Electronic Transactions on Numerical Analysis.
Volume 63, pp. 247–280, 2025.

ISSN 1068–9613.
DOI: 10.1553/etna_vol63s247

ETNA
Kent State University and
Johann Radon Institute (RICAM)

RTSMS: RANDOMIZED TUCKER WITH SINGLE-MODE SKETCHING*

BEHNAM HASHEMI† AND YUJI NAKATSUKASA‡

Abstract. We propose RTSMS (Randomized Tucker with Single-Mode-Sketching), a randomized algorithm for approximately computing a low-rank Tucker decomposition of a given tensor. It uses sketching and the least-squares method to compute the Tucker decomposition in a sequentially truncated manner. RTSMS essentially only sketches one mode at a time, so the sketch matrices are significantly smaller than for alternative approaches. It uses a rank estimator to adaptively find an appropriate rank for the Tucker decomposition, without requiring it as input. RTSMS is demonstrated to be competitive with existing methods, sometimes outperforming them by a large margin.

Key words. tensor decompositions, randomized algorithms, sketching, least-squares, leverage scores, Tikhonov regularization, iterative refinement, HOSVD

AMS subject classifications. 68W20, 65F55, 15A69

1. Introduction.

The Tucker decomposition is a family of representations that break up a given tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ into the multilinear product of a core tensor $\mathcal{C} \in \mathbb{R}^{r_1 \times r_2 \times \cdots \times r_d}$ and a factor matrix $F_k \in \mathbb{R}^{n_k \times r_k}$ $(r_k \leq n_k)$ along each mode $k = 1, 2, \ldots d$, i.e.,

$$\mathcal{A} = \mathcal{C} \times_1 F_1 \times_2 F_2 \cdots \times_d F_d;$$

see Section 2 for the definition of the mode-k product $\times_k$. The standard shorthand notation to refer to the above formula is $\mathcal{A} = [\![\mathcal{C}; F_1, F_2, \ldots F_d]\!]$. Assuming that $\mathcal{A}$ can be well approximated by a low multilinear rank decomposition ($r_k \ll n_k$ for some or all k), one takes advantage of the fact that the core tensor $\mathcal{C}$ can be significantly smaller than $\mathcal{A}$. The canonical polyadic (CP) decomposition, also very popular in multilinear data analysis, is a special case of the Tucker decomposition in which the core tensor $\mathcal{C}$ has to be diagonal.

The history goes back to the 1960s when Tucker introduced the concept as a tool in quantitative psychology [51], as well as algorithms [52] for its computation. The decomposition has various applications such as dimensionality reduction, face recognition, image compression [45, 54], etc. Several deterministic and randomized algorithms have been developed for the computation of a Tucker decomposition [1, 7, 12, 15, 33, 38, 39, 47, 53, 59].

In what follows we give a very short outline of certain aspects of the Tucker decomposition and refer the reader to the review [28] by Kolda and Bader for other aspects including citations to several contributions. The higher-order orthogonal iteration (HOOI) by De Lathauwer, De Moor, and Vandewalle [16] is an alternating least-squares (ALS) method that uses the SVD to find the best multilinear rank-$\mathbf{r}$ approximation of $\mathcal{A}$. The same authors [15] introduced a generalization of the singular value decomposition of matrices to tensors called HOSVD. Vannieuwenhoven, Vandebril, and Meerbergen [53] introduced an efficient algorithm called STHOSVD that computes the HOSVD via a sequential truncation of the tensor taking advantage of the compressions achieved when processing all of the previous modes.

Cross algorithms constitute a wide class of powerful deterministic algorithms for computing a Tucker decomposition. These algorithms compute Tucker decompositions by interpolating the given tensor on a carefully selected set of pivot elements, also referred to as

*Received May 28, 2025. Accepted June 2, 2025. Published online on July 11, 2025. Recommended by L. Reichel.
†Corresponding author. School of Computing and Mathematical Sciences, University of Leicester, Leicester, LE1 7RH, UK (b.hashemi@le.ac.uk).
‡Mathematical Institute, University of Oxford, Oxford, OX2 6GG, UK (nakatsukasa@maths.ox.ac.uk).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

"cross" points. Notable examples include the Cross-3D algorithm of Oseledets, Savostianov, and Tyrtyshnikov [43] and the fiber sampling methods of Caiafa and Cichocki [10]. Relevant Tucker decomposition algorithms in the context of 3D function approximation include the slice-Tucker decomposition [25] and the fiber sampling algorithm of Dolgov, Kressner, and Strössner [17], which relies on oblique projections using subindices chosen based on the discrete empirical interpolation (DEIM) [11] method.

In this paper, we introduce RTSMS (Randomized Tucker with Single-Mode-Sketching), which is based on sequentially finding the factor matrices in the Tucker approximation while truncating the tensor. In this sense it is similar to STHOSVD [53]. Crucially, RTSMS is based on sketching the tensor from just one side (mode) at a time, not two (or more), thus avoiding an operation that can be a computational bottleneck. This is achieved by finding a low-rank approximation of unfoldings based on the generalized Nyström (GN) method (instead of randomized SVD [23]), but replacing the second sketch with a subsampling matrix chosen via the leverage scores of the first sketch for improved efficiency. In addition, we employ regularization and iterative refinement techniques to improve the numerical stability. While we are unable to establish useful a priori error bounds for the approximation, we provide an a posteriori bound that accurately tracks the error, thus enabling us to compute a Tucker decomposition to a user-defined accuracy.

Another key aspect of RTSMS is its ability to find the rank adaptively, given a required error tolerance. We do this by blending matrix-rank estimation techniques [36] in the algorithm to determine the appropriate rank and truncating accordingly, without significant additional computation.

After reviewing preliminary results in Section 2, we provide an outline of existing randomized algorithms for computing Tucker decomposition in Section 3. Section 4 then describes our algorithm RTSMS, and we illustrate its performance with experiments in Section 5.

2. Preliminaries.

Let us begin with a brief overview of basic concepts in deterministic Tucker decompositions. Throughout the paper we denote $n_{(-k)} := \Pi_{\substack{j=1\\ j\neq k}}^{d} n_j = \Pi_{j=1}^{d} n_j/n_k$.

2.1. Modal unfoldings.

Let $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ be a tensor of order d. The *mode-k unfolding* of $\mathcal{A}$, denoted by $A_{(k)}$, is a matrix of size $n_k \times n_{(-k)}$ whose columns are the mode-k fibers of $\mathcal{A}$ [21, p. 723].

2.2. Modal product.

A simple but important family of contractions are the modal products. These contractions involve a tensor, a matrix, and a mode. In particular, we have the following definition:

DEFINITION 2.1 ([21, p. 727]). *If $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$, $M \in \mathbb{R}^{m_k\times n_k}$, and $1 \le k \le d$, then*

$$\mathcal{B} = \mathcal{A} \times_k M \in \mathbb{R}^{n_1\times\cdots n_{k-1}\times m_k\times n_{k+1}\times\cdots\times n_d}$$

denotes the mode-k product of $\mathcal{A}$ and M defined by $B_{(k)} = MA_{(k)}$.

Note that every mode-k fiber in $\mathcal{A}$ is multiplied by the matrix M requiring the condition $\text{size}(\mathcal{A}, k) = \text{size}(M, 2)$. Here, $B_{(k)}$ is a matrix of size $m_k \times n_{(-k)}$, hence $\mathcal{B}$ is a tensor of size $n_1 \times \ldots n_{k-1} \times m_k \times n_{k+1} \cdots \times n_d$.

If $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$, $F \in \mathbb{R}^{m_1\times n_k}$, and $G \in \mathbb{R}^{m_2\times m_1}$, then

$$(\mathcal{A} \times_k F) \times_k G = \mathcal{A} \times_k (GF) \tag{2.1}$$

resulting in a tensor of size $n_1 \times \cdots \times n_{k-1} \times m_2 \times n_{k+1} \times \cdots \times n_d$. See [28, p. 461] or [15, Property 3].

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Reformulating matrix multiplications in terms of modal products, the matrix SVD can be rewritten as follows [2]:

$$A = U\Sigma V^T \quad \text{means that} \quad A = \Sigma \times_1 U \times_2 V.$$

2.3. Deterministic HOSVD. For an order-d tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, a HOSVD [15] is a decomposition of the form

$$\mathcal{A} = \mathcal{C} \times_1 U_1 \times_2 U_2 \times \cdots \times_d U_d, \tag{2.2}$$

where the *factor matrices* $U_k \in \mathbb{R}^{n_k \times r_k}$ $(r_k \leq n_k)$ have orthonormal columns and the *core tensor* $\mathcal{C} = \mathcal{A} \times_1 U_1^T \times_2 U_2^T \cdots \times_d U_d^T \in \mathbb{R}^{r_1 \times r_2 \cdots \times r_d}$ is not diagonal in general but enjoys the so-called all-orthogonality property. Let vec denote the operation of stacking all fibers of a tensor into a long column vector. The all-orthogonality property of $\mathcal{C}$ means that for all possible values of $n \in \{1, 2, \ldots, d\}$, every two subtensors $\mathcal{C}_{i_n=\alpha}$ and $\mathcal{C}_{i_n=\beta}$ of $\mathcal{C}$ obtained by fixing its n-th index to α and β, respectively, satisfy $\langle \mathcal{C}_{i_n=\alpha}, \mathcal{C}_{i_n=\beta} \rangle := \text{vec}(\mathcal{C}_{i_n=\alpha})^T \text{vec}(\mathcal{C}_{i_n=\beta}) = 0$ when $\alpha \neq \beta$.

The Tucker (multilinear) rank of $\mathcal{A}$ is the vector of ranks of modal unfoldings [21, p. 734]

$$\text{rank}(\mathcal{A}) := [r_1, r_2, \ldots, r_d],$$

i.e., $r_k := \text{rank}(A_{(k)})$ is the dimension of the space spanned by the columns of $A_{(k)}$, which is equal to the span of U_k; indeed, U_k is computed via the SVD of $A_{(k)}$.

Note also that the multilinear rank-$(r_1, r_2, \ldots, r_d)$ HOSVD truncation of $\mathcal{A}$, while it is *not* the best multilinear rank-$(r_1, r_2, \ldots, r_d)$ approximation to $\mathcal{A}$ in the Frobenius norm, is quasi-optimal; see (2.3) below.

2.4. Deterministic STHOSVD. This is the sequentially truncated variant of HOSVD, which, while processing each mode (in a user-defined processing order $p = [p_1, p_2, \ldots, p_d]$, where $1 \leq p_k \leq d$ are distinct integers), truncates the tensor. We display its pseudocode in Algorithm 8 in Appendix A.

Let $\hat{\mathcal{C}}^{(i)}$ denote the partially truncated core tensor of size $r_1 \times \cdots \times r_i \times n_{i+1} \times \cdots \times n_d$ (obtained in the i-th step of Algorithm 8, where $\hat{\mathcal{C}}^{(0)} := \mathcal{A}$ and $\hat{\mathcal{C}}^{(d)} := \mathcal{C}$). Following [53] we denote the i-th partial approximation of $\mathcal{A}$ by

$$\hat{\mathcal{A}}^{(i)} := [\![\hat{\mathcal{C}}^{(i)}; U_1, \ldots, U_i, I, \ldots, I]\!] = \hat{\mathcal{C}}^{(i)} \times_1 U_1 \times_2 U_2 \cdots \times_i U_i,$$

which, loosely speaking, has rank $(r_1, \ldots, r_i, n_{i+1}, \ldots, n_d)$ in the sense that this is the size of $\hat{\mathcal{C}}^{(i)}$. In particular, we have $\hat{\mathcal{A}}^{(0)} := \mathcal{A}$, and the final approximation obtained by STHOSVD is $\hat{\mathcal{A}}^{(d)} := \hat{\mathcal{A}}$. The factor matrix U_k is computed via the SVD of $\hat{A}^{(k-1)}_{(k)}$, the mode-k unfolding of $\hat{\mathcal{A}}^{(k-1)}$. This is an overarching theme in the paper: to find an approximate Tucker decomposition, we find a low-rank approximation of the unfolding of the tensor $\hat{\mathcal{A}}^{(k-1)}$.

The following lemma states that the square of the error in approximating $\mathcal{A}$ by STHOSVD is equal to the sum of the square of the errors committed in the successive approximations and is bounded by the sum of squares of all the modal singular values that have been discarded[1]. The error in both STHOSVD and HOSVD satisfy the same upper bound, and STHOSVD tends to give a slightly smaller error [53].

[1] The statement of the lemma uses a specific processing order, but the upper bound remains the same for any other order.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

LEMMA 2.2 ([39, 53]). *Let* $\hat{\mathcal{A}} := [\![\mathcal{C}; U_1, U_2, \ldots, U_d]\!]$ *be the rank-*$(r_1, r_2, \ldots, r_d)$ STHOSVD *approximation of* $\mathcal{A}$ *with processing order* $p = [1, 2, \ldots, d]$. *Then,*

$$\|\mathcal{A} - \hat{\mathcal{A}}\|_F^2 = \sum_{i=1}^{d} \|\hat{\mathcal{A}}^{(i-1)} - \hat{\mathcal{A}}^{(i)}\|_F^2 \leq \sum_{i=1}^{d} \|\mathcal{A} \times_i (I - U_i U_i^T)\|_F^2 = \sum_{i=1}^{d} \sum_{j=r_i+1}^{n_i} \sigma_j^2(A_{(i)}).$$

Note also that approximations obtained by deterministic HOSVD and STHOSVD are both quasi-optimal in the sense that they satisfy the bound

$$\|\mathcal{A} - \hat{\mathcal{A}}\|_F \leq \sqrt{d}\|\mathcal{A} - \hat{\mathcal{A}}_{opt}\|_F, \tag{2.3}$$

where $\hat{\mathcal{A}}_{opt}$ denotes the best rank-$(r_1, r_2, \ldots, r_d)$ approximation. A popular method aiming at finding $\hat{\mathcal{A}}_{opt}$ is HOOI, a computationally expensive nonlinear iteration [21, pp. 734–735], whose convergence is not guaranteed [28] but often works well. In many cases, (ST)HOSVD suffices with the near-optimal accuracy (2.3).

Our method, RTSMS, is similar to STHOSVD in that in each step it truncates the tensor, but it differs from previous methods in how each factor is computed.

2.5. Tools in randomized linear algebra. We close this section with relevant preliminaries on randomized numerical linear algebra.

Random sketching. A Gaussian sketch $\Omega \in \mathbb{R}^{n\times k}$ is a matrix whose entries are independently drawn from the standard normal (Gaussian) distribution with mean zero and variance 1. A random matrix $X \in \mathbb{C}^{n\times k}$, with $n \geq k$, is called a subsampled random Fourier transform (SRFT) if it has the structure $X = \sqrt{\frac{n}{k}} DFS^*$, where D is an $n \times n$ diagonal matrix whose diagonal entries are independent and are either $+1$ or -1 with equal probability, F is an $n \times n$ discrete Fourier transform (DFT) matrix, and S in a $k \times n$ subsampling matrix whose entries are all zero except for a single randomly placed 1 in each column. For a general $m \times n$ matrix A, AX can be computed with a cost of $\mathcal{O}(mn \log n)$ rather than $\mathcal{O}(mnk)$. If A is real, then we take F to be the discrete cosine transform (DCT) matrix instead of the DFT.

The generalized Nyström method. Randomized low-rank approximation is among the most successful examples of randomized algorithms in numerical linear algebra. The randomized SVD in [23] is a popular and well-understood method for the task, and it is used for Tucker computations in [39, 59], as we describe in Section 3.1.1.

Our algorithm RTSMS is based more heavily on another class of randomized low-rank approximation methods, namely the generalized Nyström approximation

$$A \approx (A\Omega)(\tilde{\Omega}^T A\Omega)^{\dagger}(\tilde{\Omega}^T A). \tag{2.4}$$

Here $\Omega, \tilde{\Omega}$ are tall-skinny random sketch matrices, usually taken as a Gaussian or SRFT. The approximation (2.4) has been shown to give a nearly-optimal low-rank approximation of A when $\Omega, \tilde{\Omega}$ are (oblivious) subspace embeddings (including Gaussian) [13, 41, 50]. The implementation based on [41] is summarized in Algorithm 1.

In our context, we will apply (2.4) to successive unfoldings of the tensor $\mathcal{A}$ (or its variant), wherein A in (2.4) is very fat-wide, and we shall take $\tilde{\Omega}$ (a small matrix) to be Gaussian, whereas Ω (very tall skinny) will be a subsampling matrix (up to details, including regularization).

Randomized algorithms for least-squares problems. Later, when solving least-squares (LS) problems, we will use (approximate) leverage score sampling. Given an LS problem $\min_X \|AX - B\|_F$ involving a tall-skinny $m \times n$ matrix A, the leverage score [19, 40] of the i-th row of A is $l_i := \|q_i\|_2^2$, where q_i is the i-th row of Q in the thin QR factorization $A = QR$. Given s, the number of rows to pick from the index set $\{1, 2, \ldots, m\}$, leverage score sampling

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 1 GN Generalized Nyström method [41, Algorithm 2.1].
Inputs are a matrix $A \in \mathbb{R}^{m\times n}$ and (e.g., Gaussian) random matrices $\Omega \in \mathbb{R}^{n\times r}$ and $\tilde{\Omega} \in \mathbb{R}^{m\times \hat{r}}$, where e.g. $\hat{r} = 1.5r$.
Outputs are $\hat{U} \in \mathbb{R}^{m\times r}$ and $\hat{V} \in \mathbb{R}^{n\times r}$ such that $A \approx (A\Omega)(\tilde{\Omega}^T A\Omega)^\dagger(\tilde{\Omega}^T A) = \hat{U}\hat{V}^T$.

1: Compute $\mathtt{AX} := A\Omega$.
2: Compute $\mathtt{YA} := \tilde{\Omega}^T A$.
3: Compute $\mathtt{YAX} := \mathtt{YA}\,\Omega$.
4: Compute the thin QR decomposition $\mathtt{YAX} = QR$.
5: Compute $\hat{U}$ by solving triangular linear systems $\hat{U}R = \mathtt{AX}$.
6: Compute $\hat{V} := Q^T\mathtt{YA}$.

then means selecting s rows of A with a probability proportional to the leverage scores of the rows; see Section 4.4.2 for more details. The sampling is then equivalent to forming a row submatrix $S \in \mathbb{R}^{s\times m}$ of I_m. We then solve $\min_{\hat{X}} \|S(A\hat{X} - B)\|_F$, a problem of reduced size.

Randomized rank estimation. The focus of this paper is on rank-adaptive Tucker decomposition: Given a tolerance on the relative residual, our main algorithm computes the corresponding Tucker rank adaptively. This is done by invoking the randomized matrix rank-estimation algorithm of [36, Algorithm 1] to a modal unfolding of the current partial approximation of the tensor $\mathcal{A}$. More specifically, the estimation of the i-th numerical Tucker rank r_i of $\mathcal{A}$ exploits the observation that the decay of the modal singular values of $\hat{\mathcal{A}}^{(i)}$ can be reliably estimated by monitoring the decay of the corresponding modal singular values of the sketched $\hat{\mathcal{A}}^{(i)}$. See Steps 6–19 of Algorithm 5, where the size of the sketches is initialized with a small quantity and increased adaptively if required.

3. Existing randomized algorithms for Tucker decomposition.

A number of randomized algorithms have been proposed for computing an approximate Tucker decomposition. In what follows we explain some of those techniques with focus on the ones for which implementations are publicly available and divide them into two categories: fixed-rank algorithms that require the multilinear rank to be given as an input and those that are adaptive in rank, which are the main goal of this paper.

3.1. Fixed-rank algorithms.

Here we suppose that the output target Tucker rank $(r_1, r_2, \ldots, r_d)$ is given. The first candidate to consider is a randomized analogue of the standard HOSVD algorithm.

3.1.1. Randomized HOSVD/STHOSVD.

A natural idea to speed up the computation of HOSVD or STHOSVD is to use randomized SVD [23] (Algorithm 2) in the computation of the SVD of the unfoldings. This has been done in [59, Algorithm 2] and [39, Algorithms 3.1, 3.2], and Minster, Saibaba, and Kilmer [39] in particular present extensive analysis of its approximation quality. We display the randomized STHOSVD in Algorithm 3 and compare it against our algorithm in numerical experiments, as it is usually more efficient than the randomized HOSVD.

A well-known technique to improve the quality of the approximation obtained by randomized matrix SVD is to run a few iterations of the power method [23]. Mathematically, this means that Y in Algorithm 2 is replaced with $A(A^TA)^q$ for an integer q.

A note on sketching. An important aspect of any randomized algorithm utilizing a random sketch is: which sketch should be used? In [39] and [59, Algorithm 2], they are taken to be Gaussian matrices. This class of random matrices usually comes with the strongest theoretical guarantees and robustness. Other more structured random sketches have been proposed and

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 2 Randomized matrix SVD without power iteration (Halko-Martinsson-Tropp [23]. See also [39, Algorithm 2.1])
Inputs are a matrix $A \in \mathbb{R}^{m\times n}$, a target rank r, and possibly also a sketch matrix Ω.
Output is an SVD $A \approx \hat{U}\hat{\Sigma}\hat{V}^T$ with $\hat{U} \in \mathbb{R}^{m\times r}$, $\hat{\Sigma} \in \mathbb{R}^{r\times r}$, and $\hat{V} \in \mathbb{R}^{n\times r}$.

1: Draw a sketch (e.g., Gaussian) matrix $\Omega \in \mathbb{R}^{n\times \hat{r}}$ (if not given as input), and compute $Y := A\Omega$.
2: Compute the thin QR decomposition $Y := QR$. {Q: approximate range(A)}
3: Compute $B := Q^TA$.
4: Compute the thin SVD $B := \hat{U}_B\hat{\Sigma}\hat{V}^T$. {$\hat{U}_B$ and $\hat{\Sigma}$ are $\hat{r}\times\hat{r}$ while $\hat{V}$ is $n\times\hat{r}$.}
5: Output $\hat{U} := Q\hat{U}_B(:, 1:r)$, $\hat{\Sigma} := \hat{\Sigma}(1:r, 1:r)$, and $\hat{V} := \hat{V}(:, 1:r)$.

Algorithm 3 Randomized STHOSVD [39, Algorithm 3.2].
Inputs are $\mathcal{A} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$, a multilinear rank $(r_1, r_2, \ldots, r_d)$, an oversampling parameter $\tilde{p}$, and a processing order $\mathbf{p}$ of the modes.
Output is $\mathcal{A} \approx [\![\mathcal{C}; U_1, U_2, \ldots, U_d]\!]$.

1: Set $\mathcal{C} := \mathcal{A}$.
2: **for** $i = p_1, \ldots, p_d$ **do**
3: Draw a sketch (e.g., Gaussian) matrix Ω_i of size $z_i \times \hat{r}_i$, where $\hat{r}_i := r_i + \tilde{p}$ and $z_i := (\Pi_{j=1}^{i-1}\hat{r}_j)(\Pi_{j=i+1}^{d}n_j)$.
4: Set U_i as the left singular vectors computed with $\texttt{RandSVD}(C_{(i)}, r_i, \Omega_i)$ using Algorithm 2.
5: Update $C_{(i)} := U_i^TC_{(i)}$. {Overwriting $C_{(i)}$ overwrites $\mathcal{C}$.}
6: **end for**

shown to be effective, including sparse [55] and FFT-based [49] sketches, and in the tensor sketching context, those employing Khatri-Rao products [47].

In this paper we mostly focus on Gaussian sketches, unless specified otherwise—while other sketches may well be more efficient in theory, the smallness of our sketches (an important feature of our algorithm RTSMS) means that the advantages offered by structured sketches are likely to be limited.

The one-pass algorithms of Malik and Becker. Tucker-TS and Tucker-TTMTS are two algorithms for Tucker decomposition [33, Algorithm 2] that rely on the TensorSketch framework [44] and can handle tensors whose elements are streamed, i.e., they are lost once processed. These algorithms are variants of the standard ALS (HOOI) and iteratively employ sketching for computing solutions to large overdetermined least-squares problems and for efficiently approximating chains of tensor-matrix products that are Kronecker products of smaller matrices. Inherited from the characteristics of TensorSketch, these algorithms only require a single pass of the input tensor.

The one-pass algorithm of Sun, Guo, Luo, Tropp, and Udell. Another single-pass sketching algorithm targeting streaming Tucker decomposition is introduced in [47], where a rigorous theoretical guarantee for the approximation error was also provided. Treating the tensor as a multilinear operator, it employs the Khatri-Rao product of random matrices to identify a low-dimensional subspace for each mode of the tensor that captures the action of the operator along that mode and then produces a low-rank operator with the same action on the identified low-dimensional tensor product space, which is helpful especially when storage cost is a potential bottleneck.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

The leverage-score-sampling approach of Fahrbach, Fu, and Ghadiri. Another fixed-rank algorithm for Tucker decomposition was introduced in [20]. While designed within the ALS architecture, it aims at minimizing

$$\|\mathcal{A} - \mathcal{C} \times_1 F_1 \times_2 F_2 \cdots \times_d F_d\|_F^2 + \lambda\left(\|\mathcal{C}\|_F^2 + \sum_{k=1}^{d} \|F_k\|_F^2\right),$$

where λ is a regularization parameter. It combines leverage score sampling and Richardson iterations when solving the sketched problem with an approach for multiplying sparsified Kronecker product matrices to improve the running time of all steps of ALS for a Tucker decomposition.

3.2. Rank-adaptive algorithms. We now turn to existing algorithms that are rank-adaptive, i.e., they are able to determine the appropriate rank on the fly to achieve a prescribed approximation tolerance. Although the first natural candidate to consider is the randomized adaptive variant of the standard HOSVD algorithm [39, Algorithm 4.1], in what follows we focus on its sequentially truncated version [39, Algorithm 4.2], which is more efficient in practice.

3.2.1. Adaptive R-STHOSVD. For the sake of completeness, we outline the adaptive randomized STHOSVD method [39, Algorithm 4.2] below in Algorithm 4.

Algorithm 4 Adaptive R-STHOSVD [39, Algorithm 4.2].
Inputs are $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, a tolerance tol ≥ 0, a blocking parameter $b \geq 1$, and a processing order $\mathbf{p}$ of the modes.
Output is $\mathcal{A} \approx [\![\mathcal{C}; \hat{U}_1, \hat{U}_2, \ldots, \hat{U}_d]\!]$.

1: Set $\mathcal{C} := \mathcal{A}$.
2: **for** $i = p_1, \ldots, p_d$ **do**
3: Compute $\hat{U}_i = \texttt{AdaptRandRangeFinder}(C_{(i)}, \frac{\text{tol}}{\sqrt{d}}, b)$. {Error-controlled range finder. We use `svdsketch`.}
4: Update $C_{(i)} = \hat{U}_i^T C_{(i)}$. {Overwriting $C_{(i)}$ overwrites $\mathcal{C}$.}
5: **end for**
6: Return $\mathcal{C}$ by tensorizing $C_{(i)}$.

At the heart of Algorithm 4 is an adaptive randomized matrix range finder. Numerous techniques have been introduced in the literature for this task [23, 35, 57]. Given a matrix A and a positive tolerance tol, the primary objective is to find a (tall) matrix Q with orthonormal columns such that the relative residual in approximating the range of A by Q is bounded from above by tol, i.e.,

$$\|A - QQ^T A\| \leq \text{tol}\, \|A\|. \tag{3.1}$$

The rank of the low-rank approximation then corresponds to the number of columns in Q. The idea of adaptive randomized range finders is to start with a limited number of columns in the random matrix Ω in order to estimate the range Q. Then, the number of random columns drawn is gradually increased until Q satisfies (3.1). Among the current state-of-the-art randomized range finders is the randQB_EI_auto algorithm of [57], which is a variant of an algorithm originally introduced in [23]. randQB_EI_auto has been integrated into MATLAB since release 2020b as the `svdsketch` function. We will compare RTSMS with Algorithm 4, whose Step 4 calls `svdsketch`. See Section 5 for more details.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

4. Randomized Tucker with single-mode sketching (RTSMS). We now describe RTSMS, our new randomized algorithm for computing a Tucker decomposition and HOSVD. RTSMS follows the basic structure of R-STHOSVD of sequentially finding a low-rank approximation to the unfoldings, but crucially, it only applies a random sketch on *one mode* at each step, resulting in a dramatically smaller sketch matrix. RTSMS can thus offer speedup over existing algorithms, especially when sketching the large dimensions is costly.

4.1. RTSMS outline. To illustrate the high-level ideas, let us provide an overview of RTSMS when applied to compute a Tucker decomposition of a tensor $\mathcal{A}$ of size $n_1 \times n_2 \times n_3$ with the processing order of $\mathbf{p} = [1\ 2\ 3]$, where a multilinear rank (r_1, r_2, r_3) is also *given*. The last assumption merely serves the purpose of simplifying the explanation enabling us to momentarily set aside the rank-adaptivity feature of RTSMS (Steps 1 and 6–19 of Algorithm 5) and concentrate on the other core aspects of the algorithm. We illustrate the process in Figure 4.1. In RTSMS, a Tucker decomposition $\mathcal{A} \approx [\![\mathcal{C}; F_1, F_2, F_3]\!]$ is computed where the i-th factor matrix F_i is of size $n_i \times \hat{r}_i$ (where $\hat{r}_i \approx 1.5r_i$ to account for oversampling, which improves robustness) rather than $n_i \times r_i$, and which does not have orthonormal columns in general. The size of the final core tensor $\mathcal{C}$ is $\hat{r}_1 \times \hat{r}_2 \times \hat{r}_3$.

During the execution of RTSMS, the core tensor is being updated, which is why it has a "temporary nature". We therefore use both $\mathcal{B}^{\text{old}}$ and $\mathcal{B}^{\text{new}}$ to refer to the core tensor. More specifically, in the beginning, we initialize $\mathcal{B}^{\text{old}}$ with the given tensor $\mathcal{A}$, and once a new core is computed within each iteration, we denote it with $\mathcal{B}^{\text{new}}$. Figure 4.1 illustrates this visually.

From the full tensor $\mathcal{A}$ to Tucker1. At the beginning, $i = p_1 = 1$, $\mathcal{B}^{\text{old}}$ is initialized with $\mathcal{A}$, and a Gaussian sketch Ω_1 of size $\hat{r}_1 \times n_1$ is utilized while the overall goal is to compute a Tucker1 decomposition

$$\mathcal{B}^{\text{old}} \approx \mathcal{B}^{\text{new}} \times_1 F_1,$$

which is a decomposition with only one factor matrix, or equivalently, a Tucker decomposition which sets $d - 1 = 2$ of the factor matrices to be the identity matrix. See [28, 52] and [30, Section 4.5.1]. See also the top panel of Figure 4.1. Importantly, we directly compute the "temporary core tensor" $\mathcal{B}^{\text{new}} = \mathcal{A} \times_1 \Omega_1$ as the sketch of $\mathcal{A}$ in Steps 20–21 and then find the factor matrix F_1 in Step 22 of Algorithm 5. This is in contrast to existing approaches, e.g., (R-)STHOSVD, where F_1 is computed and orthonormalized first and then $\mathcal{B}^{\text{new}}$ is taken to be $\mathcal{A} \times_1 F_1^T$; see Steps 4 and 5 of Algorithm 3. This means that RTSMS is almost single-pass, i.e., it does not need to revisit $\mathcal{A}$ once the sketch $\mathcal{A} \times_1 \Omega_1$ is computed, aside from the need to subsample a small number of fibers of $\mathcal{A}$; see Section 4.4.2.

In Steps 20–21, we find an approximate row space of $A_{(1)}$ using a randomized sketch. We first compute the mode-1 product of $\mathcal{A}$ and Ω_1, which is equivalent to $B_{(1)}^{\text{new}} = \Omega_1 B_{(1)}^{\text{old}}$. Here, $B_{(1)}^{\text{old}} = A_{(1)}$ is a matrix of size $n_1 \times (n_2 n_3)$, Ω_1 is $\hat{r}_1 \times n_1$, and so $\mathcal{B}^{\text{new}}$ is a tensor of size $\hat{r}_1 \times n_2 \times n_3$, where $\hat{r}_1 \approx 1.5r_1$ accounts for oversampling and r_1 is either given as an input or found by the adaptive rank estimation (Steps 6–19). The cost of computing $\mathcal{B}^{\text{new}}$ is $\mathcal{O}(n_1 n_2 n_3 \hat{r}_1)$, which is cubic in terms of the size of the original tensor $\mathcal{A}$. Since Ω_1 is random by construction, provided that $\sigma_{r_1}(A_{(1)}) < \text{tol}\ \sigma_1(A_{(1)})$, with high-probability the row space of $B_{(1)}^{\text{new}}$ is a good approximation of the dominant row space of $A_{(1)}$.

The theory in randomized low-rank approximation [23, Section 10] indicates that there exists a factor matrix F_1 such that $\mathcal{B}^{\text{new}} \times_1 F_1 \approx \mathcal{B}^{\text{old}}$. In Step 22 we aim at solving the least-squares problem

$$\min_{F_1} \|\mathcal{B}^{\text{new}} \times_1 F_1 - \mathcal{B}^{\text{old}}\|_F. \tag{4.1}$$

Its efficient and robust solution proves to be subtle, and we defer the discussion to Section 4.4.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

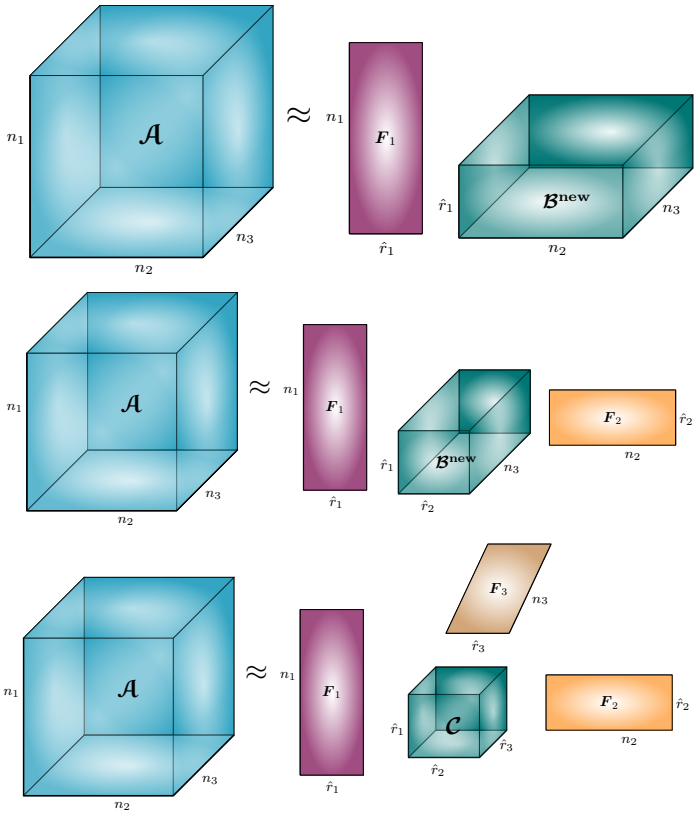

FIG. 4.1. *Illustration of RTSMS for 3D tensors, Tucker rank* (r_1, r_2, r_3)*, and mode-processing order* [1 2 3]*. Top: Tucker1 decomposition* $\mathcal{A} \approx \mathcal{B}^{\text{new}} \times_1 F_1$*, where* $\mathcal{A} =: \mathcal{B}^{\text{old}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$*,* $\mathcal{B}^{\text{new}} = \mathcal{A} \times_1 \Omega_1 \in \mathbb{R}^{\hat{r}_1 \times n_2 \times n_3}$*, and* $F_1 \in \mathbb{R}^{n_1 \times \hat{r}_1}$*. In RTSMS,* $\mathcal{B}^{\text{new}}$ *is computed before* F_1*. Middle: Tucker2 decomposition* $\mathcal{A} \approx \mathcal{B}^{\text{new}} \times_1 F_1 \times_2 F_2$*, where* $\mathcal{B}^{\text{new}} \in \mathbb{R}^{\hat{r}_1 \times \hat{r}_2 \times n_3}$ *and* $F_2 \in \mathbb{R}^{n_2 \times \hat{r}_2}$*. Note that* $\mathcal{B}^{\text{new}}$ *here is overwritten on* $\mathcal{B}^{\text{new}}$ *from top and therefore represents a different tensor. Bottom: Tucker3 decomposition* $\mathcal{A} \approx \mathcal{B}^{\text{new}} \times_1 F_1 \times_2 F_2 \times_3 F_3$*, where this time* $\mathcal{B}^{\text{new}} \in \mathbb{R}^{\hat{r}_1 \times \hat{r}_2 \times \hat{r}_3}$ *is the final core tensor* $\mathcal{C}$ *computed once Step 24 of Algorithm 5 is executed and* $F_3 \in \mathbb{R}^{n_3 \times \hat{r}_3}$*. Figure generated using* `Tensor_tikz` *[29].*

From Tucker1 to Tucker2. Next, $i = p_2 = 2$, and we aim at decomposing the updated $\mathcal{B}^{\text{old}}$ from the previous round (see Step 23 in Algorithm 5), hence eventually computing a Tucker2 decomposition of the original tensor $\mathcal{A}$. This is a decomposition with two factor matrices or equivalently a Tucker decomposition which sets $d - 2 = 1$ factor to be the identity matrix [30, Section 4.5.2]. See the middle picture of Figure 4.1. This is done by essentially repeating the process in the previous round for the second mode of the temporary core tensor $\mathcal{B}^{\text{old}}$: In Steps 20–21 we sketch to find the row space of $B^{\text{old}}_{(2)}$. We first compute the mode-2 product of $\mathcal{B}^{\text{old}}$ and Ω_2, which is equivalent to $B^{\text{new}}_{(2)} = \Omega_2 B^{\text{old}}_{(2)}$. Here, $B^{\text{old}}_{(2)}$ is a matrix of size $n_2 \times (n_3 \hat{r}_1)$, Ω_2 is $\hat{r}_2 \times n_2$, and so the updated temporary core tensor $\mathcal{B}^{\text{new}}$ is a tensor of size $\hat{r}_1 \times \hat{r}_2 \times n_3$. This process shrinks the dimension of the row space of $A_{(2)}$ from the large n_2 to the smaller $\hat{r}_2$ and is the first task in computing a decomposition of the temporary core tensor and, equivalently, a Tucker2 decomposition of $\mathcal{A}$. As before, the row space of $B^{\text{new}}_{(2)}$ approximates that of $A_{(2)}$ with high probability, provided r_2 (either given as input or found by the adaptive rank estimation) is appropriate. Here again $\hat{r}_2 \approx 1.5 r_2$ accounts for oversampling. Step 22 then solves the least-squares problem $\min_{F_2} \|\mathcal{B}^{\text{new}} \times_2 F_2 - \mathcal{B}^{\text{old}}\|_F$.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

From Tucker2 to Tucker3. Finally, $i = p_3 = 3$, and we aim at decomposing the updated $\mathcal{B}^{\text{old}}$ from the previous step, hence eventually computing a Tucker3 (or simply a Tucker) decomposition of the original tensor $\mathcal{A}$. See [30, Chapter. 4.5.3] and Figure 4.1, bottom (adapted from [29]).

The computational complexity of the entire algorithm is dominated by the first tensor-matrix multiplication $\mathcal{A} \times_1 \Omega_1$, which is equivalent to computing $\Omega_1 A_{(1)}$. When $r_1 \approx n_1$, the need to sketch r_1 vectors of dimension $n_2 n_3 \cdots n_d$ needed in the least-squares problem can be significant; we discuss this again in Section 4.7.

4.2. Details on RTSMS. We now discuss RTSMS in full generality. Its pseudocode is given in Algorithm 5. In a nutshell, we find low-rank approximations of the unfolding matrices $A_{(i)}$ (or more precisely the unfolding of the current core tensor $\mathcal{B}^{\text{old}}$) via a single-mode sketch. Namely, to approximate $A_{(1)} \in \mathbb{R}^{n_1 \times n_{(-1)}}$, we sketch from the left by an operation equivalent to computing $\Omega_1 A_{(1)}$, where $\Omega_1 \in \mathbb{R}^{\hat{r}_1 \times n_1}$, and then find $F_1 \in \mathbb{R}^{n_1 \times \hat{r}_1}$ such that $F_1 \Omega_1 A_{(1)} \approx A_{(1)}$ by solving a massively overdetermined least-squares problem with many right-hand sides, $\min_{F_1} \|A_{(1)}^T \Omega_1^T F_1^T - A_{(1)}^T\|_F$, which we do by randomized row subset selection, regularization, and iterative refinement. We discuss the details of the least-squares solution in Section 4.4.

Rank adaptivity. Another key component of RTSMS, which we did not describe above, is the rank adaptivity. In Steps 6–19, we find an appropriate Tucker rank on the fly by employing a fast rank estimator [36, Algorithm 1] on each unfolding, which are truncated in turn. The rank is estimated by applying random sketches on both sides of the matrix, and the key idea is that the magnitudes of the leading singular values are preserved in this process. Moreover, much of these sketches needed for rank estimation can be reused within RTSMS; for example, the first $\min(\tilde{r}_i, \hat{r}_i)$ rows of Ω_i and $\Omega_i M$ in Steps 21 and 22 are identical to those of Steps 8 and 9, and the QR factorization $(\tilde{\Omega}_i M Y_i)^T = QR$ can be reused by Algorithm 6. Overall, the process is more efficient than `svdsketch`. Alternatively, RTSMS is also able to take the rank as a user-defined input. The effectiveness of this adaptive multilinear rank estimation strategy is demonstrated in Section 5, particularly in Example 5.4.

Single-mode vs. multi-mode sketch. As far as we know, existing randomized algorithms for Tucker decomposition require applying dimension reduction maps to the tensor unfoldings from the *right* (i.e., the larger dimension). Focusing on mode-1, one faces $A_{(1)}$, which is a short fat matrix of size $n_1 \times n_{(-1)}$. Sketching from the right then involves computations with a tall skinny randomized matrix whose size is $n_{(-1)} \times \hat{r}_1$. R-STHOSVD and R-HOSVD are among the examples, as are most algorithms we mentioned. Considering the fact that $n_{(-1)}$ could be huge, generating or even storing the randomized sketch matrix could be problematic. The focus of the single-pass techniques [33, 47] mentioned in Section 3.1.1 has therefore been on designing algorithms which do not need generating and storing a tall skinny dimension reduction map. Instead, the fat side of the unfolding is sketched by skillfully employing Khatri-Rao products of smaller random matrices so as to avoid explicitly forming random matrices which have $n_{(-1)}$ rows. In this sense, one can classify all the aforementioned algorithms as those which sketch in all-but-one modes when processing each of the modes.

In Algorithm 5, the original tensor $\mathcal{A}$, which is potentially large in all dimensions, is directly used only in the first recursion $i = p_1$. The updating of $\mathcal{B}^{\text{old}}$ in Step 23 is where sequential truncation takes place, making sure that subsequent computations proceed with the potentially much smaller truncated tensors. This truncation can improve the efficiency significantly, just as STHOSVD does over HOSVD. Notice however that, due to the oversampling aspect common to many randomized algorithms in numerical linear algebra, the output Tucker rank will be $\hat{\mathbf{r}}$, which is slightly larger than $\mathbf{r}$. See Section 4.5 as well as the last two paragraphs in Section 4.8 entitled *Further rank truncation* for relevant details.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 5 RTSMS: Randomized Tucker with single-mode sketching
Inputs are $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, the target tolerance tol on the relative residual, and a processing order $\mathbf{p}$ of the modes.
Output is a Tucker decomposition $\mathcal{A} \approx [\![\mathcal{C}; F_1, F_2, \ldots, F_d]\!]$.

1: Initialize $\mathbf{r}$ with a small ambitious rank estimate like $(10, 10, \ldots, 10)$.
2: Set $\mathcal{B}^{\text{old}} := \mathcal{A}$.
3: **for** $i = p_1, \ldots, p_d$ **do**
4: Set flag_i to 'unhappy'.
5: Find M, the i-th modal unfolding of $\mathcal{B}^{\text{old}}$.
6: **while** flag_i is 'unhappy' **do** {rank estimation}
7: Set $\tilde{r}_i := \text{round}(1.1\, r_i)$.
8: Draw a standard random Gaussian matrix $\tilde{\Omega}_i$ of size $\tilde{r}_i \times n_i$.
9: Form the matrix $\tilde{\Omega}_i M$ of size $\tilde{r}_i \times z_i$ where $z_i := (\Pi_{j=1}^{i-1} \tilde{r}_j)(\Pi_{j=i+1}^{d} n_j)$.
10: Set $s_i := k\tilde{r}_i$ (e.g., $k = 4$), draw SRFT $Y_i \in \mathbb{R}^{z_i \times s_i}$, and form $\tilde{\Omega}_i M Y_i \in \mathbb{R}^{\tilde{r}_i \times s_i}$.
11: Compute the thin QR factorization $(\tilde{\Omega}_i M Y_i)^T = QR$.
12: Find the smallest ℓ such that $\sigma_{\ell+1}(R) \le \text{tol}\, \sigma_1(R)$.
13: **if** $\ell < r_i$ **then**
14: Change flag_i to 'happy', set the i-th multilinear rank estimate r_i to ℓ.
15: Update $\mathcal{B}^{\text{old}}$ to be $\tilde{\Omega}_i M$ tensorized in the i-th mode.
16: **else if** ℓ is 'empty' or $\ell = \tilde{r}_i$ implying the sketch was full-rank **then**
17: Increase the rank estimate, e.g., $r_i := \text{round}(1.7 r_i)$.
18: **end if**
19: **end while**
20: Draw a $\hat{r}_i \times n_i$ Gaussian matrix Ω_i, where $\hat{r}_i := \text{round}(1.5\, r_i)$. {low-rank approx}
21: Compute $\mathcal{B}^{\text{new}} = \mathcal{B}^{\text{old}} \times_i \Omega_i$ (reusing $\tilde{\Omega}_i M$ from step 9)
22: Find F_i of size $n_i \times \hat{r}_i$ to minimize $\|\mathcal{B}^{\text{new}} \times_i F_i - \mathcal{B}^{\text{old}}\|_F$, using Algorithm 6
23: Update $\mathcal{B}^{\text{old}} := \mathcal{B}^{\text{new}}$.
24: **end for**
25: Set $\mathcal{C} := \mathcal{B}^{\text{new}}$.

Parallel, non-sequential variant. We have emphasized the advantages of RTSMS, most notably the fact that the sketches are small. One possible advantage of a non-sequential algorithm is that they are highly parallelizable and suitable for distributed computation, as discussed, e.g., in [59, p. 6]. It is possible to design a variant of RTSMS that works parallelly: run the first step ($i = p_1$ in Algorithm 5) to find the factor matrices F_i for each i from $\mathcal{A}$. We then find the core tensor as follows: compute the thin QR factorizations $F_i = U_i R_i$ for each i, and project them onto $\mathcal{A}$, i.e., $\mathcal{C} = \mathcal{A} \times_1 U_1^T \cdots \times_d U_d^T$. We do not discuss this further, as in our sequential experiments the standard RTSMS is more efficient with a lower computational cost.

4.3. Analysis of RTSMS. Let us explain why RTSMS is able to find an approximate Tucker decomposition. We focus on the first step $i = 1$ (and assume without loss of generality $p_1 = 1$), as the other cases are essentially a repetition.

Suppose that $\mathcal{A}$ has an approximate HOSVD (here we assume the factors F_i are orthonormal and denote them by U_i, which simplifies the theory but is not necessary in the algorithm),

$$\mathcal{A} = \mathcal{C} \times_1 U_1 \times_2 U_2 \times \cdots \times_d U_d + \mathcal{E}, \tag{4.2}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

where each $U_k \in \mathbb{R}^{n_k \times r_k}$ is orthonormal $U_k^T U_k = I_{r_k}$ and $\mathcal{E}$ is small in norm; that is, $\mathcal{A}$ has an approximate Tucker decomposition of rank $(r_1, r_2, \ldots, r_d)$. Then in the first step of the algorithm $\mathcal{B}^{\text{new}} = \mathcal{A} \times_1 \Omega_1$, one obtains $\mathcal{B}^{\text{new}} = \mathcal{C} \times_1 (\Omega_1 U_1) \times_2 U_2 \times \cdots \times_d U_d + \mathcal{E} \times_1 \Omega_1$. Note that $\|\mathcal{E} \times_1 \Omega_1\|_F \leq O(\|\mathcal{E}\|_F)$ because multiplication by Gaussian matrices roughly preserves the norm [23, Section 10] (the O notation hides constant multiples of $\sqrt{n_1}$). Now since Ω_1 is Gaussian, so is $\Omega_1 U_1$ by orthogonal invariance, and it is a (tall) $\hat{r}_1 \times r_1$ rectangular Gaussian matrix and therefore well-conditioned with high probability by the Marchenko-Pastur rule [46] or more specifically Davidson-Szarek's result [14, Theorem II.13].

In terms of the mode-1 unfolding, we have $(\mathcal{A} \times_1 \Omega_1)_{(1)} = \Omega_1 A_{(1)}$. Now recalling (4.2), note that the unfolding of $\mathcal{C} \times_1 U_1 \times_2 U_2 \times \cdots \times_d U_d$ can be written as $U_1 G_1$ for some $G_1 \in \mathbb{R}^{r_1 \times n_{(-1)}}$, so by assumption the mode-1 unfolding of $\mathcal{A}$ is $A_{(1)} = U_1 G_1 + \tilde{E}$, where $\|\tilde{E}\|_F = \|\mathcal{E}\|_F$. As U_1 is $n_1 \times r_1$, this implies that the matrix $A_{(1)}$ can be approximated in the Frobenius norm by a rank-r_1 matrix up to $\|\tilde{E}\|_F$.

This is precisely the situation where randomized algorithms for low-rank approximation are highly effective. In particular by the analysis in [23, Section 10] (applied to $A_{(1)}^T$ rather than $A_{(1)}$), it follows that by taking Ω_1 to have $\hat{r}_1 > r_1$ rows (say $\hat{r}_1 = 1.2 r_1$), the row space of $\Omega_1 A_{(1)}$ captures that of $A_{(1)}$ up to a small multiple of $\|\tilde{E}\|_F$. This implies that using the thin QR factorization $(\Omega_1 A_{(1)})^T = QR$, the rank-$\hat{r}_1$ matrix

$$A_{(1)} Q Q^T \approx A_{(1)} \tag{4.3}$$

approximates $A_{(1)}$ up to a modest multiple of $\|\tilde{E}\|_F$ and hence of $\|\mathcal{E}\|_F$.

Note that this discussion shows that $\min_{U_1 \in \mathbb{R}^{n_1 \times \hat{r}_1}} \|U_1(\Omega_1 A_{(1)}) - A_{(1)}\|_F = O(\|\mathcal{E}\|_F)$, because Q^T and $\Omega_1 A_{(1)}$ have the same row space. This is equivalent to the least-squares problem (4.1).

4.4. Algorithm to minimize$_{\hat{\mathbf{F}}_\mathbf{i}} \|\mathcal{B}^{\text{new}} \times_\mathbf{i} \hat{\mathbf{F}}_\mathbf{i} - \mathcal{B}^{\text{old}}\|_\mathbf{F}$. In RTSMS, we do not form Q or $A_{(1)} Q Q^T$ as these operations can be expensive and dominate the computation[2]. In particular, the computation of $A_{(1)} Q$ involves the large dimension $n_{(-1)}$ and can be expensive. Instead, in RTSMS we attempt to directly find the factor matrix $\hat{F}_1 \in \mathbb{R}^{n_1 \times \hat{r}_1}$ via minimizing $\|\hat{F}_1(\Omega_1 A_{(1)}) - A_{(1)}\|_F$, which we rewrite in the standard form of a least-squares (LS) problem as

$$\underset{\hat{F}_1 \in \mathbb{R}^{n_1 \times \hat{r}_1}}{\text{minimize}} \|(A_{(1)}^T \Omega_1^T) \hat{F}_1^T - A_{(1)}^T\|_F. \tag{4.4}$$

This LS problem has several important features worth noting: (i) it is massively overdetermined $A_{(1)}^T \Omega_1^T \in \mathbb{R}^{n_{(-1)} \times \hat{r}_1}$, (ii) it has many ($n_1$) right-hand sides, and (iii) the coefficient matrix $A_{(1)}^T \Omega_1^T$ is ill-conditioned and numerically rank-deficient (by design, assuming tol is close to machine precision). Solving (4.4) exactly via the classic QR-based approach gives the approximation $QQ^T A_{(1)}^T \approx A_{(1)}^T$, which is equivalent to (4.3). We employ three techniques to devise a more efficient (yet robust) approximation algorithm.

4.4.1. Randomized sketching. In order to speed up the computation, we solve the LS problem (4.4) using randomization. This is now a standard technique for solving highly-overdetermined least-squares problems of which (4.4) is one good example.

[2]Here we again focus on the first step $i = 1$, for which $A_{(i)} = B_{(i)}^{\text{old}}$ and $\Omega_i A_{(i)} = B_{(i)}^{\text{new}}$. However, with an abuse of notation, allowing $\mathcal{A}$ to always denote the current core tensor $\mathcal{B}^{old}$, the discussion extends to any i with minimal changes.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Among the most successful ideas in the randomized solution of LS problems is *sketching*, wherein instead of (4.4) one solves the sketched problem

$$\underset{\hat{F}_1 \in \mathbb{R}^{n_1 \times \hat{r}_1}}{\text{minimize}} \|S(A_{(1)}^T \Omega_1^T \hat{F}_1^T - A_{(1)}^T)\|_F, \tag{4.5}$$

where $S \in \mathbb{R}^{s_1 \times n_{(-1)}}$ ($s_1 \geq \hat{r}_1$) is a random matrix, called the sketching matrix. Effective choices of S include Gaussian (not necessarily the most efficient choice), FFT-based (e.g., SRFT) sketches [49], and sparse sketches [13]. The solution of the problem (4.5) is $\hat{F}_1^T = (SA_{(1)}^T \Omega_1^T)^\dagger (SA_{(1)}^T)$.

The choice of S that is the easiest to analyze is when it is taken to be a random Gaussian matrix. Then the resulting rank-$\hat{r}_1$ approximation $A_{(1)} \approx \hat{F}_1 \Omega_1 A_{(1)}$ obtained by solving (4.5) becomes equal to the generalized Nyström (GN) approximation [13, 41, 56], which takes $A_{(1)} \approx A_{(1)} X (Y A_{(1)} X)^\dagger Y A_{(1)}$, where X, Y are random sketches of appropriate sizes. To see this, note that with the solution of (4.5), $\hat{F}_1 = A_{(1)} S^T (\Omega_1 A_{(1)} S^T)^\dagger$, we have the identity $\hat{F}_1 \Omega_1 A_{(1)} = A_{(1)} S^T (\Omega_1 A_{(1)} S^T)^\dagger \Omega_1 A_{(1)}$, i.e., they coincide by taking $S^T = X$ and $\Omega_1 = Y$.

Despite the close connection, an important difference here is that in RTSMS, $S^T = X$ will be generated using $\Omega_1 A_{(1)}$ as we describe in Section 4.4.2, so it depends on $\Omega_1 = Y$, unlike the standard GN approximation, where X, Y are independent. Furthermore, taking X to be an independent sketch necessitates sketching $A_{(1)}$ from the right, which violates our 'single-mode-sketch' approach and results in inefficiency.

4.4.2. Solving LS via row subset selection. An important aspect of the problem (4.5) is the large number n_1 of right-hand sides, which makes it crucial that the sketching cost for these is kept low. In fact, experiments suggest that the use of an SRFT sketch often results in the computation being dominated by sketching the right-hand sides.

To circumvent this and to enhance efficiency in RTSMS, we choose S to be (instead of Gaussian) a column subselection matrix, i.e., S is a column submatrix of the identity. We suggest the use of *subsampling*, i.e., S is a row subset of the identity. This way the cost of sketching (i.e., computing $S(A_{(1)}^T \Omega_1^T)$ and $SA_{(1)}^T$) becomes minimal.

To choose the subsampled indices we use the leverage scores [19], which is a common technique in randomized LS problems and beyond [40]. These are the squared row-norms of the orthogonal factor of the orthonormal column space of the coefficient matrix $(\Omega_1 A_{(1)})^T$ and can be approximated using sketching [19, 32] with $O(N\hat{r}_1 \log N)$ operations (with an SRFT sketch) for an $\hat{r}_1 \times N$ coefficient matrix; here, $N = n_{(-1)}$. We note that leverage scores are not the only way to find subsampling indices; however, most alternatives (such as QR with column pivoting) are either slower and/or do not allow oversampling.

In brief, approximate leverage scores are computed as follows [19]: First sketch the matrix to compute $Y^T(\Omega_1 A_{(1)})^T$ and its thin QR decomposition $Y^T(\Omega_1 A_{(1)})^T = QR$, where Y is an $N \times O(\hat{r}_1)$ SRFT sketch. Then $(\Omega_1 A_{(1)})^T R^{-1}$ is well-conditioned, so we estimate its row norms via sampling, i.e., the row norms of $(\Omega_1 A_{(1)})^T R^{-1} G$, where G is a Gaussian matrix with $O(1)$ columns. Importantly, the SRFT sketch $Y^T(\Omega_1 A_{(1)})^T$ (which requires $O(Nr_1 \log N)$ operations) is needed also in the rank estimation process, so this computation incurs no additional cost. Note that it is cheaper than computing $A_{(1)}Y$ (i.e., sketching the right-hand side of (4.5) with Y), because $\hat{r}_1 < n_1$. We then choose $s = O(\hat{r}_1 \log \hat{r}_1)$ indices from $\{1, \ldots, z_1\}$ by randomly sampling *without* replacements, with the i-th row chosen with probability $\ell_i / (\sum_{j=1}^{z_1} \ell_j)$. We then form the resulting subsample matrix $S \in \mathbb{R}^{s \times z_1}$, which is a column-submatrix of the identity $I_{n_{(-1)}}$. We then solve the subsampled problem (4.5).

ETNA
Kent State University and
Johann Radon Institute (RICAM)

To understand the role of leverage scores, let us state a result on the residual for a sketched least-squares problem (4.4) for a general S (not necessarily a subsampling matrix), which we state in terms of a standard least-squares problem $\min_X \|AX - B\|_F$.

PROPOSITION 4.1. *Consider the $m \times n$ LS problem $\min_X \|AX - B\|_F$ with $A \in \mathbb{R}^{m\times n} (m \geq n), B \in \mathbb{R}^{m\times n_1}$. Let $A = QR$ be the thin QR factorization with $Q \in \mathbb{R}^{m\times n}$, and consider $SQ \in \mathbb{R}^{s\times n}$ with $s \geq n$. Let $\hat{X}_*$ denote the solution for $\min_X \|S(AX - B)\|_F$. Then we have*

$$\|A\hat{X}_* - B\|_F \leq \frac{\|S\|_2}{\sigma_{\min}(SQ)} \min_X \|AX - B\|_F.$$

Proof. This can be seen as a repeated application of a bound for a subsampled least-squares problem, as, e.g., in [11], slightly generalized to multiple right-hand sides. For completeness we give a full proof.

Consider the i-th column, which for simplicity we write $\min_x \|Ax - b\|_2$. Then $\|S(Ax - b)\|_2 = \|(SQ)Rx - Sb\|_2$, for which the solution is $x_* = R^{-1}(SQ)^\dagger Sb$, and hence,

$$\|Ax_* - b\|_2 = \|QRR^{-1}(SQ)^\dagger Sb - b\|_2 = \|(I - Q(SQ)^\dagger S)b\|_2.$$

Now note that $Q(SQ)^\dagger S$ is an (oblique) projection matrix $(Q(SQ)^\dagger S)^2 = Q(SQ)^\dagger S$ onto the span of Q, and so $I - Q(SQ)^\dagger S$ is also a projection. It hence follows that

$$\begin{aligned}\|(I - Q(SQ)^\dagger S)b\|_2 &= \|(I - Q(SQ)^\dagger S)Q_\perp Q_\perp^T b\|_2 \\ &\leq \|(I - Q(SQ)^\dagger S)\|_2 \|Q_\perp^T b\|_2 = \|Q(SQ)^\dagger S\|_2 \|Q_\perp^T b\|_2,\end{aligned}$$

where the last equality holds because $Q(SQ)^\dagger S$ is a projection [48].

Finally, noting that $\|Q_\perp b\|_2 = \min_x \|Ax - b\|_2$, we conclude that

$$\|Ax_* - b\|_2 \leq \|Q(SQ)^\dagger S\|_2 \min_x \|Ax - b\|_2 \leq \frac{\|S\|_2}{\sigma_{\min}(SQ)} \min_x \|Ax - b\|_2.$$

The claim follows by repeating the argument for every column $i = 1, \ldots, n$. ∎

Let us discuss Proposition 4.1 when S is a subsampling matrix generated via approximate leverage scores. First, we note that the use of leverage scores for the LS problem here is different from classical ones [55] in two ways: First, usually, leverage scores are computed for the subspace of the augmented matrix $[A, B]$, including the right-hand side (and usually there is a single right-hand side). We avoid this because this necessitates sketching the right-hand sides, of which there are many (n_1) of them; this becomes the computational bottleneck, which is precisely why we opted to finding an appropriate row subsample to reduce the cost. The by-product is that the suboptimality of the computed solution is governed by $\frac{\|S\|_2}{\sigma_{\min}(SQ)}$ instead of the subspace embedding constant (which can be < 1) as in standard methods. This leads to the second difference: we do not scale the entries of S inverse-proportionally with the leverage scores ℓ_i, so as to avoid $\|S\|_2 \gg 1$, which can happen when a row with low leverage score happens to be chosen. For the same reason we sample rows without replacements. We thus ensure $\|S\|_2 = 1$, which guarantees a good solution as long as $1/\sigma_{\min}(SQ)$ is not large[3]. Note that this means that the standard theory for leverage scores do not hold directly. However, by choosing large rows of Q with high probability, we tend to increase

[3] We should be content with $\sigma_{\min}(SQ) = O(1/\sqrt{z_1})$, which is what we expect if Q was Haar distributed; it is important to note that Proposition 4.1 is an upper bound and typically an overestimate by a factor $\approx \sqrt{z_i}$.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

the singular values of SQ, and with a modest number of oversampling, we typically have a modest $(\sigma_{\min}(SQ))^{-1}$. If it is desirable to ensure this condition, then we can use the estimate $\sigma_i(SQ) \approx \sigma_i(S(\Omega_1 A_{(1)})^T R^{-1})$, which follows from the fact that $(\Omega_1 A_{(1)})^T R^{-1}$ is well-conditioned with high probability by the construction of R (this is also known as whitening [42]).

We note that LS problems with many right-hand sides were studied by Clarkson and Woodruff [13] and Larsen and Kolda [31], who show that leverage-score sampling based on A (and not b) gives solutions with good residuals. However, the failure probability decays only algebraically in s, not exponentially. By contrast, in standard leverage score theory, SQ becomes well-conditioned with failure probability decaying exponentially in s [40, Chapter 6]. However, the guarantee is weaker, related to $1/\sigma_{\min}(SQ)$ rather than $\kappa_2(SQ)$. To our knowledge, there is no subsampling strategy that simultaneously guarantees high accuracy ($O(1)$ suboptimality in the residual) and has exponentially low failure probability in the presence of many or unknown right-hand sides. Here we prefer to keep the failure probability exponentially low.

It is worth noting that leverage score sampling is just one of many methods available for column/row subset selection. Other methods include pivoted LU and QR [18, 22] and the BSS method originally developed for graph sparsification [4]. We chose leverage score sampling to avoid the $z_i n_i^2$ cost required by deterministic methods and because Proposition 4.1 and experiments suggest that oversampling (selecting more than $\hat{r}_i$ rows) can help improve the accuracy.

The complexity of the approach is $O(sr_1 n_{(-1)} + sr_1^2)$: sr_1^2 for the QR factorization $(\Omega_1 A_{(1)} S)^T = QR \in \mathbb{R}^{s \times r_1}$ and[4] $sr_1 n_1$ for computing $R^{-1} Q^T A_{(1)}$. Together with the cost for leverage score estimation, the overall cost for solving the least-squares problem (4.4) is $O(n_{(-1)} \hat{r}_1 \log n_{(-1)} + sr_1 n_1 + sr_1^2)$.

Our experiments suggest that the above process of solving (4.4) via (4.5), which can be seen as an instance of a sketch-and-solve approach for LS problems, does not always yield satisfactory results: the solution accuracy was worse by a few digits than existing algorithms such as STHOSVD. The likely reason is numerical instability; qualitatively, the matrix $\Omega_1 A_{(1)}$ is highly ill-conditioned, and hence the computation of its sketch also comes with potentially large relative error (that is, significantly larger than the error with $A_{(1)} QQ^T$ in (4.3); the cause is likely numerical errors, as the theory shows that the residual of sketch-and-solve methods is within a modest constant of optimality [55]). The situation does not improve with a sketch-to-precondition method [37].

In RTSMS we employ two techniques to remedy the instability: *regularization* and *iterative refinement*.

4.4.3. Regularization and refinement. To improve the solution quality of the subsampled LS problem $\min_{\hat{F}_1} \|S(A_{(1)}^T \Omega_1^T \hat{F}_1^T - A_{(1)}^T)\|_F$, we introduce a common technique of Tikhonov regularization [24], also known as ridge regression. That is, instead of (4.4) we solve for a fixed $\lambda > 0$

$$\min_{\hat{F}_1} \|S_1(A_{(1)}^T \Omega_1^T \hat{F}_1^T - A_{(1)}^T)\|_F^2 + \lambda \|\hat{F}_1\|_F^2. \tag{4.6}$$

This is still equivalent to an LS problem with multiple independent right-hand sides

$$\min_{\hat{F}_1} \left\| \begin{bmatrix} S_1 A_{(1)}^T \Omega_1^T \\ \sqrt{\lambda} I \end{bmatrix} \hat{F}_1^T - \begin{bmatrix} S_1 A_{(1)}^T \\ 0 \end{bmatrix} \right\|_F^2$$

[4] While $\Omega_1 A_{(1)} S$ is simply an extraction of the columns of $\Omega_1 A_{(1)}$ specified by S, this step is not always negligible in an actual execution. It is nonetheless significantly faster than sketching $\Omega_1 A_{(1)}$ from the right.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

and can be solved in $O(sr_1 n_{(-1)} + sr_1^2)$ operations. We take $\lambda = O(u\|\Omega_1 A_{(1)} S_1\|)$. Regularization is known to attenuate the effect of solution blowing up due to the presence of excessively small singular values in the coefficient matrix (some of which can be numerical artifacts).

The second technique we employ is iterative refinement [26, Chapter 12]. The idea is to simply solve the problem twice, but we found that resampling can be helpful: denoting by $\hat{F}_1^{(1)}$ the computed solution of (4.6), we compute the residual in a *different* set of subsampled columns, also obtained by the leverage scores (the two sets S_1, S_2 are allowed to overlap but they differ substantially), update the (unsketched) right-hand side matrix $B := A_{(1)}^T - A_{(1)}^T \Omega_1^T (\hat{F}_1^{(1)})^T$, and solve

$$\min_{\hat{F}_1^{(2)}} \|S_2(A_{(1)}^T \Omega_1^T (\hat{F}_1^{(2)})^T - B)\|_F^2 + \lambda \|\hat{F}_1\|_F^2. \tag{4.7}$$

We then take the overall solution to be $\hat{F}_1^{(1)} + \hat{F}_1^{(2)}$. It is important in practice that the same λ is used in (4.6) and (4.7), even though the right-hand sides $A_{(1)}$ and B are typically vastly different in norm $\|A_{(1)}\|_F \gg \|B\|_F$. This is because the goal of the second problem (4.7) is to add a correction term and not to solve (4.7) itself accurately.

Algorithm 6 Algorithm for the least-squares problem $\min_{\hat{F}_i \in \mathbb{R}^{n_i \times \hat{r}_i}} \|\mathcal{B}^{\text{new}} \times_i \hat{F}_i - \mathcal{B}^{\text{old}}\|_F$ as in (4.4) arising in RTSMS.
Inputs are $B_{(i)}^{\text{old}} \in \mathbb{R}^{z_i \times n_i}$ and $\Omega_i \in \mathbb{R}^{\hat{r}_i \times z_i}$, with $z_i := (\Pi_{j=1}^{i-1} \hat{r}_j)(\Pi_{j=i+1}^{d} n_j)$.

1: Compute approximate leverage scores ℓ_i:
2: Using the QR factorization $(\Omega_i M Y_i)^T = QR$ from Algorithm 5, ℓ_i is the i-th row-norm of $(\Omega_i B_{(i)}^{\text{old}} Y_i)^T (R_i^{-1} G)$, for a standard Gaussian $G \in \mathbb{R}^{\hat{r}_i \times 5}$.
3: If $i = p_1$, choose $s_i = 4k\hat{r}_i$ indices, otherwise choose $s_i = 3k\hat{r}_i$ indices from $\{1, \ldots, z_i\}$, the i-th row is chosen with probability $\ell_i / (\sum_{j=1}^{z_i} \ell_j)$ without repetition. Form the resulting subsample matrices $S_1, S_2 \in \mathbb{R}^{s_i \times z_i}$.
4: Use Tikhonov regularization with S_1 to compute $\hat{F}_i^{(1)}$ solving (4.6).
5: Use iterative refinement with S_2 to compute $\hat{F}_i^{(2)}$ solving (4.7).
6: Compute $\hat{F}_i := \hat{F}_i^{(1)} + \hat{F}_i^{(2)}$.

It is perhaps surprising that a standard sketch-and-solve (even with regularization) does not always yield satisfactory solutions. While the algorithm presented here always gave good computed outputs in our experiments, it is an open problem to prove its stability or the lack of it (in which case alternative methods are needed that would guarantee stability). The stability of randomized least-squares solvers is generally a delicate topic [37], particularly when the coefficient matrix is numerically rank-deficient.

We close the section with a result on the error with RTSMS.

THEOREM 4.2. *Let $\hat{\mathcal{A}} := [\![\mathcal{C}; \hat{F}_1, \hat{F}_2, \ldots, \hat{F}_d]\!]$ be the output of RTSMS (Algorithm 5), where for each i, we compute $\hat{F}_i$ such that*[5] $\|\mathcal{B}^{\text{new},i} \times_i \hat{F}_i - \mathcal{B}^{\text{old},i}\|_F = \epsilon_i$. *Then*

$$\|\hat{\mathcal{A}} - \mathcal{A}\|_F \le \sum_{i=1}^{d} \left(\prod_{j=1}^{i-1} \|\hat{F}_j\|_2 \right) \epsilon_i. \tag{4.8}$$

[5]Here we add the superscript i in $\mathcal{B}^{\text{new,i}}$ to indicate which RTSMS step it refers to. We do not introduce i in the superscripts elsewhere to avoid complicated notation especially where we mention mode-i unfolding of $\mathcal{B}^{\text{old}}$ and $\mathcal{B}^{\text{new}}$.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Proof. Without loss of generality, assume $p_i = i$ for all i. In the first step $i = 1$, the approximation error $\|\mathcal{A} - \mathcal{B}^{\text{new},1} \times_1 \hat{F}_1\|_F$ is equivalent to the error in the LS problem $\|\mathcal{B}^{\text{old}} - \mathcal{B}^{\text{new}} \times_i \hat{F}_i\|_F = \epsilon_1$, which in turn is equal to the low-rank approximation error for the unfolding. At this point we have an approximate tensor in Tucker1 format $\mathcal{B}^{\text{new},1} \times_1 \hat{F}_1$, whose error is ϵ_1.

In the next step, we work on $\mathcal{B}^{\text{new},1}$ and effectively find its approximant of the form $\mathcal{B}^{\text{new},1} \approx \mathcal{B}^{\text{new},2} \times_2 \hat{F}_2$. An important aspect here is that the error in this approximation $\epsilon_2 := \|\mathcal{B}^{\text{new},1} \approx \mathcal{B}^{\text{new},2} \times_2 \hat{F}_2\|_F$ is not identical to the resulting error in approximating $\mathcal{A}$, as the first factor matrix $\hat{F}_1$ is not orthonormal. To bound the approximation error for $\mathcal{A}$ having computed $\hat{F}_1, \hat{F}_2$, we use the triangle inequality

$$\begin{aligned}\|E_2\|_F :&= \|\mathcal{A} - \mathcal{B}^{\text{new},2} \times_1 \hat{F}_1 \times_2 \hat{F}_2\|_F \\ &\le \|\mathcal{A} - \mathcal{B}^{\text{new},1} \times_1 \hat{F}_1\|_F + \|\mathcal{B}^{\text{new},1} \times_1 \hat{F}_1 - \mathcal{B}^{\text{new},2} \times_1 \hat{F}_1 \times_2 \hat{F}_2\|_F \\ &= \epsilon_1 + \|(\mathcal{B}^{\text{new},1} - \mathcal{B}^{\text{new},2} \times_2 \hat{F}_2) \times_1 \hat{F}_1\|_F \\ &\le \epsilon_1 + \epsilon_2 \|\hat{F}_1\|_2.\end{aligned}$$

Here we used the norm inequality $\|AB\|_F \le \|A\|_F \|B\|_2$ [27, Section B.7] for the final inequality. A similar argument shows that after k outer iterations of RTSMS and having computed $\hat{F}_1, \dots, \hat{F}_k$, we have

$$\begin{aligned}\|E_k\|_F :&= \|\mathcal{A} - \mathcal{B}^{\text{new,k}} \times_1 \hat{F}_1 \times_2 \hat{F}_2 \times \cdots \times_k \hat{F}_k\|_F \\ &\le \epsilon_1 + \epsilon_2 \|\hat{F}_1\|_2 + \epsilon_3 \|\hat{F}_1\|_2 \|\hat{F}_2\|_2 + \cdots + \epsilon_k \|\hat{F}_1\|_2 \cdots \|\hat{F}_{k-1}\|_2.\end{aligned}$$

The result follows from setting $k = d$. ∎

In view of the bound (4.8), at each step one can normalize $\hat{F}_j$ such that $\|\hat{F}_j\|_2 = 1$ and accordingly scale $\mathcal{B}^{\text{new}}$. This way we have direct control over the (bound for the) error committed at each step.

Theorem 4.2 is somewhat inconvenient in two ways: first, it is not as clean as the bounds found for, e.g., STHOSVD [53, Theorem 5.1], where the approximation error is bounded with respect to the best Tucker approximation of a given rank. Second, while the situation is similar to [39, Section 5.2], it is arguably worse here, as unlike in [39], the temporary core tensors $\mathcal{B}^{\text{new,i}}$ are not a subtensor of $\mathcal{A}$, which leads to significant difficulties for an a priori error analysis. The result is the bound (4.8), which is not directly related to the error of the best Tucker approximation (unlike previous results), and the bound involves the product of the norms of the factor matrices $\prod_{j=1}^{d-1} \|\hat{F}_j^T\|_2$, which is unknown in advance but become available as the computation proceeds.

Despite these caveats, in practice, the result is useful: First, as illustrated in the experiments, the bounds are quite tight in practice, providing useful guidance on the actual error of the output approximation. Second, fortunately, the factor matrices are seen to be modest in norm; in all our experiments, the values $\|\hat{F}_i\|$ were bounded by 1. Furthermore, essentially the same quantity (recall that $\hat{F}_1^T = (SA_{(1)}^T \Omega_1^T)^\dagger (SA_{(1)}^T)$) is shown to be $O(1)$ in [41, Lemma 3.2] when the sketches are assumed to be Gaussian, giving a theoretical support that these terms are not large.

Moreover, since $\|F_i\|_2$ is easy to compute, one can ensure that the overall error is bounded by tol by forcing each low-rank approximation to have an error bounded by $\epsilon_i \le \text{tol}/(d \prod_{j=1}^{d-1} \|\hat{F}_j^T\|_2)$. Although, as mentioned above, replacing this with $\epsilon_i \le \text{tol}/d$ was seen to have little to no impact. Moreover, this argument gives another justification for our use of regularization in the LS solution: it directly forces the solution F_i to have small norm.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Finally, despite the lack of connection to the best Tucker approximation error, RTSMS is seen to compute results with accuracy similar to existing algorithms.

We note that the bound (4.8) is deterministic even though the algorithm involves randomization. This is because it is an a posteriori bound and the randomness is represented by the ε_i's. The behavior of each ε_i is well studied in the literature [23, Section 10], at least in the simple context where the sketch matrix is Gaussian and no regularization and iterative refinement are performed.

4.5. HOSVD variant of RTSMS. Our algorithm can be complemented with a further conversion to HOSVD such that factor matrices have orthonormal columns and the core tensor has the so-called all-orthogonality property [15]. One way to do this standard deterministic step is to apply Algorithm 9 in Appendix C, which has the flexibility of choosing whether a multilinear singular value thresholding should also be carried out. In our numerical experiments we denote this variant with `RHOSVDSMS`.

4.6. Fixed-rank variant of RTSMS. A significant aspect of RTSMS is its rank adaptivity; it can automatically adjust the numerical multilinear rank of the tensor according to a given tolerance for the relative residual. This is the variant of RTSMS in which the multilinear rank is assumed to be known a priori. Most of the algorithms for the Tucker decomposition in the literature are of this type. This variant is essentially Steps 20–23 of RTSMS. If desired, this variant can also be converted to the HOSVD form without thresholding.

4.7. Computational complexity. In Table 4.1 we summarize the number of arithmetic operations involved in algorithms for computing a Tucker decomposition. For simplicity we assume that the order-d tensor $\mathcal{A}$ is $n \times n \cdots \times n$ and the target rank is $r \times r \cdots \times r$ (we do not include the cost for rank estimation; it is usually comparable to the algorithm itself). In addition, we assume without loss of generality that the processing order is $1, 2, \ldots, d$. The cost of the single-pass Tucker algorithm [47] is obtained by taking the factor sketching parameters to be $k = 3/2r$ and the core sketching parameters to be $s = 2k = 3r$.

It is evident that randomization reduces the exponent of the highest order term by one, and sequential truncation reduces the corresponding coefficient.

While Table 4.1 does not immediately reveal a cost advantage of RTSMS over R-STHOSVD and single-pass Tucker, let us repeat that the single-mode nature of the sketching can be a significant strength. For example, when the sketches are taken to be Gaussian, the cost of generating the sketches is lower with RTSMS by a factor $O(n^{d-2})$. Moreover, while we mainly treat Gaussian sketches, one can reduce the complexity by using structured sketches; for instance, with an SRFT sketch the complexity becomes $O(n^d \log n)$ as listed in parenthesis[6] in the table, which can be lower than rn^d. Such reduction is not possible with other methods based on finding the (orthonormal) factor matrices first, because the computation of $\mathcal{A} \times_1 F_1$ necessarily requires rn^d operations since F_1 is generally dense and unstructured. Another advantage of RTSMS is that computing $\Omega_1 \mathcal{A}_{(1)}$ requires far less communication than $\mathcal{A}_{(1)}\Omega_2$, as with $\Omega_1 \mathcal{A}_{(1)}$ the local computation (where $\mathcal{A}_{(1)}$ is split columnwise) is directly part of the output.

The table shows a somewhat simplified complexity; for example, RTSMS also requires $O(n^{d-1}rd \log n)$ for finding the leverage scores, which is usually no larger than $n^d r$. Note

[6] In theory [34, Section 9.3] this can be reduced to $O(n^d \log r)$, resulting in strictly lower complexity than $O(n^d r)$. However, the corresponding implementation of the fast transform is intricate and often not available. One can also use sparse sketches [55] to get $O(n^d)$ complexity. It is worth emphasizing that while such techniques can in theory reduce the complexity, in practice we observe that Gaussian sketches tend to be among the fastest, at least for the values of n (up to $O(10^3)$) that we have experimented with. The use of FFT, for example, involves a larger constant than Gaussian sketches.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

TABLE 4.1
Computational complexity of fixed-rank algorithms for computing a rank-$(r, r, \ldots, r)$ Tucker decomposition of an order-d tensor of size $n \times n \times \cdots \times n$ and $r \ll n$. $\hat{r} = r + p$, where p is the oversampling factor, e.g., $p = 5$ or $p = 0.5r$.

algorithm	dominant cost	sketch size	dominant operation
HOSVD [15, 52]	dn^{d+1}		SVD of d unfoldings each of size $n \times n^{d-1}$
STHOSVD [53]	n^{d+1}		SVD of $A_{(1)}$ which is $n \times n^{d-1}$. (Later unfoldings are smaller due to truncation)
R-HOSVD [39]	drn^d [39, Tab. 1]	$\hat{r} \times n^{d-1}$	computing $A_{(i)}\Omega_i$ where Ω_i of size $n^{d-1} \times \hat{r}$ and then forming $Q_i^T A_{(i)}$ for all i
R-STHOSVD [39, 59]	rn^d [39, Tab. 1]	$\hat{r} \times n^{d-1}$	forming $A_{(1)}\Omega_1$ with Ω_1 of size $n^{d-1} \times \hat{r}$. Subsequent unfoldings and sketching matrices are smaller
single-pass [47] (also [33])	rn^d [47, Tab. 2]	$\hat{r} \times n^{d-1}$	sketching by structured (Khatri-Rao product) dimension reduction maps
RTSMS	rn^d $(n^d \log n)$	$\hat{r} \times n$	computing $\Omega_1 A_{(1)}$ with Ω_1 of size $\hat{r} \times n$

that this computation requires sketching a large dimension. Fortunately, the number of vectors to be sketched is r rather than n needed with alternative methods. Nonetheless, this can be the dominant cost when $r \approx n$ or d is large.

4.8. Implementation details. Let us address a few points concerning the specifics of our RTSMS implementation[7].

- We noticed that the final residual depends more strongly on the error made in the least-squares problem in computing the first factor matrix as opposed to the later modes. An analogous observation was made by Vannieuwenhoven, Vandebril, and Meerbergen [53, Section 6.1], who emphasized the influence of the error caused by the first projection on the quality of the STHOSVD approximation in comparison with the remaining projections. For this reason we take a slightly larger number of oversamples when processing the first mode.
- Tensor-matrix contraction is a fundamental operation in tensor computation. Such mathematical operations involve tensor unfoldings and permutations which are memory-intensive. It is therefore advantageous to decrease the need for unfolding and permutation of large tensors so as to reduce data communication through the memory hierarchy and among processors. Conventional implementations of tensor-matrix contractions involve permutation of the tensor (except for mode-1 contraction) so that the computation can be performed by calling BLAS; see the function `tmprod` in TensorLab, for instance. While our implementations employ TensorLab, we adapt the use of `tensorprod`, introduced in MATLAB R2022a, in order to accelerate tensor-matrix contractions by avoiding explicit data permutations.

[7] Our MATLAB implementation is available at https://github.com/bhashemi/rtsms.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Further rank truncation. We explored a few different strategies to further truncate the computed Tucker decomposition and chose the following according to the numerical evidence. The idea is based on the HOSVD and is applied once the execution of Algorithm 5 is completed. We first use QR factorizations of the factors F_i and deterministic STHOSVD to convert the Tucker decomposition

$$\mathcal{A} \approx [\![\mathcal{C}; F_1, F_2, \ldots, F_d]\!]$$

to a HOSVD of the form

$$\mathcal{A} \approx [\![\check{\mathcal{C}}; U_1, U_2, \ldots, U_d]\!].$$

We then compute the higher-order modal singular values of $\mathcal{A}$ from $\check{\mathcal{C}}$ and then compare those modal singular values with the input tolerance to decide where to truncate the factors as well as the core tensor. See Algorithm 9 in Appendix C.

This truncation strategy brings the Tucker decomposition into HOSVD and applies a multilinear singular value thresholding operator, commonly employed in the context of low-rank matrix recovery; see [9] for instance. In our numerical experiments we observed that this strategy gives results whose multilinear rank and accuracy are consistent with changes in the given tolerance. Details are available in a pseudocode in Appendix C (Algorithm 9).

5. Experiments. In all the following experiments we use parameters detailed here. The oversampling parameter is set to be $\tilde{p} = 5$ in all randomized techniques of [39] and [33]. In the randomized techniques of [39], we use $[1, 2, \ldots, d]$ as the processing order of the vector of modes. In the Tucker-TensorSketch [33], we set the sketch dimension parameter K to be equal to $\tilde{p}$ above, so $K = 5$, a tolerance of 1×10^{-15}, and the maximum number of iterations to be 50. In our experiments we always take the sketches to be Gaussian, as the dimensions n_i and especially r_i are not large enough for other sketches (e.g., SRFT) to outperform it. We set the oversampling parameter $k = 4$. All experiments were carried out in MATLAB 2022b on a computer with 16GB memory.

Each experiment for computing a decomposition of the form $\tilde{\mathcal{A}} = [\![\mathcal{C}; U_1, U_2, \ldots, U_d]\!]$ we repeat five times and report the average CPU time (the variance is not large) as well as the geometric mean of the relative residuals defined by

$$\text{relative residual} = \frac{\|\mathcal{A} - \tilde{\mathcal{A}}\|_F}{\|\mathcal{A}\|_F}.$$

As our main algorithm is adaptive in rank, in our first four examples we focus on experiments in which an input tolerance is specified, and, in addition to the average CPU time and residual, we also report the average numerical multilinear rank (rounded to the nearest integer) as computed by each algorithm. When the RTSMS relative residuals are plotted (Examples 5.1, 5.2, 5.3, and 5.7), we also include the corresponding error bound from Theorem 4.2, where the absolute bound (4.8) is divided by $\|A\|_F$ to render them relative errors. In the first four examples, we compare RTSMS and RHOSVDSMS (see Section 4.5) with adaptive-rank R-STHOSVD (Algorithm 4), whose Step 3 incorporates `svdsketch`. Note that while the only mandatory input to `svdsketch` is an $m \times n$ input matrix A, it allows specifying the following input parameters: The input tolerance tol, maximum subspace dimension, blocksize, maximum number of iterations, and number of power iterations performed (default value: 1). Among these, it is worth noting that the input tolerance tol is required to satisfy

$$\sqrt{\text{machine epsilon}} \approx 1.5 \times 10^{-8} \leq \text{tol} < 1 \tag{5.1}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

since `svdsketch` does not detect errors smaller than the square root of machine epsilon; see Theorem 3 and Remark 3.3 in [57]. The default value of tol is machine epsilon$^{1/4} \approx 1.2 \times 10^{-4}$, but note that in this paper we will specify different values of tol satisfying (5.1). Also, in our tensor context the maximum subspace dimension, blocksize, and the maximum number of iterations are typically m, $\lfloor 0.1m \rfloor$, and 10, respectively.

5.1. Rank-adaptive experiments.

EXAMPLE 5.1. We take $\mathcal{A}$ to be samples of the Runge function

$$f(x,y,z) = \frac{1}{5 + x^2 + y^2 + z^2}$$

on a grid of size $600 \times 600 \times 600$ consisting of Chebyshev points on $[-1,1]^3$. Figure 5.1 reports the results, in which the top-right panel shows the speed of our algorithm and R-STHOSVD.

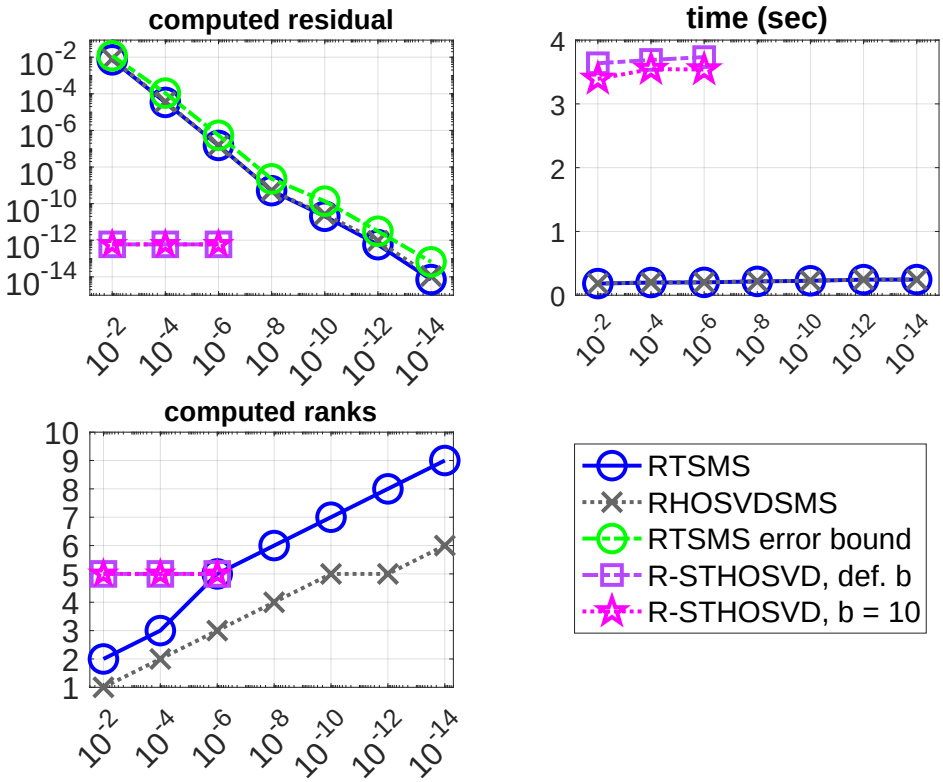

FIG. 5.1. *Comparison of rank-adaptive methods in terms of the residual, average multilinear ranks, and time for a tensor $\mathcal{A}$ of size* $600 \times 600 \times 600$ *in Example 5.1. The horizontal axes are all the input tolerance.*

The default values of the block size b used by `svdsketch` in R-STHOSVD are $\lfloor 0.1 n_i \rfloor = 60$ in modes $i = 1$ and 2. The corresponding average of the computed multilinear ranks is 5 (not 60 or 10 in the two executions of R-STHOSVD) reflecting the fact that orthogonalizations within randomized matrix range finders performed by MATLAB `orth` only keep r columns of the unfoldings, where r is the computed rank. When it comes to the third mode, the default value of the block size b used in R-STHOSVD is only 25, as

ETNA
Kent State University and
Johann Radon Institute (RICAM)

due to its sequential truncation aspect, the third unfolding matrix turns out to be of size $n_3 \times (r_1 r_2) = 600 \times 25$. In contrast to the multilinear ranks being always equal to 5 in both executions of R-STHOSVD in this example, we observe a smooth increase in the computed ranks as the input tolerance is decreased in both RTSMS and RHOSVDSMS. Our error bound is remarkably close to the actual RTSMS relative residuals. It is an interesting open problem to explain why the bound is so sharp.

EXAMPLE 5.2. We take $\mathcal{A}$ to be samples of the Wagon function on a grid of size $800 \times 1200 \times 300$ consisting of Chebyshev points on $[-1, 1]^3$. The function appears in a challenging global minimization problem and is most complicated in the second variable, which is why we took the second dimension of $\mathcal{A}$ to be the largest. See [5] for details.

Figure 5.2 illustrates our comparisons. The relative residuals in both RTSMS and RHOSVDSMS drops to about machine epsilon already for an input tolerance as large as 10^{-6} because Wagon's function, while challenging in its three variables, is intrinsically of low multilinear rank. RTSMS and RHOSVDSMS are both fast, and again, the RTSMS error bound closely matches its actual observed residuals.

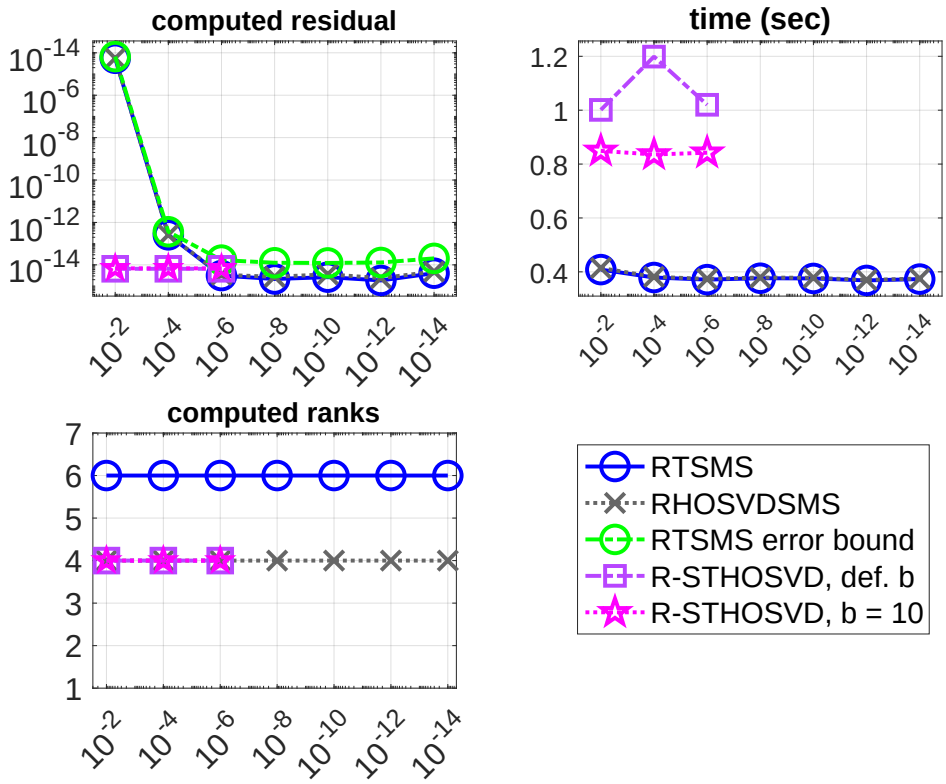

FIG. 5.2. *Comparison of rank-adaptive methods in terms of the residual, average multilinear ranks, and computing time for a tensor $\mathcal{A}$ of size $800 \times 1200 \times 300$ in Example 5.2. The horizontal axes are all the input tolerance.*

EXAMPLE 5.3. We take $\mathcal{A}$ to be samples of the function

$$f(x, y, z) = \sqrt{x^2 + y^2 + z^2}$$

ETNA
Kent State University and
Johann Radon Institute (RICAM)

on a Chebyshev grid of size $1000 \times 1000 \times 1000$ in $[-1, 1]^3$. f is called the Octant function in `cheb.gallery3` in Chebfun3 [25] and has nontrivial ranks. The results of our computations are presented in Figure 5.3. The size b of blocks used by `svdsketch` in R-STHOSVD is $\lfloor 0.1 n_i \rfloor = 100$, for $i = 1, 2, 3$, and the corresponding average of the computed multilinear ranks is 22.

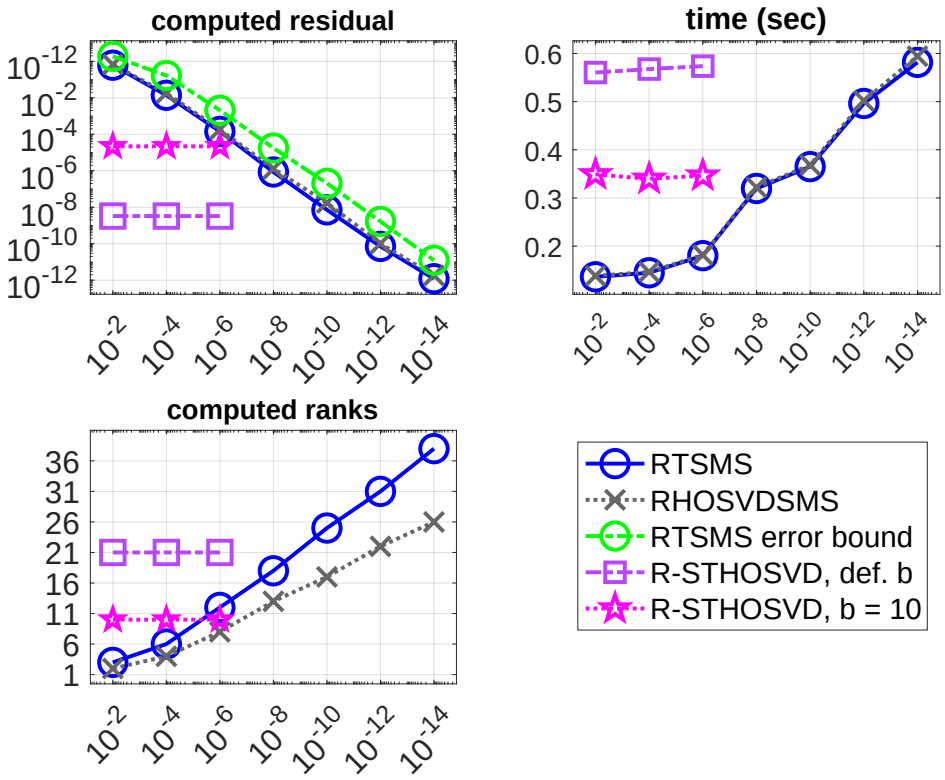

FIG. 5.3. *Comparison of methods in terms of the residual, average multilinear ranks, and computing time for a tensor $\mathcal{A}$ of size $1000 \times 1000 \times 1000$ in Example 5.3. The horizontal axes are all the input tolerance.*

Once again, RTSMS and RHOSVDSMS both perform very well in terms of speed, accuracy, and adjusting the rank to the given tolerance. Moreover, the RTSMS error bound is tight and successfully estimates its true residuals.

EXAMPLE 5.4. Our previous experiments involved isotropic functions having similar multilinear ranks across different modes. This example explores the methods for functions for which certain modes have higher ranks. We use this example mainly to illustrate the reliability of our randomized multilinear rank estimator, i.e., Steps 6–19 in Algorithm 5 when the tensor has different ranks over different direction. We do not plot the ranks computed with the `svdsketch`-based Algorithm 4, as the results remain constant for the first two input tolerances and then abruptly increase to the full tensor size for smaller tolerances.

Starting with the zero function f, we compute $f = f + g_k$, where

$$g_k(x, y, z) = \begin{cases} \tanh(ky - \frac{x}{2}) & \text{if } k \text{ is even,} \\ \tanh(ky - z) & \text{if } k \text{ is odd,} \end{cases}$$

where we take the values of k from 10 to 20. The multivariate hyperbolic function is known to

ETNA
Kent State University and
Johann Radon Institute (RICAM)

be challenging, which is why we use it here. While the variables x and z are present in half of the terms, y is present in all g_k's, hence one anticipates that the rank of f in the second direction (corresponding to y) is higher than the others. Our tensor $\mathcal{A}$ is the discretization of f on a Chebyshev grid of size $n = (100, 500, 100)$ in the domain $[-1, +1]^3$.

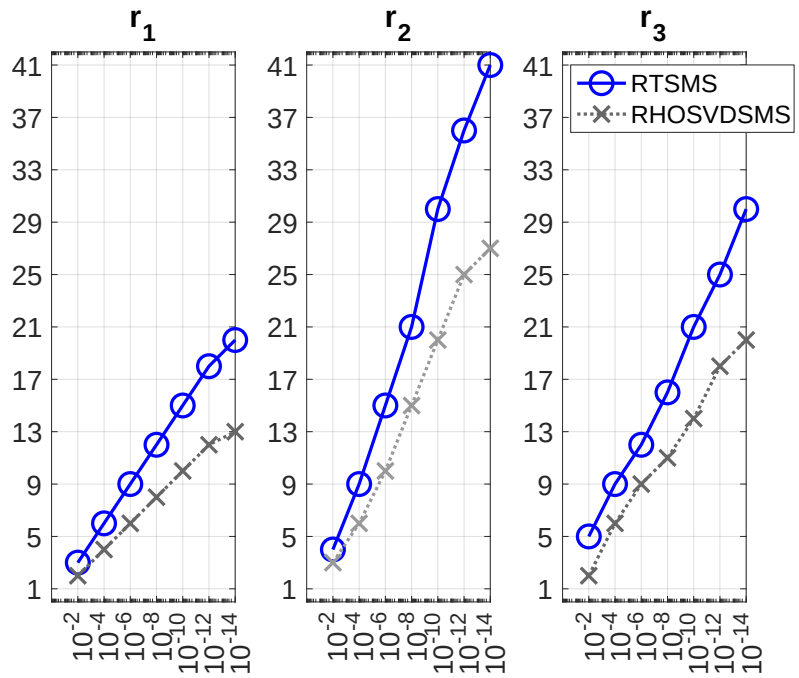

FIG. 5.4. *Different numerical multilinear ranks successfully estimated with the randomized rank estimator of RTSMS and RHOSVDSMS. $\mathcal{A}$ is of size* $100 \times 500 \times 100$ *in Example 5.4. The horizontal axes are all the input tolerance.*

The results are displayed in Figure 5.4. To confirm the reliability of the computed ranks, we take advantage of Chebfun3 finding out that f could be approximated up to the default tolerance of machine epsilon with a discretization of size $n = (18, 429, 27)$ and the numerical multilinear rank of $(r_1, r_2, r_3) = (13, 26, 19)$. This is in agreement especially with the ranks truncated by the HOSVD-variant of RTSMS, i.e., RHOSVDSMS, which computed the multilinear ranks of $(12, 25, 18)$ and $(13, 27, 20)$ for the smallest input tolerances of 10^{-12} and 10^{-14}, respectively.

EXAMPLE 5.5. We take $\mathcal{A}$ to be a 3D tensor of size $483 \times 720 \times 1280$ containing 483 frames of a video from the international space station. It corresponds to the first 16 seconds of a color video[8] which can be represented as a 4D tensor. However, we converted the color video to grayscale and only took the first 483 frames, creating an order-3 tensor $\mathcal{A}$.

We try RTSMS with two tolerances 10^{-2} and 10^{-3}. In the first case, a Tucker decomposition of multilinear rank $(80, 117, 131)$ is computed in 8.7 seconds with an actual relative residual of 2.2×10^{-1}. With tol $= 10^{-3}$, we get a Tucker decomposition of rank $(362, 552, 659)$ after 95.7 seconds with an actual relative residual of 3.9×10^{-2}. In this example we did not apply either of the two truncation strategies explained in Section 4. The

[8] https://www.youtube.com/watch?v=aIkWx6HGol0 retrieved September 13, 2023.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

variant of R-STHOSVD that, instead of the rank, takes a tolerance as input did not give an output after 5 minutes, after which we stopped its execution.

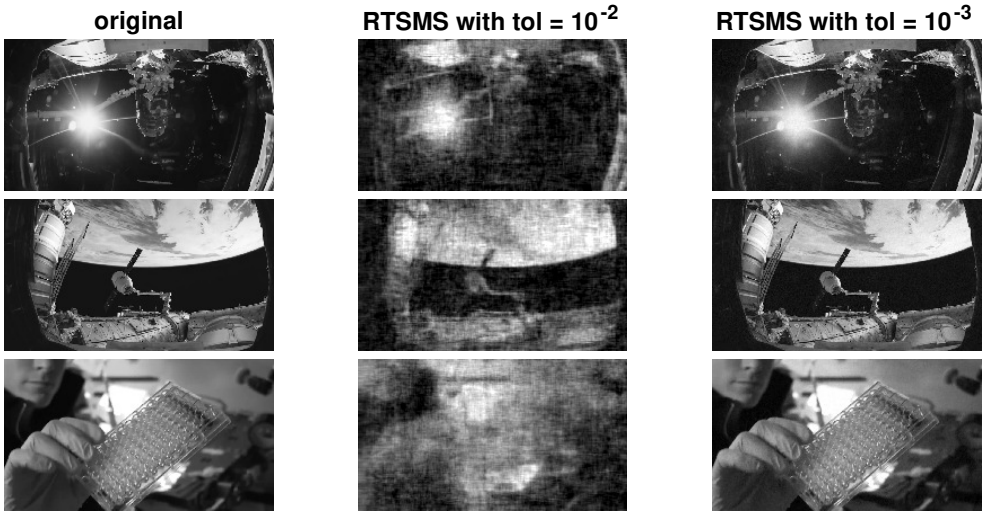

FIG. 5.5. *Frames 1 (top), 200 (middle), and 400 (bottom) from the International Space Station video and approximations obtained by RTSMS with two different tolerances. See Example 5.5. ©Copyrighted content. Creative commons license does not apply.*

The images are displayed in Figure 5.5. A 16-second video is available at
https://etna.ricam.oeaw.ac.at/volumes/2021-2030/vol63/addition/p247.php
as supplementary material showing this comparison for all 483 frames.

EXAMPLE 5.6. In our next example we work with a 3D tensor of size $2048 \times 256 \times 256$ from the Miranda Turbulent Flow tensor data from the Scientific Data Reduction Benchmark (SDRBench) [8, 58]. We acquired the dataset following the methodology outlined in [3] after which we apply the HOSVD variant of RTSMS with four tolerances, specifically $10^{-2}, 10^{-3}, 10^{-4}$, and 10^{-5}.

We report our results in Table 5.1. See also Figures 5.6 and 5.7 for illustrations. Here, the relative compression is the ratio of the total size of the original tensor X and the total storage required for the Tucker approximation.

We obtain a compression ratio of 5 requiring 20% of the size of the original tensor when the tolerance is set to 10^{-5}. On the other hand with a tolerance of 10^{-2}, a compression ratio of 4189 is achieved, requiring only 0.02% of the storage required for the original tensor. The rank-adaptive variant of R-STHOSVD is slightly slower but is more conservative giving higher ranks and hence lower compression ratio and more accuracy.

5.2. Fixed-rank experiments. In the following examples we examine algorithms which require the multilinear rank as input. While because of oversampling, the numerical Tucker rank of the approximations computed with RTSMS is more than the input rank $\mathbf{r}$, the method RHOSVDSMS truncates those approximations to the rank $\mathbf{r}$. As before we run each example five times and report the average time and residual.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

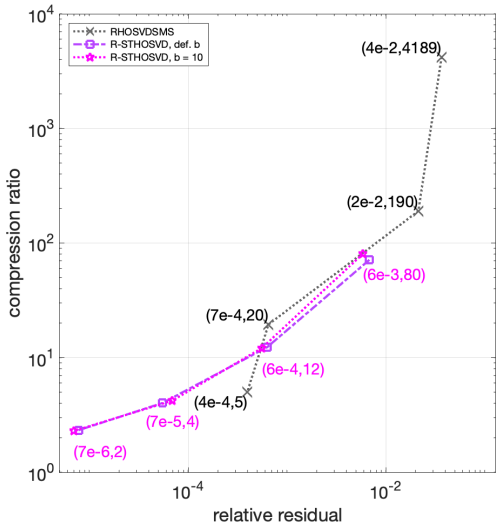

FIG. 5.6. *Compression versus residual obtained with different methods and different tolerances in Example 5.6. Coordinates are not displayed in the case of R-STHOSVD with the default block size; see also Table 5.1.*

TABLE 5.1
Relative error and compressions achieved for different tolerances in Example 5.6 .

tol	rel. error	Tucker rank	compression ratio	% of original size
10^{-2}	4×10^{-2}	(13, 9, 8)	4189	0.02
10^{-3}	2×10^{-2}	(155, 47, 50)	190	0.53
10^{-4}	7×10^{-4}	(490, 109, 109)	19.5	5
10^{-5}	4×10^{-4}	(866, 172, 168)	5	20

EXAMPLE 5.7. We consider the 4-dimensional Hilbert tensor of size $150 \times 150 \times 150 \times 150$, whose entries are

$$h_{i,j,k,l} = \frac{1}{i+j+k+l-3}.$$

We compute Tucker decompositions of multilinear rank (r, r, r, r) for the following six values of $r := 5, 10, 15, 20, 25, 30$. The results are depicted in Figure 5.8.

In addition to RTSMS, RHOSVDSMS, and R-STHOSVD (Algorithm 3), we also plot the results obtained by the multilinear generalized Nyström (MLN) method and its stabilized variant [7]. All the methods are comparable with RTSMS and RHOSVDSMS in terms of accuracy, while our algorithms are the best in terms of speed. Although, in comparison with Examples 5.1, 5.2, and 5.3, here the RTSMS error bound is a bit more conservative, it still tracks the behaviour of the observed residuals well.

EXAMPLE 5.8. Motivated by demo2 in the implementations accompanying [33], we construct synthetic data with noise as follows. Using Tensor Toolbox [2], we generate four $n \times n \times n$ tensors of true rank (r, r, r), where $n = 250, 500, 750, 1000$ and $r = 10, 12, 14, 16$, respectively. Then, Gaussian noise at the level of $10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}$ is added to the tensors, respectively. More precisely, noise at the level of 10^{-7} is added to the smallest tensor ($n = 250$), and similar noise at the level of 10^{-4} is applied to the largest tensor ($n = 1000$). We then call different methods to compute a Tucker decomposition of the noisy tensors repeating each experiment five times as before. Tucker-TS and Tucker-ALS require a

ETNA
Kent State University and
Johann Radon Institute (RICAM)

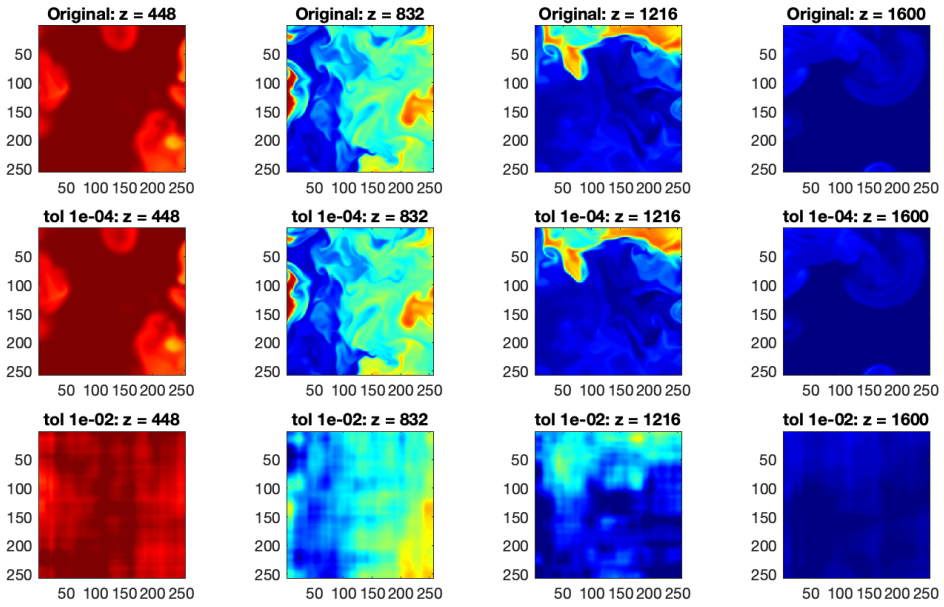

FIG. 5.7. *Visualization of four slices in the xy-plane of the original density tensor and compressed representations obtained with RHOSVDSMS in the numerical simulation of flows with turbulent mixing. Images produced with the tolerance of* 10^{-5} *look the same as those for* 10^{-4} *and hence are not displayed here. See Example 5.6.*

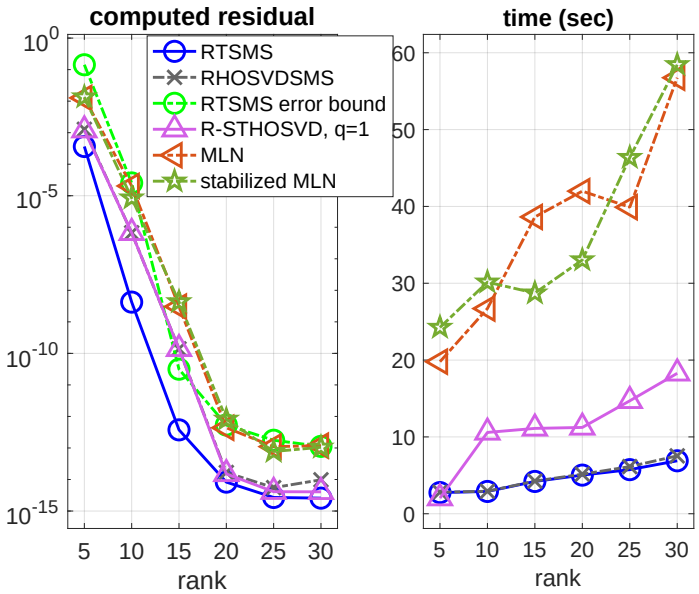

FIG. 5.8. *Comparison of fixed-rank algorithms for the 4D Hilbert tensor in terms of residual (left) and time (right) in Example 5.7.*

few parameters which we set as follows. The tolerance and target ranks are set to the same noise level and true ranks when generating each tensor as mentioned above. In addition, as recommended in [33], we set sketch the dimensions to $J_1 = Kr^2$ and $J_2 = Kr^3$ with $K = 10$.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

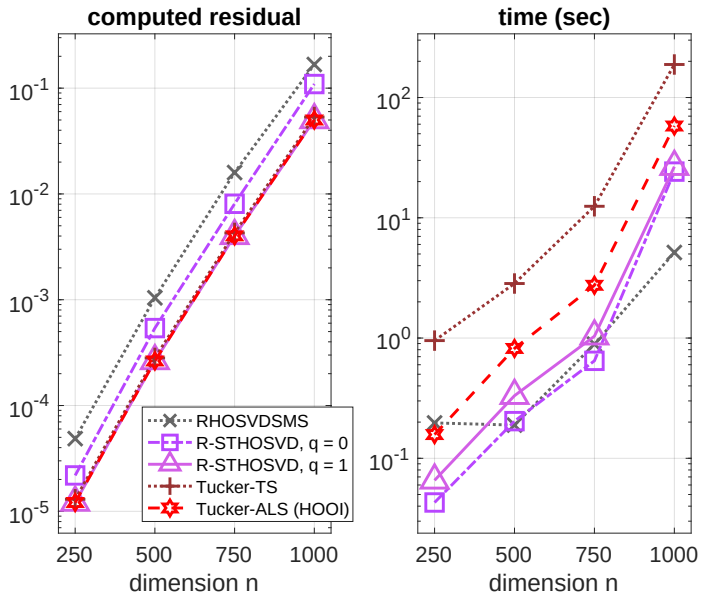

FIG. 5.9. *Comparison of fixed-rank algorithms for noisy synthetic data in terms of residual (left) and time (right) in Example 5.8.*

Figure 5.9 reports the outcome. While RHOSVDSMS gives a residual which is worse than R-STHOSVD by a factor of four, it is the fastest. In particular for the largest tensor, the average computing time of 110.5 and 21.2 seconds in Tucker-TS and R-STHOSVD, respectively, is reduced to 4.8 seconds.

We also note that we encountered errors in MATLAB when computing Khatri-Rao products involved in Tucker-TS (described in Section 3.1.1), complaining that memory required to generate the array exceeds the maximum array size preference. This happens, e.g., with $n = 1000$ and a rank as small as $r = 20$, in which case Tucker-TS generates an array which requires 36 GB of memory. No such errors arise with R-STHOSVD, RTSMS, and RHOSVDSMS. We tried Tucker-TTMTS as well, but it gave residuals at the constant level of 10^{-1} for all the four tensors, and that is why Tucker-TTMTS is not shown in the plots.

EXAMPLE 5.9. We take $\mathcal{A}$ to be the tensor of 80 snapshots from a computerized tomography (CT) kidney dataset of images[9]. More specifically, $\mathcal{A}$ is a tensor of size $512 \times 512 \times 80$ corresponding to images number 328 to 407 from the cyst directory in the dataset. We set the Tucker rank to be $(250, 250, 50)$. See Figure 5.10, suggesting that visually, R-STHOSVD with one power iteration and RTSMS give comparable approximations to the original images. In fact, RTSMS gives a mean relative residual of 1.21×10^{-1} compared with 1.07×10^{-1} with R-STHOSVD. The average time taken by R-STHOSVD and RTSMS is 4.05 and 2.60 seconds, respectively. We see that the images are approximated with roughly the same quality by all algorithms.

6. Conclusion. RTSMS incorporates randomized multilinear rank estimation into the sequential truncation approach for computing a Tucker decomposition. Sequential truncation implies that at every step, RTSMS computes low-rank approximation of a tensor whose size is smaller than those in the previous steps. Additionally, RTSMS relies on sketching and a least-squares framework. The sketch matrices utilized are substantially smaller than alternative

[9] https://www.kaggle.com/datasets/nazmul0087/ct-kidney-dataset-normal-cyst-tumor-and-stone

ETNA
Kent State University and
Johann Radon Institute (RICAM)

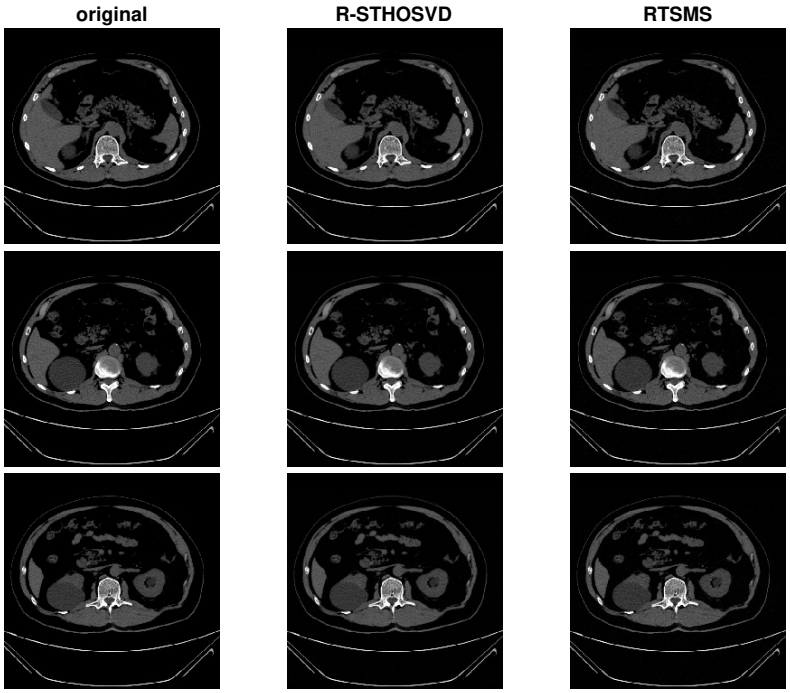

FIG. 5.10. *Original images (left) and approximations obtained by R-STHOSVD (middle) and RTSMS (right) corresponding to snapshots 1 (top row), 25 (middle row), and 50 (bottom row). See Example 5.9. ©Copyrighted content. Creative commons license does not apply.*

methods, leading to considerable performance gains. Within its least-squares framework, RTSMS takes advantage of approximate leverage scores to subsample efficiently.

On the other hand, error analysis highlights the importance of keeping the norm of the factor matrices as small as possible. Since factor matrices are solutions to sketched least-squares subproblems, RTSMS addresses this need by using Tikhonov regularization and improves the robustness with iterative refinement. Future work includes a stability analysis of RTSMS and explaining the sharpness of the bound (4.8), as well as its extension to the streaming model, where the input is given as a stream of structured tensors.

Supplementary material. The supplementary material accompanying this article is a video (MP4 format) illustrating the results in Figure 5.5 and can be found at
https://etna.ricam.oeaw.ac.at/volumes/2021-2030/vol63/addition/p247.php

Acknowledgments. We would like to thank Tammy Kolda for the insightful discussions and helpful suggestions, including the experiments in Figure 5.7. We are also grateful to Daniel Szyld for his encouragement and for suggesting the inclusion of the error bound (4.8) in the relevant plots.

Appendix A. Deterministic STHOSVD. Here we first recall the basic ideas of HOSVD and STHOSVD, whose foundations are key to RTSMS. Also, for the sake of completeness, we present standard deterministic algorithms for HOSVD and STHOSVD as well as our algorithm for converting Tucker decompositions computed by RTSMS to HOSVD.

Let $n_i \gg r_i$, for $i = 1, 2, \ldots, d$, and let $U_i \in \mathbb{R}^{n_i \times r_i}$ be *any* full-rank matrix whose columns span the column space of $A_{(i)}$, which is a subspace of $\mathbb{R}^{n_i}$. Hence, each U_i has a left-inverse, i.e., $U_i^\dagger U_i = I_{r_i} \in \mathbb{R}^{r_i \times r_i}$, and $U_i U_i^\dagger =: P_i \in \mathbb{R}^{n_i \times n_i}$ is a projection onto the

ETNA
Kent State University and
Johann Radon Institute (RICAM)

column space of U_i and $A_{(i)}$. Hence, $P_iU_i = U_i$ and $P_iA_{(i)} = A_{(i)}$. Using Definition 2.1, the latter formula can be rewritten as

$$A_{(i)} = (\mathcal{A} \times_i P_i)_{(i)}.$$

Repeating this d times, we therefore have

$$\begin{aligned}\mathcal{A} &= \mathcal{A} \times_1 P_1 \times_2 P_2 \cdots \times_d P_d \\ &= \mathcal{A} \times_1 U_1U_1^\dagger \times_2 U_2U_2^\dagger \cdots \times_d U_dU_d^\dagger \\ &= (\mathcal{A} \times_1 U_1^\dagger \times_2 U_2^\dagger \cdots \times_d U_d^\dagger) \times_1 U_1 \times_2 U_2 \cdots \times_d U_d, \end{aligned} \tag{A.1}$$

where the last equality is based on (2.1). Formula (A.1) implies that if we chose the matrices U_i as the factor matrices and take

$$\mathcal{C} := \mathcal{A} \times_1 U_1^\dagger \times_2 U_2^\dagger \cdots \times_d U_d^\dagger \tag{A.2}$$

as an $r_1 \times r_2 \cdots \times r_d$ core tensor, then we have the following Tucker decomposition:

$$\mathcal{A} = \mathcal{C} \times_1 U_1 \times_2 U_2 \cdots \times_d U_d,$$

where the equality is exact as long as r_i is larger than or equal to the rank of the column space of $A_{(i)}$.

The formulation of HOSVD in (2.2) is equivalent to

$$\operatorname{vec}(\mathcal{A}) = (U_d \otimes \cdots \otimes U_2 \otimes U_1) \operatorname{vec}(\mathcal{C}). \tag{A.3}$$

See [21, Equation (12.4.19)] for instance.

From (A.3) and (A.2) it is clear that the computation of the core tensor $\mathcal{C}$ is equivalent to solving a huge overdetermined linear system of equations of the form

$$(U_d \otimes \cdots \otimes U_2 \otimes U_1)c = a,$$

where $c := \operatorname{vec}(\mathcal{C})$ is a vector of size $(r_1r_2 \dots r_d) \times 1$, $a := \operatorname{vec}(\mathcal{A})$ is a vector of size $N \times 1$, with $N := n_1n_2 \dots n_d$, and the coefficient matrix containing Kronecker products is of size $N \times (r_1r_2 \dots r_d)$.

In the case of the deterministic HOSVD, the matrices U_i are chosen to be the left singular vectors of $A_{(i)}$, and the computation of the core relies on the orthogonality of the columns of every U_i; see Algorithm 7. In the case of the STHOSVD, U_i is chosen to be the left singular vectors of the previously-truncated tensor $\hat{\mathcal{C}}^{(i)}$.

Algorithm 7 Deterministic HOSVD (De Lathauwer, De Moor and Vandewalle, 2000 [15])
Inputs are $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ and the truncation rank $(r_1, r_2, \dots, r_d)$.
Output is $\mathcal{A} \approx [\![\mathcal{C}; U_1, U_2, \dots, U_d]\!]$.

1: **for** $i = 1, \dots, d$ **do**
2: Compute the thin SVD $A_{(i)} = \begin{bmatrix}\hat{U}_1 & \hat{U}_2\end{bmatrix} \begin{bmatrix}\Sigma_1 & \\ & \Sigma_2\end{bmatrix} \begin{bmatrix}V_1^T \\ V_2^T\end{bmatrix}$, where $\hat{U}_1 \in \mathbb{R}^{n_i \times r_i}$.
3: Set $U_i := \hat{U}_1$.
4: **end for**
5: Compute $\mathcal{C} = \mathcal{A} \times_1 U_1^T \times_2 U_2^T \cdots \times_d U_d^T$.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

Algorithm 8 Deterministic STHOSVD (Vannieuwenhoven, Vandebril, and Meerbergen, 2012 [53])
Inputs are $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, the truncation rank $(r_1, r_2, \ldots, r_d)$, and the processing order $\mathbf{p}$ of the modes (a permutation of $[1, 2, \ldots, d]$).
Output is $\mathcal{A} \approx [\![\mathcal{C}; U_1, U_2, \ldots, U_d]\!]$.

1: Set $\mathcal{C} := \mathcal{A}$.
2: **for** $i = p_1, \ldots, p_d$ **do**
3: Compute the thin SVD $C_{(i)} = \begin{bmatrix} \hat{U}_1 & \hat{U}_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}$, where $\hat{U}_1 \in \mathbb{R}^{n_i \times r_i}$.
4: Set $U_i := \hat{U}_1$.
5: Compute $C_{(i)} = \Sigma_1 V_1^T$.
6: **end for**

Appendix B. Tucker to HOSVD conversion. As mentioned in Section 4.5, converting the Tucker decomposition computed via RTSMS to the HOSVD format—where factor matrices possess orthonormal columns and the core tensor is all-orthogonal—is a straightforward process [15], based on orthonormalizing the factor matrices with a QR factorization, merging the R factors in the core tensor, and recompressing the updated core tensor for further rank truncation. Algorithm 9 is a standard deterministic approach to accomplishing this conversion.

Algorithm 9 `Tucker2HOSVD` (with or without thresholding)
Inputs are a Tucker decomposition $\mathcal{A} \approx [\![\mathcal{C}; F_1, F_2, \ldots, F_d]\!]$ whose multilinear rank is $\hat{\mathbf{r}}$ (and a tolerance tol if thresholding).
Output is HOSVD $\mathcal{A} \approx [\![\check{\mathcal{C}}; U_1, U_2, \ldots, U_d]\!]$ whose multilinear rank is either $\hat{\mathbf{r}}$ in case of no thresholding or $\mathbf{l}$ in the case of thresholding with $\ell_i \leq \hat{r}_i$.

1: **for** $i = 1, 2, \ldots, d$ **do**
2: Compute thin QR factorizations $[Q_i, R_i] = qr(F_i)$.
3: **end for**
4: Update $\mathcal{C} := \mathcal{C} \times_1 R_1 \times_2 R_2 \cdots \times_d R_d$.
5: Apply a deterministic STHOSVD to $\mathcal{C}$, and compute $[\![\check{\mathcal{C}}; \check{U}_1, \check{U}_2, \ldots, \check{U}_d]\!] \approx \mathcal{C}$ (and the mode-i higher order singular values $\sigma^{(i)}$ if thresholding). {Both $\mathcal{C}$ and $\check{\mathcal{C}}$ are of size $\hat{r}_1 \times \hat{r}_2 \cdots \times \hat{r}_d$.}
6: **if** thresholding **then**
7: **for** $i = 1, 2, \ldots, d$ **do**
8: Find the smallest ℓ_i such that $\sigma^{(i)}_{\ell_i+1} < \text{tol}\ \sigma^{(i)}_1$.
9: **end for**
10: **else**
11: Set $l_i := \hat{r}_i$ for $i = 1, 2, \ldots, d$.
12: **end if**
13: **for** $i = 1, 2, \ldots, d$ **do**
14: Compute $U_i = Q_i \check{U}_i(:, 1 : \ell_i)$. {$Q_i$ is of size $n_i \times \hat{r}_i$, $\check{U}_i$ is $\hat{r}_i \times \hat{r}_i$, and U_i is $n_i \times \ell_i$.}
15: **end for**
16: Replace $\check{\mathcal{C}}$ with $\check{\mathcal{C}}(1 : \ell_1, 1 : \ell_2, \ldots, 1 : \ell_d)$.

Appendix C. Higher order GN. Algorithm 10 is a higher-order generalized Nyström method for computing a Tucker decomposition in a sequentially truncated manner. It relies on the generalized Nyström framework for randomized low-rank approximation of unfolding

ETNA
Kent State University and
Johann Radon Institute (RICAM)

matrices as outlined in Algorithm 1 and is not the same as the multilinear Nyström algorithm in [7]. See also [6] for a variant of the higher order GN method that is suitable for tensors given in a streaming format.

Algorithm 10 `R-GN-ST-Tucker`
Inputs are $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$, a target multilinear rank $(r_1, r_2, \ldots, r_d)$, and a processing order $\mathbf{p}$ of the modes.
Output is $\mathcal{A} \approx [\![\mathcal{C}; U_1, U_2, \ldots, U_d]\!]$.

1: Set $\mathcal{C} := \mathcal{A}$.
2: **for** $i = p_1, \ldots, p_d$ **do**
3: Draw two standard random Gaussian matrices Ω_i and $\tilde{\Omega}_i$ of size $z_i \times r_i$ and $n_i \times \hat{r}_i$, respectively, where $\hat{r}_i := r_i + p$ with $p := \lceil r_i/2 \rceil$.
4: Compute $[\hat{U}, \hat{V}] = \mathtt{GN}(C_{(i)}, \Omega_i, \tilde{\Omega}_i)$ using Algorithm 1.
5: Set $U_i := \hat{U}$.
6: Update $C_{(i)} = \hat{V}^T$. {Overwriting $C_{(i)}$ overwrites $\mathcal{C}$.}
7: **end for**

REFERENCES

[1] H. AVRON AND U. MOR, *Higher order generalized Nystrom approximation*, in 2023 SIAM Conference on Computational Science and Engineering, SIAM, Amsterdam, 2023, p. 382.
[2] B. W. BADER AND T. G. KOLDA, *Algorithm 862: MATLAB tensor classes for fast algorithm prototyping*, ACM Trans. Math. Software, 32 (2006), pp. 635–653.
[3] G. BALLARD, T. G. KOLDA, AND P. LINDSTROM, *Miranda turbulent flow dataset*, 2022. https://gitlab.com/tensors/tensor_data_miranda_sim
[4] J. D. BATSON, D. A. SPIELMAN, AND N. SRIVASTAVA, *Twice-Ramanujan sparsifiers*, in STOC'09—Proceedings of the 2009 ACM International Symposium on Theory of Computing, ACM, New York, 2009, pp. 255–262.
[5] F. BORNEMANN, D. LAURIE, S. WAGON, AND J. WALDVOGEL, *The SIAM 100-Digit Challenge*, SIAM, Philadelphia, 2004.
[6] A. BUCCI AND B. HASHEMI, *A sequential multilinear Nyström algorithm for streaming low-rank approximation of tensors in Tucker format*, Appl. Math. Lett., 159 (2025), Paper No. 109271, 5 pages.
[7] A. BUCCI AND L. ROBOL, *A multilinear Nyström algorithm for low-rank approximation of tensors in Tucker format*, SIAM J. Matrix Anal. Appl., 45 (2024), pp. 1929–1953.
[8] W. CABOT AND A. COOK, *Reynolds number effects on Rayleigh–Taylor instability with possible implications for type 1a supernovae*, Nature Physics, 2 (2006), pp. 562–568.
[9] J.-F. CAI, E. J. CANDÈS, AND Z. SHEN, *A singular value thresholding algorithm for matrix completion*, SIAM J. Optim., 20 (2010), pp. 1956–1982.
[10] C. F. CAIAFA AND A. CICHOCKI, *Generalizing the column-row matrix decomposition to multi-way arrays*, Linear Algebra Appl., 433 (2010), pp. 557–573.
[11] S. CHATURANTABUT AND D. C. SORENSEN, *Nonlinear model reduction via discrete empirical interpolation*, SIAM J. Sci. Comput., 32 (2010), pp. 2737–2764.
[12] M. CHE, Y. WEI, AND H. YAN, *The computation of low multilinear rank approximations of tensors via power scheme and random projection*, SIAM J. Matrix Anal. Appl., 41 (2020), pp. 605–636.
[13] K. L. CLARKSON AND D. P. WOODRUFF, *Low-rank approximation and regression in input sparsity time*, J. ACM, 63 (2017), Paper No. 54, 45 pages.
[14] K. R. DAVIDSON AND S. J. SZAREK, *Local operator theory, random matrices and Banach spaces*, in Handbook of the Geometry of Banach Spaces, Vol. I, W. B. Johnson and J. Lindenstrauss, eds., North-Holland, Amsterdam, 2001, pp. 317–366.
[15] L. DE LATHAUWER, B. DE MOOR, AND J. VANDEWALLE, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278.
[16] ———, *On the best rank-1 and rank-$(R_1, R_2, \cdots, R_N)$ approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

[17] S. DOLGOV, D. KRESSNER, AND C. STRÖSSNER, *Functional Tucker approximation using Chebyshev interpolation*, SIAM J. Sci. Comput., 43 (2021), pp. A2190–A2210.
[18] Y. DONG AND P.-G. MARTINSSON, *Simpler is better: a comparative study of randomized algorithms for computing the CUR decomposition*, Preprint on arXiv, 2021. https://arxiv.org/abs/2104.05877
[19] P. DRINEAS, M. MAGDON-ISMAIL, M. W. MAHONEY, AND D. P. WOODRUFF, *Fast approximation of matrix coherence and statistical leverage*, J. Mach. Learn. Res., 13 (2012), pp. 3475–3506.
[20] M. FAHRBACH, G. FU, AND M. GHADIRI, *Subquadratic Kronecker regression with applications to tensor decomposition*, in Advances in Neural Information Processing Systems 35, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds., Curran Associates, Red Hook, 2022, pp. 28776–28789.
[21] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, 1996.
[22] 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.
[23] 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.
[24] P. C. HANSEN, *Rank-Deficient and Discrete Ill-Posed Problems*, SIAM Philadelphia, 1998.
[25] B. HASHEMI AND L. N. TREFETHEN, *Chebfun in three dimensions*, SIAM J. Sci. Comput., 39 (2017), pp. C341–C363.
[26] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.
[27] ———, *Functions of Matrices*, SIAM, Philadelphia, 2008.
[28] T. G. KOLDA AND B. W. BADER, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.
[29] K. KOUR, S. DOLGOV, P. BENNER, M. STOLL, AND M. PFEFFER, *A weighted subspace exponential kernel for support tensor machines*, Preprint on arXiv 2023. https://arxiv.org/abs/2302.08134
[30] P. M. KROONENBERG, *Applied Multiway Data Analysis*, Wiley-Interscience, Hoboken, 2008.
[31] B. W. LARSEN AND T. G. KOLDA, *Practical leverage-based sampling for low-rank tensor decomposition*, SIAM J. Matrix Anal. Appl., 43 (2022), pp. 1488–1517.
[32] M. W. MAHONEY, *Randomized algorithms for matrices and data*, Preprint on arXiv, 2011. https://arxiv.org/abs/1104.5557
[33] O. A. MALIK AND S. BECKER, *Low-rank Tucker decomposition of large tensors using tensorsketch*, in Advances in Neural Information Processing Systems 31, Curran Associates, Red Hook, 2018.
[34] P.-G. MARTINSSON AND J. A. TROPP, *Randomized numerical linear algebra: foundations and algorithms*, Acta Numer., 29 (2020), pp. 403–572.
[35] 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.
[36] M. MEIER AND Y. NAKATSUKASA, *Fast randomized numerical rank estimation*, Preprint on arXiv, 2021. https://arxiv.org/abs/2105.07388
[37] M. MEIER, Y. NAKATSUKASA, A. TOWNSEND, AND M. WEBB, *Are sketch-and-precondition least squares solvers numerically stable?*, Preprint on arXiv, 2023. https://arxiv.org/abs/2302.07202
[38] R. MINSTER, Z. LI, AND G. BALLARD, *Parallel randomized Tucker decomposition algorithms*, Preprint on arXiv, 2023. https://arxiv.org/abs/2211.13028
[39] R. MINSTER, A. K. SAIBABA, AND M. E. KILMER, *Randomized algorithms for low-rank tensor decompositions in the Tucker format*, SIAM J. Math. Data Sci., 2 (2020), pp. 189–215.
[40] R. MURRAY, J. DEMMEL, M. W. MAHONEY, N. B. ERICHSON, M. MELNICHENKO, O. A. MALIK, L. GRIGORI, P. LUSZCZEK, M. DEREZIŃSKI, M. E. LOPES, T. LIANG, H. LUO, AND J. DONGARRA, *Randomized numerical linear algebra: a perspective on the field with an eye to software*, Preprint on arXiv, 2023. https://arxiv.org/abs/2302.11474
[41] Y. NAKATSUKASA, *Fast and stable randomized low-rank matrix approximation*, Preprint on arXiv, 2020. https://arxiv.org/abs/2009.11392
[42] Y. NAKATSUKASA AND J. A. TROPP, *Fast & accurate randomized algorithms for linear systems and eigenvalue problems*, Preprint on arXiv, 2022. https://arxiv.org/abs/2111.00113
[43] I. V. OSELEDETS, D. V. SAVOSTIANOV, AND E. E. TYRTYSHNIKOV, *Tucker dimensionality reduction of three-dimensional arrays in linear time*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 939–956.
[44] R. PAGH, *Compressed matrix multiplication*, ACM Trans. Comput. Theory, 5 (2013), Paper No. 9, 17 pages.
[45] J. PAN, M. K. NG, Y. LIU, X. ZHANG, AND H. YAN, *Orthogonal nonnegative Tucker decomposition*, SIAM J. Sci. Comput., 43 (2021), pp. B55–B81.
[46] L. A. PASTUR AND V. A. MARCHENKO, *The distribution of eigenvalues in certain sets of random matrices*, Math. USSR Sb., 1 (1967), pp. 457–483.
[47] Y. SUN, Y. GUO, C. LUO, J. TROPP, AND M. UDELL, *Low-rank Tucker approximation of a tensor from streaming data*, SIAM J. Math. Data Sci., 2 (2020), pp. 1123–1150.

ETNA
Kent State University and
Johann Radon Institute (RICAM)

[48] D. B. SZYLD, *The many proofs of an identity on the norm of oblique projections*, Numer. Algorithms, 42 (2006), pp. 309–323.
[49] J. A. TROPP, *Improved analysis of the subsampled randomized Hadamard transform*, Adv. Adapt. Data Anal., 3 (2011), pp. 115–126.
[50] J. A. TROPP, A. YURTSEVER, M. UDELL, AND V. CEVHER, *Practical sketching algorithms for low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1454–1485.
[51] L. R. TUCKER, *Implications of factor analysis of three-way matrices for measurement of change*, in Problems in Measuring Change, C. W. Harris, ed., University of Wisconsin Press, Madison, 1963, pp. 122–137.
[52] ———, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.
[53] 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.
[54] M. A. O. VASILESCU, *A Multilinear (Tensor) Algebraic Framework for Computer Graphics, Computer Vision, and Machine Learning*, Ph.D. Thesis, Department of Computer Science, University of Toronto, 2009.
[55] D. P. WOODRUFF, *Sketching as a tool for numerical linear algebra*, Found. Trends Theor. Comput. Sci., 10 (2014), pp. iv+157.
[56] F. WOOLFE, E. LIBERTY, V. ROKHLIN, AND M. TYGERT, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 335–366.
[57] 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.
[58] K. ZHAO, S. DI, X. LIAN, S. LI, D. TAO, J. BESSAC, Z. CHEN, AND F. CAPPELLO, *SDRBench: scientific data reduction benchmark for lossy compressors*, in 2020 IEEE International Conference on Big Data (Big Data), IEEE Conference Proceedings, Los Alamitos, 2020, pp. 2716–2724.
[59] G. ZHOU, A. CICHOCKI, AND S. XIE, *Decomposition of big tensors with low multilinear rank*, Preprint on arXiv, 2014. https://arxiv.org/abs/1412.1885