

Practical alternating least squares for tensor ring decomposition

Yajie Yu | Hanyu Li

College of Mathematics and Statistics, Chongqing University, Chongqing, People’s Republic of China

Correspondence
Hanyu Li, College of Mathematics and Statistics, Chongqing University, Chongqing 401331, People’s Republic of China.
Email: lihy.hy@gmail.com and hyli@cqu.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Number: 11671060;
Natural Science Foundation Project of Chongqing, Chongqing Science and Technology Commission, Grant/Award Number: cstc2019jcyj-msxmX0267

Abstract
Tensor ring (TR) decomposition has been widely applied as an effective approach in a variety of applications to discover the hidden low-rank patterns in multidimensional and higher-order data. A well-known method for TR decomposition is the alternating least squares (ALS). However, solving the ALS subproblems often suffers from high cost issue, especially for large-scale tensors. In this paper, we provide two strategies to tackle this issue and design three ALS-based algorithms. Specifically, the first strategy is used to simplify the calculation of the coefficient matrices of the normal equations for the ALS subproblems, which takes full advantage of the structure of the coefficient matrices of the subproblems and hence makes the corresponding algorithm perform much better than the regular ALS method in terms of computing time. The second strategy is to stabilize the ALS subproblems by QR factorizations on TR-cores, and hence the corresponding algorithms are more numerically stable compared with our first algorithm. Extensive numerical experiments on synthetic and real data are given to illustrate and confirm the above results. In addition, we also present the complexity analyses of the proposed algorithms.

KEYWORDS
alternating least squares, normal equation, QR factorization, tensor product, tensor ring decomposition

1 INTRODUCTION

Tensor ring (TR) decomposition is a simple but powerful tensor network for analyzing and interpreting latent patterns for multidimensional and higher-order data, that is, tensor, due to its excellent compression and data representation capacities. It is also referred to as the tensor chain format in previous mathematics literature, or the matrix product states format with periodic boundaries in physics literature. Specifically, the decomposition aims to represent a higher-order tensor by a sequence of third-order tensors that are multiplied circularly, and the specific format of element-wise form for the tensor $X \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is:

$$X(i_1, \ldots, i_N) = \text{Trace} \left( G_1(i_1)G_2(i_2)\cdots G_N(i_N) \right) = \text{Trace} \left( \prod_{n=1}^{N} G_n(i_n) \right),$$

where $G_n(i_n) = S_n(: , i_n , :) \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}$ is the $i_n$-th lateral slice of the core tensor (TR-core) $S_n \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}$. Note that a slice is a second-order section, that is, a matrix, of a tensor obtained by fixing all the tensor indices but two, and $R_{N+1} = R_1$. 

Received: 17 May 2023 | Revised: 20 October 2023 | Accepted: 17 November 2023
DOI: 10.1002/nla.2542
Numer Linear Algebra Appl. 2024;31:e2542. wileyonlinelibrary.com/journal/nla © 2023 John Wiley & Sons, Ltd.

1 of 19
The sizes of TR-cores, that is, $R_k$ with $k = 1, \ldots, N$, are called \textit{TR-ranks}. Additionally, we use the notation $\text{TR}(\{S_n\}_{n=1}^N)$ to denote the TR decomposition of a tensor.

Besides TR decomposition, there exist a number of other tensor decompositions, such as CANDECOMP/PARAFAC (CP) decomposition,\textsuperscript{5,6} Tucker decomposition,\textsuperscript{7} and tensor train (TT) decomposition.\textsuperscript{8} However, finding an optimal CP decomposition is NP-hard, Tucker decomposition usually suffers from the curse of dimensionality, and TT decomposition requires the rank constraint, that is, $R_1 = R_{N+1}$ has to be 1, and strict order on products of TT-cores, that is, it has no ability on rotational invariance. More details on limitations of TT decomposition can be found in Reference 1. TR decomposition can avoid the aforementioned deficiencies effectively and hence has become popular in recent years.

The problem of fitting $\text{TR}(\{S_n\}_{n=1}^N)$ to a tensor $\mathbf{X}$ can be written as the following minimization problem:

$$
\text{arg min}_{\{G_n\}_{n=1}^N} \|\text{TR}(\{S_n\}_{n=1}^N) - \mathbf{X}\|_F,
$$

where $\| \cdot \|_F$ denotes the Frobenius norm of a matrix or tensor. Deterministic algorithms for computing the above TR decomposition can be mainly divided into two categories.\textsuperscript{1,9} One is SVD-based, and the other is the alternating least squares (ALS). We use the abbreviation TR-SVD and TR-ALS for these two methods in the following text.

The former involves the computation of a series of SVDs of unfolding matrices and the latter needs to compute the pseudoinverses of the coefficient matrices in the ALS subproblems within iterations. Clearly, both of them are computationally prohibitive for large-scale tensors. See References 1,9 for further details on deterministic algorithms for TR decomposition.

Randomized algorithms for TR decomposition are effective methods to deal with the prohibitive problem mentioned above, which can reduce the computational complexities of deterministic algorithms and the communications between different levels of the memory hierarchy. Hence, some works were developed recently. For more specific, Yuan et al.\textsuperscript{10} devised a method that first applies a randomized Tucker decomposition, followed by a TR decomposition of the core tensor using either TR-SVD or TR-ALS. Then, the TR cores are contracted appropriately. Ahmadi-Asl et al.\textsuperscript{11} developed several randomized variants of the deterministic TR-SVD by replacing the SVDs with their randomized counterparts. As for TR-ALS, Malik and Becker\textsuperscript{12} proposed the TR-ALS-Sampled, which uses the leverage score sampling to reduce the size of the ALS subproblems. Malik\textsuperscript{13} further provided a new approach to approximate the leverage scores and devised the corresponding sampling algorithm. Yu and Li\textsuperscript{14} proposed two sketching-based randomized algorithms called TR-KSRFT-ALS and TR-TS-ALS, which use the Kronecker sub-sampled randomized Fourier transform and TensorSketch to reduce the subproblems, respectively.

Although the ALS-based randomized algorithms can reduce the computational complexities greatly, like deterministic algorithms, they also usually have to directly solve the ALS subproblems eventually and hence may suffer from expensive cost. Recently, the special structure of the coefficient matrices of the ALS subproblems was described explicitly in Reference 14 by a tensor product called \textit{subchain product}. It offers a possibility to tackle the above issue. Specifically, with the structure and a property of subchain product, we first propose a strategy to accelerate the calculation of the coefficient matrices of the normal equations for the ALS subproblems. This is similar to what is done in CP-ALS for CP decomposition, where the property of Khatri-Rao product is employed to simplify the calculations.\textsuperscript{5,6,11} Then, we investigate the QR factorizations of the coefficient matrices of the ALS subproblems via the aforementioned structure and defining the QR factorization for the third-order tensor to stabilize and facilitate the subproblems. This strategy is similar to the one for CP decomposition adopted in Reference 16. Tracing the source, it is the famous idea for stabilizing the least squares (LS) problem. As a result, the algorithm with this strategy is more stable than the one with our first strategy. Furthermore, we also combine these two algorithms to balance the running time and stability.

The remainder of this paper is organized as follows. Section 2 introduces some necessary definitions, propositions, and TR-ALS. In Section 3, we propose our main algorithms and analyze their computational complexities. Section 4 is devoted to numerical experiments to illustrate and confirm our methods. Finally, the concluding remarks of the whole paper are presented.

## 2 | PRELIMINARIES

Before presenting the necessary definitions regarding tensors, we denote $[I] \equiv \{1, \ldots, I\}$ for a positive integer $I$, and set $\frac{n_1 n_2 \cdots n_N}{1 \, \ldots \, 1 N} \equiv \sum_{n=1}^{N} (n_1 - 1) \prod_{j=1}^{N-1} n_j$ for the indices $i_1 \in [I_1], \ldots, i_N \in [I_N]$. 

Definition 1 (Classical mode-$n$ unfolding). The **classical mode-$n$ unfolding** of a tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the matrix $\mathbf{X}_{(n)}$ of size $I_n \times \prod_{j \neq n} I_j$ defined element-wise via

$$\mathbf{X}_{(n)}(i_n, i_1 \cdots i_{n-1} I_{n+1} \cdots I_N) = \mathbf{X}(i_1, \ldots, i_N).$$

Definition 2 (Mode-$n$ unfolding). The **mode-$n$ unfolding** of a tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the matrix $\mathbf{X}_{[n]}$ of size $I_n \times \prod_{j \neq n} I_j$ defined element-wise via

$$\mathbf{X}_{[n]}(i_n, i_{n+1} \cdots i_N I_1 \cdots I_{n-1}) = \mathbf{X}(i_1, \ldots, i_N).$$

Definition 3 ($n$-unfolding). The **$n$-unfolding** of a tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is the matrix $\mathbf{X}_{<n>}$ of size $\prod_{j=1}^N I_j \times \prod_{j=n+1}^N I_j$ defined element-wise via

$$\mathbf{X}_{<n>}(i_1, \ldots, i_n, i_{n+1} \cdots i_N) = \mathbf{X}(i_1, \ldots, i_N).$$

Definition 4 (Outer product). For $\mathbf{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, their **outer product** is a tensor of size $I_1 \times \cdots \times I_N \times J_1 \times \cdots \times J_M$ denoted by $\mathbf{A} \circ \mathbf{B}$ and defined element-wise via

$$(\mathbf{A} \circ \mathbf{B})(i_1, \ldots, I_N, j_1, \ldots, J_M) = \mathbf{A}(i_1, \ldots, I_N) \mathbf{B}(j_1, \ldots, J_M).$$

Definition 5 (Contracted tensor product). For $\mathbf{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$ with $I_n = J_m$, their **contracted tensor product** is a tensor of size $I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{m-1} \times J_{m+1} \times \cdots \times J_M$ denoted by $\mathbf{A} \times^{m,n} \mathbf{B}$ and defined element-wise via

$$(\mathbf{A} \times^{m,n} \mathbf{B})(i_1, \ldots, I_{n-1}, i_n, \ldots, I_{n+1}, \ldots, I_N, j_1, \ldots, J_{m-1}, j_m, \ldots, J_M) = \sum_{i=1}^{I_n} \mathbf{A}(i_1, \ldots, I_{n-1}, i, I_{n+1}, \ldots, I_N) \mathbf{B}(j_1, \ldots, J_{m-1}, j, J_{m+1}, \ldots, J_M).$$

Definition 6 (TTM). For $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathbf{U} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$, their **tensor-times-matrix (TTM) multiplication** is a tensor of size $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$ denoted by $\mathbf{X} \times_n \mathbf{U}$ and defined element-wise via

$$(\mathbf{X} \times_n \mathbf{U})(i_1, \ldots, I_{n-1}, j, I_{n+1}, \ldots, I_N) = \sum_{i_n=1}^{I_n} \mathbf{X}(i_1, \ldots, i_n, \ldots, I_N) \mathbf{U}(j, i_n).$$

Note that TTM is also frequently referred to as **$n$-mode product** in the literature. Multiplying an $N$th-order tensor by multiple matrices on distinct modes is known as **Multi-TTM**, whose computation can be performed by using a sequence of individual mode TTM, and can be done in any order. In particular, multiplying an $N$th-order tensor $\mathbf{X}$ by the matrices $\mathbf{U}_j$ with $j = 1, \ldots, N$ in each mode implies $\mathbf{Y} = \mathbf{X} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \cdots \times_N \mathbf{U}_N$. Its mode-$n$ unfolding can be presented as follows.

**Proposition 1.** Let the tensor $\mathbf{Y}$ have the form $\mathbf{Y} = \mathbf{X} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \cdots \times_N \mathbf{U}_N$, where $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, and $\mathbf{U}_n \in \mathbb{R}^{I_n \times J_n}$ for $n = 1, \ldots, N$. Then

$$\mathbf{Y}_{[n]} = \mathbf{U}_n \mathbf{X}_{[n]} (\mathbf{U}_{n-1} \circ \cdots \circ \mathbf{U}_1 \circ \mathbf{U}_N \circ \cdots \circ \mathbf{U}_{n+1}).$$

We now detail the TR-ALS mentioned in Section 1, which is the most popular algorithm for TR decomposition. To achieve this, we need the following definition.

Definition 7. Let $\mathbf{X} = \text{TR} \left( \left( \mathbf{S}_n \right)_{n=1}^N \right) \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. The **subchain tensor** $\mathbf{G}^{\otimes n} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ is the merging of all TR-cores expect the $n$-th one and can be written slice-wise via

$$\mathbf{G}^{\otimes n}(i_{n+1} \cdots i_N i_1 \cdots i_{n-1}) = \prod_{j=0}^{n-1} G_{ij}(i_{n+1}) \prod_{j=1}^{n} G_{ij}(i).$$

Thus, according to theorem 3.5 in Reference 1, the objective in (1) can be rewritten as the following $N$ subproblems

$$\arg \min_{\mathbf{G}^{\otimes n}} \left\| \mathbf{G}^{\otimes n} \mathbf{G}^{\otimes n} - \mathbf{X}^{\otimes n} \right\|_F, \quad n = 1, \ldots, N.$$
The so-called TR-ALS is a method that keeps all cores fixed except the \( n \)-th one and finds the solution to the LS problem (2) with respect to it. We summarize the method in Algorithm 1.

**Algorithm 1.** TR-ALS

**Input:** \( \mathcal{X} \in \mathbb{R}^{I_{1} \times \cdots \times I_J} \), TR-ranks \( R_1, \ldots, R_N \)

**Output:** TR-cores \( \{ \mathcal{S}_n \in \mathbb{R}^{R_n \times I_{1} \times \cdots \times I_{n-1}} \}_{n=1}^{N} \)

1. Initialize TR-cores \( \mathcal{S}_1, \ldots, \mathcal{S}_N \)
2. repeat
   3. for \( n = 1, \ldots, N \) do
      4. Compute \( \mathcal{G}_{[n]} \) from cores
      5. Update \( \mathcal{S}_n = \arg \min_{\mathcal{S}_n} \| \mathcal{X}^{[n]} \mathcal{G}_{[n]}^{\top} - \mathcal{X}^{[n]} \|_F 
   6. end for
3. until termination criteria met

The matrix \( \mathcal{G}_{[n]} \) in (2), that is, the subchain tensor \( \mathcal{G}^{[n]} \), has a special structure, which can be revealed elegantly with the subchain product defined as follows.

**Definition 8** (Subchain product;\(^{[14]}\)). For \( \mathcal{A} \in \mathbb{R}^{I_1 \times J_1 \times K} \) and \( \mathcal{B} \in \mathbb{R}^{K \times J_2 \times L_2} \), their mode-2 subchain product is a tensor of size \( I_1 \times J_1 J_2 \times I_2 \) denoted by \( \mathcal{A} \boxtimes_2 \mathcal{B} \) and defined as
\[
(\mathcal{A} \boxtimes_2 \mathcal{B})(j_1 j_2) = \mathcal{A}(j_1) \mathcal{B}(j_2),
\]
where \( \mathcal{A}(j_1) \) and \( \mathcal{B}(j_2) \) are the \( j_1 \)-th and \( j_2 \)-th lateral slices of \( \mathcal{A} \) and \( \mathcal{B} \), respectively.

Thus, \( \mathcal{G}_{[n]}^{[n]} \) can be expressed as Reference \(^{14}\):
\[
\mathcal{G}_{[n]}^{[n]} = \mathcal{S}_{n+1} \boxtimes_2 \cdots \boxtimes_2 \mathcal{S}_N \boxtimes_2 \mathcal{S}_1 \boxtimes_2 \cdots \boxtimes_2 \mathcal{S}_{n-1}.
\]

With this expression and the property of subchain product given in Proposition 2, we devised TR-KSRFT-ALS and TR-TS-ALS in Reference \(^{14}\).

**Proposition 2** (\(^{[14]}\)). Let \( \mathcal{A} \in \mathbb{R}^{I_1 \times J_1 \times K} \) and \( \mathcal{B} \in \mathbb{R}^{K \times J_2 \times L_2} \) be two third-order tensors, and \( \mathcal{A} \in \mathbb{R}^{R_1 \times J_1} \) and \( \mathcal{B} \in \mathbb{R}^{R_2 \times J_2} \) be two matrices. Then
\[
(\mathcal{A} \times_2 \mathcal{B}) \boxtimes_2 (\mathcal{B} \times_2 \mathcal{A}) = (\mathcal{A} \boxtimes_2 \mathcal{B}) \times_2 (\mathcal{B} \boxtimes_2 \mathcal{A}).
\]

3 | PROPOSED METHODS

We will present three practical ALS-based algorithms for TR decomposition. The first one is built on normal equation, the second one is based on QR factorization, and the last one is the combination of the preceding two algorithms.

3.1 | TR-ALS based on normal equation

Recall that the LS problem from (2) is typically solved by using the normal equation, that is,
\[
\mathcal{X}^{[n]} \mathcal{G}_{[2]}^{[n]} = \mathcal{G}_{[2]}^{[n]} \left( \mathcal{G}_{[2]}^{[n]} \right)^{\top}.
\]
From (3), it follows that \( \mathcal{G}_{[2]}^{[n]} \) can be written as \( \mathcal{G}_{[2]}^{[n]} = (\mathcal{S}_{n+1} \boxtimes_2 \cdots \boxtimes_2 \mathcal{S}_N \boxtimes_2 \mathcal{S}_1 \boxtimes_2 \cdots \boxtimes_2 \mathcal{S}_{n-1})^\top \mathcal{G}_{[2]}^{[n]} \). To compute it efficiently, we now propose a property of subchain product.

**Proposition 3.** Let \( \mathcal{A} \in \mathbb{R}^{I_1 \times J_1 \times K_1} \), \( \mathcal{B} \in \mathbb{R}^{K_1 \times R_1 \times L_1} \), \( \mathcal{C} \in \mathbb{R}^{I_2 \times J_2 \times K_2} \) and \( \mathcal{D} \in \mathbb{R}^{K_2 \times R_2 \times L_2} \) be third-order tensors. Then
\[
(\mathcal{A} \boxtimes_2 \mathcal{B})^\top (\mathcal{C} \boxtimes_2 \mathcal{D})^\top = \left( \left( \sum_{r=1}^{R} \mathcal{B}(r)^{\top} \circ \mathcal{D}(r)^{\top} \right) \cdot \left( \sum_{l=1}^{L_2} (\mathcal{A}(j) \circ \mathcal{C}(j)^{\top} \circ \mathcal{D}(l)^{\top}) \right) \right)^{\top}.
\]
Proof. By the definitions of subchain product and mode-\( n \) unfolding, upon some computations, we have

\[
\left( (A \boxtimes_2 B)^{[1]} (C \boxtimes_2 D)^{[2]} \right) (l_1 l_2) = \sum_{j=1}^J \left( \sum_{k_1=1}^{K_1} A(i_1,j,k_1)B(k_1,r,l_1) \sum_{k_2=1}^{K_2} C(i_2,j,k_2)D(k_2,r,l_2) \right).
\]

On the other hand, by the definitions of contracted tensor product, outer product and \( n \)-unfolding, we can get

\[
\left( \left( \sum_{r=1}^R B(r)^T \circ D(r)^T \right) \otimes_{2,4} \left( \sum_{j=1}^J A(j)^T \circ C(j)^T \right) \right) (l_1 l_2) = \sum_{j,k} \left( \sum_{r=1}^R B(r)^T \circ D(r)^T \right) (l_1,k_1,l_2) \left( \sum_{j=1}^J A(j)^T \circ C(j)^T \right) (k_1,i_1,k_2).\]

Thus, according to the exchangeability of summation symbols, the proof is completed. \( \blacksquare \)

Remark 1. The contracted tensor product used in Proposition 3 is a little different from the one in Definition 5. It can be regarded as a general contracted tensor product. Specifically, for \( A \in \mathbb{R}^{I_1 \times \cdots \times I_K} \) and \( B \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_K} \), their general product is a fourth-order tensor of size \( I_1 \times I_2 \times R_1 \times R_2 \) defined as

\[
(A \otimes_{2,4} B)(i_1,i_2,r_1,r_2) = \sum_{j,k} A(i_1,j,k)B(j,i_2,k).\]

According to Proposition 3, the coefficient matrix \( \left( G_{[2]}^{\otimes n} \right)^T \left( G_{[2]}^{\otimes n} \right) \) can be computed efficiently as \( \left( G_{[2]}^{\otimes n} \right)^T G_{[2]}^{\otimes n} = S_{n<2}^{-1} \), where

\[
S_n = \mathcal{P}_{n-1} \times_{2,4} \cdots \times_{2,4} \mathcal{P}_1 \times_{2,4} \mathcal{P}_N \times_{2,4} \cdots \times_{2,4} \mathcal{P}_{n+1}.
\]

and

\[
\mathcal{P}_j = \sum_{i_j=1}^{l_j} G_j(i_j)^T \circ G_j(i_j), \quad j \neq n.
\]

Putting the above together, we can devise an algorithm called TR-ALS-NE, whose details are listed in Algorithm 2.

Algorithm 2. TR-ALS-NE (Proposal)

**Input:** \( X \in \mathbb{R}^{I_1 \times \cdots \times I_N} \), TR-ranks \( R_1, \ldots, R_N \)

**Output:** TR-cores \( \{ S_n \in \mathbb{R}^{R_n \times I_n \times R_{n+1}} \}_{n=1}^N \)

1. Initialize TR-cores \( S_1, \ldots, S_N \)
2. Compute \( \mathcal{P}_1 = \sum_{i_1=1}^{l_1} G_1(i_1)^T \circ G_1(i_1)^T, \ldots, \mathcal{P}_N = \sum_{i_N=1}^{l_N} G_N(i_N)^T \circ G_N(i_N)^T \)
3. repeat
4. \hspace{1cm} for \( n = 1, \ldots, N \) do
5. \hspace{2cm} \( S_n \leftarrow \mathcal{P}_{n-1} \times_{2,4} \cdots \times_{2,4} \mathcal{P}_1 \times_{2,4} \mathcal{P}_N \times_{2,4} \cdots \times_{2,4} \mathcal{P}_{n+1} \)
6. \hspace{2cm} \( S_{n<2}^{-1} \leftarrow S_{n+1} \otimes_1 \otimes_2 \cdots \otimes_2 \mathcal{S}_N \otimes_1 \mathcal{S}_1 \otimes_1 \otimes_2 \cdots \otimes_2 \mathcal{S}_{n-1} \)
7. \hspace{2cm} \( M_n \leftarrow X(n) G_{n<2}^{[2]} \)
8. \hspace{2cm} solve \( G_{n<2}^{[2]} S_{n<2} = M_n \)
9. \hspace{2cm} Recompute \( \mathcal{P}_n = \sum_{i_n=1}^{l_n} G_n(i_n)^T \circ G_n(i_n)^T \) for the updated TR-core \( S_n \)
10. \end for
11. until termination criteria met
Remark 2. It can be found that the methods TR-ALS and TR-ALS-NE are equivalent in mathematics theory. Hence, they have the same conclusions on convergence, which is also confirmed by the numerical results in Experiment A-II in Section 4.1. That is, they require almost the same number of iterations to achieve the same error. Since a new way is adopted to compute \((G_{[2]}^{\otimes n})^T G_{[2]}^{\otimes n}\), the cost of our method is cheaper than that of TR-ALS, which is supported by the discussions on the complexities of various methods in Section 3.3 and the numerical results in Experiments A-I and A-II in Section 4.1. For the convergence analyses of TR-ALS, refer to, for example, References 20,21 and references therein.

Now, we analyze the computational complexity of TR-ALS-NE shown in Algorithm 2. Recall that \(X\) has dimensions \(I_1 \times \cdots \times I_N\) and its TR-ranks are \(R_1, \ldots, R_N\). To simplify notation, we make the assumptions throughout the analysis that \(I_n = I\) and \(R_n = R\) for all \(n \in [N]\). And, we ignore any cost associated with, for example, checking termination conditions.

Upfront costs of TR-ALS-NE:

**Line 1: Initializing cores.** This depends on how to initiate the cores. We assume that they are randomly drawn, for example, from a Gaussian distribution, resulting in a cost \(\mathcal{O}(NIR^4)\).

**Line 2:** It costs \(\mathcal{O}(NIR^4)\) according to the definition of the outer product.

The costs of per outer loop iteration in TR-ALS-NE:

**Line 5: Computing the general contracted tensor product.** It costs \(\mathcal{O}((N - 2)R^6)\). Doing this for each of the \(N\) cores in the inner loop brings the cost \(\mathcal{O}(N(N - 2)R^6)\).

**Line 6: Computing the unfolding subchain tensor.** If the \(N - 1\) cores are dense and contracted in sequence, the cost is

\[ R^3(I^2 + I^3 + \cdots + I^{N-1}) \leq R^3(NI^{N-2} + I^{N-1}) \leq 2R^3I^{N-1} = \mathcal{O}(I^{N-1}R^3), \]

where \(I > N\) is used. Doing this for each of the \(N\) cores in the inner loop brings the cost \(\mathcal{O}(NI^{N-1}R^3)\).

**Line 7: Computing MTSSP.** It costs \(\mathcal{O}(INR^2)\) per inner loop iteration, that is, \(\mathcal{O}(NI^N R^2)\) per outer loop iteration.

**Line 8: Solving the normal equation.** We consider the cost of the method for normal equation described in sec. 5.3.2 in Reference 22. It costs \(\mathcal{O}(R^6 + IR^4)\). Doing this for each of the \(N\) cores in the inner loop brings the cost \(\mathcal{O}(NR^6 + NIR^4)\).

**Line 9:** It costs \(\mathcal{O}(IR^3)\) per inner loop iteration, that is, \(\mathcal{O}(NIR^4)\) per outer loop iteration.

Putting them all together, we have the total complexity of TR-ALS-NE when \(I > N\):

\[ \mathcal{O}(NIR^2 + NIR^4 + IT \cdot (N^2R^6 + NI^{N-1}R^3 + NI^N R^2 + NR^6 + 2NR^4)). \]

where “\(IT\)” denotes the number of outer loop iterations. Further, applying \(I > N\) and \(I > R\), the leading order complexity can be derived as

\[ \mathcal{O}(NIR^4 + IT \cdot (N^2R^6 + NI^N R^2)). \]

Hence, in TR-ALS-NE, the cost is dominated by computing the general contracted tensor product in Line 5 and computing MTSSP in Line 7.

### 3.2 TR-ALS based on QR factorization

We begin with a definition of QR factorization for the 3rd-order tensor, which is essentially equivalent to the QR factorization of mode-\(n\) unfolding. The reason why we write it in the following forms is to keep the operations in tensor format and hence make the formulation of the derivation for the algorithms be more elegant.

**Definition 9** (Mode-\(n\) QR factorization). For \(A \in \mathbb{R}^{I_1 \times I_2 \times I_3}\), its mode-\(n\) QR factorization is defined as follows:

1. If \(I_n \geq \prod_{j \neq n} I_j, n = 1, 2, 3,\)

\[ A = R \times_n Q, \quad n = 1, 2, 3, \]

where \(Q \in \mathbb{R}^{I_n \times \prod_{j \neq n} I_j}\) is an orthogonal matrix, and \(R\) is a third-order tensor whose mode-\(n\) unfolding matrix is a upper triangular matrix of size \(\prod_{j \neq n} I_j \times \prod_{j \neq n} I_j\).
Finally, by forming $W_n = Y_{[n]}Q_0$, we obtain a smaller LS problem

$$
\text{arg min}_{G_{n(2)}} \left\| W_n - G_{n(2)}R_{[2]}^\top \right\|_F,
$$

(5)

from which we can compute $G_{n(2)}$ and hence the TR-core $G_n$. We call this method TR-ALS-QR and detail it in Algorithm 3.

**Algorithm 3. TR-ALS-QR (Proposal)**

| Input: $X \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, TR-ranks $R_1, \ldots, R_N$ |
| Output: TR-cores $\{G_n \in \mathbb{R}^{R_n \times I_n \times R_{n+1}}\}_{n=1}^N$ |

1. Initialize TR-cores $G_1, \ldots, G_N$
2. Compute the mode-2 QR factorizations $R_1 \times_2 Q_1, \ldots, R_N \times_2 Q_N$ of TR-cores
3. repeat
4. for $n = 1, \ldots, N$ do
5. $V_n \leftarrow R_{n+1} \otimes_2 \cdots \otimes_2 R_N \otimes_2 R_1 \otimes_2 \cdots \otimes_2 R_{n-1}$
6. Compute mode-2 QR factorization $V_n = R_2Q_0$
7. $Y_n \leftarrow X \otimes_2 Q_1 \otimes_2 \cdots \otimes_2 Q_{n-1} \otimes_2 Q_{n+1} \otimes_2 \cdots \otimes_2 Q_N$
8. $W_n \leftarrow Y_{[n]}Q_0$
9. Solve $G_{n(2)}R_{[2]}^\top = W_n$ by substitution
10. Recompute the mode-2 QR factorization $R_n \times_2 Q_n$ for the updated TR-core $G_n$
4. end for
12. until termination criteria met
Remark 3. The method TR-ALS-QR is also equivalent to TR-ALS in mathematics theory. As pointed out in sec. 5.3 in Reference 22, QR factorization can stabilize the LS problem and the methods based on normal equation are more sensitive. Hence, for ill-conditioned problems, TR-ALS-QR may need fewer iterations compared with TR-ALS-NE. This also suggests that TR-ALS-QR usually performs better than TR-ALS-NE in computing time though the former is a little more expensive than the latter. These results are supported by the comparisons on the complexities between these two methods in Section 3.3 and Experiments B-II and B-III in Section 4.2.

Now, we consider the complexity analysis of Algorithm 3 with the same assumptions as done for TR-ALS-NE. Here, we treat $V_n$ as a dense tensor without discussing the possibility of exploiting sparsity.

Upfront costs of TR-ALS-QR:

**Line 1: Initializing cores.** This is the same as the one for TR-ALS-NE. That is, we assume that the cores are randomly drawn, for example, from a Gaussian distribution, resulting in a cost $\mathcal{O}(NIR^2)$.

**Line 2: Compute mode-2 QR factorization of TR-cores.** This is equivalent to computing the QR factorization of $G_{m(2)}$ for $n = 1, \ldots, N$ resulting in a cost $\mathcal{O}(NIR^2)$ if $I > R^2$, and $\mathcal{O}(NI^2R^2)$ otherwise.

The costs of per outer loop iteration in TR-ALS-QR:

**Line 5: Compute $V_n$.** If $\mathcal{R}_j$ for $j \neq n$ are dense and the product is implemented in sequence, the cost is

$$\mathcal{O}(R^2(R^4 + R^6 + \cdots + R^{2N-2})) \leq \mathcal{O}(R^{2N+1}).$$

Doing this for each of the $N$ cores in the inner loop brings the cost $\mathcal{O}(NR^{2N+1})$.

**Line 6: Compute mode-2 QR factorization of $V_n$.** It costs $\mathcal{O}(R^{2N+2})$ because $V_n$ has dimensions $R \times R^{2N-2} \times R$, that is, $\mathcal{O}(NR^{2N+2})$ per outer loop iteration.

**Line 7: Multi-TTM.** We compute the resulting tensor $\mathcal{Y}$, which has dimensions $R^2 \times \cdots \times I \times \cdots \times R^2$, by performing the single TTM's in sequence. Thus, the overall cost of the Multi-TTM is

$$\mathcal{O}\left(R^{2N}\left(1 + \frac{R^2}{I} + \frac{R^4}{I^2} + \cdots + \frac{R^{2N-4}}{I^{N-2}}\right)\right)$$

per inner loop iteration. Hence, the multi-TTM costs $\mathcal{O}(NI^2R^2)$ per outer loop iteration if $I > R^2$, and $\mathcal{O}(NI^2R^{2N-2})$ otherwise.

**Line 8: Compute $W_n$.** It costs $\mathcal{O}(IR^{2N})$ per inner loop iteration, that is, $\mathcal{O}(NIR^{2N})$ per outer loop iteration.

**Line 9: Solve triangular system.** It costs $\mathcal{O}(IR^4)$ per inner loop iteration, that is, $\mathcal{O}(NIR^4)$ per outer loop iteration.

**Line 10: Recompute mode-2 QR factorization of the $n$-th TR-core.** It costs $\mathcal{O}(IR^4)$ per inner loop iteration if $I > R^2$, and $\mathcal{O}(IR^2R^2)$ otherwise, that is, $\mathcal{O}(NIR^4)$ per outer loop iteration if $I > R^2$, and $\mathcal{O}(NI^2R^2)$ otherwise.

Putting them all together, we have the total complexity of TR-ALS-QR:

$$\begin{cases} \mathcal{O}(NIR^2 + NIR^4 + it \cdot (NR^{2N+1} + NR^{2N+2} + NI^2R^2 + NIR^2 + 2NR^4)), & I > R^2, \\ \mathcal{O}(NIR^2 + NI^2R^2 + it \cdot (NR^{2N+1} + NR^{2N+2} + NI^2R^{2N-2} + NIR^2 + NIR^4 + NI^2R^2)), & I < R^2, \end{cases}$$

where “it” denotes the number of outer loop iterations. Further, the leading order complexity can be written as

$$\begin{cases} \mathcal{O}(NIR^4 + it \cdot NI^2R^2), & I > R^2, \\ \mathcal{O}(NI^2R^2 + it \cdot NR^{2N+2}), & I < R^2. \end{cases}$$

So, in TR-ALS-QR, the cost is dominated by computing the Multi-TTM in Line 7 if $I > R^2$, and the mode-2 QR factorization of $V_n$ in Line 6 otherwise.

### 3.3 TR-ALS based on QR factorization and normal equation

Recalling the derivation of TR-ALS-QR, we find that $V_n$ also has the subchain product structure. Moreover, the complexity analysis for TR-ALS-QR shows that computing the mode-2 QR factorization of $V_n$ is expensive. Thus, we propose to
solve (4), that is,
\[
\arg \min_{G_{n(2)}} \| Y_{[n]} - G_{n(2)} V_{n(2)}^T \|_F.
\]

using TR-ALS-NE. We call this method TR-ALS-QRNE and detail it in Algorithm 4.

**Algorithm 4.** TR-ALS-QRNE (Proposal)

| Input: | $X \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, TR-ranks $R_1, \ldots, R_N$ |
|---|---|
| Output: | TR-cores $(S_n)_{n=1}^N$ in $\mathbb{R}^{R_n \times I_n \times R_{n+1}}$ |
| 1: | Initialize TR-cores $S_1, \ldots, S_N$
| 2: | Compute the mode-2 QR factorizations $R_1 \times_2 Q_1, \ldots, R_N \times_2 Q_N$ of TR-cores
| 3: | Compute $P_1 = \sum_{i_1=1}^{I_1} R_1(i_1) \circ R_1(i_1)^T$, $\ldots$, $P_N = \sum_{i_N=1}^{I_N} R_N(i_N) \circ R_N(i_N)^T$
| 4: | repeat
| 5: | for $n = 1, \ldots, N$ do
| 6: | $S_n \leftarrow P_{n-1} \times_2 \cdots \times_2 P_1 \times_2 \cdots \times_2 P_N \times_2 \cdots \times_2 P_{n+1}$
| 7: | $V_n \leftarrow R_1 \times_2 \cdots \times_2 R_N \times_2 M_{n-1}$
| 8: | $Y \leftarrow X \times_1 Q_1 \times_1 \cdots \times_1 Q_{n-1} \times_1 \cdots \times_1 Q_{N}$
| 9: | $M_n \leftarrow Y_{[n]}^T V_{n(2)}$
| 10: | Solve $G_{n(2)} S_{n<2} = M_n$
| 11: | Recompute the mode-2 QR factorization $R_n \times_2 Q_n$ for the updated TR-core $S_n$
| 12: | Recompute $P_n = \sum_{i_n=1}^{I_n} R_n(i_n) \circ R_n(i_n)^T$ for the updated $R_n$
| 13: | end for
| 14: | until termination criteria met

The complexity analysis of Algorithm 4 is similar to those of TR-ALS-NE and TR-ALS-QR when treating $V_n$ as a dense tensor, so we won’t go into details here. The following Table 1 summarizes the computational complexities of the algorithms involved in this paper in detail.

From Table 1, we can find the following results for higher-order tensors.

- **TR-ALS and TR-ALS-NE.** The dominating cost of TR-ALS in outer loop is $O(N I^{N-1} R^4 + N I^N R^2)$, which appears in solving the ALS subproblems. Accordingly, the dominating cost of TR-ALS-NE appears in computing the general contracted tensor product and MTTSP, which cost $O(N^2 R^6)$ and $O(N I^N R^2)$, respectively. They have the same leading order when $I > R^2$, but TR-ALS-NE is much closer to twice as fast as TR-ALS. This is mainly because we use a new method to compute the coefficient matrices of the normal equations for the ALS subproblems. It reduces the cost from $O(I^{N-1} R^4)$ to $O(N I R^4 + N R^6)$. If $I < R^2$, then $N I^{N-1} R^4 \geq N I^N R^2$, and $N I^{N-1} R^4 \geq N^2 R^6$ for $N \geq 4$. So, in this case, the leading order of the complexity for TR-ALS in outer loop is larger than that for TR-ALS-NE and the aforementioned difference between cost will be more remarkable.

- **TR-ALS-NE and TR-ALS-QR.** The dominating cost of TR-ALS-QR when $I < R^2$, appearing in computing mode-2 QR factorization of $V_n$, is $O(N R^{2N+2})$, which is larger than $O(N I^N R^2)$ for TR-ALS-NE, especially when $R^2$ is significantly large. In addition, for the aforementioned case, the complexity in the ‘Others’ part for TR-ALS-QR is not less than that for TR-ALS-NE either. Therefore, a wrap-up is that TR-ALS-QR is slower than TR-ALS-NE. However, in the case of $I > R^2$, the two algorithms have the identical leading order computational complexity. Considering that TR-ALS-QR is more stable than TR-ALS-NE, the former may perform better in practice, especially for ill-conditioned problems.

- **TR-ALS-QR and TR-ALS-QRNE.** Compared to TR-ALS-QR, TR-ALS-QRNE mainly reduces the computational complexity in the ‘Others’ part. So a wrap-up is that TR-ALS-QRNE is faster than TR-ALS-QR.

## 4 NUMERICAL EXPERIMENTS

To test our proposed methods, we choose TR-ALS as the main baseline. The function $\texttt{tr}_\texttt{als}$. $m$ from Reference 12 and available at [https://github.com/OsmanMalik/tr-als-sampled](https://github.com/OsmanMalik/tr-als-sampled) is used in the specific experiments. All experiments are run
| Method | Upfront costs | Per outer loop iteration |
|--------|---------------|--------------------------|
|        | Initializing  | Gram*/QR | MTTSP/TTM | Solving | Gram/QR | Others |
| TR-ALS | $\mathcal{O}(NIR^2)$ | —         | —         | $\mathcal{O}(NI^{N-1}R^4) + \mathcal{O}(N^4R^4) + \mathcal{O}(NR^6 + NIR^4)$ | —         | $\mathcal{O}(NIR^4)$ |
| (normal equation-based) | $\mathcal{O}(NIR^2)$ | —         | —         | $\mathcal{O}(NI^{N-1}R^4) + \mathcal{O}(N^4R^4) + \mathcal{O}(NR^6 + NIR^4)$ | —         | $\mathcal{O}(NIR^4)$ |
| TR-ALS | $\mathcal{O}(NIR^2)$ | $\mathcal{O}(NIR^4)$ | $\mathcal{O}(NI^3R^2)$ | $\mathcal{O}(NR^6 + NIR^4)$ | $\mathcal{O}(NIR^4)$ | $\mathcal{O}(NIR^4)$ + $\mathcal{O}(N(N - 2)R^2)$ + $\mathcal{O}(NI^{N-3}R^3)$ |
| (QR-based) | $\mathcal{O}(NIR^2)$ | $\mathcal{O}(NIR^4)$ | $\mathcal{O}(NR^6 + NIR^4)$ | $\mathcal{O}(NIR^4)$ | $\mathcal{O}(NIR^4)$ + $\mathcal{O}(N(N - 2)R^2)$ + $\mathcal{O}(NI^{N-3}R^3)$ |
| TR-ALS-NE | $\mathcal{O}(NIR^2)$ | $\mathcal{O}(NIR^4)$ | $\mathcal{O}(NR^6 + NIR^4)$ | $\mathcal{O}(NR^6 + NIR^4)$ | $\mathcal{O}(NIR^4)$ + $\mathcal{O}(N(N - 2)R^2)$ + $\mathcal{O}(NI^{N-3}R^3)$ |
| TR-ALS-QR | $\mathcal{O}(NIR^2)$ | $\mathcal{O}(NIR^4)$ or $\mathcal{O}(NI^2R^3)$ | $\mathcal{O}\left(\frac{NR^2I^N(1 + \frac{p^2}{R} + \frac{R^4}{R^2})}{I + \frac{R^2}{R}} + \cdots + \frac{R^{2N-4}}{R^{N-2}}\right)$ | $\mathcal{O}(NIR^4)$ | $\mathcal{O}(NIR^4)$ or $\mathcal{O}(N^2R^2)$ | $\mathcal{O}(NR^{N+1}) + \mathcal{O}(NR^{2N+2}) + \mathcal{O}(NIR^N)$ |
| TR-ALS-QRNE | $\mathcal{O}(NIR^2)$ | $\mathcal{O}(NIR^4)$ or $\mathcal{O}(NI^2R^3)$ or $\mathcal{O}(NIR^6)$ | $\mathcal{O}\left(\frac{NR^2I^N(1 + \frac{p^2}{R} + \frac{R^4}{R^2})}{I + \frac{R^2}{R}} + \cdots + \frac{R^{2N-4}}{R^{N-2}}\right)$ | $\mathcal{O}(NR^6 + NIR^4)$ | $\mathcal{O}(NIR^4)$ or $\mathcal{O}(NIR^4)$ | $\mathcal{O}(NIR^4)$ or $\mathcal{O}(NIR^4)$ + $\mathcal{O}(NR^6)$ + $\mathcal{O}(NR^{N+1})$ |

*We call the tensor like $\mathcal{P}_n$ the Gram tensor.
on Matlab R2022b on a computer with an Intel Xeon W-2255 3.7 GHz CPU and 256 GB memory. Additionally, we also use the MATLAB Tensor Toolbox.\textsuperscript{23}

All the synthetic tensors have the same dimensions in all modes and they are generated by creating $N$ TR-cores of size $R_\text{true} \times I \times R_\text{true}$ firstly (note that for the target rank of all algorithms, we denote it as $R$). These TR-cores may be generated in different ways, which will be detailed in subsequent experiments. Then, we form the tensors by $\mathbf{X}_{\text{true}} = TR\left(\{S_n\}_{n=1}^{N}\right)$. Finally, the noise is added to obtain the observed tensors:

$$
\mathbf{X} = \mathbf{X}_{\text{true}} + \eta \left(\frac{\|\mathbf{X}_{\text{true}}\|_F}{\|\mathbf{N}\|_F}\right) \mathbf{N},
$$

where the entries of $\mathbf{N} \in \mathbb{R}^{I \times \cdots \times I}$ are drawn from a standard normal distribution and the parameter $\eta$ is the amount of noise.

As stated in the discussions of computational complexities, we use random Gaussian tensors to initiate the TR-cores for all the related algorithms. For the termination criterion, all algorithms are terminated only after the maximum number of iterations being reached, and, unless otherwise stated, we set the maximum number to be 20. Then, the running time and the relative errors via the formula

$$
\frac{\|\mathbf{X} - \hat{\mathbf{X}}\|_F}{\|\mathbf{X}\|_F} = \frac{\|\mathbf{X} - TR\left(\{\hat{S}_n\}_{n=1}^{N}\right)\|_F}{\|\mathbf{X}\|_F},
$$

where the TR-cores $\hat{S}_n$ are computed by various algorithms, can be reported and compared. Again, unless otherwise stated, the numerical results are the averages over 10 runs.

### 4.1 Efficiency of TR-ALS-NE

We use two experiments to test the speedup of TR-ALS-NE over TR-ALS.

#### 4.1.1 Experiment A-I

Our first experiment ignores the convergence of TR-ALS and TR-ALS-NE and merely compares the computational time for each iteration of these two methods.

We consider third- and fifth-order tensors of various sizes without noise. Each TR-core is generated by a random Gaussian tensor with entries drawn independently from a standard normal distribution. Figure 1 shows how much cheaper each iteration of the ALS is when using TR-ALS-NE. Obviously, TR-ALS-NE runs faster than TR-ALS in both the third- and fifth-order tensors.

#### 4.1.2 Experiment A-II

Our second experiment considers both the convergence and computational cost. We still use the data in Experiment A-I. Figure 2 shows the numerical results on decreasing trend of the relative errors as the number of iterations and time increase. We can see that TR-ALS-NE can achieve almost the same convergence errors as TR-ALS but with much less computing time for various cases. And, the time gap becomes more obvious as the tensor order, that is, $N$, and the dimensionality, that is, $I$, increase. This means that our method is more effective for large-scale data.

### 4.2 Stability of TR-ALS-QR and TR-ALS-QRNE

Three experiments on different datasets are presented to show the stronger stability of TR-ALS-QR and TR-ALS-QRNE compared with TR-ALS-NE.
FIGURE 1  Mean time per iteration of TR-ALS and TR-ALS-NE for third- and fifth-order tensors. Each dot represents the mean iteration time over 20 iterations (no checks for convergence).

4.2.1  Experiment B-I

Our first experiment is run on the data used in Experiment A-I, which is well-conditioned. The numerical results are presented in Figure 3, from which we can see that the above three methods have similar performance in accuracy but a little difference in running time if $I > R_{true}^2$, but the aforementioned differences become a little larger otherwise. More detailedly, in the case of $I > R_{true}^2$, for the two lower-order tensors, the two QR-based algorithms perform a little better, but for the 40-dimensional fifth-order tensor, TR-ALS-NE is a little faster. When switching to the 60-dimensional 5th-order tensor, the performance of all the methods is almost the same. In the case of $I < R_{true}^2$, the gap on running time between TR-ALS-QR and the other two methods is a little larger. The main reasons for the above findings may be that, for well-conditioned data, the advantage of the stability of QR-based methods is not remarkable, and TR-ALS-QR itself is also more expensive than TR-ALS-NE and TR-ALS-QRNE when $I < R_{true}^2$. In addition, in the above two cases, for TR-ALS-QRNE and TR-ALS-QR, the former always runs a little faster than the latter as expected.

4.2.2  Experiment B-II

In this experiment, we generate the TR-cores as done in Reference 24 to control their collinearity. Specifically, we first use the Matlab function MATRAN DC ONG(100, 25, $\gamma$) in the MATLAB Tensor Toolbox, which can create a matrix of size $100 \times 25$ such that its each column has norm 1 and any two columns have an inner product equal to $\gamma$, to generate three $100 \times 25$ matrices. Then, these matrices are reshaped to $5 \times 100 \times 5$ TR-cores. The parameter $\gamma$ mentioned above is used to control the congruence of matrices and hence the collinearity of TR-cores. Note that for the congruence 0.5, the level of collinearity is relatively low. Actually, even for the congruence 0.9, the underlying cores may be only mildly collinear. For further details on the method and the function, see References 23, 24.

We test all combinations of three different noise levels $10^{-4}$, $10^{-7}$, and $10^{-10}$, and three different collinearity levels $1 - 10^{-4}$, $1 - 10^{-7}$, and $1 - 10^{-10}$. And, we run 100 trials of each algorithm for each configuration. The numerical results are presented in Figure 4, where we see that the combination of noise and collinearity affects the ill-conditioning of the problem in different ways, and the QR-based algorithms have better performance than TR-ALS-NE in terms of relative error. Some specific discussions are in order.

1) For the case of the noise level being $10^{-4}$ and the collinearity level being $1 - 10^{-4}$, that is, the data is not ill-conditioned, all the algorithms have similar performance as expected. However, from the first row of the figure, we see that, for the fixed noise level $10^{-4}$, the volatility of the behavior of TR-ALS-NE is rising as the collinearity level increases.
2) From the first column of the figure, that is, the case of fixed collinearity level \(1 - 10^{-4}\) and changing noise level, we see a much more accurate solution output by TR-ALS-QR and TR-ALS-QRNE as the noise level decreases. This is mainly because the high levels of Gaussian noise can alleviate the ill-conditioning.

3) In the remaining four cases, that is, the combinations of higher collinearity \((1 - 10^{-7}\) and \(1 - 10^{-10}\)) and low noise \((10^{-7}\) and \(10^{-10}\)), which lead to ill-conditioned subproblems, TR-ALS-QR and TR-ALS-QRNE are always robust. That is, they obtain the lowest relative errors and have stable performances in all scenarios. Whereas, TR-ALS-NE tends not to converge quickly and also suffers from higher forward or backward errors.

The above numerical findings are consistent with the explanations in Remark 3 and the theoretical discussions on computational complexities in Section 3.3.
**FIGURE 3**Number of iterations versus relative errors and time versus relative errors output by algorithms for various tensors.

**FIGURE 4**Boxplots and line chart of relative errors for TR-ALS-NE, TR-ALS-QR, and TR-ALS-QRNE on a $100 \times 100 \times 100$ synthetic tensor of rank 5 with three different levels of collinearity for the true TR-cores and three different levels of Gaussian noise added. Each algorithm is run 100 trials.
4.2.3 Experiment B-III

In this experiment, the $5 \times 100 \times 5$ TR-cores are also constructed by reshaping some special $100 \times 25$ random matrices. The difference is that the entries of matrices are drawn from the multivariate $t$-distribution. Specifically, we use the Matlab function `Mvtrnd(C, d, 100)` to implement this process, where $C \in \mathbb{R}^{25 \times 25}$ is a correlation matrix whose $(i, j)$-th element is equal to $\theta^{|i-j|}$ with $\theta$ describing the correlation level, and $d$ is the degrees of freedom. We always set $d = 1$ in our specific experiment.

We test all combinations of three different noise levels $10^{-4}$, $10^{-7}$, and $10^{-10}$, and three different correlation levels $1 - 10^{-1}$, $1 - 10^{-4}$, and $1 - 10^{-7}$, and run 100 trials of each algorithm for each configuration. Figure 5 reports the numerical results, which are similar to the ones for Experiment B-II. That is, for well-conditioned cases, for example, the one of $\eta = 10^{-4}$ and $\theta = 1 - 10^{-1}$, all the algorithms have similar performance. However, for ill-conditioned cases, for example, the combinations of $\eta = 10^{-7}$, $10^{-10}$ and $\theta = 1 - 10^{-4}$, $1 - 10^{-7}$, TR-ALS-QR and TR-ALS-QRNE are still robust, that is, they obtain lower relative errors with little variation compared with TR-ALS-NE.

Figure 5: Boxplots and line chart of relative errors for TR-ALS-NE, TR-ALS-QR, and TR-ALS-QRNE on a $100 \times 100 \times 100$ synthetic tensor of rank 5 with three different kinds of the correlation matrices for the true TR-cores and three different levels of Gaussian noise added. Each algorithm is run 100 trials.
### TABLE 2  
Size and type of real datasets.

| Dataset      | Size                     | Type               |
|--------------|--------------------------|--------------------|
| DC mall<sup>b</sup> | 1280 × 307 × 191         | Hyperspectral image |
| Park bench<sup>b</sup> | 1080 × 1920 × 364       | Video              |
| Tabby cat<sup>c</sup> | 720 × 1280 × 286        | Video              |

<sup>a</sup> https://engineering.purdue.edu/~biehl/MultiSpec/.
<sup>b</sup> https://www.pexels.com/video/man-sitting-on-a-bench-853751.
<sup>c</sup> https://www.pexels.com/video/video-of-a-tabby-cat-854982/.

### TABLE 3  
Decompositions for real datasets with target rank $R = 3$.

| Method     | DC mall     | Park bench  | Tabby cat   |
|------------|-------------|-------------|-------------|
|            | Error       | Time (s)    | Error       | Time (s)    | Error       | Time (s)    |
| TR-ALS     | 0.331       | 54.6        | 0.183       | 519.2       | 0.189       | 188.7       |
| TR-ALS-NE  | 0.331       | 5.6         | 0.183       | 47.9        | 0.189       | 19.8        |
| TR-ALS-QR  | 0.331       | 2.8         | 0.183       | 33.3        | 0.189       | 12.2        |
| TR-ALS-QRNE| 0.331       | 2.8         | 0.183       | 33.3        | 0.189       | 12.1        |

### TABLE 4  
Decompositions for real datasets with target rank $R = 10$.

| Method     | DC mall     | Park bench  | Tabby cat   |
|------------|-------------|-------------|-------------|
|            | Error       | Time (s)    | Error       | Time (s)    | Error       | Time (s)    |
| TR-ALS     | 0.159       | 435.5       | 0.073       | 4114.4      | 0.134       | 1475.2      |
| TR-ALS-NE  | 0.159       | 10.8        | 0.073       | 75.8        | 0.134       | 31.3        |
| TR-ALS-QR  | 0.159       | 8.5         | 0.073       | 64.1        | 0.134       | 25.7        |
| TR-ALS-QRNE| 0.159       | 8.0         | 0.073       | 63.2        | 0.134       | 25.4        |

### 4.3  
Performance on image and video data

The brief information of three real image and video datasets is listed in Table 2. More specifically, DC Mall is a third-order tensor containing the hyperspectral image, whose first two orders are the image height and width, and the third one is the number of spectral bands. Park Bench and Tabby Cat are third-order tensors representing grayscale videos of a man sitting on a park bench and a tabby cat, respectively. The first two orders of them are the height and width of frames, and the third one is the number of frames. In addition, we also provide the links of the above data in footnotes.

#### 4.3.1  
Lower-order tensors

We implement our methods and TR-ALS on the data in Table 2 directly with different target rank $R$. The numerical results are summarized in Tables 3 and 4, respectively. From these two tables, we can see that our three algorithms can achieve the same errors as TR-ALS but take much less time, and TR-ALS-QRNE is the fastest method. In addition, we can also find that when running a same algorithm with different target ranks, different errors are obtained. This is mainly because the closer the target rank is to the true rank, the more accurate the result of the decomposition is. On the other hand, the larger the target rank is, the longer time it takes to run the algorithm.

To further clarify the above two findings, we choose the DC mall and a synthetic tensor generated as done in Experiment A-I to investigate the variation of errors and time of various algorithms when taking different target ranks. The numerical results are shown in Figure 6. It can be seen that as the rank increases, the time increases, and our three algorithms always outperform TR-ALS. If the rank is increased to a certain size, TR-ALS-NE is fastest but TR-ALS-QR
performs worst. This further validates the complexity analysis in Section 3. With respect to errors, they decrease when the rank increases for DC mall, while the results for the synthetic tensor show a V-shape. This is because the rank of DC Mall is larger than 20, while the synthetic tensor has the true rank 8. So, it is indeed that the errors are minimized when the target rank is equal to the true rank.

4.3.2 Higher-order tensors

The so-called higher-order tensors listed in Table 5 are truncated and reshaped from the data in Table 2. For example, DC Mall is first truncated to size 1280 × 306 × 190 and then reshaped into a 32 × 40 × 18 × 17 × 10 × 19 tensor. The numerical results of various algorithms with target rank \( R = 3 \) on these higher-order tensors are summarized in Table 6, where we see a similar finding to lower-order cases. That is, our algorithms have much better performance compared with TR-ALS. In addition, comparing Tables 6 with 3 shows that although the same target rank is used for the same dataset, the decompositions obtained from the reshaped tensors are less accurate than those obtained from the original tensors and the former also takes a longer time than the latter. This is mainly because the reshaped tensors change the structural information of the original data.

5 CONCLUDING REMARKS

In this paper, we propose three practical ALS-based algorithms for TR decomposition, that is, TR-ALS-NE, TR-ALS-QR, and TR-ALS-QRNE. They can make full use of the structure of the coefficient matrices of the TR-ALS subproblems.
To achieve this, we present a new property of the subchain product of tensors and extend the QR factorization of the matrix to the third-order tensor. Numerical results show that TR-ALS-NE can be much faster than the regular TR-ALS, and the QR-based methods are in turn more stable than TR-ALS-NE.

There are several potential performance improvements to pursue in future work. One is to exploit the structure of the subchain product of some third-order tensors whose mode-2 unfolding matrices are upper triangular. Another one is to speed up MTTSP, whose counterpart in CP decomposition is MTTKRP which has been investigated extensively. Moreover, it is also interesting to combine our methods with randomized techniques to further reduce computational costs.

**ACKNOWLEDGMENTS**

The authors would like to thank the editor and the anonymous reviewers for their detailed comments and helpful suggestions which helped considerably to improve the quality of the paper.

**CONFLICT OF INTEREST STATEMENT**

This study does not have any conflicts to disclose.

**DATA AVAILABILITY STATEMENT**

The data that support the findings of this study are available from the corresponding author upon reasonable request.

**ENDNOTE**

1Very recently, a similar technique for computing the coefficient matrix \((G_{i j k}^m)^\top G_{i j k}^m\) was proposed by Gao et al.,\(^{18}\) where the authors mainly investigated the Riemannian gradient descent and Riemannian conjugate gradient algorithms for tensor completion via TR decomposition. However, they considered the representation on elements to recursively compute \((G_{i j k}^m)^\top G_{i j k}^m\). In comparison, we have used the expression (3) and Proposition 3, and hence the derivation and representation of our result seem to be more concise and elegant in mathematics. In addition, the preliminary version of the present paper\(^{19}\) was completed earlier than.\(^{18}\)

**ORCID**

Hanyu Li https://orcid.org/0000-0001-5033-578X

**REFERENCES**

1. Zhao Q, Zhou G, Xie S, Zhang L, Cichocki A. Tensor Ring Decomposition. arXiv preprint arXiv:160605535 2016.
2. Khoromskij BN. \(O(d \log N)\)-quantics approximation of \(N\)-\(d\) tensors in high-dimensional numerical modeling. Constr Approx. 2011;34(2):257–80.
3. Affleck I, Kennedy T, Lieb EH, Tasaki H. Valence bond ground states in isotropic quantum antiferromagnets. Comm Math Phys. 1988;115(3):477–528.
4. Perez-Garcia D, Verstraete F, Wolf MM, Cirac JI. Matrix product state representations. Quantum Inform Comput. 2007;7(5-6):401–30.
5. Carroll JD, Chang JJ. Analysis of individual differences in multidimensional scaling via an \(N\)-way generalization of ”Eckart-young” decomposition. Psychometrika. 1970;35(3):283–319.
6. Harshman RA. Foundations of the PARAFAC procedure: models and conditions for an. UCLA Working Papers in Phonetics. 1970 16:1–84.
7. Tucker LR. Some mathematical notes on three-mode factor analysis. Psychometrika. 1966;31(3):279–311.
8. Oseledets IV. Tensor-train decomposition. SIAM J Sci Comput. 2011;33(5):2295–317.
9. Mickelin O, Karaman S. On algorithms for and computing with the tensor ring decomposition. Numer Linear Algebra Appl. 2020;27(3):e2289.
10. Yuan L, Li C, Cao J, Zhao Q. Randomized tensor ring decomposition and its application to large-scale data reconstruction. ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP). Brighton Conference Centre Brighton, UK: IEEE. 2019 2127–2131.
11. Ahmadi-Asl S, Cichocki A, Phan AH, Asante-Mensah MG, Ghazani MM, Tanaka T, et al. Randomized algorithms for fast computation of low rank tensor ring model. Mach Learn: Sci Technol. 2020;2(1):011001.
12. Malik OA, Becker S. A sampling-based method for tensor ring decomposition. Proceedings of the 38th international conference on machine learning. vol. 139. Virtual Event: PMLR. 2021 7400–7411.
13. Malik OA. More efficient sampling for tensor decomposition with worst-case guarantees. Proceedings of the 39th international conference on machine learning. vol. 162. Virtual Event: PMLR. 2022 14887–14917.
14. Yu Y, Li H. Practical Sketching-Based Randomized Tensor Ring Decomposition. arXiv preprint arXiv:220905647 2022.
15. Kolda TG, Bader BW. Tensor decompositions and applications. SIAM Rev. 2009;51(3):455–500.
16. Minster R, Viviano I, Liu X, Ballard G. CP decomposition for tensors via alternating least squares with QR decomposition. Numer Linear Algebra Appl. 2023;30(6):e2511.
17. Cichocki A, Lee N, Oseledets IV, Phan AH, Zhao Q, Mandic DP. Tensor networks for dimensionality reduction and large-scale optimization: part 1 low-rank tensor decompositions. Found Trends Mach Learn. 2016;9(4-5):249–429.
18. Gao B, Peng R, Yuan Y. Riemannian preconditioned algorithms for tensor completion via tensor ring decomposition. arXiv preprint arXiv:230214456 2023.
19. Yu Y, Li H. Practical Alternating Least Squares for Tensor Ring Decomposition. arXiv preprint arXiv:221011362. 2022.
20. Espig M, Hackbusch W, Khachatryan A. On the Convergence of Alternating Least Squares Optimisation in Tensor Format Representations. arXiv preprint arXiv:150600062 2015.
21. Chen Z, Li Y, Lu J. Tensor ring decomposition: optimization landscape and one-loop convergence of alternating least squares. SIAM J Matrix Anal Appl. 2020;41(3):1416–42.
22. Golub GH, Loan CFV. Matrix computations. Baltimore, Maryland: The Johns Hopkins University Press; 2013.
23. Bader BW, Kolda TG. Tensor toolbox for MATLAB. Version 3.2.1. 2021 https://www.tensortoolbox.org
24. Tomasi G, Bro R. A comparison of algorithms for fitting the PARAFAC model. Comput Statist Data Anal. 2006;50(7):1700–34.

**How to cite this article:** Yu Y, Li H. Practical alternating least squares for tensor ring decomposition. Numer Linear Algebra Appl. 2024;31(3):e2542. [https://doi.org/10.1002/nla.2542](https://doi.org/10.1002/nla.2542)