SIAM J. SCI. COMPUT.
Vol. 39, No. 5, pp. A1879–A1902

RECOMPRESSION OF HADAMARD PRODUCTS OF TENSORS IN TUCKER FORMAT*

DANIEL KRESSNER† AND LANA PERIŠA‡

Abstract. The Hadamard product features prominently in tensor-based algorithms in scientific computing and data analysis. Due to its tendency to significantly increase ranks, the Hadamard product can represent a major computational obstacle in algorithms based on low-rank tensor representations. It is therefore of interest to develop recompression techniques that mitigate the effects of this rank increase. In this work, we investigate such techniques for the case of the Tucker format, which is well suited for tensors of low order and small to moderate multilinear ranks. Fast algorithms are attained by combining iterative methods, such as the Lanczos method and randomized algorithms, with fast matrix-vector products that exploit the structure of Hadamard products. The resulting complexity reduction is particularly relevant for tensors featuring large mode sizes I and small to moderate multilinear ranks R. To implement our algorithms, we have created a new Julia library for tensors in Tucker format.

Key words. tensors, Tucker format, HOSVD, Hadamard product

AMS subject classifications. 15A69, 65F99, 65Y20

DOI. 10.1137/16M1093896

1. Introduction.

A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an N-dimensional array of real numbers, with the integer I_n denoting the size of the nth mode for $n = 1, \ldots, N$. The Hadamard product $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ for two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is defined as the entrywise product $z_{i_1,\ldots,i_N} = x_{i_1,\ldots,i_N} y_{i_1,\ldots,i_N}$ for $i_n = 1, \ldots, I_n$, $n = 1, \ldots, N$.

The Hadamard product is a fundamental building block of tensor-based algorithms in scientific computing and data analysis. This is partly because Hadamard products of tensors correspond to products of multivariate functions. To see this, consider two functions $u, v : [0,1]^N \to \mathbb{R}$ and discretizations $0 \le \xi_{n,1} < \xi_{n,2} < \cdots < \xi_{n,I_n} \le 1$ of the interval $[0,1]$. Let the tensors $\mathcal{X}$ and $\mathcal{Y}$ contain the functions u and v evaluated on these grid points, that is, $x_{i_1,\ldots,i_N} = u(\xi_{i_1}, \ldots, \xi_{i_N})$ and $y_{i_1,\ldots,i_N} = v(\xi_{i_1}, \ldots, \xi_{i_N})$ for $i_n = 1, \ldots, I_n$. Then the tensor $\mathcal{Z}$ containing the function values of the product uv satisfies $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$. Applied recursively, Hadamard products allow one to deal with other nonlinearities, including polynomials in u, such as $1 + u + u^2 + u^3 + \cdots$, and functions that can be well approximated by a polynomial, such as $\exp(u)$. Other applications of Hadamard products include the computation of level sets [6], reciprocals [18], minima and maxima [8], variances, and higher-order moments in uncertainty quantification [4, 7], as well as weighted tensor completion [9]. To give an example for a specific application that will benefit from the developments of this paper, let us consider the three-dimensional reaction-diffusion equation from [21, sect. 7]:

$$\frac{\partial u}{\partial t} = \triangle u + u^3, \qquad u(t) : [0,1]^3 \to \mathbb{R} \tag{1}$$

*Submitted to the journal's Methods and Algorithms for Scientific Computing section September 14, 2016; accepted for publication (in revised form) June 28, 2017; published electronically September 7, 2017.

http://www.siam.org/journals/sisc/39-5/M109389.html

†MATHICSE-ANCHP, École Polytechnique Fédérale de Lausanne, Station 8, 1015 Lausanne, Switzerland (daniel.kressner@epfl.ch).

‡Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, University of Split, Rudjera Boškovića 32, 21000 Split, Croatia (lana.perisa@fesb.hr).

with Dirichlet boundary conditions and suitably chosen initial data. After a standard finite-difference discretization in space, (1) becomes an ordinary differential equation (ODE) with a state vector that can be reshaped as a tensor of order three. When using a low-rank method for ODEs, such as the dynamical tensor approximation by Koch and Lubich [16], the evaluation of the nonlinearity u^3 corresponds to approximating $\mathcal{X} * \mathcal{X} * \mathcal{X}$ for a low-rank tensor $\mathcal{X}$ approximating u. For sufficiently large tensors, this operation can be expected to dominate the computational time, simply because of the rank growth associated with it. The methods presented in this paper allow one to successively approximate $(\mathcal{X} * \mathcal{X}) * \mathcal{X}$ much more efficiently for the Tucker format, which is the low-rank format considered in [16].

The Tucker format compactly represents tensors of low (multilinear) rank. For an $I_1 \times \cdots \times I_N$ tensor of multilinear rank $(R_1, \ldots, R_N)$ only $R_1 \cdots R_N + R_1 I_1 + \cdots + R_N I_N$ instead of $I_1 \cdots I_N$ entries need to be stored. This makes the Tucker format particularly suitable for tensors of low order (say, $N = 3, 4, 5$) and it in fact constitutes the basis of the Chebfun3 software for trivariate functions [14]. For function-related tensors, it is known that smoothness of u, v implies that $\mathcal{X}, \mathcal{Y}$ can be well approximated by low-rank tensors [10, 26]. In turn, the tensor $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ also admits a good low-rank approximation. This property is not fully reflected on the algebraic side; the Hadamard product generically multiplies (and thus often drastically increases) multilinear ranks. In turn, this makes it difficult to exploit low ranks in the design of computationally efficient algorithms for Hadamard products. For example when $N = 3$, $I_1 = I_2 = I_3 = I$, and all involved multilinear ranks are bounded by R. We will see below that a straightforward recompression technique, called HOSVD2, based on a combination of the higher-order singular value decomposition (HOSVD) with the randomized algorithms from [13] requires $\mathcal{O}(R^4 I + R^8)$ operations and $\mathcal{O}(R^2 I + R^6)$ memory. This excessive cost is primarily due to the construction of an intermediate $R^2 \times R^2 \times R^2$ core tensor, limiting such an approach to small values of R. We design algorithms that avoid this effect by exploiting structure when performing multiplications with the matricizations of $\mathcal{Z}$. One of our algorithms requires only $\mathcal{O}(R^3 I + R^6)$ operations and $\mathcal{O}(R^2 I + R^4)$ memory.

All algorithms presented in this paper have been implemented in the Julia programming language in order to attain reasonable speed in our numerical experiments at the convenience of a MATLAB implementation. As a by-product of this work, we provide a novel Julia library called TensorToolbox.jl, which is meant to be a general-purpose library for tensors in the Tucker format. It is available from https://github.com/lanaperisa/TensorToolbox.jl and includes all functionality of the `ttensor` class from the MATLAB Tensor toolbox [1].

The rest of this paper is organized as follows. In section 2, we recall basic tensor operations related to the Hadamard product and the Tucker format. Section 3 discusses two variants of the HOSVD for compressing Hadamard products. Section 4 recalls iterative methods for low-rank approximation and their combination with the HOSVD. In section 5, we explain how the structure of $\mathcal{Z}$ can be exploited to accelerate matrix-vector multiplications with its Gramians and the approximation of the core tensor. Section 7 contains the numerical experiments highlighting which algorithmic variant is suitable depending on the tensor sizes and multilinear ranks. Appendix A gives an overview of our Julia library.

2. Preliminaries. In this section, we recall the matrix and tensor operations needed for our developments. If not mentioned otherwise, the material discussed in this section is based on [17].

2.1. Matrix products. Apart from the standard matrix product, the following products between matrices will play a role.

- *Kronecker product*: Given $\mathbf{A} \in \mathbb{R}^{I\times J}, \mathbf{B} \in \mathbb{R}^{K\times L}$,

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{(IK)\times(JL)}.$$

- *Hadamard (elementwise) product*: Given $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{I\times J}$,

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \cdots & a_{IJ}b_{IJ} \end{bmatrix} \in \mathbb{R}^{I\times J}.$$

- *Khatri–Rao product*: Given $\mathbf{A} \in \mathbb{R}^{I\times K}, \mathbf{B} \in \mathbb{R}^{J\times K}$,

$$\mathbf{A}\odot\mathbf{B} = \begin{bmatrix} \mathbf{a}_1\otimes\mathbf{b}_1 & \mathbf{a}_2\otimes\mathbf{b}_2 & \cdots & \mathbf{a}_K\otimes\mathbf{b}_K \end{bmatrix} \in \mathbb{R}^{(IJ)\times K},$$

  where $\mathbf{a}_j$ and $\mathbf{b}_j$ denote the jth columns of $\mathbf{A}$ and $\mathbf{B}$, respectively.
- *Transpose Khatri–Rao product*: Given $\mathbf{A} \in \mathbb{R}^{I\times J}, \mathbf{B} \in \mathbb{R}^{I\times K}$,

$$\mathbf{A}\odot^T\mathbf{B} = \left(\mathbf{A}^T\odot\mathbf{B}^T\right)^T = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_I^T \end{bmatrix} \odot^T \begin{bmatrix} \mathbf{b}_1^T \\ \mathbf{b}_2^T \\ \vdots \\ \mathbf{b}_I^T \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^T\otimes\mathbf{b}_1^T \\ \mathbf{a}_2^T\otimes\mathbf{b}_2^T \\ \vdots \\ \mathbf{a}_I^T\otimes\mathbf{b}_I^T \end{bmatrix} \in \mathbb{R}^{I\times(JK)},$$

  where $\mathbf{a}_i^T$ and $\mathbf{b}_i^T$ denote the ith rows of $\mathbf{A}$ and $\mathbf{B}$, respectively.

We let $\mathrm{vec} : \mathbb{R}^{I\times J} \to \mathbb{R}^{IJ}$ denote the standard vectorization of a matrix obtained from stacking its columns on top of each other. For an $I\times I$ matrix $\mathbf{A}$, we let $\mathrm{diag}(\mathbf{A})$ denote the vector of length I containing the diagonal entries of $\mathbf{A}$. For a vector $\mathbf{v} \in \mathbb{R}^I$, we let $\mathrm{diag}(\mathbf{v})$ denote the $I \times I$ diagonal matrix with diagonal entries $v_1, \ldots, v_I$. We will make use of the following properties, which are mostly well known and can be directly derived from the definitions:

(2) $(\mathbf{A}\otimes\mathbf{B})^T = \mathbf{A}^T\otimes\mathbf{B}^T$,

(3) $(\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) = \mathbf{AC}\otimes\mathbf{BD}$,

(4) $(\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\odot\mathbf{D}) = \mathbf{AC}\odot\mathbf{BD}$,

(5) $(\mathbf{A}\otimes\mathbf{B})\mathbf{v} = \mathrm{vec}(\mathbf{BVA}^T), \quad \mathbf{v} = \mathrm{vec}(\mathbf{V})$,

(6) $(\mathbf{A}\odot\mathbf{B})\mathbf{v} = \mathrm{vec}(\mathbf{B}\,\mathrm{diag}(\mathbf{v})\mathbf{A}^T)$,

(7) $(\mathbf{A}\odot^T\mathbf{B})\mathbf{v} = \mathrm{diag}(\mathbf{BVA}^T), \quad \mathbf{v} = \mathrm{vec}(\mathbf{V})$.

2.2. Norm and matricizations. The *Frobenius norm* of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_N}$ is given by

$$\|\mathcal{X}\|_F = \sqrt{\sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}\cdots\sum_{i_N=1}^{I_N} x_{i_1 i_2\cdots i_N}^2}.$$

The *n-mode matricization* turns a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ into an $I_n \times (I_1 \cdots I_{n-1} I_{n+1} \cdots I_N)$ matrix $\mathbf{X}_{(n)}$. More specifically, the nth index of $\mathcal{X}$ becomes the row index of $\mathbf{X}_{(n)}$ and the other indices of $\mathcal{X}$ are merged into the column index of $\mathbf{X}_{(n)}$ in reverse lexicographical order; see [17] for more details.

The *n-mode product* of $\mathcal{X}$ with a matrix $\mathbf{A} \in \mathbb{R}^{J \times I_n}$ yields the tensor $\mathcal{Y} = \mathcal{X} \times_n \mathbf{A} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$ defined by

$$y_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} a_{j i_n}.$$

In terms of matricizations, the n-mode product becomes a matrix-matrix multiplication,

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

In particular, properties of matrix products, such as associativity, directly carry over to n-mode products. In terms of the vectorization of a tensor, denoted by $\mathrm{vec}(\mathcal{X})$, the n-mode product corresponds to

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{A} \quad \Leftrightarrow \quad \mathrm{vec}(\mathcal{Y}) = (\mathbf{I}_N \otimes \cdots \otimes \mathbf{I}_{n+1} \otimes \mathbf{A} \otimes \mathbf{I}_{n-1} \otimes \cdots \otimes \mathbf{I}_1)\mathrm{vec}(\mathcal{X}),$$

where $\mathbf{I}_j$ is the $I_j \times I_j$ identity matrix. Combining both characterizations, we obtain

$$\begin{aligned} (8) \quad & \mathcal{Y} = \mathcal{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)} \\ & \Leftrightarrow \mathbf{Y}_{(n)} = \mathbf{A}^{(n)} \mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \cdots \otimes \mathbf{A}^{(1)} \right)^T. \end{aligned}$$

2.3. Tensors in Tucker format and multilinear rank. We say that a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is in *Tucker format* if it is represented as

$$(9) \qquad \mathcal{X} = \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \cdots \times_N \mathbf{A}^{(N)}$$

with the so-called *core tensor* $\mathcal{F} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ and *factor matrices* $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, $n = 1, \ldots, N$.

The *multilinear rank* of a tensor is the N-tuple

$$\left(\mathrm{rank}\left(\mathbf{X}_{(1)}\right), \ldots, \mathrm{rank}\left(\mathbf{X}_{(N)}\right) \right).$$

Because of (8), a tensor $\mathcal{X}$ in Tucker format (9) satisfies $\mathrm{rank}(\mathbf{X}_{(n)}) \leq R_n$. Vice versa, any tensor admits a *Tucker decomposition* with $R_n = \mathrm{rank}(\mathbf{X}_{(n)})$, i.e., it can be transformed into the Tucker format (9). The HOSVD discussed in section 3.1 is one way of performing such a Tucker decomposition.

2.4. Matricization of Kronecker products. The *Kronecker product* of two tensors $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\mathcal{Y} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ is the tensor $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y} \in \mathbb{R}^{I_1 J_1 \times \cdots \times I_N J_N}$ with entries $z_{k_1 \cdots k_N} = x_{i_1 \cdots i_N} y_{j_1 \cdots j_N}$ for $k_\ell = j_\ell + (i_\ell - 1)J_\ell$. In particular, the Hadamard product $\mathcal{X} * \mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is a subtensor of $\mathcal{X} \otimes \mathcal{Y}$.

The matricization of $\mathcal{X} \otimes \mathcal{Y}$ is in general *not* equal to the Kronecker product of matricizations of $\mathcal{X}$ and $\mathcal{Y}$. However, as discussed by Ragnarsson [23] and Ragnarsson and Van Loan [24], there exists a permutation matrix $\mathbf{P}_n$, only depending on the sizes of $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, $\mathcal{Y} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ and the mode of the matricization, such that

$$(10) \qquad (\mathcal{X} \otimes \mathcal{Y})_{(n)} = \left(\mathbf{X}_{(n)} \otimes \mathbf{Y}_{(n)} \right) \mathbf{P}_n.$$

Instead of giving an explicit expression for $\mathbf{P}_n$, which is tedious and not needed in the following, we will only discuss the realization of a matrix-vector product $\mathbf{P}_n\mathbf{v}$, or equivalently $\mathbf{v}^T\mathbf{P}_n^T$, for a vector $\mathbf{v}$ of length $I_1J_1\cdots I_{n-1}J_{n-1}I_{n+1}J_{n+1}\cdots I_NJ_N$. We first reshape $\mathbf{v}$ into a tensor $\mathcal{V}$ of size $J_1 \times I_1 \times \cdots \times J_{n-1} \times I_{n-1} \times J_{n+1} \times I_{n+1} \times \cdots \times J_N \times I_N$, which matches the structure of a row on the left-hand side of (10). In order to match the structure of a row on the right-hand side of (10), we need to apply the perfect shuffle permutation

$$\pi_n = \begin{bmatrix} 1 & 3 & 5 & \cdots & (2N-3) & 2 & 4 & \cdots & (2N-2) \end{bmatrix} \tag{11}$$

to the modes of $\mathcal{V}$, that is, $\tilde{\mathcal{V}}(i_1,\ldots,i_{2N-2}) = \mathcal{V}(\pi_n(i_1),\ldots,\pi_n(i_{2N-2}))$. The vectorization of $\tilde{\mathcal{V}}$ yields $\mathbf{P}_n\mathbf{v}$.

In Julia, the above procedure can be easily implemented using the commands `reshape` and `permutedims` for reshaping and permuting the modes of a multivariate array, respectively.

```
perfect_shuffle = [ [2*k-1 for k=1:N-1]; [2*k for k=1:N-1] ]
tenshape = vec([J[setdiff([1:N],n)] I[setdiff([1:N],n)]]')
w = vec(permutedims(reshape(v,tenshape),perfect_shuffle))
```

A matrix-vector product $\mathbf{P}_n^T\mathbf{w}$ is computed in an analogous fashion. First, $\mathbf{w}$ is reshaped into a tensor $\mathcal{W}$ of size $J_1 \times \cdots \times J_{n-1} \times J_{n+1} \times \cdots \times J_N \times I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N$. After applying the inverse permutation of (11) to the modes of $\mathcal{W}$, the vectorization of this tensor yields $\mathbf{P}_n^T\mathbf{w}$.

```
tenshape=[J[setdiff([1:N],n)];I[setdiff([1:N],n)]]
vec(permutedims(reshape(w,tenshape),invperm(perfect_shuffle)))
```

3. Recompression of Hadamard products by HOSVD. In the following, we discuss two basic algorithms for recompressing the Hadamard product of two tensors of low multilinear rank. To simplify complexity considerations and the presentation, we will often consider tensors with equal mode sizes and equal multilinear ranks. All algorithms presented in this paper apply, with obvious modifications, to tensors featuring unequal sizes and ranks. The Tucker format is best suited for tensors of low order N. We will therefore restrict most of our description to tensors of order $N = 3$.

Suppose that two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I\times I\times I}$ of multilinear ranks (R, R, R) are given in Tucker format:

$$\mathcal{X} = \mathcal{F} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \mathbf{A}^{(3)}, \qquad \mathcal{Y} = \mathcal{G} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \times_3 \mathbf{B}^{(3)}, \tag{12}$$

where $\mathcal{F}, \mathcal{G} \in \mathbb{R}^{R\times R\times R}$ and $\mathbf{A}^{(n)}, \mathbf{B}^{(n)} \in \mathbb{R}^{I\times R}$, for $n = 1, 2, 3$. Then the following lemma shows that the Hadamard product also admits a representation in the Tucker format. However, the multilinear ranks get squared. While the latter fact is well known and follows directly from the corresponding result for matrices [11], the explicit expression can be found in [19, Proposition 2.2]; we include its proof for completeness.

LEMMA 3.1. *For $\mathcal{X}, \mathcal{Y}$ given by* (12)*, it holds that*

$$\mathcal{X} * \mathcal{Y} = (\mathcal{F} \otimes \mathcal{G}) \times_1 \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \times_3 \left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right). \tag{13}$$

Proof. By the definition of the Kronecker product of tensors, we have $(\mathcal{F} \times_n \mathbf{a}^T) \otimes (\mathcal{G} \times_n \mathbf{b}^T) = (\mathcal{F} \otimes \mathcal{G}) \times_n (\mathbf{a}^T \otimes \mathbf{b}^T)$ for vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^R$. Denoting the jth unit vector

by e_j, this implies that $\mathcal{Z}_{i_1,i_2,i_3}$ satisfies

$$\begin{aligned}
&(\mathcal{X} * \mathcal{Y})_{i_1,i_2,i_3} \\
&= (\mathcal{X} \times_1 e_{i_1}^T \times_2 e_{i_2}^T \times_3 e_{i_3}^T)(\mathcal{Y} \times_1 e_{i_1}^T \times_2 e_{i_2}^T \times_3 e_{i_3}^T) \\
&= \big(\mathcal{F} \times_1 e_{i_1}^T \mathbf{A}^{(1)} \times_2 e_{i_2}^T \mathbf{A}^{(2)} \times_3 e_{i_3}^T \mathbf{A}^{(3)}\big)\big(\mathcal{G} \times_1 e_{i_1}^T \mathbf{B}^{(1)} \times_2 e_{i_2}^T \mathbf{B}^{(2)} \times_3 e_{i_3}^T \mathbf{B}^{(3)}\big) \\
&= (\mathcal{F} \otimes \mathcal{G}) \times_1 (e_{i_1}^T \mathbf{A}^{(1)} \otimes e_{i_1}^T \mathbf{B}^{(1)}) \times_2 (e_{i_2}^T \mathbf{A}^{(2)} \otimes e_{i_2}^T \mathbf{B}^{(2)}) \times_3 (e_{i_3}^T \mathbf{A}^{(3)} \otimes e_{i_3}^T \mathbf{B}^{(3)}) \\
&= (\mathcal{F} \otimes \mathcal{G}) \times_1 e_{i_1}^T (\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}) \times_2 e_{i_2}^T (\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}) \times_3 e_{i_3}^T (\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}) \\
&= \big((\mathcal{F} \otimes \mathcal{G}) \times_1 \big(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\big) \times_2 \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big) \times_3 \big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\big)_{i_1,i_2,i_3},
\end{aligned}$$

which shows the desired result. ∎

3.1. Higher order singular value decomposition (HOSVD). The (truncated) HOSVD [5] is a well-established approach to obtain a quasi-optimal approximation of low multilinear rank to a given tensor $\mathcal{X}$. It chooses the nth factor matrix $\mathbf{A}^{(n)}$ of the approximation to contain the leading left singular vectors of the n-mode matricization and then chooses the core $\mathcal{F}$ optimally by projecting $\mathcal{X}$. The procedure is summarized in Algorithm 1.

Algorithm 1 HOSVD for computing low multilinear rank approximation to tensor $\mathcal{X}$.

1: **for** $n = 1, \ldots, N$ **do**
2: $\quad \mathbf{A}^{(n)} \leftarrow R_n$ leading left singular vectors of $\mathbf{X}_{(n)}$
3: **end for**
4: $\mathcal{F} \leftarrow \mathcal{X} \times_1 \mathbf{A}^{(1)^T} \times_2 \mathbf{A}^{(2)^T} \times_3 \cdots \times_N \mathbf{A}^{(N)^T}$
5: **return** $\mathcal{F}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}$

For $N = 3$, Algorithm 1 requires $\mathcal{O}(I^4)$ operations for computing the left singular vectors of the $I \times I^2$ matrix $\mathbf{X}_{(n)}$ via the SVD and, additionally, $\mathcal{O}(RI^3)$ operations for forming $\mathcal{F}$. The complexity of the first step can be reduced significantly when $R \ll I$ by using iterative methods instead of the SVD in the first step. For example, when using the randomized algorithm described in section 4.2 to approximate the leading left singular vectors of each matricization $\mathbf{X}_{(n)}$, the overall complexity of Algorithm 1 reduces to $\mathcal{O}(RI^3)$.

Remark 3.2. In Algorithm 1, number of singular vectors can either be specified or chosen adaptively based on how many singular values are above requested tolerance.

3.2. Recompression by HOSVD. The most straightforward method to (re)compress a Hadamard product is to form $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ explicitly and apply Algorithm 1. However, this becomes too expensive, in terms of operations and memory requirements, as the mode size I increases. We can combine the representation (13) of $\mathcal{Z}$ from Lemma 3.1 with a standard recompression technique to reduce the cost when $R^2 < I$. For this purpose, the columns of the factor matrices in (13) are first orthonormalized by QR decompositions. Then the HOSVD of the correspondingly updated core tensor can be turned into an HOSVD of $\mathcal{Z}$. The resulting procedure is described in Algorithm 2.

Assuming that all multilinear ranks appearing in Algorithm 2 equal R and $R^2 < I$, the cost of this algorithm is as follows for $N = 3$. Forming the $I \times R^2$ matrices $\hat{\mathbf{C}}^{(n)}$ and computing their QR decompositions requires $\mathcal{O}(R^4 I)$ operations. Forming $\hat{\mathcal{H}}$ and computing its HOSVD requires $\mathcal{O}(R^6)$ memory and $\mathcal{O}(R^8)$ operations when using a

Algorithm 2 HOSVD of tensor $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ with structure (13).

for $n = 1, \dots, N$ **do**
  $\hat{\mathbf{C}}^{(n)} \leftarrow \mathbf{A}^{(n)} \odot^T \mathbf{B}^{(n)}$.
  Compute QR decomposition $\hat{\mathbf{C}}^{(n)} = \mathbf{Q}^{(n)}\mathbf{R}^{(n)}$.
end for
$\hat{\mathcal{H}} \leftarrow (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{R}^{(1)} \times_2 \cdots \times_N \mathbf{R}^{(N)}$
Apply Algorithm 1: $(\mathcal{H}, \tilde{\mathbf{C}}^{(1)}, \dots, \tilde{\mathbf{C}}^{(N)}) = \mathrm{HOSVD}(\hat{\mathcal{H}})$
for $n = 1, \dots, N$ **do**
  $\mathbf{C}^{(n)} \leftarrow \mathbf{Q}^{(n)}\tilde{\mathbf{C}}^{(n)}$
end for
return $\mathcal{H}, \mathbf{C}^{(1)}, \dots, \mathbf{C}^{(N)}$

randomized algorithm, as discussed in section 3.1. The computation of the factor matrices $\mathbf{C}^{(n)}$ requires $\mathcal{O}(R^3 I)$ operations. This $\mathcal{O}(R^8 + R^4 I)$ complexity makes Algorithm 2 very expensive except when R is small.

Remark 3.3. As discussed in [18] for a related situation, one possibility to speed up Algorithm 2 is to prune parts that are known to be sufficiently small a priori. Because of (10), the R^2 singular values of $(\mathcal{F} \otimes \mathcal{G})_{(n)}$ are given by all pairwise products of the singular values of $\mathbf{F}_{(n)}$ and $\mathbf{G}_{(n)}$. In turn, a significant fraction of the singular values of $(\mathcal{F} \otimes \mathcal{G})_{(n)}$ will be very small, which allows one to prune the corresponding rows of $(\mathcal{F} \otimes \mathcal{G})_{(n)}$ and columns of $\mathbf{A}^{(n)} \odot^T \mathbf{B}^{(n)}$ before performing any computation. To illustrate the potential of this technique, suppose that the singular values of $\mathbf{F}_{(n)}$ and $\mathbf{G}_{(n)}$ are given by $e^{-\alpha(j-1)}$ for some $\alpha > 0$ and $j = 1, \dots, R$. Then the R^2 singular values of $(\mathcal{F} \otimes \mathcal{G})_{(n)}$ are given by

$$e^{-\alpha(j-1)} e^{-\alpha(k-1)} = e^{-\alpha(j+k-2)}, \quad j, k = 1, \dots, R.$$

Thus, the number of singular values not smaller than $e^{-\alpha(R-1)}$ is given by

$$\mathrm{card}\{(j,k) : 1 \le j \le R, 1 \le k \le R, j + k \ge R\} = R + (R-1) + \cdots + 1 = \frac{R(R+1)}{2}.$$

This allows one to discard $R(R-1)/2$ columns a priori, which brings an improvement but does not reduce the order of complexity of Algorithm 2.

4. Algorithms for low-rank approximation. The HOSVD, Algorithm 1, requires approximating a dominant R-dimensional subspace for each matricization $\mathbf{X}_{(n)}$. Equivalently, one can consider the Gramian $\mathbf{A} = \mathbf{X}_{(n)}\mathbf{X}_{(n)}^T$ and aim at a low-rank approximation of the form

$$\mathbf{A} \approx \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \qquad \mathbf{\Lambda} \approx \mathbf{U}^T\mathbf{A}\mathbf{U}, \tag{14}$$

where the matrix $\mathbf{U} \in \mathbb{R}^{I \times R}$ is a basis of the desired subspace. This section describes variants of two popular algorithms for performing the approximation (14) using matrix-vector multiplications with $\mathbf{A}$.

4.1. Lanczos algorithm for low-rank approximation. Simon and Zha [28] explained how Lanczos bidiagonalization and tridiagonalization can be used to approximate general and symmetric matrices, respectively. After k steps of Lanczos tridiagonalization applied to $\mathbf{A} \in \mathbb{R}^{I \times I}$ (see Algorithm 3), one obtains a Lanczos

decomposition of the form

$$\mathbf{AQ} = \mathbf{QT} + \beta_k \mathbf{q}_{k+1} \mathbf{e}_k^T, \tag{15}$$

where $\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 & \cdots & \mathbf{q}_k \end{bmatrix}$ has orthonormal columns, $\mathbf{e}_k \in \mathbb{R}^k$ denotes the kth unit vector, and

$$\mathbf{T} = \mathbf{Q}^T \mathbf{A} \mathbf{Q} = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-2} & \alpha_{k-1} & \beta_{k-1} \\ & & & \beta_{k-1} & \alpha_k \end{bmatrix}.$$

Algorithm 3 Lanczos tridiagonalization with full orthogonalization.

procedure LANCZOS_TRIDIAG(tol $= 10^{-8}$, maxit $= 1000$)
  Choose random vector $\mathbf{v}$ and set $\mathbf{q} = \mathbf{v}/\|\mathbf{v}\|$, $\mathbf{Q} = \begin{bmatrix} \mathbf{q} \end{bmatrix}$
  for $k = 1, 2, \ldots,$ maxit **do**
    $\mathbf{r} = \mathbf{A}\mathbf{q}$
    $\alpha_k = \mathbf{q}^T \mathbf{r}$
    $\mathbf{r} = \mathbf{r} - \alpha_k \mathbf{q}$
    Orthogonalize $\mathbf{r}$ versus the columns of $\mathbf{Q}$, reorthogonalize if needed
    Set $\beta_k = \|\mathbf{r}\|$ and compute ω_k according to (16).
    if $\omega_k <$ tol **then** quit
    $\mathbf{q} = \mathbf{r}/\beta_k$
    $\mathbf{Q} = \begin{bmatrix} \mathbf{Q} & \mathbf{q} \end{bmatrix}$
  end for
  $\mathbf{T} = \text{tridiag}((\alpha_1, \ldots, \alpha_k), (\beta_1, \ldots, \beta_{k-1}))$
end procedure

The relation (15) implies

$$\|\mathbf{A} - \mathbf{QTQ}^T\|_F^2 = \|\mathbf{A}\|_F^2 - \sum_{j=1}^{k-1} (\alpha_j^2 + 2\beta_j^2) - \alpha_k^2 =: \omega_k^2; \tag{16}$$

see [2] for details. This formula potentially suffers from cancellation and one may instead use the heuristic criterion $\beta_k <$ tol for stopping Algorithm 3.

Given the output of Algorithm 3, one computes a spectral decomposition of $\mathbf{T}$ and performs a low-rank approximation

$$\mathbf{T} \approx \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T, \qquad \mathbf{\Lambda} \in \mathbb{R}^{R \times R}, \quad \mathbf{V} \in \mathbb{R}^{k \times R},$$

where R is chosen such that $\|\mathbf{T} - \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T\|_F \le \mathsf{tol}$. Setting $\mathbf{U} = \mathbf{QV}$, we obtain

$$\|\mathbf{A} - \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\|_F \le 2\,\mathsf{tol}.$$

Numerical experiments from [2] indicate that $\mathcal{O}(R)$ iterations of Algorithm 3 are sufficient to produce the desired accuracy, independent of the singular value distribution of $\mathbf{A}$, but we are not aware of a formal proof of this statement.

4.2. Randomized algorithm for low-rank approximation. Randomized algorithms [13] offer a simple and powerful alternative to the Lanczos algorithm. If $\mathbf{A}$ had exactly rank R, one could choose a random $I \times R$ matrix $\mathbf{\Omega}$ and, with probability one, the columns of $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ would span the range of $\mathbf{A}$. For the more realistic case of a matrix $\mathbf{A}$ that can be well approximated by a rank-R matrix we choose a random $I \times (R+p)$ matrix $\mathbf{\Omega}$ where p is a small oversampling parameter, say $p = 10$. Let $\mathbf{Q} \in \mathbb{R}^{I \times (R+p)}$ denote the orthogonal factor from the QR decomposition of $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$. Then, with high probability,

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{A}$$

holds with an error not much larger than the best rank-R approximation error; see [13] for the precise mathematical statement. This procedure is summarized in Algorithm 4.

Algorithm 4 Approximate range of fixed dimension.

1: **procedure** FIXED_RAND_RANGE(R, $p = 10$)
2: Draw an $I \times (R+p)$ Gaussian random matrix $\mathbf{\Omega}$.
3: Form $I \times (R+p)$ matrix $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$
4: Compute QR decomposition $\mathbf{Y} = \mathbf{Q}\mathbf{R}$ and return $\mathbf{Q}$.
5: **end procedure**

Usually, the target rank R is not known and we need to modify Algorithm 4 such that the size of $\mathbf{Q}$ is chosen adaptively until

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}\|_2 \leq \mathsf{tol} \tag{17}$$

for a prescribed tolerance tol is satisfied. Checking this criterion is too expensive and we therefore test it with a sequence $\{\mathbf{w}^{(i)} : i = 1, 2, \ldots, q\}$ of standard Gaussian vectors for a small integer q. By a result from [13, Lemma 4.1], the relation

$$\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{A}\|_2 \leq 10\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,q} \|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{A}\mathbf{w}^{(i)}\|_2,$$

with probability $1 - 10^{-q}$. Thus, if

$$\|(\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{A}\mathbf{w}^{(i)}\|_2 \leq \sqrt{\frac{\pi}{2}}\frac{\mathsf{tol}}{10}$$

holds for $i = 1, \ldots, q$, we have found, with high probability, an approximation $\mathbf{Q}$ satisfying (17). Otherwise, we include $\mathbf{A}\mathbf{w}^{(i)}$ into $\mathbf{Q}$ and repeat the procedure. The resulting procedure is summarized in Algorithm 5.

After $\mathbf{Q}$ has been computed by Algorithm 4 or Algorithm 5, the matrix $\mathbf{U} \in \mathbb{R}^{I \times R}$ is computed via a spectral decomposition of $\mathbf{Q}^T\mathbf{A}\mathbf{Q}$, as explained in section 4.1. Overall, the complexity for obtaining $\mathbf{U}$ is $\mathcal{O}(R)$ matrix-vector multiplications with $\mathbf{A}$ and, additionally, $\mathcal{O}(IR^2)$ operations.

Remark 4.1. In principle, one can apply the randomized algorithms discussed in this section directly to $\mathbf{A} = \mathbf{X}_{(n)}$ in order to approximate its range, instead of proceeding via the (symmetric) Gramian $\mathbf{X}_{(n)}\mathbf{X}_{(n)}^T$, as suggested in the beginning of the section. The obvious disadvantage is that $\mathbf{A}$ now has I^2 columns, which requires one to use random vectors of length I^2 and the involved matrix-vector products become expensive. As we will see in section 5.2, one can greatly reduce the cost when using rank-1 vectors for the situation at hand. In this variant of randomized algorithms one

Algorithm 5 Approximate range with adaptively chosen dimension.

1: **procedure** ADAPTIVE_RAND_RANGE($\mathsf{tol} = 10^{-8}$, $q = 10$)
2: Draw standard Gaussian vectors $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(q)}$ of length I
3: $\begin{bmatrix}\mathbf{y}^{(1)} & \mathbf{y}^{(2)} & \cdots & \mathbf{y}^{(q)}\end{bmatrix} = \mathbf{A}\begin{bmatrix}\mathbf{w}^{(1)} & \mathbf{w}^{(2)} & \cdots & \mathbf{w}^{(q)}\end{bmatrix}$
4: Set $j = 0$ and $\mathbf{Q} = \begin{bmatrix}\ \end{bmatrix}$ (empty $I \times 0$ matrix)
5: **while** $\max\{\|\mathbf{y}^{(j+1)}\|, \|\mathbf{y}^{(j+2)}\|, \ldots, \|\mathbf{y}^{(j+q)}\|\} > \mathsf{tol}/(10\sqrt{2/\pi})$ **do**
6: $j = j + 1$
7: $\mathbf{y}^{(j)} \leftarrow (\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{y}^{(j)}$
8: $\mathbf{q} = \mathbf{y}^{(j)}/\|\mathbf{y}^{(j)}\|, \mathbf{Q} = \begin{bmatrix}\mathbf{Q} & \mathbf{q}\end{bmatrix}$
9: Draw standard Gaussian vector $\mathbf{w}^{(j+q)}$ of length I
10: $\mathbf{y}^{(j+q)} = (\mathbf{I} - \mathbf{Q}\mathbf{Q}^T)\mathbf{A}\mathbf{w}^{(j+q)}$
11: **for** $i = j + 1, j + 2, \ldots, j + q - 1$ **do**
12: $\mathbf{y}^{(i)} \leftarrow \mathbf{y}^{(i)} - \mathbf{q}\langle \mathbf{q}, \mathbf{y}^{(i)}\rangle$
13: **end for**
14: **end while**
15: **end procedure**

proceeds as follows. Instead of generating $R + p$ unstructured random vectors of size I^2, we first generate standard Gaussian vectors $\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \in \mathbb{R}^I$, for $i = 1, \ldots, R + p$, and then multiply $\mathbf{A}$ by $\mathbf{w}^{(i)} = \mathbf{x}^{(i)} \otimes \mathbf{y}^{(i)}$. After that we continue with the procedure described in Algorithms 4 and 5.

We would like to emphasize that the use of such rank-1 random vectors is not covered by the existing analyses of randomized algorithms. It is by no means trivial to extend the results from [13] to this random model. Nevertheless, our numerical experiments indicate that they work, perhaps surprisingly, well; see, in particular, Figure 3.

4.3. Lanczos algorithm versus randomized algorithm. According to experiments from [2], the Lanczos and randomized algorithms described above are often quite similar in terms of the number of matrix-vector multiplications needed to attain a certain accuracy. For slow singular value decays, randomized algorithms tend to require slightly more iterations. On the other hand, randomized algorithms benefit in situations where the multiplication of $\mathbf{A}$ with a block of vectors is faster than multiplying $\mathbf{A}$ with each individual vector. This is the case, for example, when $\mathbf{A}$ is explicitly available. We therefore use randomized algorithms in such a situation. When $\mathbf{A}$ is only available via matrix-vector products, we use the Lanczos algorithm.

An alternative, which we have not explored, is to combine randomized and Lanczos algorithms into a block Lanczos algorithm; see [20] for an example.

4.4. HOSVD based on approximate ranges. Algorithm 6 describes a variant of the HOSVD that is based on approximations of the range of $\mathbf{X}_{(n)}$ instead of leading left singular vectors. When using a randomized algorithm for obtaining this approximation, the required oversampling will lead to an orthonormal basis $\tilde{\mathbf{A}}^{(n)}$ that is unnecessarily large. To mitigate this effect, the resulting core tensor is recompressed in line 5.

In the following, we perform an error analysis of Algorithm 6, similar to the analysis in [5] for the HOSVD. Given a tolerance ε, it is assumed that

$$\|(\mathbf{I} - \Pi_n)\mathbf{X}_{(n)}\|_F \leq \varepsilon, \tag{18}$$

Algorithm 6 HOSVD based on approximate ranges of matricizations.

1: **for** $n = 1, \dots, N$ **do**
2: $\quad \tilde{\mathbf{A}}^{(n)} \leftarrow$ orthonormal basis for approximation of range of $\mathbf{X}_{(n)}$
3: **end for**
4: $\tilde{\boldsymbol{\mathcal{F}}} \leftarrow \boldsymbol{\mathcal{X}} \times_1 \tilde{\mathbf{A}}^{(1)^T} \times_2 \tilde{\mathbf{A}}^{(2)^T} \times_3 \cdots \times_N \tilde{\mathbf{A}}^{(N)^T}$
5: Apply Algorithm 1: $(\boldsymbol{\mathcal{F}}, \mathbf{B}^{(1)}, \dots, \mathbf{B}^{(N)}) = \mathrm{HOSVD}(\tilde{\boldsymbol{\mathcal{F}}})$
6: **for** $n = 1, \dots, N$ **do**
7: $\quad \mathbf{A}^{(n)} \leftarrow \tilde{\mathbf{A}}^{(n)} \mathbf{B}^{(n)}$
8: **end for**
9: **return** $\boldsymbol{\mathcal{F}}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)}$

with the orthogonal projector $\Pi_n = \tilde{\mathbf{A}}^{(n)} \tilde{\mathbf{A}}^{(n)}$. Using Algorithm 5, this can be achieved by choosing $\mathsf{tol} = \varepsilon/\sqrt{I}$; see (17). For the Lanczos algorithm, the Frobenius norm can be controlled directly; see, e.g., [2]. By writing

$$\begin{aligned}\boldsymbol{\mathcal{X}} &= \boldsymbol{\mathcal{X}} \times_1 \Pi_1 + \boldsymbol{\mathcal{X}} \times_1 (\mathbf{I} - \Pi_1) \\ &= \boldsymbol{\mathcal{X}} \times_1 \Pi_1 \times_2 \Pi_2 + \boldsymbol{\mathcal{X}} \times_1 \Pi_1 \times_2 (\mathbf{I} - \Pi_2) + \boldsymbol{\mathcal{X}} \times_1 (\mathbf{I} - \Pi_1) \\ &= \cdots = \boldsymbol{\mathcal{X}} \times_1 \Pi_1 \cdots \times_N \Pi_N + \sum_{n=1}^{N} \boldsymbol{\mathcal{X}} \times_1 \Pi_1 \cdots \times_{n-1} \Pi_{n-1} \times_n (\mathbf{I} - \Pi_n),\end{aligned}$$

and letting

$$\tilde{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{X}} \times_1 \Pi_1 \cdots \times_N \Pi_N = \tilde{\boldsymbol{\mathcal{F}}} \times_1 \tilde{\mathbf{A}}^{(1)} \cdots \times_N \tilde{\mathbf{A}}^{(N)},$$

we obtain

$$\begin{aligned}\|\boldsymbol{\mathcal{X}} - \tilde{\boldsymbol{\mathcal{X}}}\|_F &\le \sum_{n=1}^{N} \|\boldsymbol{\mathcal{X}} \times_1 \Pi_1 \cdots \times_{n-1} \Pi_{n-1} \times_n (\mathbf{I} - \Pi_n)\|_F \\ &\le \sum_{n=1}^{N} \|\boldsymbol{\mathcal{X}} \times_n (\mathbf{I} - \Pi_n)\|_F \le N\varepsilon.\end{aligned}$$

By requiring the 2-norm of the truncated singular values in each matricization of $\boldsymbol{\mathcal{F}}$ to be not larger than ε, the HOSVD performed in line 5 satisfies an analogous error estimate:

$$\|\tilde{\boldsymbol{\mathcal{F}}} - \boldsymbol{\mathcal{F}} \times_1 \mathbf{B}^{(1)} \cdots \times_n \mathbf{B}^{(N)}\|_F \le N\varepsilon;$$

see [5]. In turn, the output of Algorithm 6 satisfies

$$\begin{aligned}\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{F}} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}\|_F &\le \|\boldsymbol{\mathcal{X}} - \tilde{\boldsymbol{\mathcal{X}}}\|_F + \|\tilde{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{F}} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}\|_F \\ &\le \|\boldsymbol{\mathcal{X}} - \tilde{\boldsymbol{\mathcal{X}}}\|_F + \|\tilde{\boldsymbol{\mathcal{F}}} - \boldsymbol{\mathcal{F}} \times_1 \mathbf{B}^{(1)} \cdots \times_n \mathbf{B}^{(N)}\|_F \le 2N\varepsilon.\end{aligned}$$

This allows one to prescribe the accuracy of Algorithm 6.

5. Exploiting structure in the recompression of Hadamard products. The complexity of the basic algorithms presented in section 3 renders them unattractive in situations where the mode sizes I are fairly large and the involved multilinear ranks R are not small. In this section, we develop tools that exploit the structure (13) of $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} * \boldsymbol{\mathcal{Y}}$. In section 6, these tools will be combined with iterative methods in order to yield faster algorithms for recompressing $\boldsymbol{\mathcal{Z}}$.

5.1. Fast matrix-vector multiplication with $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$. Given the n-mode matricization $\mathbf{Z}_{(n)}$ of $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$, we develop a fast algorithm for multiplying $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$ with a general vector $\mathbf{v} \in \mathbb{R}^I$. This operation will be used in the Lanczos method (see section 4.1) for approximating the left singular vectors of $\mathbf{Z}_{(n)}$.

In the following, we assume $n = 1$ without loss of generality. From (13), (8), and (10), we obtain

$$\begin{aligned}\mathbf{Z}_{(1)} &= \left[(\mathcal{F} \otimes \mathcal{G}) \times_1 \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \times_3 \left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \right]_{(1)} \\ &= \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) (\mathcal{F} \otimes \mathcal{G})_{(1)} \left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right)\right]^T \\ &= \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \left(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\right) \mathbf{P}_1 \left[\left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right) \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right)\right].\end{aligned}$$

We perform the matrix-vector multiplication $\mathbf{w} = \mathbf{Z}_{(1)}\mathbf{Z}_{(1)}^T\mathbf{v}$ in four steps.

Step 1. $\mathbf{w}_1 = \left(\mathbf{A}^{(1)^T} \odot \mathbf{B}^{(1)^T}\right)\mathbf{v}$.
Step 2. $\mathbf{w}_2 = \mathbf{P}_1^T\left(\mathbf{F}_{(1)}^T \otimes \mathbf{G}_{(1)}^T\right)\mathbf{w}_1$.
Step 3. $\mathbf{w}_3 = \left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \right]\mathbf{w}_2$,
$\mathbf{w}_4 = \left[\left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right) \otimes \left(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\right)\right]\mathbf{w}_3$.
Step 4. $\mathbf{w} = \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right)\left(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\right)\mathbf{P}_1\mathbf{w}_4$.

In the following, we discuss an efficient implementation for each of these steps.

Step 1. Using (6), we obtain

$$\mathbf{w}_1 = \left(\mathbf{A}^{(1)^T} \odot \mathbf{B}^{(1)^T}\right)\mathbf{v} = \operatorname{vec}\left(\mathbf{B}^{(1)^T} \operatorname{diag}(\mathbf{v})\mathbf{A}^{(1)}\right).$$

Equivalently, by reshaping $\mathbf{w}_1$ into an $R \times R$ matrix $\mathbf{W}_1$,

$$\mathbf{W}_1 = \underbrace{\mathbf{B}^{(1)^T}}_{R \times I}\left(\underbrace{\operatorname{diag}(\mathbf{v})\mathbf{A}^{(1)}}_{I \times R}\right).$$

This matrix multiplication requires $\mathcal{O}(R^2 I)$ operations.

Step 2. We have

$$\mathbf{w}_2 = \mathbf{P}_1^T\left(\mathbf{F}_{(1)}^T \otimes \mathbf{G}_{(1)}^T\right)\mathbf{w}_1 = \mathbf{P}_1^T \operatorname{vec}\Bigg(\underbrace{\mathbf{G}_{(1)}^T}_{R^2 \times R}\underbrace{\mathbf{W}_1}_{R \times R}\underbrace{\mathbf{F}_{(1)}}_{R \times R^2}\Bigg).$$

The matrix multiplication requires $\mathcal{O}(R^5)$ operations, while the cost for applying $\mathbf{P}_1^T$—using the techniques described in section 2.4 —is negligible.

Step 3. *Variant A.* We present two variants for performing Step 3 and, as will be explained below, the choice of the variant depends on the relation between I and R. In Variant A, we first reshape $\mathbf{w}_2$ into a matrix $\mathbf{W}_2$ of size $R^2 \times R^2$ and obtain

$$\mathbf{w}_3 = \left[\left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right) \otimes \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right)\right], \quad \mathbf{w}_2 = \operatorname{vec}\left[\left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right)\mathbf{W}_2\left(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\right)\right].$$

Let $\mathbf{c}_j \in \mathbb{R}^{R^2}$ denote the jth column of $\mathbf{W}_2$ for $j = 1, \ldots, R^2$, which is reshaped into the $R \times R$ matrix $\mathbf{C}_j$. Then the jth column of $\mathbf{W}_3' = \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right)\mathbf{W}_2$ is a vector of length I given by

$$\left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right)\mathbf{c}_j = \operatorname{diag}\left(\underbrace{\mathbf{B}^{(2)}}_{I \times R}\mathbf{C}_j\underbrace{\mathbf{A}^{(2)^T}}_{R \times I}\right), \tag{19}$$

where we have used (7). Exploiting that only the diagonal entries are needed, this requires $\mathcal{O}(R^2 I)$ operations for each $\mathbf{c}_j$ and hence $\mathcal{O}(R^4 I)$ operations in total for computing $\mathbf{W}_3' \in \mathbb{R}^{I \times R^2}$.

The matrix $\mathbf{W}_3 = \mathbf{W}_3'\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big) = \big[\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\mathbf{W}_3'^T\big]^T$ is computed analogously. Letting $\mathbf{c}_j' \in \mathbb{R}^{R^2}$ denote the jth column of $\mathbf{W}_3'^T$ for $j = 1, \ldots, I$, the jth column of $\mathbf{W}_3^T$ is given by

$$\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\mathbf{c}_j' = \operatorname{diag}\big(\mathbf{B}^{(3)}\mathbf{C}_j'\mathbf{A}^{(3)^T}\big). \tag{20}$$

Therefore, the computation of $\mathbf{W}_3 \in \mathbb{R}^{I \times I}$ from $\mathbf{W}_3'$ requires $\mathcal{O}(R^2 I^2)$ operations.

We now consider the computation of

$$\begin{aligned}\mathbf{w}_4 &= \big[\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big) \otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\big]\mathbf{w}_3 \\ &= \operatorname{vec}\big[\big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\mathbf{W}_3\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\big].\end{aligned}$$

Letting $\mathbf{d}_j$ denote the jth column of $\mathbf{W}_3$ for $j = 1, \ldots, I$, the jth column of $\mathbf{W}_4' = \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\mathbf{W}_3$ is given by

$$\big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\mathbf{d}_j = \operatorname{vec}\big(\mathbf{B}^{(2)^T} \operatorname{diag}(\mathbf{d}_j)\mathbf{A}^{(2)}\big). \tag{21}$$

Therefore, the computation of $\mathbf{W}_4' \in \mathbb{R}^{R^2 \times I}$ requires $\mathcal{O}(R^2 I^2)$ operations.

Analogously, the jth column of $\mathbf{W}_4^T$ is computed from the jth column $\mathbf{d}_j'$ of $\mathbf{W}_4'^T$ for $j = 1, \ldots, R^2$ via

$$\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big)\mathbf{d}_j' = \operatorname{vec}\big(\mathbf{B}^{(3)^T} \operatorname{diag}(\mathbf{d}_j')\mathbf{A}^{(3)}\big).$$

This requires $\mathcal{O}(R^4 I)$ operations for all columns of $\mathbf{W}_4$.

Overall, Variant A of Step 3 has a computational complexity of $\mathcal{O}(R^2 I^2 + R^4 I)$.

Step 3. *Variant B.* For tensors of large mode sizes, the quadratic growth with respect to I in the complexity of Variant A is undesirable. We can avoid this by utilizing (3):

$$\begin{aligned}\mathbf{w}_4 &= \big[\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big) \otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\big]\big[\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big) \otimes \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\big]\mathbf{w}_2 \\ &= \big[\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big)\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big) \otimes \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\big]\mathbf{w}_2 \\ &= \operatorname{vec}\big[\big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\mathbf{W}_2\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big)\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)\big].\end{aligned}$$

As in Variant A, we first perform $\mathbf{W}_3' = \big(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\big)\mathbf{W}_2$, which requires $\mathcal{O}(R^4 I)$ operations. Then, analogous to (21), we compute the $R^2 \times R^2$ matrix $\mathbf{W}_3 = \big(\mathbf{A}^{(2)^T} \odot \mathbf{B}^{(2)^T}\big)\mathbf{W}_3'$ within $\mathcal{O}(R^4 I)$ operations. Analogous to (20), the computation of the $R^2 \times I$ matrix $\mathbf{W}_4' = \mathbf{W}_3\big(\mathbf{A}^{(3)^T} \odot \mathbf{B}^{(3)^T}\big)$ requires again $\mathcal{O}(R^4 I)$ operations. Finally, as in Variant A, we compute $\mathbf{W}_4 = \mathbf{W}_4'\big(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\big)$ within $\mathcal{O}(R^4 I)$ operations.

In total, Variant B has complexity $\mathcal{O}(R^4 I)$ and thus avoids the term $R^2 I^2$ in the complexity of Variant A. Let us compare the sizes of the intermediate matrices created by the two variants:

	$\mathbf{W}_3'$	$\mathbf{W}_3$	$\mathbf{W}_4'$	$\mathbf{W}_4$
Variant A	$I \times R^2$	$I \times I$	$R^2 \times I$	$R^2 \times R^2$
Variant B	$I \times R^2$	$R^2 \times R^2$	$R^2 \times I$	$R^2 \times R^2$

In terms of storage requirements it could be preferable to use Variant A when $I < R^2$. In this situation, Variant A also turns out to be faster than Variant B. This can be clearly seen in Figure 1, which shows that a matrix-vector multiplication based on Variant A becomes faster for $R \geq 35 \approx \sqrt{1000} = \sqrt{I}$; see section 7.1 for a description of the computational environment.

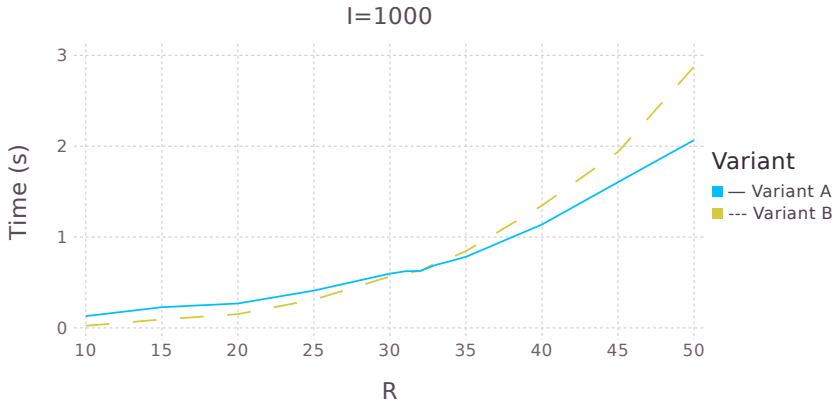

FIG. 1. *Execution time in seconds for matrix-vector multiplication with* $\mathbf{Z}_{(1)}\mathbf{Z}_{(1)}^T$ *for random tensors* $\mathcal{X}$ *and* $\mathcal{Y}$ *with* $I = 1000$ *and ranks* $R = 10, 15, 20, 25, 30, 31, 32, 33, 34, 35, 40, 45, 50$ *using either Variant A or Variant B in Step* 3.

To conclude, we use Variant A for $I < R^2$ and Variant B otherwise.

Step 4. In the final step, we calculate $\mathbf{w} = \big(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\big)\big(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\big)\mathbf{P}_1\mathbf{w}_4$. The permutation matrix $\mathbf{P}_1$ is applied as explained in section 2.4. The Kronecker product $\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}$ is applied with two matrix multiplications, requiring $\mathcal{O}(R^5)$ operations. The matrix $\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}$ is applied as in (19), requiring $\mathcal{O}(R^2 I)$ operations.

Summary. When Variant B in Step 3 is used, the matrix-vector multiplication $\mathbf{w} = \mathbf{Z}_{(1)}\mathbf{Z}_{(1)}^T\mathbf{v}$ can be performed within $\mathcal{O}(R^4 I + R^5)$ operations, which compares favorably when $I > R^2$ with the $\mathcal{O}(I^3)$ operations needed by applying $\mathbf{Z}_{(1)}$ and $\mathbf{Z}_{(1)}^T$ without exploiting structure.

5.2. Fast multiplication of $\mathbf{Z}_{(n)}$ with a rank-1 vector. Given a vector $\mathbf{w} = \mathbf{x} \otimes \mathbf{y}$ with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^I$, we now discuss the fast multiplication of $\mathbf{Z}_{(n)}$ with $\mathbf{w}$. This operation will be used in a randomized algorithm for estimating the range of $\mathbf{Z}_{(n)}$. Again, we assume $n = 1$ w.l.o.g.

In a precomputation step, we construct $\mathbf{D}^{(m)} = \mathbf{A}^{(m)} \odot^T \mathbf{B}^{(m)}$ for $m = 1, 2, 3$. This gives

$$\begin{aligned}\mathbf{Z}_{(1)}\mathbf{w} &= \mathbf{D}^{(1)}(\mathcal{F} \otimes \mathcal{G})_{(1)}\big(\mathbf{D}^{(3)} \otimes \mathbf{D}^{(2)}\big)^T(\mathbf{x} \otimes \mathbf{y}) \\ &= \mathbf{D}^{(1)}(\mathcal{F} \otimes \mathcal{G})_{(1)}\big({\mathbf{D}^{(3)}}^T\mathbf{x} \otimes {\mathbf{D}^{(2)}}^T\mathbf{y}\big).\end{aligned}$$

We compute $\tilde{\mathbf{x}} = {\mathbf{D}^{(3)}}^T\mathbf{x} \in \mathbb{R}^{R^2}$ and $\tilde{\mathbf{y}} = {\mathbf{D}^{(2)}}^T\mathbf{y} \in \mathbb{R}^{R^2}$ in $\mathcal{O}(R^2 I)$ operations and then form

$$\mathbf{z}_1 = \tilde{\mathbf{x}} \otimes \tilde{\mathbf{y}}$$

in $\mathcal{O}(R^4)$ operations. To perform the product

$$(\mathcal{F} \otimes \mathcal{G})_{(1)} \mathbf{z}_1 = \left(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\right) \mathbf{P}_1 \mathbf{z}_1, \tag{22}$$

we first perform the permutation $\mathbf{z}_2 = \mathbf{P}_1 \mathbf{z}_1$ as discussed above. The multiplication $\mathbf{z}_3 = \left(\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}\right) \mathbf{z}_2$ requires $\mathcal{O}(R^5)$ operations when using (5). The last step consists of the product $\mathbf{D}^{(1)} \mathbf{z}_3$, which requires $\mathcal{O}(R^2 I)$ operations.

In summary, the matrix-vector product $\mathbf{Z}_{(1)} \mathbf{w}$ can be performed in $\mathcal{O}(R^2 I + R^5)$ operations. In fact, this can be reduced further to $\mathcal{O}(R^2 I + R^4)$ by evaluating (22) more carefully, but we refrain from providing this more complicated procedure because it does not reduce the overall complexity of our algorithms.

5.3. Calculating core tensor of Hadamard product. Assuming that we have calculated factor matrices $\mathbf{C}^{(1)}, \mathbf{C}^{(2)}, \mathbf{C}^{(3)}$, we now discuss the efficient computation of the core tensor

$$\mathcal{H} = \mathcal{Z} \times_1 \mathbf{C}^{(1)^T} \times_2 \mathbf{C}^{(2)^T} \times_3 \mathbf{C}^{(3)^T}$$

required in step 4 of Algorithm 1. Using (13), we have

$$\mathcal{H} = (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{C}^{(1)^T} \left(\mathbf{A}^{(1)} \odot^T \mathbf{B}^{(1)}\right) \times_2 \mathbf{C}^{(2)^T} \left(\mathbf{A}^{(2)} \odot^T \mathbf{B}^{(2)}\right) \times_3 \mathbf{C}^{(3)^T} \left(\mathbf{A}^{(3)} \odot^T \mathbf{B}^{(3)}\right).$$

To simplify the complexity considerations, we assume that the size of each matrix $\mathbf{C}^{(n)}$ is $I \times R$, matching the sizes of $\mathbf{A}^{(n)}$ and $\mathbf{B}^{(n)}$.

We first calculate

$$\mathbf{D}^{(n)} = \left(\mathbf{A}^{(n)^T} \odot \mathbf{B}^{(n)^T}\right) \mathbf{C}^{(n)},$$

for $n = 1, 2, 3$, which requires $\mathcal{O}(R^3 I)$ operations. We rewrite $\mathcal{H} = (\mathcal{F} \otimes \mathcal{G}) \times_1 \mathbf{D}^{(1)^T} \times_2 \mathbf{D}^{(2)^T} \times_3 \mathbf{D}^{(3)^T}$, in terms of the mode-1 matricization:

$$\mathbf{H}_{(1)} = \underbrace{\mathbf{D}^{(1)^T}}_{R \times R^2} \underbrace{(\mathcal{F} \otimes \mathcal{G})_{(1)}}_{R^2 \times R^4} \left(\underbrace{\mathbf{D}^{(3)} \otimes \mathbf{D}^{(2)}}_{R^4 \times R^2}\right) = \mathbf{D}^{(1)^T} (\mathbf{F}_{(1)} \otimes \mathbf{G}_{(1)}) \mathbf{P}_1 (\mathbf{D}^{(3)} \otimes \mathbf{D}^{(2)}).$$

Letting $\mathbf{d}_i \in \mathbb{R}^{R^2}$ and $\mathbf{l}_i \in \mathbb{R}^{R^4}$ denote the ith column of $\mathbf{D}^{(1)}$ and $\left(\mathbf{D}^{(1)^T} \left(\mathcal{F} \otimes \mathcal{G}\right)_{(1)}\right)^T$, respectively, we have

$$\mathbf{l}_i = \mathbf{P}_1^T \left(\mathbf{F}_{(1)}^T \otimes \mathbf{G}_{(1)}^T\right) \mathbf{d}_i.$$

Exploiting the property (5) of the Kronecker product, computing $\mathbf{l}_i$ requires $\mathcal{O}(R^5)$ operations. Similarly, computing

$$\mathbf{l}_i^T \left(\mathbf{D}^{(3)} \otimes \mathbf{D}^{(2)}\right)$$

has complexity $\mathcal{O}(R^5)$ and gives the ith row of $\mathbf{H}_{(1)}$ for $i = 1, \ldots, R$. In summary, the computation of $\mathcal{H}$ requires $\mathcal{O}(R^3 I + R^6)$ operations.

6. Algorithms. This section presents four different algorithms for recompressing $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ into the Tucker format. In preliminary numerical experiments, we have ruled out other algorithmic combinations and believe that these four algorithms represent the most meaningful combinations of the techniques discussed in this paper. For the complexity considerations, we again assume that all mode sizes equal I, all multilinear ranks equal R, and the iterative algorithms from section 4.2 require $O(R)$ matrix-vector multiplications.

HOSVD1: Algorithm 1 for full tensor $\mathcal{Z}$ + randomized. HOSVD1 creates the full tensor $\mathcal{Z}$ explicitly and applies Algorithm 1 to $\mathcal{Z}$ using the randomized algorithm from section 4.2 for approximating the factor matrices. This requires $\mathcal{O}(RI^3)$ operations and $\mathcal{O}(I^3)$ memory.

HOSVD2: Algorithm 2 + randomized. HOSVD2 is Algorithm 2, with the HOSVD of the $R^2 \times R^2 \times R^2$ core tensor $\hat{\mathcal{H}}$ performed by Algorithm 1 using again the randomized algorithm (section 4.2) for approximating the factor matrices. This requires $\mathcal{O}(R^4I + R^8)$ operations and $\mathcal{O}(R^2I + R^6)$ memory.

HOSVD3: Algorithm 1 + structure-exploiting Lanczos. HOSVD3 is a variant of Algorithm 1, where the Lanczos algorithm (section 4.1) is combined with the fast matrix-vector multiplication described in section 5.1 to obtain approximations of the factor matrices. The procedure from section 5.3 is used for calculating the core tensor. For $I < R^2$, using Variant A in Step 3 of the matrix-vector multiplication, $\mathcal{O}(R^3I^2 + R^5I + R^6)$ operations and $\mathcal{O}(I^2 + R^2I + R^4)$ memory are required. For $I \geq R^2$, using Variant B in Step 3 of the matrix-vector multiplication, $\mathcal{O}(R^5I + R^6)$ operations and $\mathcal{O}(R^2I + R^4)$ memory are required.

HOSVD4: Algorithm 1 + rank-1 randomized. HOSVD4 applies a randomized algorithm (i.e., Algorithm 4) to estimate the range of $\mathbf{Z}_{(n)} \in \mathbb{R}^{I \times I^2}$, using rank-1 vectors $\mathbf{w}^{(i)} = \mathbf{x}^{(i)} \otimes \mathbf{y}^{(i)}$, where $\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \in \mathbb{R}^I$ are randomly chosen for $i = 1, \ldots, R + p$. This allows for using the procedure described in section 5.2 when performing the matrix-vector multiplications $\mathbf{Z}_{(n)}\mathbf{w}^{(i)}$.

The orthonormal bases $\tilde{\mathbf{A}}^{(n)}$ of the estimated ranges are used to compress the size of the tensor, followed by Algorithm 1; see Algorithm 6. Again, the procedure from section 5.3 is used for calculating the core tensor $\tilde{\mathcal{F}}$ in line 4. Overall, HOSVD4 requires $\mathcal{O}(R^3I + R^6)$ operations and $\mathcal{O}(R^2I + R^4)$ memory.

Remark 6.1. The discussion above on the complexity easily generalizes to $N > 3$. HOSVD1 requires $\mathcal{O}(RI^N)$ operations and $\mathcal{O}(I^N)$ memory, HOSVD2 $\mathcal{O}(R^4I + R^{2N+2})$ operations and $\mathcal{O}(R^2I + R^{2N})$ memory, and $\mathcal{O}(R^3I + R^{2N})$ operations and $\mathcal{O}(R^2I + R^{2N-2})$ memory for HOSVD4. For HOSVD3, when $I < R^2$, using Variant A in Step 3 of matrix-vector multiplication, $\mathcal{O}(\sum_{n=1}^{N-1} I^n R^{2(N-n)+1} + R^{2N})$ operations and $\mathcal{O}(\sum_{n=1}^{N-1} I^{N-n} R^{2n-2} + R^{2N-2})$ memory are required, while when $I \geq R^2$ and Variant B is used in Step 3 of matrix-vector multiplication $\mathcal{O}(IR^{2N-1} + R^{2N})$ operations and $\mathcal{O}(IR^{2N-4} + R^{2N-2})$ memory are required.

7. Numerical experiments.

7.1. Computational environment.

We have implemented and tested the algorithms from section 6 in Julia version 0.5.2, on a PC with an Intel Core i5-3470 quadcore 3.20GHz CPU, 256KB L2 cache, 6MB L3 cache, and 4GB of RAM. Multithreading is turned on and all four cores are utilized.

7.2. Execution times for function-related tensors.

To test our algorithms, we generate function-related tensors $\mathcal{X}$, $\mathcal{Y}$ by evaluating the functions

$$f(x, y, z) = \frac{1}{x + y + z}, \quad g(x, y, z) = \frac{1}{\sqrt{x + y + z}}$$

on the grid $\{0.1, 0.2, \ldots, I/10\}$ for x, y, z. The following table reports the approximate multilinear ranks (R, R, R) of $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ obtained when discarding singular

values smaller than 10^{-8}:

	$I = 50$	$I = 100$	$I = 200$	$I = 400$
$\mathcal{X}$	11	12	15	17
$\mathcal{Y}$	11	12	15	17
$\mathcal{Z}$	12	13	15	17

Figure 2 shows the execution times obtained for the four different HOSVD algorithms from section 6 applied to the tensors $\mathcal{X}, \mathcal{Y}$ in Tucker format and targeting an accuracy of 10^{-8}. In particular, the latter implies that Algorithm 5 is used for the randomized algorithms. Except for $I = 50$, HOSVD4 outperforms all other algorithms. For example, for $I = 400$, HOSVD4 requires 0.16 seconds while HOSVD3, the next best algorithm, requires 2 seconds.

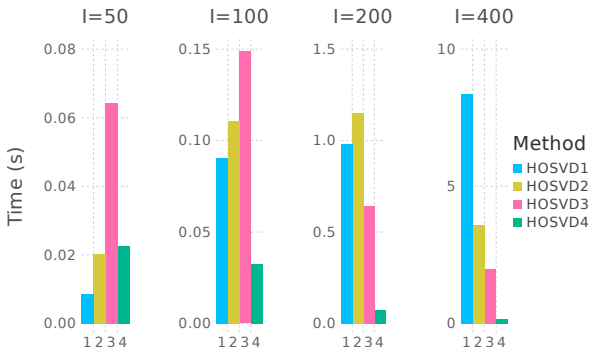

FIG. 2. *Execution times (in seconds) of HOSVD algorithms applied to Hadamard product of function-related tensors from section* 7.2.

7.3. Accuracy of HOSVD3 versus HOSVD4. We tested the accuracy of the structure-exploiting algorithms HOSVD3 and HOSVD4 in dependence of the oversampling parameter p. For HOSVD3, p refers to the $R + p$ iterations of the Lanczos method, while for HOSVD4 it refers to the number $R + p$ of columns in the random matrix Ω. We used the function-related tensors $\mathcal{X}$ and $\mathcal{Y}$ from section 7.2 with $I = 50$ and required the multilinear rank to be (R, R, R), with $R = 9$ and $R = 15$. Figure 3 shows the resulting error calculated as $\|\mathcal{X} * \mathcal{Y} - \mathcal{T}\|_F$, where $\mathcal{T}$ is the output of the corresponding algorithm. Generally, it can be said that very small values of p are sufficient for both algorithms, in the sense that the limiting accuracy determined by the choice of R is reached. HOSVD4 is more accurate in situations where the limiting accuracy is below $\mathcal{O}(10^{-8})$, because it works directly with $\mathbf{Z}_{(n)}$ instead of the Gramian $\mathbf{Z}_{(n)}\mathbf{Z}_{(n)}^T$.

7.4. Execution times for random tensors. We now choose $I \times I \times I$ tensors $\mathcal{X}$ and $\mathcal{Y}$ of prescribed multilinear rank (R, R, R) by letting all coefficients of the Tucker format contain random numbers from the standard normal distribution. We measure the time needed by our algorithms for truncating $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ back to multilinear rank (R, R, R).

Precisely, we use Algorithm 4 with oversampling parameter $p = 10$ inside algorithms HOSVD1, HOSVD2, and HOSVD4 and Algorithm 3 with maxit $= R + p$,

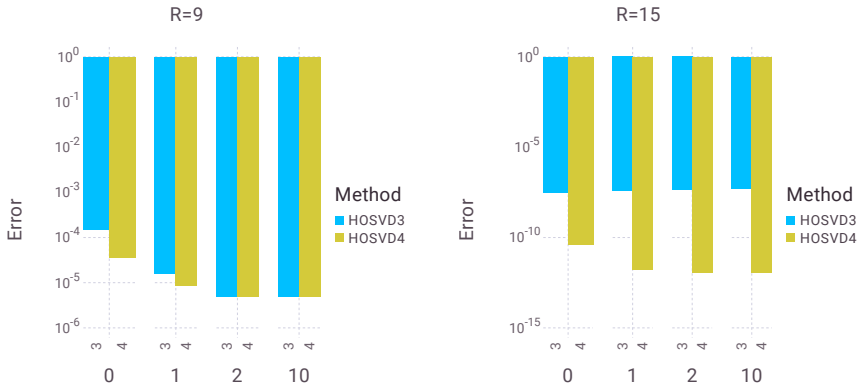

FIG. 3. *Error of HOSVD3 and HOSVD4 versus oversampling parameter* $p = 0, 2, 4, \ldots, 20$.

for $p = 10$, in the HOSVD3 algorithm. Choosing different reasonable oversampling parameter (e.g., any $p \leq 20$) does not notably change the execution times.

The times obtained for $I = 50, 100, 200, 400$ with respect to R are shown in Figure 4. As expected, the performance of HOSVD1, based on forming the full tensor $\mathcal{Z}$, depends only mildly on R. All other algorithms have a cost that is initially smaller than HOSVD1 but grows as R increases. The observed breakeven points match the theoretical breakeven points between $R = \mathcal{O}(I^{2/5})$ and $R = \mathcal{O}(I^{3/5})$ quite well. For larger I, it is impossible to store $\mathcal{Z}$ and hence Figure 5 only displays the times of HOSVD2, HOSVD3, and HOSVD4 for $I = 500, 900$. Among these three algorithms, HOSVD4 is nearly always the best.

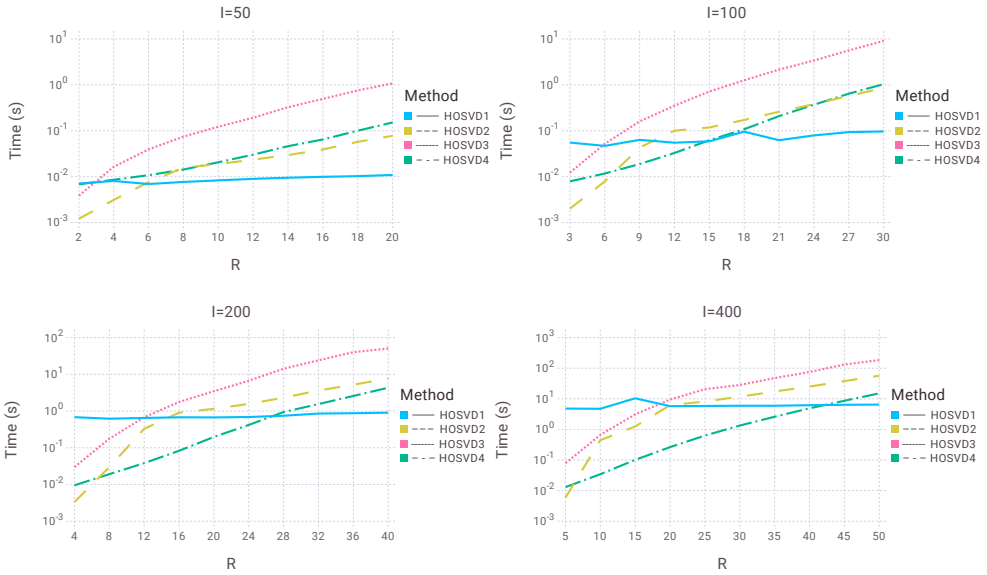

FIG. 4. *Execution times (in seconds) of HOSVD algorithms applied to the Hadamard product of random tensors of low multilinear rank from section* 7.4.

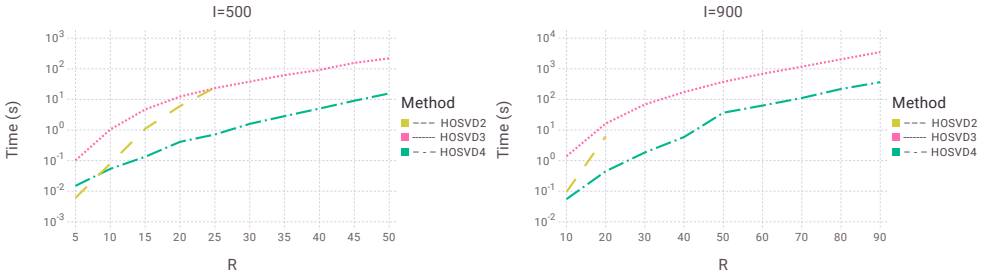

FIG. 5. *Execution times (in seconds) of HOSVD algorithms applied to the Hadamard product of random tensors of low multilinear rank from section* 7.4.

Based on these observations, we conclude the following recommendation:

- Use HOSVD1 when $\mathcal{Z}$ fits into memory *and* the involved multilinear ranks are expected to exceed $I^{3/5}$.
- In all other situations, use HOSVD4.

Finally, Figure 6 shows the performance of HOSVD4 with respect to R and I. Note that some of the displayed configurations, such as $I = 5\,000$ and $R = 90$, are intractable for any other algorithm discussed in this paper.

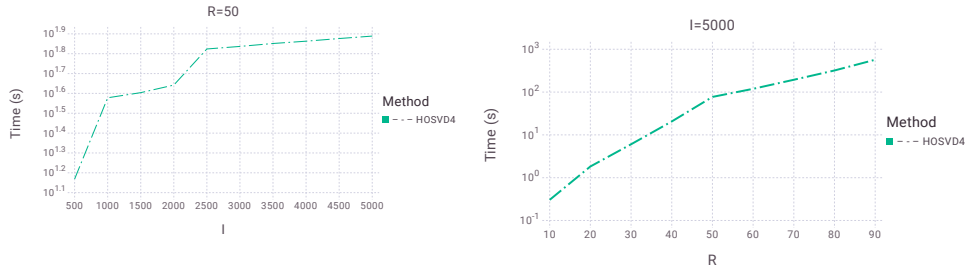

FIG. 6. *Execution times (in seconds) of HOSVD4 applied to the Hadamard product of random tensors of low multilinear rank from section* 7.4.

Remark 7.1. Presented algorithms can also be used on matrices, i.e., tensors of order $N = 2$. In that case the recommendation for when to use HOSVD1 still holds, while the HOSVD3 algorithm outperforms the HOSVD2 algorithm in other situations, as is evident from the complexities.

7.5. Testing complexities. Here we test that the complexities presented in this paper match the actual complexities of the algorithms HOSVD1, HOSVD2, HOSVD3, and HOSVD4. Using randomly generated tensors $\mathcal{X}$ and $\mathcal{Y}$, as in section 7.4, of size $I \times I \times I$ and multilinear ranks (R, R, R), with $t(I, R)$ we denote the time needed by our algorithms to recompress $\mathcal{Z} = \mathcal{X} * \mathcal{Y}$ to multilinear rank (R, R, R). We measure times for two different pairs (I_1, R_1) and (I_2, R_2) and compare the ratio $t(I_1, R_1)/t(I_2, R_2)$ with the expected ratio e, which is for each algorithm defined as follows:

- HOSVD1. Since I^3 is the dominating term in the complexity of HOSVD1 algorithm, we choose $I_2 = 2I_1$ and $R_1 = R_2$, so the expected ratio is

$$e = \frac{R_1 I_1^3}{R_2 I_2^3} = \frac{R_1 I_1^3}{R_1 2^3 I_1^3} = \frac{1}{2^3}.$$

TABLE 1
Comparison of ratios $t(I_1,R_1)/t(I_2,R_2)$ and expected ratios e for algorithms HOSVD1, HOSVD2, HOSVD3, and HOSVD4.

	I_1	I_2	R_1	R_2	$t(I_1,R_1)/t(I_2,R_2)$	e
HOSVD1	200	400	20	20	0.11615999	0.125
HOSVD2	50	800	10	20	0.00225188	0.00390625
HOSVD3 + Var. A	200	400	20	40	0.06338479	0.015625
HOSVD3 + Var. B	400	800	10	20	0.06642512	0.015625
HOSVD4	100	800	20	40	0.02764474	0.015625

- HOSVD2. Choosing $I_2 = 2^4 I_1$ and $R_2 = 2R_1$, we have

$$e = \frac{R_1^4 I_1 + R_1^8}{R_2^4 I_2 + R_2^8} = \frac{R_1^4 I_1 + R^8}{2^4 R_1^4 2^4 I_1 + 2^8 R^8} = \frac{1}{2^8}.$$

- HOSVD3 + Variant A. We use this variant when $I < R^2$ so we have to choose I and R accordingly. If $I_2 = 2I_1$ and $R_2 = 2R_1$, then

$$e = \frac{R_1^3 I_1^2 + R_1^5 I_1 + R_1^6}{R_2^3 I_2^2 + R_2^5 I_2 + R_2^6} = \frac{R_1^3 I_1^2 + R_1^5 I_1 + R_1^6}{2^3 R_1^3 2^2 I_1^2 + 2^5 R_1^5 2 I_1 + 2^6 R_1^6} \approx \frac{1}{2^6}.$$

- HOSVD3 + Variant B. We use this variant when $I \geq R^2$ so we have to choose I and R accordingly. If $I_2 = 2I_1$ and $R_2 = 2R_1$, then

$$e = \frac{R_1^5 I_1 + R_1^6}{R_2^5 I_2 + R_2^6} = \frac{R_1^5 I_1 + R^6}{2^5 R_1^5 2 I_1 + 2^6 R^6} = \frac{1}{2^6}.$$

- HOSVD4. Choosing $I_2 = 2^3 I_1$ and $R_2 = 2R_1$, we have

$$e = \frac{R_1^3 I_1 + R_1^6}{R_2^3 I_2 + R_2^6} = \frac{R_1^3 I_1 + R^6}{2^3 R_1^3 2^3 I_1 + 2^6 R^6} = \frac{1}{2^6}.$$

The results are presented in Table 1.

8. Conclusions. Exploiting structure significantly reduces the computational effort for recompressing the Hadamard product of two tensors in the Tucker format. Among the algorithms discussed in this paper, HOSVD4 clearly stands out. It uses a randomized algorithm that employs random vectors with rank-1 structure for approximating the range of the matricizations. The analysis of this randomized algorithm is subject to further work. It is not unlikely that the ideas of HOSVD4 extend to other SVD-based low-rank tensor formats, such as the tensor train format [22] and the hierarchical Tucker format [12].

We view the algorithms presented in this work as a complement to existing tools for recompressing tensors. Apart from the HOSVD, other existing tools include the sequentially truncated HOSVD [29], alternating optimization such as HOOI [27], cross approximation [25], and a Jacobi method [15]. As far as we know, none of these techniques has been tailored to the recompression of Hadamard products.

Appendix A. Julia package. We have created a Julia [3] package for tensors in Tucker format in programming language Julia, following the nomenclature of the MATLAB Tensor Toolbox [1]. A notable difference to the Tensor Toolbox is that we do not construct a separate object `tensor` for dealing with full tensors but instead directly use built-in multidimensional arrays. Our package has a standard structure:

```
TensorToolbox.jl/
├── src/
│   ├── TensorToolbox.jl
│   ├── tensor.jl
│   ├── ttensor.jl
│   └── helper.jl
└── test/
    ├── create_test_data.jl
    ├── paperTests.jl
    ├── runtests.jl
    └── test_data_func.jld
```

The module *TensorToolbox* is contained in `TensorToolbox.jl`, functionality for full tensors is in `tensor.jl`, and functionality for tensors in the Tucker format is in `ttensor.jl`.

The object (or composite type) for tensors in the Tucker format is called `ttensor` and consists of the fields `cten` (core tensor), `fmat` (factor matrices), as well as `isorth` (flag for orthonormality of factor matrices).

```
type ttensor{T<:Number}
  cten::Array{T}
  fmat::Array{Matrix,1}
  isorth::Bool
end
```

All figures from the paper can be recreated using functions from `test/paperTests.jl` file, following instructions in the file.

Even though in section 7 we present numerical results only for $N = 3$ and tensors of size $I \times I \times I$ with multilinear ranks (R, R, R), our package implements these algorithms for arbitrary order, different mode sizes, and different multilinear ranks.

The following tables summarize the functionality of our package.

Functions for tensors - `tensor.jl`	
`hosvd`	HOSVD.
`innerprod`	Inner product of two tensors.
`krontm`	Kronecker product of two tensors times matrix (n-mode multiplication).
`matten`	Matrix tensorization—fold matrix into a tensor.
`mkrontv`	Multiplication of matricized Kronecker product of two tensors by a vector.
`mrank`	Multilinear rank of a tensor.
`mttkrp`	Matricized tensor times Khatri–Rao product.
`nrank`	n-rank of a tensor.
`sthosvd`	Sequentially truncated HOSVD.
`tenmat`	Tensor matricization—unfold tensor into matrix.
`tkron`	Kronecker product of two tensors.
`ttm`	Tensor times matrix (n-mode multiplication).
`ttt`	Outer product of two tensors.
`ttv`	Tensor times vector (n-mode multiplication).

Functions for tensors in Tucker format - `ttensor.jl`

`ttensor`	Construct Tucker tensor for specified core tensor and factor matrices.
`randttensor`	Construct random Tucker tensor.
`coresize`*	Size of the core of `ttensor`.
`full`	Construct full tensor (array) from `ttensor`.
`had (.*)`*	Hadamard product of two `ttensor`.
`hadcten`*	Construct core tensor for Hadamard product of two `ttensor` with specified factor matrices.
`hosvd`*	HOSVD for `ttensor`.
`hosvd1`*	HOSVD1—computes Tucker representation of Hadamard product of two `ttensor`.
`hosvd2`*	HOSVD2—computes Tucker representation of Hadamard product of two `ttensor`.
`hosvd3`*	HOSVD3—computes Tucker representation of Hadamard product of two `ttensor`.
`hosvd4`*	HOSVD4—computes Tucker representation of Hadamard product of two `ttensor`.
`innerprod`	Inner product of two `ttensor`.
`isequal (==)`	True if each component of two `ttensor` is numerically identical.
`lanczos`*	Lanczos algorithm adapted for getting factor matrices in algorithm `hosvd3`.
`mhadtv`*	Matricized Hadamard product of two `ttensor` multiplied by a vector.
`minus (-)`*	Subtraction of two `ttensor`.
`mrank`*	Multilinear rank of a `ttensor`.
`msvdvals`*	Singular values of matricized `ttensor`.
`mtimes (*)`	Scalar multiplication for a `ttensor`.
`mttkrp`	Matricized `ttensor` times Khatri–Rao product.
`ndims`	Number of modes for `ttensor`.
`norm`	Norm of a `ttensor`.
`nrank`*	n-rank of a `ttensor`.
`nvecs`	Compute the leading mode-n vectors for `ttensor`.
`permutedims`	Permute dimensions of `ttensor`.
`plus (+)`*	Addition of two `ttensor`.
`randrange`*	Range approximation using randomized algorithm adapted for Hadamard product of two `ttensor`.
`randsvd`*	Randomized SVD algorithm adapted for getting factor matrices in algorithm `hosvd3`.
`reorth`*	Reorthogonalization of `ttensor`. Creates new `ttensor`.
`reorth!`*	Reorthogonalization of `ttensor`. Overwrites existing `ttensor`.
`size`	Size of a `ttensor`.
`tenmat`*	Unfold tensor into matrix—matricization.
`ttm`	Tensor times matrix for `ttensor` (n-mode multiplication).
`ttv`	Tensor times vector for `ttensor` (n-mode multiplication).
`uminus (-)`	Unary minus for `ttensor`.

Functions denoted by $\star$ are not part of the MATLAB Tensors Toolbox and the function `permutedims` is called `permute` in MATLAB.

Acknowledgments. The authors thank the referees for helpful remarks on an earlier version that improved the manuscript, and Ivan Slapničar for helpful discussions.

REFERENCES

[1] B. W. Bader, T. G. Kolda, et al., *Matlab tensor toolbox version* 2.6. Available online at http://www.sandia.gov/~tgkolda/TensorToolbox/, February 2015.

[2] J. Ballani and D. Kressner, *Matrices with hierarchical low-rank structures*, in Exploiting Hidden Structure in Matrix Computations: Algorithms and Applications, Lecture Notes in Math. 2173, Springer, Cham, 2016, pp. 161–209.

[3] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, *Julia: A fresh approach to numerical computing*, SIAM Rev., 59 (2017), pp. 65–98, https://doi.org/10.1137/141000671.

[4] F. Bonizzoni, F. Nobile, and D. Kressner, *Tensor train approximation of moment equations for elliptic equations with lognormal coefficient*, Comput. Methods Appl. Mech. Engrg., 308 (2016), pp. 349–376, https://doi.org/10.1016/j.cma.2016.05.026.

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

[6] S. Dolgov, B. N. Khoromskij, A. Litvinenko, and H. G. Matthies, *Polynomial chaos expansion of random coefficients and the solution of stochastic partial differential equations in the tensor train format*, SIAM/ASA J. Uncertain. Quantif., 3 (2015), pp. 1109–1135, https://doi.org/10.1137/140972536.

[7] A. Doostan and G. Iaccarino, *A least-squares approximation of partial differential equations with high-dimensional random inputs*, J. Comput. Phys., 228 (2009), pp. 4332–4345, https://doi.org/10.1016/j.jcp.2009.03.006.

[8] M. Espig, W. Hackbusch, A. Litvinenko, H. Matthies, and E. Zander, *Efficient analysis of high dimensional data in tensor formats*, in Sparse Grids and Applications, J. Garcke and M. Griebel, eds., Lect. Notes Comput. Sci. Eng. 88, Springer, 2013, pp. 31–56, https://doi.org/10.1007/978-3-642-31703-3_2.

[9] M. Filipović and A. Jukić, *Tucker factorization with missing data with application to low-n-rank tensor completion*, Multidimens. Syst. Signal Processing, 26 (2013), pp. 677–692, https://doi.org/10.1007/s11045-013-0269-9.

[10] M. Griebel and H. Harbrecht, *Approximation of bi-variate functions: Singular value decomposition versus sparse grids*, IMA J. Numer. Anal., 34 (2014), pp. 28–54, https://doi.org/10.1093/imanum/drs047.

[11] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, Springer, Heidelberg, 2012, https://doi.org/10.1007/978-3-642-28027-6.

[12] W. Hackbusch and S. Kühn, *A new scheme for the tensor representation*, J. Fourier Anal. Appl., 15 (2009), pp. 706–722, https://doi.org/10.1007/s00041-009-9094-9.

[13] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288, https://doi.org/10.1137/090771806.

[14] B. Hashemi and N. Trefethen, *Chebfun in three dimensions*, SIAM J. Sci. Comput., to appear.

[15] M. Ishteva, P.-A. Absil, and P. Van Dooren, *Jacobi Algorithm for the Best Low Multilinear Rank Approximation of Symmetric Tensors*, Tech. rep. UCL-INMA-2011.011, UC Louvain, Belgium, 2011.

[16] O. Koch and C. Lubich, *Dynamical tensor approximation*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2360–2375, https://doi.org/10.1137/09076578X.

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

[18] D. Kressner and C. Tobler, *Algorithm* 941*:* `htucker`*—A Matlab toolbox for tensors in hierarchical tucker format*, ACM Trans. Math. Softw., 40 (2014), 22, http://doi.acm.org/10.1145/2538688.

[19] N. Lee and A. Cichocki, *Fundamental Tensor Operations for Large-scale Data Analysis in Tensor Train Formats*, preprint, https://arxiv.org/abs/1405.7786, 2016.

[20] C. Musco and C. Musco, *Randomized block Krylov methods for stronger and faster approximate singular value decomposition*, in Conference on Neural Information Processing Systems (NIPS), MIT Press, Cambridge, MA, 2015, pp. 1396–1404.

[21] A. Nonnenmacher and C. Lubich, *Dynamical low-rank approximation: Applications and numerical experiments*, Math. Comput. Simulation, 79 (2008), pp. 1346–1357, https://doi.org/10.1016/j.matcom.2008.03.007.

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

[23] S. Ragnarsson, *Structured Tensor Computations: Blocking, Symmetries and Kronecker Factorizations*, Ph.D. thesis, Cornell University, Ithaca, NY, 2012.

[24] S. Ragnarsson and C. F. Van Loan, *Block tensor unfoldings*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 149–169, https://doi.org/10.1137/110820609.

[25] M. V. Rakhuba and I. V. Oseledets, *Fast multidimensional convolution in low-rank tensor formats via cross approximation*, SIAM J. Sci. Comput., 37 (2015), pp. A565–A582, https://doi.org/10.1137/140958529.

[26] R. Schneider and A. Uschmajew, *Approximation rates for the hierarchical tensor format in periodic Sobolev spaces*, J. Complexity, 30 (2014), pp. 56–71, https://doi.org/10.1016/j.jco.2013.10.001.

[27] B. N. Sheehan and Y. Saad, *Higher order orthogonal iteration of tensors (HOOI) and its relation to PCA and GLRAM*, in Proceedings of the 2007 SIAM International Conference on Data Mining, SIAM, Philadelphia, 2007, pp. 355–365, https://doi.org/10.1137/1.9781611972771.32.

[28] H. D. Simon and H. Zha, *Low-rank matrix approximation using the Lanczos bidiagonalization process with applications*, SIAM J. Sci. Comput., 21 (2000), pp. 2257–2274, https://doi.org/10.1137/S1064827597327309.

[29] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, *A new truncation strategy for the higher-order singular value decomposition*, SIAM J. Sci. Comput., 34 (2012), pp. A1027–A1052, https://doi.org/10.1137/110836067.