SIAM J. MATH. DATA SCI.
Vol. 2, No. 1, pp. 189–215

Randomized Algorithms for Low-Rank Tensor Decompositions in the Tucker Format*

Rachel Minster†, Arvind K. Saibaba†, and Misha E. Kilmer‡

Abstract. Many applications in data science and scientific computing involve large-scale datasets that are expensive to store and manipulate. However, these datasets possess inherent multidimensional structure that can be exploited to compress and store the dataset in an appropriate tensor format. In recent years, randomized matrix methods have been used to efficiently and accurately compute low-rank matrix decompositions. Motivated by this success, we develop randomized algorithms for tensor decompositions in the Tucker representation. Specifically, we present randomized versions of two well-known compression algorithms, namely, HOSVD and STHOSVD, and a detailed probabilistic analysis of the error in using both algorithms. We also develop variants of these algorithms that tackle specific challenges posed by large-scale datasets. The first variant adaptively finds a low-rank representation satisfying a given tolerance, and it is beneficial when the target rank is not known in advance. The second variant preserves the structure of the original tensor and is beneficial for large sparse tensors that are difficult to load in memory. We consider several different datasets for our numerical experiments: synthetic test tensors and realistic applications such as the compression of facial image samples in the Olivetti database and word counts in the Enron email dataset.

Key words. randomized algorithms, tensors, Tucker decompositions, low-rank, multilinear algebra, structure-preserving

AMS subject classifications. 65F99, 15A69, 15A18, 15B52, 68W20, 65F15

DOI. 10.1137/19M1261043

1. Introduction.

Tensors, or multiway arrays, appear in a wide range of applications such as signal processing; neuroscientific applications such as electroencephalography; data mining; seismic data processing; machine learning applications such as facial recognition, handwriting digit classification, and latent semantic indexing; imaging; astronomy; and uncertainty quantification. For example, a database of gray-scale images constitutes a third order array when each image is stored as a matrix, while a numerical simulation of a system of partial differential equations (PDEs) in three-dimensional space when tracking several parameters over time yields a five-dimensional dataset. Often, these datasets are treated as matrices rather than as tensors, suggesting that additional structure that could be leveraged for gaining insight and lowering computational cost is often underutilized and undiscovered.

A key step in processing and studying these datasets involves a compression step either

*Received by the editors May 17, 2019; accepted for publication (in revised form) December 5, 2019; published electronically February 25, 2020.
https://doi.org/10.1137/19M1261043
Funding: The third author was supported by the National Science Foundation through the grant DMS-1821148. The first and second authors were supported by the National Science Foundation through the grant DMS-1821149.
†Department of Mathematics, North Carolina State University, Raleigh, NC 27695 (rlminste@ncsu.edu, asaibab@ncsu.edu).
‡Department of Mathematics, Tufts University, Medford, MA 02155 (misha.kilmer@tufts.edu).

to find an economical representation in memory, or to find principal directions of variability. While working with tensors there are many possible formats one may consider, and each format is equipped with a different notion of compression and rank. Examples of tensor formats include CANDECOMP/PARAFAC (CP), Tucker, hierarchical Tucker, and tensor train, all of which have their respective benefits (see the surveys [27, 18, 9, 10]). The CP format which represents a tensor as a sum of rank-1 outer products gives a compact and unique (under certain conditions) representation. Tucker generally finds a better fit for data by estimating the subspace of each mode, while hierarchical Tucker and tensor train are useful for very high-dimensional tensors, i.e., tensors with a large number of modes, since the cost of storing the tensor scales exponentially with the dimension of the tensor [18]. In this paper, we focus on the Tucker representation, which is known to have good compression properties. Given a multilinear rank $\mathbf{r}$, the Tucker form gives a representation of a tensor as a product of a core tensor and factor matrices typically having orthonormal columns. Popular algorithms for compression in the Tucker format can be found in [11, 40, 12], and a survey of approximation techniques can be found in [18]. Using these algorithms, high compression ratios can be achieved if the target rank for the Tucker approximation is small compared to the original dimensions. Even if the data is not highly compressible, representing it in the Tucker format can give insight into its principal directions [41].

In recent years, randomized matrix algorithms have gained popularity for developing low-rank matrix approximations (see the reviews [22, 29, 14]). These algorithms are easy to implement, are computationally efficient for a wide range of matrices (e.g., sparse matrices, matrices that can be accessed only via matrix-vector products, and dense matrices that are difficult to load in memory), and have accuracy comparable with nonrandomized algorithms. There is also well-developed error analysis applicable to several classes of random matrices for randomized algorithms. Even more recently, randomized algorithms have been developed for tensor decompositions (see below for a detailed review). Motivated by this success, we analyze existing randomized tensor algorithms and propose and analyze new randomized tensor algorithms.

Contributions and contents. In section 2, we first review the necessary background information on tensors and randomized algorithms. Then, in section 3, we present analyses of randomized versions of higher order SVD (HOSVD) and sequentially truncated higher order SVD (STHOSVD) (proposed in [46] and [7], respectively). Our contributions here include the probabilistic analysis of the randomized versions of these algorithms, as well as analysis of the associated computational costs. In section 4 we present adaptive randomized algorithms to compute low-rank tensor decompositions for use in applications where the target rank is not known beforehand. In section 5, we present a new randomized compression algorithm for large tensors, which produces a low-rank decomposition whose core tensor has entries taken directly from the tensor of interest. In this sense, the core tensor preserves the structure (e.g., sparsity, nonnegativity) of the original tensor. For sparse tensors, our algorithm has the added benefit that the intermediate and final decompositions can be stored efficiently, thus enabling the computation of low-rank tensor decompositions of large, sparse tensors. We also provide a probabilistic error analysis of this algorithm. Finally, in section 6, we test the performance of all algorithms on several synthetic tensors and real-world datasets and discuss the performance of the proposed bounds.

Related work. Several randomized algorithms have been proposed for computing low-rank tensor decompositions, e.g., Tucker format [7, 45, 28, 31, 39, 17, 46], CP format [17, 4, 5, 42], t-product [45], tensor networks [3], and tensor train format [7, 24]. Our work is most similar to [7, 17, 46]. The algorithm for randomized HOSVD is presented in [46], and the corresponding analysis is presented in [17] (both unpublished manuscripts). Randomized and adaptive versions of the STHOSVD were proposed and analyzed in [7], but our manuscript provides probabilistic error analysis for a different distribution of random matrices (see section 3 for a justification of our choice). To the best of our knowledge, our proposed algorithm for producing structure-preserving tensor decompositions and the corresponding error analysis are novel. Related to this algorithm is the CUR-type decomposition for tensors proposed in [33, 13]. In contrast, our algorithm produces decompositions in which the core tensor (rather than the factor matrices in the aforementioned references) retains entries from the original tensor. This allows for the decomposition of extremely large sparse tensors as the sparsity is maintained at each step of the algorithm.

2. Background.

In this section, we introduce the necessary background information for working with tensors and review the standard compression algorithms. We also discuss the optimal approximation of a tensor for comparison purposes. Finally, we review the relevant background for randomized matrix algorithms—specifically the randomized SVD.

2.1. Notation and preliminaries.

We denote a d-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with entries

$$x_{i_1,\ldots,i_d}, \qquad 1 \le i_j \le I_j, \qquad j = 1, \ldots, d.$$

A tensor can be "unfolded" into a matrix by reordering the elements, and this process is known as *matricization*. There are d different unfoldings for a d-mode tensor. Each mode-j unfolding arranges the resulting matrix such that the columns are the mode-j fibers of the tensor. The mode-j unfolding is denoted as $\mathbf{X}_{(j)} \in \mathbb{R}^{I_j \times (\prod_{k \neq j} I_k)}$ for $j = 1, \ldots, d$.

Tensor product. The tensor product (or mode product) is a fundamental operation for multiplying a tensor by a matrix. Given a matrix $\mathbf{A} \in \mathbb{R}^{K \times I_j}$, the mode-$j$ product of a tensor $\boldsymbol{\mathcal{X}}$ with $\mathbf{A}$ is denoted $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_j \mathbf{A}$ and has dimension $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \ldots I_{j-1} \times K \times I_{j+1} \times \cdots \times I_d}$. More specifically, the product can be expressed in terms of the entries of the tensor as

$$\boldsymbol{\mathcal{Y}}_{i_1,\ldots,i_{j-1},k,i_{j+1},\ldots,i_d} = \sum_{i_j=1}^{I_j} x_{i_1,\ldots,i_d} a_{k i_j}, \qquad 1 \le k \le K, \qquad j = 1, \ldots, d.$$

The tensor product can also be expressed as the product of two matrices. That is, we can write $\boldsymbol{\mathcal{Y}}_{(j)} = \mathbf{A}\mathbf{X}_{(j)}$ for $j = 1, \ldots, d$. Note that tensor products across distinct modes commute, and multiplying a tensor $\boldsymbol{\mathcal{X}}$ with d matrices $\mathbf{A}_j$, $j = 1, \ldots, d$, across modes $1, \ldots, d$, respectively, is written as $\boldsymbol{\mathcal{X}} \times_{j=1}^{d} \mathbf{A}_j$. The following lemma will be useful in our analysis.

Lemma 2.1. *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ and let $\boldsymbol{\Pi}_j \in \mathbb{R}^{I_j \times I_j}$ be a sequence of d orthogonal projectors. Then for $j = 1, 2, \ldots, d$,*

$$\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \bigtimes_{j=1}^{d} \boldsymbol{\Pi}_j\|_F^2 = \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}} \bigtimes_{i=1}^{j-1} \boldsymbol{\Pi}_i \times_j (\mathbf{I} - \boldsymbol{\Pi}_j)\|_F^2 \le \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_j \boldsymbol{\Pi}_j\|_F^2.$$

The proof of this lemma can be found in [40, Theorem 5.1].

Tucker representation. The Tucker format of a tensor $\boldsymbol{\mathcal{X}}$ of rank $(r_1, \ldots, r_d)$ consists of a core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times \cdots \times r_d}$ and factor matrices $\{\mathbf{A}_j\}_{j=1}^d$ with each $\mathbf{A}_j \in \mathbb{R}^{I_j \times r_j}$ such that $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \times_{j=1}^d \mathbf{A}_j$. We define the desired size of the core tensor, $(r_1, \ldots, r_d)$, as the target rank. For short, the Tucker representation is written as $[\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$.

Note that storing a tensor in Tucker form is beneficial as it requires less storage than a full tensor when the rank of the Tucker form is significantly less than the original dimension. For a d-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times I \times \cdots \times I}$ and target rank $(r, r, \ldots, r)$ with $r \ll I$, the cost of storing the Tucker form of $\boldsymbol{\mathcal{X}}$ is $\mathcal{O}(r^d + drI)$, compared to $\mathcal{O}(I^d)$ for a full tensor.

Kronecker products. The Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{k \times \ell}$ is

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{mk \times n\ell}.$$

We also note some properties of Kronecker products that will be useful in our analysis, namely

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}, \qquad (\mathbf{A} \otimes \mathbf{B})^\top = \mathbf{A}^\top \otimes \mathbf{B}^\top.$$

Kronecker products are also useful for expressing tensor mode products in terms of matrix-matrix multiplications. Suppose $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_{j=1}^d \mathbf{A}_j$; then

$$\mathbf{Y}_{(j)} = \mathbf{A}_j \mathbf{X}_{(j)} (\mathbf{A}_d^\top \otimes \mathbf{A}_{d-1}^\top \otimes \cdots \otimes \mathbf{A}_{j+1}^\top \otimes \mathbf{A}_{j-1}^\top \otimes \cdots \otimes \mathbf{A}_1^\top). \tag{2.1}$$

2.2. HOSVD/STHOSVD.

The *higher order SVD* (HOSVD) and *sequentially truncated higher order SVD* (STHOSVD) are two popular algorithms for computing low-rank tensor decompositions in the Tucker format. Note that, for these algorithms, the factor matrices $\mathbf{A}_j$ all have orthonormal columns.

HOSVD. In the HOSVD algorithm, each mode is handled separately. The factor matrix $\mathbf{A}_j$ is formed from the first r_j left singular vectors of $\mathbf{X}_{(j)}$. Once all factor matrices $\mathbf{A}_j$ are found, the core tensor is formed by $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_{j=1}^d \mathbf{A}_j^\top$. The error in approximating $\boldsymbol{\mathcal{X}}$ using the HOSVD is bounded by the sum of the error in each mode, as shown in the following theorem, taken from [40, Theorem 5.1].

Theorem 2.2. *Let $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ be the rank-$\mathbf{r}$ approximation to d-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ using the HOSVD algorithm. Then*

$$\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 \le \sum_{j=1}^d \|\boldsymbol{\mathcal{X}} \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F^2 = \sum_{j=1}^d \sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{X}_{(j)}).$$

This theorem says that the error in the rank-$\mathbf{r}$ approximation of the tensor $\boldsymbol{\mathcal{X}}$ computed using the HOSVD is the sum of squares of the discarded singular values from each mode unfolding. To simplify the upper bound, we introduce the notation

$$\Delta_j^2(\boldsymbol{\mathcal{X}}) \equiv \sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{X}_{(j)}), \qquad j = 1, \ldots, d. \tag{2.2}$$

With this notation, the error in the HOSVD satisfies $\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \le (\sum_{j=1}^d \Delta_j^2(\boldsymbol{\mathcal{X}}))^{1/2}$.

STHOSVD. An alternative to the HOSVD is the *sequentially truncated HOSVD* (STHOSVD) algorithm which also produces a compressed representation in the Tucker format. STHOSVD processes the modes sequentially, which makes the order in which the modes are processed important since we may obtain different approximations by using different orders.

At each stage, the core tensor $\mathcal{G}$ is unfolded, and the factor matrix $\mathbf{A}_j$ is formed by taking the first r_j left singular vectors. The new core tensor $\mathcal{G}^{(j)}$ is obtained by projecting the previous core tensor onto the subspace spanned by the columns of $\mathbf{A}_j$. We then have a jth partial approximation, defined as $\widehat{\mathcal{X}}^{(j)} = \mathcal{G}^{(j)} \times_{i=1}^{j} \mathbf{A}_i$.

The approximation error in this case is the sum of errors in the successive approximations and has the same upper bound as that of HOSVD. This is shown in the following theorem, which assumes that the processing order is $\rho = [1, 2, \ldots, d]$. If a different processing order is taken, the upper bound will remain the same, so this assumption is made for ease of notation. The proof of this theorem can be found in [40, Theorem 6.5].

Theorem 2.3. *Let $\widehat{\mathcal{X}} = [\mathcal{G}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ be the rank-$\mathbf{r}$ STHOSVD approximation to d-mode tensor $\mathcal{X}$ with processing order $\rho = [1, 2, \ldots, d]$. Then*

$$\|\mathcal{X} - \widehat{\mathcal{X}}\|_F^2 = \sum_{j=1}^{d} \|\widehat{\mathcal{X}}^{(j-1)} - \widehat{\mathcal{X}}^{(j)}\|_F^2 \leq \sum_{j=1}^{d} \|\mathcal{X} \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F^2 = \sum_{j=1}^{d} \Delta_j^2(\mathbf{X}_{(j)}).$$

The computational cost of the STHOSVD is lower than that of the HOSVD, which was established in [40] but is also reviewed in subsection 3.3. Although the error in the HOSVD and the error in the STHOSVD satisfy the same upper bound, it is not clear which algorithm has a lower error. There is strong numerical evidence to suggest that STHOSVD typically has a lower error, although counterexamples to this claim have been found [40]. For these reasons, STHOSVD is preferable to HOSVD since it has a lower cost and the same worst case error bound. A downside to STHOSVD is that the processing order ρ has to be determined in advance; some heuristics for this choice are given in [40].

2.3. Best approximation. We would like to find an optimal rank-$\mathbf{r}$ approximation of a given tensor $\mathcal{X}$, which we will denote as $\widehat{\mathcal{X}}_{\text{opt}}$. Let $\mathcal{S} = \{\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d} : \text{rank}(\mathbf{Y}_{(j)}) \leq r_j, j = 1, \ldots, d\}$. Then $\widehat{\mathcal{X}}_{\text{opt}}$ is an optimal tensor defined as satisfying the condition

$$\min_{\mathcal{Y} \in \mathcal{S}} \|\mathcal{X} - \mathcal{Y}\|_F = \|\mathcal{X} - \widehat{\mathcal{X}}_{\text{opt}}\|_F.$$

The Eckart–Young theorem [16] states that an optimal rank-r approximation to a matrix $\mathbf{A}$ can be constructed using the SVD truncated to rank r. Unfortunately, an analogue of this result for Tucker forms does not exist in higher dimensions [26]. The existence of $\widehat{\mathcal{X}}_{\text{opt}}$ is guaranteed by [21, Theorem 10.8]; however, this minimizer is not unique since Tucker representations are not unique [27, section 4.3]. In general, computing $\widehat{\mathcal{X}}_{\text{opt}}$ requires solving an optimization problem that has no closed-form solution. In [12], the higher order orthogonal iteration (HOOI) was proposed to compute the "best" approximation by generating a sequence of iterates by repeatedly cycling through the modes sequentially. Because the HOOI algorithm requires many iterations with the tensor $\mathcal{X}$, its implementation for large-scale tensors is

challenging because of the overwhelming computational cost. Although neither the HOSVD nor the STHOSVD produces an optimal rank-$\mathbf{r}$ approximation, they do satisfy the inequality

$$\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \sqrt{d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F. \tag{2.3}$$

The proofs are available for the HOSVD (Theorem 10.3) and STHOSVD (Theorem 10.5) in [21]. The proof requires the observation that

$$\Delta_j(\boldsymbol{\mathcal{X}}) \leq \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F, \qquad j = 1, \ldots, d. \tag{2.4}$$

We highlight this inequality since it will be important for our subsequent analysis. The inequality (2.3) suggests that the outputs of the HOSVD and the STHOSVD are accurate for low dimensions and can be employed in three different ways: as approximations to $\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}$, as starting guesses to the HOOI algorithm, or to fit CP models.

2.4. Randomized SVD. It is well known that computing the full SVD of a matrix costs $\mathcal{O}(mn^2)$, assuming $m \geq n$. When the dimensions of a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ are very large, the computational cost of a full SVD may be prohibitively expensive. Randomized SVD, popularized by [22], is a computationally efficient way to compute a rank-r approximation of $\mathbf{X}$. Assuming that $\mathbf{X}$ is approximately low-rank or has singular values that decay rapidly, the randomized SVD delivers a good low-rank representation of $\mathbf{X}$. Compared to the full SVD, the randomized SVD is much more computationally efficient across a wide range of matrices.

The randomized SVD algorithm has two distinct stages. In the first stage, or the range finding stage, we multiply a matrix $\mathbf{X}$ by a random matrix (we choose this to be Gaussian in this paper) in order to have random linear combinations of the columns of $\mathbf{X}$. We then take a thin QR factorization to obtain a matrix $\mathbf{Q}$ that gives a good approximation of the range of $\mathbf{X}$. This allows us to express $\mathbf{X} \approx \mathbf{Q}\mathbf{Q}^\top\mathbf{X}$. Then, in the second stage, or the postprocessing stage, we compute a thin SVD of the much smaller matrix $\mathbf{Q}^\top\mathbf{X} = \widehat{\mathbf{U}}_{\mathbf{B}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$, truncate down to the target rank r, and compute $\widehat{\mathbf{U}} = \mathbf{Q}\widehat{\mathbf{U}}_B$ to obtain the low-rank approximation $\widehat{\mathbf{X}} = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$. Algorithm 2.1 summarizes the process.

Algorithm 2.1. $[\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}}, \widehat{\mathbf{V}}] = \text{RandSVD}(\mathbf{X}, r, p, \mathbf{\Omega})$.

Input: matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, target rank r,
oversampling parameter $p \geq 0$ such that $r + p \leq \min\{m, n\}$,
Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{n\times(r+p)}$

Output: $\widehat{\mathbf{U}} \in \mathbb{R}^{m\times r}$, $\widehat{\mathbf{\Sigma}} \in \mathbb{R}^{r\times r}$, and $\widehat{\mathbf{V}} \in \mathbb{R}^{n\times r}$ such that $\mathbf{X} \approx \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$

1: Multiply $\mathbf{Y} \leftarrow \mathbf{X}\mathbf{\Omega}$
2: Thin QR factorization $\mathbf{Y} = \mathbf{Q}\mathbf{R}$
3: Form $\mathbf{B} \leftarrow \mathbf{Q}^\top\mathbf{X}$
4: Calculate thin SVD $\mathbf{B} = \widehat{\mathbf{U}}_{\mathbf{B}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$
5: Form $\widehat{\mathbf{U}} \leftarrow \mathbf{Q}\widehat{\mathbf{U}}_{\mathbf{B}}(:, 1:r)$
6: Compress $\widehat{\mathbf{\Sigma}} \leftarrow \widehat{\mathbf{\Sigma}}(1:r, 1:r)$, and $\widehat{\mathbf{V}} \leftarrow \widehat{\mathbf{V}}(:, 1:r)$

The computational cost of the algorithm is

$$\text{Cost} = 2(r+p)\mathcal{O}(\mathsf{nnz}(\mathbf{X})) + \mathcal{O}(r^2(m+n)),$$

where nnz denotes the number of nonzeros of $\mathbf{X}$.

An error bound for Algorithm 2.1 in the Frobenius norm is presented below, and we will use this result frequently in our analysis. This theorem can be found in [45, Theorem 3]. However, the proof appeared to have a couple of errors, which were easy to fix. We provide the proof in section SM1.

Theorem 2.4. *Let $\mathbf{X} \in \mathbb{R}^{m\times n}$, and let $\mathbf{\Omega} \in \mathbb{R}^{n\times(r+p)}$ be a Gaussian random matrix. Suppose $\mathbf{Q}$ is obtained from Algorithm 2.1 with inputs target rank $r \leq \mathsf{rank}(\mathbf{X})$ and oversampling parameter $p \geq 2$ such that $r + p \leq \min\{m, n\}$, and let $\mathbf{B}_r$ be the rank-r truncated SVD of $\mathbf{Q}^\top\mathbf{X}$. Then, the error in expectation satisfies*

$$\mathbb{E}_{\mathbf{\Omega}}\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top\mathbf{X}\|_F^2 \leq \mathbb{E}_{\mathbf{\Omega}}\|\mathbf{X} - \mathbf{Q}\mathbf{B}_r\|_F^2 \leq \left(1 + \frac{r}{p-1}\right)\sum_{j=r+1}^{\min\{m,n\}}\sigma_j^2(\mathbf{X}).$$

Remark 1. We will use two slightly different formulations of this theorem in our later results. Instead of $\|\mathbf{X} - \mathbf{Q}\mathbf{B}_r\|_F^2$, we will use $\|\mathbf{X} - \widehat{\mathbf{U}}\widehat{\mathbf{U}}^\top\mathbf{X}\|_F^2$. It is straightforward to show the equivalence between the two forms; see [34, section 5.3] for the explicit details. We will also use the following result, which uses Hölder's inequality [25, Theorem 23.10]:

$$\mathbb{E}_{\mathbf{\Omega}}\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top\widehat{\mathbf{X}}\|_F \leq \sqrt{1 + \frac{r}{p-1}}\left(\sum_{j=r+1}^{\min\{m,n\}}\sigma_j^2(\mathbf{X})\right)^{1/2}. \tag{2.5}$$

3. Randomized HOSVD/STHOSVD.

In this section, we review randomized algorithms that are modified versions of the HOSVD and STHOSVD. We also develop a rigorous error analysis and compare the two algorithms in terms of computational cost.

3.1. Algorithms.

To obtain the randomized version of HOSVD, first proposed in [46], a full SVD of each mode unfolding is replaced with a randomized SVD of each mode unfolding to construct the factor matrix. The procedure to compute the core tensor remains unchanged. This is reflected in Algorithm 3.1, and we call this the R-HOSVD algorithm. The randomized version of STHOSVD is obtained in a similar way; at each step, the SVD of the unfolded core tensor is replaced with a randomized SVD. This is shown in Algorithm 3.2 (we call this R-STHOSVD) and is similar to the algorithm proposed in [7]. We briefly comment on the choice of the random matrices $\{\mathbf{\Omega}_j\}_{j=1}^d$. The R-HOSVD proposed in [46] uses standard Gaussian random matrices, which we also adopt in this paper. The authors in [7] advocate for $\mathbf{\Omega}_j$ constructed as a Khatri–Rao product of Gaussian random matrices, which is less memory intensive compared to standard Gaussian random matrices. A recent article [37] also recommends the sparse Rademacher and scrambled subsampled randomized Fourier transform as appropriate choices for the random matrices. While the standard Gaussian random matrices may be criticized for large storage costs, we note that these matrices need not be stored explicitly and can be generated on the fly, either columnwise or in appropriately sized blocks.

3.2. Error analysis.

In the results below, we assume that the matrices $\{\mathbf{\Omega}_j\}_{j=1}^d$ are standard Gaussian random matrices of appropriate sizes. Here, we provide error bounds in expectation, but we can extend this analysis to develop concentration results that give insight into

Algorithm 3.1. Randomized HOSVD.

Input: d-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, target rank vector $\mathbf{r} \in \mathbb{N}^d$, oversampling parameter $p \geq 0$ satisfying (3.1)

Output: $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

1: **for** $j = 1 : d$ **do**
2: Draw random Gaussian matrix $\boldsymbol{\Omega}_j \in \mathbb{R}^{\prod_{i \neq j} I_i \times (r_j + p)}$
3: $[\widehat{\mathbf{U}}, \widehat{\boldsymbol{\Sigma}}, \widehat{\mathbf{V}}] = \text{RandSVD}(\mathbf{X}_{(j)}, r_j, p, \boldsymbol{\Omega}_j)$
4: Set $\mathbf{A}_j \leftarrow \widehat{\mathbf{U}}$
5: **end for**
6: Form $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_{j=1}^{d} \mathbf{A}_j^\top$

Algorithm 3.2. Randomized STHOSVD.

Input: d-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, processing order ρ, target rank vector $\mathbf{r} \in \mathbb{N}^d$, oversampling parameter $p \geq 0$ satisfying (3.5)

Output: $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

1: Set $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}}$
2: **for** $j = 1 : d$ **do**
3: Draw random Gaussian matrix $\boldsymbol{\Omega}_{\rho_j} \in \mathbb{R}^{z_j \times (r_{\rho_j} + p)}$, where z_j is as defined in (3.4)
4: $[\widehat{\mathbf{U}}, \widehat{\boldsymbol{\Sigma}}, \widehat{\mathbf{V}}] = \text{RandSVD}(\mathbf{G}_{(\rho_j)}, r_{\rho_j}, p, \boldsymbol{\Omega}_{\rho_j})$
5: Set $\mathbf{A}_{\rho_j} \leftarrow \widehat{\mathbf{U}}$
6: Update $\mathbf{G}_{(\rho_j)} \leftarrow \widehat{\boldsymbol{\Sigma}}\widehat{\mathbf{V}}^\top$ {Note: overwriting $\mathbf{G}_{(\rho_j)}$ overwrites $\boldsymbol{\mathcal{G}}$}.
7: **end for**
8: $\boldsymbol{\mathcal{G}} \leftarrow \mathbf{G}_{(\rho_d)}$, in tensor form.

the tail bounds. These can be obtained by combining our analysis with the results from, e.g., [19, Theorem 5.8].

Theorem 3.1 (randomized HOSVD). *Let $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ be the output of Algorithm 3.1 with inputs target rank $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ satisfying $r_j \leq \mathsf{rank}(\mathbf{X}_{(j)})$ for $j = 1, \ldots, d$ and oversampling parameter $p \geq 2$ satisfying*

$$r_j + p \leq \min\{I_j, \prod_{i \neq j} I_i\}, \qquad j = 1, \ldots, d. \tag{3.1}$$

Then, the expected error in the approximation satisfies

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left(\sum_{j=1}^{d} \left(1 + \frac{r_j}{p-1}\right) \Delta_j^2(\boldsymbol{\mathcal{X}}) \right)^{1/2} \tag{3.2}$$

$$\leq \left(d + \frac{\sum_{j=1}^{d} r_j}{p-1} \right)^{1/2} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{opt}\|_F. \tag{3.3}$$

Proof. Using Lemma 2.1, we can write

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}\|_F^2 \le \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\sum_{j=1}^d \|\boldsymbol{\mathcal{X}}\times_j(\mathbf{I}-\mathbf{A}_j\mathbf{A}_j^\top)\|_F^2 = \sum_{j=1}^d \mathbb{E}_{\boldsymbol{\Omega}_j}\|\boldsymbol{\mathcal{X}}\times_j(\mathbf{I}-\mathbf{A}_j\mathbf{A}_j^\top)\|_F^2,$$

where the equality comes from linearity of expectations and the independence of $\boldsymbol{\Omega}_j$ for each mode j. We can unfold each term in the summation as $\|\boldsymbol{\mathcal{X}}\times_j(\mathbf{I}-\mathbf{A}_j\mathbf{A}_j^\top)\|_F^2 = \|(\mathbf{I}-\mathbf{A}_j\mathbf{A}_j^\top)\mathbf{X}_{(j)}\|_F^2$. Then, by applying Theorem 2.4, we can bound the expected value of the squared error in each mode to obtain

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}\|_F^2 \le \sum_{j=1}^d\left(1+\frac{r_j}{p-1}\right)\Delta_j^2(\boldsymbol{\mathcal{X}}).$$

Finally, Hölder's inequality gives

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}\|_F \le \left(\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}\|_F^2\right)^{1/2} \le \left(\sum_{j=1}^d\left(1+\frac{r_j}{p-1}\right)\Delta_j^2(\boldsymbol{\mathcal{X}})\right)^{1/2}.$$

For the second inequality, recall that $\Delta_j^2(\boldsymbol{\mathcal{X}}) \le \|\boldsymbol{\mathcal{X}}-\hat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F^2$ from (2.4). Thus, combined with the previous inequality, we have

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}\|_F \le \left(d+\frac{\sum_{j=1}^d r_j}{p-1}\right)^{1/2}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

■

To compare this result to the approximation error obtained using the HOSVD algorithm, we consider a few special cases. Let $r=\max_{1\le j\le d} r_j$. Then, if $p=r+1$, these bounds take the form

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}\|_F \le \sqrt{2}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}_{\text{HOSVD}}\|_F \le \sqrt{2d}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

Similarly, if we choose $p=\lceil\frac{r}{\epsilon}\rceil+1$ for some $\epsilon>0$, the error satisfies

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}\|_F \le \sqrt{1+\epsilon}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}_{\text{HOSVD}}\|_F \le \sqrt{d(1+\epsilon)}\|\boldsymbol{\mathcal{X}}-\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

This shows that the application of a randomized SVD in each mode of the tensor does not seriously deteriorate the accuracy compared to using an SVD. Note that the computational costs of these choices are higher.

Now consider the randomized STHOSVD approximation. For the probabilistic error analysis, it is important to note that at each intermediate step, the partially truncated core tensor is a random tensor. This is in contrast to the R-HOSVD, where we only needed to account for $\boldsymbol{\Omega}_j$ for each mode because the operations are independent across the modes.

For this theorem, we use the same notation introduced in subsection 2.2 for the STHOSVD, in that the partially truncated core tensor at step j is $\boldsymbol{\mathcal{G}}^{(j)}=\boldsymbol{\mathcal{X}}\times_{i=1}^j\mathbf{A}_i^\top$, giving a partial approximation $\widehat{\boldsymbol{\mathcal{X}}}^{(j)}=\boldsymbol{\mathcal{G}}^{(j)}\times_{i=1}^j\mathbf{A}_i$. For convenience, given a processing order ρ we define

$$z_j=\left(\prod_{i=\rho_1}^{\rho_{j-1}} r_i\right)\left(\prod_{i=\rho_{j+1}}^{\rho_d} I_i\right), \qquad j=1,\dots,d. \tag{3.4}$$

Theorem 3.2 (randomized STHOSVD). *Let* $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ *be the output of Algorithm 3.2 with inputs target rank* $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ *satisfying* $r_j \leq \mathsf{rank}(\mathbf{G}_{(j)}^{(j-1)})$ *for* $j = 1, \ldots, d$*, processing order* ρ*, and oversampling parameter* $p \geq 2$ *and*

$$r_j + p \leq \min\{I_j, z_j\}, \qquad j = 1, \ldots, d. \tag{3.5}$$

Then, the approximation error in expectation satisfies

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left(\sum_{j=1}^{d} \left(1 + \frac{r_j}{p-1}\right) \Delta_j^2(\boldsymbol{\mathcal{X}}) \right)^{1/2} \tag{3.6}$$

$$\leq \left(d + \frac{\sum_{j=1}^d r_j}{p-1} \right)^{1/2} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{opt}\|_F. \tag{3.7}$$

Proof. We first assume that the processing order is $\rho = [1, \ldots, d]$ and then consider the general case. The first equality in Lemma 2.1 and the linearity of expectations together give

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 = \sum_{j=1}^{d} \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 = \sum_{j=1}^{d} \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^j} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2. \tag{3.8}$$

We have used the fact that the jth term in the summation does not depend on the random matrices $\{\boldsymbol{\Omega}_k\}_{k>j}$. We first consider $\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^j} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2$. Since all the $\boldsymbol{\Omega}_k$'s are independent, we can write the expectation in an iterated form as

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^j} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 = \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j-1}} \left\{ \mathbb{E}_{\boldsymbol{\Omega}_j} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 \right\}.$$

The jth term, which measures the difference in the sequential iterates, can be expressed as

$$\|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 = \|\boldsymbol{\mathcal{G}}^{(j-1)} \bigtimes_{i=1}^{j-1} \mathbf{A}_i \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F^2.$$

Now let

$$\mathbf{Z}_j \equiv \underbrace{\mathbf{I} \otimes \cdots \otimes \mathbf{I}}_{d-j \text{ terms}} \otimes \mathbf{A}_{j-1} \otimes \cdots \otimes \mathbf{A}_1.$$

If we unfold the difference $\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}$ along the jth mode, using (2.1) we have

$$\begin{aligned} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 &= \|(\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top) \mathbf{G}_{(j)}^{(j-1)} \mathbf{Z}_j^\top\|_F^2 \\ &\leq \|(\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top) \mathbf{G}_{(j)}^{(j-1)}\|_F^2. \end{aligned} \tag{3.9}$$

The inequality comes as $\mathbf{Z}_j$ has orthonormal columns for every j since the factor matrices $\mathbf{A}_j$ all have orthonormal columns.

Now, let $\mathbf{\Gamma}_j = \mathbf{G}_{(j)}^{(j-1)} = \mathbf{X}_{(j)}\mathbf{Z}_j$ for simplicity. We take expectations and bound this last quantity in (3.9) using Theorem 2.4 (keeping $\{\mathbf{\Omega}_k\}_{k=1}^{j-1}$ fixed), as

$$\mathbb{E}_{\mathbf{\Omega}_j}\|(\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\mathbf{\Gamma}_j\|_F^2 \leq \left(1 + \frac{r_j}{p-1}\right) \sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{\Gamma}_j). \tag{3.10}$$

We recall the definition and properties of Loewner partial ordering [23, section 7.7]. Let $\mathbf{M}, \mathbf{N} \in \mathbb{R}^{n\times n}$ be symmetric; $\mathbf{M} \preceq \mathbf{N}$ means $\mathbf{N} - \mathbf{M}$ is positive semidefinite. For $\mathbf{S} \in \mathbb{R}^{n\times m}$, then $\mathbf{S}^\top\mathbf{M}\mathbf{S} \preceq \mathbf{S}^\top\mathbf{N}\mathbf{S}$. Furthermore, $\lambda_i(\mathbf{M}) \leq \lambda_i(\mathbf{N})$ for $i = 1, \ldots, n$. Since $\mathbf{Z}_j$ has orthonormal columns, $\mathbf{Z}_j\mathbf{Z}_j^\top$ is a projector so that

$$\mathbf{\Gamma}_j\mathbf{\Gamma}_j^\top = \mathbf{X}_{(j)}\mathbf{Z}_j\mathbf{Z}_j^\top\mathbf{X}_{(j)}^\top \preceq \mathbf{X}_{(j)}\mathbf{X}_{(j)}^\top,$$

and the singular values of $\mathbf{\Gamma}_j$, which are squared eigenvalues of $\mathbf{\Gamma}_j\mathbf{\Gamma}_j^\top$, satisfy

$$\sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{\Gamma}_j) \leq \sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{X}_{(j)}) = \Delta_j^2(\boldsymbol{\mathcal{X}}). \tag{3.11}$$

To summarize, (3.8)–(3.11) combined give

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 \leq \sum_{j=1}^d \mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^{j-1}} \left(1 + \frac{r_j}{p-1}\right) \Delta_j^2(\boldsymbol{\mathcal{X}}) = \sum_{j=1}^d \left(1 + \frac{r_j}{p-1}\right) \Delta_j^2(\boldsymbol{\mathcal{X}}).$$

The equality follows since the tensor $\boldsymbol{\mathcal{X}}$ is deterministic. Finally, we have by Hölder's inequality and (2.2) that

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left(\sum_{j=1}^d \left(1 + \frac{r_j}{p-1}\right) \Delta_j^2(\boldsymbol{\mathcal{X}})\right)^{1/2} \leq \left(d + \frac{\sum_{j=1}^d r_j}{p-1}\right)^{1/2} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

In the general case, when the processing order does not equal $\rho = [1, \ldots, d]$, the proof is similar. We only need to work with the processed order, and we omit the details. ∎

We make several observations regarding Theorem 3.2. First, the upper bound for the error is the same for the R-STHOSVD as for the R-HOSVD (Theorem 3.1); however, once again, we note that the performance of the two algorithms may be different. Second, this result says that the upper bound for R-STHOSVD is independent of the processing order. This means that while some processing orders may result in more accurate decompositions, every processing order has the same worst case error bound. Our recommendation is to pick a processing order that minimizes the computational cost; see subsection 3.3 for details and Table 4 for numerical results. Third, the discussion following Theorem 3.1 regarding the choice of the oversampling parameter is applicable to the R-STHOSVD as well.

3.3. Computational cost. We discuss the computational costs of the proposed randomized algorithms and compare them against the HOSVD and the STHOSVD algorithms. We make the following assumptions. First, we assume that the tensors are dense and our implementations take no advantage of their structure. Second, we assume that the target ranks in each dimension are sufficiently small, i.e., $r_j \ll I_j$, so that we can neglect the computational cost of the QR factorization and the truncation steps of the RandSVD algorithm. Third, we assume that the random matrices used in the algorithms are standard Gaussian random matrices. If other distributions are used, the computational cost may be lower. Finally, for the STHOSVD and R-STHOSVD algorithms, we assume that the processing order is $\rho = [1, 2, \ldots, d]$.

The computational cost of both HOSVD and STHOSVD was discussed in [40] and is reproduced in Table 1. In this paper, we also provide an analysis of the computational cost of R-HOSVD and R-STHOSVD, which is summarized in Table 1. The table includes the costs for a general tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ with target rank $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ as well as for the special case when $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times I \times \cdots \times I}$ with target rank $\mathbf{r} = (r, r, \ldots, r)$. For ease of notation, denote the product $\prod_{k=i}^{j} I_k$ by $I_{i:j}$, and similarly $\prod_{k=i}^{j} r_k = r_{i:j}$ for $1 \le i \le j \le d$. The dominant costs of each algorithm lie in computing the SVD of the unfoldings (the first term in each summation) and forming the core tensor (the second term in each summation). We can see from Table 1 that the savings in randomizing both algorithms is roughly r/I. Since by assumption $r \ll I$, the randomized algorithms are expected to be much faster. See subsection 6.2 for experiments exploring this.

Table 1

Computational cost for the HOSVD, R-HOSVD, STHOSVD, and R-STHOSVD algorithms. The first term in each expression is the cost of computing an SVD of the mode unfoldings, and the second is the cost of forming the core tensor.

Algorithm	Cost for $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$	Cost for $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times \cdots \times I}$
HOSVD	$\mathcal{O}\left(\sum_{j=1}^{d} I_j I_{1:d} + \sum_{j=1}^{d} r_{1:j} I_{j:d}\right)$	$\mathcal{O}\left(dI^{d+1} + \sum_{j=1}^{d} r^j I^{d-j+1}\right)$
R-HOSVD	$\mathcal{O}\left(\sum_{j=1}^{d} r_j I_{1:d} + \sum_{j=1}^{d} r_{1:j} I_{j:d}\right)$	$\mathcal{O}\left(drI^{d} + \sum_{j=1}^{d} r^j I^{d-j+1}\right)$
STHOSVD	$\mathcal{O}\left(\sum_{j=1}^{d} I_j r_{1:j-1} I_{j:d} + \sum_{j=1}^{d} r_{1:j} I_{j+1:d}\right)$	$\mathcal{O}\left(\sum_{j=1}^{d} r^{j-1} I^{d-j+2} + r^j I^{d-j}\right)$
R-STHOSVD	$\mathcal{O}\left(\sum_{j=1}^{d} r_{1:j} I_{j:d} + \sum_{j=1}^{d} r_{1:j} I_{j+1:d}\right)$	$\mathcal{O}\left(\sum_{j=1}^{d} r^j I^{d-j+1} + r^j I^{d-j}\right)$

Processing order. The error in the approximation obtained using the R-STHOSVD depends on the choice of the processing order; see the numerical experiment in subsection 6.2. However, Theorem 3.2 suggests that the worst case error is independent of the processing mode. For this reason, we choose a processing order that minimizes the computational cost. Since the dominant cost at each step j is a randomized SVD with a cost of $\mathcal{O}(r_{\rho_1:\rho_j} I_{\rho_j:\rho_d})$, we can minimize this cost by choosing to process the modes in decreasing order of sizes; i.e., we process the largest modes first. Note that this is in contrast to the approach taken by [40] for the STHOSVD, in that they process in the order of increasing mode sizes to minimize the cost of the standard SVD at each step.

4. Adaptive randomized tensor decompositions. In the algorithms described in the previous section, we had to assume prior knowledge of the target rank. This knowledge may not be available, or may be difficult to estimate, in practice. Given a tensor $\boldsymbol{\mathcal{X}}$, it is often desirable to produce a decomposition $\widehat{\boldsymbol{\mathcal{X}}}$ that satisfies

$$\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \varepsilon \|\boldsymbol{\mathcal{X}}\|_F,$$

where $0 < \varepsilon < 1$ is a user-defined parameter. Note that there may not be a unique tensor $\widehat{\boldsymbol{\mathcal{X}}}$ that satisfies this inequality, but it is desirable to find a tensor with a small multirank that does satisfy this inequality. We first explain the adaptive randomized algorithm to find a low-rank matrix approximation and then explain how this can be extended to the tensor case.

Several adaptive randomized range finders for matrices have been proposed in the literature [22, 44]. Given a matrix $\mathbf{X}$ and a tolerance $\varepsilon > 0$, the goal is to find a matrix $\mathbf{Q}$ with orthonormal columns that satisfies

$$\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top \mathbf{X}\| \leq \varepsilon \|\mathbf{X}\|. \tag{4.1}$$

The number of columns of $\mathbf{Q}$ is taken to be the rank of the low-rank approximation. The adaptive algorithms begin with a small number of columns of the random matrix $\boldsymbol{\Omega}$ to estimate the range $\mathbf{Q}$ and then sequentially increase the number of columns of $\boldsymbol{\Omega}$ until the matrix $\mathbf{Q}$ satisfies (4.1). In our paper, we use a version of the adaptive randomized range finding algorithm first proposed by [32] and subsequently refined in [44, Algorithm 2]. We use [44, Algorithm 2] in our paper and assume that it can be invoked as $\mathbf{Q} = \text{AdaptRangeFinder}(\mathbf{X}, \varepsilon, b)$, where $\mathbf{X}$ is the matrix to be approximated, ε is the requested relative error tolerance, and b is a blocking integer to determine how many columns of $\boldsymbol{\Omega}$ to draw at a time.

Adaptive R-HOSVD. We now explain how the adaptive range finder can be used for computing tensor factorizations. For the R-HOSVD, we apply this adaptive matrix algorithm to each mode unfolding $\mathbf{X}_{(j)}$. This gives a factor matrix $\mathbf{A}_j$ for each mode $j = 1, \ldots, d$. Given some tolerance ε, the approximation error $\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \varepsilon \|\boldsymbol{\mathcal{X}}\|_F$ can be achieved if we choose the factor matrices $\mathbf{A}_j$ to satisfy

$$\|\mathbf{X}_{(j)} - \mathbf{A}_j \mathbf{A}_j^\top \mathbf{X}_{(j)}\|_F = \|\boldsymbol{\mathcal{X}} \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F \leq \varepsilon \|\boldsymbol{\mathcal{X}}\|_F / \sqrt{d}.$$

Thus, we have apportioned an equal amount of error tolerance to each mode unfolding. Combined with Lemma 2.1, this ensures that an overall relative error ε is achieved. This approach is summarized in Algorithm 4.1. Suppose we want a more flexible approach, in which a different tolerance ϵ_j is chosen for mode $j = 1, \ldots, d$. This may be necessary if we know in advance that we want to avoid compressing along some modes. In general, we may use any sequence ϵ_j, so long as it satisfies $(\sum_{j=1}^d \epsilon_j^2) = \varepsilon^2$. Indeed, setting $\epsilon_j = 0$ for selected modes ensures that no compression is performed across those modes. Note that the choice $\epsilon_j = \varepsilon/\sqrt{d}$ for $j = 1, \ldots, d$ automatically satisfies this equality.

Adaptive R-STHOSVD. The same approach can be extended to the R-STHOSVD algorithm. We define the intermediate tensors $\boldsymbol{\mathcal{X}}^{(j)} = \boldsymbol{\mathcal{X}} \times_{i=1}^{j} \mathbf{A}_i \mathbf{A}_i^\top$ for $j = 1, \ldots, d$ and $\boldsymbol{\mathcal{X}}^{(0)} = \boldsymbol{\mathcal{X}}$. Analogously, we define the intermediate core tensor $\boldsymbol{\mathcal{G}}^{(j)} = \boldsymbol{\mathcal{X}} \times_{i=1}^{j} \mathbf{A}_i^\top$ with

Algorithm 4.1. Adaptive R-HOSVD.

Input: d-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, tolerance $\varepsilon \geq 0$, blocking integer $b \geq 1$
Output: $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

1: **for** $j = 1 : d$ **do**
2: $\quad \mathbf{A}_j = \text{AdaptRangeFinder}(\mathbf{X}_{(j)}, \frac{\varepsilon}{\sqrt{d}}, b)$
3: **end for**
4: Form $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_{j=1}^{d} \mathbf{A}_j^\top$

$\boldsymbol{\mathcal{G}}^{(0)} = \boldsymbol{\mathcal{G}}$. Furthermore, we choose the processing order $\rho = [1, 2, \ldots, d]$ for simplicity. In this case, we choose the factor matrices $\mathbf{A}_j$ in order to ensure that the successive iterates satisfy

$$\|\boldsymbol{\mathcal{X}}^{(j-1)} - \boldsymbol{\mathcal{X}}^{(j)}\|_F = \|\boldsymbol{\mathcal{G}}^{(j-1)} \bigtimes_{i=1}^{j-1} \mathbf{A}_j \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F \leq \frac{\varepsilon}{\sqrt{d}} \|\boldsymbol{\mathcal{X}}\|_F, \quad j = 1, \ldots, d.$$

Applying the first part of Lemma 2.1, we obtain

$$\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}^{(d)}\|_F^2 = \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}}^{(j-1)} - \boldsymbol{\mathcal{X}}^{(j)}\|_F^2 \leq \varepsilon^2 \|\boldsymbol{\mathcal{X}}\|_F^2.$$

The details are provided in Algorithm 4.2, which uses a general processing order. As with R-HOSVD, we may elect to use the same error tolerance $\varepsilon/\sqrt{d}$ for each mode unfolding or use a different tolerance ϵ_j for iterate $j = 1, \ldots, d$.

Algorithm 4.2. Adaptive R-STHOSVD.

Input: d-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, processing order ρ, tolerance $\varepsilon \geq 0$, blocking integer $b \geq 1$
Output: $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

1: Set $\boldsymbol{\mathcal{G}} \leftarrow \boldsymbol{\mathcal{X}}$
2: **for** $j = 1 : d$ **do**
3: $\quad \mathbf{A}_{\rho_j} = \text{AdaptRangeFinder}(\mathbf{G}_{(\rho_j)}, \frac{\varepsilon}{\sqrt{d}}, b)$
4: $\quad$ Update $\mathbf{G}_{(\rho_j)} \leftarrow \mathbf{A}_{\rho_j}^\top \mathbf{G}_{(\rho_j)}$
5: **end for**
6: $\boldsymbol{\mathcal{G}} \leftarrow \mathbf{G}_{(\rho_d)}$, in tensor format

5. Structure-preserving decompositions.

In this section, we are interested in computing a low-rank decomposition in the Tucker format in which the core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times \cdots \times r_d}$ has entries that are explicitly taken from the original tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$. That is, $\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \times_{j=1}^{d} \mathbf{A}_j$ where $\mathbf{A}_j \in \mathbb{R}^{I_j \times r_j}$ with $j = 1, \ldots, d$ are the factor matrices (that do not necessarily have orthonormal columns). We call such a decomposition *structure-preserving* since the core tensor retains favorable properties (e.g., sparsity, nonnegativity, binary or integer counts) of the original tensor. This generalizes a related decomposition proposed for matrices in

[8]. Related to the structure-preserving decompositions, prior work includes a higher-order interpolatory decomposition [33, 13, 30] of the form

$$\mathcal{X} \approx \mathcal{G} \bigtimes_{j=1}^{d} \mathbf{C}_j, \qquad \mathcal{G} = \mathcal{X} \bigtimes_{j=1}^{d} \mathbf{C}_j^{\dagger},$$

where the matrices $\{\mathbf{C}_j\}_{j=1}^{d}$ have entries from the original tensor (specifically, columns selected from the appropriate mode-unfoldings), and † represents the Moore–Penrose pseudoinverse.

5.1. Algorithm. We first explain our algorithm for a matrix $\mathbf{X}$ and then generalize it to tensors. We compute a basis $\mathbf{Q}$ using the randomized range finding algorithm. Instead of computing the low-rank approximation $\mathbf{Q}\mathbf{Q}^{\top}\mathbf{X}$, as was done in Algorithm 2.1, we first identify a set of well-conditioned rows of $\mathbf{Q}$. This is implemented as a selection operator denoted by the matrix $\mathbf{P}$, which contains columns from the identity matrix. We then use the low-rank representation

$$\mathbf{X} \approx \mathbf{Q}(\mathbf{P}^{\top}\mathbf{Q})^{-1}\mathbf{P}^{\top}\mathbf{X} = \mathbf{A}\widehat{\mathbf{X}},$$

where $\mathbf{A} = \mathbf{Q}(\mathbf{P}^{\top}\mathbf{Q})^{-1}$ and $\widehat{\mathbf{X}} = \mathbf{P}^{\top}\mathbf{X}$. We see that the matrix $\mathbf{A}$ does not have orthonormal columns, but is well conditioned, and that the matrix $\widehat{\mathbf{X}}$ contains rows from the matrix $\mathbf{X}$ as determined by the selection operator $\mathbf{P}$. We use strong rank-revealing QR (sRRQR) factorization [20, 15] for subset selection; other possibilities are discussed in section SM2.

The idea behind our algorithms is the following: at each step, given the core tensor $\mathcal{G}$, we first unfold this tensor and use a randomized range finder on the unfolding to obtain a basis $\mathbf{Q}_j$. Then, we use the sRRQR algorithm to determine the selection operator that identifies well-conditioned rows of $\mathbf{Q}_j$; we use this same selection operator to select rows of $\mathbf{G}_{(j)}$ which then determines the core tensor for the next step. The details of the algorithm are provided in Algorithm 5.1. A few things are worth noting. First, the algorithm is structure-preserving in two ways: the core tensor at each step contains elements from the original tensor, so that the final core tensor also has entries from the original tensor; and the low-rank approximation reproduces certain elements of the tensor exactly (in exact arithmetic). Second, in contrast to Algorithms 3.1 and 3.2, the factor matrices do not have orthonormal columns; if orthonormal columns are desired, a postprocessing step can be performed (a thin QR factorization of each factor matrix to obtain the basis, followed by an aggregation step in which the core tensor is multiplied with all the triangular factors). Third, the resulting tensor is of rank $(r_1 + p, \ldots, r_d + p)$, which is also in contrast to other algorithms that produce decompositions of the rank $(r_1, \ldots, r_d)$. Once again, these factors can be recompressed using a postprocessing step; see, for example, [28].

Algorithm 5.1 is particularly beneficial for sparse tensors. Although sparse tensors can be efficiently stored in an appropriate tensor format (e.g., [2]), a straightforward application of either Algorithm 3.1 or Algorithm 3.2 produces dense intermediate tensors that may be prohibitively expensive to store, even though the final decomposition may be economical in terms of storage costs. On the other hand, in Algorithm 5.1, each intermediate core tensor is sparse and only contains entries from the original tensor. Therefore, the intermediary core tensors can be efficiently stored in the same sparse tensor format. Additionally, the sparse tensor format permits cheaper tensor and matrix product computations.

Algorithm 5.1. Structure-preserving STHOSVD (SP-STHOSVD).

Input: d-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, target rank vector $\mathbf{r} \in \mathbb{N}^d$, oversampling parameter $p \geq 0$ satisfying (5.2), processing order ρ

Output: $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

Set $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}}$
for $j = 1 : d$ **do**
  Draw Gaussian matrix $\boldsymbol{\Omega}_{\rho_j} \in \mathbb{R}^{z'_j \times (r_{\rho_j}+p)}$, where z'_j is as defined in (5.1)
  Form $\mathbf{Y} \leftarrow \mathbf{G}_{(\rho_j)}\boldsymbol{\Omega}_{\rho_j}$
  Thin QR factorization $\mathbf{Y} = \mathbf{Q}_{\rho_j}\mathbf{R}$
  Use strong RRQR on $\mathbf{Q}_{\rho_j}^\top$ with parameter $\eta = 2$,

$$\mathbf{Q}_{\rho_j}^\top \begin{bmatrix} \mathbf{S}_1 & \mathbf{S}_2 \end{bmatrix} = \mathbf{Z} \begin{bmatrix} \mathbf{N}_{11} & \mathbf{N}_{12} \end{bmatrix},$$

  where $\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{S}_2 \end{bmatrix}$ is a permutation matrix, $\mathbf{Z}$ is an orthogonal matrix, and $\mathbf{N}_{11}$ is upper triangular
  Let $\mathbf{P}_{\rho_j} = \mathbf{S}_1 \in \mathbb{R}^{I_{\rho_j} \times (r_{\rho_j}+p)}$ which contains the columns from the identity matrix
  Form $\mathbf{A}_{\rho_j} = \mathbf{Q}_{\rho_j}(\mathbf{P}_{\rho_j}^\top \mathbf{Q}_{\rho_j})^{-1}$
  Update $\mathbf{G}_{(\rho_j)} \leftarrow \mathbf{P}_{\rho_j}^\top \mathbf{G}_{(\rho_j)}$
end for
Set $\boldsymbol{\mathcal{G}} = \mathbf{G}_{(\rho_d)}$, in tensor format

Computational cost. For simplicity, we assume a processing order of $\rho = [1, 2, \ldots, d]$. The two dominant costs of Algorithm 5.1 for each mode are obtaining the basis $\mathbf{Q}_j$ and computing an sRRQR to determine $\mathbf{P}_j$. Let $\mathsf{nnz}(\boldsymbol{\mathcal{G}}^{(j)})$ denote the number of nonzeros in the core tensor at step j. Letting $\ell_j = r_j + p$, the cost of forming the product of the (unfolded) core tensor with a random matrix $\boldsymbol{\Omega}_j$ over all d modes is $\mathcal{O}(\sum_{j=1}^d \mathsf{nnz}(\boldsymbol{\mathcal{G}}^{(j)})\ell_j)$. Computing an sRRQR of an $I_j \times \ell_j$ matrix with parameter $\eta = 2$ (the parameter was called f in [20]) costs $\mathcal{O}(I_j \ell_j^2)$ per mode. Combining both the dominant costs gives a total cost of $\mathcal{O}(\sum_{j=1}^d \mathsf{nnz}(\boldsymbol{\mathcal{G}}^{(j)})\ell_j + \sum_{j=1}^d I_j \ell_j^2)$. This analysis shows that the computational cost of SP-STHOSVD is significantly smaller than the other algorithms presented thus far, particularly with a sparse tensor. Even when the original tensor is dense, there is a savings in computational cost compared to STHOSVD since a full SVD is not computed, and compared to R-STHOSVD since the core tensor $\boldsymbol{\mathcal{G}}^{(j)}$ is only multiplied once per iteration. We use the processing order in subsection 3.3.

5.2. Error analysis. We now present the error analysis for Algorithm 5.1. There are two major difficulties here in extending the proofs of Theorems 2.2 and 2.3. First, we have to work with an oblique projector $\boldsymbol{\Pi}_j = \mathbf{Q}_j(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1}\mathbf{P}_j^\top$, whereas in the previous analysis we used an orthogonal projector. As a consequence, we can no longer use the Pythagorean theorem to obtain Lemma 2.1; instead we have to employ the triangle inequality, resulting in a weaker bound. Second, we have to work with the factor matrices $\mathbf{A}_j$ which no longer have orthonormal columns. Let us define $\ell_i = r_i + p$, $g(I, r) = \sqrt{1 + 4r(I - r)}$ and $f_p(r) = \sqrt{1 + \frac{r}{p-1}}$, and given

a processing order ρ we also define

$$z'_j = \left(\prod_{i=\rho_1}^{\rho_{j-1}} \ell_i\right)\left(\prod_{i=\rho_{j+1}}^{\rho_d} I_i\right), \qquad j = 1, \ldots, d. \tag{5.1}$$

We are able to derive the following error bound.

Theorem 5.1. *Let $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ be the output of Algorithm 5.1 with inputs target rank $\mathbf{r} = (r_1, \ldots, r_d)$ satisfying $r_j \leq \mathsf{rank}(\mathbf{G}_{(j)}^{(j-1)})$ for $j = 1, \ldots, d$, a processing order of $\rho = [1, 2, \ldots, d]$, and oversampling parameter $p \geq 2$ satisfying*

$$r_j + p < \min\{I_j, z'_j\}, \qquad j = 1, \ldots, d. \tag{5.2}$$

Furthermore, let $\eta = 2$ be the sRRQR parameter. Then,

1. *the matrices $\{\mathbf{A}_j\}_{j=1}^d$ each contain an $\ell_j \times \ell_j$ identity matrix and*
$$1 \leq \|\mathbf{A}_j\|_2 \leq g(I_j, \ell_j), \qquad j = 1, \ldots, d,$$
2. *$\widehat{\boldsymbol{\mathcal{X}}}$ exactly reproduces $\prod_{i=1}^d \ell_i$ entries of the original tensor $\boldsymbol{\mathcal{X}}$ (in exact arithmetic), and*
3. *the expected approximation error satisfies*
$$\begin{aligned}\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F &\leq \sum_{j=1}^d \left(\prod_{k=1}^j g(I_k, \ell_k)\right) f_p(r_j)\Delta_j(\boldsymbol{\mathcal{X}}) \\ &\leq \sum_{j=1}^d \left(\prod_{k=1}^j g(I_k, \ell_k)\right) f_p(r_j)\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{opt}\|_F.\end{aligned}$$

Proof. We tackle each item individually.

1. Consider the factor matrices $\mathbf{A}_j = \mathbf{Q}_j(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1}$ for $j = 1, \ldots, d$. Since $\mathbf{P}_j$ contains columns from the identity matrix, it is easy to verify that $\mathbf{A}_j$ contains the identity matrix as its submatrix. The lower bound on $\|\mathbf{A}_j\|_2$ follows immediately from this fact. For the upper bound, since $\mathbf{Q}_j$ has orthonormal columns,

$$\|\mathbf{A}_j\|_2 = \|(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1}\|_2 \leq g(I_j, \ell_j). \tag{5.3}$$

The last step follows from [15, Lemma 2.1], where $\eta = 2$ is used as the tuning parameter for the sRRQR algorithm.

2. Let I_j be the chosen index set for each $\mathbf{P}_j$, $j = 1, \ldots, d$. Then, if we pick the corresponding elements from $\widehat{\boldsymbol{\mathcal{X}}}$, denoted by $\widehat{\boldsymbol{\mathcal{X}}}(I_1, \ldots, I_d)$, we have

$$\widehat{\boldsymbol{\mathcal{X}}}(I_1, \ldots, I_d) = \widehat{\boldsymbol{\mathcal{X}}} \bigtimes_{j=1}^d \mathbf{P}_j^\top = \left(\boldsymbol{\mathcal{G}} \bigtimes_{j=1}^d \mathbf{A}_j\right) \bigtimes_{j=1}^d \mathbf{P}_j^\top = \boldsymbol{\mathcal{G}}.$$

Note that the last equality comes as $\mathbf{P}_j^\top \mathbf{A}_j = (\mathbf{P}_j^\top \mathbf{Q}_j)(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1} = \mathbf{I}$ for $j = 1, \ldots, d$. Then as $\boldsymbol{\mathcal{G}}$ consists of elements of the original tensor $\boldsymbol{\mathcal{X}}$ by construction, $\widehat{\boldsymbol{\mathcal{X}}}$ reproduces $\prod_{j=1}^d \ell_j$ elements of $\boldsymbol{\mathcal{X}}$.

3. Next, let $\mathcal{G}^{(j)} = \mathcal{X} \times_{i=1}^{j} \mathbf{P}_i^\top$ denote the partially truncated core tensor after the jth step of Algorithm 5.1. Also let $\widehat{\mathcal{X}}^{(j)} = \mathcal{G}^{(j)} \times_{i=1}^{j} \mathbf{A}_i$ be the jth partial approximation of $\mathcal{X}$. Then, by the triangle inequality and the linearity of expectations, we have

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d} \|\mathcal{X} - \widehat{\mathcal{X}}\|_F \leq \sum_{j=1}^{d} \mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d} \|\widehat{\mathcal{X}}^{(j-1)} - \widehat{\mathcal{X}}^{(j)}\|_F. \tag{5.4}$$

Consider the term $\|\hat{\mathcal{X}}^{(j-1)} - \hat{\mathcal{X}}^{(j)}\|_F$. We can simplify $\hat{\mathcal{X}}^{(j)}$ as

$$\begin{aligned}\hat{\mathcal{X}}^{(j)} &= \mathcal{G}^{(j)} \bigtimes_{i=1}^{j} \mathbf{A}_i = \left(\mathcal{G}^{(j-1)} \times_j \mathbf{P}_j^\top\right) \bigtimes_{i=1}^{j} \mathbf{A}_i \\ &= \mathcal{G}^{(j-1)} \bigtimes_{i=1}^{j-1} \mathbf{A}_i \times_j \mathbf{A}_j \mathbf{P}_j^\top = \mathcal{G}^{(j-1)} \bigtimes_{i=1}^{j-1} \mathbf{A}_i \times_j \mathbf{\Pi}_j.\end{aligned}$$

Therefore, $\hat{\mathcal{X}}^{(j-1)} - \hat{\mathcal{X}}^{(j)} = \mathcal{G}^{(j-1)} \times_{i=1}^{j-1} \mathbf{A}_i \times_j (\mathbf{I} - \mathbf{\Pi}_j)$. Repeated use of the submultiplicativity inequality $\|\mathbf{MN}\|_F \leq \|\mathbf{M}\|_F \|\mathbf{N}\|_2$ gives

$$\|\hat{\mathcal{X}}^{(j-1)} - \hat{\mathcal{X}}^{(j)}\|_F \leq \|\mathcal{G}^{(j-1)} \times_j (\mathbf{I} - \mathbf{\Pi}_j)\|_F \prod_{i=1}^{j-1} \|\mathbf{A}_i\|_2. \tag{5.5}$$

Now, observe that $\mathbf{\Pi}_j \mathbf{Q}_j \mathbf{Q}_j^\top = \mathbf{Q}_j \mathbf{Q}_j^\top$, implying that $\mathbf{I} - \mathbf{\Pi}_j = (\mathbf{I} - \mathbf{\Pi}_j)(\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)$. Therefore, once again using the submultiplicativity inequality, we get

$$\begin{aligned}\|\mathcal{G}^{(j-1)} \times_j (\mathbf{I} - \mathbf{\Pi}_j)\|_F &= \|\mathcal{G}^{(j-1)} \times_j (\mathbf{I} - \mathbf{\Pi}_j)(\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)\|_F \\ &\leq \|\mathcal{G}^{(j-1)} \times_j (\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)\|_F \|\mathbf{I} - \mathbf{\Pi}_j\|_2.\end{aligned} \tag{5.6}$$

We note that $\mathbf{\Pi}_j \neq \mathbf{I}$ since $\mathsf{rank}(\mathbf{\Pi}_j) \leq \mathsf{rank}(\mathbf{Q}_j) = r_j + p < \min\{I_j, z_j'\}$, and $\mathbf{\Pi}_j \neq \mathbf{0}$ since $\mathbf{Q}_j$ has orthonormal columns and $\mathbf{P}_j^\top \mathbf{Q}_j$ is invertible. Therefore, we can use [38, Theorem 2.1] to conclude $\|\mathbf{I} - \mathbf{\Pi}_j\|_2 = \|\mathbf{\Pi}_j\|_2$. Once again, using [15, Lemma 2.1], we get

$$\|\mathbf{I} - \mathbf{\Pi}_j\|_2 = \|\mathbf{\Pi}_j\|_2 = \|(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1}\|_2 \leq g(I_j, \ell_j).$$

Combining this inequality with (5.3), (5.5), and (5.6), we have

$$\|\widehat{\mathcal{X}}^{(j)} - \widehat{\mathcal{X}}^{(j-1)}\|_F \leq \|\mathcal{G}^{(j-1)} \times_j (\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)\|_F \prod_{i=1}^{j} g(I_i, \ell_i).$$

Taking expectations, and using the independence of the random matrices, we obtain

$$\begin{aligned}\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d} \|\widehat{\mathcal{X}}^{(j)} - \widehat{\mathcal{X}}^{(j-1)}\|_F &= \mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^{j-1}} \mathbb{E}_{\mathbf{\Omega}_j} \|\mathcal{G}^{(j-1)} \times_j (\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)\|_F \prod_{i=1}^{j} g(I_i, \ell_i) \\ &= \mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^{j-1}} \mathbb{E}_{\mathbf{\Omega}_j} \|(\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top) \mathbf{G}_{(j)}^{(j-1)}\|_F \prod_{i=1}^{j} g(I_i, \ell_i) \\ &\leq \left(\prod_{i=1}^{j} g(I_i, \ell_i)\right) f_p(r_j) \mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^{j-1}} \left(\sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{G}_{(j)}^{(j-1)})\right)^{1/2}.\end{aligned}$$

In the last step, we have used (2.5) and kept the random matrices $\{\mathbf{\Omega}_k\}_{i=1}^{j-1}$ fixed. By construction $\mathbf{G}_{(j)}^{(j-1)}$ is a submatrix of the mode-unfolding $\mathbf{X}_{(j)}$. Arguing as in the proof of Theorem 3.2, we can show $\sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{G}_{(j)}^{(j-1)}) \leq \Delta_j^2(\boldsymbol{\mathcal{X}})$ for $j = 1, \ldots, d$ and, therefore,

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j-1)}\|_F \leq \left(\prod_{i=1}^{j} g(I_i, \ell_i)\right) f_p(r_j) \mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^{j-1}} \Delta_j(\boldsymbol{\mathcal{X}}).$$

Plugging this into (5.4) and using (2.4), we get the desired bound. ∎

In this result, we have assumed the standard processing order $\rho = [1, \ldots, d]$. The analysis can be extended to other processing orders; however, note that in this case the upper bound derived here will depend on the processing order, which is in contrast to the bound in Theorem 3.2. Although the error bound in Theorem 5.1 can be much higher than that in Theorem 3.2, numerical results suggest that the bound is somewhat pessimistic and, in practice, Algorithm 5.1 produces accurate low-rank decompositions.

6. Numerical results.

In this section, we study the accuracy and the computational cost of our algorithms on several synthetic and real-world tensors. Most of our results were run on a desktop with a 3.4 GHz Intel Core i7 processing unit and 16GB memory. A few experiments were run on the NCSU Mathematics Department HPC Cluster with 72GB memory (see supplementary materials). We used two tensor packages in MATLAB, namely Tensor Toolbox [2] and Tensorlab [43].

6.1. Test problems.

We briefly describe the different tensors that we use to validate our algorithms.

1. *Hilbert tensor.* Our first test tensor is a synthetic, supersymmetric tensor (invariant under the permutation of indices), where each entry is defined as

$$\boldsymbol{\mathcal{X}}_{i_1 i_2 \ldots i_d} = \frac{1}{i_1 + i_2 + \cdots + i_d}, \qquad 1 \leq i_j \leq I_j,\ j = 1, \ldots, d. \tag{6.1}$$

We call this the *Hilbert tensor*, which generalizes the Hilbert matrix ($d = 2$). When $d = 5$ and each $I_j = 25$, this tensor has $25^5 = 9,765,625$ nonzero entries. The singular values of each mode-unfolding decay rapidly, which suggests that the randomized algorithms proposed in this paper are likely to be accurate.

2. *Synthetic sparse tensor.* For our second example, we construct a three-dimensional sparse tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{200 \times 200 \times 200}$ as the sum of outer products as

$$\boldsymbol{\mathcal{X}} = \sum_{i=1}^{10} \frac{\gamma}{i^2}\, \mathbf{x}_i \circ \mathbf{y}_i \circ \mathbf{z}_i + \sum_{i=11}^{200} \frac{1}{i^2}\, \mathbf{x}_i \circ \mathbf{y}_i \circ \mathbf{z}_i, \tag{6.2}$$

where $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i \in \mathbb{R}^n$ are sparse vectors for all i, and $\circ$ denotes the outer product. The sparse vectors are all generated using the `sprand` command with 5% nonzeros each. In this instance, the tensor $\boldsymbol{\mathcal{X}}$ has $185,211$ nonzeros in total. Furthermore, γ is a user-defined parameter which determines the strength of the gap between the first ten terms and the last terms.

3. *Olivetti dataset.* The classification of facial images, or "tensorfaces" as popularized by [41], has two main steps. The first is a compression phase, where a higher order SVD is applied to a training images dataset, arranged as a tensor, to compute a low-rank decomposition. The second step is a classification phase, in which the decomposed tensor is used to classify images in the test dataset. We focus on the first step to efficiently decompose the tensor formed using training images from the Olivetti dataset [1]. This dataset contains 400 images (64×64 pixels) of 40 people in 10 different poses and can be expressed as a three-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{40\times 4096\times 10}$, in which the three modes represent people, pixels, and poses, respectively.

4. *FROSTT database.* Our final test problems come from the formidable repository of sparse tensors and tools (FROSTT) database [36]. From this database, we choose two representative large, sparse tensors whose features are summarized in Table 2. The NELL-2 dataset [6] is a portion of the Never Ending Language Learning knowledge base from the "Read the Web" project at Carnegie Mellon University. NELL is a machine learning system that relates different entities, creating a three-dimensional dataset whose modes represent entity, relation, and entity. The Enron dataset [35] contains word counts in emails released during an investigation by the Federal Energy Regulatory Commission. Here, the modes represent sender, receiver, word, and date, respectively.

Table 2

Summary of sparse tensor examples from the FROSTT database—we include the details for both the full datasets and the condensed datasets used in our experiments.

Original tensor	Order	Size	Nonzeros
NELL-2	3	$12092 \times 9184 \times 28818$	$76,879,419$
Enron	4	$6066 \times 5699 \times 244268 \times 1176$	$54,202,099$
Condensed tensor	**Order**	**Size**	**Nonzeros**
NELL-2	3	$807 \times 613 \times 1922$	$19,841$
Enron	3	$405 \times 380 \times 9771$	$6,131$

Although our implementation of SP-STHOSVD is capable of handling both full tensors in Table 2, the tensors are too large to compute the approximation error. In order to be able to compute this error, we first pared down the tensors, which allows us to compare the performance of our SP-STHOSVD algorithm with other algorithms. For the NELL-2 dataset [6], we subsample every 15 elements in each mode to obtain a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{807\times 613\times 1922}$. For the Enron dataset [35], we first condense the dataset to three dimensions by summing over the fourth mode. Then we subsample this tensor by taking every 15 elements from the first two modes and every 25 from the third mode. This results in a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{405\times 380\times 9771}$.

6.2. Numerical experiments. We now describe the experiments performed on the test tensors introduced in the previous subsection.

6.2.1. Fixed rank. Our first experiment compares the accuracy of the HOSVD and STHOSVD algorithms with their randomized counterparts, R-HOSVD and R-STHOSVD (Algorithms 3.1 and 3.2). As inputs, we take $\boldsymbol{\mathcal{X}}$ as defined in (6.1) with $d = 5$ modes and $I_j = 25$ for $j = 1, \ldots, d$. For each algorithm, we use the target rank (r, r, r, r, r), where r varies from 1 to 25, and the same oversampling parameter $p = 5$ was used in every mode. Since this is a supersymmetric tensor, the processing order of modes does not affect the results, so we

take the processing order $\rho = [1, 2, 3, 4, 5]$. The relative error is plotted in Figure 1, where we can see that the approximation error of all four algorithms is very similar and that the randomized algorithms are also highly accurate. The comparison of the cost in subsection 3.3 implies that the proposed randomized algorithms are less expensive compared to their deterministic counterparts. To illustrate this, we report the runtime of the HOSVD, R-HOSVD, STHOSVD, and R-STHOSVD algorithms on $\boldsymbol{\mathcal{X}}$ as the size of each dimension increases. For inputs, we fixed the target rank to be $(5, 5, 5, 5, 5)$ and the oversampling parameter as $p = 5$, and we used processing order $\rho = [1, 2, 3, 4, 5]$ in the sequential algorithms. The runtime in seconds, averaged over three runs, is shown in Table 3. The analysis of the computational cost implies that the randomized algorithms should be a factor of $I/(r + p) = 2.5$ faster than the nonrandomized algorithms. This is evident in our results. Also, the sequential algorithms are significantly faster than the HOSVD/R-HOSVD algorithms.

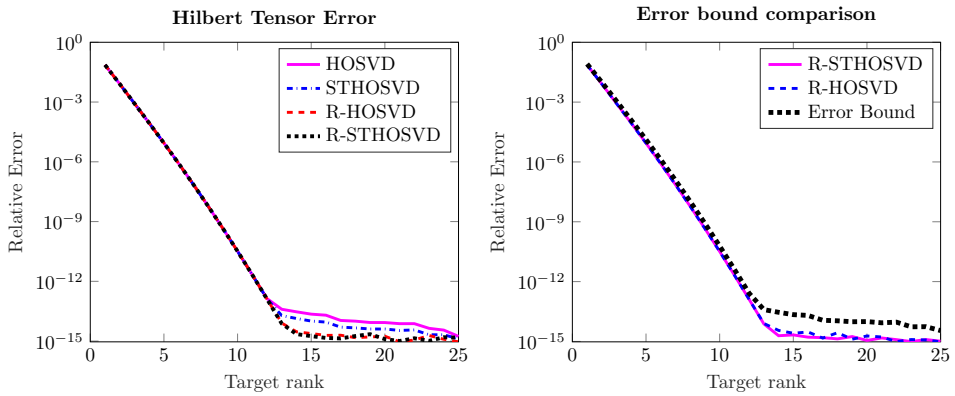

Figure 1. Left: *Relative approximation error for 5-mode Hilbert tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{25\times25\times25\times25\times25}$ *defined in* (6.1)*, with target rank* (r, r, r, r, r) *and oversampling parameter* $p = 5$. Right: *Actual relative error for* $\boldsymbol{\mathcal{X}}$ *from the R-HOSVD and R-STHOSVD algorithms compared to the calculated error bound as the target rank* (r, r, r, r, r) *increases. Both algorithms use oversampling parameter* $p = 5$*, and R-STHOSVD uses the processing order* $\rho = [1, 2, 3, 4, 5]$.

Table 3

Runtime in seconds of the HOSVD, R-HOSVD, STHOSVD, and R-STHOSVD algorithms on the Hilbert tensor $\boldsymbol{\mathcal{X}}$ *with* $d = 5$*, averaged over three runs. Each algorithm is run with target rank* $(5, 5, 5, 5, 5)$*, and the randomized algorithms use oversampling parameter* $p = 5$*. The STHOSVD and R-STHOSVD algorithms use the processing order* $\rho = [1, 2, 3, 4, 5]$.

		HOSVD	R-HOSVD	STHOSVD	R-STHOSVD
	25	3.0030	1.2609	0.6455	0.2875
I_j	35	14.5255	5.5288	3.1974	1.2090
$1 \le j \le d$	45	70.0536	17.3744	15.0629	3.5587
	50	101.4981	27.7725	21.9745	5.5606

We now compare the numerical performance of the R-HOSVD and R-STHOSVD algorithms to the theoretical bounds derived in Theorems 3.1 and 3.2. Since the upper bound obtained using both theorems is the same (see Theorems 3.1 and 3.2), we display this bound only once. We run both algorithms with increasing target rank (r, r, r, r, r), oversampling

parameter $p = 5$, and processing order $\rho = [1, 2, 3, 4, 5]$. From the right panel of Figure 1, we can see the error in STHOSVD and R-STHOSVD are both comparable to the theoretical bound, which shows that the analysis captures the behavior of the error quite well.

We now present an experiment that justifies the choice of processing order described in subsection 3.3. We consider the Olivetti dataset described in subsection 6.1. In Table 4, we compare the relative error and runtime of R-STHOSVD, averaged over three runs, for all six different processing orders ρ. To account for the randomness, we set the same initial seed for each run, take the target rank as $(20, 40, 5)$, and take the oversampling parameter as $p = 5$. We can see that all processing orders have comparable relative errors, so we instead focus on the runtime. Our heuristic, which involves processing the modes in decreasing size per mode, has the shortest average time as expected from the analysis in subsection 3.3. It is clear from the runtime that processing the largest mode first (mode 2 in this case) had the fastest runtime, and processing the smallest mode first had the slowest runtime.

Table 4

Relative error and average runtime in seconds of R-STHOSVD on the Olivetti dataset for different processing orders. The runtime was averaged over three runs, and the R-STHOSVD used a target rank of $(20, 40, 5)$ *and an oversampling parameter of* $p = 5$ *as inputs.*

Processing order	$[1, 2, 3]$	$[1, 3, 2]$	$[2, 1, 3]$	$[2, 3, 1]$	$[3, 1, 2]$	$[3, 2, 1]$
Relative error	0.1652	0.1608	0.1669	0.1633	0.1576	0.1602
Average time (sec)	0.0774	0.0933	0.0271	0.0246	0.1094	0.0952

6.2.2. Adaptive algorithms. We now demonstrate the performance of the adaptive algorithms, Algorithms 4.1 and 4.2, on the Olivetti dataset, with a processing order $\rho = [2, 1, 3]$. We initialize the adaptive R-STHOSVD algorithm with different desired relative tolerances ϵ and obtain an approximate decomposition. In Table 5, we display the rank of this decomposition $\mathbf{r}$ (the size of the core tensor) as well as the actual relative error of the decomposition. In each instance, we observe that the resulting error of the decomposition is lower than the desired error. Next, with $\mathbf{r}$ as the target rank, we compute a low-rank decomposition using STHOSVD with the same processing order. The resulting error of this decomposition is only slightly smaller than the error in the adaptive algorithm, suggesting that the adaptive algorithm is capable of adaptively estimating the target rank. Numerical experiments for adaptive R-HOSVD are presented in the supplementary materials.

Next, we consider the relative error obtained by only compressing two modes of the tensor, namely the people and pixels (modes 1 and 2, respectively), formed using the Olivetti dataset. We only present results corresponding to the sequentially truncated algorithms here in the form of "heat" plots which display the relative error as a function of the target rank; see Figure 2. In general, we see that the error decreases with increasing rank for both the randomized and deterministic algorithms. We also compare the performance of the adaptive randomized STHOSVD algorithm (Algorithm 4.2) by displaying the target rank obtained for a given relative error ϵ. For all the fixed rank algorithms discussed above, the oversampling parameter was $p = 5$, and the processing order was $\rho = [2, 1, *]$. The third mode is not compressed, indicated here by the asterisk.

Table 5

*A comparison of the adaptive R-STHOSVD algorithm (Algorithm 4.2) to STHOSVD. We first obtained the rank of the core tensor with the requested relative error tolerance from the adaptive algorithm. Then we compared the actual error of the approximation from the adaptive R-STHOSVD to that of an STHOSVD with the same rank. The processing order for all runs was $\rho = [2, 1, 3]$. *Each STHOSVD was computed with the corresponding rank found in the second column.*

Error tolerance ϵ	Corresponding rank $\mathbf{r}$	Actual error	Rank-$\mathbf{r}$ STHOSVD error*
0.25	$(3, 10, 1)$	0.1995	0.1995
0.2	$(10, 23, 1)$	0.1799	0.1796
0.15	$(22, 51, 5)$	0.1421	0.1403
0.1	$(32, 114, 8)$	0.0965	0.0946
0.05	$(38, 237, 10)$	0.0400	0.0381
0.01	$(40, 381, 10)$	0.0057	0.0055

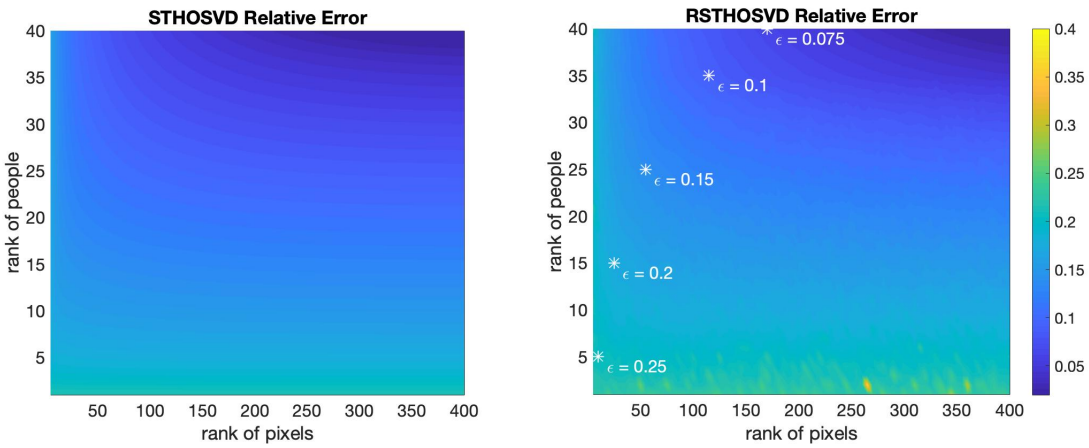

Figure 2. *Relative error for $\mathcal{X}$ as target rank increases using the STHOSVD, compressing pixels and people (modes 2 and 1), plotted with rank given by the Adaptive R-STHOSVD (Algorithm 4.2) with the desired relative error tolerance ϵ. The processing order was $\rho = [2, 1, *]$, and the oversampling parameter was $p = 5$.*

6.2.3. Algorithms for sparse tensors. We now test our randomized algorithms on sparse tensors. First, consider the synthetic sparse tensor $\mathcal{X}$ defined in (6.2). For three different γ values $\gamma = 2, 10, 200$, we compare the SP-STHOSVD algorithm to the STHOSVD and R-STHOSVD algorithms by plotting the relative error as the target rank (r, r, r) increases. Note that we are only comparing to the sequentially truncated algorithms in Figure 3, since they have lower cost and comparable errors. As inputs to our test algorithms, we used oversampling parameter $p = 5$ and processing order $\rho = [1, 2, 3]$. We can see that the error for the sparse algorithm is only slightly higher for smaller values of r, suggesting that the error analysis in Theorem 5.1 may be pessimistic.

We now test our SP-STHOSVD algorithm on the real-world sparse tensors. Note that this algorithm does not have a truncation step, which means that the rank of the resulting approximation given target rank (r, r, r) will be $(r+p, r+p, r+p)$. To compare the approximations of SP-STHOSVD with R-STHOSVD, we use a target rank $(r+p, r+p, r+p)$ with an additional oversampling parameter $p = 5$. We ran the SP-STHOSVD and the R-STHOSVD algorithms

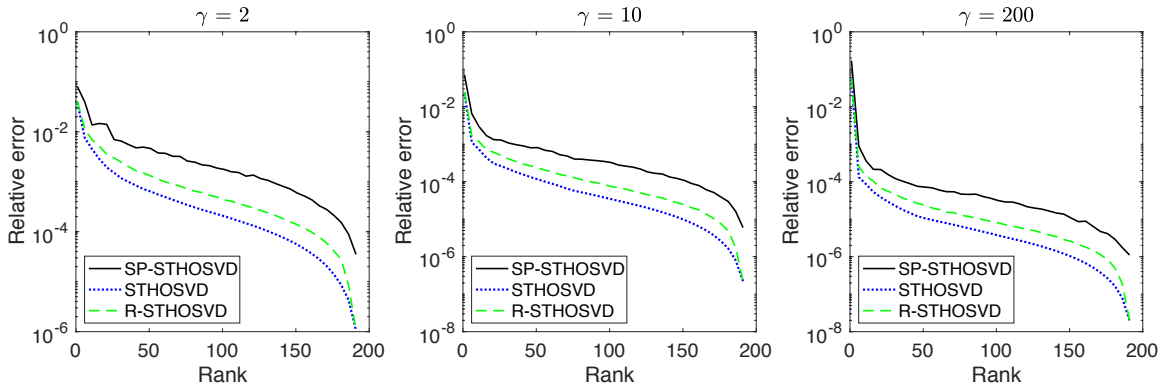

Figure 3. *Relative error for synthetic sparse tensor $\mathcal{X}$ defined in (6.2) with $\gamma = 2, 10, 200$ and increasing target rank (r, r, r). We compare the SP-STHOSVD algorithm (Algorithm 5.1) to the STHOSVD and R-STHOSVD algorithms with inputs of oversampling parameter $p = 5$ and processing order $\rho = [1, 2, 3]$.*

on the condensed Enron tensor (details in Table 2) with processing order $\rho = [3, 1, 2]$ and increasing target rank (r, r, r). The relative errors obtained are shown in Table 6. We can see that the error for SP-STHOSVD is higher than that of the R-STHOSVD, as is anticipated from the theory. We also compared the runtime of these two algorithms averaged over three runs to see their respective costs, which are also shown in Table 6. We see that the SP-STHOSVD, in addition to preserving the structure, has significantly lower computational costs. We repeated the previous experiment for the condensed NELL-2 dataset and saw similar results; see supplementary materials Table SM3.

Table 6

The relative error and runtime of both SP-STHOSVD and R-STHOSVD on the tensors defined in Table 2 as the target rank (r, r, r) increases. The processing order was $\rho = [3, 1, 2]$, and the oversampling parameter was $p = 5$. Note that, for simplicity, the rank is the same for each mode, and that the input rank for the R-STHOSVD was $(r + p, r + p, r + p)$, and so the approximations have the same size.

	Relative error		Runtime in seconds	
Target rank	SP-STHOSVD	R-STHOSVD	SP-STHOSVD	R-STHOSVD
20	0.6015	0.2081	0.4086	31.5615
45	0.3854	0.1259	0.7965	34.5802
70	0.3548	0.0870	1.3276	36.6431
95	0.2038	0.0632	2.3465	39.3095
120	0.1503	0.0458	2.8175	39.7169
145	0.0976	0.0332	3.5659	42.0969
170	0.0756	0.0239	6.2158	45.8429
195	0.0578	0.0180	6.8285	50.2907

7. Conclusion.

In this paper, we developed new randomized algorithms and analysis for low-rank tensor decompositions in the Tucker form. Specifically, we proposed adaptive algorithms for problems where the target rank is not known beforehand and algorithms that preserve the structure of the original tensor. We also provided probabilistic analysis of randomized compression algorithms, R-HOSVD and R-STHOSVD, as well as for the newly proposed

algorithms. We showed, through the analysis and numerical examples, that using randomized techniques still allows for accurate approximations to tensors, and that the approximation error is comparable to deterministic algorithms, with much lower computational costs.

Acknowledgment. The authors would like to thank Ilse Ipsen for reading through the paper and giving useful feedback.

REFERENCES

[1] AT&T Laboratories at Cambridge, *Olivetti Database of Faces*, https://cs.nyu.edu/~roweis/data.html, 2002.

[2] B. W. Bader and T. G. Kolda, *Efficient MATLAB computations with sparse and factored tensors*, SIAM J. Sci. Comput., 30 (2007), pp. 205–231, https://doi.org/10.1137/060676489.

[3] K. Batselier, W. Yu, L. Daniel, and N. Wong, *Computing low-rank approximations of large-scale matrices with the tensor network randomized SVD*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1221–1244, https://doi.org/10.1137/17M1140480.

[4] C. Battaglino, G. Ballard, and T. G. Kolda, *A practical randomized CP tensor decomposition*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 876–901, https://doi.org/10.1137/17M1112303.

[5] D. J. Biagioni, D. Beylkin, and G. Beylkin, *Randomized interpolative decomposition of separated representations*, J. Comput. Phys., 281 (2015), pp. 116–134.

[6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, Jr., and T. M. Mitchell, *Toward an architecture for never-ending language learning*, in AAAI'10: Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI, Menlo Park, CA, 2010, pp. 1306–1313.

[7] M. Che and Y. Wei, *Randomized algorithms for the approximations of Tucker and the tensor train decompositions*, Adv. Comput. Math., 45 (2019), pp. 395–428.

[8] H. Cheng, Z. Gimbutas, P.-G. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404, https://doi.org/10.1137/030602678.

[9] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, *Tensor networks for dimensionality reduction and large-scale optimization: Part 1. Low-rank tensor decompositions*, Found. Trends Machine Learning, 9 (2016), pp. 249–429.

[10] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, and D. P. Mandic, *Tensor networks for dimensionality reduction and large-scale optimization: Part 2. Applications and future perspectives*, Found. Trends Machine Learning, 9 (2017), pp. 431–673.

[11] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278, https://doi.org/10.1137/S0895479896305696.

[12] L. De Lathauwer, B. De Moor, and J. Vandewalle, *On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342, https://doi.org/10.1137/S0895479898346995.

[13] P. Drineas and M. W. Mahoney, *A randomized algorithm for a tensor-based generalization of the singular value decomposition*, Linear Algebra Appl., 420 (2007), pp. 553–571.

[14] P. Drineas and M. W. Mahoney, *RandNLA: Randomized numerical linear algebra*, Commun. ACM, 59 (2016), pp. 80–90.

[15] Z. Drmač and A. K. Saibaba, *The discrete empirical interpolation method: Canonical structure and formulation in weighted inner product spaces*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1152–1180, https://doi.org/10.1137/17M1129635.

[16] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.

[17] N. B. Erichson, K. Manohar, S. L. Brunton, and J. N. Kutz, *Randomized CP Tensor Decomposition*, preprint, https://arxiv.org/abs/1703.09074, 2017.

[18] L. Grasedyck, D. Kressner, and C. Tobler, *A literature survey of low-rank tensor approximation techniques*, GAMM-Mitt., 36 (2013), pp. 53–78.

[19] M. Gu, *Subspace iteration randomization and singular value problems*, SIAM J. Sci. Comput., 37 (2015), pp. A1139–A1173, https://doi.org/10.1137/130938700.
[20] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869, https://doi.org/10.1137/0917055.
[21] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, Springer Ser. Comput. Math. 42, Springer, Berlin, 2012.
[22] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288, https://doi.org/10.1137/090771806.
[23] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 2012.
[24] B. Huber, R. Schneider, and S. Wolf, *A randomized tensor train singular value decomposition*, in Compressed Sensing and Its Applications, Springer, New York, 2017, pp. 261–290.
[25] J. Jacod and P. Protter, *Probability Essentials*, Springer, Berlin, 2012.
[26] T. G. Kolda, *A counterexample to the possibility of an extension of the Eckart–Young low-rank approximation theorem for the orthogonal rank tensor decomposition*, SIAM J. Matrix Anal. Appl., 24 (2003), pp. 762–767, https://doi.org/10.1137/S0895479801394465.
[27] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500, https://doi.org/10.1137/07070111X.
[28] D. Kressner and L. Periša, *Recompression of Hadamard products of tensors in Tucker format*, SIAM J. Sci. Comput., 39 (2017), pp. A1879–A1902, https://doi.org/10.1137/16M1093896.
[29] M. W. Mahoney, *Randomized algorithms for matrices and data*, Found. Trends Machine Learning, 3 (2011), pp. 123–224.
[30] M. W. Mahoney, M. Maggioni, and P. Drineas, *Tensor-CUR decompositions for tensor-based data*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 957–987, https://doi.org/10.1137/060665336.
[31] O. A. Malik and S. Becker, *Fast Randomized Matrix and Tensor Interpolative Decomposition Using CountSketch*, preprint, https://arxiv.org/abs/1901.10559, 2019.
[32] P.-G. Martinsson and S. Voronin, *A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices*, SIAM J. Sci. Comput., 38 (2016), pp. S485–S507, https://doi.org/10.1137/15M1026080.
[33] A. K. Saibaba, *HOID: Higher order interpolatory decomposition for tensors based on Tucker representation*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 1223–1249, https://doi.org/10.1137/15M1048628.
[34] A. K. Saibaba, *Randomized subspace iteration: Analysis of canonical angles and unitarily invariant norms*, SIAM J. Matrix Anal. Appl., 40 (2019), pp. 23–48, https://doi.org/10.1137/18M1179432.
[35] J. Shetty and J. Adibi, *The Enron Email Dataset Database Schema and Brief Statistical Report*, Information Sciences Institute Technical Report, University of Southern California, Los Angeles, CA, 2004.
[36] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis, *FROSTT: The Formidable Repository of Open Sparse Tensors and Tools*, http://frostt.io/, 2017.
[37] Y. Sun, Y. Guo, C. Luo, J. Tropp, and M. Udell, *Low-Rank Tucker Approximation of a Tensor from Streaming Data*, preprint, https://arxiv.org/abs/1904.10951, 2019.
[38] D. B. Szyld, *The many proofs of an identity on the norm of oblique projections*, Numer. Algorithms, 42 (2006), pp. 309–323.
[39] C. E. Tsourakakis, *MACH: Fast randomized tensor decompositions*, in Proceedings of the 2010 SIAM International Conference on Data Mining, SIAM, Philadelphia, 2010, pp. 689–700, https://doi.org/10.1137/1.9781611972801.60.
[40] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, *A new truncation strategy for the higher-order singular value decomposition*, SIAM J. Sci. Comput., 34 (2012), pp. A1027–A1052, https://doi.org/10.1137/110836067.
[41] M. A. O. Vasilescu and D. Terzopoulos, *Multilinear analysis of image ensembles: Tensorfaces*, in Proceedings of the 7th European Conference on Computer Vision, Springer, Berlin, 2002, pp. 447–460.
[42] N. Vervliet and L. De Lathauwer, *A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors*, IEEE J. Selected Topics Signal Process., 10 (2016), pp. 284–295.
[43] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer, *Tensorlab* 3.0, https://www.tensorlab.net, 2016.

[44] W. Yu, Y. Gu, and Y. Li, *Efficient randomized algorithms for the fixed-precision low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1339–1359, https://doi.org/10.1137/17M1141977.

[45] J. Zhang, A. K. Saibaba, M. E. Kilmer, and S. Aeron, *A randomized tensor singular value decomposition based on the t-product*, Numer. Linear Algebra Appl., 25 (2018), e2179.

[46] G. Zhou, A. Cichocki, and S. Xie, *Decomposition of Big Tensors with Low Multilinear Rank*, preprint, https://arxiv.org/abs/1412.1885, 2014.