Received: 14 June 2022 | Revised: 8 May 2023 | Accepted: 15 May 2023

DOI: 10.1002/nla.2511

RESEARCH ARTICLE

WILEY

CP decomposition for tensors via alternating least squares with QR decomposition

Rachel Minster[1] | **Irina Viviano**[2] | **Xiaotian Liu**[1] | **Grey Ballard**[1]

[1]Department of Computer Science, Wake Forest University, Winston-Salem, North Carolina, USA

[2]Clinical & Translational Science Institute, Wake Forest University School of Medicine, Winston-Salem, North Carolina, USA

Correspondence
Rachel Minster, Department of Computer Science, Wake Forest University, Winston-Salem, NC, USA.
Email: minsterr@wfu.edu

Funding information
National Science Foundation, Grant/Award Number: CCF-1942892

Abstract

The CP tensor decomposition is used in applications such as machine learning and signal processing to discover latent low-rank structure in multidimensional data. Computing a CP decomposition via an alternating least squares (ALS) method reduces the problem to several linear least squares problems. The standard way to solve these linear least squares subproblems is to use the normal equations, which inherit special tensor structure that can be exploited for computational efficiency. However, the normal equations are sensitive to numerical ill-conditioning, which can compromise the results of the decomposition. In this paper, we develop versions of the CP-ALS algorithm using the QR decomposition and the singular value decomposition, which are more numerically stable than the normal equations, to solve the linear least squares problems. Our algorithms utilize the tensor structure of the CP-ALS subproblems efficiently, have the same complexity as the standard CP-ALS algorithm when the input is dense and the rank is small, and are shown via examples to produce more stable results when ill-conditioning is present. Our MATLAB implementation achieves the same running time as the standard algorithm for small ranks, and we show that the new methods can obtain lower approximation error.

KEYWORDS

canonical polyadic tensor decomposition, CANDECOMP/PARAFAC, multilinear algebra, numerical stability, tensors

1 | INTRODUCTION

The CANDECOMP/PARAFAC or canonical polyadic (CP) decomposition for multidimensional data, or tensors, is a popular tool for analyzing and interpreting latent patterns that may be present in multidimensional data. One of the most popular methods used to compute a CP decomposition is the alternating least squares (CP-ALS) approach, which solves a series of linear least squares problems.[1-3] To solve these linear least squares problems, CP-ALS uses the normal equations, which are well known to be sensitive to roundoff error for moderately ill-conditioned matrices. We propose to use a more stable approach, where we solve the linear least squares problems using the QR decomposition instead.

Consider the standard linear least squares problem with multiple outputs, that is, $\min_{\mathbf{X}} \|\mathbf{AX} - \mathbf{B}\|_F$. The normal equations for this problem are $\mathbf{A}^\top\mathbf{AX} = \mathbf{A}^\top\mathbf{B}$. Given the compact/thin QR decomposition $\mathbf{A} = \mathbf{QR}$, the more numerically

stable solution is computed from $\mathbf{RX} = \mathbf{Q}^\top\mathbf{B}$. When $\mathbf{A}$ is tall and skinny and $\mathbf{B}$ has few columns, the normal equations approach has about half the cost of the QR-based approach because the dominant costs are computing $\mathbf{A}^\top\mathbf{A}$ and the QR decomposition, respectively. However, if $\mathbf{B}$ has many more columns than $\mathbf{A}$, then the dominant costs are those of computing $\mathbf{A}^\top\mathbf{B}$ and $\mathbf{Q}^\top\mathbf{B}$, which are equivalent when $\mathbf{Q}$ is formed explicitly. In this case, the QR-based approach is more numerically stable and requires practically no more computation compared to the normal equations approach. Furthermore, for rank-deficient problems, the QR-based approach can be cheaply extended to use the singular value decomposition (SVD) to solve the least squares problem, computing the minimum norm solution among the set of solutions with equivalent residual norm.

As we describe in more detail in Section 2, when solving linear least squares problems within CP-ALS, $\mathbf{A}$ corresponds to a Khatri–Rao product of factor matrices, and $\mathbf{B}$ corresponds to the transpose of a matricized tensor. In this case, the number of columns of $\mathbf{A}$ is the rank of the CP decomposition, and the number of columns of $\mathbf{B}$ corresponds to one of the tensor dimensions. The normal equations are particularly convenient within CP-ALS because the Khatri–Rao structure of $\mathbf{A}$ can be exploited to compute $\mathbf{A}^\top\mathbf{A}$ very efficiently, and thus, even for large ranks, the dominant cost is computing $\mathbf{A}^\top\mathbf{B}$, whose transpose is known as the matricized-tensor times Khatri–Rao product (MTTKRP). The MTTKRP is a well-studied and well-optimized computation because of its importance for the performance of CP-ALS and other gradient-based optimization algorithms for CP.[4-7]

In Section 3.1, we present a QR-based approach to solving the linear least squares problems within CP-ALS. In order to achieve comparable computational complexity with the normal equations approach, we exploit the Khatri–Rao structure of $\mathbf{A}$ in the computation of the QR decomposition as well as $\mathbf{Q}^\top\mathbf{B}$. In particular, we show that the QR decomposition of a Khatri–Rao product of matrices can be computed efficiently from the QR decompositions of the individual factor matrices. Using the structure of the orthonormal component $\mathbf{Q}$, the computation of $\mathbf{Q}^\top\mathbf{B}$ involves multiple tensor-times-matrix (Multi-TTM) products. We prove in Section 3.2 that when the rank is small relative to the tensor dimensions, the Multi-TTM is the dominant cost, and it has the same leading-order complexity as MTTKRP for dense tensors. Multi-TTM is also a well-optimized tensor computation, as it is important for the computation of Tucker decompositions.[8-11]

When the rank is comparable to or much larger than the tensor dimensions, the QR-based approach can require significantly more computation time than standard CP-ALS using the normal equations as computing the QR of the factor matrices does not reduce the problem size. Both the QR decomposition and application of the orthonormal component have costs that are lower order only when the rank is small. In this case, the numerical stability provided by QR comes at the expense of performance.

We demonstrate the performance and accuracy of our methods using several example input tensors in Section 4. Our MATLAB implementation of the algorithms uses the Tensor Toolbox,[1] and we compare against the CP-ALS algorithm implemented in that library. In Section 4.1, we validate the theoretical complexity analysis and show that there is no increase in periteration costs when the rank is small, and we demonstrate with a time breakdown which of the computations become bottlenecks as the rank grows larger relative to the tensor dimensions. To illustrate the differences in accuracy, we present two sets of examples in Sections 4.2 and 4.3 that lead to ill-conditioned subproblems and show that the instability of the normal equations can lead to the degradation of desirable features including approximation accuracy and in some cases, convergence of the overall algorithm.

We conclude in Section 5 that using the QR-based approaches to solve the CP-ALS subproblems increases the robustness of the overall algorithm without sacrificing performance in the typical case of small ranks. However, due to the complexity of the algorithm and the extra computation that becomes significant for large ranks, we envision a CP-ALS solver that uses the fast-and-inaccurate normal equations approach by default and falls back on the accurate-but-possibly-slow SVD approach when necessary. For problems that do not involve ill-conditioning, which is the case for many tensors representing noisy data, the normal equations are sufficient for obtaining accurate solutions. However, when ill-conditioning degrades the accuracy of solutions computed from the normal equations, we show that it is possible to obtain the stability of the SVD with feasible computational cost.

2 | BACKGROUND

In this section, we first review typical methods for solving linear least squares problems. We also discuss the relevant information regarding tensors and the CP decomposition, focusing on the CP-ALS algorithm. We also briefly describe an optimization approach for CP using the Gauss–Newton algorithm.

2.1 | Linear least squares methods

A common approach to solve linear least squares problems is by solving the associated normal equations. When applied to ill-conditioned problems, however, using the normal equations results in numerical instability. More numerically stable methods to solve least squares problems include using the QR decomposition or the SVD.

Consider a least squares problem of the form

$$\min_{\mathbf{X}} \|\mathbf{B} - \mathbf{X}\mathbf{A}^\top\|_F.$$

Note that the coefficient matrix appears to the right of the variable matrix rather than the left (as appears in Section 1) in order to match the form of the CP-ALS subproblems described below. The normal equations for this problem are $\mathbf{X}\mathbf{A}^\top\mathbf{A} = \mathbf{B}\mathbf{A}$, which is equivalent to $\mathbf{A}^\top\mathbf{A}\mathbf{X}^\top = \mathbf{A}^\top\mathbf{B}^\top$. To solve this least squares problem using QR, we first compute the compact/thin QR factorization of $\mathbf{A} = \mathbf{Q}\mathbf{R}$, so that $\mathbf{Q}$ has the same dimensions as $\mathbf{A}$ and $\mathbf{R}$ is square. We then apply $\mathbf{Q}$ to matrix $\mathbf{B}$ on the right, and use the result to solve the triangular system $\mathbf{X}\mathbf{R}^\top = \mathbf{B}\mathbf{Q}$ for $\mathbf{X}$. When the coefficient matrix $\mathbf{A}$ is tall and skinny, the QR approach can be cheaply extended to use the SVD. Given the QR factorization of $\mathbf{A}$, we compute the SVD of $\mathbf{R} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$. If we apply $\mathbf{U}$ to $\mathbf{B}\mathbf{Q}$ on the right, we can then solve the system $\mathbf{Y}\boldsymbol{\Sigma} = \mathbf{B}\mathbf{Q}\mathbf{U}$ for $\mathbf{Y}$ and compute $\mathbf{X} = \mathbf{Y}\mathbf{V}^\top$. If $\mathbf{A}$ is numerically low rank, we can solve the system using the pseudoinverse of $\boldsymbol{\Sigma}$ to find the minimum-norm solution to the original problem. For more details on methods to solve linear least squares problems, see References 12-14.

2.2 | Tensor notation and preliminaries

Throughout this paper, we follow the notation from Reference 15. A scalar is denoted by lowercase letters, for example a, while vectors are denoted by boldface lowercase letters, for example $\mathbf{a}$. Matrices are denoted by boldface uppercase letters, for example $\mathbf{A}$, and tensors are denoted by boldface uppercase calligraphic letters, for example $\mathcal{X}$. We use the MATLAB notation $\mathbf{A}(i,:)$ to refer to the i^{th} row of $\mathbf{A}$ and $\mathbf{A}(:,j)$ to refer to the j^{th} column of $\mathbf{A}$.

2.2.1 | Matrix products

We define three matrix products that will appear in our algorithms. First, the Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{I\times J}$ and $\mathbf{B} \in \mathbb{R}^{K\times M}$ is denoted $\mathbf{A}\otimes\mathbf{B} \in \mathbb{R}^{(IK)\times(JM)}$ matrix with entries $[\mathbf{A}\otimes\mathbf{B}](K(i-1)+k, M(j-1)+\ell) = \mathbf{A}(i,j)\mathbf{B}(k,\ell)$.

The Khatri–Rao product of matrices $\mathbf{A} \in \mathbb{R}^{I\times K}$ and $\mathbf{B} \in \mathbb{R}^{J\times K}$ is denoted $\mathbf{A}\odot\mathbf{B} \in \mathbb{R}^{(IJ)\times K}$ matrix with columns $[\mathbf{A}\odot\mathbf{B}](:,k) = \mathbf{A}(:,k)\otimes\mathbf{B}(:,k)$ for every $k = 1, \ldots, K$. We present a property regarding the Khatri–Rao product of a product of two matrices from equation 2.5 of Reference 16, which will be useful in a later section. Let $\mathbf{A} \in \mathbb{R}^{K\times J}$, $\mathbf{B} \in \mathbb{R}^{I\times J}$, $\mathbf{C} \in \mathbb{R}^{I\times K}$, and $\mathbf{D} \in \mathbb{R}^{J\times I}$ be four matrices. Then,

$$(\mathbf{C}\otimes\mathbf{D})(\mathbf{A}\odot\mathbf{B}) = (\mathbf{C}\mathbf{A})\odot(\mathbf{D}\mathbf{B}). \tag{1}$$

Finally, the Hadamard product of two matrices $\mathbf{A} \in \mathbb{R}^{I\times J}$ and $\mathbf{B} \in \mathbb{R}^{I\times J}$ is the elementwise product denoted as $\mathbf{A} * \mathbf{B} \in \mathbb{R}^{I\times J}$, with entries $[\mathbf{A} * \mathbf{B}](i,j) = \mathbf{A}(i,j)\mathbf{B}(i,j)$.

2.2.2 | Tensor components

As generalizations to matrix rows and columns, tensors have mode-j fibers, which are vectors formed by fixing all but one index of a tensor.

2.2.3 | Tensor operations

There are two major tensor operations we will frequently use throughout this paper, namely matricization and multiplying a tensor with a matrix. The n-mode matricization, or unfolding, of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_N}$, denoted $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n\times\left(\prod_{j\neq n} I_j\right)}$, and the matrix $\mathbf{X}_{(n)}$ is formed so that the columns are the mode-n fibers of $\mathcal{X}$.

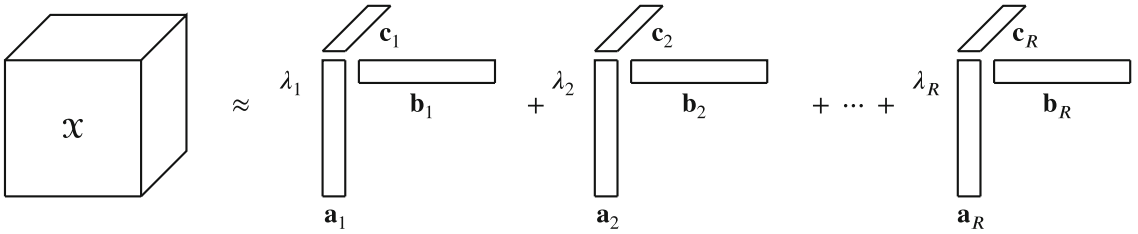


FIGURE 1 CP decomposition of rank R for a three-dimensional tensor $\mathcal{X}$.

Also useful is the TTM multiplication, or the n-mode product of a tensor and matrix. Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathbf{U} \in \mathbb{R}^{J \times I_n}$. The resulting tensor $\mathcal{Y} = \mathcal{X} \times_n \mathbf{U} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$, and it has entries $\mathcal{Y}(i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N) = \sum_{i_n=1}^{I_n} \mathcal{X}(i_1, \ldots, i_N)\mathbf{U}(j, i_n)$. The TTM can also be computed via the mode-n matricization $\mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}$.

Multiplying an N-mode tensor by multiple matrices in distinct modes is known as Multi-TTM. The computation can be performed using a sequence of individual mode TTMs, and they can be done in any order. In particular, we will be interested in the case where we multiply an N-mode tensor by matrices $\mathbf{U}_j$ in every mode except n, denoted $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{U}_1 \cdots \times_{n+1} \mathbf{U}_{n-1} \times_{n+1} \mathbf{U}_{n+1} \cdots \times_N \mathbf{U}_N$. This is expressed in the mode-n matricization as

$$\mathbf{Y}_{(n)} = \mathbf{X}_{(n)}(\mathbf{U}_N \otimes \cdots \otimes \mathbf{U}_{n+1} \otimes \mathbf{U}_{n-1} \otimes \cdots \otimes \mathbf{U}_1)^\top. \tag{2}$$

2.3 | CP-ALS algorithm

We now detail both the CP decomposition as well as the CP-ALS approach, one of the most popular algorithms used to compute the CP decomposition.

2.3.1 | CP decomposition

The aim of the CP decomposition is to represent a tensor as a sum of rank-one components. For an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, the rank-$R$ CP decomposition of $\mathcal{X}$ is the approximation

$$\mathcal{K} = \sum_{r=1}^{R} \lambda_r \, \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}, \tag{3}$$

where $\mathbf{a}_r^{(n)} \in \mathbb{R}^{I_n}$ are unit vectors with weight vector $\lambda \in \mathbb{R}^R$, and $\circ$ denotes the outer product. The collection of all $\mathbf{a}_r^{(n)}$ vectors for each mode is called a factor matrix, that is, $\mathbf{A}_n = \begin{bmatrix} \mathbf{a}_1^{(n)} & \mathbf{a}_2^{(n)} & \ldots & \mathbf{a}_r^{(n)} \end{bmatrix} \in \mathbb{R}^{I_n \times R}$. The CP decomposition of $\mathcal{X}$ can also be denoted as $\mathcal{K} = [\![\lambda; \mathbf{A}_1, \ldots, \mathbf{A}_N]\!]$. A visualization of the three-dimensional version of this representation is in Figure 1.

Using the notation $\hat{\mathbf{A}}_n = \mathbf{A}_n \cdot \mathrm{diag}(\lambda)$, we can express the mode-n matricization of $\mathcal{K}$ as

$$\mathbf{K}_{(n)} = \hat{\mathbf{A}}_n(\mathbf{A}_N \odot \cdots \odot \mathbf{A}_{n+1} \odot \mathbf{A}_{n-1} \odot \cdots \odot \mathbf{A}_1)^\top = \hat{\mathbf{A}}_n \mathbf{Z}_n^\top, \tag{4}$$

letting $\mathbf{Z}_n = \mathbf{A}_N \odot \cdots \odot \mathbf{A}_{n+1} \odot \mathbf{A}_{n-1} \odot \cdots \odot \mathbf{A}_1$.

2.3.2 | CP-ALS algorithm

In order to compute the CP-decomposition, the CP-ALS algorithm solves a least squares problem for the matricized tensors $\mathbf{X}_{(n)}$ and $\mathbf{K}_{(n)}$ along each mode. For mode n, we fix every factor matrix except $\hat{\mathbf{A}}_n$ and then solve for $\hat{\mathbf{A}}_n$. This process is repeated by alternating between modes until some termination criteria is met. We solve the linear least squares problem

$$\min_{\hat{\mathbf{A}}_n} \|\mathbf{X}_{(n)} - \hat{\mathbf{A}}_n \mathbf{Z}_n^\top\|_F, \tag{5}$$

using the representation in (4). The linear least squares problem from (5) is typically solved using the normal equations, that is,

$$\mathbf{X}_{(n)}\mathbf{Z}_n = \hat{\mathbf{A}}_n(\mathbf{Z}_n^\top \mathbf{Z}_n).$$

The coefficient matrix $\mathbf{Z}_n^\top \mathbf{Z}_n$ is computed efficiently as $\mathbf{A}_1^\top \mathbf{A}_1 * \cdots * \mathbf{A}_{n-1}^\top \mathbf{A}_{n-1} * \mathbf{A}_{n+1}^\top \mathbf{A}_{n+1} * \cdots * \mathbf{A}_N^\top \mathbf{A}_N$. We obtain the desired factor matrix $\mathbf{A}_n$ by normalizing the columns of $\hat{\mathbf{A}}_n$ and updating the weight vector λ. This approach is detailed in Algorithm 1.

Algorithm 1. CP-ALS

1: **function** $[\lambda, \{\mathbf{A}_n\}] = \text{CP-ALS}(\mathcal{X}, R)$ ▷ $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
2: Initialize factor matrices $\mathbf{A}_1, \ldots, \mathbf{A}_N$
3: Compute Gram matrices $\mathbf{G}_1 = \mathbf{A}_1^\top \mathbf{A}_1, \ldots, \mathbf{G}_N = \mathbf{A}_N^\top \mathbf{A}_N$
4: **repeat**
5: **for** $n = 1, \ldots, N$ **do**
6: $\mathbf{S}_n \leftarrow \mathbf{G}_N * \cdots * \mathbf{G}_{n+1} * \mathbf{G}_{n-1} * \cdots * \mathbf{G}_1$
7: $\mathbf{Z}_n \leftarrow \mathbf{A}_N \odot \cdots \odot \mathbf{A}_{n+1} \odot \mathbf{A}_{n-1} \odot \cdots \odot \mathbf{A}_1$
8: $\mathbf{M}_n \leftarrow \mathbf{X}_{(n)}\mathbf{Z}_n$ ▷ MTTKRP
9: Solve $\hat{\mathbf{A}}_n \mathbf{S}_n = \mathbf{M}_n$ for $\hat{\mathbf{A}}_n$ via Cholesky (CP-ALS) or SVD (CP-ALS-PINV) ▷ Normal equations
10: Normalize columns of $\hat{\mathbf{A}}_n$ to obtain $\mathbf{A}_n$ and λ
11: Recompute Gram matrix $\mathbf{G}_n = \mathbf{A}_n^\top \mathbf{A}_n$ for updated factor matrix $\mathbf{A}_n$
12: **end for**
13: **until** termination criteria met
14: **return** λ, factor matrices $\{\mathbf{A}_n\}$
15: **end function**

Note that in Line 9, the standard approach (implemented in the Tensor Toolbox[1]) is to solve $\mathbf{A}_n \mathbf{S}_n = \mathbf{M}_n$ for $\mathbf{A}_n$ using a Cholesky decomposition, which is implemented in MATLAB using the backslash operator. An alternative method for solving the linear system is to use the SVD, which is implemented in MATLAB using the `pinv` function. We call the algorithm obtained by taking this alternate approach CP-ALS-PINV.

2.4 | Gauss–Newton optimization approach

An alternate approach to ALS is to minimize the CP model "all at once" using optimization techniques. Instead of the linear least squares problem from (5), this approach solves the nonlinear least squares problem $\min \|\mathcal{X} - \mathcal{K}\|_F^2$, subject to factor matrices $\mathbf{A}_j$ with $j = 1, \ldots, N$, where $\mathcal{K}$ is defined in (3). Gauss–Newton attempts to minimize this nonlinear residual by using a linear Taylor series approximation at each iteration and minimizing that function using standard linear least squares methods. Typically, these linear least squares problems are solved via the normal equations as follows. For residual $\mathbf{r} = \text{vec}(\mathcal{X} - \mathcal{K})$ and $\mathbf{J}$ the Jacobian of $\mathbf{r}$, the normal equations to be solved involve $\mathbf{J}^\top \mathbf{J}$ and $\mathbf{J}^\top \mathbf{r}$. While CP-ALS is typically fast and easy to implement, the Gauss–Newton approach is beneficial as it can converge quadratically if the residual is small. We use the implementation of Gauss–Newton in Tensorlab[2] in the experiments in Section 4. For more details on this approach, see References 17-19.

3 | PROPOSED METHODS

3.1 | CP-ALS-QR algorithms

In our proposed CP-ALS approach, we incorporate the more stable QR decomposition and SVD and avoid using the normal equations. Suppose $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an N-mode tensor and we wish to approximate a solution $\mathcal{K} = [\![\lambda; \mathbf{A}_1, \ldots, \mathbf{A}_N]\!]$ of rank R. The key to the efficiency of our algorithm is to form the QR decomposition of $\mathbf{Z}_n$ using the Khatri–Rao

structure as follows. As a first step, we compute compact QR factorizations of each individual factor matrix, so that $\mathbf{A}_j = \mathbf{Q}_j\mathbf{R}_j$. Then

$$\begin{aligned}\mathbf{Z}_n &= \mathbf{A}_N \odot \cdots \odot \mathbf{A}_{n+1} \odot \mathbf{A}_{n-1} \odot \cdots \odot \mathbf{A}_1 \\ &= \mathbf{Q}_n\mathbf{R}_N \odot \cdots \odot \mathbf{Q}_{n+1}\mathbf{R}_{n+1} \odot \mathbf{Q}_{n-1}\mathbf{R}_{n-1} \odot \cdots \odot \mathbf{Q}_1\mathbf{R}_1 \\ &= (\mathbf{Q}_N \otimes \cdots \otimes \mathbf{Q}_{n+1} \otimes \mathbf{Q}_{n-1} \otimes \cdots \otimes \mathbf{Q}_1)\underbrace{(\mathbf{R}_N \odot \cdots \odot \mathbf{R}_{n+1} \odot \mathbf{R}_{n-1} \odot \cdots \odot \mathbf{R}_1)}_{\mathbf{V}_n},\end{aligned}$$

where the last equality comes from (1). We then compute the QR factorization of the Khatri–Rao product $\mathbf{V}_n = \mathbf{R}_N \odot \cdots \odot \mathbf{R}_{n+1} \odot \mathbf{R}_{n-1} \odot \cdots \odot \mathbf{R}_1 = \mathbf{Q}_0\mathbf{R}_0$. This allows us to express the QR of $\mathbf{Z}_n$ as

$$\begin{aligned}\mathbf{Z}_n &= (\mathbf{Q}_N \otimes \cdots \otimes \mathbf{Q}_{n+1} \otimes \mathbf{Q}_{n-1} \otimes \cdots \otimes \mathbf{Q}_1)(\mathbf{R}_N \odot \cdots \odot \mathbf{R}_{n+1} \odot \mathbf{R}_{n-1} \odot \cdots \odot \mathbf{R}_1) \\ &= \underbrace{(\mathbf{Q}_N \otimes \cdots \otimes \mathbf{Q}_{n+1} \otimes \mathbf{Q}_{n-1} \otimes \cdots \otimes \mathbf{Q}_1)\mathbf{Q}_0}_{\mathbf{Q}} \underbrace{\mathbf{R}_0}_{\mathbf{R}}.\end{aligned} \tag{6}$$

Note that $\mathbf{Q}$ has orthonormal columns, and $\mathbf{R}$ is upper triangular. The Khatri–Rao product of triangular matrices $\mathbf{V}_n$ has sparse structure. The last column is dense, but the rest have many zeros. As discussed in Section 3.3.3, this sparse structure can be exploited while implementing the QR of $\mathbf{V}_n$, but our current implementation treats it as dense.

We start solving for factor matrix $\hat{\mathbf{A}}_n$ by computing the QR of Khatri–Rao product $\mathbf{Z}_n$ as in (6). Next, we apply the product $\mathbf{Q}_N \otimes \cdots \otimes \mathbf{Q}_{n+1} \otimes \mathbf{Q}_{n-1} \otimes \cdots \otimes \mathbf{Q}_1$ to $\mathbf{X}_{(n)}$ on the right via the Multi-TTM $\mathcal{Y} = \mathcal{X} \times_1 \mathbf{Q}_1^\top \cdots \times_{n-1} \mathbf{Q}_{n-1}^\top \times_{n+1} \mathbf{Q}_{n+1}^\top \cdots \times_N \mathbf{Q}_N^\top$, following (2). After computing the Multi-TTM, we form $\mathbf{W}_n = \mathbf{Y}_{(n)}\mathbf{Q}_0$ so that we are now solving $\hat{\mathbf{A}}_n\mathbf{R}^\top = \mathbf{W}_n$. Finally, we use substitution with $\mathbf{R}^\top$ to compute the factor matrix $\hat{\mathbf{A}}_n$. We call this algorithm CP-ALS-QR.

Another more stable way of solving $\hat{\mathbf{A}}_n\mathbf{R}^\top = \mathbf{W}_n$, particularly when Khatri–Rao product $\mathbf{Z}_n$ is rank deficient, is to use the SVD of $\mathbf{R}$ in addition to the QR factorization. Let $\mathbf{R} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be the SVD. Our solution then becomes $\hat{\mathbf{A}}_n = \mathbf{W}_n\mathbf{U}\boldsymbol{\Sigma}^\dagger\mathbf{V}^\top$. Note that because we are utilizing the pseudoinverse on $\boldsymbol{\Sigma}$, we can truncate small singular values and thereby manage an ill-conditioned least-squares problem in a more stable manner. Using the SVD in this way gives us our second algorithm CP-ALS-QR-SVD.

Both CP-ALS-QR and CP-ALS-QR-SVD are summarized in Algorithm 2, with a choice in line 10 to distinguish between the two methods. The two algorithms are implemented in MATLAB using the Tensor Toolbox,[1] and are available here: https://github.com/rlminste/CP-ALS-QR.

Algorithm 2. CP-ALS-QR

1: **function** $[\lambda, \{\mathbf{A}_n\}] = \text{CP-ALS-QR}(\mathcal{X}, R)$ ▷ $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
2: Initialize factor matrices $\hat{\mathbf{A}}_2, \ldots, \hat{\mathbf{A}}_N$
3: Compute compact QR-decomposition $\mathbf{Q}_2\mathbf{R}_2, \ldots, \mathbf{Q}_N\mathbf{R}_N$ of factor matrices
4: **repeat**
5: **for** $n = 1, \ldots, N$ **do**
6: $\mathbf{V}_n \leftarrow \mathbf{R}_N \odot \cdots \odot \mathbf{R}_{n+1} \odot \mathbf{R}_{n-1} \odot \cdots \odot \mathbf{R}_1$
7: Compute compact QR-decomposition $\mathbf{V}_n = \mathbf{Q}_0\mathbf{R}$ ▷ Last step of QR decomposition
8: $\mathcal{Y} \leftarrow \mathcal{X} \times_1 \mathbf{Q}_1^\top \times_2 \cdots \times_{n-1} \mathbf{Q}_{n-1}^\top \times_{n+1} \mathbf{Q}_{n+1}^\top \times_{n+2} \cdots \times_N \mathbf{Q}_N^\top$ ▷ Multi-TTM
9: $\mathbf{W}_n \leftarrow \mathbf{Y}_{(n)}\mathbf{Q}_0$
10: Solve $\hat{\mathbf{A}}_n\mathbf{R}^\top = \mathbf{W}_n$ for $\hat{\mathbf{A}}_n$ by substitution (CP-ALS-QR) or SVD (CP-ALS-QR-SVD)
11: Normalize columns of $\hat{\mathbf{A}}_n$ to obtain $\mathbf{A}_n$ and λ
12: Recompute QR-decomposition for updated factor matrix $\mathbf{A}_n = \mathbf{Q}_n\mathbf{R}_n$
13: **end for**
14: **until** termination criteria met
15: **return** λ, factor matrices $\{\mathbf{A}_n\}$
16: **end function**

3.2 | CP-ALS-QR cost analysis

We now analyze the computational complexity of each iteration of CP-ALS-QR and CP-ALS-QR-SVD as presented in Algorithm 2. Recall that $\mathcal{X}$ has dimensions $I_1 \times \cdots \times I_N$ and the CP approximation has rank R. To simplify notation, we assume in the analysis that $I_1 \geq \cdots \geq I_N$.

The cost of forming the Khatri–Rao product $\mathbf{V}_n$ of $N-1$ upper-triangular factors (Line 6) is $R^N + \mathcal{O}(R^{N-1})$, assuming the individual Khatri–Rao products are formed pairwise and no sparsity is exploited. Here $\mathbf{V}_n$ has dimensions $R^{N-1} \times R$, and it has $R^N/N + \mathcal{O}(R^{N-1})$ nonzeros. We treat $\mathbf{V}_n$ as a dense matrix here but discuss the possibility of exploiting sparsity in Section 3.3.3. Computing the QR decomposition of $\mathbf{V}_n$ to obtain $\mathbf{Q}_0$ and $\mathbf{R}$ in Line 7 costs

$$4R^{N+1} + \mathcal{O}(R^3), \tag{7}$$

assuming $\mathbf{Q}_0$ is formed explicitly and again no sparsity is exploited.

The Multi-TTM is performed in Line 8 and involves the input tensor. We compute the resulting tensor $\mathcal{Y}$, which has dimensions $R \times \cdots \times I_n \times \cdots \times R$, by performing single TTMs in sequence; to minimize flops we perform the $N-1$ TTMs in order of decreasing tensor dimension, which is left to right given our assumption above. The cost of the first TTM is $2I_1 \cdots I_N R$, the cost of the second is $2I_2 \cdots I_N R^2$, and so on. Thus, we can write the overall cost of the Multi-TTM as

$$2I_1 \cdots I_N R \left(1 + \frac{R}{I_1} + \frac{R^2}{I_1 I_2} + \cdots + \frac{R^{N-2}}{I_1 \cdots I_{n-1} I_{n+1} \cdots I_{N-1}}\right). \tag{8}$$

We apply $\mathbf{Q}_0$ to $\mathbf{Y}_{(n)}$ via matrix multiplication in Line 9 with cost $2I_n R^N$, assuming we use an explicit, dense $\mathbf{Q}_0$. Solving the linear system in Line 10 costs $\mathcal{O}(I_n R^2)$, with an extra $\mathcal{O}(R^3)$ cost if the SVD of $\mathbf{R}$ is computed for the CP-ALS-QR-SVD method. (Note that using the more stable SVD approach to solve the linear system has no significant impact on the overall computational complexity.) Finally, computing the QR decomposition of the updated nth factor matrix in Line 12 and forming the orthonormal factor $\mathbf{Q}_n$ explicitly costs $4I_n R^2 + \mathcal{O}(R^3)$. These costs are all summarized in Table 1.

If the rank R is significantly smaller than the tensor dimensions, then the cost is dominated by the first TTM, which has cost $2I_1 \cdots I_N R$ from (8). In this case, the remaining TTMs are each at least a factor of R/I_1 times cheaper, and the computations involving $\mathbf{Q}_0$ are cheaper than any of the TTMs, because those computational costs are independent of any tensor dimensions. The dominant cost of CP-ALS is the MTTKRP in Line 8 of Algorithm 1, which also has cost $2I_1 \cdots I_N R$. Thus, in the case of small R, the two algorithms have identical leading-order computational complexity per iteration.

If the rank R is larger than all tensor dimensions, then the cost of the QR of $\mathbf{V}_n$ given in (7) will be the dominant cost. If the rank is comparable to the tensor dimensions, then the computation and application of $\mathbf{Q}_0$ in Line 9 and the subsequent TTMs after the first may also contribute to the running time in a significant way.

TABLE 1 Breakdown of main components in each subiteration of CP-ALS, CP-ALS-PINV, CP-ALS-QR, and CP-ALS-QR-SVD.

CP-ALS, CP-ALS-PINV		CP-ALS-QR, CP-ALS-QR-SVD	
Component	**Cost**	**Component**	**Cost**
MTTKRP	$2I^N R + \mathcal{O}(I^{N-1}R)$	Multi-TTM	$2I^N R + \mathcal{O}(I^{N-1}R^2)$
Gram of factor matrices	IR^2	QR of factor matrices	$4IR^2$
N/A		Computing $\mathbf{Q}_0$	$4R^{N+1}$
N/A		Applying $\mathbf{Q}_0$	$2IR^N$

Note: We use this breakdown in our performance experiments shown in Figure 2.

Abbreviations: ALS, alternating least squares; CP, canonical polyadic; MTTKRP, matricized-tensor times Khatri-Rao product; SVD, singular value decomposition.

3.3 | Implementation details and extensions

3.3.1 | Efficient computation of approximation error

For each of the CP algorithms (Algorithms 1 and 2), we consider computing the approximation error in two ways. The more accurate but less efficient approach is to form the explicit representation of $\mathcal{K} = [\![\lambda; \mathbf{A}_1, \ldots, \mathbf{A}_N]\!]$ and compute the residual norm $\|\mathcal{X} - \mathcal{K}\|$ directly. The less accurate but more efficient approach exploits the identity $\|\mathcal{X} - \mathcal{K}\|^2 = \|\mathcal{X}\|^2 - 2\langle \mathcal{X}, \mathcal{K}\rangle + \|\mathcal{K}\|^2$ and computes $\langle \mathcal{X}, \mathcal{K}\rangle$ and $\|\mathcal{K}\|$ cheaply by using temporary quantities already computed by the ALS iterations ($\|\mathcal{X}\|$ is precomputed and does not change over iterations).

In the case of CP-ALS (Algorithm 1), we have $\mathbf{K}_{(N)} = \hat{\mathbf{A}}_N \mathbf{Z}_N^\top$, for $\hat{\mathbf{A}}_N = \mathbf{A}_N \cdot \text{diag}(\lambda)$ and $\mathbf{Z}_N = \mathbf{A}_{N-1} \odot \cdots \odot \mathbf{A}_1$. Then

$$\langle \mathcal{X}, \mathcal{K}\rangle = \langle \mathbf{X}_{(N)}, \hat{\mathbf{A}}_N \mathbf{Z}_N^\top\rangle = \langle \mathbf{X}_{(N)} \mathbf{Z}_N, \hat{\mathbf{A}}_N\rangle = \langle \mathbf{M}_N, \hat{\mathbf{A}}_N\rangle,$$

where $\mathbf{M}_N$ is the result of the MTTKRP computation in mode N, the mode of the last subiteration. Thus, computing the inner product between the data and model tensors requires only $\mathcal{O}(I_N R)$ extra operations. Likewise, we have

$$\|\mathcal{K}\|^2 = \langle \hat{\mathbf{A}}_N \mathbf{Z}_N^\top, \hat{\mathbf{A}}_N \mathbf{Z}_N^\top\rangle = \langle \mathbf{Z}_N^\top \mathbf{Z}_N, \hat{\mathbf{A}}_N^\top \hat{\mathbf{A}}_N\rangle = \langle \mathbf{S}_N, \text{diag}(\lambda) \mathbf{G}_N \text{diag}(\lambda)\rangle,$$

where $\mathbf{G}_N = \mathbf{A}_N^\top \mathbf{A}_N$ is the Gram matrix of the (normalized) Nth factor and $\mathbf{S}_N$ is the Hadamard product of the Gram matrices of the first $N-1$ modes. Computing the norm of $\mathcal{K}$ thus requires only $\mathcal{O}(R^2)$ extra operations. This efficient error computation is well known.[1,20-23] Note that this approach is slightly less accurate than a direct computation: the identity applies to the square of the residual norm, so taking the square root of the difference of these quantities limits the accuracy of the relative error to the square root of machine precision.

We complete the efficient error computation for CP-ALS-QR (Algorithm 2) with similar cost. In this case, we have $\mathbf{K}_{(N)} = \hat{\mathbf{A}}_N \mathbf{Z}_N^\top$ with $\mathbf{Z}_N = (\mathbf{Q}_{N-1} \otimes \cdots \otimes \mathbf{Q}_1)\mathbf{Q}_0 \mathbf{R}$. Then

$$\langle \mathcal{X}, \mathcal{K}\rangle = \langle \mathbf{X}_{(N)}, \hat{\mathbf{A}}_N \mathbf{Z}_N^\top\rangle = \langle \mathbf{X}_{(N)}(\mathbf{Q}_{N-1} \otimes \cdots \otimes \mathbf{Q}_1)\mathbf{Q}_0, \hat{\mathbf{A}}_N \mathbf{R}^\top\rangle = \langle \mathbf{W}_N, \hat{\mathbf{A}}_N \mathbf{R}^\top\rangle,$$

where in Algorithm 2, $\mathbf{W}_N$ is the result of the Multi-TTM (except in mode N) and the multiplication with $\mathbf{Q}_0$, and thus has dimension $I_N \times R$. Thus, the cost of this computation is dominated by that of computing $\hat{\mathbf{A}}_N \mathbf{R}^\top$, or $\mathcal{O}(I_N R^2)$. We also have

$$\|\mathcal{K}\|^2 = \langle \hat{\mathbf{A}}_N \mathbf{Z}_N^\top, \hat{\mathbf{A}}_N \mathbf{Z}_N^\top\rangle = \langle \mathbf{Z}_N^\top \mathbf{Z}_N, \hat{\mathbf{A}}_N^\top \hat{\mathbf{A}}_N\rangle = \langle \mathbf{R}^\top \mathbf{R}, \text{diag}(\lambda) \mathbf{R}_N^\top \mathbf{R}_N \text{diag}(\lambda)\rangle,$$

where $\mathbf{R}_N$ is the triangular factor in the QR decomposition of $\mathbf{A}_N$. The cost of this extra computation is $\mathcal{O}(R^3)$, and thus the overall cost is within a factor of R of the efficient error computation of CP-ALS.

3.3.2 | Kruskal tensor input

The analysis of both CP-ALS-QR and CP-ALS-QR-SVD, as explained in Section 3.2, assume the input tensor is a dense tensor. When the input tensor has special structure, the key operations can be computed more efficiently. We also implemented a version of each algorithm that exploits inputs with Kruskal structure, that is, a tensor stored as factor matrices and corresponding weights, which we use for the input in Section 4.3. Exploiting this structure is beneficial as we avoid forming the input tensor or Multi-TTM product tensor $\mathcal{Y}$, because all computations can be performed using the factor matrices instead.

Note that the Tensor Toolbox has optimized the MTTRKP (Algorithm 1, Line 8) and Multi-TTM (Algorithm 2, Line 8) computations for a Kruskal tensor input.[24] For an N-mode Kruskal tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ of rank R and N matrices $\mathbf{V}_j \in \mathbb{R}^{I_j \times S}$ for $j = 1, \ldots, N$, the MTTKRP

$$\mathbf{X}_{(n)}(\mathbf{V}_N \odot \cdots \odot \mathbf{V}_{n+1} \odot \mathbf{V}_{n-1} \odot \cdots \odot \mathbf{V}_1),$$

has cost $\mathcal{O}(RS\sum_{j=1}^{N} I_j)$. This is an improvement compared to the cost of the MTTKRP in the dense case, which is on the order of the product of the N dimensions instead of their sum. To compute a Multi-TTM product of the same Kruskal tensor $\mathcal{X}$ and matrices $\mathbf{V}_j$ with the same dimensions as before, that is,

$$\mathcal{Y} = \mathcal{X} \times_1 \mathbf{V}_1^\top \times \cdots \times_{n-1} \mathbf{V}_{n-1}^\top \times_{n+1} \mathbf{V}_{n+1}^\top \times \cdots \times_N \mathbf{V}_N^\top,$$

the complexity becomes $\mathcal{O}(RS \sum_{j\neq n} I_j)$, assuming $\mathcal{Y}$ is maintained in Kruskal format.

The Kruskal structure has the added benefit of reducing the cost of computing the product $\mathbf{W}_n = \mathbf{Y}_{(n)}\mathbf{Q}_0$ in our QR-based algorithms (see Line 9 in Algorithm 2), as we do not need to explicitly matricize $\mathcal{Y}$. Let us consider the first mode as an example. In this case, $\mathcal{Y}$ is an N-dimensional Kruskal tensor with rank R and $\mathcal{Y} = [\![\lambda; \mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_N]\!]$, with $\mathbf{B}_1 \in \mathbb{R}^{I_1 \times R}$, $\mathbf{B}_j \in \mathbb{R}^{R\times R}$ for $j = 2, \ldots, N$, and $\mathbf{Q}_0 \in \mathbb{R}^{R^{N-1}\times R}$. Then, we can write $\mathbf{Y}_{(1)} = \mathbf{B}_1(\mathbf{B}_N \odot \cdots \odot \mathbf{B}_2)^\top$, treat $\mathbf{Q}_0$ as a matricized tensor and compute the $\mathbf{W} = \mathbf{B}_1[(\mathbf{B}_N \odot \cdots \odot \mathbf{B}_2)^\top \mathbf{Q}_0]$ using an MTTKRP followed by a small matrix product. This gives us a total cost of $2(I_1R^2 + R^{N+1})$.

Note that the cost of these QR-based methods is exponential in N, the number of modes. This cost arises in computing and applying $\mathbf{Q}_0$, which is represented explicitly. The costs of CP-ALS methods that use the normal equations are linear in N when applied to inputs represented as Kruskal tensors. Thus, achieving better numerical stability using QR-based methods comes at substantial computational price as N increases.

The matrix $\mathbf{Q}_0$, which is the orthogonal factor of a QR decomposition of a Khatri–Rao product, has internal low-rank structure. Consider, for example, the QR decomposition $\mathbf{QR} = \mathbf{A} \odot \mathbf{B} \odot \mathbf{C}$. Mathematically, we have $\mathbf{Q} = (\mathbf{A} \odot \mathbf{B} \odot \mathbf{C})\mathbf{R}^{-1}$, assuming the matrix is full rank. Each column of $\mathbf{A} \odot \mathbf{B} \odot \mathbf{C}$ is a Kronecker product of three vectors, which is equivalent to a vectorization of an order-three, rank-one tensor. Because $\mathbf{R}^{-1}$ is an upper triangular matrix, this implies that the kth column of $\mathbf{Q}$ is a linear combination of k Kronecker products, which is a vectorization of an order-three, rank-k Kruskal tensor. While it is possible to compute the QR decomposition using a Gram–Schmidt process or the Householder QR algorithm and maintain the internal low rank structure (the CP ranks of the tensorized Householder vectors are twice as large as the corresponding columns of $\mathbf{Q}$), we are unaware of an algorithm that also maintains the numerical stability properties of an explicit QR decomposition.

3.3.3 | Other computation-reducing optimizations

As noted in Section 3.2, the Khatri–Rao product of triangular matrices $\mathbf{V}_n$ computed in Algorithm 2 is a sparse matrix with density proportional to $1/N$, where N is the number of modes. This is because the ith column of $\mathbf{V}_n$ is a Kronecker product of $N-1$ vectors each with i nonzeros and therefore has i^{N-1} nonzeros. This sparsity could be exploited in the computation of $\mathbf{V}_n$ (Line 6), computing its QR decomposition (Line 7), and applying its orthonormal factor $\mathbf{Q}_0$ (Line 9). In particular, when using a sparse QR decomposition algorithm, there will be no fill-in (zero entries becoming nonzeros), as every row is dense to the right of its first nonzero, and the number of flops required is a factor of $\mathcal{O}(1/N^2)$ times that of the dense QR algorithm. The computational savings in computing $\mathbf{V}_n$ and applying $\mathbf{Q}_0$ is $\mathcal{O}(1/N)$. However, the use of (general) sparse computational kernels comes at a price of performance, so for small N we do not expect much reduction in time and did not exploit sparsity in our implementation.

An important optimization for CP-ALS (Algorithm 1) is to avoid recomputation across MTTKRPs of the different modes. For example, the Khatri–Rao products $\mathbf{Z}_n$ and $\mathbf{Z}_{n+1}$ share $N-2$ different factors, so the computations of $\mathbf{M}_n$ and $\mathbf{M}_{n+1}$ have significant overlap. The general approach to avoid this recomputation via memorization is known as dimension trees, as a tree of temporary matrices can be computed, stored, and re-used for the MTTKRPs across modes.[6,20] Using dimension trees reduces the outer-iteration CP-ALS cost from N MTTKRPs to the cost of two MTTKRPs.

Similar savings can be obtained by applying dimension trees to the set of Multi-TTM operations in CP-ALS-QR (Algorithm 2). In this case, we exploit the overlap in individual TTMs across modes and store a different set of intermediate tensors that can be re-used across modes. Dimension trees have been used for Multi-TTM before in the context of Tucker decompositions for sparse tensors and the Higher-Order Orthogonal Iteration algorithm.[10,25] As in the case of CP-ALS, dimension trees can reduce the outer-iteration CP-ALS-QR cost from N TTMs involving the data tensor to two TTMs. Neither of these reductions come at the expense of lower performance, so we can expect $\mathcal{O}(N)$ speedup in each case. For CP-ALS-QR, there are other overlapping computations that can be similarly exploited. For example, the QR decomposition of $\mathbf{V}_n$ can be performed using a tree across the Khatri–Rao factors, some of which are shared across

modes, though the structure of the orthonormal factor would need to be maintained when applying it. Because the Tensor Toolbox implementation of CP-ALS does not employ dimension trees, for fair comparison, we do not use them for CP-ALS-QR either.

4 | NUMERICAL EXPERIMENTS

In this section, we explore several examples that demonstrate the benefits of CP-ALS-QR and CP-ALS-QR-SVD over the typical ALS approaches. Specifically, we will demonstrate the performance of our algorithms as well as show the stability of our algorithms by considering ill-conditioned problems.

4.1 | Performance results

As seen in our analysis in Section 3.2, the dominant cost for our new algorithms is the same as for CP-ALS and CP-ALS-PINV when R is small. For large R, we see that the lower-order terms for CP-ALS-QR and CP-ALS-QR-SVD do have an effect on the runtime. We verify the comparable runtimes for small R and examine the slowdown for large R with a few experiments here.

We break each algorithm down to its key components and time each individually. These components are listed in Table 1. We also include the computational complexity of the dominant parts of each algorithm for clarity, assuming an N-way tensor with the same dimension in each mode $\mathcal{X} \in \mathbb{R}^{I \times \cdots \times I}$. Each row of the table represents corresponding parts of the two different types of algorithm.

For CP-ALS and CP-ALS-PINV (Algorithm 1), the MTTKRP refers to forming $\mathbf{M}_n = \mathbf{X}_{(n)}\mathbf{Z}_n$, see Line 8, while we compute the Gram matrices for each factor matrix in Line 3. For CP-ALS-QR and CP-ALS-QR-SVD (Algorithm 2), the Multi-TTM is computed when applying the Kronecker product of $\mathbf{Q}_j$ matrices to $\mathbf{X}_{(n)}$, see Line 8, and we compute the QR factorization of each factor matrix in Line 3. Computing $\mathbf{Q}_0$ involves computing a QR factorization of Khatri–Rao product $\mathbf{V}_n$, see Line 7, and applying $\mathbf{Q}_0$ is a matrix multiplication, see Line 9. Other steps not explicitly listed include solving (by substitution or SVD), finding the weight vector $\boldsymbol{\lambda}$, and computing the error. These steps are combined into the "Other" category in Figure 2, as none represent a significant portion of the runtime for any of the four algorithms.

The three tensors we test are randomly generated cubical tensors of three, four, and five modes. The three-way tensor has dimension 700, the four-way tensor has dimension 300, and the five-way tensor has dimension 75. We computed the average iteration time over 10 iterations (omitting the first iteration to ensure a warm cache). The tolerance we used for all algorithms was 10^{-10}, and we computed the error in the efficient manner described in Section 3.3.1. The results for these three tensors with increasing rank values are in Figure 2. For each rank value, we also report the slowdown ratio we see between the overall runtimes of CP-ALS-QR and CP-ALS.

For all three tensors, the dominant cost per iteration is the MTTKRP for CP-ALS and CP-ALS-PINV and the Multi-TTM for CP-ALS-QR and CP-ALS-QR-SVD. Only for high ranks do other costs, computing and applying the QR of the Khatri–Rao product, even appear visibly in the plot. In the three-way case, all the slowdown ratios are close to 1×. This is similar for low ranks in four and five modes, but the ratio for high ranks jumps up to 5×, with costs for computing and applying $\mathbf{Q}_0$ appearing in this case. These results demonstrate that the slowdown incurred by using our QR-based algorithms is not significant when the CP rank is small.

4.2 | Ill-conditioned factor matrices

In this example, we test our algorithms on synthetic tensors constructed so that the factor matrices (and Khatri–Rao products of factor matrices) are ill-conditioned. We create this tensor from randomly generated factor matrices and weights so that we are able to compare the results of our algorithms to the true solution, and we add Gaussian noise. The randomly generated factor matrices are constructed so that we can control their condition number in the following way. Each factor matrix is computed as $\mathbf{A}_j = \mathbf{U}_j\boldsymbol{\Sigma}\mathbf{V}^\top$, where $\mathbf{U}_j \in \mathbb{R}^{I_j \times R}$ is a random matrix with orthonormal columns, $\boldsymbol{\Sigma} \in \mathbb{R}^{R \times R}$ consists of geometrically decreasing singular values so that the condition number of $\mathbf{A}_j$ is some predetermined κ, and $\mathbf{V} \in \mathbb{R}^{R \times R}$ is an orthogonal matrix shared between all factor matrices $\mathbf{A}_j$ for $j = 1, \ldots, N$. Constructing our factor matrices in this manner ensures that we can control their condition number κ as well as the condition number of the Khatri–Rao product of $N - 1$ factor matrices.

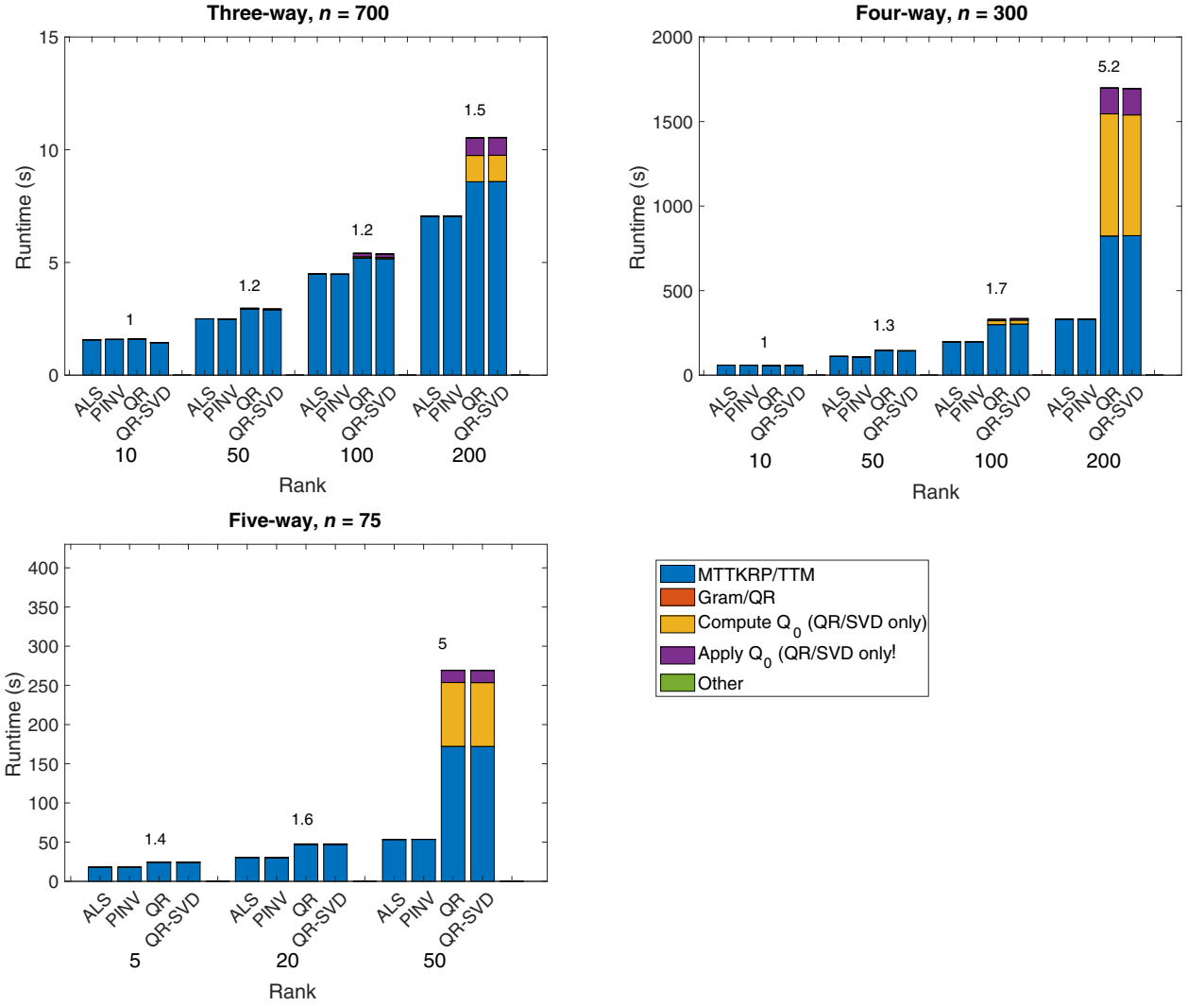
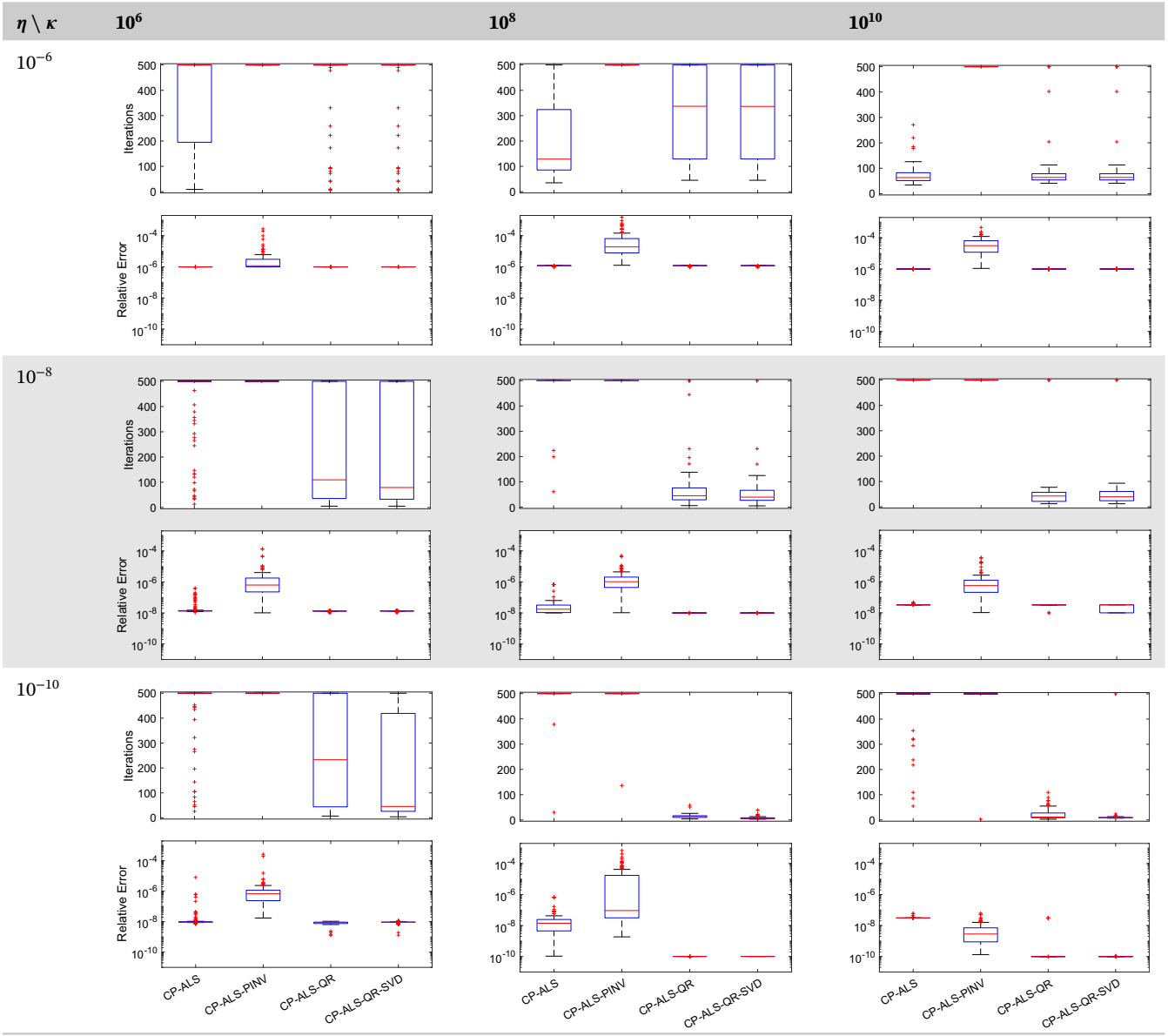


FIGURE 2 Average runtime in seconds of a single iteration for CP-ALS, CP-ALS-PINV, CP-ALS-QR, and CP-ALS-QR-SVD for a three-way tensor of size 700 (top left), a four-way tensor of size 300 (top right), and a five-way tensor of size 75 (bottom left). Results are plotted for increasing rank values, and the slowdown ratio between the runtimes of CP-ALS-QR and CP-ALS is plotted above the group of results for each rank.

For our experiments, we construct a three-way $50 \times 50 \times 50$ tensor with rank 4 and varying condition numbers κ and noise levels η. We compare our QR-based algorithms, CP-ALS-QR and CP-ALS-QR-SVD, to CP-ALS and CP-ALS-PINV. We test all combinations of three different condition numbers $\kappa = 10^6, 10^8, 10^{10}$ and noise levels $\eta = 10^{-6}, 10^{-8}, 10^{-10}$. This results in different levels of ill-conditioning; the most ill-conditioned problems are where the condition numbers are high and the noise is low. For each configuration, we run 100 trials of each algorithm to approximate a rank-4 CP factorization. The maximum number of iterations is 500 and the convergence tolerance for change in the relative error is 10^{-15}. We use a tight tolerance to ensure the computed metrics reflect what is attainable by the algorithm and not an artifact of early convergence. A random guess is used for each initialization, and each algorithm is configured with the same initial factor matrices. We also compared all four ALS algorithms to an optimization-based CP method using Gauss-Newton implemented in Tensorlab[2] that we describe in Section 2.4, but for the condition numbers we show, this method had such a high relative error that we do not include the results here. It is worth noting that for lower κ values, the Gauss–Newton method performed well along with all four ALS algorithms, with faster convergence for small η values. We anticipate that the Gauss–Newton algorithm fails for higher condition numbers ($\kappa > 10^4$) because the Jacobian inherits the high condition numbers of the Khatri–Rao products and the implementation solves the associated normal equations.

We present boxplots of the number of iterations and relative error over 100 trials of each algorithm in Table 2, where we see that the combination of noise and high condition numbers affects the ill-conditioning of the problem in different ways.

TABLE 2 Number of iterations and relative error boxplots for CP-ALS, CP-ALS-PINV, CP-ALS-QR, and CP-ALS-QR-SVD on a $50 \times 50 \times 50$ synthetic tensor of rank 4 with three different condition numbers κ for the true factor matrices and three different levels of Gaussian noise η added.



Abbreviations: ALS, alternating least squares; CP, canonical polyadic; MTTKRP, matricized-tensor times Khatri-Rao product; SVD, singular value decomposition.

Overall, we observe that the number of iterations needed for convergence for the QR-based algorithms is significantly less than either CP-ALS or CP-ALS-PINV, which reach the maximum number of iterations in almost every case. For all nine cases, the median number of iterations for both CP-ALS and CP-ALS-PINV is 500, compared to 58 for CP-ALS-QR, and 51 for CP-ALS-QR-SVD, resulting in speedups of 8.6× and 9.8×, respectively. CP-ALS does converge more quickly in the first row ($\eta = 10^{-6}$), and has several outliers that converge more quickly in the first column ($\kappa = 10^{6}$), corresponding to the more well-conditioned problems. CP-ALS-PINV, on the other hand, almost never converges in fewer than 500 iterations. The gap between number of iterations in the standard ALS algorithms compared to our QR-based algorithms increases with smaller η values (moving down the table).

Concerning the relative error, CP-ALS-PINV has the highest relative error of any algorithm by several orders of magnitude except in the bottom right case ($\eta = 10^{-10}, \kappa = 10^{10}$), which corresponds to its lack of convergence as seen in the iterations plots. Both CP-ALS-QR and CP-ALS-QR-SVD consistently attain low relative error in each case. CP-ALS sometimes attains that same low relative error, though at the cost of slower convergence. In the most ill-conditioned cases

($\eta = 10^{-8}$ to 10^{-10} and $\kappa = 10^{8}$ to 10^{10}), CP-ALS frequently has higher relative error than either QR-based algorithm by several orders of magnitude, and has higher relative error than CP-ALS-PINV in the bottom right case.

Overall, we see that in ill-conditioned cases, CP-ALS-QR and CP-ALS-QR-SVD perform more stably than either CP-ALS or CP-ALS-PINV, attaining a lower relative error in fewer iterations.

4.3 | Sine of sums

We further test the stability of our QR-based algorithms on a function approximation problem considered by Beylkin and Mohlenkamp.[26] The sine of sums $\sin(x_1 + \cdots + x_N)$ is an example of an N-dimensional function that can be approximated in such a way where complexity grows linearly with N instead of exponentially. These efficient approximations are sometimes referred to as separated representations, and they are closely related to CP decompositions. In this case, we are simulating the numerical discovery of an efficient separated representation, as we already know it exists. That is, given a separated representation with an exponentially large rank as input, we seek to compute a lower-rank (linear) representation that approximates it to numerical precision. We consider this problem as it can lead to ill-conditioned least squares problems within CP-ALS.

4.3.1 | Setup

The multivariate sine of sums function $\sin(x_1 + \cdots + x_N)$ can be discretized as a dense N-mode tensor $\mathcal{X} \in \mathbb{R}^{n \times \cdots \times n}$ with $x_j \in \mathbb{R}^n$ as vectors discretizing the interval $[0, 2\pi)$ for $j = 1, \ldots, N$. As the sine of sums can be expressed as a sum of 2^{N-1} terms using sum identities for sine and cosine, we can expand all terms to obtain a rank-2^{N-1} representation of $\mathcal{X}$, which corresponds to an exact CP decomposition. Another exact CP representation of rank N also exists, of the form

$$\sin\left(\sum_{j=1}^{N} x_j\right) = \sum_{j=1}^{N} \sin(x_j) \prod_{k=1, k\neq j}^{N} \frac{\sin(x_k + \alpha_k - \alpha_j)}{\sin(\alpha_k - \alpha_j)},$$

where α_j must satisfy $\sin(\alpha_k - \alpha_j) \neq 0$ for all $j \neq k$. This rank-N representation is nonunique, and can be numerically unstable depending on the choice of α_j.

The input representations are already rank-2^{N-1} Kruskal tensors, and we exploit that structure in our algorithms as discussed in Section 3.3.2. In the experiments below, we vary the number of modes N, corresponding to the number of variables in the sine of sums function, and the dimension n of each mode, corresponding to the number of discretization points in the interval $[0, 2\pi)$. In each experiment, we consider the relative error of four algorithms, CP-ALS, CP-ALS-PINV, CP-ALS-QR, and CP-ALS-QR-SVD, at the end of each of the first 40 iterations. We use the same random initial guesses for the factor matrices across algorithms and a convergence tolerance of 0. We also compute the relative error $\|\mathcal{X} - \mathcal{K}\|/\|\mathcal{X}\|$ directly (as opposed to the methods described in Section 3.3.1) as we need a more accurate computation of the error to truly compare the accuracy of our algorithms.

We examine three tensors of different number of modes N and dimensions n (or size of vectors x_j). In Figure 3, we plot the relative error at the end of each iteration for all four ALS algorithms for the sine of sums tensor with four modes and $n = 128$, the seven-way sine of sums tensor with $n = 16$, and the 10-mode tensor with $n = 8$. For all four plots, CP-ALS-QR and CP-ALS-QR-SVD are able to achieve a lower relative error than CP-ALS and CP-ALS-PINV, and the gap increases with N, more significantly for the $N = 7$ and $N = 10$ cases. For the $N = 10$ tensor, we use $n = 8$ for both cases, but use two different random initializations to show two different types of results we obtain. For the first random initialization (left), we see similar results to the lower-order cases, with all four algorithms converging, but the gap between the relative error for the QR-based algorithms and the normal equations-based algorithms is much larger than in lower-order tensors. With the second random initialization (right), CP-ALS and CP-ALS-PINV do not converge to anything, while the relative error for CP-ALS-QR and CP-ALS-QR-SVD converge normally to low values. When repeating this experiment for multiple random initializations, we found that this second case was more common, occurring in four out of five trials.

From these experiments, we can see that for ill-conditioned problems, our QR-based algorithms are more stable than the typical algorithms in higher dimensions. For all dimensions, we are also able to attain higher accuracy than traditional ALS algorithms.

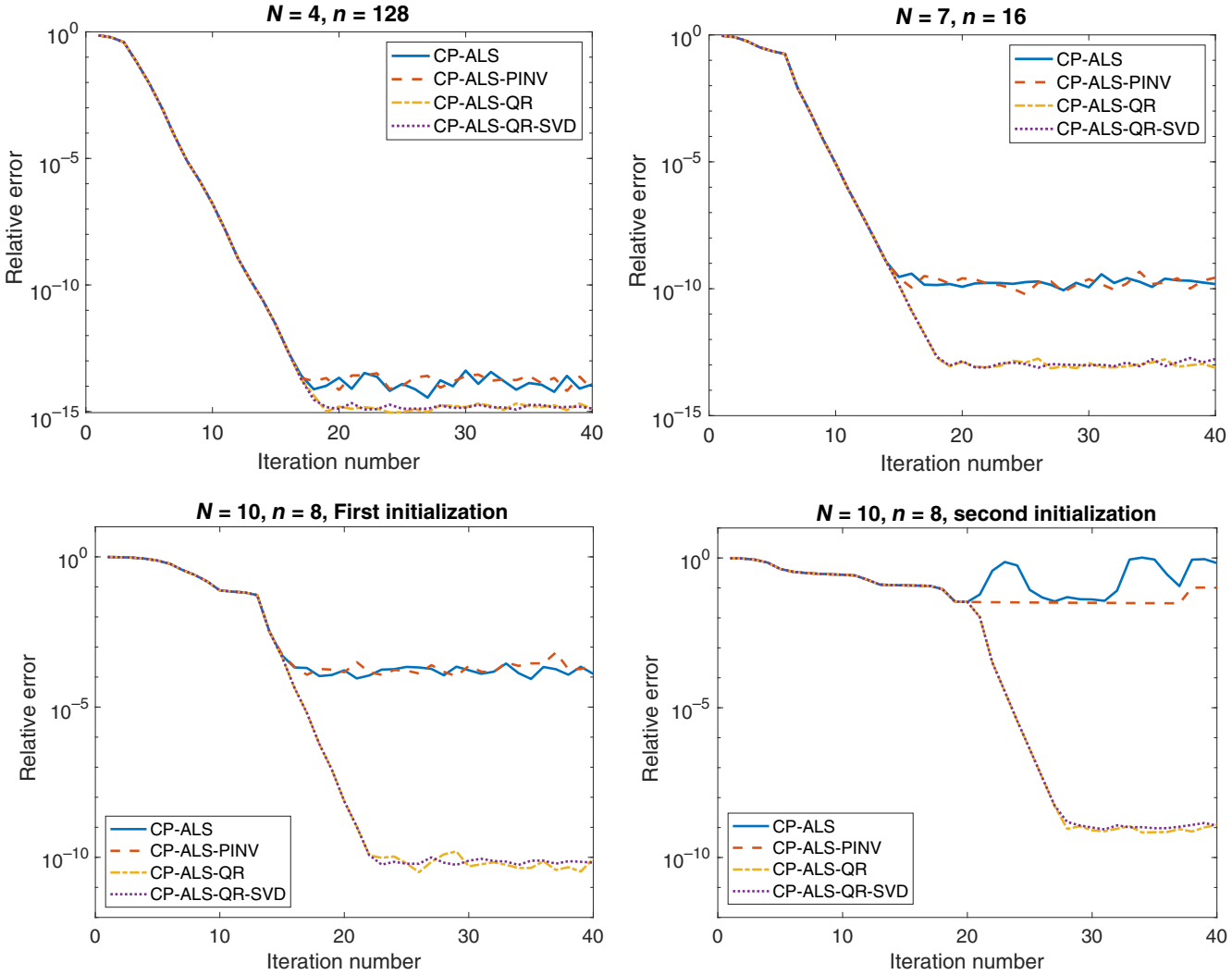


FIGURE 3 Relative error of CP-ALS, CP-ALS-PINV, CP-ALS-QR, and CP-ALS-QR-SVD on the sine of sums tensor at the end of each iteration for four modes with dimension $n = 128$ (top left), and seven modes with dimension $n = 16$ (top right), and two different random initializations for ten modes with dimension $n = 8$. The second initialization case was more common, occurring in four of five trials.

4.4 | Fast matrix multiplication algorithms

Fast matrix multiplication algorithms are algorithms that perform $o(n^3)$ operations to multiply $n \times n$ matrices. Strassen's original algorithm[27] is the most famous fast algorithm, with complexity $\mathcal{O}(n^{2.81})$. There are many such algorithms, and the optimal complexity is an open problem. Each fast matrix multiplication algorithm corresponds to an exact CP decomposition of a three-way tensor that encodes matrix multiplication of matrices of fixed dimensions. The matrix multiplication tensor corresponding to multiplying $m \times k$ and $k \times n$ matrices has dimensions $mk \times nk \times mn$. Computing CP decompositions of these tensors has been a subject of research for many decades.[28-31]

Bini, Capovani, Romani, and Lotti[32] introduced the first Arbitrary Precision Approximating (APA) algorithm for matrix multiplication, which is designed for multiplying 3×2 and 2×2 matrices. An APA algorithm corresponds to a parametrized CP approximation of a matrix multiplication tensor that can be tuned to achieve arbitrarily small approximation error, but it is not an exact decomposition. APA algorithms can be much more efficient than fast algorithms that correspond to exact decompositions, and they can be converted to exact algorithms (assuming exact arithmetic) with only logarithmic overhead. Numerical methods for computing such CP approximations must account for the ill-conditioning that occurs, so we consider the smallest such example to compare QR-based ALS methods with those that use the normal equations.

TABLE 3 Relative error and the number of iterations to converge for the matrix multiplication tensor.

Method	CP-ALS		CP-ALS-PINV		CP-ALS-QR		CP-ALS-QR-SVD	
ϵ	Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations
10^{-4}	5×10^{-5}	3	5×10^{-5}	7	5×10^{-5}	2	5×10^{-5}	2
10^{-5}	5×10^{-6}	5000	7×10^{-6}	5000	5×10^{-6}	2	5×10^{-6}	2
10^{-6}	3×10^{-4}	5000	4×10^{-4}	5000	5×10^{-7}	2	5×10^{-7}	2
10^{-7}	4×10^{-1}	5000	6×10^{-3}	5000	5×10^{-8}	110	5×10^{-8}	174

Abbreviations: ALS, alternating least squares; CP, canonical polyadic; MTTKRP, matricized-tensor times Khatri-Rao product; SVD, singular value decomposition.

We compare the relative error and number of iterations to converge of CP-ALS, CP-ALS-PINV, CP-ALS-QR, and CP-ALS-QR-SVD on the matrix multiplication tensor corresponding to multiplying 3×2 and 2×2 matrices. All algorithms perform similarly until the model approaches a solution that corresponds to an APA algorithm. Once close to a true APA solution, the normal equations are either unable to converge, or they converge to an incorrect solution. To further examine this behavior, we initialize our algorithms with a set of factor matrices close to a true solution. This solution has factor matrices that depend on a parameter ϵ and its reciprocal[32]:

$$\mathbf{A}=\begin{bmatrix}1&0&1&0&1&0&0&0&0&0\\0&0&0&\epsilon&\epsilon&0&0&0&0&0\\0&0&0&0&0&1&0&1&0&1\\1&1&0&1&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&\epsilon&\epsilon\\0&0&0&0&0&1&1&0&1&0\end{bmatrix},\quad \mathbf{B}=\begin{bmatrix}\epsilon&0&0&-\epsilon&0&1&1&-1&1&0\\0&0&0&0&\epsilon&0&0&-1&0&1\\0&-1&0&1&0&0&0&0&\epsilon&0\\1&-1&1&0&1&\epsilon&0&0&0&-\epsilon\end{bmatrix},$$

$$\mathbf{C}=\begin{bmatrix}\epsilon^{-1}&\epsilon^{-1}&-\epsilon^{-1}&\epsilon^{-1}&0&0&0&0&0&0\\0&0&-\epsilon^{-1}&0&\epsilon^{-1}&0&0&0&0&0\\0&0&0&1&0&1&0&0&-1&0\\1&0&0&0&-1&0&0&0&0&1\\0&0&0&0&0&0&-\epsilon^{-1}&0&\epsilon^{-1}&0\\0&0&0&0&0&\epsilon^{-1}&-\epsilon^{-1}&\epsilon^{-1}&0&\epsilon^{-1}\end{bmatrix}.$$

As $\epsilon\to0$, the approximation error decreases but the ill-conditioning increases. We depict the average relative error and iterations to converge for different ϵ values in Table 3, using a maximum of 5000 iterations and a tolerance of 10^{-12}. For the largest ϵ value we consider, $\epsilon=10^{-4}$, all algorithms converge to equivalent relative errors in a similarly low number of iterations. CP-ALS-QR and CP-ALS-QR-SVD converge the fastest in only two iterations. For smaller ϵ values, neither CP-ALS nor CP-ALS-PINV converge, while the QR-based algorithms converge in a small number of iterations, as few as two. Additionally, our QR-based algorithms converge to a more accurate solution for the smaller ϵ values, achieving higher accuracy of up to seven orders of magnitude compared to CP-ALS.

5 | CONCLUSIONS AND FUTURE WORK

We develop and implement versions of the CP-ALS algorithm using the QR decomposition and SVD in an effort to address the numerical ill-conditioning to which the normal equations in the traditional algorithm are susceptible. The first version uses a QR factorization to solve the linear least squares problems within CP-ALS. We also present the CP-ALS-QR-SVD algorithm, which applies the SVD as an extra step in the algorithm to handle numerically rank-deficient problems. In addition to the algorithms themselves, we provide analysis of their complexity, which is comparable to that of the widely-used CP-ALS algorithm when the rank is small. Our new algorithms prove useful for computing CP tensor decompositions with

more stability in the event of ill-conditioned coefficient matrices, and present an alternative when analyzing tensor data for which the CP-ALS algorithm produces dissatisfactory results, or is unable to produce any result due to ill-conditioning. These situations, shown through our numerical experiments, occur when we are looking for an exact decomposition or where ill-conditioning is not hidden by noise. We envision our QR-based algorithms being used as part of a robust CP-ALS solver that uses the traditional normal equations approach by default, but solves the least squares problems using QR if any ill-conditioning is detected. We also anticipate that this approach can be applied to other methods computing CP decompositions to improve accuracy, such as the error preserving correction method in Reference 33.

There are several potential performance improvements to pursue in future work. In situations where the target rank is high, computing $\mathbf{Q}_0$, which involves a QR of a Khatri–Rao product of upper triangular matrices, becomes a more dominant cost of the CP-ALS-QR algorithm. The Khatri–Rao product of upper triangular matrices has structure which we do not exploit in our implementation, and which could lead to a more efficient implementation. We could also use dimension trees to speed up our implementation of the Multi-TTM function, the major dominant cost in both new algorithms. We currently use the Tensor Toolbox implementation but could improve the performance by reusing computations as in a dimension tree. The Gauss–Newton algorithm we use solves the approximate linear least squares problems via the normal equations. Another interesting direction to pursue would be to use the QR of the Jacobian to solve the least squares problems instead to improve the stability in the presence of ill-conditioning.

ACKNOWLEDGMENTS

The authors would like to acknowledge the support provided by the National Science Foundation through the grant CCF-1942892. This study does not have any conflicts to disclose.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ORCID

Xiaotian Liu https://orcid.org/0000-0001-9369-4029

REFERENCES

1. Bader BW, Kolda TG, et al. Tensor Toolbox for MATLAB, Version 3.2.1. 2021. Available from: www.tensortoolbox.org
2. Vervliet N, Debals O, Sorber L, Van Barel M, De Lathauwer L. Tensorlab 3.0. 2016. Available from: https://www.tensorlab.net
3. Kossaifi J, Panagakis Y, Anandkumar A, Pantic M. TensorLy: tensor learning in python. J Mach Learn Res. 2019;20(26):1–6. Available from: http://jmlr.org/papers/v20/18-277.html
4. Ballard G, Rouse K. General memory-independent lower bound for MTTKRP. In: Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing; 2020. p. 1–11.
5. Nisa I, Li J, Sukumaran-Rajam A, Vuduc R, Sadayappan P. Load-balanced sparse MTTKRP on GPUs. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2019. p. 123–133.
6. Phan AH, Tichavsky P, Cichocki A. Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations. IEEE Trans Signal Process. 2013;61(19):4834–46.
7. Smith S, Ravindran N, Sidiropoulos ND, Karypis G. SPLATT: efficient and parallel sparse tensor-matrix multiplication. In: Parallel and Distributed Processing Symposium (IPDPS). IEEE; 2015. p. 61–70.
8. Ballard G, Klinvex A, Kolda TG. TuckerMPI: a parallel C++/MPI software package for large-scale data compression via the Tucker tensor decomposition. ACM Trans Math Softw. 2020;46(2):13:1-13:31.
9. Chakaravarthy VT, Choi JW, Joseph DJ, Liu X, Murali P, Sabharwal Y, et al. On optimizing distributed Tucker decomposition for dense tensors. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2017. p. 1038–1047.
10. Kaya O, Uçar B. High performance parallel algorithms for the Tucker decomposition of sparse tensors. In: 45th International Conference on Parallel Processing (ICPP '16); 2016. p. 103–112.
11. Smith S, Karypis G. Accelerating the Tucker decomposition with compressed sparse tensors. In: European Conference on Parallel Processing; 2017. p. 653–68.
12. Demmel J. Applied numerical linear algebra. Philadelphia: SIAM; 1997.
13. Golub GH, Van Loan CF. Matrix computations. 4th ed. Baltimore: Johns Hopkins University Press; 2013.
14. Trefethen LN, Bau D. Numerical linear algebra. Philadelphia: SIAM; 1997.
15. Kolda TG, Bader BW. Tensor decompositions and applications. SIAM Rev. 2009;51(3):455–500.
16. Khatri C, Rao CR. Solutions to some functional equations and their applications to characterization of probability distributions. Sankhyā Ind J Stat Ser A. 1968;30(2):167–80.
17. Singh N, Ma L, Yang H, Solomonik E. Comparison of accuracy and scalability of Gauss-Newton and alternating least squares for CANDECOMP/PARAFAC decomposition. SIAM J Sci Comput. 2021;43:C290–311.

18. Sorber L, Van Barel M, De Lathauwer L. Optimization-based algorithms for tensor decompositions: canonical polyadic decomposition, decomposition in rank-(L_r,L_r,1) terms, and a new generalization. SIAM J Optim. 2013;23(2):695–720.
19. Vervliet N, De Lathauwer L. Numerical optimization-based algorithms for data fusion. Data handling in science and technology. Volume 31. Amsterdam: Elsevier; 2019. p. 81–128.
20. Eswar S, Hayashi K, Ballard G, Kannan R, Matheson MA, Park H. PLANC: parallel low-rank approximation with nonnegativity constraints. ACM Trans Math Softw. 2021;47(3):1–37.
21. Liavas AP, Kostoulas G, Lourakis G, Huang K, Sidiropoulos ND. Nesterov-based alternating optimization for nonnegative tensor factorization: algorithm and parallel implementation. IEEE Trans Signal Process. 2017;66:944–953.
22. Phan AH, Tichavsky P, Cichocki A. TENSORBOX: a MATLAB package for tensor decomposition; 2013. Available from. https://github.com/phananhhuy/TensorBox
23. Smith S, Karypis G. A medium-grained algorithm for distributed sparse tensor factorization. In: IEEE 30th International Parallel and Distributed Processing Symposium; 2016. p. 902–11.
24. Bader BW, Kolda TG. Efficient MATLAB computations with sparse and factored tensors. SIAM J Sci Comput. 2007;3(1):205–31.
25. Baskaran M, Meister B, Vasilache N, Lethin R. Efficient and scalable computations with sparse tensors. In: IEEE Conference on High Performance Extreme Computing; 2012. p. 1–6.
26. Beylkin G, Mohlenkamp MJ. Algorithms for numerical analysis in high dimensions. SIAM J Sci Comput. 2005;26(6):2133–59.
27. Strassen V. Gaussian elimination is not optimal. Numer Math. 1969;13:354–6. https://doi.org/10.1007/BF02165411
28. Brent RP. Algorithms for matrix multiplication. Stanford: Stanford University; 1970.
29. Smirnov AV. The bilinear complexity and practical algorithms for matrix multiplication. Comput Math Math Phys. 2013;53(12):1781–95. https://doi.org/10.1134/S0965542513120129
30. Benson AR, Ballard G. A framework for practical parallel fast matrix multiplication. Proceedings of the 20th ACM SIGPLAN symposium on principles and practice of parallel programming. PPoPP 2015. New York: ACM; 2015. p. 42–53.
31. Fawzi A, Balog M, Huang A, Hubert T, Romera-Paredes B, Barekatain M, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature. 2022;610(7930):47–53.
32. Bini D, Capovani M, Romani F, Lotti G. $O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. Inform Process Lett. 1979;8(5):234–5.
33. Phan AH, Tichavskỳ P, Cichocki A. Error preserving correction: a method for CP decomposition at a target error bound. IEEE Trans Signal Process. 2018;67(5):1175–90.

How to cite this article: Minster R, Viviano I, Liu X, Ballard G. CP decomposition for tensors via alternating least squares with QR decomposition. Numer Linear Algebra Appl. 2023;e2511. https://doi.org/10.1002/nla.2511