SANDIA REPORT

SAND2009-0857
Unlimited Release
Printed February 2009

An Optimization Approach for Fitting Canonical Tensor Decompositions

Evrim Acar, Tamara G. Kolda, and Daniel M. Dunlavy

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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-AC04-94-AL85000.

Sandia National Laboratories

SAND2009-0857
Unlimited Release
Printed February 2009

An Optimization Approach for Fitting Canonical Tensor Decompositions

Evrim Acar and Tamara G. Kolda
Computational Science and Mathematics Science Research Department
Sandia National Laboratories
Livermore, CA 94551-9159
Email: `eacarat,tgkolda@sandia.gov`

Daniel M. Dunlavy
Computer Science & Informatics Department
Sandia National Laboratories, Albuquerque, NM 87123-1318
Email: `dmdunla@sandia.gov`

Abstract

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 the CANDECOMP/PARAFAC decomposition (CPD), 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 the CPD, however, can be difficult. The typical approach is based on alternating least squares (ALS) optimization, which can be remarkably fast but is not very accurate. Previously, nonlinear least squares (NLS) methods have also been recommended; existing NLS methods are accurate but slow. In this paper, we propose the use of gradient-based optimization methods. We discuss the mathematical calculation of the derivatives and further 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 much more accurate than ALS and orders of magnitude faster than NLS.

Acknowledgments

This work was fully supported by Sandia's Laboratory Directed Research & Development (LDRD) program. We thank Dianne O'Leary for helpful conversations on regularization and Rasmus Bro for providing insight on the amino acids dataset.

Contents

Appendix

Figures

Tables

This page intentionally left blank.

1 Introduction

A tensor is a multidimensional or N-way array. The canonical tensor decomposition [8, 20] 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 the surveys [1, 26]. We refer to the canonical decomposition as the CANDECOMP/PARAFAC decomposition (CPD) to recognize the original names given to it by Carroll and Chang [8] and Harshman [20], respectively.

The CPD is an analogue of the matrix SVD because it decomposes a tensor into the sum of component rank-one tensors. To understand the CPD, suppose that $\mathcal{Z}$ is a 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 CPD 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 the CPD are defined by $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_R \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_R \end{bmatrix}$, and $\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_R \end{bmatrix}$, 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 CPD 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 [26, §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 the CPD can be formulated in this way; we assume for convenience that any multiplier is absorbed into the factors. Despite CPD's lack of orthogonality or even linear independence of factors in each mode, Kruskal [27, 28] has shown that it does have the advantage of uniqueness, up to permutation and scaling, under mild conditions; see [26, §3.2] for details. In contrast, it is well known that uniqueness of the SVD is due to its orthogonality constraints and even then is not unique when multiple singular values are equal. It is perhaps because of CPD'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 [2] commonly used in chemistry.

In terms of computing the CPD, the first question, not directly addressed here, is the choice of R, the number of component rank-one tensors. We generally do not know the tensor rank and its computation is NP-complete [28, 21]. Therefore, in practice, the user tries different values of R and picks the best value based on some criteria, e.g., the model fit and the core consistency (CORCONDIA) diagnostic as proposed by Bro and Kiers [7]. Another issue, not addressed in this paper, is that computing the CPD can be difficult because some tensors may have approximations of a lower rank that are arbitrarily close in terms of fit; this is called *degeneracy* and can cause problems in practice [33, 41, 40, 45].

In this paper we focus on the question of how to compute the factor matrices of the CPD for a given value of R (not necessarily equal to the rank of the tensor). The typical method for finding the components of the CPD is alternating least squares (ALS) optimization, as proposed in the original CP papers [8, 20]. The essential idea in ALS is to start with an initial guess for the factor matrices, solve 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, it often fails to obtain the correct solution, especially in the case of overfactoring, i.e., when R is greater than the rank of the tensor. Another promising alternative which we discuss is a nonlinear least squares (NLS) formulation; the NLS approach is far superior to ALS in terms of finding the correct solution, but significantly slower. Here, we propose using a more 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 has the same accuracy as NLS but is orders of magnitude faster, nearly as fast as ALS.

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 [15] compared ALS with a number of competing algorithms: direct trilinear decomposition (DTLD) [16, 17, 14, 18, 29, 31, 42], alternating trilinear decomposition (ATLD) [50], self-weighted alternating trilinear decomposition (SWATLD) [10, 11], pseudo-alternating least squares (PALS) [9], alternating coupled vectors resolution (ACOVER) [23], alternating slice-wise diagonalization (ASD) [22], and alternating coupled matrices resolution (ACOMAR) [30]. 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 [49] have compared ALS with some of the algorithms mentioned above (DTLD, ASD, SWATLD) as well as two NLS approaches, PMF3 [38] and INDAFAC [47, 48]. The performance results show that ASD, the best alternative in the previous study [15], 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 [49] as the basis for the numerical results in this paper. We once again compare ALS and NLS (specifically, INDAFAC) along with our OPT methods.

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 [13] cast CPD as a simultaneous generalized Schur decomposition (SGSD) and this approach has been applied to overcoming the problem of degeneracy [45]. De Lathauwer [12] also developed a method based on simultaneous matrix diagonalization, which is somewhat related to the DTLD approach mentioned above.

The main contributions of this paper are summarized as follows:

- Presenting a new formulation of CPD as a general optimization problem. This formulation helps to address permutation and scaling indeterminacies directly via regularization.

- Obtaining derivatives for a general optimization formulation of CPD and showing that the derivatives can be computed efficiently. The analysis can be used to obtain analogous formulations for other tensor decompositions.

- Demonstrating and comparing the performance of several methods for computing the CPD on both real and synthetic data. These studies indicate the benefits of using the optimization methods presented in this paper over 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 §3. The OPT method is presented in §4, with a focus on the formulation of the gradient and Hessian, as well as discussion of practical issues such as regularization. We contrast the NLS approach in §5. We follow the experimental procedure of Tomasi and Bro [49] to compare the methods; the details of the methods that are employed and the results are presented in §6. Conclusions are discussed in §7. Detailed numerical results are available in Appendix A.

2 Notation

We generally use the notation of [26], which was adapted from [24]. 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., $\mathbf{\mathcal{X}}$. 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 $\mathbf{\mathcal{X}}$ 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 $\mathbf{\mathcal{X}}, \mathbf{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the sum of the products of their entries, i.e.,

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

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

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

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

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

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

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

The *Khatri-Rao product* [44] 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

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_K \otimes \mathbf{b}_K \end{bmatrix}.$$

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 [44]:

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

where $*$ denotes the elementwise product. Furthermore, the pseudo-inverse of the Khatri-Rao product has special form [44]:

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

where $\mathbf{A}^{\dagger}$ denotes the Moore-Penrose pseudo-inverse of $\mathbf{A}$ [19]. Recall that the pseudo-inverse of the transpose is the transpose of the pseudo-inverse, so

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

Further, because the Khatri-Rao product is associative, this property can be extended to more than two matrices; see [44] 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

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

Since matricization is just a rearrangement of the elements, clearly $\|\mathcal{X}\| = \|\mathbf{X}_{(n)}\|$ for $n = 1, \ldots, N$; see [26, §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.,

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

Multiplication in every mode except mode n results in a vector of length I_n, i.e., the i_nth element of that vector is

$$\begin{aligned} \left(\mathcal{X} \bigtimes_{\substack{m=1 \\ m \neq n}}^{N} \mathbf{v} \right)_{i_n} &= \sum_{i_1=1}^{I_1} \cdots \sum_{i_{n-1}=1}^{I_{n-1}} \sum_{i_{n+1}=1}^{I_{n+1}} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N} \, v_{i_1}^{(1)} \cdots v_{i_{n-1}}^{(n-1)} v_{i_{n+1}}^{(n+1)} \cdots v_{i_N}^{(N)} \\ &= \mathbf{X}_{(n)} \left(\mathbf{v}^{(N)} \otimes \cdots \otimes \mathbf{v}^{(n+1)} \otimes \mathbf{v}^{(n-1)} \otimes \cdots \otimes \mathbf{v}^{(1)} \right). \end{aligned}$$

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

3 CPD and alternating least squares

In the introduction, we presented the CPD of three-way tensors. Here, we present the CPD for a general N-way tensor. Let $\mathbfcal{Z}$ be an 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 CPD [8, 20] such that

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

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 [25]:

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

where the *factor matrices* are defined as

$$\mathbf{A}^{(n)} = \begin{bmatrix} \mathbf{a}_1^{(n)} & \cdots & \mathbf{a}_R^{(n)} \end{bmatrix},$$

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, [25]) as

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

The problem of computing the CPD is, given a tensor $\mathbfcal{Z}$ and a choice for R (not necessarily the rank of $\mathbfcal{Z}$), 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 CPD as a least squares optimization problem:

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

The ALS method for CPD was proposed in the original papers by Carroll and Chang [8] and Harshman [20], and still remains the primary workhorse algorithm today due to its speed and ease of implementation [49]. 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

$$\min_{\mathbf{A}^{(n)}} f(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \equiv \frac{1}{2} \left\| \mathbfcal{Z} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] \right\|^2, \tag{3}$$

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

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

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

$$\mathbf{A}^{(n)} = \mathbf{Z}_{(n)} \left(\left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}\right)^{\mathsf{T}}\right)^{\dagger}.$$

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

$$\mathbf{A}^{(n)} = \mathbf{Z}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right) (\mathbf{\Gamma}^{(n)})^{\dagger}, \tag{4}$$

where

$$\mathbf{\Gamma}^{(n)} = (\mathbf{A}^{(1)\mathsf{T}}\mathbf{A}^{(1)}) * \cdots * (\mathbf{A}^{(n-1)\mathsf{T}}\mathbf{A}^{(n-1)}) * (\mathbf{A}^{(n+1)\mathsf{T}}\mathbf{A}^{(n+1)}) * \cdots * (\mathbf{A}^{(N)\mathsf{T}}\mathbf{A}^{(N)}) \quad \text{for } n = 1, \ldots, N. \tag{5}$$

Note that this formulation only requires computing the pseudo-inverse of a matrix of size $R \times R$. The ALS procedure for CPD is well known; see, e.g., [44].

4 Optimization for CPD

As an alternative to the ALS approach for CPD, 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 a NLS method; this is discussed in §5.) We can consider the CPD 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

$$P = R \sum_{n=1}^{N} I_n \tag{6}$$

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.,

$$\mathbf{x} = \begin{bmatrix} \mathbf{a}_1^{(1)} \\ \vdots \\ \mathbf{a}_R^{(1)} \\ \vdots \\ \mathbf{a}_1^{(N)} \\ \vdots \\ \mathbf{a}_R^{(N)} \end{bmatrix}. \tag{7}$$

In this view, $f : \mathbb{R}^P \mapsto \mathbb{R}$, and it is straightforward to derive its gradient and Hessian, which we do in §4.1 and §4.2, respectively. We discuss the effect of the scaling and permutation indeterminacies of the CPD in §4.3, explaining the benefits of regularization and deriving the derivatives of the regularized objective. Once the derivatives are known, any optimization method can be used. In §6, we show results using a nonlinear conjugate gradient method [36].

4.1 CPD 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 $n = 1, \ldots, N$. In other words,

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{a}_1^{(1)}} \\ \vdots \\ \frac{\partial f}{\partial \mathbf{a}_R^{(1)}} \\ \vdots \\ \frac{\partial f}{\partial \mathbf{a}_1^{(N)}} \\ \vdots \\ \frac{\partial f}{\partial \mathbf{a}_R^{(N)}} \end{bmatrix}.$$

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 [43] in the context of non-negative tensor factorizations.

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

$$\frac{\partial f}{\partial \mathbf{a}_r^{(n)}} = -\left(\boldsymbol{\mathscr{Z}} \bigtimes_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)} \right) + \sum_{\ell=1}^{R} \gamma_{r\ell}^{(n)} \mathbf{a}_\ell^{(n)}, \tag{8}$$

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

$$\gamma_{r\ell}^{(n)} \equiv \prod_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)\mathsf{T}} \mathbf{a}_\ell^{(m)}. \tag{9}$$

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

$$f(\mathbf{x}) = \frac{1}{2} \underbrace{\| \boldsymbol{\mathscr{Z}} \|^2}_{f_1(\mathbf{x})} - \underbrace{\langle \boldsymbol{\mathscr{Z}}, [\![\mathbf{A}^{(1)},\dots,\mathbf{A}^{(N)}]\!] \rangle}_{f_2(\mathbf{x})} + \frac{1}{2} \underbrace{\left\| [\![\mathbf{A}^{(1)},\dots,\mathbf{A}^{(N)}]\!] \right\|^2}_{f_3(\mathbf{x})}. \tag{10}$$

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 $\boldsymbol{\mathscr{Z}}$ and its CPD approximation, given by

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

Writing f_2 this way makes it obvious that

$$\frac{\partial f_2}{\partial \mathbf{a}_r^{(n)}} = \left(\boldsymbol{\mathscr{Z}} \bigtimes_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)} \right). \tag{11}$$

The third summand is

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

Therefore,

$$\begin{aligned}\frac{\partial f_3}{\partial \mathbf{a}_r^{(n)}} &= 2\left(\prod_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)\mathsf{T}}\mathbf{a}_r^{(m)}\right)\mathbf{a}_r^{(n)} + 2\sum_{\substack{\ell=1\\\ell\neq r}}^{R}\left(\prod_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)\mathsf{T}}\mathbf{a}_\ell^{(m)}\right)\mathbf{a}_\ell^{(n)}\\ &= 2\sum_{\ell=1}^{R}\left(\prod_{\substack{m=1\\m\neq n}}^{N} \mathbf{a}_r^{(m)\mathsf{T}}\mathbf{a}_\ell^{(m)}\right)\mathbf{a}_\ell^{(n)}. \end{aligned} \tag{12}$$

Combining (11) and (12) yields the desired result. □

Observe that $\gamma_{r\ell}^{(n)}$ is indeed the (r,ℓ) entry of the matrix $\mathbf{\Gamma}^{(n)}$ defined in (5). These values can be computed as follows. Compute the following $R \times R$ matrices (one per mode):

$$\mathbf{\Upsilon}^{(n)} = \mathbf{A}^{(n)\mathsf{T}}\mathbf{A}^{(n)} \quad \text{for } n = 1,\dots,N. \tag{13}$$

Then it is clear that

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

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*

$$\frac{\partial f}{\partial \mathbf{A}^{(n)}} = -\mathbf{Z}_{(n)}\left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}\right) + \mathbf{A}^{(n)}\mathbf{\Gamma}^{(n)}, \tag{14}$$

for $n = 1,\dots,N$, where $\mathbf{\Gamma}^{(n)}$ is defined in (5).

Proof. Equation (8) in Theorem 4.1 can be rewritten as

$$\frac{\partial f}{\partial \mathbf{a}_r^{(n)}} = -\mathbf{Z}_{(n)}\left(\mathbf{a}_r^{(N)} \otimes \cdots \otimes \mathbf{a}_r^{(n+1)} \otimes \mathbf{a}_r^{(n-1)} \otimes \cdots \otimes \mathbf{a}_r^{(1)}\right) + \mathbf{A}^{(n)}\boldsymbol{\gamma}_r^{(n)},$$

for $r = 1,\dots,R$. Note that this expression exploits the fact that $\mathbf{\Gamma}^{(n)}$ is symmetric. Associating each $r = 1,\dots,R$ with the column of a matrix yields (14). □

It is interesting to observe here a connection to ALS. Setting the gradient of f with respect to $\mathbf{A}^{(n)}$ equal to zero yields

$$\mathbf{A}^{(n)}\mathbf{\Gamma}^{(n)} = \mathbf{Z}_{(n)}\left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}\right).$$

The ALS equation for updating $\mathbf{A}^{(n)}$ given in (4) then follows immediately.

4.2 CPD Hessian

We will not use the Hessian in our computations, but we present it here for completeness. We can assemble the Hessian of size $P \times P$ by calculating the second partial derivative with respect to each $\mathbf{a}_r^{(n)}$ for $r = 1,\dots,R$

and $n = 1, \dots, N$. In other words,

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial \mathbf{a}_1^{(1)} \partial \mathbf{a}_1^{(1)}} & \cdots & \frac{\partial^2 f}{\partial \mathbf{a}_1^{(1)} \partial \mathbf{a}_R^{(1)}} & \cdots\cdots & \frac{\partial^2 f}{\partial \mathbf{a}_1^{(1)} \partial \mathbf{a}_1^{(N)}} & \cdots & \frac{\partial^2 f}{\partial \mathbf{a}_1^{(1)} \partial \mathbf{a}_R^{(N)}} \\ \vdots & \ddots & \vdots & \cdots\cdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial \mathbf{a}_R^{(1)} \partial \mathbf{a}_1^{(1)}} & \cdots & \frac{\partial^2 f}{\partial \mathbf{a}_R^{(1)} \partial \mathbf{a}_R^{(1)}} & \cdots\cdots & \frac{\partial^2 f}{\partial \mathbf{a}_R^{(1)} \partial \mathbf{a}_1^{(N)}} & \cdots & \frac{\partial^2 f}{\partial \mathbf{a}_R^{(1)} \partial \mathbf{a}_R^{(N)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial \mathbf{a}_1^{(N)} \partial \mathbf{a}_1^{(1)}} & \cdots & \frac{\partial^2 f}{\partial \mathbf{a}_1^{(N)} \partial \mathbf{a}_R^{(1)}} & \cdots\cdots & \frac{\partial^2 f}{\partial \mathbf{a}_1^{(N)} \partial \mathbf{a}_1^{(N)}} & \cdots & \frac{\partial^2 f}{\partial \mathbf{a}_1^{(N)} \partial \mathbf{a}_R^{(N)}} \\ \vdots & \ddots & \vdots & \cdots\cdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial \mathbf{a}_R^{(N)} \partial \mathbf{a}_1^{(1)}} & \cdots & \frac{\partial^2 f}{\partial \mathbf{a}_R^{(N)} \partial \mathbf{a}_R^{(1)}} & \cdots\cdots & \frac{\partial^2 f}{\partial \mathbf{a}_R^{(N)} \partial \mathbf{a}_1^{(N)}} & \cdots & \frac{\partial^2 f}{\partial \mathbf{a}_R^{(N)} \partial \mathbf{a}_R^{(N)}} \end{bmatrix}.$$

Note that the second partial derivative $\frac{\partial^2 f}{\partial \mathbf{a}_r^{(n)} \partial \mathbf{a}_s^{(p)}}$ is a matrix of size $I_n \times I_p$. Theorem 4.3 specifies the second partial derivatives.

Theorem 4.3 *The second partial derivative of (2) is given by*

$$\frac{\partial^2 f}{\partial \mathbf{a}_r^{(n)} \partial \mathbf{a}_s^{(n)}} = \gamma_{rs}^{(n)} \, \mathbf{I} \tag{15}$$

where $\mathbf{I}$ *is the* $I_n \times I_n$ *identity matrix, for* $n = 1, \dots, N$ *and* $r, s = 1, \dots, R$*;*

$$\frac{\partial^2 f}{\partial \mathbf{a}_r^{(n)} \partial \mathbf{a}_r^{(p)}} = -\left(\mathfrak{Z} \bigtimes_{\substack{m=1 \\ m \neq n,p}}^{N} \mathbf{a}_r^{(m)} \right) + 2\psi_{rr}^{(np)} \mathbf{a}_r^{(n)} \circ \mathbf{a}_r^{(p)} + \sum_{\substack{\ell=1 \\ \ell \neq r}}^{R} \psi_{r\ell}^{(np)} \mathbf{a}_\ell^{(n)} \circ \mathbf{a}_\ell^{(p)} \tag{16}$$

for $n \neq p$, $n, p = 1, \dots, N$ *and* $r = 1, \dots, R$*; and*

$$\frac{\partial^2 f}{\partial \mathbf{a}_r^{(n)} \partial \mathbf{a}_s^{(p)}} = \psi_{rs}^{(np)} \mathbf{a}_s^{(n)} \circ \mathbf{a}_r^{(p)}. \tag{17}$$

for $n \neq p$, $n, p = 1, \dots, N$ *and* $r \neq s$, $r, s = 1, \dots, R$*. Here, we define*

$$\psi_{k\ell}^{(np)} \equiv \prod_{\substack{m=1 \\ m \neq n,p}}^{N} \mathbf{a}_k^{(m)\mathsf{T}} \mathbf{a}_\ell^{(n)}, \tag{18}$$

for all $n, p = 1, \dots, N$ *and* $k, \ell = 1, \dots, R$*.*

Proof. First consider the case of taking derivative of the first derivative with respect to $\mathbf{a}_s^{(n)}$. We can rewrite the first derivative in (8) as

$$\frac{\partial f}{\partial \mathbf{a}_r^{(n)}} = -\left(\mathfrak{Z} \bigtimes_{\substack{m=1 \\ m \neq n}}^{N} \mathbf{a}_r^{(m)} \right) + \gamma_{rs}^{(n)} \, \mathbf{a}_s^{(n)} + \sum_{\substack{\ell=1 \\ \ell \neq s}}^{R} \gamma_{r\ell}^{(n)} \mathbf{a}_\ell^{(n)}.$$

Clearly, $\mathbf{a}_s^{(n)}$ only appears in the second term, so (15) follows immediately.

Next, consider the case of $\mathbf{a}_r^{(p)}$ for $p \neq n$. Observe from the definition of $\gamma_{k\ell}^{(n)}$ from Theorem 4.1 and $\psi_{k\ell}^{(np)}$ from here, we have

$$\gamma_{k\ell}^{(n)} = \psi_{k\ell}^{(np)} (\mathbf{a}_k^{(p)\mathsf{T}} \mathbf{a}_\ell^{(p)}).$$

Therefore, we can rewrite the first partial derivative in (8) as

$$\frac{\partial f}{\partial \mathbf{a}_r^{(n)}} = -\left(\mathcal{Z} \bigtimes_{\substack{m=1\\ m\neq n,p}}^{N} \mathbf{a}_r^{(m)} \right) \mathbf{a}_r^{(p)} + \psi_{rr}^{(np)} (\mathbf{a}_r^{(p)\mathsf{T}} \mathbf{a}_r^{(p)})\, \mathbf{a}_r^{(n)} + \sum_{\substack{\ell=1\\ \ell\neq r}}^{R} \psi_{r\ell}^{(np)} (\mathbf{a}_r^{(p)\mathsf{T}} \mathbf{a}_\ell^{(p)})\, \mathbf{a}_\ell^{(n)}.$$

The variables $\mathbf{a}_r^{(p)}$ participate in every summand and (16) follows.

Finally, consider the case of $\mathbf{a}_s^{(p)}$ for $p \neq n$ and $s \neq r$. We can rewrite the first derivative in (8) as

$$\frac{\partial f}{\partial \mathbf{a}_r^{(n)}} = -\left(\mathcal{Z} \bigtimes_{\substack{m=1\\ m\neq n}}^{N} \mathbf{a}_r^{(m)} \right) + \psi_{rs}^{(np)} (\mathbf{a}_r^{(p)\mathsf{T}} \mathbf{a}_s^{(p)})\, \mathbf{a}_s^{(n)} + \sum_{\substack{\ell=1\\ \ell\neq s}}^{R} \gamma_{r\ell}^{(n)} \mathbf{a}_\ell^{(n)}.$$

Observe that $\mathbf{a}_s^{(p)}$ only appears in the second term, and (17) follows. □

4.3 Regularizing the optimization formulation of CPD

CPD is known to be unique when it satisfies the Kruskal conditions [27, 28], but only up to permutation and scaling of the factor matrices. In other words, the CPD 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, the CPD is unchanged by scaling, e.g.,

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

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 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:

$$\hat{f}(\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}) \equiv \frac{1}{2} \left\| \mathcal{Z} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!] \right\|^2 + \frac{\lambda}{2} \sum_{n=1}^{N} \left\| \mathbf{A}^{(n)} \right\|^2. \tag{19}$$

This is the same approach proposed by Paatero in his NLS formulation [38]. The regularization has the effect of encouraging the norms of the factor matrices to be equal, i.e.,

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

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 4.4 *The partial derivatives of the objective function $\hat{f}$ in (19) are given by*

$$\frac{\partial \hat{f}}{\partial \mathbf{A}^{(n)}} = -\mathbf{Z}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right) + \mathbf{A}^{(n)}\mathbf{\Gamma}^{(n)} + \lambda \mathbf{A}^{(n)}, \tag{20}$$

for $n = 1, \ldots, N$, where $\mathbf{\Gamma}^{(n)}$ is defined in (5).

The proof is straightforward and so is omitted. We compare both regularized and unregularized formulations in §6. It is worth noting that there are many other regularization approaches that can be explored; for example, Navasca et al. [35] penalize the change between factor matrices across iterations and vary the regularization parameter at each iteration.

This page intentionally left blank.

5 Nonlinear least squares approach

Paatero [38, 37, 39] and Tomasi and Bro [46, 48, 49, 47] have formulated the CPD problem as a NLS problem, explicitly computing its Jacobian $\mathbf{J}$ or the normalized form $\mathbf{J}^\mathsf{T}\mathbf{J}$.

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

$$F(\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}) \equiv \boldsymbol{\mathcal{Z}} - [\![\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}]\!] = 0. \tag{21}$$

Thus, we have

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

This can be solved using NLS approaches. In fact, the f from (2) is simply $f(\mathbf{x}) = \frac{1}{2}F(\mathbf{x})^\mathsf{T}F(\mathbf{x})$ where $\mathbf{x}$ is as defined in (7). The derivative is given elementwise as follows.

Theorem 5.1 *The first partial derivative of (21) with respect to $a_{jr}^{(n)}$ is a tensor of size $I_1\times I_2\times\cdots\times I_N$ defined elementwise as*

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

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

$$Q = \prod_{n=1}^{N} I_n. \tag{22}$$

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

$$\mathbf{J} = \begin{bmatrix}\mathbf{J}^{(1)} & \mathbf{J}^{(2)} & \cdots & \mathbf{J}^{(N)}\end{bmatrix},$$

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:

$$\mathbf{J}^{(n)} = \begin{bmatrix}\mathbf{J}^{(n1)} & \mathbf{J}^{(n2)} & \cdots & \mathbf{J}^{(nR)}\end{bmatrix},$$

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 [46].

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 $\boldsymbol{\mathcal{Z}}$ is a tensor of size $5\times4\times3$ 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 $QNR = 360$ nonzeros. Note that this figure is similar to Figure 1 in [38] but the columns are ordered differently, corresponding to our methodology for vectorizing a set of factor matrices as in (7).

Using the Jacobian, (21) can be solved via the Gauss-Newton method. This, however, requires solving a system with Jacobian matrix $\mathbf{J}$ which can be extremely large (size $P\times Q$, even though it is relatively sparse). Thus, Tomasi and Bro [48, 49] have argued that it is preferable to work instead with $\mathbf{J}^\mathsf{T}\mathbf{J}$ using a

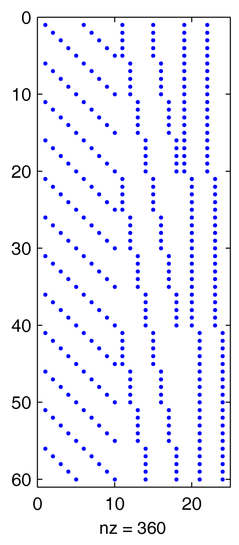

Figure 1: Jacobian nonzero pattern for a tensor of size $5 \times 4 \times 3$ with $R = 2$

Levenberg-Marquardt (LM) method. In this case, the matrix is only of size $P \times P$. Moreover, the inclusion of a multiple of the identity matrix in the LM method serves a regularization function similar to that in (19).

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 $\boldsymbol{\mathcal{U}}$, 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

$$\mathbf{V}^{(n)} = \mathbf{U}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right).$$

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 $\boldsymbol{\mathcal{F}}$ 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

$$\begin{aligned}
\mathbf{G}^{(n)} &= -\mathbf{F}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right) \\
&= \left(-\mathbf{Z}_{(n)} + \mathbf{A}^{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right)^{\mathsf{T}} \right) \\
&\qquad \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right) \\
&= -\mathbf{Z}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right) + \mathbf{A}^{(n)} \boldsymbol{\Gamma}^{(n)}.
\end{aligned}$$

This is, as expected, the gradient in (14).

6 Numerical results

We compare the ALS, OPT, and NLS approaches for computing CPD, employing the data generation methodology of [49]. The data, described in §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 the CPD with the number of components equal to the rank of the tensor and equal to one more than its rank (this is called *overfactoring*). The details of the implementations and parameter settings are given in §6.2. In §6.3, we analyze the results. Our comparisons of ALS and NLS are consistent with [49]. ALS can be remarkably fast but is not very accurate, whereas NLS is accurate but slow. Our OPT methods are as accurate as NLS but competitive with ALS in terms of time (OPT is approximately three times slower but scales at the same rate as ALS).

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 [49], 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 [49]. 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

$$\mathbf{a}_r^{(n)\mathsf{T}}\mathbf{a}_s^{(n)} = C \text{ for } r \neq s,\ r,s = 1,\ldots,R_{\text{true}}, \text{ and } n = 1,\ldots,N. \tag{23}$$

The goal is to recover these underlying factor matrices once they have been assembled into a tensor and noise has been added. 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}.

From each set of factor matrices, nine test tensors are created by adding different levels of homoscedastic and heteroscedastic noise as follows. We set $l_1 = 1, 5, 10$ and $l_2 = 0, 1, 5$ to be the desired noise ratios of homoscedastic and heteroscedastic noise, respectively (corresponding to the values used in [49]). 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:

$$\begin{aligned}
\mathcal{Z} &= [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!], && \text{(original tensor)}\\
\mathcal{Z}' &= \mathcal{Z} + (100/l_1 - 1)^{-1/2}\frac{\|\mathcal{Z}\|}{\|\mathcal{N}_1\|}\mathcal{N}_1, && \text{(homoscedastic noise added)}\\
\mathcal{Z}'' &= \mathcal{Z}' + (100/l_2 - 1)^{-1/2}\frac{\|\mathcal{Z}'\|}{\|\mathcal{N}_2 * \mathcal{Z}'\|}\mathcal{N}_2 * \mathcal{Z}', && \text{(heteroscedastic noise added)}
\end{aligned}$$

where $\mathcal{Z}'$ is used when $l_2 = 0$.

To summarize, we generate a total of 720 cubic three-way test tensors for sizes 20, 50, and 100; and we do a total of 1440 CPD calculations (using $R = R_{\text{true}}$ and $R = R_{\text{true}} + 1$). The 720 test tensors come from two true ranks ($R_{\text{true}} = 3, 5$), two collinearity levels ($C = 0.5, 0.9$), twenty sets of factor matrices per combination of C and R_{true}, and nine noise level combinations ($l_1 = 1, 5, 10$ and $l_2 = 0, 1, 5$). We generate only 360 test matrices (resulting in 720 tests) for size 250 because we only use $C = 0.5$.

6.2 Implementation details

In the results presented here, CPALS denotes the ALS method presented in §3, CPNLS denotes the NLS method presented in §5, and CPOPT (CPOPTR) denotes the new gradient-based optimization approach (with regularization) presented in §4. All experiments were performed using MATLAB v7.6 and the Tensor Toolbox v2.2 [3, 4]. The details of the implementation of each method, as well as the expected computational cost, are discussed below. Initial points for all methods were generated using the n-mode singular vectors of the tensor (i.e., the `nvecs` command in the Tensor Toolbox).

6.2.1 CPALS

For CPALS, the `parafac_als` implementation in the Tensor Toolbox was used. Each iteration requires the computation of $\mathbf{A}^{(n)}$ for $n = 1, \ldots, N$ (see Eq. 4). 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 (4) is the matricized-tensor times Khatri-Rao product, i.e.,

$$\mathbf{Z}_{(n)} \left(\mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)} \right). \tag{24}$$

There is a special function for this computation in the Tensor Toolbox called `mttkrp`. The primary cost in (24) is multiplying the matrix $\mathbf{Z}_{(n)}$ of size $I_n \times (Q/I_n)$, where Q is defined in (22), times 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 (24), is $O(NQR)$.

6.2.2 CPNLS

For CPNLS, the `INDAFAC` code by Tomasi [47] was used to solve the CPD formulated as in (21). This code implements a damped Gauss-Newton algorithm devised by Levenberg and Marquardt [32] which works with modified normal equations, i.e.,

$$\left(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \lambda \mathbf{I} \right) \Delta \mathbf{x} = -\mathbf{J}^{\mathsf{T}} F, \tag{25}$$

where λ is updated at each iteration. The Jacobian $\mathbf{J}$ is singular due to the scaling indeterminacy of CPD [38, 39, 48, 49]; therefore, the modified normal equations in the LM approach can be thought of as a form of regularization as discussed in §4.3 for the OPT approach. As noted in §5, the Jacobian is sparse and therefore $\mathbf{J}^{\mathsf{T}}\mathbf{J}$ can be computed efficiently. The primary expense at each iteration of CPNLS is solving the system in (25) at a cost of $O(P^3)$ operations, where P is as defined in (6). Iterative approaches such as the conjugate gradient method can also be employed (see, e.g., [39]) and are a topic of future study. The INDAFAC code was selected because it implements the LM method described in [49] and is freely available. We note that `INDAFAC` can also handle missing values as described in [48], but that functionality is not used in these experiments.

6.2.3 CPOPT

The methods of CPOPT and CPOPTR were developed using the Tensor Toolbox and the Poblano Toolbox[1]. At each iteration, the function in (2) and the gradient in (14) for $n = 1, \ldots, N$ must be computed. This is very similar to the computation of the ALS update in (4). In fact, the same matricized-tensor times Khatri-Rao product (`mttkrp`) computation given in (24) 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 CPOPTR is the addition of the normalization term in the function and gradient, which has little impact on the cost per evaluation; see (19) and (20).

Gradient-based optimization is performed using the nonlinear conjugate gradient (NCG) method with Hestenes-Stiefel updates; see, e.g., [36]. The Moré-Thuente line search [34], adapted for MATLAB by Dianne P. O'Leary, was used for globalization of the NCG method.

[1]A MATLAB toolbox developed at Sandia National Laboratories for gradient-based optimization.

It is important to observe that the cost per function/gradient evaluation in CPOPT is equivalent to one outer iteration of CPALS; therefore, even though we cannot predict how many iterations of CPALS or function evaluations in CPOPT we require for a given CPD, we might expect the overall computational costs of CPOPT and CPALS 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 [35].

6.2.4 Stopping conditions

In order to produce comparable results, all methods were modified to share a common stopping criterion, the one commonly used for termination in ALS methods. This is the relative change in the function value of f in (2); specifically, the method 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 of the method, respectively.

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 [49]. 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 A. 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 CPD calculation, and timings are written as $a \pm b$ where a is the average time and b is the sample standard deviation. The accuracy of each method is assessed in terms of its ability to recover the original set of factor matrices that was used to generate the test tensor before noise was added. Since CPD is unique only up to a permutation of the component rank-one tensors, we have to consider all permutations of the extracted components, choosing the permutation having the highest sum "congruence". 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 [49]:

$$\text{cong}(\mathcal{X}, \mathcal{Y}) = \frac{|\mathbf{a}^T\mathbf{p}|}{||\mathbf{a}||_2||\mathbf{p}||_2} \times \frac{|\mathbf{b}^T\mathbf{q}|}{||\mathbf{b}||_2||\mathbf{q}||_2} \times \frac{|\mathbf{c}^T\mathbf{r}|}{||\mathbf{c}||_2||\mathbf{r}||_2}. \tag{26}$$

There is also sign ambiguity among the vectors comprising each component rank-one tensor, i.e., $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c} = (-\mathbf{a}) \circ (-\mathbf{b}) \circ \mathbf{c}$, which is why absolute values are used in the numerators of (26). If the best sum congruence (across all permutations) is above a threshold of 0.97 ($\approx 0.99^3$) for all components, then we say that the CPD computed the correct factorization. Here, we consider the results summarized according to different experimental parameters.

6.3.1 Tensor size

The CPOPT method is as accurate as CPNLS and competitive with CPALS in terms of computation time. Table 1 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.

	Time (sec)			
Size	**CPALS**	**CPNLS**	**CPOPT**	**CPOPTR**
$20 \times 20 \times 20$	0.5 ± 1.0	1.0 ± 1.1	0.3 ± 0.2	0.2 ± 0.1
$50 \times 50 \times 50$	0.3 ± 0.3	16.0 ± 17.7	0.7 ± 0.5	0.5 ± 0.1
$100 \times 100 \times 100$	1.7 ± 1.1	153.2 ± 142.3	5.6 ± 3.6	4.3 ± 1.3
$250 \times 250 \times 250$	26.6 ± 9.1	—	83.5 ± 35.2	81.9 ± 22.8

	Accuracy (%)			
Size	**CPALS**	**CPNLS**	**CPOPT**	**CPOPTR**
$20 \times 20 \times 20$	78.8	99.7	99.9	100.0
$50 \times 50 \times 50$	65.7	99.9	100.0	100.0
$100 \times 100 \times 100$	63.5	99.9	100.0	100.0
$250 \times 250 \times 250$	62.2	—	100.0	100.0

Table 1: Speed and accuracy comparison with collinearity $C = 0.5$

Recall that the cost for one iteration of CPALS is equal to that of 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 much slower than the other methods because it must solve the modified Gauss-Newton equation at each inner iteration. In fact, results for CPNLS on problems of size $250 \times 250 \times 250$ were not conducted because *each factorization* required approximately five hours of computation time, as compared to 26.6 and 83.5 seconds for CPALS and CPOPT, respectively. CPALS and CPOPT scale much better than CPNLS.

In terms of accuracy, both CPNLS and CPOPT are essentially perfect, but CPALS only obtains accuracies of 62-79%.

6.3.2 Number of components in the factorization

The difference in the accuracy of the methods can be attributed to their performance in the case of overfactoring ($R = R_{\text{true}} + 1$). Figure 2 illustrates the accuracy and timing results for tensors of size $50 \times 50 \times 50$ in Table 1, separating the $R = R_{\text{true}}$ (blue) and $R = R_{\text{true}} + 1$ (red) cases.

In Figure 2, we can observe that CPALS has trouble with overfactoring. We also note that the run times go up for all methods in the overfactoring case, particularly CPNLS.

To further explore the phenomena, we considered publically available data [6] which comprises five chemical samples measured by fluorescence at 61 excitation and 201 emission wavelengths forming a third-order tensor with modes: samples, emission, and excitation wavelengths. Each of the five samples contains different amounts of three amino acids [5]. It is known that the number of true underlying components in the data is three; therefore, the signatures of these chemicals in emission and excitation modes are accurately captured by computing a CPD 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 [5] such that an extra component may partially capture it. We, therefore, compute CPDs with $R = 3$, 4, and 5 to compare the effects of overfactoring for CPALS and CPOPT.

Figure 3 plots the emission-mode factors, i.e., the columns of a CPD factor matrix for the emission mode, where each column of the matrix is scaled by the norm of the corresponding component. These should be the *emission signatures* of the chemical analytes in the sample. We compare $R = 3$ (top), $R = 4$ (middle) and $R = 5$ (bottom) components.

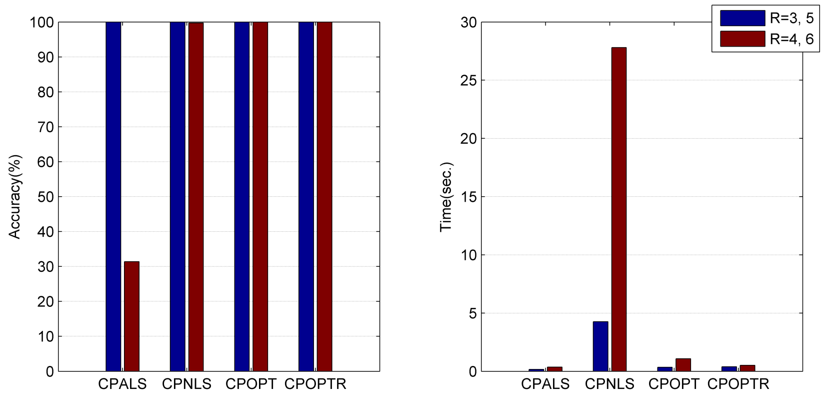

Figure 2: Speed and accuracy comparison for extracting the number of true underlying components (blue) and overfactoring (red) on tensors of size $50 \times 50 \times 50$ and collinearity $C = 0.5$

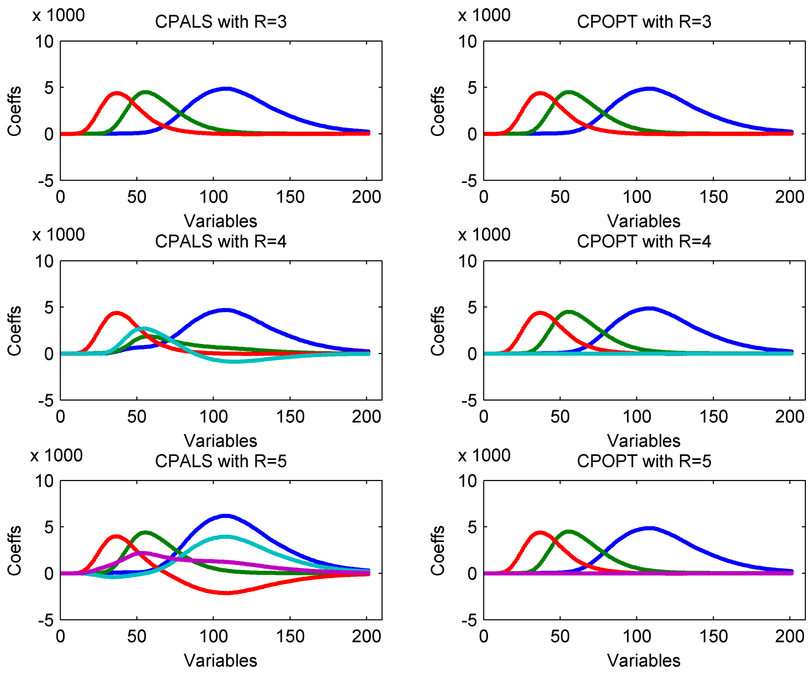

Figure 3: Factors corresponding to the emission mode of the amino acid fluorescence data set [5, 6]. Plots are shown for the CPDs computed using CPALS (left) and CPOPT (right) with $R = 3$ (top), $R = 4$ (middle) and $R = 5$ (bottom) components

From Figure 3, we can see that both CPALS and CPOPT extract the same emission factors for $R = 3$. In the case of $R = 4$, the factors for CPALS change, especially the green factor. For CPOPT, however, the first three factors for $R = 4$ match those in the $R = 3$ case and the fourth factor is very small. The case for $R = 5$ is similar. The norms of the factors and timings of the methods are reported in Table 2.

	CPALS						**CPOPT**					
	Component Norm ($\times 10^4$)					**Time**	**Component Norm** ($\times 10^4$)					**Time**
Components	**1**	**2**	**3**	**4**	**5**	**(sec)**	**1**	**2**	**3**	**4**	**5**	**(sec)**
$R = 3$	3.3	2.3	2.1			0.5	3.3	2.3	2.1			1.5
$R = 4$	3.3	1.1	2.1	1.4		6.2	3.3	2.3	2.1	10^{-4}		1.7
$R = 5$	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 2: Norms of the component rank-one tensors and timings for the results shown in Figure 3

From Table 2, we can see that the extra components computed by CPOPT are relatively small in magnitude. Moreover, we see that CPOPT is actually much faster than CPALS in both cases of overfactoring.

6.3.3 Collinearity

The performance of all four methods degrade as the collinearity, see (23), is increased. This means that there is more overlap in the factors. Table 3 shows results with $C = 0.9$ comparable to those in Table 1, except that this table only goes up to size 100.

	Time (sec)			
Size	**CPALS**	**CPNLS**	**CPOPT**	**CPOPTR**
$20 \times 20 \times 20$	1.1 ± 0.8	2.3 ± 2.1	0.7 ± 0.3	0.7 ± 0.3
$50 \times 50 \times 50$	1.8 ± 0.9	27.7 ± 23.5	1.5 ± 0.7	1.5 ± 0.6
$100 \times 100 \times 100$	14.2 ± 5.5	275.7 ± 266.4	12.9 ± 5.0	12.9 ± 4.8

	Accuracy (%)			
Size	**CPALS**	**CPNLS**	**CPOPT**	**CPOPTR**
$20 \times 20 \times 20$	29.0	32.2	32.2	32.6
$50 \times 50 \times 50$	65.6	71.4	69.9	69.9
$100 \times 100 \times 100$	73.1	81.3	79.4	79.7

Table 3: Speed and accuracy comparison with collinearity $C = 0.9$

We can see from Table 3 that the higher collinearity problems take more time to solve and are much 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 very 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. A breakdown of results for size 50 is shown in Figure 4.

Figure 4 shows speed and accuracy results for tensors of size $50 \times 50 \times 50$ and $C = 0.9$. In comparison to Figure 2 where $C = 0.5$, Figure 4 shows that the accuracies of the methods are low even if the number of true underlying rank-one factors is computed by setting $R = R_{\text{true}}$. A comparable result is also stated in [49]. In fact, accuracy improves for CPNLS, CPOPT, and CPOPTR in the case of overfactoring (red).

6.3.4 Noise level

As noise level increases, particularly the heteroscedastic noise (l_2), the accuracy of the methods decreases if $R = R_{\text{true}}$, as shown on the left-hand side of Figure 5. Interestingly, a higher noise level *increased* the accuracy of CPALS when $R = R_{\text{true}} + 1$, as shown on the right column of Figure 5. We hypothesize that the extra component is modeling the noise.

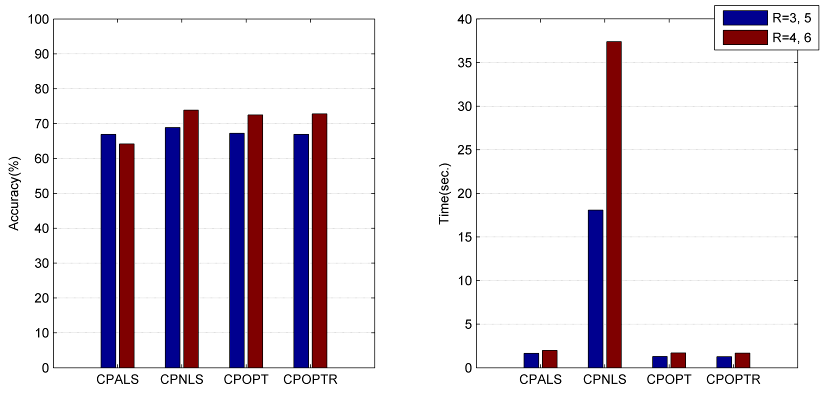

Figure 4: Speed and accuracy comparison for computing the number of true underlying components (blue) and overfactoring (red) for tensors of size $50 \times 50 \times 50$ and collinearity $C = 0.9$

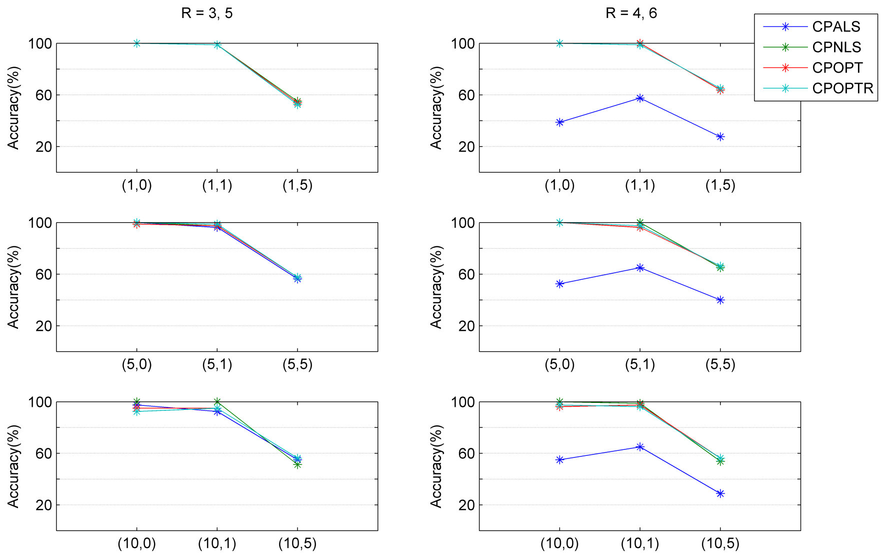

Figure 5: Accuracy of different methods for computing CPD for $R = R_{\text{true}}$ (left) and $R = R_{\text{true}} + 1$ (right) at increasing levels of noise for tensors of size $50 \times 50 \times 50$ taking into consideration both collinearity levels, $C = 0.5$ and $C = 0.9$. The numbers on the x-axis indicate (l_1, l_2) pairs. Each subplot keeps homoscedastic noise ratio (l_1) constant and changes heteroscedastic noise ratio (l_2).

6.3.5 Rank

The main effect of the rank of the tensor is in the computation time. Figure 6 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 analogous timings for $R_{\text{true}} = 5$. We can see that the computation time of CPNLS increases significantly when 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 §5 for more details).

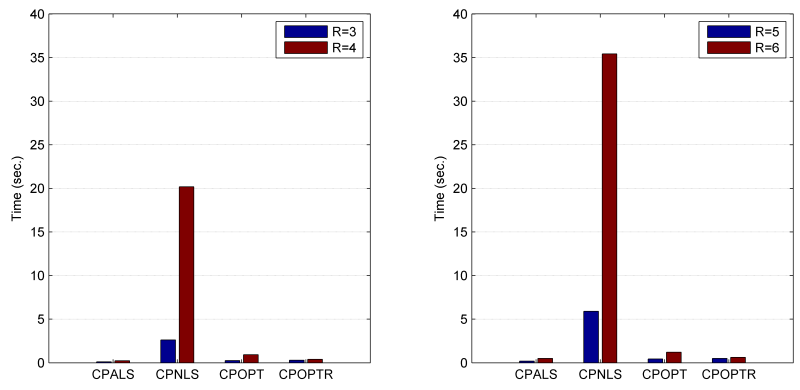

Figure 6: Speed for $R_{\text{true}} = 3$ and $R_{\text{true}} = 5$ for tensors of size $50 \times 50 \times 50$ at $C = 0.5$

We also consider the effect on accuracy in Figure 7. In terms of accuracy, all methods except CPALS perform close to 100%, while CPALS suffers from overfactoring. Interestingly, CPALS computes more accurate CPDs in the case of overfactoring for $R_{\text{true}} = 5$ compared to those for $R_{\text{true}} = 3$. Further studies are needed to determine if this trend continues as the tensor rank increases.

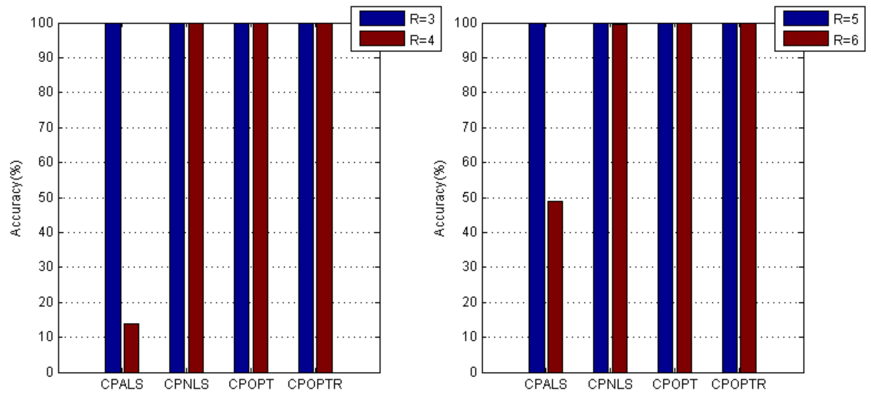

Figure 7: Accuracy for $R_{\text{true}} = 3$ and $R_{\text{true}} = 5$ for tensors of size $50 \times 50 \times 50$ at $C = 0.5$

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 A.2a in Appendix A illustrates this trend for the high collinearity case, which is not present in the low collinearity case (i.e., $C = 0.5$). In terms of computation time, for $R_{\text{true}} = 5$, the relative performance of the algorithms is the same as for $R_{\text{true}} = 3$ but there is an increase in computation time when we go from $R_{\text{true}} = 3$ to $R_{\text{true}} = 5$ (Table A.2b and Table A.2c in Appendix A).

7 Conclusions

Although both ALS [8, 20] and NLS [38, 37, 39, 46, 48] are optimization-based approaches to solving the CPD problem, we revisit the problem and consider yet another alternative. The OPT method proposed here is a gradient-based optimization method; specifically, we use a nonlinear conjugate gradient method to solve the CPD optimization problem in (2). In contrast to ALS, OPT solves for all factor matrices simultaneously and our numerical results show that this leads to increased accuracy, especially in the case of overfactoring. In contrast to NLS, we sacrifice the quadratic convergence rate of the Levenberg-Marquardt method, but the overall speed of OPT is significantly faster because its cost per iteration is much less.

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. An interesting result of our work is that there is a clear connection between the OPT and ALS: ALS sets the gradient to zero for just one factor matrix at a time, whereas OPT sets the gradient to zero for all factor matrices simultaneously. This is a new way of deriving the ALS equations. Moreover, this connection between ALS and OPT means that the same methods to making ALS applicable to large-scale problems [3] 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 CPD problem is addressing the scaling and permutation indeterminacies. We extended the OPT method to include a Tikhonov regularization term and have described how this addresses the indeterminacies. We have also illustrated the connections between OPT and the Levenberg-Marquardt method in NLS. Future work will consider investigating how to choose the best regularization parameter.

The formulation of the CPD 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 we use the n-mode singular vectors to initialize the methods and this would need to change with the introduction of such constraints and therefore be another avenue of investigation itself.

References

[1] E. Acar and B. Yener, *Unsupervised multiway data analysis: A literature survey*, IEEE Transactions on Knowledge and Data Engineering, 21 (2009), pp. 6–20.

[2] C. M. Andersen and R. Bro, *Practical aspects of PARAFAC modeling of fluorescence excitation-emission data*, Journal of Chemometrics, 17 (2003), pp. 200–215.

[3] B. W. Bader and T. G. Kolda, *Efficient MATLAB computations with sparse and factored tensors*, SIAM Journal on Scientific Computing, 30 (2007), pp. 205–231.

[4] ——, *Tensor toolbox for matlab, version 2.2*, last accessed November, 2008. `http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/`.

[5] R. Bro, *PARAFAC. Tutorial and applications*, Chemometrics and Intelligent Laboratory Systems, 38 (1997), pp. 149–171.

[6] ——, *Amino acids fluorescence data*, last accessed December, 2008. `http://www.models.kvl.dk/research/data/Amino_Acid_fluo/index.asp`.

[7] R. Bro and H. A. L. Kiers, *A new efficient method for determining the number of components in PARAFAC models*, Journal of Chemometrics, 17 (2003), pp. 274–286.

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

[9] Z. Chen, Y. Li, and R. Yu, *Pseudo alternating least squares algorithm for trilinear decomposition*, Journal of Chemometrics, 15 (2001), pp. 149–167. Cited by [15].

[10] Z.-P. Chen, H.-L. Wu, J.-H. Jiang, Y. Li, and R.-Q. Yu, *A novel trilinear decomposition algorithm for second-order linear calibration*, Chemometrics and Intelligent Laboratory Systems, 51 (2000), pp. 75–86.

[11] Z.-P. Chen, H.-L. Wu, and R.-Q. Yu, *On the self-weighted alternating trilinear decomposition algorithm — the property of being insensitive to excess factors used in calculation*, Journal of Chemometrics, 15 (2001), pp. 439–453.

[12] L. De Lathauwer, *A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 642–666.

[13] L. De Lathauwer, B. De Moor, and J. Vandewalle, *Computation of the canonical decomposition by means of a simultaneous generalized Schur decomposition*, SIAM Journal on Matrix Analysis and Applications, 26 (2004), pp. 295–327.

[14] K. Faber, *Short communication: On solving generalized eigenvalue problems using matlab*, Journal of Chemometrics, 11 (1997), pp. 87–91. Cited by [15].

[15] N. K. M. Faber, R. Bro, and P. K. Hopke, *Recent developments in CANDECOMP/PARAFAC algorithms: A critical review*, Chemometrics and Intelligent Laboratory Systems, 65 (2003), pp. 119–137.

[16] N. M. Faber, L. M. C. Buydens, and G. Kateman, *Generalized rank annihilation method. i: Derivation of eigenvalue problems*, Journal of Chemometrics, 8 (1994), pp. 147–154. Cited by [15].

[17] ——, *Generalized rank annihilation method. iii: Practical implementation*, Journal of Chemometrics, 8 (1994), pp. 273–285. Cited by [15].

[18] M. Gerritsen, H. Tanis, B. Vandeginste, and G. Kateman, *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, 64 (1992), pp. 2029–2035. Cited by [15].

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, 1996.

[20] R. A. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis*, UCLA working papers in phonetics, 16 (1970), pp. 1–84. Available at `http://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf`.

[21] J. Håstad, *Tensor rank is NP-complete*, Journal of Algorithms, 11 (1990), pp. 644–654.

[22] J. Jiang, H. Wu, Y. Li, and R. Yu, *Three-way data resolution by alternating slice-wise diagonalization (ASD) method*, Journal of Chemometrics, 14 (2000), pp. 15–36.

[23] J.-H. Jiang, H.-L. Wu, Y. Li, and R.-Q. Yu, *Alternating coupled vectors resolution (acover) method for trilinear analysis of three-way data*, Journal of Chemometrics, 13 (1999), pp. 557–578. Cited by [15].

[24] H. A. L. Kiers, *Towards a standardized notation and terminology in multiway analysis*, Journal of Chemometrics, 14 (2000), pp. 105–122.

[25] T. G. Kolda, *Multilinear operators for higher-order decompositions*, Tech. Report SAND2006-2081, Sandia National Laboratories, Albuquerque, New Mexico and Livermore, California, Apr. 2006.

[26] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Review. To appear (accepted June 2008).

[27] J. B. Kruskal, *Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics*, Linear Algebra and its Applications, 18 (1977), pp. 95–138.

[28] J. B. Kruskal, *Rank, decomposition, and uniqueness for 3-way and N-way arrays*, in Multiway Data Analysis, R. Coppi and S. Bolasco, eds., North-Holland, Amsterdam, 1989, pp. 7–18.

[29] S. Li, J. Hamilton, and P. Gemperline, *Generalized rank annihilation method using similarity transformations*, Analytical Chemistry, 64 (1992), pp. 599–607. Cited by [15].

[30] Y. Li, J. Jiang, H. Wu, Z. Chen, and R. Yu, *Alternating coupled matrices resolution method for three-way arrays analysis*, Chemometrics and Intelligent Laboratory Systems, 52 (2000), pp. 33–43. Cited by [15].

[31] A. Lorber, *Features of quantifying chemical composition from two-dimensional data array by the rank annihilation factor analysis method*, Analytical Chemistry, 57 (1985), pp. 2395–2397. Cited by [15].

[32] K. Madsen, H. B. Nielson, and O. Tingleff, *Methods for non-linear least squares problems, 2nd edition*, Informatics and Mathematical Modelling, Technical University of Denmark, Apr. 2004.

[33] B. C. Mitchell and D. S. Burdick, *Slowly converging PARAFAC sequences: Swamps and two-factor degeneracies*, Journal of Chemometrics, 8 (1994), pp. 155–168.

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

[35] C. Navasca, L. De Lathauwer, and S. Kinderman, *Swamp reducing technique for tensor decompositions*. Submitted for publication, Mar. 2008.

[36] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, 1999.

[37] P. Paatero, *Least squares formulation of robust non-negative factor analysis*, Chemometrics and Intelligent Laboratory Systems, 37 (1997), pp. 23–35.

[38] ——, *A weighted non-negative least squares algorithm for three-way "PARAFAC" factor analysis*, Chemometrics and Intelligent Laboratory Systems, 38 (1997), pp. 223–242.

[39] ——, *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, 8 (1999), pp. 854–888.

[40] ——, *Construction and analysis of degenerate PARAFAC models*, Journal of Chemometrics, 14 (2000), pp. 285–299.

[41] W. S. Rayens and B. C. Mitchell, *Two-factor degeneracies and a stabilization of PARAFAC*, Chemometrics and Intelligent Laboratory Systems, 38 (1997), p. 173.

[42] E. Sanchez and B. Kowalski, *Generalized rank annihilation factor analysis*, Analytical Chemistry, 58 (1986), pp. 496–499. Cited by [15].

[43] A. Shashua and T. Hazan, *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.

[44] A. Smilde, R. Bro, and P. Geladi, *Multi-Way Analysis: Applications in the Chemical Sciences*, Wiley, West Sussex, England, 2004.

[45] A. Stegeman and L. De Lathauwer, *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, 30 (2009), pp. 1614–1638.

[46] G. Tomasi, *Use of the properties of the Khatri-Rao product for the computation of Jacobian, Hessian, and gradient of the PARAFAC model under MATLAB*, 2005. Private communication.

[47] ——, *Incomplete data PARAFAC (INDAFAC)*, last accessed November, 2008. `http://www.models.kvl.dk/source/indafac/index.asp`.

[48] G. Tomasi and R. Bro, *PARAFAC and missing values*, Chemometrics and Intelligent Laboratory Systems, 75 (2005), pp. 163–180.

[49] ——, *A comparison of algorithms for fitting the PARAFAC model*, Computational Statistics & Data Analysis, 50 (2006), pp. 1700–1734.

[50] H.-L. Wu, M. Shibukawa, and K. Oguma, *An alternating trilinear decomposition algorithm with application to calibration of HPLC–DAD for simultaneous determination of overlapped chlorinated aromatic hydrocarbons*, Journal of Chemometrics, 12 (1998), pp. 1–26.

A Detailed numerical results

The numerical results in §6.3 are derived from the detailed results presented in Tables A.1 – A.4. Each cell corresponds to twenty sets of factor matrices, each contaminated with nine levels of noise as described in §6.1, for a total of 180 test tensors. Throughout, we report the time per CPD calculation and timings are written as $a \pm b$ where a is the average time and b is the sample standard deviation.

	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	100.0	52.2	52.2	100.0	98.9	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

(a) Accuracy

	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	0.2 ± 0.0	1.2 ± 0.6	0.6 ± 0.5	1.7 ± 0.9
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

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

	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	0.4 ± 0.1	2.3 ± 1.4	2.8 ± 1.9	4.3 ± 2.2
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

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

Table A.1: Detailed results for $20 \times 20 \times 20$ tensors

	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	100.0	72.8	81.7	100.0	99.4	65.0	66.1
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

(a) Accuracy

	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	2.6 ± 0.3	20.2 ± 9.9	9.5 ± 9.1	24.3 ± 13.4
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

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

	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	5.9 ± 1.0	35.4 ± 21.8	26.7 ± 21.1	50.4 ± 25.4
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

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

Table A.2: Detailed results for $50 \times 50 \times 50$ tensors

	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	100.0	90.6	99.4	100.0	99.4	67.2	67.8
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

(a) Accuracy

	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	29.2 ± 3.2	194.9 ± 84.7	72.7 ± 59.4	209.0 ± 82.0
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

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

	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	64.4 ± 8.9	324.4 ± 139.4	295.0 ± 265.0	525.9 ± 308.7
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

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

Table A.3: Detailed results for $100 \times 100 \times 100$ tensors

	Accuracy (%)			
Rank (R_{true})	**3**		**5**	
Collinearity (C)	**0.5**		**0.5**	
Extr. Components (R)	**3**	**4**	**5**	**6**
CPALS	100.0	7.2	100.0	41.7
CPOPT	100.0	100.0	100.0	100.0
CPOPTR	100.0	100.0	100.0	100.0

(a) Accuracy

	Time (sec.)			
Rank (R_{true})	**3**		**5**	
Collinearity (C)	**0.5**		**0.5**	
Extr. Components (R)	**3**	**4**	**5**	**6**
CPALS	22.2 ± 1.5	19.8 ± 4.5	26.5 ± 1.5	37.7 ± 11.0
CPOPT	62.4 ± 13.1	70.2 ± 28.3	77.0 ± 13.7	124.4 ± 38.2
CPOPTR	70.5 ± 13.9	60.7 ± 9.5	85.9 ± 13.9	110.2 ± 14.7

(b) Computational time

Table A.4: Detailed results for $250 \times 250 \times 250$ tensors