HOQRI: HIGHER-ORDER QR ITERATION FOR SCALABLE TUCKER DECOMPOSITION

Yuchen Sun and Kejun Huang

Department of CISE, University of Florida, Gainesville, FL 32611
(yuchen.sun,kejun.huang)@ufl.edu

ABSTRACT

We propose a new algorithm called higher-order QR iteration (HOQRI) for computing the Tucker decomposition of large and sparse tensors. Compared to the celebrated higher-order orthogonal iterations (HOOI), HOQRI relies on a simple orthogonalization step in each iteration rather than a more sophisticated singular value decomposition step as in HOOI. More importantly, when dealing with extremely large and sparse data tensors, HOQRI completely eliminates the intermediate memory explosion by defining a new sparse tensor operation called TTMcTC. Furthermore, HOQRI is shown to monotonically improve the objective function, thus enjoying the same convergence guarantee as that of HOOI. Numerical experiments on synthetic and real data showcase the effectiveness of HOQRI.

1. INTRODUCTION

A tensor is a data array indexed by three or more indices. It has been proven to be extremely useful in numerous applications [1]. Two of the fundamental tensor factorization models are the canonical polyadic decomposition (CPD) [2] and the Tucker decomposition [3]. In this paper, we propose a new algorithm called higher-order QR iteration (HOQRI) for Tucker decomposition, which is particularly suitable with large and sparse data tensors.

The Tucker decomposition is highly related to the higher-order SVD (HOSVD) and the best rank-$(K_1, \dots, K_N)$ approximation [4, 5]. The state-of-the-art algorithm for (approximately) computing the Tucker decomposition remains the higher-order orthogonal iteration (HOOI). However, HOOI is not without limitations when being used in practice. One general issue that exists for tensors of all sizes is that it requires a reliable subroutine to calculate the singular value decomposition (SVD) of a matrix—it would be nice if the algorithm does not rely on that in general. More importantly, when dealing with extremely large but sparse tensor data, HOOI generates very large and dense intermediate matrices. Sometimes the intermediate memory explosion exceeds the original data tensor, making the subsequent computations almost impossible to carry on [6, 7].

We aim to address both issues with the proposition of HOQRI. First of all, HOQRI defines a new tensor operation called TTMcTC that entirely eliminates the intermediate memory explosion. Furthermore, each iteration of HOQRI involves only an orthogonalization step, which can be efficiently (and explicitly) calculated using the QR factorization, a.k.a. Gram-Schmidt, rather than a sophisticated SVD step. Furthermore, we show that HOQRI guarantees monotonic improvement of the objective function, therefore its convergence is the same as that of HOOI. Numerical experiments on synthetic and real data showcase the effectiveness of HOQRI.

2. PRELIMINARIES

In this section, we define the notation used throughout the paper and also provide a brief overview of tensor factorization. For additional information on tensors and their factorizations, we direct the reader to the surveys by Kolda and Bader [8], Papalexakis et al. [9], and Sidiropoulos et al. [1].

2.1. Tensors and notations

We denote the input N-way tensor, of size $I_1 \times I_2 \times \cdots \times I_N$, as $\mathcal{X}$. In general, we denote tensors by boldface Euler script capital letters, e.g., $\mathcal{X}$ and $\mathcal{Y}$, while matrices and vectors are denoted by boldface italic capital letters (e.g., $\boldsymbol{X}$ and $\boldsymbol{Y}$) and boldface italic lowercase letters (e.g., $\boldsymbol{x}$ and $\boldsymbol{y}$), respectively. The Euclidean norm of a tensor $\mathcal{X}$ is denoted as $\|\mathcal{X}\|$, which is defined as

$$\|\mathcal{X}\| = \sqrt{\sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} \mathcal{X}(i_1, ..., i_N)^2}.$$

Unfolding. A tensor can be unfolded, or *matricized*, along any of its mode into a matrix. The tensor unfolding along the nth mode is denoted $\boldsymbol{X}_{(n)} \in \mathbb{R}^{I_n \times \prod_{v \neq n} I_v}$. More simply, the nth mode of $\mathcal{X}$ forms the rows of $\boldsymbol{X}_{(n)}$ and the remaining modes form the columns.

Tensor-matrix product. The n-mode tensor-matrix product multiplies a tensor with a matrix along the nth mode. Suppose $\boldsymbol{B}$ is a $K \times I_n$ matrix, the n-mode tensor-matrix product, denoted as $\mathcal{X} \times_n \boldsymbol{B}$, outputs a tensor of size $I_1 \times \cdots \times I_{n-1} \times K \times I_{n+1} \times \cdots \times I_N$. Elementwise,

$$\left[\mathcal{X} \times_n \boldsymbol{B}\right](i_1, ..., i_{n-1}, k, i_{n+1}, ..., i_N) = \sum_{i_n=1}^{I_N} \boldsymbol{B}(k, i_n) \mathcal{X}(i_1, ..., i_N).$$

Using mode-n unfolding, it can be equivalently written as

$$\left[\mathcal{X} \times_n \boldsymbol{B}\right]_{(n)} = \boldsymbol{B}\boldsymbol{X}_{(n)}.$$

Note that the resulting tensor is in general dense regardless of the sparsity pattern of $\mathcal{X}$.

A common task is to multiply a tensor by a set of matrices. This operation is called the tensor-times-matrix chain (TTMc). When multiplication is performed with all N modes, it is denoted as $\mathcal{X} \times \{\boldsymbol{B}\}$, where $\{\boldsymbol{B}\}$ is the set of N matrices $\boldsymbol{B}^{(1)}, ..., \boldsymbol{B}^{(N)}$. Sometimes the multiplication is performed with all modes *except one*. This is denoted as $\mathcal{X} \times_{-n} \{\boldsymbol{B}\}$, where n is the mode not being multiplied:

$$\mathcal{X} \times_{-n} \{\boldsymbol{B}\} = \mathcal{X} \times_1 \boldsymbol{B}^{(1)} \cdots \times_{n-1} \boldsymbol{B}^{(n-1)} \times_{n+1} \boldsymbol{B}^{(n+1)} \cdots \times_N \boldsymbol{B}^{(N)}.$$

Table 1: List of notations

Symbol	Definition
N	number of modes
$\mathcal{X}$	N-way data tensor of size $I_1 \times I_2 \times \cdots \times I_N$
$\mathcal{X}(i_1, ..., i_N)$	$(i_1, ..., i_N)$-th entry of $\mathcal{X}$
$\mathcal{I}(\mathcal{X})$	index set of all nonzero entries in $\mathcal{X}$
$\mathcal{I}_{i_n}^{(n)}(\mathcal{X})$	subset of $\mathcal{I}(\mathcal{X})$ where the n-mode index is i_n
$\boldsymbol{X}_{(n)}$	mode-n matrix unfolding of $\mathcal{X}$
I_n	dimension of the nth mode of $\mathcal{X}$
K_n	multilinear rank of the nth mode
$\mathcal{G}$	core tensor of the Tucker model $\in \mathbb{R}^{K_1 \times ... \times K_N}$
$\boldsymbol{U}^{(n)}$	mode-n factor of the Tucker model $\in \mathbb{R}^{I_n \times K_n}$
$\{\boldsymbol{U}\}$	set of all factors $\{\boldsymbol{U}^{(1)}, ..., \boldsymbol{U}^{(N)}\}$
$\times_n$	n-mode tensor-matrix product
$\times_{-n}$	chain of mode products except the nth one

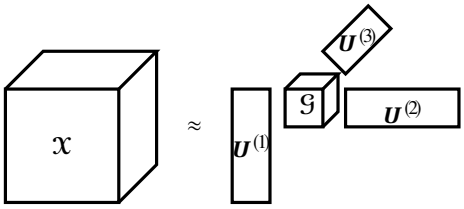

Fig. 1: Tucker decomposition of a 3-way tensor

Kronecker product. The Kroneckder product (KP) of $\boldsymbol{A} \in \mathbb{R}^{\ell \times m}$ and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, denoted as $\boldsymbol{A} \otimes \boldsymbol{B}$, is an $\ell p \times mq$ matrix defined as

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} \boldsymbol{A}(1,1)\boldsymbol{B} & \cdots & \boldsymbol{A}(1,m)\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}(\ell,1)\boldsymbol{B} & \cdots & \boldsymbol{A}(\ell,m)\boldsymbol{B} \end{bmatrix}.$$

Mathematically, the n-mode TTMc can be equivalently written as the product of mode-n unfolding times a chain of Kronecker products:

$$\left[\mathcal{X} \times_{-n} \{\boldsymbol{B}\}\right]_{(n)} = \boldsymbol{X}_{(n)} \left(\boldsymbol{B}^{(1)} \otimes \cdots \otimes \boldsymbol{B}^{(n-1)} \otimes \boldsymbol{B}^{(n+1)} \otimes \cdots \otimes \boldsymbol{B}^{(N)}\right)^{\top}. \tag{1}$$

More notations are shown in Table 1.

2.2. Tucker decomposition

The goal of Tucker decomposition is to approximate a data tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with the product of a core tensor $\mathcal{G} \in \mathbb{R}^{K_1 \times \cdots \times K_N}$ and a set of N factor matrices $\boldsymbol{U}^{(n)} \in \mathbb{R}^{I_n \times K_n}, n = 1, ..., N$, i.e., $\mathcal{X} \approx \mathcal{G} \times \{\boldsymbol{U}\}$. An illustration of Tucker decomposition for 3-way tensors is shown in Figure 1. To find the Tucker decomposition of a given tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and a target reduced dimension $K_1 \times \cdots \times K_N$, one formulates the following problem:

$$\underset{\substack{\mathcal{G} \in \mathbb{R}^{K_1 \times \cdots \times K_N} \\ \{\boldsymbol{U}^{(n)} \in \mathbb{R}^{I_n \times K_n}\}_{n=1}^{N}}}{\text{minimize}} \quad \|\mathcal{X} - \mathcal{G} \times \{\boldsymbol{U}\}\|^2. \tag{2}$$

Because of the rotational ambiguities, one can impose the constraints that the columns of each $\boldsymbol{U}^{(n)}$ are orthonormal without loss of generality, leading to the formulation:

$$\begin{aligned} \underset{\mathcal{G}, \{\boldsymbol{U}\}}{\text{minimize}} \quad & \|\mathcal{X} - \mathcal{G} \times \{\boldsymbol{U}\}\|^2 \\ \text{subject to} \quad & \boldsymbol{U}^{(n)\top}\boldsymbol{U}^{(n)} = \boldsymbol{I}, n = 1, ..., N. \end{aligned}$$

Algorithm 1 Higher-Order Orthogonal Iteration (HOOI)

1: initialize all $\boldsymbol{U}^{(n)}$ ▷ randomly or from HOSVD
2: **repeat**
3: **for** $n = 1, ..., N$ **do**
4: $\mathcal{Y} \leftarrow \mathcal{X} \times_{-n} \{\boldsymbol{U}^{\top}\}$
5: $\boldsymbol{U}^{(n)} \leftarrow K_n$ leading left singular vectors of $\boldsymbol{Y}_{(n)}$
6: **end for**
7: **until** convergence
8: $\mathcal{G} \leftarrow \mathcal{X} \times \{\boldsymbol{U}^{\top}\}$

It can be easily shown that the optimal $\mathcal{G}$ should be taken as $\mathcal{G} = \mathcal{X} \times \{\boldsymbol{U}\}$, and plugging it back leads to the simplified formulation [5]

$$\begin{aligned} \underset{\{\boldsymbol{U}\}}{\text{maximize}} \quad & \left\|\mathcal{X} \times \{\boldsymbol{U}^{\top}\}\right\|^2 \\ \text{subject to} \quad & \boldsymbol{U}^{(n)\top}\boldsymbol{U}^{(n)} = \boldsymbol{I}, n = 1, ..., N. \end{aligned} \tag{3}$$

A well-known algorithmic framework for approximately solving (3) is the higher-order orthogonal iteration (HOOI), which takes a block coordinate descent approach and updates each $\boldsymbol{U}^{(n)}$ in a cyclic fashion. Fixing all variables except $\boldsymbol{U}^{(n)}$, it is easy to show that a conditionally optimal update for $\boldsymbol{U}^{(n)}$ can be obtained by taking the K_n leading left singular vectors of $\boldsymbol{Y}_{(n)}$, which is obtained by taking the n-mode unfolding of the tensor $\mathcal{Y} \triangleq \mathcal{X} \times_{-n} \{\boldsymbol{U}^{\top}\}$. A detailed description of HOOI is given in Algorithm 1. Solving (3) exactly is NP-hard [10], but the HOOI algorithm is guaranteed to monotonically increase the objective of (3), and in a lot of cases find a good Tucker decomposition. When computationally viable, HOOI is usually initialized with the leading components of the higher-order SVD (HOSVD) of $\mathcal{X}$ [4].

3. PROBLEM STATEMENT

There are several computational issues regarding HOOI, especially when the data tensor $\mathcal{X}$ is large and sparse. The immediate issue people noticed was that a naive implementation to calculate $\mathcal{X} \times_{-n} \{\boldsymbol{U}^{\top}\}$ may require an enormous amount of memory due to intermediate dense data storage according to the explicit matrix product (1). This issue is somewhat resolved by exploiting the sparsity structure of the data tensor ensuring that the intermediate memory load is no more than the maximum of $\mathcal{X}$ and any of the $\mathcal{Y}$'s [6, 11, 12].

As the order of the tensor increases nowadays, it has been observed that even the amount of memory to record $\mathcal{Y}$ may not be practical for scalable Tucker decomposition. One way is to avoid explicitly materializing $\mathcal{Y}$ while still trying to find its leading singular vectors, as suggested by the S-HOT approach [7]. All the previous works pertain to the framework of HOOI, which requires computing the singular value decomposition (SVD) of some large and dense matrices as a sub-routine. This may be an issue as well, as there is not a single best way to implement SVD for all cases [13], and may require inner iterations to compute the dominant singular vectors. The memory complexities of these Tucker decomposition algorithms are compared in Table 2.

4. PROPOSED ALGORITHM: HOQRI

In this paper, we propose a new algorithmic framework for Tucker decomposition called higher-order QR iteration (HOQRI), which seamlessly resolves the issues we mentioned for HOOI: It is free from calling SVD, and its memory requirement is minimal (no larger

Table 2: Summary of per-iteration memory/computation complexities of various scalable Tucker decomposition algorithms. For simplicity, we assume $J_n = J$ and $K_n = K$. We calculate the *intermediate* memory requirement for all methods, assuming there are already spaces allocated for the data $\mathcal{X}$ and the solution $\mathcal{G}$ and $\{\boldsymbol{U}\}$, which takes space $\text{nnz}(\mathcal{X}) + NIK + K^N$.

	memory	computation
Kronecker product (1)	$I^{N-1}K^{N-1}$	$I^N K^{N-1} + IK^N$
MET [11] / TTMc [12]	IK^{N-1}	$\text{nnz}(\mathcal{X})K^{N-1} + IK^N$
S-HOT [7]	K^{N-1}	$\text{nnz}(\mathcal{X})K^{N-1} + IK^N$
HOQRI	IK	$\text{nnz}(\mathcal{X})K^N$

Algorithm 2 Higher-Order QR Iteration (HOQRI)

1: initialize all $\boldsymbol{U}^{(n)}$ ▷ randomly or from HOSVD
2: **repeat**
3: **for** $n = 1, ..., N$ **do**
4: $\mathcal{G} \leftarrow \mathcal{X} \times \{\boldsymbol{U}^\top\}$
5: $\boldsymbol{A}^{(n)} \leftarrow \text{TTMcTC}(\mathcal{X}, \{\boldsymbol{U}\}, \mathcal{G}, n)$
6: $\boldsymbol{U}^{(n)} \leftarrow$ an orthonormal basis of $\boldsymbol{A}^{(n)}$ via QR
7: **end for**
8: **until** convergence

than that of storing the solution $\{\boldsymbol{U}\}$ and $\mathcal{G}$). In addition, we also show that HOQRI converges similarly to HOOI.

4.1. Algorithm Description

An implementation of the proposed HOQRI is shown in Algorithm 2. As we can see, it is a very simple algorithm with a few but very important modifications to the existing HOOI. The only ambiguous part of Algorithm 2 is in line 5, where we define a new tensor-matrix operation called tensor times matrix chain times core (TTMcTC). Mathematically, it is simply the product of TTMc [12] and the corresponding matricized core-tensor:

$$\boldsymbol{A}^{(n)} = \boldsymbol{Y}_{(n)}\boldsymbol{G}_{(n)}^\top, \tag{4}$$

where $\boldsymbol{Y}_{(n)}$ is the mode-n matricization of the tensor $\mathcal{Y} = \mathcal{X} \times_{-n} \{\boldsymbol{U}^\top\}$ as in line 4 of HOOI, and $\boldsymbol{G}_{(n)}$ is the mode-n matricization of the core tensor $\mathcal{G}$. The reason this operation is described as a subroutine in Algorithm 2 is that the intermediate $\mathcal{Y}$ is never explicitly materialized when the data tensor is large and sparse—the result is directly calculated without additional memory overheads as will be explained in the sequel. This is the key step that renders the huge amount of savings in memory, thus making it most scalable.

The intuition behind the design of HOQRI is as follows. In line 5 of HOOI, we are supposed to calculate the K_n leading singular vectors of $\boldsymbol{Y}_{(n)}$, or equivalently, the K_n leading eigenvectors of $\boldsymbol{Y}_{(n)}\boldsymbol{Y}_{(n)}^\top$. One way of computing it is to apply the orthogonal iteration [13, §8.2.4] which iteratively calculates the matrix product $\boldsymbol{Y}_{(n)}\boldsymbol{Y}_{(n)}^\top\boldsymbol{U}^{(n)}$ and then orthogonalized the columns via the QR factorization. Notice that $\boldsymbol{Y}_{(n)}^\top\boldsymbol{U}^{(n)}$ effectively calculates the mode-n matricization of the core tensor $\mathcal{G}$. On the other hand, instead of applying it multiple times to converge to the K_n leading eigenvectors, we propose to calculate it only once per iteration, using the factor $\boldsymbol{U}^{(n)}$ obtained from the previous iteration. As we will see, this seemingly ad hoc modification not only greatly reduces the memory and computational overhead in each iteration, but the overall convergence is still guaranteed.

Algorithm 3 $\text{TTMcTC}(\mathcal{X}, \{\boldsymbol{U}\}, \mathcal{G}, n)$ element-wise

1: **for** $i_n = 1, ..., I_n$ **do**
2: **for** $k_n = 1, ..., K_n$ **do**
3: $$\boldsymbol{A}^{(n)}(i_n, k_n) = \sum_{i_{-n} \in \mathcal{I}_{i_n}^{(n)}(\mathcal{X})} \sum_{k_{-n}} \mathcal{X}(i_1, ..., i_N)\mathcal{G}(k_1, ..., k_N) \times \prod_{v \neq n} \boldsymbol{U}^{(v)}(i_v, k_v)$$
4: **end for**
5: **end for**
6: **return** $\boldsymbol{A}^{(n)}$

Algorithm 4 $\text{TTMcTC}(\mathcal{X}, \{\boldsymbol{U}\}, \mathcal{G}, n)$ matrix version

1: $\boldsymbol{A}^{(n)} \leftarrow 0$
2: **for** k_{-n} **do**
3: $\boldsymbol{a} = \mathcal{X} \times_{-n} \{\boldsymbol{u}_{k_1}, \ldots, \boldsymbol{u}_{k_N}\}$ ▷ sparse mode product
4: $\boldsymbol{g} = \boldsymbol{G}(k_1, \ldots, k_{n-1}, :, k_{n+1}, \ldots, k_N)$
5: $\boldsymbol{A}^{(n)} \leftarrow \boldsymbol{A} + \boldsymbol{a}\boldsymbol{g}^\top$
6: **end for**

In terms of per-iteration complexity, it is clear that it is dominated by the novel TTMcTC kernel. As will be explained in detail next, the complexity of TTMcTC is $O(\text{nnz}(\mathcal{X})K_1 \cdots K_N)$. As expected, it is K_n times higher than the computation of $\mathcal{Y}$ as in line 4 of HOOI. However, the benefit is that it does not require additional memory to store the large and dense tensor $\mathcal{Y}$. Moreover, HOOI requires calculating the K_n leading singular vectors of $\boldsymbol{Y}_{(n)}$, which in general requires $O(I_n K_1 \cdots K_N)$ complexity, while HOQRI follows with a simple QR factorization with $O(IK^2)$ flops, which is negligible compared to TTMcTC.

4.2. The TTMcTC kernel

In Algorithms 3 and 4, we provide two detailed implementations of TTMcTC that are efficient for a large and sparse data tensor $\mathcal{X}$. Mathematically they are no different from (4), but computationally they both result in the minimum number of flop counts by properly making use of the sparse data structure of the data tensor $\mathcal{X}$. Algorithm 3 computes the matrix $\boldsymbol{A}^{(n)}$ entry by entry, while Algorithm 4 computes it as a summation of vector outer products while calling the sparse tensor mode products as a subroutine. In either implementation, all the nonzero elements of $\mathcal{X}$ are exceed $K_1 \cdots K_N$ times in each iteration, leading to the $O(\text{nnz}(\mathcal{X})K_1 \cdots K_N)$ complexity.

5. OBJECTIVE CONVERGENCE

None of the existing Tucker decomposition algorithms has very strong convergence guarantees. The celebrated HOOI algorithm guarantees that the objective function (3) is monotonically increasing [5]; the factor matrices $\{\boldsymbol{U}\}$ are not guaranteed to converge to a stationary point. The convergence would be the same if we abandon the orthonormal constraints and solve the least squares formulation (2) using alternating least squares.

At first glance, it may not seem apparent that HOQRI also guarantees a monotonic increase of the objective function (3). Here we prove that the claim is in fact correct, thus the convergence of HOQRI is actually as strong as that of HOOI.

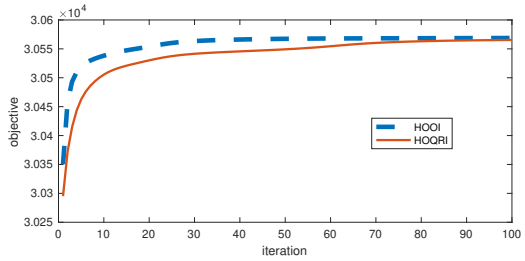

Fig. 2: One instance of convergence comparison on synthetic data.

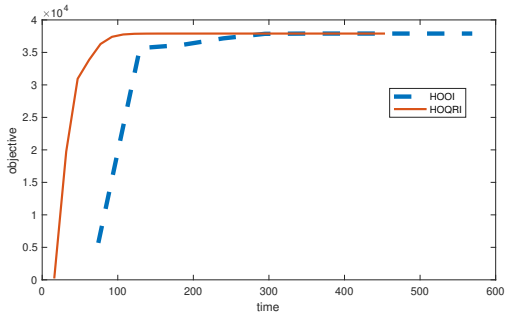

Fig. 3: One instance of convergence comparison on the Facebook wall posts data.

Theorem 1. *The objective function* (3) *is monotonically increasing under the HOQRI algorithm* (2).

Proof. The algorithm takes the general form of alternating updates for the factor $\boldsymbol{U}^{(n)}$ while fixing all the other factors $\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(n-1)}$, $\boldsymbol{U}^{(n+1)}, \ldots, \boldsymbol{U}^{(N)}$. The subproblem

$$\underset{\boldsymbol{U}}{\text{maximize}} \quad \left\|\boldsymbol{U}^\top \boldsymbol{Y}_{(n)}\right\|^2 \qquad \text{subject to } \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I} \tag{5}$$

is a convex quadratic maximization problem. For convex quadratics, the first-order Taylor approximation gives a global lower bound of the objective function. Thus, by maximizing the linear approximation of the objective function at the previous update $\boldsymbol{U}^{(n)}$, the objective function is guaranteed to be greater than or equal to the previous one. Define $\boldsymbol{A}^{(n)} = \boldsymbol{Y}_{(n)}\boldsymbol{G}_{(n)}^\top = \text{TTMcTC}(\mathcal{X}, \{\boldsymbol{U}\}, \mathcal{G}, n)$, the linearized problem becomes

$$\underset{\boldsymbol{U}}{\text{maximize}} \quad \text{Tr}\, \boldsymbol{A}^{(n)\top}\boldsymbol{U} \qquad \text{subject to } \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}.$$

The solution is given by the Procrustes rotation $\boldsymbol{Q}\boldsymbol{V}^\top$ [14], where $\boldsymbol{Q}$ and $\boldsymbol{V}$ are the left and right singular vectors of $\boldsymbol{A}^{(n)}$.

However, the Procrustes rotation is not the same as the update given by HOQRI as it is an arbitrary orthonormal basis for the column space of $\boldsymbol{A}^{(n)}$. To see that any orthonormal basis of $\boldsymbol{A}^{(n)}$ results in an increase of the objective function, we notice that the subproblem (5) is invariant under unitary rotation: for a unitary matrix $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{U}\boldsymbol{W}$ are both feasible to (5) and they results in the same objective value. The matrix $\boldsymbol{A}^{(n)}$ has full column rank, therefore any orthonormal basis of its columns is a unitary transformation of each other, including the one obtained from the Procrustes rotation $\boldsymbol{Q}\boldsymbol{V}^\top$. Meanwhile, the Procrustes rotation increases the objective value since it maximizes a global lower bound of (5). As a result the update from the QR factorization also increases the objective value. □

6. VALIDATION ON DENSE SYNTHETIC DATA

In this section, we illustrate the correctness of HOQRI on small and dense synthetic data. The purpose of this experiment is to demonstrate that HOQRI indeed enjoys a monotonic increase of the objective value (3) albeit it is a very simple algorithm that does not rely on a subroutine to calculate the SVD. We compare with the `tucker_als.m` function provided by the tensor toolbox in MATLAB [15]. As is the case on small and dense data, it is no surprise that HOOI converges slightly faster than HOQRI. The interesting observation is that the advantage that HOOI gains by calculating the exact SVD is not particularly significant, and as the iteration progresses HOQRI manages to achieve a similar objective value.

In this demonstration the size of the data tensor is $40 \times 50 \times 60$, with entries randomly drawn from i.i.d. standard normal distribution $\mathcal{N}(0, 1)$. As a result, the data tensor is almost surely full rank. We aim to find the best rank-$(10, 14, 13)$ approximation of the Tucker decomposition, with completely random initialization. A typical convergence (per-iteration) is shown in Figure 2. The horizontal axis shows iteration number instead of time elapsed since the purpose of this experiment is to showcase the correctness of the proposed HOQRI. Nonetheless, each iteration of HOQRI is about 2 to 3 times faster than that of HOOI. The real advantage of HOQRI reveals when dealing with large and sparse data tensors, as we show next.

7. REAL DATA EXPERIMENTS

Here we demonstrate the performance of HOQRI compared to HOOI on a sparse social network dataset: Facebook Wall Posts 8, of size $46952 \times 46951 \times 1592$, that collects the number of wall posts from one Facebook user to another over a period of 1592 days. The sparse tensor is stored in the sptensor format supported by the tensor toolbox [15]. The target multilinear rank for the Tucker approximation is $7 \times 8 \times 9$, and one instance of convergence is shown in Figure 3 with elapsed time as the horizontal axis. Notice that HOOI executed 10 iterations, while HOQRI runs 30 iterations, while it is clear that HOQRI converges to an approximate solution much faster than HOOI provided by the tensor toolbox.

The S-HOT approach [7] mitigates the intermediate memory explosion by applying Arnoldi's method to compute the leading singular vectors. The accompanying source code was implemented in C++, which would be an unfair comparison with our HOQRI purely implemented in MATLAB. In the upcoming journal version, we will implement HOQRI in SPLATT [16] and provide a comparison with S-HOT.

8. CONCLUSION

In this paper, we proposed a new algorithm for the tensor Tucker decomposition called higher-order QR iteration (HOQRI). Compared to the celebrated higher-order orthogonal iteration (HOOI), HOQRI does not require a separate subroutine for the singular value decomposition. More importantly, HOQRI entirely resolves the intermediate memory explosion when computing the Tucker decomposition of very large and sparse tensors by defining a new tensor operation TTMcTC. HOQRI is shown to monotonically increase the objective function, enjoying the same convergence guarantee as that of HOOI. Its outstanding performance is demonstrated on both synthetic and real data.

9. REFERENCES

[1] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582.

[2] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, 1927.

[3] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[4] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[5] ——, "On the best rank-1 and rank-(R_1,R_2,...,R_N) approximation of higher-order tensors," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.

[6] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *IEEE International Conference on Data Engineering*. IEEE, 2015, pp. 1047–1058.

[7] J. Oh, K. Shin, E. E. Papalexakis, C. Faloutsos, and H. Yu, "S-HOT: Scalable high-order Tucker decomposition," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017, pp. 761–770.

[8] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[9] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, "Tensors for data mining and data fusion: Models, applications, and scalable algorithms," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 2, p. 16, 2017.

[10] C. J. Hillar and L.-H. Lim, "Most tensor problems are NP-hard," *Journal of the ACM (JACM)*, vol. 60, no. 6, p. 45, 2013.

[11] T. G. Kolda and J. Sun, "Scalable tensor decompositions for multi-aspect data mining," in *IEEE International Conference on Data Mining*. IEEE, 2008, pp. 363–372.

[12] S. Smith and G. Karypis, "Accelerating the Tucker decomposition with compressed sparse tensors," in *European Conference on Parallel Processing*. Springer, 2017.

[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. JHU Press, 1996.

[14] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[15] B. W. Bader and T. G. Kolda, "Tensor toolbox for matlab, version 3.2.1," www.tensortoolbox.org, Apr. 2021.

[16] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," *29th IEEE International Parallel & Distributed Processing Symposium*, 2015.