SIAM J. MATRIX ANAL. APPL.
Vol. 39, No. 2, pp. 876–901

A PRACTICAL RANDOMIZED CP TENSOR DECOMPOSITION*

CASEY BATTAGLINO†, GREY BALLARD‡, AND TAMARA G. KOLDA§

Abstract. The CANDECOMP/PARAFAC (CP) decomposition is a leading method for the analysis of multiway data. The standard alternating least squares algorithm for the CP decomposition (CP-ALS) involves a series of highly overdetermined linear least squares problems. We extend randomized least squares methods to tensors and show that the workload of CP-ALS can be drastically reduced without a sacrifice in quality. We introduce techniques for efficiently preprocessing, sampling, and computing randomized least squares on a dense tensor of arbitrary order, as well as an efficient sampling-based technique for checking the stopping condition. We also show more generally that the Khatri–Rao product (used within the CP-ALS iteration) produces conditions favorable for direct sampling. In numerical results, we see improvements in speed, reductions in memory requirements, and robustness with respect to initialization.

Key words. canonical polyadic tensor decomposition, CANDECOMP/PARAFAC (CP), multilinear algebra, randomized algorithms, randomized least squares

AMS subject classifications. 15A69, 68W20

DOI. 10.1137/17M1112303

1. Introduction.

The CANDECOMP/PARAFAC (CP) tensor decomposition is an important tool for data analysis in applications such as chemometrics [27], biogeochemistry [21], neuroscience [1, 15, 14], signal processing [36], cyber traffic analysis [25], and many others. We consider the problem of accelerating the alternating least squares (CP-ALS) algorithm using randomization.

Because randomized methods have been used successfully for solving linear least squares problems [17, 3, 44], it is natural that they might prove beneficial to CP-ALS since its key kernel is the solution of a least squares problem. However, the CP-ALS least squares subproblem has a special structure that already greatly reduces its cost, so it is unclear whether or not sketching would be beneficial. Nevertheless, we find that our randomized algorithms significantly reduce the memory and computational overhead of the CP-ALS process for dense tensors and, moreover, positively impact algorithmic robustness. To the best of our knowledge, this is the first successful application of matrix sketching methods in the context of CP. The contributions of this paper are as follows:

- The least squares coefficient matrix in the CP-ALS subproblem is a Khatri–Rao product of factor matrices. Our randomized algorithm prefers *incoherent*

*Received by the editors January 19, 2017; accepted for publication (in revised form) February 20, 2018; published electronically May 22, 2018. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

http://www.siam.org/journals/simax/39-2/M111230.html

Funding: This work was supported by the Sandia Truman Postdoctoral Fellowship and the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

†Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA 30332 (cbattaglino3@gatech.edu).

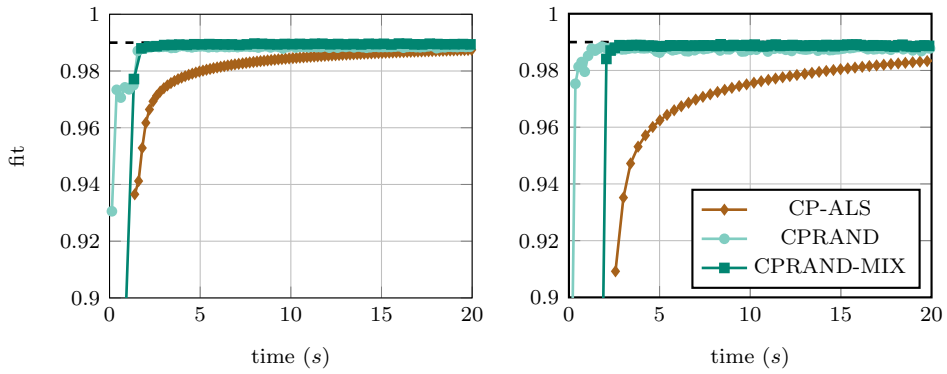
‡Computer Science, Wake Forest University, Winston-Salem, NC 27109 (ballard@wfu.edu).

§Sandia National Laboratories, Livermore, CA 94551-9159 (tgkolda@sandia.gov).

matrices. We prove that the coherence of the Khatri–Rao product is bounded above by the product of the coherence of its factors.

- We introduce the CPRAND algorithm that uses a randomized least squares solver for the subproblems in CP-ALS and never explicitly forms the full Khatri–Rao matrices used in the subproblems. We also introduce the complementary CPRAND-MIX algorithm that employs efficient *mixing* to promote incoherence and thereby improves the robustness of the method.
- We derive a novel, lightweight stopping condition that estimates the model fit error, and we prove its accuracy using Chernoff–Hoeffding bounds.
- We demonstrate the speed and robustness of our algorithms over a large number of synthetic tensors as well as real-world data sets. In comparison with CP-ALS, CPRAND is faster and much less sensitive to the starting point.

We give an example of our methods' fast time to solution in Figure 1, comparing CPRAND and CPRAND-MIX with CP-ALS. For the CPRAND methods, we use 100 sampled rows for each least squares solve. The randomized methods converge much more quickly, in only a few iterations. The fit is not monotonically increasing for the randomized methods due to (small) variations in the solution to each randomized subproblem. See section 4 for full details on problem generation and further experiments.



(a) Random $300 \times 300 \times 300$ tensor (b) Random $80 \times 80 \times 80 \times 80$ tensor

FIG. 1. *Runtime comparison for fitting the CP tensor decomposition on random synthetic tensors generated to have rank* 5*, a factor collinearity of* 0.9*, and* 1% *noise. We compare a single run of three methods using a target rank of* 5*. CPRAND and CPRAND-MIX use random initialization,* 100 *sampled rows for each least squares solve. CP-ALS uses HOSVD initialization. The marks indicate each iteration. The thin dashed black line represents a fit of* 99%*, which is the best we expect when the noise is* 1%*.*

2. Background and definitions.

In this section we provide information on key matrix and tensor operations, as well as randomized least squares.

2.1. Matrix and tensor background.

A tensor is an element in a tensor product of one or more vector spaces. In data analysis it suffices to think about a tensor as a multidimensional array. We represent a tensor as a Euler script capital letter, e.g., $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. The number of *modes* (or dimensions) is referred to as the *order*, denoted here by N. The *mode-n fibers* of a tensor are the higher-order analogue of matrix column and row vectors. The mode-n *unfolding* or *matricization*

of a tensor aligns the mode-n fibers as the columns of an $I_n \times \prod_{m \neq n} I_m$ matrix. Assuming 1-indexing, tensor entry $x_{i_1,i_2,\ldots,i_N}$ then maps to entry (i_n, j) of $\mathbf{X}_{(n)}$ via the relation

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k, \qquad \text{where} \qquad J_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m. \tag{1}$$

Given matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$, their *Kronecker product* is

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{IK \times JL}.$$

Assuming $J = L$, their *Khatri–Rao product*, also known as the *matching columnwise Kronecker product*, is

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_J \otimes \mathbf{b}_J \end{bmatrix}.$$

Assuming $I = K$ and $J = L$, their *Hadamard product* is $\mathbf{A} \circledast \mathbf{B} \in \mathbb{R}^{I \times J}$, the elementwise product of the matrices. Three useful identities are

$$(\mathbf{A} \odot \mathbf{B})^{\mathsf{T}} (\mathbf{A} \odot \mathbf{B}) = \mathbf{A}^{\mathsf{T}}\mathbf{A} \circledast \mathbf{B}^{\mathsf{T}}\mathbf{B}, \tag{2}$$

$$\mathbf{AB} \otimes \mathbf{CD} = (\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \otimes \mathbf{D}), \tag{3}$$

$$\mathbf{AB} \odot \mathbf{CD} = (\mathbf{A} \otimes \mathbf{C})(\mathbf{B} \odot \mathbf{D}). \tag{4}$$

The *mode-n tensor-times-matrix product* is a contraction between a matrix and a tensor in its nth mode. The two operations below are equivalent and can be computed in place (i.e., without explicitly unfolding $\mathbf{X}_{(n)}$) [24]:

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{A} \quad \Leftrightarrow \quad \mathbf{Y}_{(n)} = \mathbf{A}\mathbf{X}_{(n)}.$$

If a tensor is followed by a series of mode-n products, its mode-n matricization has a particular structure [23] that will be useful in the discussion of "mixing" in subsection 3.3:

$$\begin{aligned} &\mathcal{Y} = \mathcal{X} \times_1 \mathbf{U}^{(1)} \cdots \times_N \mathbf{U}^{(N)} \\ \Leftrightarrow\ &\mathbf{Y}_{(n)} = \mathbf{U}^{(n)} \mathbf{X}_{(n)} (\mathbf{U}^{(N)} \otimes \cdots \otimes \mathbf{U}^{(n+1)} \otimes \mathbf{U}^{(n-1)} \otimes \cdots \otimes \mathbf{U}^{(1)})^{\mathsf{T}}. \end{aligned} \tag{5}$$

The CP tensor decomposition aims to approximate an order-N tensor as a sum of R rank-one tensors [20, 10, 19, 23]:

$$\mathcal{X} \approx \tilde{\mathcal{X}} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}, \tag{6}$$

where *factor vector* $\mathbf{a}_r^{(n)}$ has length I_n. Each rank-one tensor is called a *component*. The collection of all factor vectors for a given mode is called a *factor matrix*:

$$\mathbf{A}^{(n)} = \begin{bmatrix} \mathbf{a}_1^{(n)} & \mathbf{a}_2^{(n)} & \cdots & \mathbf{a}_r^{(n)} \end{bmatrix} \in \mathbb{R}^{I_n \times R}.$$

The mode-n matricization of $\tilde{\mathcal{X}}$ can be written in terms of the factor matrices as

$$\tilde{\mathbf{X}}_{(n)} = \mathbf{A}^{(n)} \mathbf{Z}^{(n)\mathsf{T}}, \quad \text{where} \quad \mathbf{Z}^{(n)} = \mathbf{A}^{(N)} \odot \cdots \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}. \tag{7}$$

We may alternatively represent (6) by normalizing all the factor vectors to unit length and expressing the product of the normalization factors as a scalar weight λ_r for each component:

$$\tilde{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r \, \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}. \tag{8}$$

2.2. Randomized least squares and sketching. Sketching is a technique for solving linear algebra problems by constructing a smaller problem whose solution is a reasonable approximation to the original problem with high probability [44]. For instance, a large matrix may be formed by applying random sampling or random projections to form a smaller *sketch* matrix. We focus on the case where a regression problem $\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_2$ (with overdetermined $\mathbf{A} \in \mathbb{R}^{n \times d}$) is transformed using some random projection $\mathbf{M} \in \mathbb{R}^{S \times n}$, with $S \ll n$, such that an exact solution to $\min_{\mathbf{x}} \|\mathbf{MAx} - \mathbf{Mb}\|_2$ is an approximate solution to the original problem [35, 17, 3].

There are two leading sampling approaches for randomized least squares problems. One involves sampling from the coefficient matrix in a weighted manner, e.g., by computing (or estimating) *leverage scores* for each row and sampling based on their distribution. The other approach, which we use in this work, is to *mix* the coefficient matrix with the intention of evenly distributing leverage scores across all rows in such a way that *uniform* sampling is effective.

DEFINITION 1. *Given* $\mathbf{A} \in \mathbb{R}^{n \times d}$, $n > d$, *the* leverage score *of row* i *of* $\mathbf{A}$ *is* $l_i = ||\mathbf{U}(i,:)||_2^2$ *for* $i \in \{1, \ldots, n\}$, *where* $\mathbf{U}$ *contains the* d *left singular vectors of* $\mathbf{A}$.

Thus, the leverage score of a row corresponds in some sense to the importance of that row in constructing the column space of the coefficient matrix.

In 2007, Drineas et al. [17] presented a relative-error least squares algorithm that gives a $1+\epsilon$ approximation. They first mix the coefficient matrix using a randomized Hadamard transform (discussed later) and then sample

$$\mathcal{O}(\max\{d \log(n) \log(d \log(n)), d \log(nd)/\epsilon\}) \tag{9}$$

rows of the resulting matrix before computing the solution using normal equations. The dependence of the sampling size on ϵ makes this algorithm fairly impractical for typical direct solvers. However, subsequent work by Rokhlin and Tygert [35] applied a related sketching strategy to the *preconditioning* of a Krylov-subspace method, establishing a relationship between sample size and condition number.

Avron, Maymounkov, and Toledo [3] synthesized these concepts into a high-performance solver called Blendenpik. They first apply a randomized Hadamard transform (or similar transform), compute a QR decomposition of the result, and use its R-factor as a preconditioner for the standard LSQR solver. Additionally they show that the condition number of their system depends on the *maximal* leverage score of the matrix, referred to as *coherence*.

DEFINITION 2 (see [16, 9]). Coherence *is the maximum leverage score of* $\mathbf{A}$, *i.e.,*

$$\mu(\mathbf{A}) = \max_{i \in \{1,\ldots,n\}} l_i,$$

where l_i *is the leverage score of the* i*th row of* $\mathbf{A}$. *It holds that* $\frac{d}{n} \leq \mu(\mathbf{A}) \leq 1$.

Intuitively, if a row of a matrix $\mathbf{A}$ contains the only nonzero in a column, then $\mu(\mathbf{A}) = 1$ and any row-sampling $\mathbf{SA}$ must include that row (which has leverage score

1) or it will be rank deficient. If coherence is close to 1, a uniform row sampling is likely to be *nearly* rank-deficient, leading to a poorly conditioned reduced-size least squares problem and an inaccurate approximate solution vector.

In subsection 3.2 we prove that the standard formulation of CP-ALS may increase incoherence, making uniform sampling effective in many situations. However, in order to guarantee incoherence (w.h.p.) regardless of input, it is necessary to preprocess with a mixing step.

This mixing strategy relates to a more general class of transformations that rely on quality guarantees provided by the Johnson–Lindenstrauss lemma [22]. This lemma specifies a class of random projections that preserve the distances between all pairs of vectors with reasonable accuracy. The *fast* Johnson–Lindenstrauss transform (FJLT) is able to avoid explicit matrix multiplications by utilizing efficient algorithms such as the fast Fourier transform (FFT), discrete cosine transform (DCT), or Walsh–Hadamard transform (WHT) [2]. These transforms can operate on a vector $\mathbf{x} \in \mathbb{R}^n$ in $\mathcal{O}(n \log_2 n)$ time. What these algorithms have in common is that they improve incoherence, mixing information across every element of a vector, while at the same time being orthogonal operations (i.e., a change of basis). The theoretical quality guarantees and theoretical computational costs are the same for all fast transforms.

The FJLT consists of three steps. First, each row of the coefficient matrix is sign-flipped with probability 1/2. This is equivalent to computing $\mathbf{DA}$ with diagonal matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, where each diagonal element is ± 1 with equal probability. Second, we apply the fast mixing operation $\mathcal{F}$. Third, we uniformly sample S rows of the result with uniform probability. Thus, the entire operation can be written out as $\mathbf{S}\mathcal{F}\mathbf{DA}$, where $\mathbf{S}$ is a row-sampling operator (containing unit row vectors $\mathbf{e}_i$ for each sampled row i). The reasoning behind first applying $\mathbf{D}$ is that input data is often sparse in the frequency domain, and randomly flipping the signs of the coefficient matrix is an orthogonal operation that spreads out the frequency domain of the signal [2]. In this paper we will exclusively use the FFT for the $\mathcal{F}$ operation due to its ease of reproducibility and its efficiency in MATLAB. Though portable, this has the result of making all data complex-valued, which we discuss in more detail later on. We observed that using alternative transforms had no effect on the quality of our solutions, but real-valued transforms were slower to apply because of a lack of efficient implementations within MATLAB.

3. Algorithms. In this section we introduce CPRAND, which samples without mixing, and CPRAND-MIX, which efficiently applies mixing before sampling. We first recall the standard CP-ALS method, which is the starting point for our modifications.

3.1. CP-ALS. The standard method for fitting the CP model is alternating least squares (CP-ALS) [19, 23]. The method alternates among the modes, fixing every factor matrix but $\mathbf{A}^{(n)}$ and solving for it. From (7), we see that we can find $\mathbf{A}^{(n)}$ by solving the linear least squares problem given by

$$\arg\min_{\mathbf{A}^{(n)}} \|\mathbf{X}_{(n)} - \mathbf{A}^{(n)}\mathbf{Z}^{(n)\mathsf{T}}\|_F. \tag{10}$$

In CP-ALS, we work with the normal equations for (10),

$$\mathbf{X}_{(n)}\mathbf{Z}^{(n)} = \mathbf{A}^{(n)}(\mathbf{Z}^{(n)\mathsf{T}}\mathbf{Z}^{(n)}),$$

and solve for $\mathbf{A}^{(n)}$ for given $\mathbf{X}_{(n)}$ and $\mathbf{Z}^{(n)}$. By identity (2), we have

$$\mathbf{Z}^{(n)\mathsf{T}}\mathbf{Z}^{(n)} = \mathbf{A}^{(N)\mathsf{T}}\mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(n+1)\mathsf{T}}\mathbf{A}^{(n+1)} \circledast \mathbf{A}^{(n-1)\mathsf{T}}\mathbf{A}^{(n-1)} \circledast \cdots \circledast \mathbf{A}^{(1)\mathsf{T}}\mathbf{A}^{(1)}.$$

The CP-ALS algorithm [23] is presented in Algorithm 1. Note the step where vector $\boldsymbol{\lambda}$ stores normalization values of each column so that the final approximation is as in (8); this normalization helps alleviate issues due to scaling ambiguity.

The initialization of the factor matrices in line 2 can impact the performance of the algorithm. There are many possible ways to do the the initialization. One way to initialize is to set $\mathbf{A}^{(n)}$ to be the leading R left singular vectors of the mode-n unfolding, $\mathbf{X}_{(n)}$, and we call this HOSVD initialization, as it corresponds to the factor matrices in the rank-$(R\times\cdots\times R)$ higher-order SVD. A less expensive but less effective initialization is to choose random factor matrices.

Algorithm 1. CP-ALS.

1: **function** $[\boldsymbol{\lambda}, \{\,\mathbf{A}^{(n)}\,\}] = \text{CP-ALS}(\mathcal{X}, R)$ ▷ $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_N}$
2: Initialize factor matrices $\mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$
3: **repeat**
4: **for** $n = 1, \ldots, N$ **do**
5: $\mathbf{V} \leftarrow \mathbf{A}^{(N)\mathsf{T}}\mathbf{A}^{(N)} \circledast \cdots \circledast \mathbf{A}^{(n+1)\mathsf{T}}\mathbf{A}^{(n+1)} \circledast \mathbf{A}^{(n-1)\mathsf{T}}\mathbf{A}^{(n-1)} \circledast \cdots \circledast \mathbf{A}^{(1)\mathsf{T}}\mathbf{A}^{(1)}$
6: $\mathbf{Z}^{(n)} \leftarrow \mathbf{A}^{(N)} \odot \cdots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \cdots \odot \mathbf{A}^{(1)}$
7: $\mathbf{W} \leftarrow \mathbf{X}_{(n)}\mathbf{Z}^{(n)}$
8: Solve $\mathbf{A}^{(n)}\mathbf{V} = \mathbf{W}$ for $\mathbf{A}^{(n)}$
9: Normalize columns of $\mathbf{A}^{(n)}$ and update $\boldsymbol{\lambda}$
10: **end for**
11: **until** termination criteria met
12: **return** $\boldsymbol{\lambda}$, factor matrices $\{\,\mathbf{A}^{(n)}\,\}$
13: **end function**

3.1.1. Cost. We consider the cost of a single outer iteration of CP-ALS. In line 5, the cost of computing the mth Gram matrix is $R^2 I_m$; so the entire cost is $R^2 \sum_{m\neq n} I_m$ flops to compute all the Gram matrices plus $\mathcal{O}(R^2 N)$ to multiply them all together to form $\mathbf{V}$. The combination of lines 6 and 7 form an operation called the matricized tensor times Khatri–Rao product (MTTKRP). This is a frequent target of optimization [5, 32, 37, 13]. The Khatri–Rao product in line 6 requires $\mathcal{O}(R\prod_{m\neq n} I_m)$ flops (flops may be reduced at the cost of more memory by storing and reusing partial products, but we ignore this detail in our discussion). The computation of $\mathbf{X}_{(n)}\mathbf{Z}^{(n)}$ in line 7 is the most expensive step, with a cost of $2R\prod_m I_m$ flops. We note that Phan, Tichavský, and Cichocki [32] give a clever reorganization of lines 6 and 7, which avoids data movement and thus may reduce overall runtime. The cost of solving the linear system in line 8 using Cholesky decomposition is dominated by the triangular solves with the Cholesky factors, which requires $2R^2 I_n$ flops. The overall cost of each outer iteration (updating each factor matrix once) is $\mathcal{O}(NR\prod_n I_n)$.

If HOSVD initialization is used in line 2, then the costs of forming the mode-n Gram matrix is $I_n \prod_m I_m$ and the cost of computing the eigenvectors is I_n^3. Hence, the total initialization cost is $(\sum_{n>1} I_n)\prod_m I_m + \sum_{n>1} I_n^3$.

3.2. CPRAND. Consider the overdetermined least squares problem in (10), which is convenient to rewrite as

$$\arg\min_{\mathbf{A}^{(n)}} \|\mathbf{Z}^{(n)}\mathbf{A}^{(n)\mathsf{T}} - \mathbf{X}_{(n)}^{\mathsf{T}}\|_F. \tag{11}$$

The simplest sampling algorithm uniformly samples rows from $\mathbf{Z}_{(n)}$ and the corresponding rows from $\mathbf{X}_{(n)}^{\mathsf{T}}$. We let S denote the number of desired number of samples. Without loss of generality, we assume $S > \max\{\,I_1, \ldots, I_N, R\,\}$. Let $\mathcal{S}$ denote the samples from $\{\,1, \ldots, \prod_{m\neq n} I_m\,\}$ such that $|\mathcal{S}| = S$. We do *uniform sampling with*

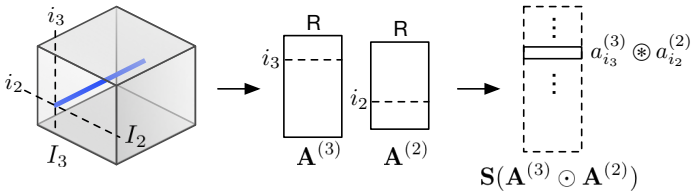


FIG. 2. *Sampled Khatri–Rao rows correspond to sampled fibers in* $\mathcal{X}$.

replacement which means that every row has an equal chance of being selected and the same row may be selected more than once. We can express the sampling operation as the application of a selection matrix $\mathbf{S} \in \mathbb{R}^{S \times \prod_{m \neq n} I_m}$, where the rows of $\mathbf{S}$ are rows of the $\prod_{m \neq n} I_m \times \prod_{m \neq n} I_m$ identity matrix.

Forming the (full) Khatri–Rao product $\mathbf{Z}^{(n)}$ is expensive, so we want to compute the sampled matrix $\mathbf{S}\mathbf{Z}^{(n)}$ without ever explicitly forming $\mathbf{Z}^{(n)}$. Consider sampling the jth row of $\mathbf{Z}_{(n)}$. Using the mapping in (1), we can map j to indices $(i_1, \ldots, i_{n-1}, i_{n+1}, \ldots, i_N)$ (the nth index is omitted). In fact, the jth row of $\mathbf{Z}^{(n)}$ is the Hadamard product of the appropriate rows of the factor matrices, i.e.,

$$\mathbf{Z}^{(n)}(j,:) = \mathbf{A}^{(1)}(i_1,:) \circledast \cdots \circledast \mathbf{A}^{(n-1)}(i_{n-1},:) \circledast \mathbf{A}^{(n+1)}(i_{n+1},:) \circledast \cdots \circledast \mathbf{A}^{(N)}(i_N,:).$$

This is illustrated in Figure 2 for a three-way tensor. We give the algorithm for computing *sampled Khatri–Rao* (SKR) in Algorithm 2, where `idxs` is the set of tuples

$$\{i_1^{(j)}, \ldots, i_{n-1}^{(j)}, i_{n+1}^{(j)}, \ldots, i_N^{(j)}\} \quad \text{for} \quad j \in \mathcal{S}.$$

We assume these tuples are stacked in matrix form for efficiency. Thus, each multiplicand $\mathbf{A}_S^{(m)}$ is of size $S \times R$.

Algorithm 2. SKR product.

1: **function** $\mathbf{Z}_S = \text{SKR}(\mathbf{S}, \mathbf{A}^{(N)}, \ldots, \mathbf{A}^{(n+1)}, \mathbf{A}^{(n-1)}, \ldots, \mathbf{A}^{(1)})$
2: Retrieve `idxs` from $\mathbf{S}$
3: $\mathbf{Z}_S \leftarrow \mathbf{1}$ ▷ $\mathbf{1} \in \mathbb{R}^{S \times R}$
4: **for** $m = 1, \ldots, n-1, n+1, \ldots, N$ **do**
5: $\mathbf{A}_S^{(m)} \leftarrow \mathbf{A}^{(m)}(\texttt{idxs}(:,m),:)$ ▷ MATLAB-style indexing
6: $\mathbf{Z}_S \leftarrow \mathbf{Z}_S \circledast \mathbf{A}_S^{(m)}$
7: **end for**
8: **return** $\mathbf{Z}_S$
9: **end function**

In the same way we wanted to avoid forming $\mathbf{Z}^{(n)}$ explicitly, we also want to avoid forming $\mathbf{X}_{(n)}$ (i.e., in this case that means avoiding the data movement). Instead, we observe that if we sample the jth row of $\mathbf{X}_{(n)}$, then we want the fiber $\mathbf{x}_{i_1,\ldots,i_{n-1},:,i_{n+1},\ldots,i_N}$ where we are using the same mapping as for SKR. Thus, we can avoid matricization and pull entries from the tensor directly to form $\mathbf{S}\mathbf{X}_{(n)}^\mathsf{T}$.

Our randomized version of CP-ALS is called CPRAND and shown in Algorithm 3, where we solve a sampled version of the least squares problem in (11). In line 6, we use SKR from Algorithm 2 to get the sampled version of $\mathbf{Z}^{(n)}$. In line 7, we sample rows of $\mathbf{X}_{(n)}^\mathsf{T}$. In line 8, we solve the sampled least squares problem where the coefficient matrix $\mathbf{Z}_S$ is of size $S \times R$ and the corresponding sampled right-hand side $\mathbf{X}_S^\mathsf{T}$ is of size $S \times I_n$. The solution $\mathbf{A}^{(n)}$ is of size $I_n \times R$.

In all of the experiments in this paper, a sample size of $S = 10R\log(R)$ has proven sufficient, provided that the data is incoherent.

Algorithm 3. CPRAND.

1: **function** CPRAND($\mathcal{X}, R, S$) ▷ $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
2: Initialize factor matrices $\mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$
3: **repeat**
4: **for** $n = 1, \ldots, N$ **do**
5: Define sampling operator $\mathbf{S} \in \mathbb{R}^{S \times \prod_{m \neq n} I_m}$
6: $\mathbf{Z}_S \leftarrow \text{SKR}(\mathbf{S}, \mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(n-1)}, \mathbf{A}^{(n+1)}, \ldots, \mathbf{A}^{(N)})$
7: $\mathbf{X}_S^\mathsf{T} \leftarrow \mathbf{S}\mathbf{X}_{(n)}^\mathsf{T}$
8: $\mathbf{A}^{(n)} \leftarrow \arg\min_{\mathbf{A}} \left\| \mathbf{Z}_S \mathbf{A}^\mathsf{T} - \mathbf{X}_S^\mathsf{T} \right\|_F$
9: Normalize columns of $\mathbf{A}^{(n)}$ and update $\boldsymbol{\lambda}$
10: **end for**
11: **until** termination criteria met
12: **return** $\boldsymbol{\lambda}$, factor matrices $\{ \mathbf{A}^{(n)} \}$
13: **end function**

3.2.1. Cost. The arithmetic cost of Algorithm 3 comprises the cost of sampling, to set up the smaller least squares problem, and the cost of solving the least squares problem. Generating S random multiindices requires $O(SN)$ operations. Sampling the Khatri–Rao product in line 6 using Algorithm 2 requires $SR(N-1)$ flops, as each of the S length-R rows of $\mathbf{Z}_S^{(n)}$ is formed as a Hadamard product of $N-1$ rows of the fixed factor matrices. Sampling S fibers from $\mathcal{X}$ to form $\mathbf{X}_S^{(n)}$ requires no flops, but it does require irregular data access to construct the $I_n \times S$ matrix and is actually the most time-consuming operation for large tensors. Using QR to solve the reduced least squares problem in line 8 is $2SR^2$ flops, the cost of computing $\mathbf{X}_S^{(n)}\mathbf{Q}$ (applying $\mathbf{Q}$) is $2SRI_n$ operations, and the cost of the triangular solve is $R^2 I_n$ operations. Assuming $I_n > R$ and $S > R$, the leading-order cost is $2SRI_n$ operations. The overall cost of each outer iteration (updating each mode once) is $\mathcal{O}(SR\sum_n I_n)$.

3.2.2. Coherence in CPRAND. The effectiveness of CPRAND depends on the coherence of coefficient matrix $\mathbf{Z}^{(n)}$. Since $\mathbf{Z}^{(n)}$ is formed as the Khatri–Rao product of factor matrices, it is natural to ask what effect the Khatri–Rao product has on coherence.

More rigorously we now show that there is, at the very least, a multiplicative reduction in coherence when we form $\mathbf{Z}^{(n)}$. We begin by bounding the coherence of the Kronecker product and use this to bound the coherence of the Khatri–Rao product.

LEMMA 3. *Given* $\mathbf{A} \in \mathbb{R}^{I \times J}$ *and* $\mathbf{B} \in \mathbb{R}^{K \times L}$, $\mu(\mathbf{A} \otimes \mathbf{B}) = \mu(\mathbf{A})\mu(\mathbf{B})$.

Proof. We take the reduced QR factorizations of the two terms and then apply (3):

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{Q}_A\mathbf{R}_A \otimes \mathbf{Q}_B\mathbf{R}_B = (\mathbf{Q}_A \otimes \mathbf{Q}_B)(\mathbf{R}_A \otimes \mathbf{R}_B).$$

This is a reduced QR factorization of $\mathbf{A} \otimes \mathbf{B}$, and the $\mathbf{Q}$ factor has rows $\mathbf{Q}_A(i,:) \otimes \mathbf{Q}_B(j,:)$ for every possible pair (i,j). We know through simple arithmetic that $\|\mathbf{a} \otimes \mathbf{b}\| = \|\mathbf{a}\|\|\mathbf{b}\|$, so the max row norm of $\mathbf{Q}_A \otimes \mathbf{Q}_B$ is the product of the max row norms of $\mathbf{Q}_A$ and $\mathbf{Q}_B$. ∎

LEMMA 4. *Given* $\mathbf{A} \in \mathbb{R}^{I \times J}$ *and* $\mathbf{B} \in \mathbb{R}^{K \times L}$, $\mu(\mathbf{A} \odot \mathbf{B}) \leq \mu(\mathbf{A})\mu(\mathbf{B})$.

Proof. We again take the reduced QR factorizations of the two terms and then apply (4):

$$\mathbf{A} \odot \mathbf{B} = \mathbf{Q}_A\mathbf{R}_A \odot \mathbf{Q}_B\mathbf{R}_B = (\mathbf{Q}_A \otimes \mathbf{Q}_B)(\mathbf{R}_A \odot \mathbf{R}_B).$$

We then take the QR decomposition of the second term:

$$(\mathbf{Q}_A \otimes \mathbf{Q}_B)(\mathbf{R}_A \odot \mathbf{R}_B) = (\mathbf{Q}_A \otimes \mathbf{Q}_B)\mathbf{Q}_R\mathbf{R}_R = \hat{\mathbf{Q}}\mathbf{R}_R.$$

The reduced $\mathbf{Q}$ term for $\mathbf{A} \odot \mathbf{B}$ is $\hat{\mathbf{Q}} = (\mathbf{Q}_A \otimes \mathbf{Q}_B)\mathbf{Q}_R$, so the norm of row i in $\hat{\mathbf{Q}}$ is $\hat{l}_i$, the ith leverage score of $\mathbf{A} \odot \mathbf{B}$. Letting $\hat{\mathbf{q}}^\mathsf{T}$ be row i of $\mathbf{Q}_A \otimes \mathbf{Q}_B$,

$$\hat{l}_i = \|\hat{\mathbf{q}}_i^\mathsf{T}\mathbf{Q}_R\| = \|\mathbf{Q}_R^\mathsf{T}\hat{\mathbf{q}}_i\| \le \|\mathbf{Q}_R^\mathsf{T}\|\|\hat{\mathbf{q}}_i\|.$$

$\mathbf{Q}_R^\mathsf{T}$ has orthonormal rows, so $\|\mathbf{Q}_R^\mathsf{T}\| = 1$, yielding

$$\mu(\mathbf{A} \odot \mathbf{B}) = \mu(\hat{\mathbf{Q}}) = \max_i \hat{l}_i \ \le \ \max_i \|\hat{\mathbf{q}}_i\| = \mu(\mathbf{Q}_A \otimes \mathbf{Q}_B) = \mu(\mathbf{A})\mu(\mathbf{B}). \qquad \square$$

Using the notation of the prior proof, the exact coherence expands to

$$\mu(\mathbf{A} \odot \mathbf{B}) = \max_i \sqrt{\hat{\mathbf{q}}_i^\mathsf{T}\mathbf{Q}_R\mathbf{Q}_R^\mathsf{T}\hat{\mathbf{q}}_i},$$

where the amount of truncation in $\mathbf{Q}_R$ corresponds to how loose the inequality is. This bound is tight, e.g., for $\mathbf{A} = (1, 1)^\mathsf{T}$ and $\mathbf{B} = (1, -1)^\mathsf{T}$, but we typically see a large factor of coherence reduction.

3.3. CPRAND-MIX. Lemma 4 shows that the Khatri–Rao product inherits incoherence from its factors. However, if the individual factor matrices happen to be highly coherent, CPRAND may fail to converge. We can prevent this by *mixing* the terms before sampling occurs, as in Algorithm 5. Consider the least squares problem in CP-ALS in (11). Recall the FJLT introduced in subsection 2.2. We *could* apply such a transformation directly to the inner iteration before sampling, yielding

$$\arg\min_{\mathbf{A}^{(n)}} \|\mathcal{F}\mathbf{D}\mathbf{Z}^{(n)}\mathbf{A}^{(n)\mathsf{T}} - \mathcal{F}\mathbf{D}\mathbf{X}_{(n)}\|_F,$$

where $\mathbf{D}$ is a diagonal random sign matrix and $\mathcal{F}$ is the FFT matrix. However, this would involve mixing the matricization of $\mathcal{X}$ in each mode, as well as forming and mixing the full Khatri–Rao product at each iteration. Instead of mixing the rows of the entire Khatri–Rao product $\mathbf{Z}^{(n)}$, we mix the rows of each factor $\mathbf{A}^{(m)}$ individually. Using the distributive property in (4), we see that this is equivalent to applying a Kronecker product of mixing terms:

$$\hat{\mathbf{Z}}^{(n)} = \bigodot_{\substack{m=N\\m\neq n}}^{1} \mathcal{F}_m\mathbf{D}_m\mathbf{A}^{(m)} = \left(\bigotimes_{\substack{m=N\\m\neq n}}^{1} \mathcal{F}_m\mathbf{D}_m\right)\mathbf{Z}^{(n)} = \left(\bigotimes_{\substack{m=N\\m\neq n}}^{1} \mathcal{F}_m\right)\left(\bigotimes_{\substack{m=N\\m\neq n}}^{1} \mathbf{D}_m\right)\mathbf{Z}^{(n)}.$$

Applying this operation to the CP least squares problem leads to the mixed formulation of the least squares problem:

$$\arg\min_{\mathbf{A}^{(n)}} \left\| \left(\bigotimes_{\substack{m=N\\m\neq n}}^{1} \mathcal{F}_m\mathbf{D}_m\right)\mathbf{Z}^{(n)}\mathbf{A}^{(n)\mathsf{T}} - \left(\bigotimes_{\substack{m=N\\m\neq n}}^{1} \mathcal{F}_m\mathbf{D}_m\right)\mathbf{X}_{(n)}^\mathsf{T} \right\|_F. \tag{12}$$

The Kronecker product preserves orthogonality (unitarity), so (12) is equivalent to (11).

Note that while we do not prove that applying uniform sampling to the columns of (12) is an FJLT (we conjecture it is), we know that $\mu(\hat{\mathbf{Z}}^{(n)})$ is upper bounded by Lemma 4, so we expect uniform sampling to be sufficient for an accurate approximate solution. Using (5), we can see that the second term of (12) is equivalent to

$$\left(\mathbf{D}_n \mathcal{F}_n^* \hat{\mathbf{X}}_{(n)}\right)^{\mathsf{T}}, \quad \text{where} \quad \hat{\mathcal{X}} = \mathcal{X} \times_1 \mathcal{F}_1 \mathbf{D}_1 \cdots \times_N \mathcal{F}_N \mathbf{D}_N, \tag{13}$$

where we note that $\mathbf{D}_n$ is its own inverse, $\mathcal{F}_n$ is unitary in the case of the FFT, and the asterisk denotes the conjugate transpose. Using a uniform sampling matrix $\mathbf{S}$, our reduced problem has the form

$$\underset{\mathbf{A}^{(n)}}{\arg\min} \left\| \left(\mathbf{S}\hat{\mathbf{Z}}^{(n)}\right) \mathbf{A}^{(n)\mathsf{T}} - \mathbf{D}_n \mathcal{F}_n^* \left(\mathbf{S}\hat{\mathbf{X}}_{(n)}^{\mathsf{T}}\right)^{\mathsf{T}} \right\|_F. \tag{14}$$

This approach is presented as CPRAND-MIX in Algorithm 4. We highlight two computational optimizations in Algorithm 4. First, by (13), we can apply a single upfront mixing of the tensor in all modes in a preprocessing step (line 5). Then, at each inner iteration for mode n, we *unmix* the tensor in only mode n. Furthermore, this unmixing can be done *after* the columns of $\hat{\mathbf{X}}_{(n)}$ are sampled, as shown in line 10. Second, we can avoid mixing the entire Khatri–Rao product matrix at each step because only one factor matrix changes each iteration. Thus, we mix the nth factor matrix to produce $\hat{\mathbf{A}}^{(n)}$ in line 13 only once per outer iteration, immediately after computing $\mathbf{A}^{(n)}$. Then we can sample the mixed Khatri–Rao product without forming it explicitly using Algorithm 2.

Algorithm 4. CPRAND-MIX.

1: **procedure** CPRAND-MIX($\mathcal{X}, R, S$) ▷ $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
2: Initialize factor matrices $\mathbf{A}^{(m)}$, $m \in \{2 \dots N\}$
3: Define random sign-flip operators $\mathbf{D}_m$ and unitary matrices $\mathcal{F}_m$, $m \in \{1, \dots, N\}$
4: Mix factor matrices: $\hat{\mathbf{A}}^{(m)} \leftarrow \mathcal{F}_m \mathbf{D}_m \mathbf{A}^{(m)}$, $m \in \{2 \dots N\}$
5: Mix tensor: $\hat{\mathcal{X}} \leftarrow \mathcal{X} \times_1 \mathcal{F}_1 \mathbf{D}_1 \cdots \times_N \mathcal{F}_N \mathbf{D}_N$
6: **repeat**
7: **for** $n = 1, \dots, N$ **do**
8: Define sampling operator $\mathbf{S} \in \mathbb{R}^{S \times \prod_{m \neq n} I_m}$
9: $\hat{\mathbf{Z}}_S \leftarrow SKR(\mathbf{S}, \hat{\mathbf{A}}^{(N)}, \dots, \hat{\mathbf{A}}^{(n+1)}, \hat{\mathbf{A}}^{(n-1)}, \dots, \hat{\mathbf{A}}^{(1)})$
10: $\hat{\mathbf{X}}_S^{\mathsf{T}} \leftarrow \mathbf{D}_n \mathcal{F}_n^* \left(\mathbf{S}\hat{\mathbf{X}}_{(n)}^{\mathsf{T}}\right)^{\mathsf{T}}$
11: $\mathbf{A}^{(n)} \leftarrow \underset{\mathbf{A}}{\arg\min} \left\| \hat{\mathbf{Z}}_S \mathbf{A}^{\mathsf{T}} - \hat{\mathbf{X}}_S^{\mathsf{T}} \right\|_F$ subject to $\mathbf{A}$ being real-valued
12: Normalize columns of $\mathbf{A}^{(n)}$ and update $\boldsymbol{\lambda}$
13: $\hat{\mathbf{A}}^{(n)} \leftarrow \mathcal{F}_n \mathbf{D}_n \mathbf{A}^{(n)}$
14: **end for**
15: **until** termination criteria met
16: **return** $\boldsymbol{\lambda}$, factor matrices $\{\mathbf{A}^{(n)}\}$
17: **end procedure**

Next, we point out a subtlety within Algorithm 4 due to our use of the FFT, which is complex-valued. Assuming the input tensor is real-valued, we seek a CP decomposition that is also real-valued. However, the least squares problem in line 11 involves complex-valued matrices, and the solution can also be complex-valued. We note that if the least squares problem is mixed but not sampled, and if the Khatri–Rao

product is full rank, then the solution would still be real-valued. However, because sampling implies that we are solving the original least squares problem approximately, the solution can drift into the complex plane. In order to maintain a real-valued CP approximation, we solve the least squares problem over only real values by using the equivalence

$$\arg\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2 = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \left\| \begin{bmatrix} \Re(\mathbf{A}) \\ \Im(\mathbf{A}) \end{bmatrix} \mathbf{x} - \begin{bmatrix} \Re(\mathbf{b}) \\ \Im(\mathbf{b}) \end{bmatrix} \right\|_2, \tag{15}$$

where $\mathbf{A}\in\mathbb{C}^{m\times n}$ and $\mathbf{b}\in\mathbb{C}^m$. When a real-valued orthogonal matrix is used instead of the FFT (such as the DCT or the WHT), the aforementioned subtlety can be ignored. In fact, the algorithm can be further simplified. We note that if we write the new ALS update in terms of the pseudoinverse, we get the following:

$$\mathbf{A}^{(n)} \leftarrow \mathbf{D}_n \mathcal{F}_n^* (\hat{\mathbf{X}}_{(n)} \mathbf{S}^T)[(\mathbf{S}\hat{\mathbf{Z}}^{(n)})^{\mathsf{T}}]^{\dagger}.$$

This update implies that we store the unmixed factor matrices, mix them before each iteration, and then unmix afterwards. We can actually avoid this process by maintaining the factor matrices in their mixed state for the duration of the algorithm:

$$\hat{\mathbf{A}}^{(n)} \leftarrow \mathcal{F}_n \mathbf{D}_n \mathbf{A}^{(n)} = \hat{\mathbf{X}}_{(n)} \mathbf{S}^T [(\mathbf{S}\hat{\mathbf{Z}}^{(n)})^{\mathsf{T}}]^{\dagger}.$$

This transformation yields the following least squares problem:

$$\arg\min_{\hat{\mathbf{A}}^{(n)}} \left\| \hat{\mathbf{A}}^{(n)} (\mathbf{S}\hat{\mathbf{Z}}^{(n)})^{\mathsf{T}} - \hat{\mathbf{X}}_{(n)} \mathbf{S}^{\mathsf{T}} \right\|_F,$$

which is equivalent to line 8 in Algorithm 3, just in the mixed basis. Thus, in the case of real-valued transforms, it is sufficient to mix $\mathcal{X}$, call CPRAND as a subroutine, and then unmix the solution factors, as shown in Algorithm 5.

Algorithm 5. CPRAND-PREMIX.

1: **function** CPRAND-PREMIX($\mathcal{X}, R, S$) ▷ $\mathcal{X}\in\mathbb{R}^{I_1\times\cdots\times I_N}$
2: Define random sign-flip operators $\mathbf{D}_m$ and orthogonal matrices $\mathcal{F}_m$, $m\in\{1,\ldots,N\}$
3: Mix: $\hat{\mathcal{X}} \leftarrow \mathcal{X} \times_1 \mathcal{F}_1\mathbf{D}_1 \times \cdots \times_N \mathcal{F}_N\mathbf{D}_N$
4: $[\boldsymbol{\lambda}, \{\hat{\mathbf{A}}^{(n)}\}] =$ CPRAND($\hat{\mathcal{X}}, R, S$)
5: **for** $n = 1,\ldots,N$ **do**
6: Unmix: $\mathbf{A}^{(n)} = \mathbf{D}_n \mathcal{F}_n^{\mathsf{T}} \hat{\mathbf{A}}^{(n)}$
7: **end for**
8: **return** λ, factor matrices $\{\mathbf{A}^{(n)}\}$
9: **end function**

3.3.1. Cost. In the case of real-valued orthogonal transformations, as we show in Algorithm 5, we can implement CPRAND-MIX using CPRAND along with preprocessing (mixing) and postprocessing (unmixing) steps. The initial mixing of the tensor in line 3 requires significant upfront cost, but the unmixing of the factor matrices in line 6 is relatively cheap. Compared to Algorithm 3, the dominant extra cost (of line 3) is

$$\mathcal{O}\left(\sum_{k=1}^{N}\prod_{m}[I_m \log I_k]\right) = \mathcal{O}\left(\left(\prod_m I_m\right)\log\left(\prod_m I_m\right)\right). \tag{16}$$

The cost of Algorithm 4 includes the mixing cost given by (16), and the cost per iteration is slightly larger than in Algorithm 3. In particular, due to the complex

values and complex arithmetic, the cost of each operation is increased by a constant factor between 2 and 4. The leading-order cost of Algorithm 3 comes from solving the least squares problem in line 8, and the leading-order cost of Algorithm 4 also comes from solving the least squares problem (line 11). Following (15), the least squares problem is solved in real arithmetic, but the number of rows of the coefficient matrix is twice as many as in Algorithm 3. This yields an increase in the leading-order per-iteration cost of a factor of 2 compared to CPRAND.

3.4. Stopping criteria. Given the original tensor $\mathcal{X}$ and CP approximation $\tilde{\mathcal{X}}$ from (6), the relative residual norm is $\|\mathcal{X} - \tilde{\mathcal{X}}\|/\|\mathcal{X}\|$. However, the sampled least squares computations are so inexpensive that checking this stopping condition can take longer than the rest of the iteration. This is particularly true for out-of-core problem sizes [42]. Thus, we propose a different sampling-based method for computing an approximate stopping criterion and present a theoretical rationale for why it works. One tempting strategy would be to track the norm of the residual within the sampled least squares computation. Unfortunately, the variance of this value is very high due to the small number of fibers sampled from $\mathcal{X}$. Thus, we propose an alternative approach.

We use the notation $[N]$ to denote the set $\{1, \ldots, N\}$. For a given natural number $\hat{P}$, let

$$\hat{\mathcal{I}} \subset \mathcal{I} \equiv [I_1] \otimes [I_2] \otimes \cdots \otimes [I_N]$$

be a uniform random subset of $\hat{P}$ indices of $\mathcal{X}$. Let $\mathbf{i} = (i_1, i_2, \ldots, i_N)$ denote a multiindex, i.e., $x_{\mathbf{i}} = x_{i_1 i_2 \cdots i_N}$. Define $\mathcal{E} = \mathcal{X} - \tilde{\mathcal{X}}$, and observe that

$$\|\mathcal{E}\|^2 = \sum_{\mathbf{i} \in \mathcal{I}} e_{\mathbf{i}}^2 = P\mu, \quad \text{where} \quad P = \prod_n I_n \quad \text{and} \quad \mu = \text{mean}\{\, e_{\mathbf{i}}^2 \mid \mathbf{i} \in \mathcal{I} \,\}.$$

We can approximate the mean μ with the mean $\hat{\mu}$ of the subset of entries in $\hat{\mathcal{I}}$:

$$\mu \approx \hat{\mu}, \quad \text{where} \quad \hat{\mu} = \text{mean}\{\, e_{\mathbf{i}}^2 \mid \mathbf{i} \in \hat{\mathcal{I}} \,\}.$$

The relative residual norm can be estimated as

$$\frac{\|\mathcal{E}\|}{\|\mathcal{X}\|} = \frac{(P\mu)^{1/2}}{\|\mathcal{X}\|} \approx \frac{(P\hat{\mu})^{1/2}}{\|\mathcal{X}\|}.$$

We can now apply the multiplicative Chernoff–Hoeffding bounds if we make some assumptions on our data [26]. Assume the errors are drawn from a finite distribution and μ is the *true mean* (or close enough to it). This is a reasonable assumption if we assume that the CP models elicits the low-rank structure which is contaminated by noise. We do not make assumptions about what the specific distribution is. Our samples are assumed to be i.i.d. Let $\mu_{\max} = \max_{\mathbf{i}}(e_{\mathbf{i}}^2)$ be the maximum allowable value. For any $\gamma > 0$, we have the following very conservative upper- and lower-tail bounds:

$$\begin{aligned} \Pr\{\hat{\mu} \geq (1+\gamma)\mu\} &\leq \exp\left(-\frac{2\gamma^2\mu^2\hat{P}}{\mu_{\max}^2}\right), \\ \Pr\{\hat{\mu} \leq (1-\gamma)\mu\} &\leq \exp\left(-\frac{\gamma^2\mu^2\hat{P}}{\mu_{\max}^2}\right). \end{aligned} \tag{17}$$

We can then write this in terms of the residual norm.

LEMMA 5. *For any $\gamma \in (0,1)$, we can bound the relative difference in the approximated and true error as*

$$\Pr\left\{\sqrt{1-\gamma} \leq \frac{(P\hat{\mu})^{1/2}}{\|\mathcal{E}\|} \leq \sqrt{1+\gamma}\right\} \leq \exp\left(-2\frac{\gamma^2\mu^2\hat{P}}{\mu_{max}^2}\right).$$

Proof. Multiply both sides within the probability terms of (17) by P, take the square root, and simplify. ∎

Let $0 \leq \mu \leq 0.5$ (if the error is higher than this, we are far from terminating). Let $\sqrt{1+\gamma} = 1.05$; that is, we allow our estimate to be wrong by 5% multiplicatively. Then $\gamma = 0.1025$. A confidence of 98% is maintained when $\hat{P} \geq 372\mu_{\max}^2$. The fortunate aspect of this conservative bound is that we expect $\mu_{\max}$ to become smaller and smaller as the ALS algorithm proceeds. In general, we usually assume $\mu_{\max} = 1$.

The cost of computing $\hat{P}\hat{\mu}$ to get an estimate of the error is $\mathcal{O}(\hat{P}RN)$ flops. For each sampled entry of the tensor, the corresponding entry of the model tensor must be computed via a sum of R terms, each with $N+1$ multiplicands (including the weights). For comparison, the cost of computing the exact error is $\mathcal{O}(R\prod_n I_n)$ flops.

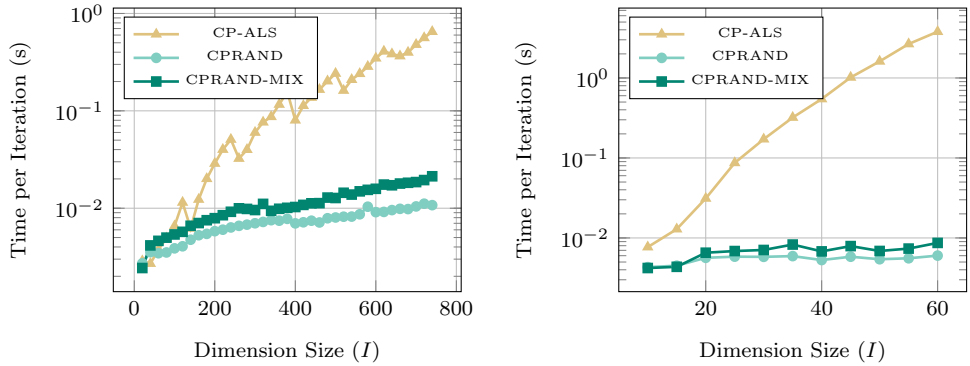
The relative residual of CP-ALS is guaranteed to decrease at each iteration, making a termination condition easy to specify (stop when the change in error drops below a threshold). Termination of CPRAND is more complicated because neither the true nor the approximate error is guaranteed to decrease at each iteration. In practice, the simplest strategy is to store the lowest relative error achieved so far and terminate when a particular number of iterations have elapsed without any reduction in this minimum.

4. Experiments. We evaluate the performance of the randomized algorithms on both synthetic and real-world data. The synthetic experiments enable us to generate tensors from known latent factors and thus measure whether the ground truth is recovered. We also consider real data sets which are free of the simplifying assumptions of synthetic data (e.g., Gaussian noise) and demonstrate our algorithms' effectiveness in practice.

All experiments are run on MATLAB R2016a using Tensor Toolbox v2.6 [4, 5, 6] on an Intel Xeon E5-2650 Ivy Bridge 2.0 GHz machine with 32 GB of memory. The CP-ALS implementation we compare against incorporates the recent optimizations of Phan, Tichavský, and Cichocki [32], without which it would be several times slower.

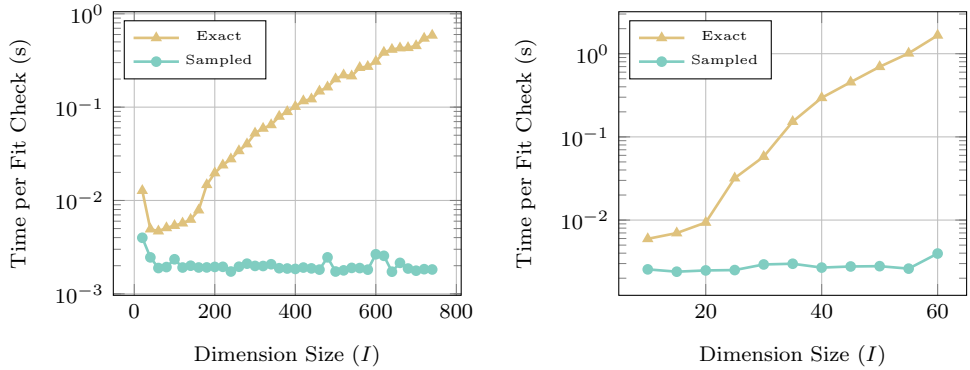
4.1. Computational time. Our first experiments ignore the convergence of the randomized methods and compare only the computational time for each iteration. Although the randomized methods will typically require more iterations, these experiments enable us to understand the difference in costs for the least squares solves and the initialization. We consider convergence and solution quality in subsequent subsections.

Figure 3 shows how much cheaper each iteration of the ALS algorithm is when using the randomized least squares solvers. We consider third- and fifth-order tensors of various sizes, where the dimensions of all modes are the same. We used a target rank $R = 5$ in all experiments. We sampled $S = 90$ rows for the randomized methods, although the exact number of rows makes little difference in runtime. For each size, we compute the mean time for 100 iterations over three tensors. The convergence checks are entirely omitted in the computation and do not contribute to the timings. Since we do a fixed number of iterations, the initial guesses are irrelevant. We see that as the size increases, the relative speedup of the randomized algorithms also increases to as



(a) Order 3: $I \times I \times I$ (b) Order 5: $I \times I \times I \times I \times I$

FIG. 3. *Mean time per iteration of CP-ALS, CPRAND, and CPRAND-MIX for third- and fifth-order tensors. The target rank is* $R = 5$. *The randomized methods use* $S = 90$ *sampled rows. Each dot represents the mean iteration time for three different tensors over* 100 *iterations (no checks for convergence).*



(a) Order 3: $I \times I \times I$ (b) Order 5: $I \times I \times I \times I \times I$

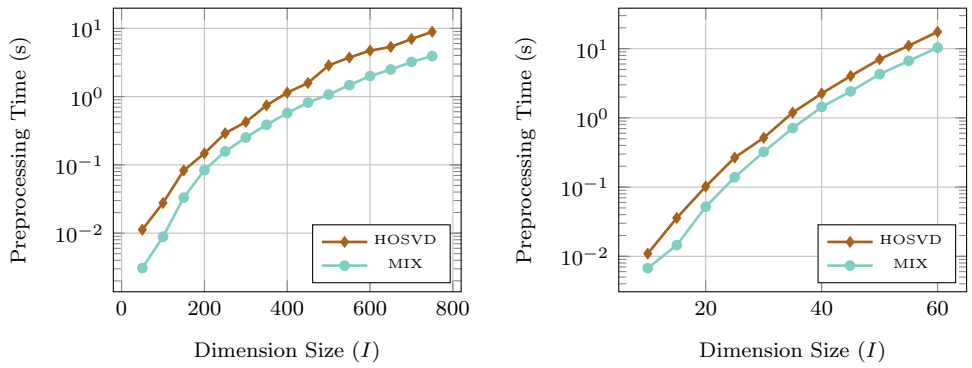
FIG. 4. *Termination criterion (fit check) time of exact (for CP-ALS) and sampled (for CPRAND and CPRAND-MIX) for third- and fifth-order tensors. The target rank is* $R = 5$. *The estimate of the fit is computed using* $\hat{P} = 2^{14}$ *sampled entries. Each dot is the mean time to compute fit over* 10 *trials.*

much as 50× for order-3 tensors over 500× for order-5 tensors. This is mainly due to the per-iteration computational cost of the randomized algorithms being $\mathcal{O}(NRIS)$, where N is the order of the tensor and I is the size of each dimension, as derived in subsection 3.2.1. For comparison, the cost of CP-ALS is $\mathcal{O}(NRI^N)$ flops per iteration.

Figure 4 demonstrates how much faster it is to compute the stopping criterion based on the sampling approach described in subsection 3.4. In this experiment, we compute the model fit both exactly and using $\hat{P} = 2^{14}$ samples for order-3 and order-5 tensors. The tensors and models are generated synthetically, with a prescribed fit of 95%. For these problems, the largest relative error between the true and sampled fits less than 10^{-3}, which is more than enough accuracy to make the correct decision on when to stop the iteration. For both order-3 and order-5 tensors, we see speedups of over two orders of magnitude for the largest problems, and we can expect larger

speedups for larger problems because the number of flops required by the sampling method is independent of the tensor dimensions.

We also consider the initialization costs of the methods. For CPRAND-MIX, we need to apply an FFT to the fibers of the tensor in each mode. If we use the HOSVD initialization for CP-ALS, then we need to consider its cost. We note that we do not actually compute the full HOSVD but rather the leading left singular vectors of the unfolded tensor $\mathbf{X}_{(n)}$ for $n = 2, \ldots, N$. Figure 5 shows the preprocessing time for third- and fifth-order tensors of various sizes, again where all modes have the same dimension. The target rank (needed for HOSVD) is $R = 5$. Each data point is the mean time over three trials of 100 iterations each for the given size. The cost of preprocessing for CPRAND-MIX is $\mathcal{O}(NI^N \log I)$, as derived in subsection 3.3.1, while the cost of HOSVD to initialize CP-ALS(H) is $\mathcal{O}(NI^{N+1})$ (we use an iterative eigenvector solver on the Gram matrix of each mode). While the cost of HOSVD is generally larger than mixing the tensor with FFTs, the data access patterns of the methods are similar, and we observe very similar timing results. We mention that CP-ALS requires a good starting point, like HOSVD, to ensure good performance. However, the randomized methods gain no advantage from using the HOSVD initial guess, so they use a random initialization.



(a) Order 3: $I \times I \times I$ (b) Order 5: $I \times I \times I \times I \times I$

FIG. 5. *Initialization time comparison between MIX (for CPRAND) and HOSVD (for CP-ALS) for third- and fifth-order tensors. The target rank is $R = 5$ for HOSVD. Each dot represents the mean of three trials.*

4.2. Synthetic data. For our experiments on synthetic tensors, we use various generation parameters. We create tensors based on known randomly generated weight vectors ($\boldsymbol{\lambda}$) and factor matrices ($\{\mathbf{A}^{(n)}\}$). In this way, we know the true solution. We consider third- and fourth-order problems, i.e., $N \in \{3, 4\}$. In the third-order case, we set the size to be $400 \times 400 \times 400$, and in the fourth-order case we set the size to be $90 \times 90 \times 90 \times 90$. For all experiments, we set the rank to be $R_{\text{true}} = 5$. The weight vector $\boldsymbol{\lambda} \in \mathbb{R}^{R_{\text{true}}}$ has entries drawn uniformly from $[0.2, 0.8]$. The factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_{\text{true}}}$ are randomly generated as described in [39] so that the columns have collinearity C, which means that any two column vectors from the same factor matrix satisfy

$$C = \frac{\mathbf{a}_r^{(n)\mathsf{T}} \mathbf{a}_s^{(n)}}{\|\mathbf{a}_r^{(n)\mathsf{T}}\| \|\mathbf{a}_s^{(n)}\|}.$$

Intuitively, higher collinearity corresponds to greater overlap between factors, while geometrically it corresponds to smaller angles between factor vectors. High collinearity makes the original factors harder to recover using CP-ALS and can introduce swamping behavior [33]. In our experiments we generate tensors with $C \in \{ 0.5, 0.9 \}$. Using the synthetic weight vectors ($\boldsymbol{\lambda}$) and factor matrices ($\{ \mathbf{A}^{(n)} \}$), the tensor we are trying to recover is

$$\mathcal{X}_{\text{true}} = \sum_{r=1}^{R_{\text{true}}} \lambda_r \, \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}.$$

Finally, we add noise to obtain the observed tensor. Let $\mathcal{N} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ be a noise tensor with entries drawn from a standard normal distribution. Then our observed tensor is

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

where the parameter $\eta \in \{ 0.01, 0.10 \}$ is the amount of noise. Generally, the rank is unknown, so we run the algorithms with $R \in \{ R_{\text{true}}, R_{\text{true}} + 1 \}$. The parameters for the experiments are summarized in Table 1.

TABLE 1
Parameters varied in synthetic experiments.

Parameter	Values
Order & size (N, I)	$\{ (3, 400), (4, 90) \}$
True # components (R_{true})	5
Collinearity (C)	$\{ 0.5, 0.9 \}$
Noise (η)	$\{ 0.01, 0.1 \}$
Model # components (R)	$\{ 5, 6 \}$

We use the standard *score* metric to measure how well the ground truth is recovered by a CP decomposition in those cases where the true factors are known [39]. The score between two rank-one tensors $\mathcal{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ and $\mathcal{Y} = \mathbf{p} \circ \mathbf{q} \circ \mathbf{r}$ is defined as

$$\text{score}(\mathcal{X}, \mathcal{Y}) = \frac{\mathbf{a}^\mathsf{T}\mathbf{p}}{\|\mathbf{a}\|\|\mathbf{p}\|} \times \frac{\mathbf{b}^\mathsf{T}\mathbf{q}}{\|\mathbf{b}\|\|\mathbf{q}\|} \times \frac{\mathbf{c}^\mathsf{T}\mathbf{r}}{\|\mathbf{c}\|\|\mathbf{r}\|}. \tag{18}$$

The $\boldsymbol{\lambda}$ values are ignored. For $R > 1$, we average the scores for all pairs of components. Alas, the CP decomposition does not recover factors in their original order, so the score is the maximal average across all permutations of rank-one components.

We use the *fit* to determine convergence, and the fit is defined as

$$F = 1 - \frac{\|\mathcal{X} - \tilde{\mathcal{X}}\|}{\|\mathcal{X}\|}.$$

For a tensor with η noise, we expect the final fit to be at best $1 - \eta$. In the presence of noise, maximizing fit does not exactly correspond to maximizing score. The method terminates when either the number of iterations exceeds 200, the change in fit goes below 10^{-4} ($|F_t - F_{t-1}| \leq 10^{-4}$), or the fit is within 20% of the noise level ($F_t \geq 1 - 1.2\eta$). All methods use the same criteria for termination with the exception that CPRAND and CPRAND-MIX use an approximation to the fit, as discussed in subsection 3.4. Specifically, these methods compute the error at $\hat{P}$ entries and use that to estimate the overall fit. We use $\hat{P} = 2^{14}$ in these experiments, and we stress that the same $\hat{P}$ entries are used across all iterations.

As mentioned previously, we use the CP-ALS method provided by the Tensor Toolbox for MATLAB. For CPRAND and CPRAND-MIX, the number of rows sampled for each least squares solve is $S = 80$ for $R = 5$ and $S = 108$ for $R = 6$. Here we stress that we make a new random selection of rows at each iteration.

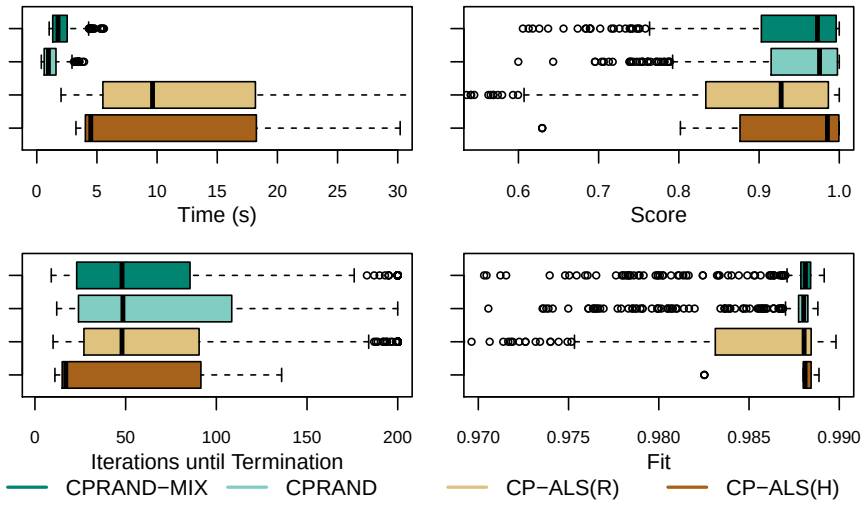
For each possible combination of tensor order/size, collinearity, and noise level (listed in Table 1), we generate 50 synthetic tensors. In Figures 6 and 7, we show box plots comparing CPRAND and CPRAND-MIX with random initialization versus CP-ALS with HOSVD (H) and random (R) initialization. The box plots show a box that indicates the 25th–75th quartiles, and the median is indicated by a vertical line inside the box. Outliers are displayed as circles. Each subfigure shows the results on 100 distinct tensors, i.e., fixed order/size and noise level with 50 tensors each for $C \in \{ 0.5, 0.9 \}$. We test each method with $R \in \{ 5, 6 \}$. For random initialization, we have three starting points. Therefore, each row in the box plot is the result of 600 runs for CPRAND, CPRAND-MIX, and CP-ALS (R) and 200 runs for CP-ALS (H). For each trial we measure the time, number of outer iterations, fit, and score at termination. More detailed results, separating $R = 5$ and $R = 6$, are provided in the supplementary materials, linked from the main article webpage.

First, we consider the quality of the solutions in terms of the fit (always in the range $[0, 1]$), i.e., the objective function being maximized. This is shown in the lower right plot of each subfigure, and we report the true final fits even if approximate fits are used in the algorithm. The median fits are essentially identical, with a maximum difference of 0.006. The CP-ALS (R) has the highest variance in the fit since it is highly dependent on the quality of the starting point. The CPRAND and CPRAND-MIX have less variance in their fits. In general, the CPRAND methods are much less sensitive to starting point—so much so that we do not even include results using the HOSVD initialization.

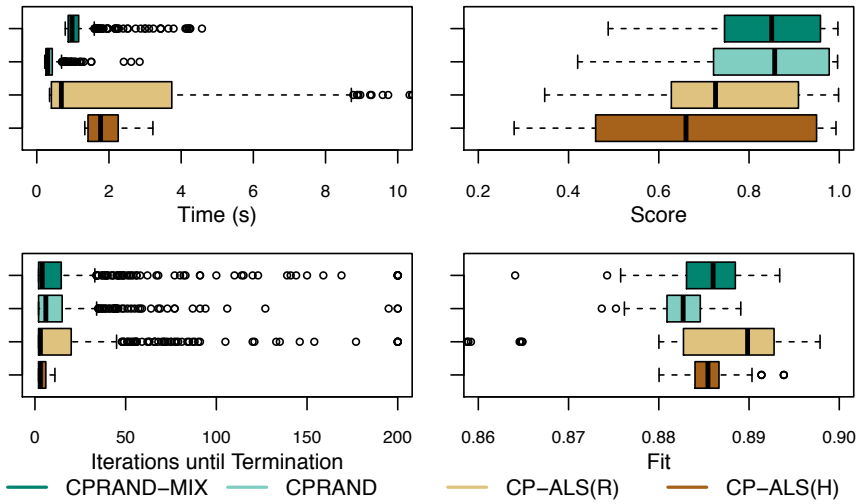
Second, because these problems are synthetic, we know the true underlying factors and thus can use the score from (18), with 1.00 being a perfect match. The scores are shown in the upper right plot of each subfigure. We note that we see little difference between CPRAND and CPRAND-MIX in terms of quality because these artificially generated problems do not have high coherence. At low noise (1%), the methods all do well, with medians above 0.97 for CPRAND, CPRAND-MIX, and CP-ALS(H) in Figure 6a and above 0.99 for all methods in Figure 7a. We see a striking difference, however, at the 10% noise level. The medians for CPRAND and CPRAND-MIX are above 0.85, whereas those for CP-ALS are below 0.73 in Figure 6b. Similarly, medians for CPRAND and CPRAND-MIX are above 0.92, whereas those for CP-ALS are below 0.89 in Figure 7b. We contend that the randomized methods are more robust because they avoid the problem of overfitting to noise thanks to the randomization. This is evidenced by the fact that all methods achieve similar fit but the randomized methods tend to achieve better score.

Third, we look at the number of iterations, shown in the lower left plot of each subfigure. The median number of iterations for the randomized methods is always higher than CP-ALS (H) since it has the advantage of a good starting point. This is because the randomized methods generally make less progress per iteration. With the exception of CP-ALS (H), every method hits the maximum number of iterations (200) at least once.

Fourth, we consider runtime, where we expect to get the most benefit. This is shown in the upper left plot of each subfigure. In the low noise (1%) case, we see an improvement in median runtime of 10X for CPRAND versus CP-ALS (R) in Figure 6a and 30X for the same pair in Figure 7a. The cost of the preprocessing for CPRAND-



(a) Noise $\eta = 1\%$

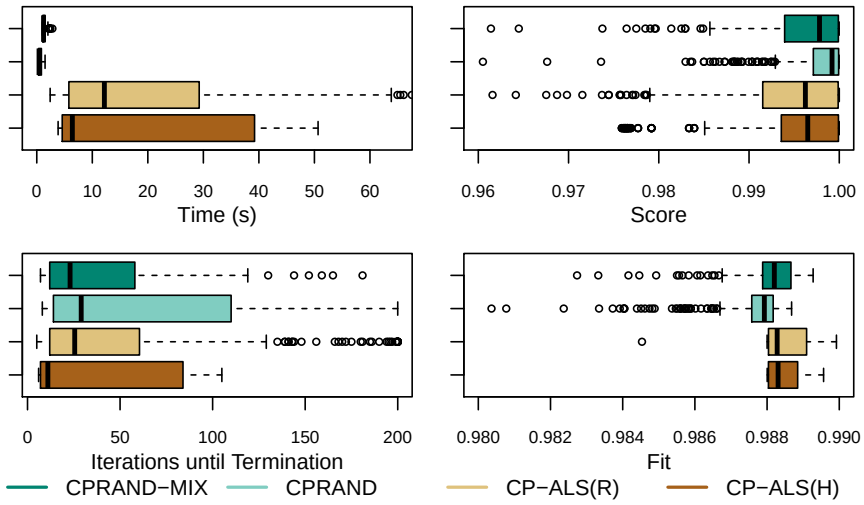


(b) Noise $\eta = 10\%$

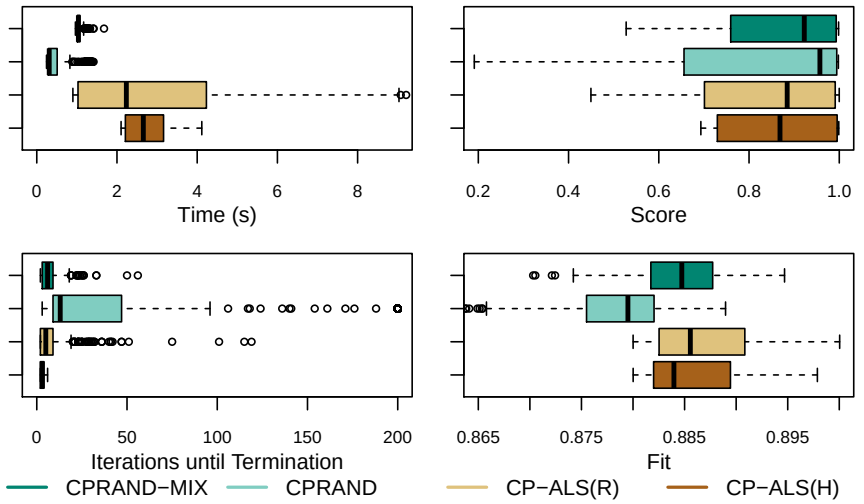
FIG. 6. *Results on* 200 *synthetic tensors of size* $400 \times 400 \times 400$ *with* $R_{true} = 5$ *and collinearity* $C \in \{0.5, 0.9\}$ *in the factors. We stop when the number of iterations exceeds* 200*, the fit stagnates* $|F_t - F_{t-1}| \leq 10^{-4}$*, or the fit exceeds a preset threshold* $F_t \geq 1 - 1.2\eta$*. Each of the four methods is tested with target ranks* $R \in \{5, 6\}$ *and three random starts, except for CP-ALS (H), which has one fixed start. We report results from all starts (*2000 *in all). For CPRAND and CPRAND-MIX, we use* $\hat{P} = 2^{14}$ *random entries to check convergence, and the number of row samples in the least squares solve is* $S = 80$ *for* $R = 5$ *and* $S = 108$ *for* $R = 6$*.*

FFT means it may be about 3X slower than CPRAND without mixing, depending on the number of iterations. In the high noise (10%) case, the difference in time is less dramatic, but we have the improvement in scores discussed above.

In summary, the synthetic results suggest that the CPRAND and CPRAND-MIX methods produce solutions that are at least as good and sometimes much better than



(a) Noise $\eta = 1\%$



(b) Noise $\eta = 10\%$

FIG. 7. *Results on* 200 *synthetic tensors of size* $90 \times 90 \times 90 \times 90$ *with* $R_{true} = 5$ *and collinearity* $C \in \{0.5, 0.9\}$ *in the factors. We stop when the number of iterations exceeds* 200*, the fit stagnates* $|F_t - F_{t-1}| \leq 10^{-4}$*, or the fit exceeds a preset threshold* $F_t \geq 1 - 1.2\eta$*. Each of the four methods is tested with target ranks* $R \in \{5, 6\}$ *and three random starts, except for CP-ALS (H), which has one fixed start. We report results from all starts (*2000 *in all). For CPRAND and CPRAND-MIX, we use* $\hat{P} = 2^{14}$ *random entries to check convergence, and the number of row samples in the least squares solve is* $S = 80$ *for* $R = 5$ *and* $S = 108$ *for* $R = 6$*.*

CP-ALS in terms of quality (fit and score). Moreover, the randomized algorithms are at least as fast as the standard methods and sometimes much faster.

4.3. COIL data set. COIL-100 is an image-recognition data set that contains images of objects in different poses [28] and has been used previously by Zhou, Cichocki, and Xie [45] as a tensor decomposition benchmark. The problem is set up as

TABLE 2

CPRAND-MIX speedup and accuracy on a COIL tensor of size $128 \times 128 \times 3 \times 7200$. *Reporting median runtimes over five trials with random starting points,* $R = 20$ *components,* $\hat{P} = 2^{14}$ *entries for approximate fit, and varying numbers of samples* S. *Speedup compared against the median runtime of CP-ALS over five trials with random starting points.*

# Samples (S)	Speedup	Fit
400	8.38	0.674
450	7.98	0.676
500	6.63	0.677
550	7.29	0.678
600	4.75	0.680
650	4.73	0.680
700	4.77	0.680
750	4.52	0.681
800	3.70	0.682
850	4.90	0.678
900	4.95	0.679
950	4.22	0.682
1000	2.84	0.684
CP-ALS	1.00	0.686

follows. There are 100 different object classes, each of which is imaged from 72 different angles. Each image is sized to 128×128 pixels in three color channels (RGB). If we discard the ground truth, we have a $128 \times 128 \times 3 \times 7200$ tensor of size 2.8GB. The irregular dimensions and large size of the data make for an interesting CP benchmark.

In our experiment, we compare the runtimes of CP-ALS (R) and CPRAND-MIX. Unlike the synthetic experiments, this experiment required mixing to converge for a reasonable number of samples. We use $R = 20$ factors. We ran five trials of CP-ALS and terminated when the change in fit went below 10^{-4}. This yielded a median runtime of 204 seconds and a fit of 0.686. For CPRAND-MIX, we terminate when the fit fails to improve for five consecutive iterations and vary the number of samples (S). For each S, we run five trials with random starting points. We compute the approximate fit for CPRAND-MIX with sample size $\hat{P} = 2^{14}$. We vary S and show the results in Table 2. The fits are very close to the fit obtained by CP-ALS, with speedups as high as $8\times$. The speedup does not decrease monotonically with S since differences in the number of iterations to converge may have some impact. Nevertheless, increasing the number of samples increases the cost per iteration and thus reduces the overall speedup on average.

We show more comparisons between CP-ALS and CPRAND-MIX with a smaller amino acid data set [8] in the supplementary materials.

4.4. Hazardous gases. Vervliet and De Lathauwer [42] demonstrate their randomized block sampling approach for CP on a hazardous gas classification task [41], so we compare on the same data set. The data comes from 899 experiments (actually 900 experiments, but one is omitted) where one of three different hazardous gases (carbon monoxide, acetaldehyde, or ammonia) is released into a wind tunnel and its concentration is measured across 72 sensors for 25,900 time steps. We use the exact same preprocessing script as Vervliet and De Lathauwer [42]: missing values are interpolated, and the data is normalized, cropped, and centered. The resulting tensor of size $25{,}900 \times 72 \times 899$ requires 13.4 GB storage. The first mode corresponds to the 25,900 time steps, the second mode corresponds to the 72 sensors, and the third mode corresponds to the 899 experiments. We compute the CP decomposition with $R = 5$ (as in [42]). We run CP-ALS with both random (R) and HOSVD (H)

TABLE 3

Results over 10 trials on a $25{,}900 \times 72 \times 899$ *tensor representing experiments with three hazardous gases. We use* $R = 5$ *factors and stop when the fit stagnates, i.e.,* $|F_t - F_{t-1}| \leq 10^{-4}$ *for CP-ALS, or when estimated fit fails to improve after* 10 *iterations of CPRAND. For CPRAND, we use* $\hat{P} = 2^{14}$ *random entries to check convergence and* $S = 1000$ *row samples in each least squares solve. We run k-means on three columns of the experiment factor matrix with a target of three clusters. Using the k-means output we report the median proportion of* 899 *experiments that are misclassified (according to which hazardous gas was used).*

Method	Median time (s)	Median fit	Median classification error
CPRAND	53.6	0.715	0.61%
CP-ALS (H)	578.4	0.724	0.67%
CP-ALS (R)	204.7	0.724	0.67%

initialization. Due to the size of the tensor, we run CPRAND without mixing, using random initialization, a sample size of $S = 1000$ rows per least squares solve, and $\hat{P} = 2^{14}$ entries for the stopping condition. The ALS methods terminate when the change in fit goes below 10^{-4} (i.e., $|F_t - F_{t-1}| \leq 10^{-4}$), and CPRAND terminates when the estimated fit fails to improve after 10 iterations. We run 10 trials each of CPRAND and CP-ALS(R) and a single trial of CP-ALS(H). The median runtimes are listed in Table 3, and we can see that the median time for CPRAND is less than one minute. CPRAND was nearly $4\times$ faster than CP-ALS(R) and achieves roughly the same classification error. CPRAND is $10\times$ faster than CP-ALS(H), which incurs a high initialization cost. We cannot compare runtimes with Vervliet and De Lathauwer [42] since they are on a different computational architecture, but they report a runtime of less than three minutes for their method, which is in the same ballpark.

We next consider the quality of the decomposition. Vervliet and De Lathauwer manually selected three factors (column vectors) from the experiment factor matrix and used those to classify the experiments according to which of the three gases was released. We do a similar experiment, except rather than choosing the three vectors manually, we tried all 10 (five choose three) possible choices of three vectors and report on the best one. The rows of this subfactor matrix can then be thought of as three-dimensional points. We run k-means on these points and measure the classification error, which is the percentage of the 899 experiments that are misclassified. For each trial we performed a single run of k-means with random initialization. The median fits and classification errors over all trials are shown in Table 3. Both ALS methods achieve a median classification error of 0.67% (6 misclassified), while CPRAND achieves a marginally better classification error of 0.61% (5.5 misclassified) despite a slightly lower fit value. For comparison, Vervliet and De Lathauwer [42] report classification errors of 0.3–0.8% for 100 runs of their randomized block sampling approach, with the addition of a specialized step criterion; without the specialized step, their performance degrades to 5% error.

We visualize the factors computed by CPRAND in Figure 8. It is easy to see that the results can be used to classify the gases. For instance, the first factor clearly separates the purple gas from the green and red. Similarly, the fourth factor clearly separates the red from the green and purple. The fifth factor is the smallest magnitude and appears to be a "noise" factor.

5. Related work. Vervliet and De Lauthauwer present a stochastic gradient descent (SGD) algorithm for CP that samples blocks from the original tensor to update corresponding blocks of the factor matrices [42]. This approach is similar in spirit to CPRAND but takes an altogether different approach to the randomization. They use

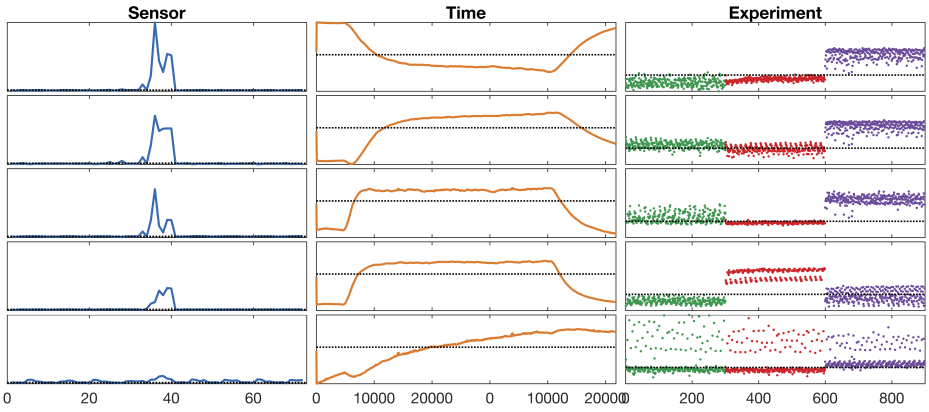


FIG. 8. *Visualization of the five factors from the* $25900 \times 72 \times 899$ *hazardous gas tensor, as computed by CRAND. The factors are sorted by magnitude, from largest at the top to smallest at the bottom. The magnitude is reflected in the sensor factors (left). The time (middle) and experiment (right) factors are normalized to unit norm. The three gas types are color-coded (seen only in the online version) in the experiments' factors. All factors in the same mode are plotted on the same y-scale, and the dashed line is zero.*

contiguous samples in the block updates and finer control over step sizes. They also update only a portion of each factor matrix in each iteration. An interesting contrast between the two methods can be seen by comparing the factors that are computed in the example from subsection 4.4. Our factors are shown in Figure 8. The time factors are much smoother than the factors pictured in [42]. Since the block method updates only a small subset of each factor at a time, we conjecture that this may explain the blockiness of the solution. Vervliet and De Lauthauwer [42] also propose an inexpensive stopping condition based on Cramer–Rao bounds, but these require some additional knowledge about the noise level. Another framework that draws from SGD is FlexiFaCT, which targets coupled tensor decompositions for parallel computation [7].

Cheng et al. have recently applied a leverage score–based sampling to the least squares step of the sparse CP decomposition by showing how leverage scores of an unfolded tensor can be estimated by the leverage scores of the factor matrices [11]. This approach is similar to the way that we bound the coherence of Khatri–Rao products. Reynolds, Doostan, and Beylkin also use randomization within CP-ALS, specifically for the case of rank reduction, where the input to the algorithm is already in CP format [34]. They use randomization to improve the conditioning of the individual least squares problems in order to compute better overall approximations. The randomization makes each iteration of their method more costly, but they observe faster convergence (and overall running time) than ALS for ill-conditioned problems.

Wang et al. have applied sketching methods to *orthogonal* tensors with provable guarantees [43]. Song, Woodruff, and Zhang show that this sketch can be computed without reading the entire tensor (in sublinear time) under certain conditions [38].

An alternative to sketching is to compress the tensor using lossy methods before computation. Zhou, Cichocki, and Xie examine the effectiveness of performing a CP decomposition on a compressed representation of the data using the lossy Tucker decomposition to produce a smaller problem size [45]. ParCube [30] compresses the original tensor by directly sampling and performs a decomposition on the result. Another useful approach for high-order tensors is known as tensor reshaping, which

can cast much of the computation in terms of three-way tensors [31].

6. Conclusion. We have provided an example of the power of randomized methods in the context of CP decompositions for dense tensors. As discussed in the related work (section 5), a few approaches have recently been proposed. Ours is a unique approach that focuses on the least squares subproblem. The advantage of this approach is that we can leverage existing theory and methodology. Specifically, we have a practical implementation, using MATLAB and the Tensor Toolbox, that efficiently employs randomized least squares in the context of CP-ALS.

The least squares subproblems have a special structure that allows for very efficient solution; however, this still requires formation of the Khatri–Rao matrix and multiplication with the matricized tensor, which is the primary computational bottleneck. In our implementation of the randomized approach, we entirely avoid forming the Khatri–Rao matrix and thus greatly reduce the expense of the least squares solve. We refer to this method as CPRAND.

It is oftentimes a good idea to apply an FJLT to the least squares problem to ensure incoherence. We explain how this can be done in a preprocessing step rather than for every least squares solve. Assuming that we use an FFT in the FJLT, we have to reverse part of the transformation for each linear solve. However, we need only apply the inverse transform to the small sampled matrix at trivial cost. We refer to this method as CPRAND-MIX. A small disadvantage of the FFT is that it transforms a real-valued problem to be complex-valued, doubling the memory requirement. The computational cost difference, however, is negligible. On the other hand, we could use a real-valued transform, but these proved to be slower than the FFT in MATLAB with no improvement in the quality of the CP decomposition.

Checking the stopping condition is also a significant expense. This is based on the fit of the model to the original data and thus requires forming the Khatri–Rao matrix, an expense we avoid in the least squares solves. We employ another type of randomization in this case, based on using just a subsample of the tensor entries for comparison. Assuming the errors are i.i.d. and drawn from a finite distribution, we can approximate the model fit error with reasonable accuracy and substantially reduced cost, so we employ this stopping condition in our randomized algorithms.

We have demonstrated the benefits of CPRAND and CPRAND-MIX in both synthetic and real data experiments, including large-scale tensors of up to 13 GB in size. The randomized methods are overall much faster than CP-ALS for equivalent quality decompositions. Moreover, the CPRAND methods are much less sensitive to the initial guess. We conjecture that the randomization prevents getting stuck in a local minimum caused by overfitting the noise. The CPRAND-MIX is more expensive to initialize (requiring the application of an FFT in each mode) but has the advantage of ensuring coherence. We found that mixing was critical for good performance on the COIL-100 dataset in subsection 4.3 but unnecessary for the hazardous gas data set in subsection 4.4. The expense of the mixing is roughly equivalent to computing the HOSVD initialization.

Stopping conditions present an interesting dilemma for any randomized method. Since each subproblem is now solved inexactly, the fit is no longer monotonically increasing. In particular, it is difficult to detect when improvement has stagnated. If we know the amount of noise in advance, we can terminate once the desired fit is achieved. However, this assumes not only that we know the noise but also that our model is good in the sense that the data has inherent multilinear structure and the rank is known. Instead, we propose a modification of the standard stagnation metric:

track the best fit and stop when it fails to improve for more than, say, 10 iterations.

Although there is some theory on the number of samples required for least squares as in (9), they are impractical for most implementations. We lack a rigorous way of estimating a good sample size. In practice, we have found that a small multiple of R is sufficient, i.e., 10–100 times R. Clearly, more theoretical work to justify this choice is needed.

Since the CP fitting problem is nonconvex, CP-ALS cannot guarantee global optimality. This is unchanged for randomized methods. Moreover, as mentioned above, we do not even have a guarantee of improving the objective function at each step. However, our experimental results indicate that randomized methods are more robust to the starting point, so this is a potential advantage and perhaps a topic for future research.

Many lines of future research remain, in addition to those mentioned above. As discussed in subsection 3.2.1, the most expensive part of CPRAND is extracting the random fibers from the dense tensor (i.e., memory operations), so we may consider both algorithmic adjustments and specialized implementations to alleviate that expense. We would also like to compare directly to other improved methods for computing CP, such as those in [32, 42]. Another topic of investigation is to prove that the mixing operator we develop in subsection 3.3 is an FJLT. An obvious extension is consideration of sparse tensors. In the sparse case, we never form the Khatri–Rao matrix (see [5]), so the bottlenecks are different. We note that our mixing process would convert the sparse tensor to a dense one, so we suspect that a nonuniform sampling scheme without mixing (as in [11]) will be more effective for sparse data. These sketching methods also naturally extend to out-of-core algorithms and may even be used to increase scalability in distributed memory. Finally, it is natural to consider application of randomization to other decompositions, such as Tucker [40], tensor train [29], or functional tensor [12, 18].

The CPRAND method is called `cp_arls` in the Tensor Toolbox for MATLAB, available as open source at www.tensortoolbox.org.

Acknowledgments. We thank Nico Vervliet for generously sharing his scripts to preprocess the hazardous gases experiment data used in subsection 4.4. We would like to thank Alex Williams for the CP decomposition visualization script used to create Figure 8.

REFERENCES

[1] E. Acar, C. A. Bingol, H. Bingol, R. Bro, and B. Yener, *Multiway analysis of epilepsy tensors*, Bioinformatics, 23 (2007), pp. i10–i18, https://doi.org/10.1093/bioinformatics/btm210.

[2] N. Ailon and B. Chazelle, *Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform*, in Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC'06, 2006, pp. 557–563, https://doi.org/10.1145/1132516.1132597.

[3] H. Avron, P. Maymounkov, and S. Toledo, *Blendenpik: Supercharging LAPACK's least-squares solver*, SIAM J. Sci. Comput., 32 (2010), pp. 1217–1236, https://doi.org/10.1137/090767911.

[4] 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, https://doi.org/10.1145/1186785.1186794.

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

[6] B. W. Bader, T. G. Kolda, et al., *MATLAB Tensor Toolbox (Version* 2.6*)*, available online February 6, 2015, http://www.sandia.gov/~tgkolda/TensorToolbox/.

[7] A. Beutel, P. P. Talukdar, A. Kumar, C. Faloutsos, E. E. Papalexakis, and E. P. Xing, *FlexiFaCT: Scalable flexible factorization of coupled tensors on Hadoop*, in Proceedings of the 2014 SIAM International Conference on Data Mining, SDM'14, 2014, pp. 109–117, https://doi.org/10.1137/1.9781611973440.13.

[8] R. Bro, *PARAFAC. Tutorial and applications*, Chemom. Intell. Lab. Syst., 38 (1997), pp. 149–171, https://doi.org/10.1016/S0169-7439(97)00032-4.

[9] E. Candès and B. Recht, *Exact matrix completion via convex optimization*, Commun. ACM, 55 (2012), pp. 111–119, https://doi.org/10.1145/2184319.2184343.

[10] J. Carroll and J.-J. Chang, *Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition*, Psychometrika, 35 (1970), pp. 283–319, https://doi.org/10.1007/BF02310791.

[11] D. Cheng, R. Peng, I. Perros, and Y. Liu, *SPALS: Fast alternating least squares via implicit leverage scores sampling*, in Proceedings of the 29th Annual Conference on Neural Information Processing Systems, NIPS'16, Currant Associates, Inc., Red Hook, NY, 2016, http://papers.nips.cc/paper/6436-spals-fast-alternating-least-squares-via-implicit-leverage-scores-sampling.pdf.

[12] M. Chevreuil, R. Lebrun, A. Nouy, and P. Rai, *A least-squares method for sparse low rank approximation of multivariate functions*, SIAM/ASA J. Uncertain. Quantif., 3 (2015), pp. 897–921, https://doi.org/10.1137/13091899X.

[13] J. H. Choi and S. Vishwanathan, *DFacTo: Distributed factorization of tensors*, in Proceedings of the 27th Annual Conference on Neural Information Processing Systems, NIPS'14, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds., Curran Associates, Inc., Red Hook, NY, 2014, pp. 1296–1304, http://papers.nips.cc/paper/5395-dfacto-distributed-factorization-of-tensors.pdf.

[14] F. Cong, Q.-H. Lin, L.-D. Kuang, X.-F. Gong, P. Astikainen, and T. Ristaniemi, *Tensor decomposition of EEG signals: A brief review*, J. Neurosci. Methods, 248 (2015), pp. 59–69, https://doi.org/10.1016/j.jneumeth.2015.03.018.

[15] I. Davidson, S. Gilpin, O. Carmichael, and P. Walker, *Network discovery via constrained tensor analysis of fMRI data*, in Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (KDD'13), 2013, pp. 194–202, https://doi.org/10.1145/2487575.2487619.

[16] D. L. Donoho and X. Huo, *Uncertainty principles and ideal atomic decomposition*, IEEE Trans. Inform. Theory, 47 (2006), pp. 2845–2862, https://doi.org/10.1109/18.959265.

[17] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, *Faster least squares approximation*, Numer. Math., 117 (2011), pp. 219–249, https://doi.org/10.1007/s00211-010-0331-6.

[18] A. A. Gorodetsky, S. Karaman, and Y. M. Marzouk, *Function-Train: A Continuous Analogue of the Tensor-Train Decomposition*, preprint, https://arxiv.org/abs/1510.09088v2, 2015.

[19] R. A. Harshman, *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis*, UCLA Working Papers in Phonetics, 16 (1970).

[20] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Phys., 6 (1927), pp. 164–189, https://doi.org/10.1002/sapm192761164.

[21] R. Jaffé, K. M. Cawley, and Y. Yamashita, *Applications of excitation emission matrix fluorescence with parallel factor analysis (EEM-PARAFAC) in assessing environmental dynamics of natural dissolved organic matter (DOM) in aquatic environments: A review*, in Advances in the Physicochemical Characterization of Dissolved Organic Matter: Impact on Natural and Engineered Systems, ACS Symposium Series 1160, American Chemical Society (ACS), Washington, D.C., 2014, pp. 27–73, https://doi.org/10.1021/bk-2014-1160.ch003.

[22] W. Johnson and J. Lindenstrauss, *Extensions of Lipschitz mappings into a Hilbert space*, in Conference in Modern Analysis and Probability (New Haven, CT, 1982), Contemp. Math. 26, American Mathematical Society, Providence, RI, 1984, pp. 189–206, https://doi.org/10.1007/BF02764938.

[23] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500, https://doi.org/10.1137/07070111X.

[24] J. Li, C. Battaglino, I. Perros, J. Sun, and R. Vuduc, *An input-adaptive and in-place approach to dense tensor-times-matrix multiply*, in Proceedings of the ACM/IEEE Conference on Supercomputing, SC '15, Austin, TX, 2015, https://doi.org/10.1145/2807591.2807671.

[25] K. Maruhashi, F. Guo, and C. Faloutsos, *MultiAspectForensics: Pattern mining on large-scale heterogeneous networks with tensor analysis*, in Proceedings of the 2011 International

Conference on Advances in Social Networks Analysis and Mining, ASONAM '11, 2011, pp. 203–210, https://doi.org/10.1109/asonam.2011.80.

[26] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, 1995.

[27] K. R. Murphy, C. A. Stedmon, D. Graeber, and R. Bro, *Fluorescence spectroscopy and multi-way techniques. PARAFAC*, Anal. Methods, 5 (2013), pp. 6557–6566, https://doi.org/10.1039/c3ay41160e.

[28] S. Nene, S. Nayar, and H. Murase, *Columbia Object Image Library (COIL-100)*, Tech. Report CUCS-006-96, Columbia University, New York, NY, 1996.

[29] I. V. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317, https://doi.org/10.1137/090752286.

[30] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos, *ParCube: Sparse parallelizable tensor decompositions*, in Machine Learning and Knowledge Discovery in Databases (European Conference, ECML PKDD 2012), Lecture Notes in Comput. Sci. 7523, Springer, Cham, 2012, pp. 521–536, https://doi.org/10.1007/978-3-642-33460-3_39.

[31] A.-H. Phan, P. Tichavský, and A. Cichocki, *CANDECOMP/PARAFAC decomposition of high-order tensors through tensor reshaping*, IEEE Trans. Signal Process., 61 (2013), pp. 4847–4860, https://doi.org/10.1109/TSP.2013.2269046.

[32] A.-H. Phan, P. Tichavský, and A. Cichocki, *Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations*, IEEE Trans. Signal. Process., 61 (2013), pp. 4834–4846, https://doi.org/10.1109/TSP.2013.2269903.

[33] M. Rajih, P. Comon, and R. A. Harshman, *Enhanced line search: A novel method to accelerate PARAFAC*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1128–1147, https://doi.org/10.1137/06065577.

[34] M. J. Reynolds, A. Doostan, and G. Beylkin, *Randomized alternating least squares for canonical tensor decompositions: Application to a PDE with random data*, SIAM J. Sci. Comput., 38 (2016), pp. A2634–A2664, https://doi.org/10.1137/15M1042802.

[35] V. Rokhlin and M. Tygert, *A fast randomized algorithm for overdetermined linear least-squares regression*, Proc. Natl. Acad. Sci. USA, 105 (2008), pp. 13212–13217, https://doi.org/10.1073/pnas.0804869105.

[36] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, *Tensor Decomposition for Signal Processing and Machine Learning*, preprint, https://arxiv.org/abs/1607.01668, 2016.

[37] S. Smith, N. Ravindran, N. Sidiropoulos, and G. Karypis, *SPLATT: Efficient and parallel sparse tensor-matrix multiplication*, in Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium, IPDPS '15, 2015, https://doi.org/10.1109/IPDPS.2015.27.

[38] Z. Song, D. P. Woodruff, and H. Zhang, *Sublinear time orthogonal tensor decomposition*, in Proceedings of the 29th Annual Conference on Neural Information Processing Systems, NIPS '16, Currant Associates, Inc., Red Hook, NY, 2016, http://www.cs.cmu.edu/afs/cs/user/dwoodruf/www/swz16.pdf.

[39] G. Tomasi and R. Bro, *A comparison of algorithms for fitting the PARAFAC model*, Comput. Stat. Data Anal., 50 (2006), pp. 1700–1734, https://doi.org/10.1016/j.csda.2004.11.013.

[40] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311, https://doi.org/10.1007/BF02289464.

[41] A. Vergara, J. Fonollosa, J. Mahiques, M. Trincavelli, N. Rulkov, and R. Huerta, *On the performance of gas sensor arrays in open sampling systems using inhibitory support vector machines*, Sensors Actuat. B Chem., 185 (2013), pp. 462–477, https://doi.org/10.1016/j.snb.2013.05.027.

[42] N. Vervliet and L. De Lathauwer, *A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors*, IEEE J. Sel. Top. Signal Process., 10 (2016), pp. 284–295, https://doi.org/10.1109/JSTSP.2015.2503260.

[43] Y. Wang, H.-Y. Tung, A. J. Smola, and A. Anandkumar, *Fast and guaranteed tensor decomposition via sketching*, in Proceedings of the 28th Annual Conference on Neural Information Processing Systems, NIP '15, Currant Associates, Inc., Red Hook, NY, 2015, pp. 991–999, http://papers.nips.cc/paper/5944-fast-and-guaranteed-tensor-decomposition-via-sketching.pdf.

[44] D. P. Woodruff, *Sketching as a tool for numerical linear algebra*, Found. Trends Theor. Comput. Sci., 10 (2014), pp. 1–157, https://doi.org/10.1561/0400000060.

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