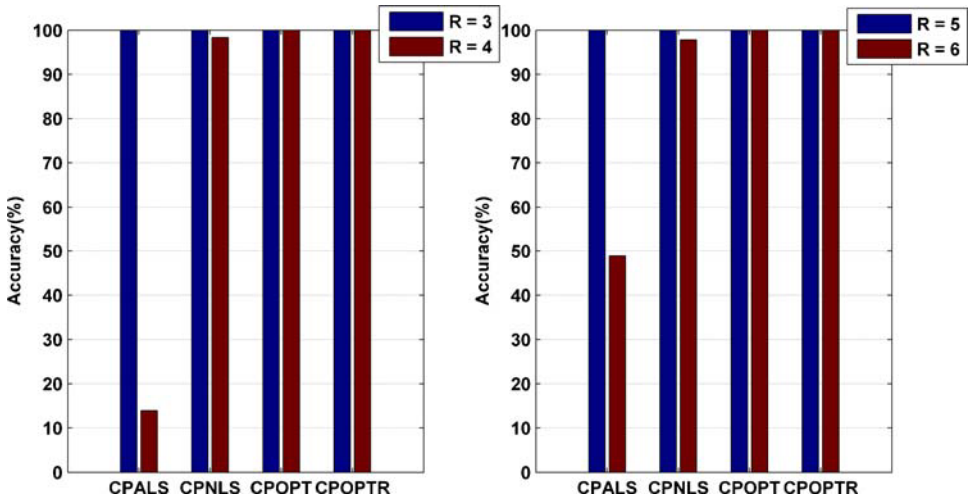

Figure 8. Accuracy for $R_{\text{true}} = 3$ and $R_{\text{true}} = 5$ for tensors of size $50 \times 50 \times 50$ at $C = 0.5$.

6.3.5. Rank

The main effect of the rank of the tensor is in the computation time. Figure 7 splits out the timing results for tensors of size $50 \times 50 \times 50$ with $C = 0.5$. On the left are timings for $R_{\text{true}} = 3$ (blue is $R = R_{\text{true}}$ and red is $R = R_{\text{true}} + 1$), and on the right are the corresponding timings for $R_{\text{true}} = 5$. We see that the computation time of CPNLS increases significantly as the rank increases. This is as expected, however, due to the $O(P^3)$ cost of the method, where P depends linearly on R (see Section 5 for more details).

We also consider the effect of rank on accuracy in Figure 8. In terms of accuracy, all methods except CPALS perform close to 100%, while CPALS suffers from overfactoring. Interestingly, CPALS computes more accurate factorizations in the case of overfactoring for $R_{\text{true}} = 5$ compared with those for $R_{\text{true}} = 3$. Further studies are needed to determine whether this trend continues as the tensor rank increases.

At high collinearity (i.e., $C = 0.9$), as the rank of the original tensor increases from $R_{\text{true}} = 3$ to $R_{\text{true}} = 5$, the accuracy of all methods decreases. Table VIa in Appendix B illustrates this trend for the high collinearity case, which is not present in the low collinearity case (i.e., $C = 0.5$).

7. CONCLUSIONS

Although both ALS [1,2] and NLS [29,31] are optimization-based approaches to solving the CP problem, we revisit the problem and consider another alternative. The OPT approach proposed here is a first-order gradient-based optimization method to solve the CP optimization problem in (2); specifically, we demonstrate the performance of OPT using a nonlinear conjugate gradient method. Compared to ALS, OPT solves for all factor matrices simultaneously and our numerical results show that this leads to increased accuracy in the case of overfactoring. In contrast to NLS, which uses (approximate) second-order information, OPT uses only first-order derivative information; the overall speed of OPT is faster in the experiments presented here due to a reduced cost per iteration.

Our results are consistent with those of Tomasi [31], who considered the scalability of the methods for tensors of order three ($168 \times 168 \times 168$) through seven ($9 \times 9 \times 9 \times \cdots \times 9$). Like we have observed, the results in Reference [31] for order three show that OPT is higher than NLS. As the order grows, NLS becomes slightly faster than OPT, but this is to be expected because the number of variables in the optimization problem is shrinking and the cost of the matrix factorization is consequently less significant.

Key to the good performance of OPT is the efficient tensor formulation of the first derivative of (2). This formulation can serve as a model for deriving analogous formulas for derivatives of other tensor decomposition objective functions. This work makes clear the connection between the gradient and the ALS method: ALS sets the gradient to zero for just one factor matrix at a time, whereas the optimization approach sets the gradient to zero for all factor matrices simultaneously. This connection between ALS and OPT further means that the same methods to making ALS applicable to large-scale problems [47] can be applied to OPT. Future work will consider the scalability of OPT to large-scale sparse problems.

One of the major difficulties of solving the CP problem is addressing the scaling and permutation indeterminacies. The OPTR method includes a Tikhonov regularization term, which addresses the scaling indeterminacy. We have illustrated the connections between OPTR and the implicit regularization in the Levenberg–Marquardt method in NLS. In our numerical experiments, OPTR gave no advantage in accuracy but was faster than OPT in the case of low ($C = 0.5$) collinearity; however, this advantage did not persist in the high ($C = 0.9$) collinearity case. Future work will investigate how to choose the best regularization parameter.

The formulation of the CP problem in (2) can be extended to include non-negativity or sparsity constraints, and we propose that OPT can be extended in a straightforward way to incorporate such constraints. We note, however, that the subject of initialization would need to be addressed when using such constraints, since we currently use the n-mode singular vectors. This is another avenue of future investigation.

Acknowledgements

We thank Dianne O'Leary for helpful conversations on regularization. We also thank Rasmus Bro, Giorgio Tomasi and the anonymous referees for carefully reading earlier versions of the manuscript and providing helpful ideas for improvement.

This work was supported by the Advanced Scientific Computing Research Applied Mathematics Program at the United States Department of Energy and the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

REFERENCES

1. Carroll JD, Chang JJ. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika* 1970; **35**: 283–319.
2. Harshman RA. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA working papers in phonetics* 1970; **16**: 1–84. Available at http://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf.
3. Acar E, Yener B. Unsupervised multiway data analysis: A literature survey. *IEEE Transactions on Knowledge and Data Engineering* 2009; **21**(1): 6–20.
4. Kolda TG, Bader BW. Tensor decompositions and applications. *SIAM Review* 2009; **51**(3): 455–500.
5. Kruskal JB. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and its Applications* 1977; **18**(2): 95–138.
6. Andersen CM, Bro R. Practical aspects of PARAFAC modeling of fluorescence excitation-emission data. *Journal of Chemometrics* 2003; **17**(4): 200–215.
7. Bro R, Kiers HAL. A new efficient method for determining the number of components in PARAFAC models. *Journal of Chemometrics* 2003; **17**(5): 274–286.
8. Acar E, Bingol CA, Bingol H, Bro R, Yener B. Multiway analysis of epilepsy tensors. *Bioinformatics* 2007; **23**(13): i10–i18.
9. Mitchell BC, Burdick DS. Slowly converging PARAFAC sequences: Swamps and two-factor degeneracies. *Journal of Chemometrics* 1994; **8**(2): 155–168.
10. Paatero P. Construction and analysis of degenerate PARAFAC models. *Journal of Chemometrics* 2000; **14**(3): 285–299.
11. Rayens WS, Mitchell BC. Two-factor degeneracies and a stabilization of PARAFAC. *Chemometrics and Intelligent Laboratory Systems* 1997; **38**(2): 173.
12. Stegeman A, De Lathauwer L. A method to avoid diverging components in the Candecomp/Parafac model for generic $I \times J \times 2$ arrays. *SIAM Journal on Matrix Analysis and Applications* 2009; **30**(4): 1614–1638.
13. Faber NKM, Bro R, Hopke PK. Recent developments in CANDECOMP/PARAFAC algorithms: A critical review. *Chemometrics and Intelligent Laboratory Systems* 2003; **65**(1): 119–137.
14. Faber K. Short communication: On solving generalized eigenvalue problems using matlab. *Journal of Chemometrics* 1997; **11**(1): 87–91. Cited by Reference [13].
15. Faber NM, Buydens LMC, Kateman G. Generalized rank annihilation method. I: Derivation of eigenvalue problems. *Journal of Chemometrics* 1994; **8**(2): 147–154. Cited by Reference [13].
16. Faber NM, Buydens LMC, Kateman G. Generalized rank annihilation method. III: Practical implementation. *Journal of Chemometrics* 1994; **8**(4): 273–285. Cited by Reference [13].
17. Gerritsen M, Tanis H, Vandeginste B, Kateman G. Generalized rank annihilation factor analysis, iterative target transformation factor analysis, and residual bilinearization for the quantitative analysis of data from liquid chromatography with photodiode array detection. *Analytical Chemistry* 1992; **64**(18): 2029–2035. Cited by Reference [13].
18. Li S, Hamilton J, Gemperline P. Generalized rank annihilation method using similarity transformations. *Analytical Chemistry* 1992; **64**(6): 599–607. Cited by Reference [13].
19. Lorber A. Features of quantifying chemical composition from two-dimensional data array by the rank annihilation factor analysis method. *Analytical Chemistry* 1985; **57**(12): 2395–2397. Cited by Reference [13].
20. Sanchez E, Kowalski B. Generalized rank annihilation factor analysis. *Analytical Chemistry* 1986; **58**(2): 496–499. Cited by Reference [13].
21. Wu HL, Shibukawa M, Oguma K. An alternating trilinear decomposition algorithm with application to calibration of HPLC–DAD for simultaneous determination of overlapped chlorinated aromatic hydrocarbons. *Journal of Chemometrics* 1998; **12**(1): 1–26.
22. Chen ZP, Wu HL, Jiang JH, Li Y, Yu RQ. A novel trilinear decomposition algorithm for second-order linear calibration. *Chemometrics and Intelligent Laboratory Systems* 2000; **51**(1): 75–86.
23. Chen ZP, Wu HL, Yu RQ. On the self-weighted alternating trilinear decomposition algorithm—the property of being insensitive to excess factors used in calculation. *Journal of Chemometrics* 2001; **15**(5): 439–453.
24. Chen ZP, Li Y, Yu RQ. Pseudo alternating least squares algorithm for trilinear decomposition. *Journal of Chemometrics* 2001; **15**(3): 149–167. Cited by Reference [13].
25. Jiang JH, Wu HL, Li Y, Yu RQ. Alternating coupled vectors resolution (acover) method for trilinear analysis of three-way data. *Journal of Chemometrics* 1999; **13**: 557–578. Cited by Reference [13].
26. Jiang J, Wu H, Li Y, Yu R. Three-way data resolution by alternating slice-wise diagonalization (ASD) method. *Journal of Chemometrics* 2000; **14**(1): 15–36.
27. Li Y, Jiang J, Wu H, Chen Z, Yu R. Alternating coupled matrices resolution method for three-way arrays analysis. *Chemometrics and Intelligent Laboratory Systems* 2000; **52**(1): 33–43. Cited by Reference [13].
28. Tomasi G, Bro R. A comparison of algorithms for fitting the PARAFAC model. *Computational Statistics & Data Analysis* 2006; **50**(7): 1700–1734.
29. Paatero P. A weighted non-negative least squares algorithm for three-way "PARAFAC" factor analysis. *Chemometrics and Intelligent Laboratory Systems* 1997; **38**(2): 223–242.
30. Tomasi G, Bro R. PARAFAC and missing values. *Chemometrics and Intelligent Laboratory Systems* 2005; **75**(2): 163–180.
31. Tomasi G. *Practical and computational aspects in chemometric data analysis*. Ph.D. thesis, Department of Food Science, The Royal Veterinary and Agricultural University, Frederiksberg, Denmark 2006. Available at http://www.models.life.ku.dk/research/theses/.
32. De Lathauwer L, De Moor B, Vandewalle J. Computation of the canonical decomposition by means of a simultaneous generalized Schur decomposition. *SIAM Journal on Matrix Analysis and Applications* 2004; **26**(2): 295–327.
33. De Lathauwer L. A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization. *SIAM Journal on Matrix Analysis and Applications* 2006; **28**(3): 642–666.
34. Vorobyov SA, Rong Y, Sidiropoulos ND, Gershman AB. Robust iterative fitting of multilinear models. *IEEE Transactions on Signal Processing* 2005; **53**(8-1): 2678–2689.
35. Kiers HAL. Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics* 2000; **14**(3): 105–122.
36. Smilde A, Bro R, Geladi P. *Multi-Way Analysis: Applications in the Chemical Sciences*. Wiley: West Sussex, England, 2004.
37. Golub GH, Van Loan CF. *Matrix Computations*. Johns Hopkins Univ. Press 1996.
38. Khatri CG, Rao CR. Solutions to some functional equations and their applications to characterization of probability distributions. *Sankhya: The Indian Journal of Statistics, Series A* 1968; **30**(2): 167–180.
39. Tomasi G, Bro R. Multilinear models: Iterative methods. In *Comprehensive Chemometrics*, vol. 2, Brown S, Tauler R, Walczak R (eds). Elsevier: Oxford, 2009; pp. 411–451.
40. Kolda TG. Multilinear operators for higher-order decompositions. Tech. Rep. SAND2006-2081, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California 2006.
41. Acar E, Kolda TG, Dunlavy DM. An optimization approach for fitting canonical tensor decompositions. Tech. Rep. SAND2009-0857, Sandia National Laboratories, Albuquerque, NM and Livermore, CA 2009.
42. Paatero P. The multilinear engine: A table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *Journal of Computational and Graphical Statistics* 1999; **8**(4): 854–888.
43. Wang JH, Hopke PK. Equation-oriented system: an efficient programming approach to solve multilinear and polynomial equations by the conjugate gradient algorithm. *Chemometrics and Intelligent Laboratory Systems* 2001; **55**(1–2): 13–22.
44. Shashua A, Hazan T. Non-negative tensor factorization with applications to statistics and computer vision. In *ICML 2005: Proceedings of the 22nd International Conference on Machine Learning* 2005; pp. 792–799.

45. Navasca C, De Lathauwer L, Kinderman S. Swamp reducing technique for tensor decompositions. In *EUSIPCO'08: Proceedings of the 16th European Signal Processing Conference* 2008; .
46. Bader BW, Kolda TG. Tensor Toolbox for MATLAB, version 2.2 last accessed November, 2008. http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/.
47. Bader BW, Kolda TG. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing* 2007; **30**(1): 205–231.
48. Bro R. PARAFAC. Tutorial and applications. *Chemometrics and Intelligent Laboratory Systems* 1997; **38**(2): 149–171.
49. Rajih M, Comon P, Harshman RA. Enhanced line search: A novel method to accelerate parafac. *SIAM Journal on Matrix Analysis and Applications* 2008; **30**(3): 1128–1147.
50. Tomasi G. INDAFAC and PARAFAC3W last accessed March, 2009. http://www.models.kvl.dk/source/indafac/index.asp.
51. Madsen K, Nielson HB, Tingleff O. Methods for non-linear least squares problems, 2nd edition. Informatics and Mathematical Modelling, Technical University of Denmark 2004.
52. Dunlavy DM, Kolda TG, Acar E. Poblano v1.0: A Matlab toolbox for gradient-based optimization. Tech. Rep. SAND2010-1422, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California 2010.
53. Moré JJ, Thuente DJ. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software* 1994; **20**(3): 286–307.
54. Nocedal J, Wright SJ. *Numerical Optimization*. Springer 1999.
55. Hager WW, Zhang H. A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimiation* 2006; **2**(1): 35–58.
56. Bro R. Amino acids fluorescence data last accessed December, 2008. http://www.models.kvl.dk/research/data/Amino_Acid_fluo/index.asp.
57. Hopke PK, Paatero P, Jia H, Ross RT, Harshman RA. Three-way (parafac) factor analysis: examination and comparison of alternative computational methods as applied to ill-conditioned data. *Chemometrics and Intelligent Laboratory Systems* 1998; **43**: 25–42.
58. Nash SG, Sofer A. *Linear and Nonlinear Programming*. McGraw–Hill: New York, NY, 1996.

APPENDIX A: DETAILED REGULARIZATION DISCUSSION

We motivate the role that regularization plays in defining a unique solution to (3). Suppose that a minimizer of (3) is given by

$$[\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}.$$

Then a continuous manifold of equivalent minimizers are given by

$$\sum_{r=1}^{R} \beta_{1r} \frac{\mathbf{a}_r^{(1)}}{\|\mathbf{a}_r^{(1)}\|} \circ \beta_{2r} \frac{\mathbf{a}_r^{(2)}}{\|\mathbf{a}_r^{(2)}\|} \circ \cdots \circ \beta_{Nr} \frac{\mathbf{a}_r^{(N)}}{\|\mathbf{a}_r^{(N)}\|}$$

for any set of β-values satisfying

$$\prod_{n=1}^{N} \beta_{nr} = \gamma_r \equiv \prod_{n=1}^{N} \|\mathbf{a}_r^{(n)}\| \quad \text{for} \quad r = 1, \ldots, R.$$

This defines an infinite family of solutions.

Consider the case that the matrices $(\mathbf{A}^{(n)})$ are fixed and only the β values are allowed to change. Then the regularized optimization problem in (16) (ignoring the constant in the objective) reduces to

$$\min \frac{\lambda}{2} \sum_{r=1}^{R} \sum_{n=1}^{N} \beta_{nr}^2 \quad \text{subject to} \quad \prod_{n=1}^{N} \beta_{nr} = \gamma_r$$

for $r = 1, \ldots, R$. This clearly separates into R independent problems so we can drop the r subscript and just consider the problem

$$\min \sum_{n=1}^{N} \beta_n^2 \quad \text{subject to} \quad \prod_{n=1}^{N} \beta_n = \gamma. \tag{24}$$

Theorem A.1. *The vector $\boldsymbol{\beta}^*$ defined by $\beta_n^* = \sqrt[N]{\gamma}$ for $n = 1, \ldots, N$ is a strict local minimizer of* (24).

proof.[‖] The Lagrangian of (24) is defined by

$$\mathcal{L}(\boldsymbol{\beta}, \omega) = \sum_{n=1}^{N} \beta_n^2 - \omega \left(\prod_{n=1}^{N} \beta_n - \gamma \right),$$

where ω is the Lagrangian multiplier for the single constraint. Consequently, the first-order optimality conditions are:

$$2\beta_n - \omega \prod_{\substack{m=1 \\ m \neq n}}^{N} \beta_m = 0 \text{ for } n \in \{1, N\} \text{ and } \prod_{n=1}^{N} \beta_n - \gamma = 0.$$

This is satisfied by $\boldsymbol{\beta}^*$ with $\omega^* = 2\gamma^{\frac{2-N}{N}}$. Thus, $\boldsymbol{\beta}^*$ is a constrained extremum point of (24).

Let $\mathbf{H}(\beta, \omega)$ and $\mathbf{g}(\beta)$ denote, respectively, the Hessian of the Lagrangian and the gradient of the constraint with respect to β. Further, let $\mathbf{H}^* = \mathbf{H}(\boldsymbol{\beta}^*, \omega^*)$ and $\mathbf{g}^* = \mathbf{g}(\boldsymbol{\beta}^*)$. The second-order optimality conditions state that $\boldsymbol{\beta}^*$ is a strict local minimizer of (24) if $\mathbf{z}^\mathsf{T}\mathbf{H}^*\mathbf{z} > 0$ for all $\mathbf{z}$ such that $\mathbf{z}^\mathsf{T}\mathbf{g}^* = 0$. Since the elements of the vector $\mathbf{g}^*$ are all equal (i.e., $g_n^* = \gamma^{\frac{N-1}{N}}$ for all $n = 1, \ldots, N$), the null space with respect to the constraint gradients is defined by

$$\mathcal{N}^* = \left\{ \mathbf{z} \,|\, \sum z_n = 0 \right\}.$$

Thus, if $\mathbf{z}$ lives in the nullspace of the constraints, then it must be the case that

$$\left(\sum_{n=1}^{N} z_n \right)^2 = 0 \Rightarrow \sum_{n=1}^{N} z_n^2 + 2 \sum_{n=1}^{N} \sum_{m=n+1}^{N} z_n z_m = 0$$

$$\Rightarrow 2 \sum_{n=1}^{N} \sum_{m=n+1}^{N} z_n z_m < 0.$$

It is also straightforward to show that

$$h_{ij}^* = \begin{cases} 2 & \text{if } i = j, \\ -2 & \text{if } i \neq j. \end{cases}$$

[‖] The proof follows the proofs and examples on optimality conditions for constrained optimization problems given in Reference [58].

Table V. Detailed results for $20 \times 20 \times 20$ tensors

(a) Accuracy

	Accuracy (%)							
Rank (R_{true})	**3**				**5**			
Collinearity (C)	**0.5**		**0.9**		**0.5**		**0.9**	
Extr. Components (R)	**3**	**4**	**3**	**4**	**5**	**6**	**5**	**6**
CPALS	100.0	47.2	49.4	45.6	100.0	67.8	11.1	10.0
CPNLS	100.0	98.3	51.7	51.1	100.0	97.8	11.1	13.3
CPOPT	100.0	100.0	52.2	52.2	100.0	99.4	11.1	13.3
CPOPTR	100.0	100.0	52.8	53.3	100.0	100.0	11.1	13.3

(b) Computation time for $R_{true} = 3$

	Time (sec)			
Rank (R_{true})	**3**			
Collinearity (C)	**0.5**		**0.9**	
Extr. Components (R)	**3**	**4**	**3**	**4**
CPALS	0.1 ± 0.0	1.0 ± 1.4	0.7 ± 0.3	1.0 ± 0.7
CPNLS	0.1 ± 0.0	0.3 ± 0.2	0.1 ± 0.1	0.4 ± 0.2
CPOPT	0.1 ± 0.0	0.4 ± 0.2	0.4 ± 0.1	0.6 ± 0.2
CPOPTR	0.1 ± 0.0	0.2 ± 0.0	0.4 ± 0.1	0.6 ± 0.2

(c) Computation time for $R_{true} = 5$

	Time (sec)			
Rank (R_{true})	**5**			
Collinearity (C)	**0.5**		**0.9**	
Extr. Components (R)	**5**	**6**	**5**	**6**
CPALS	0.1 ± 0.0	0.9 ± 1.2	1.2 ± 0.7	1.6 ± 1.1
CPNLS	0.1 ± 0.0	0.6 ± 0.3	0.6 ± 0.5	1.0 ± 0.6
CPOPT	0.2 ± 0.0	0.5 ± 0.2	0.8 ± 0.2	1.0 ± 0.4
CPOPTR	0.2 ± 0.0	0.3 ± 0.1	0.8 ± 0.2	1.0 ± 0.4

Thus, for any vector $\mathbf{z} \in \mathcal{N}^*$ we have

$$\mathbf{z}^T\mathbf{H}^*\mathbf{z} = 2\left(\sum_{n=1}^{N} z_n^2 - 2\sum_{n=1}^{N}\sum_{m=n+1}^{N} z_n z_m\right) > 0.$$

Hence, the claim. □

From the previous theorem, we can see that if we fix the solution of (16) *except for scaling*, then the best solution is given when all N vectors for the rth component are scaled to equal size, i.e.,

$$\left\|\mathbf{a}_r^{(1)}\right\|^2 = \left\|\mathbf{a}_r^{(2)}\right\|^2 = \ldots = \left\|\mathbf{a}_r^{(N)}\right\|^2.$$

When that is the case for all R components, the Frobenius norms of the factor matrices are also equal.

APPENDIX B: DETAILED NUMERICAL RESULTS

The experimental set-up described in Section 6.1 results in 23,040 individual experiments. In this appendix, we provide summaries across a range of parameters in Table V-VIII, but we cannot provide all possible break-downs due to space limitations. Therefore, we provide full results (timing and accuracy) of the individual experiments as a downloadable MATLAB MAT-file (`data.mat`) along with a MATLAB M-file (`data_exploration.m`)[¶] showing examples of how to process the data to produce some of the figures and tables in the paper. In the results presented here, each cell corresponds to twenty sets of factor matrices and nine levels of noise (as described in Section 6.1) for a total of 180 test tensors.

[¶] These files are available on http://csmr.ca.sandia.gov/~tgkolda/cpopt2010/.

Table VI. Detailed results for $50 \times 50 \times 50$ tensors

(a) Accuracy

	Accuracy (%)							
Rank (R_{true})	**3**				**5**			
Collinearity (C)	**0.5**		**0.9**		**0.5**		**0.9**	
Extr. Components (R)	**3**	**4**	**3**	**4**	**5**	**6**	**5**	**6**
CPALS	100.0	13.9	73.9	67.8	100.0	48.9	60.0	60.6
CPNLS	100.0	98.3	68.3	80.6	100.0	97.8	65.0	65.6
CPOPT	100.0	100.0	74.4	82.8	100.0	100.0	60.0	62.2
CPOPTR	100.0	100.0	73.9	83.3	100.0	100.0	60.0	62.2

(b) Computation time for $R_{true} = 3$

	Time (sec)			
Rank (R_{true})	**3**			
Collinearity (C)	**0.5**		**0.9**	
Extr. Components (R)	**3**	**4**	**3**	**4**
CPALS	0.1 ± 0.0	0.2 ± 0.3	1.2 ± 0.5	1.7 ± 0.7
CPNLS	0.3 ± 0.1	2.1 ± 1.2	0.8 ± 0.6	2.4 ± 1.3
CPOPT	0.3 ± 0.1	0.9 ± 0.4	0.9 ± 0.3	1.3 ± 0.4
CPOPTR	0.3 ± 0.1	0.4 ± 0.1	0.9 ± 0.3	1.3 ± 0.3

(c) Computation time for $R_{true} = 5$

	Time (sec)			
Rank (R_{true})	**5**			
Collinearity (C)	**0.5**		**0.9**	
Extr. Components (R)	**5**	**6**	**5**	**6**
CPALS	0.2 ± 0.0	0.5 ± 0.5	2.1 ± 1.0	2.3 ± 0.8
CPNLS	0.8 ± 0.3	4.9 ± 3.6	2.9 ± 2.5	7.0 ± 4.0
CPOPT	0.4 ± 0.1	1.2 ± 0.6	1.7 ± 0.5	2.1 ± 0.7
CPOPTR	0.5 ± 0.1	0.6 ± 0.1	1.7 ± 0.5	2.1 ± 0.6

Table VII. Detailed results for 100 × 100 × 100 tensors

(a) Accuracy

	Accuracy (%)							
Rank (R_{true})	**3**				**5**			
Collinearity (C)	**0.5**		**0.9**		**0.5**		**0.9**	
Extr. Components (R)	**3**	**4**	**3**	**4**	**5**	**6**	**5**	**6**
CPALS	100.0	11.1	91.7	72.8	100.0	42.8	66.7	61.1
CPNLS	100.0	96.7	81.1	94.4	100.0	95.0	67.2	67.2
CPOPT	100.0	100.0	89.4	96.7	100.0	100.0	64.4	67.2
CPOPTR	100.0	100.0	89.4	96.7	100.0	100.0	65.6	67.2

(b) Computation time for $R_{true} = 3$

	Time (sec)			
Rank (R_{true})	**3**			
Collinearity (C)	**0.5**		**0.9**	
Extr. Components (R)	**3**	**4**	**3**	**4**
CPALS	1.2 ± 0.1	1.2 ± 0.6	13.6 ± 3.2	11.2 ± 3.5
CPNLS	2.3 ± 0.8	11.5 ± 5.1	4.8 ± 2.3	12.5 ± 6.3
CPOPT	3.2 ± 0.8	5.9 ± 3.3	10.5 ± 2.8	8.8 ± 2.3
CPOPTR	3.6 ± 0.8	3.2 ± 0.6	10.5 ± 2.6	8.9 ± 2.4

(c) Computation time for $R_{true} = 5$

	Time (sec)			
Rank (R_{true})	**5**			
Collinearity (C)	**0.5**		**0.9**	
Extr. Components (R)	**5**	**6**	**5**	**6**
CPALS	1.5 ± 0.1	2.8 ± 1.6	15.5 ± 6.2	16.6 ± 6.7
CPNLS	5.6 ± 1.8	26.7 ± 12.2	19.7 ± 18.1	39.8 ± 25.0
CPOPT	4.0 ± 0.7	9.3 ± 4.3	14.0 ± 3.3	18.5 ± 4.6
CPOPTR	4.5 ± 0.7	5.9 ± 0.8	14.2 ± 3.3	18.2 ± 4.4

Table VIII. Detailed results for 250 × 250 × 250 tensors

(a) Accuracy

	Accuracy (%)							
Rank (R_{true})	**3**				**5**			
Collinearity (C)	**0.5**		**0.9**		**0.5**		**0.9**	
Extr. Components (R)	**3**	**4**	**3**	**4**	**5**	**6**	**5**	**6**
CPALS	100.0	7.2	100.0	58.9	100.0	41.7	81.1	79.4
CPNLS	100.0	99.4	80.0	91.7	100.0	96.7	83.3	93.3
CPOPT	100.0	100.0	100.0	100.0	100.0	100.0	74.4	83.9
CPOPTR	100.0	100.0	100.0	100.0	100.0	100.0	73.9	83.9

(b) Computation time for $R_{true} = 3$

	Time (sec)			
Rank (R_{true})	**3**			
Collinearity (C)	**0.5**		**0.9**	
Extr. Components (R)	**3**	**4**	**3**	**4**
CPALS	22.2 ± 1.5	19.8 ± 4.5	264.9 ± 54.9	167.5 ± 51.8
CPNLS	42.1 ± 13.8	135.4 ± 65.0	65.2 ± 21.8	140.4 ± 54.7
CPOPT	62.4 ± 13.1	70.2 ± 28.3	200.3 ± 46.2	154.2 ± 33.0
CPOPTR	70.5 ± 13.9	60.7 ± 9.5	197.1 ± 44.6	155.6 ± 33.3

(c) Computation time for $R_{true} = 5$

	Time (sec)			
Rank (R_{true})	**5**			
Collinearity (C)	**0.5**		**0.9**	
Extr. Components (R)	**5**	**6**	**5**	**6**
CPALS	26.5 ± 1.5	37.7 ± 11.0	294.4 ± 98.0	241.7 ± 126.3
CPNLS	97.0 ± 27.8	301.1 ± 141.5	199.6 ± 105.0	389.6 ± 214.6
CPOPT	77.0 ± 13.7	124.4 ± 38.2	255.6 ± 69.7	315.7 ± 68.8
CPOPTR	85.9 ± 13.9	110.2 ± 14.7	250.7 ± 68.9	313.9 ± 73.7

