`htucker` – A MATLAB toolbox for tensors in hierarchical Tucker format*

Daniel Kressner[1] Christine Tobler[1]

November 11, 2013

– Extended version –

Abstract

The hierarchical Tucker format is a storage-efficient scheme to approximate and represent tensors of possibly high order. This paper presents a MATLAB toolbox, along with the underlying methodology and algorithms, which provides a convenient way to work with this format. The toolbox not only allows for the efficient storage and manipulation of tensors in hierarchical Tucker format but also offers a set of tools for the development of higher-level algorithms. Several examples for the use of the toolbox are given.

1 Introduction

A *tensor* $\mathcal{X} \in \mathbb{C}^{n_1 \times n_2 \times \cdots \times n_d}$ with $n_1, \ldots, n_d \in \mathbb{N}$ is a d-dimensional array with entries $\mathcal{X}_{i_1 i_2 \cdots i_d} \in \mathbb{C}$. Usually, d is called the *order* of the tensor and the focus of this paper is on tensors of higher order, say, $d = 5$ or $d = 10$ or even $d = 100$. A typical scenario is that $\mathcal{X}$ represents a d-variate function $f : [0,1]^d \to \mathbb{C}$ sampled on a tensor grid or approximated in a tensorized basis.

It is in general impossible to store a higher-order tensor explicitly, simply because the number of entries grows exponentially with d. Various data-sparse formats have been developed to address this issue. Depending on the application, these formats may allow for the approximate representation and manipulation of a tensor under dramatically reduced storage and computing requirements. For example, consider the approximation of $\mathcal{X}$ by a rank-1 tensor:

$$\operatorname{vec}(\mathcal{X}) \approx u_d \otimes u_{d-1} \otimes \cdots \otimes u_1, \qquad u_1 \in \mathbb{C}^{n_1},\ \ldots,\ u_d \in \mathbb{C}^{n_d}, \tag{1}$$

where vec stacks the entries of a tensor in reverse lexicographical order into a long column vector and $\otimes$ denotes the standard Kronecker product. Then, instead of the $n_1 \cdot n_2 \cdots n_d$ entries of $\mathcal{X}$, only the $n_1 + n_2 + \cdots + n_d$ entries of $u_1, \ldots, u_d$ need to be stored. On the function level, this corresponds to an approximation of f by a separable function.

A typical application we have in mind is when $\mathcal{X}$ arises from the discretization of a high-dimensional or parameter-dependent partial differential equation and is only given implicitly as

[1] Chair of Numerical Algorithms and HPC, MATHICSE, EPF Lausanne, CH-1015 Lausanne, Switzerland. {daniel.kressner,christine.tobler}@epfl.ch

[*] Supported by the FNS research module *Preconditioned methods for large-scale model reduction* within the FNS ProDoc *Efficient Numerical Methods for Partial Differential Equations.*

the solution to a typically huge (non)linear system or eigenvalue problem. There are two quite different strategies to employ a data-sparse format for the solution of such problems. The more straightforward one is to apply a standard iterative method, e.g., a conjugate gradient method, and approximate each iterate in the data-sparse format. For this purpose, it is desirable to keep the approximation error negligible; otherwise the accuracy and convergence of the method may be compromised. Examples for this strategy can be found in [7, 22, 28, 30, 34]. The second strategy is to reformulate the problem at hand as an optimization problem with the admissible set of solutions restricted to data-sparse tensors, see [14, 25, 26, 31, 39, 40] and the references therein. Beyond these two main strategies, there exist further approaches tailored to particularly structured problems, see, e.g., [17, 33]. While the mathematical understanding is still somewhat limited, there is strong numerical evidence that such data-sparse algorithms can handle a wide variety of problems that are far from tractable by classical numerical methods.

In most applications, it is unlikely that a rank-1 representation (1) yields a satisfactory approximation error. This can be improved by considering the more general CP (Canonical Polyadic) decomposition [24, 10]

$$\mathrm{vec}(\mathcal{X}) \approx \sum_{j=1}^{R} u_d^{(j)} \otimes u_{d-1}^{(j)} \otimes \cdots \otimes u_1^{(j)}, \qquad u_1^{(j)} \in \mathbb{C}^{n_1},\ \ldots,\ u_d^{(j)} \in \mathbb{C}^{n_d}, \tag{2}$$

which still requires little memory, provided that R does not become too large. Unfortunately, developing a robust and efficient algorithm for this format, which yields an approximation to any desirable accuracy, remains a subtle problem, see [1, 12] for recent progress. This problem is much less subtle for the Tucker decomposition [42],

$$\mathrm{vec}(\mathcal{X}) \approx \big(U_d \otimes U_{d-1} \otimes \cdots \otimes U_1\big)\,\mathrm{vec}(\mathcal{C}), \qquad U_1 \in \mathbb{C}^{n_1\times r_1},\ \ldots,\ U_d \in \mathbb{C}^{n_d\times r_d}, \tag{3}$$

with the so called core tensor $\mathcal{C} \in \mathbb{C}^{r_1\times r_2\times\cdots\times r_d}$. The HOSVD (Higher-Order SVD) [11] provides a simple, nearly optimal solution to the approximation problem (3). However, the need for storing $\mathcal{C}$ still results in memory requirements that grow exponentially with d.

Motivated by the limitations of the two classical decompositions (2) and (3), various other decompositions have been developed in the numerical analysis community with the aim of combining the advantages of both. This includes the tensor train decomposition [29], the closely related but somewhat more general HTD (hierarchical Tucker decomposition) [19, 23], and even more general tensor networks [13]. In the computational physics community, matrix product states and tensor networks play a central role in DMRG (density matrix renormalization group) techniques for computing ground states of quantum many-body systems, see [40] for an introduction. In the case of tensors that contain values of a multivariate function, additional properties inherited from the regularity of the function can in certain cases lead to a significantly higher compression in HTD (and related decompositions) compared to Tucker, see, e.g., [21] for such a discussion.

Existing Matlab toolboxes for working with low-rank tensor formats are the N-way toolbox by Andersson and Bro [2], the Tensor Toolbox by Bader and Kolda [4, 5], as well as the TT-Toolbox by Oseledets [37, 38]. TensorCalculus [15] is a C++ library for more general tensor networks. In computational physics, a number of related software packages have been developed in the context of DMRG techniques for simulating quantum networks, see, e.g., [9].

The main goal of this paper is to provide a convenient framework for the development and implementation of algorithms based on the hierarchical Tucker decomposition. In particular, methods for the truncation of a given tensor to HTD, i.e., the approximation by a low-rank tensor in HTD, are implemented and discussed in Section 6. A set of advanced tools allows for the development of higher-level algorithms. Moreover, this work contains a number of new contributions:

- Efficient algorithms for HTD tensor-tensor contraction, including the inner product between two tensors, are given in Sections 5.3 and 5.4.
- A new variant of truncation to HTD without initial orthogonalization is presented in Section 6.2.2 and demonstrated to result in an efficient and numerically robust way for adding tensors in Section 6.3.
- New algorithms for exact and approximate elementwise multiplication of tensors in HTD are presented in Section 7.
- A framework for representing linear operators in HTD is presented in Section 8 and an explicit representation for a discretized Laplace operator has been given.

All our algorithms are mainly based on calls to level 3 BLAS and LAPACK functionality.

The rest of this paper is organized as follows. Section 2 introduces basic tools for working with tensors as well as the HTD. In Section 3, we describe the basic functionality and data structures of our MATLAB toolbox `htucker`. Section 4 is concerned with basic operations on tensors in HTD, such as μ-mode matrix products and orthogonalization, and their implementation in `htucker`. Tensor-tensor contractions are discussed in Section 5, including, e.g., inner products. In Section 6, we present several methods for approximating a tensor either given explicitly or given in HTD by a tensor in HTD (of lower rank). Section 7 is concerned with elementwise multiplication, and Section 8 with the representation of linear operators on tensors in HTD. Finally, several examples for working with `htucker` are given in Section 9 and a list of the complete functionality of `htucker` can be found in Appendix A, while proofs of some of the error bounds given in Section 6 can be found in Appendix B.

2 Preliminaries

This section summarizes the mathematical foundation of the hierarchical Tucker decomposition (HTD) and our MATLAB toolbox. Necessary tensor concepts will be briefly introduced. We refer the reader to the survey paper [32] and to the book [21] for a more comprehensive introduction.

2.1 Matricization and HOSVD

To understand the principles behind HTD, it is helpful to recall the matricization and HOSVD of tensors. A tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times n_2 \times \cdots \times n_d}$ has d different *modes* $1, \dots, d$. Consider a splitting of these modes into two disjoint sets: $\{1, \dots, d\} = t \cup s$ with $t = \{t_1, \dots, t_k\}$ and $s = \{s_1, \dots, s_{d-k}\}$. Then the corresponding *matricization* of these modes is obtained by merging the first group into row indices and the second group into column indices:

$$X^{(t)} \in \mathbb{C}^{(n_{t_1} \cdots n_{t_k}) \times (n_{s_1} \cdots n_{s_{d-k}})} \quad \text{with} \quad \left(X^{(t)}\right)_{(i_{t_1}, \dots, i_{t_k}), (i_{s_1}, \dots, i_{s_{d-k}})} := \mathcal{X}_{i_1, \dots, i_d}$$

for any indices $i_1, \dots, i_d$ in the multi-index set $\{1, \dots, n_1\} \times \cdots \times \{1, \dots, n_d\}$. Of course, the order in which the indices are merged is important. In the following, we assume reverse lexicographical

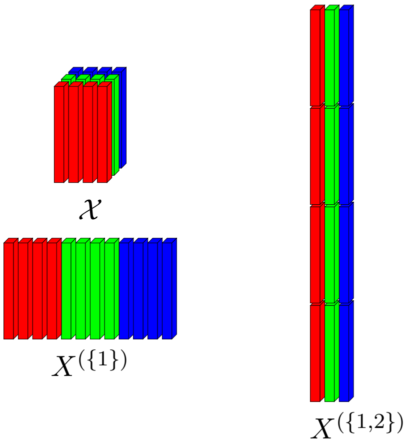


Figure 1: Illustration of an $n_1 \times 4 \times 3$ tensor $\mathcal{X}$ with matricizations $X^{(\{1\})}$ and $X^{(\{1,2\})}$.

order of the multi-indices but any other consistently employed order would be suitable. Note that the matricization is *not* independent of the ordering of the modes $t_1, \dots, t_k$ and $s_1, \dots, s_{d-k}$ in the sets t and s, respectively. If not noted otherwise, we assume the modes to be in increasing order.

As a special case, consider the so called μ-mode matricization

$$X^{(\mu)} \in \mathbb{C}^{n_\mu \times (n_1 \cdots n_{\mu-1} n_{\mu+1} \cdots n_d)}, \qquad \mu = 1, \dots, d.$$

Then the tuple $(r_1, \dots, r_d)$ with $r_\mu = \operatorname{rank}\bigl(X^{(\mu)}\bigr)$ is called the *multilinear rank* of $\mathcal{X}$. To obtain an approximation of lower multilinear rank $(\widetilde{r}_1, \dots, \widetilde{r}_d)$, with $\widetilde{r}_\mu \le r_\mu \le n_\mu$, we let $U_\mu \in \mathbb{C}^{n_\mu \times \widetilde{r}_\mu}$ contain the $\widetilde{r}_\mu$ dominant left singular vectors of $X^{(\mu)}$, which can be obtained, e.g., from a truncated SVD $X^{(\mu)} \approx U_\mu \Sigma_\mu V_\mu^H$. Then the HOSVD takes the form of a Tucker decomposition

$$\operatorname{vec}(\mathcal{X}) \approx \operatorname{vec}(\widetilde{\mathcal{X}}) := (U_d \otimes \cdots \otimes U_1) \operatorname{vec}(\mathcal{C}), \tag{4}$$

with the core tensor

$$\operatorname{vec}(\mathcal{C}) := (U_d^H \otimes \cdots \otimes U_1^H) \operatorname{vec}(\mathcal{X}) \in \mathbb{C}^{\widetilde{r}_1 \times \cdots \times \widetilde{r}_d}.$$

This choice of $\mathcal{C}$ minimizes $\|\mathcal{X} - \widetilde{\mathcal{X}}\|_2$ for given $U_1, \dots, U_d$ with orthonormal columns. Here and in the following, $\|\mathcal{Y}\|_2$ denotes the Euclidean norm of the vectorization: $\|\mathcal{Y}\|_2 = \|\operatorname{vec}(\mathcal{Y})\|_2$. An important feature of the HOSVD, it can be shown [11] that the obtained approximation is nearly optimal among *all* tensors of multilinear rank $(\widetilde{r}_1, \dots, \widetilde{r}_d)$ or lower:

$$\|\mathcal{X} - \widetilde{\mathcal{X}}\|_2 \le \sqrt{d} \cdot \inf\bigl\{\|\mathcal{X} - \mathcal{Y}\|_2 : \operatorname{rank}(Y^{(\mu)}) \le \widetilde{r}_\mu,\ \mu = 1, \dots, d\bigr\}. \tag{5}$$

2.2 The Hierarchical Tucker Decomposition (HTD)

In contrast to the Tucker decomposition, HTD employs a hierarchy of matricizations, motivated by the following nestedness property. Note that our use of the symbol $\subset$ for set inclusion allows for equal sets.

Lemma 2.1 ([18, Lemma 17]). *Let $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ and $t = t_l \cup t_r$ for $t_l = \{i_l, i_l + 1, \dots, i_m\}$ and $t_r = \{i_m + 1, \dots, i_r\}$. Then* $\operatorname{span}\bigl(X^{(t)}\bigr) \subset \operatorname{span}\bigl(X^{(t_r)} \otimes X^{(t_l)}\bigr)$.

Proof. Any column of $X^{(t)} = X^{(i_l,\dots,i_r)}$ can be considered as the vectorization of a tensor $\mathcal{C} \in \mathbb{C}^{n_{i_l}\times\cdots\times n_{i_r}}$. The columns of the matricization $C^{(t_l)}$ are clearly contained in $\operatorname{span}\big(X^{(t_l)}\big)$ and hence

$$C^{(t_l)} = X^{(t_l)}\big(X^{(t_l)}\big)^{+}C^{(t_l)},$$

where M^+ denotes the Moore-Penrose pseudoinverse of a matrix M. Analogously,

$$\big(C^{(t_l)}\big)^T = C^{(t_r)} = X^{(t_r)}\big(X^{(t_r)}\big)^{+}C^{(t_r)}.$$

These two relations imply

$$C^{(t_l)} = X^{(t_l)}\underbrace{\Big(\big(X^{(t_l)}\big)^{+}C^{(t_l)}\big(X^{(t_r)}\big)^{+T}\Big)}_{=:V}\big(X^{(t_r)}\big)^{T} \;\Rightarrow\; \operatorname{vec}(\mathcal{C}) = \big(X^{(t_r)}\otimes X^{(t_l)}\big)\operatorname{vec}(V).$$

□

Given any bases U_t, U_{t_l}, U_{t_r} for the column spaces of $X^{(t)}, X^{(t_l)}, X^{(t_r)}$, the result of Lemma 2.1 implies the existence of a so called transfer matrix B_t such that

$$U_t = (U_{t_r}\otimes U_{t_l})B_t, \quad B_t \in \mathbb{C}^{r_{t_l}r_{t_r}\times r_t}, \tag{6}$$

where r_t, r_{t_l}, r_{t_r} denote the ranks of the corresponding matricizations. Applying this relation recursively, until t_l and t_r become singletons, leads to the HTD.

Example 2.2. *Repeated application of (6) for $d = 4$:*

$$\begin{aligned} vec(\mathcal{X}) = X^{(\{1,2,3,4\})} &= (U_{34}\otimes U_{12})B_{1234} \\ U_{12} &= (U_2\otimes U_1)B_{12} \\ U_{34} &= (U_4\otimes U_3)B_{34} \\ \Rightarrow \quad vec(\mathcal{X}) &= (U_4\otimes U_3\otimes U_2\otimes U_1)(B_{34}\otimes B_{12})B_{1234}. \end{aligned} \tag{7}$$

It is often advantageous to reshape the transfer matrices:

$$\begin{aligned} B_{1234} \in \mathbb{C}^{r_{12}r_{34}\times 1} &\;\Rightarrow\; \mathcal{B}_{1234} \in \mathbb{C}^{r_{12}\times r_{34}\times 1}, \\ B_{12} \in \mathbb{C}^{r_1 r_2\times r_{12}} &\;\Rightarrow\; \mathcal{B}_{12} \in \mathbb{C}^{r_1\times r_2\times r_{12}}, \\ B_{34} \in \mathbb{C}^{r_3 r_4\times r_{34}} &\;\Rightarrow\; \mathcal{B}_{34} \in \mathbb{C}^{r_3\times r_4\times r_{34}}. \end{aligned}$$

To avoid cluttering the notation we write B_{12} instead of $B_{\{1,2\}}$, U_{12} instead of $U_{\{1,2\}}$, r_{12} instead of $r_{\{1,2\}}$, and so on.

An illustration of the hierarchical structure and the data to be stored for the HTD (7) is given in Figure 2.

The general construction of an HTD requires a hierarchical splitting of the modes $1,\dots,d$.

Definition 2.3. *A binary tree $\mathcal{T}$ with each node represented by a subset of $\{1,\dots,d\}$ is called a* dimension tree *if the root node is $\{1,\dots,d\}$, each leaf node is a singleton, and each parent node is the disjoint union of its two children. In the following, we denote:*

$\mathcal{L}(\mathcal{T})$ *set of all leaf nodes;*
$\mathcal{N}(\mathcal{T})$ *set of all non-leaf nodes, $\mathcal{N}(\mathcal{T}) = \mathcal{T}\setminus\mathcal{L}(\mathcal{T})$.*

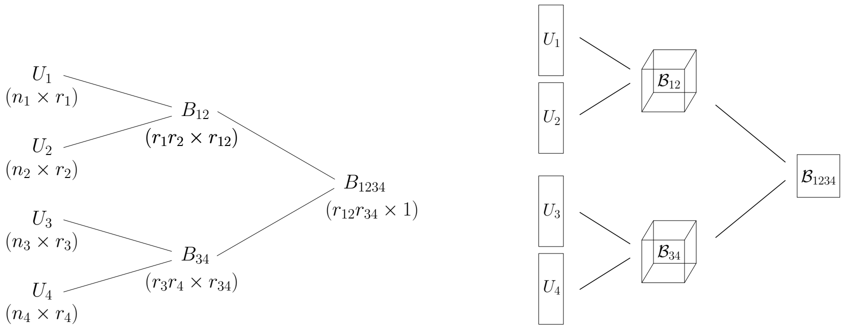


Figure 2: Illustration of the HTD (7) for $d = 4$.

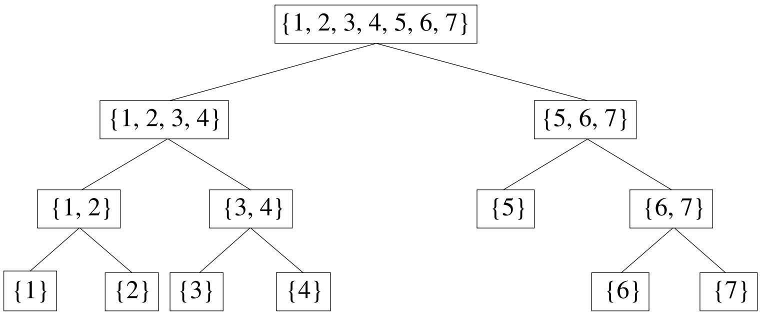


Figure 3: A dimension tree for $d = 7$.

Remark 2.4. *For convenience of notation, we impose the following additional assumption on the left and right children t_l, t_r of a node t in the dimension tree: Each element of t_l is smaller than any element of t_r. Note that this assumption can always be satisfied by an appropriate reordering of the modes.*

It is not hard to see that the number of non-leaf nodes is always $d - 1$. An example of a dimension tree is given in Figure 3.

Having prescribed a maximal rank k_t for each node $t \in \mathcal{T}$, the set of *hierarchical Tucker tensors of hierarchical rank at most* $(k_t)_{t \in \mathcal{T}}$ is defined as

$$\mathcal{H}\text{-Tucker}\big((k_t)_{t\in\mathcal{T}}\big) = \Big\{\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d} : \operatorname{rank}\big(X^{(t)}\big) \leq k_t \text{ for all } t \in \mathcal{T}\Big\}.$$

Such a hierarchical Tucker tensor $\mathcal{X}$ is stored in the *hierarchical Tucker format* as follows. At each leaf node $\{\mu\}$ a basis $U_\mu \in \mathbb{C}^{n_\mu \times r_\mu}$, where $r_\mu := \operatorname{rank}\big(X^{(\mu)}\big) \leq k_\mu$, is stored. At each parent node t with children t_l and t_r, the third-order transfer tensor $\mathcal{B}_t \in \mathbb{C}^{r_{t_l} \times r_{t_r} \times r_t}$ satisfying (6) with $B_t \equiv B_t^{(\{1,2\})}$ is stored. Equivalently, (6) can be written as

$$(U_t)_{:,q} = \sum_{i=1}^{r_{t_l}} \sum_{j=1}^{r_{t_r}} \Big((U_{t_r})_{:,j} \otimes (U_{t_l})_{:,i}\Big)(\mathcal{B}_t)_{i,j,q}, \quad q = 1, 2, \ldots, r_t. \tag{8}$$

The HTD for $\mathcal{X}$ is obtained by recursively inserting (6) as illustrated in Example 2.2.

In summary, the hierarchical Tucker format is represented by d matrices U_μ and $(d-1)$ transfer tensors $\mathcal{B}_t$. Hence, if $r = \max\{r_t : t \in \mathcal{T}\}$ and $n = \max\{n_1, \ldots, n_d\}$, the storage requirements are bounded by

$$dnr + (d-2)r^3 + r^2, \tag{9}$$

where we have used that the transfer tensor at the root can actually be considered as a matrix of size at most $r \times r$. While the complexity bound (9) appears to indicate linear dependence in d, it is important to note that such a conclusion only holds when the maximal hierarchical rank r can be assumed to stay constant as d increases. This is a rather strong assumption that is satisfied only in specific applications.

3 Basic Functionality of the Toolbox

To conveniently work with tensors in HTD, we have implemented a new MATLAB class `htensor`, inspired by the classes `ktensor` (for tensors in CP decomposition) and `ttensor` (for tensors in Tucker decomposition) available in the Tensor Toolbox [4]. In the following, we describe the structure of `htensor` as well as its basic functionality.

3.1 Fields and properties of `htensor`

An instance of `htensor` contains two arrays specifying the dimension tree, an orthogonalization flag, as well as the matrices and transfer tensors representing an HTD corresponding to this dimension tree.

Each node of the dimension tree is associated with an index $i \in \{1, \ldots, 2d-1\}$, such that each child node has a larger index than the parent node. Consequently, the root node has index 1. The $(2d-1) \times 2$ integer array `children` specifies the structure of the dimension tree as follows: `children(i, 1)` is the left child of node i, and `children(i, 2)` is the right child of node i. Both entries are zero if node i is a leaf node. The $1 \times d$ integer array `dim2ind` gives the index of the leaf node associated with each mode $\mu = 1, \ldots, d$. The matrices U_t and transfer tensors B_t are stored in the cell arrays `U` and `B`, respectively. Note that `U{i}` is a matrix if `i` is a leaf node and an empty array for any other node. Finally, the boolean flag `is_orthog` indicates whether the HTD is orthogonalized, see Section 4.3.

`htensor` for $d = 4$ (see also Example 2.2):

```
x.children: [2, 3; 4, 5; 6, 7; 0, 0; 0, 0; 0, 0; 0, 0]
x.dim2ind: [4 5 6 7]
x.U: {[] [] [] [4x4 double] [5x4 double] [6x6 double] [7x3 double]}
x.B: {[4x5 double] [4x4x4 double] [6x3x5 double] [] [] [] []}
x.is_orthog: false
```

Apart from the fields defining the HTD, `htensor` has additional fields for accessing frequently required properties.

Properties of `htensor`:

`x.nr_nodes`	number of nodes in the dimension tree.
`x.parent(i)`	returns the index of the parent of node i.
`x.sibling(i)`	returns the index of the sibling of node i.
`x.is_leaf(i)`	true if node i is a leaf node.
`x.is_left(i)`	true if node i is a left child.
`x.is_right(i)`	true if node i is a right child.
`x.lvl(i)`	level of node i (distance from root node).
`x.dims{i}`	modes represented by node i.
`x.bin_dims(i, :)`	modes represented by node i (logical array).
`x.rank(i)`	hierarchical rank at node i.

3.2 Constructors of `htensor`

There are several ways to construct an `htensor` instance. In the following, we only illustrate the most common ways and refer to the documentation, e.g., `help htensor/htensor`, for more details.

Examples for constructors of `htensor`:

`x = htensor([4 5 6 7])` constructs a zero `htensor` of size $4 \times 5 \times 6 \times 7$.

`x = htensor([4 5 6 7], 'TT')` constructs a zero `htensor` of size $4 \times 5 \times 6 \times 7$, with a degenerate, TT-like dimension tree.

`x = htensor({U1, U2, U3})` constructs an `htensor` from the CP tensor defined by $\mathcal{X}(i_1, i_2, i_3) = \sum_{j=1}^{R} U_1(i_1, j) U_2(i_2, j) U_3(i_3, j)$.

`x = htenones([4 5 6 7])` constructs an `htensor` of size $4 \times 5 \times 6 \times 7$, with all entries one.

`x = htenrandn([4 5 6 7])` constructs an `htensor` of size $4 \times 5 \times 6 \times 7$, with random ranks and random entries.

By default, any `htensor` has a balanced dimension tree. The motivation for the option `'TT'` is to resemble the structure of the TTD (tensor train decomposition). However, it is important to note that there is no exact correspondence between HTD and TTD, as TTD does not require the storage of basis matrices. An arbitrary dimension tree can be generated by supplying the fields `children` and `dim2ind` to the constructor. Users of the Tensor Toolbox may also provide a `ktensor` for constructing an `htensor` from a CP decomposition, instead of providing the factors in a cell array as illustrated above.

3.3 Basic functionality of `htensor`

Table 1 in the appendix contains all basic functions for working with `htensor` objects. The following example illustrates their use for a $5 \times 4 \times 6 \times 3$ tensor $\mathcal{X}$ in HTD as in Example 2.2.

`x(1, 3, 4, 2)` returns the entry $\mathcal{X}_{1,3,4,2}$.

`x(1, 3, :, :)` returns an `htensor` representing the 6×3 tensor of the slice $\mathcal{X}_{1,3,:,:}$.

`full(x)` returns the full tensor represented by $\mathcal{X}$.

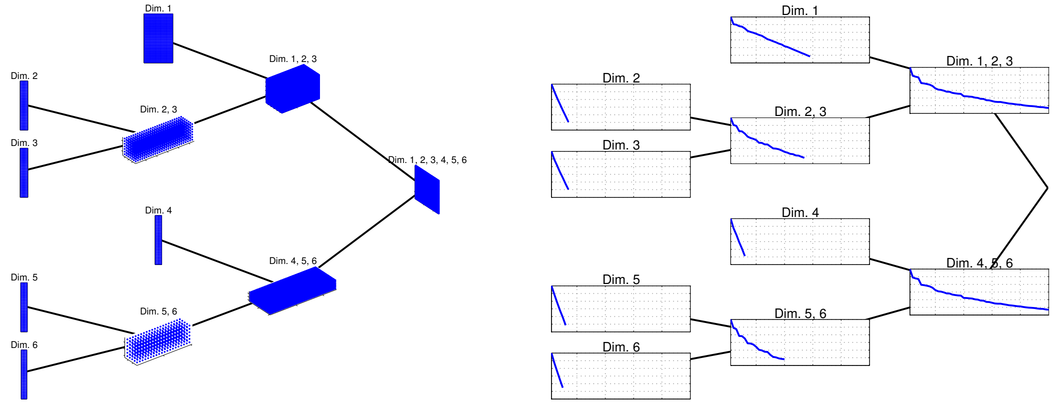


Figure 4: Examples for `spy` and `plot_sv`

`x(:)` returns the vectorization of $\mathcal{X}$.
`size(x)` returns the array of dimensions, `[5, 4, 6, 3]`.
`ndims(x)` returns the order of the tensor, `4`.
`disp(htenrandn([5 4 6 3]))` returns the tree structure and the sizes of the transfer tensors/basis matrices

```
  1-4       1; 6 3 1
    1-2     2; 3 4 6
      1     4; 5 3
      2     5; 4 4
    3-4     3; 3 3 3
      3     6; 6 3
      4     7; 3 3
```

`disp_all(x)` additionally displays all transfer tensors and basis matrices.
`spy(x)` displays the dimension tree with spy plots of U_t and B_t, see Figure 4.
`plot_sv(x)` displays the dimension tree with semi-log plots of the singular values of the matricizations at each node, see Figure 4.

4 Basic operations

This section describes algorithms and implementations for a range of typically required basic operations.

4.1 μ-mode matrix products

Given a tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, the μ-mode product with a matrix $A \in \mathbb{C}^{m \times n_\mu}$ is defined via the μ-mode matricization:

$$\mathcal{Y} = A \circ_\mu \mathcal{X} \quad \Leftrightarrow \quad Y^{(\mu)} = AX^{(\mu)}.$$

This operation can easily be performed if $\mathcal{X}$ is in HTD: The μth basis matrix U_μ is simply replaced by $A\,U_\mu$. We provide a function `ttm` for performing this operation. The calling sequence of `ttm` is nearly identical with the function `ttm` in the Tensor Toolbox. A notable difference is that we allow the matrix the matrix A to be provided implicitly as a handle to a function that returns the product of A with the input matrix.

`y = ttm(x, A, 2)` applies the matrix A to an `htensor` in mode 2.
`y = ttm(x, {A, B, C}, [2, 3, 4])` successively applies A, B, C to $\mathcal{X}$ in modes $2, 3, 4$.
`y = ttm(x, @(x)(fft(x)), 2)` applies the fast Fourier transformation to $\mathcal{X}$ in mode 2.
`y = ttm(x, {A, B, C}, [2, 3, 4], 'h')` successively applies A^H, B^H, C^H to $\mathcal{X}$ in modes $2, 3, 4$.

In the special case of μ-mode multiplication with a row vector, the μth mode becomes a singleton dimension. The function `ttv` treats this case differently by eliminating the μth mode after performing the product $v^T \circ_\mu \mathcal{X}$ for a vector $v \in \mathbb{C}^{n_\mu}$. The following example illustrates the difference.

`x = htenrandn([5, 4, 6, 3]); u = randn(4, 1);`
`y = ttm(x, u.', 2)` results in an `htensor` $\mathcal{Y}$ of size $5 \times 1 \times 6 \times 3$
`y = ttv(x, u, 2)` results in an `htensor` $\mathcal{Y}$ of size $5 \times 6 \times 3$

Note that there is also a function `squeeze` for eliminating singleton dimensions.

4.2 Addition

The addition of tensors in HTD can be performed at no arithmetic cost by a simple embedding. The underlying principle can easily be seen for two factorized matrices $A = U_A \Sigma_A V_A^H$ and $B = U_B \Sigma_B V_B^H$:

$$A + B = \begin{bmatrix} U_A & U_B \end{bmatrix} \begin{bmatrix} \Sigma_A & 0 \\ 0 & \Sigma_B \end{bmatrix} \begin{bmatrix} V_A & V_B \end{bmatrix}^H.$$

This embedding is performed similarly for the addition of two or more tensors in HTD, by concatenation of the leaf matrices and a block diagonal embedding of the transfer tensors. We refrain from giving a technical description and refer to Figure 5 for an illustration. It is important to note that the storage requirements grow cubically in the number of tensors to be added if the block diagonal structure of the transfer tensors is not exploited. Such an alternative method is discussed in Section 6.3.

Addition is implemented in the command `plus`, which overloads the binary operator `+` for `htucker` objects. Subtraction is implemented in the command `minus`, which overloads the binary operator `-`.

4.3 Orthogonalization

An HTD of a tensor $\mathcal{X}$ is called *orthogonalized* if the columns of U_t form an orthonormal basis for each node t except for the root node. Recall that the basis matrices U_t are defined recursively according to (6). As will be seen later, orthogonalized HTDs simplify some important operations related to tensor contraction. Moreover, they reduce the risk of numerical cancellation.

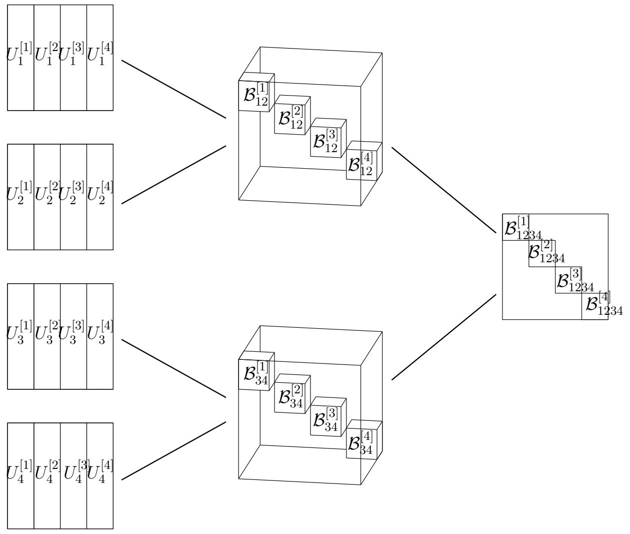

$U_1^{[1]}$ $U_1^{[2]}$ $U_1^{[3]}$ $U_1^{[4]}$
$U_2^{[1]}$ $U_2^{[2]}$ $U_2^{[3]}$ $U_2^{[4]}$
$U_3^{[1]}$ $U_3^{[2]}$ $U_3^{[3]}$ $U_3^{[4]}$
$U_4^{[1]}$ $U_4^{[2]}$ $U_4^{[3]}$ $U_4^{[4]}$
$\mathcal{B}_{12}^{[1]}$ $\mathcal{B}_{12}^{[2]}$ $\mathcal{B}_{12}^{[3]}$ $\mathcal{B}_{12}^{[4]}$
$\mathcal{B}_{34}^{[1]}$ $\mathcal{B}_{34}^{[2]}$ $\mathcal{B}_{34}^{[3]}$ $\mathcal{B}_{34}^{[4]}$
$\mathcal{B}_{1234}^{[1]}$ $\mathcal{B}_{1234}^{[2]}$ $\mathcal{B}_{1234}^{[3]}$ $\mathcal{B}_{1234}^{[4]}$

Figure 5: Addition of four tensors $\mathcal{X}_1 + \mathcal{X}_2 + \mathcal{X}_3 + \mathcal{X}_4$ in HTD.

We illustrate the process of orthogonalization for the tensor in standard HTD from Example 2.2:

$$\mathrm{vec}(\mathcal{X}) = (U_4 \otimes U_3 \otimes U_2 \otimes U_1)(B_{34} \otimes B_{12})B_{1234}.$$

In the first step, QR decompositions of the basis matrices are performed: $U_t = \widetilde{U}_t R_t$ for $t = 1, \dots, 4$. Here and in the following, "economic" QR decompositions [16] are performed, i.e., U_t and $\widetilde{U}_t$ have the same number of columns. Propagating the factors R_t into the transfer matrices results in

$$\mathrm{vec}(\mathcal{X}) = (\widetilde{U}_4 \otimes \widetilde{U}_3 \otimes \widetilde{U}_2 \otimes \widetilde{U}_1)(\widehat{B}_{34} \otimes \widehat{B}_{12})B_{1234}$$

with $\widehat{B}_{34} := (R_4 \otimes R_3)B_{34}$, $\widehat{B}_{12} := (R_2 \otimes R_1)B_{12}$. In the next step, QR decompositions $\widehat{B}_{34} = \widetilde{B}_{34}R_{34}$, $\widehat{B}_{12} = \widetilde{B}_{12}R_{12}$ are performed, resulting in

$$\mathrm{vec}(\mathcal{X}) = (\widetilde{U}_4 \otimes \widetilde{U}_3 \otimes \widetilde{U}_2 \otimes \widetilde{U}_1)(\widetilde{B}_{34} \otimes \widetilde{B}_{12})\widetilde{B}_{1234} \tag{10}$$

with $\widetilde{B}_{1234} := (R_{34} \otimes R_{12})B_{1234}$. Clearly, (10) constitutes an orthogonalized HTD, completing the orthogonalization procedure.

This procedure easily extends to the general case, see Algorithm 1. Properly implemented, orthogonalization requires $\mathcal{O}(dnr^2 + dr^4)$ operations. Unless r is very small, the factor dr^4, caused by the QR decompositions of the transfer matrices, will be dominant.

In the `htucker` toolbox, orthogonalization is performed by calling `x = orthog(x)`. On return, the flag `is_orthog` of the `htensor` object `x` is set to true. This prevents unnecessary orthogonalization in subsequent calls to `orthog`. When this property is destroyed by an operation, such as addition and μ-mode matrix products, the flag `is_orthog` is set to false.

5 Tensor-Tensor Contraction

The contraction of two tensors $\mathcal{X} \in \mathbb{C}^{m_1 \times \cdots \times m_c}$ and $\mathcal{Y} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ is a fundamental operation which generalizes the concepts of inner product, outer product, and matrix multiplication. In

Algorithm 1 Orthogonalization of a tensor in HTD

```
Input: Basis matrices $U_t$ and transfer tensors $\mathcal{B}_t$ defining a general HTD of a tensor $\mathcal{X}$.
Output: Basis matrices $\widetilde{U}_t$ and transfer tensors $\widetilde{\mathcal{B}}_t$ defining an orthogonalized HTD of $\mathcal{X}$.
  for $t \in \mathcal{L}(\mathcal{T})$: Compute QR decomposition $U_t =: \widetilde{U}_t R_t$.
  for $t \in \mathcal{N}(\mathcal{T})$ (visiting both child nodes before the parent node) do
    Form $\widehat{B}_t = (R_{t_r} \otimes R_{t_l}) B_t$.
    if $t$ is root node then
      Set $\widetilde{B}_t = \widehat{B}_t$.
    else
      Compute QR decomposition $\widehat{B}_t =: \widetilde{B}_t R_t$.
    end if
  end for
```

its most general form, we select a tuple of modes $s = (i_1, \ldots, i_p)$ from $\mathcal{X}$ and another tuple of modes $t = (j_1, \ldots, j_p)$ from $\mathcal{Y}$. Then the corresponding contraction of $\mathcal{X}$ and $\mathcal{Y}$ amounts to taking the inner product with respect to each pair (i_μ, j_μ) of selected modes. This implicitly assumes that the sizes of the selected modes match, i.e., $n_{i_\mu} = m_{j_\mu}$ for $\mu = 1, \ldots, p$. Contraction results in a tensor $\mathcal{Z}$ whose order equals the number of non-selected modes. For example, for $c = 4, d = 3$, and $s = (3,1), t = (2,3)$, the contracted tensor $\mathcal{Z} \in \mathbb{C}^{m_2 \times m_4 \times n_1}$ is given by

$$\mathcal{Z}_{i_1,i_2,i_3} = \langle \mathcal{X}, \mathcal{Y} \rangle_{(3,1),(2,3)} := \sum_{j=1}^{n_3} \sum_{k=1}^{n_1} \bar{\mathcal{X}}_{k,i_1,j,i_2} \mathcal{Y}_{i_3,j,k}. \tag{11}$$

A matrix-matrix multiplication $X^H Y$ for $X \in \mathbb{C}^{n_1 \times n_2}$, $Y \in \mathbb{C}^{m_1 \times m_2}$ with $n_1 = m_1$ can be seen as the contraction corresponding to $s = (1)$, $t = (1)$. Conversely, a general tensor-tensor contraction of $\mathcal{X} \in \mathbb{C}^{m_1 \times \cdots \times m_c}$ and $\mathcal{Y} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ along p selected pairs of nodes can be defined via matrix-matrix multiplication:

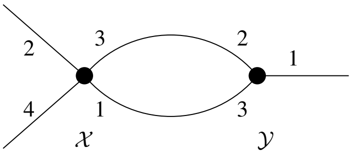
$$Z^{(\bar{t})} = \left(\langle \mathcal{X}, \mathcal{Y} \rangle_{(t;s)} \right)^{(\bar{t})} := (X^{(t)})^H Y^{(s)}, \quad \text{with} \quad \bar{t} = (1, 2, \ldots, c-p). \tag{12}$$

Note that the matricization needs to respect the order of the modes in t and s, see also Section 2.1.

5.1 Tensor Network Diagrams

In the following, we briefly introduce tensor network diagrams (also called Penrose diagrams) to conveniently describe algorithms for contractions of tensors in HTD, see also [25, 26]. Such a diagram represents a tensor in terms of contractions of other tensors. Each node in the diagram represents a tensor and each edge represents a mode. An edge connecting two nodes corresponds to the contraction of these tensors in the associated pair of modes. We allow for dangling edges, which are connected to only one node and correspond to modes that are not contracted. Hence, the order of the tensor represented by the network is given by the number of dangling edges.

To illustrate the concept of tensor network diagrams, let us consider the contraction given in (11). This is represented by the following diagram:



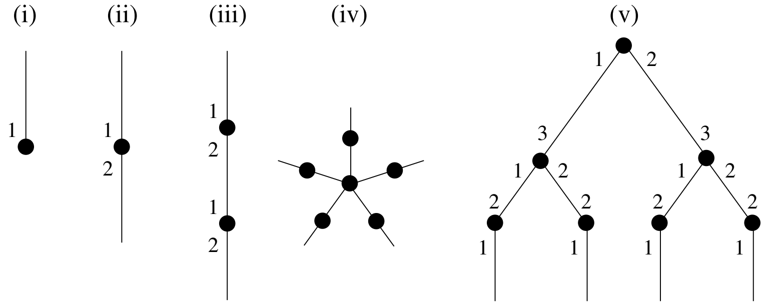


Figure 6: Tensor network diagrams representing (i) a vector, (ii) a matrix, (iii) a matrix-matrix multiplication, (iv) a tensor in Tucker decomposition, and (v) a tensor in HTD.

The node $\mathcal{X}$ has four edges, each corresponding to one of the four different modes of the tensor $\mathcal{X}$. Similarly, the node $\mathcal{Y}$ has three edges. Since mode 3 of $\mathcal{X}$ is contracted with mode 2 of $\mathcal{Y}$, the nodes share the corresponding edge. Similarly, the contraction of modes 1 and 3 is represented by another shared edge.

Some further examples of tensor network diagrams are given in Figure 6. Note that we will only indicate the precise mode(s) belonging to an edge when necessary. In the case of HTD, each edge connecting two nodes corresponds to a matricization $X^{(t)}$ for some $t \in \mathcal{T}$.

5.2 Inner Product and Norm for Tensors in HTD

The inner product of two tensors $\mathcal{X}, \mathcal{Y} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ is an important special case of contraction:

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \bar{\mathcal{X}}_{i_1,\ldots,i_d} \mathcal{Y}_{i_1,\ldots,i_d}$$

or, equivalently, $\langle \mathcal{X}, \mathcal{Y} \rangle = \langle \mathrm{vec}(\mathcal{X}), \mathrm{vec}(\mathcal{Y}) \rangle$. In terms of tensor network diagrams, this operation corresponds to a pairwise connection of the dangling edges of $\bar{\mathcal{X}}$ and $\mathcal{Y}$.

To illustrate how to evaluate inner products of tensors in HTD efficiently, we first consider two tensors of order 4:

$$\langle \mathcal{X}, \mathcal{Y} \rangle = (B^x_{1234})^H (B^x_{34} \otimes B^x_{12})^H (U^x_4 \otimes U^x_3 \otimes U^x_2 \otimes U^x_1)^H (U^y_4 \otimes U^y_3 \otimes U^y_2 \otimes U^y_1)(B^y_{34} \otimes B^y_{12}) B^y_{1234}.$$

This product is evaluated from inside to outside or, when considering the hierarchical tree, from the leaves to the root node. In a first step, the matrices

$$M_t = \left(U^x_t\right)^H U^y_t, \quad t = 1, \ldots, 4,$$

are computed. Then the product

$$M_t = (U^x_t)^H U^y_t = (B^x_t)^H \Big(M_{t_r} \otimes M_{t_l}\Big) B^y_t \tag{13}$$

is computed for $t = \{1,2\}$ and $t = \{3,4\}$. Note that the matrix $M_{t_r} \otimes M_{t_l}$ is not formed explicitly but the matrices M_{t_l} and M_{t_r} are applied to the modes 1 and 2 of B^y_t, respectively. The product (13) then requires $6r^4$ operations if both $\mathcal{X}$ and $\mathcal{Y}$ have constant hierarchical rank

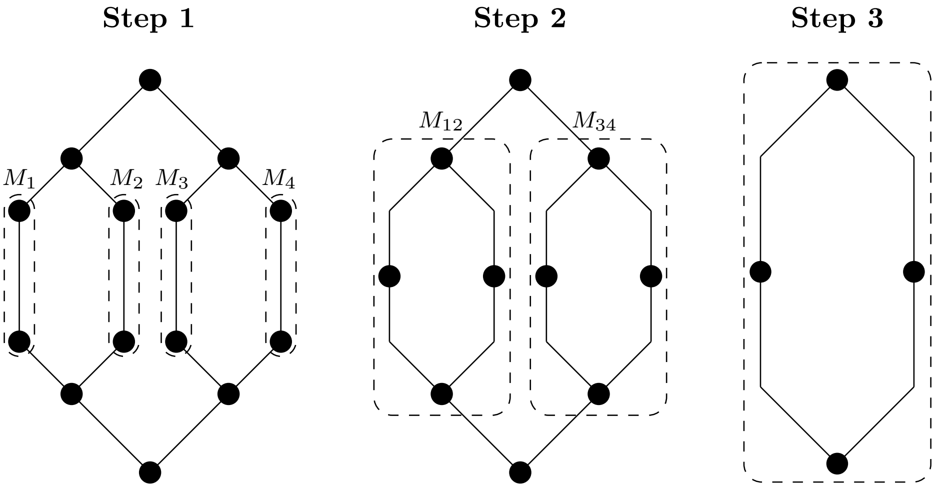


Figure 7: Inner product of two tensors of order 4 in HTD.

r. In the last step, $\langle \mathcal{X}, \mathcal{Y} \rangle$ is obtained by evaluating (13) for $t = \{1, 2, 3, 4\}$. Figure 7 illustrates the described procedure with tensor network diagrams.

The generalization to tensors of arbitrary order is straightforwardand summarized in Algorithm 2. Note that the algorithm assumes that $\mathcal{X}$ and $\mathcal{Y}$ have the same dimension tree. This requirement can be slightly relaxed as discussed in the next section for a more general setting. In total, forming the inner product of two tensors with constant hierarchical rank r requires $6(d-1)r^4 + \sum_{\mu=1}^{d} 2n_\mu r^2$ operations. Algorithm 2 is implemented in the `htucker` function `innerprod`.

Algorithm 2 Inner product of two tensors in HTD

```
Input: Tensors $\mathcal{X}, \mathcal{Y}$ in HTD, of equal dimension tree $\mathcal{T}$ and equal size $n_1 \times \cdots \times n_d$, defined by basis
  matrices $U_t^x, U_t^y$ and transfer tensors $\mathcal{B}_t^x, \mathcal{B}_t^y$.
Output: Inner product $\langle \mathcal{X}, \mathcal{Y} \rangle$.
  for $t \in \mathcal{L}(\mathcal{T})$ do
    Form $M_t = (U_t^x)^H U_t^y$.
  end for
  for $t \in \mathcal{N}(\mathcal{T})$ (visiting both child nodes before the parent node) do
    Form $M_t = (B_t^x)^H \big(M_{t_r} \otimes M_{t_l}\big)(B_t^y)$.
  end for
  Return $\langle \mathcal{X}, \mathcal{Y} \rangle = M_{t_\mathrm{root}}$.
```

In principle, the Euclidean norm of a tensor $\mathcal{X}$ in HTD can be calculated from $\|\mathcal{X}\|_2 = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle}$ by means of an inner product. However, it is well known that such an approach suffers from numerical instabilities and may introduce an error proportional to the square root of machine precision. A usually more accurate alternative is to first orthogonalize the HTD of $\mathcal{X}$ and then compute the norm. Note that the second part is trivial; it is easy to see that $\|\mathcal{X}\|_2 = \|\mathcal{B}_{1,2,\ldots,d}\|_F$ in the case of an orthogonalized HTD. The orthogonalization step makes the second approach slightly more expensive. The accuracy difference between the two approaches is illustrated in the following Matlab example.

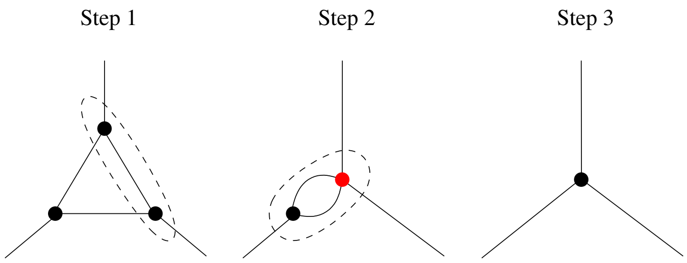


Figure 8: Example for which the elimination of a cycle creates a temporary node of larger degree.

```
x = htenrandn([5, 4, 6, 3]);
norm(x - x)/norm(x)
1.5355e-08
norm(orthog(x - x))/norm(x)
5.6998e-16
```

5.3 General Contraction of Tensors in HTD

In tensor network diagrams, a general contraction of two tensors in HTD is performed by connecting the corresponding pairs of dangling edges. This will create a tensor network with cycles, which need to be eliminated. This elimination is performed by successive contraction of tensors, similarly as above, until the network becomes a tree. This can only be organized efficiently if the maximum degree of all intermediate tensor networks does not become too large. Figure 8 provides a simple example where the maximum degree inevitably grows. To avoid this effect, we *assume* that the maximum degree remains at most 3. This also ensures that the eventually obtained tensor network corresponds to an HTD. In fact, the super node containing the aggregation of all cycles can be shown to have degree 2. Hence, it is natural to use this node as the root node in the HTD of the contraction.

Under the conditions mentioned above, the algorithm for performing a general contraction is a direct, but rather technical extension of Algorithm 2. For implementation details, we refer to [41, Sec. 3.5.2] and the source code of the function `ttt` in the `htucker` toolbox, which provides an implementation of the sketched algorithm.

`x = htenrandn([4, 2, 3]); y = htenrandn([3, 4, 2]);`
`z = ttt(x, y,[1 3],[2 1]);` Contracted product, connecting mode 1 of $\mathcal{X}$ with mode 2 of $\mathcal{Y}$, and mode 2 of $\mathcal{X}$ with mode 1 of $\mathcal{Y}$. Results in $\mathcal{Z} \in \mathbb{R}^{2\times 2}$.
`ttt(x, x, 1:3);` Inner product, identical with `innerprod(x, x)`.
`z = ttt(x, y);` Outer product $\mathcal{Z} \in \mathbb{R}^{4\times 2\times 3\times 3\times 4\times 2}$.

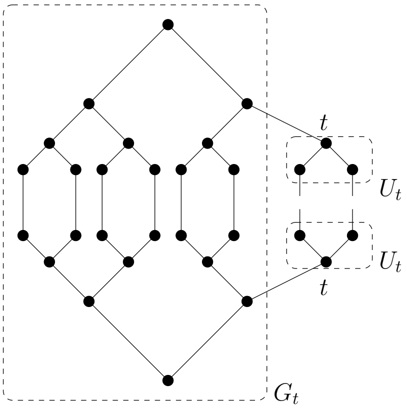


Figure 9: Reduced Gramian of a tensor of order 8. The reduced Gramian G_t corresponds to the left subnetwork encircled by a dashed line.

5.4 Reduced Gramians of a Tensor in HTD

An important application of contractions is the calculation of reduced Gramians, which are defined as follows. For every $t \in \mathcal{T}$, the matrix U_t defined in (6) contains a basis for the column span of the matricization $X^{(t)}$. Hence, there is a matrix V_t such that $X^{(t)} = U_t V_t^H$. The *reduced Gramian* at t is then defined as the Hermitian positive semi-definite matrix $G_t = V_t^H V_t \in \mathbb{C}^{r_t \times r_t}$.

Reduced Gramians are a central tool in the truncation of tensors. For example, they provide an efficient way to compute the singular values of $X^{(t)}$, which is used in the function `plot_sv`. From

$$X^{(t)}(X^{(t)})^H = U_t V_t^H V_t U_t^H = U_t G_t U_t^H \tag{14}$$

it follows that the singular values of $X^{(t)}$ are the square roots of the eigenvalues of the reduced Gramian G_t, provided that $U_t^H U_t = I_{r_t}$. This condition is always satisfied after the HTD has been orthogonalized.

In the following, we briefly discuss the computation of a reduced Gramian G_t via contraction. The standard, unreduced Gramian corresponds to the contracted product of $\mathcal{X}$ with itself along the modes $t^c = \{1, \dots, d\} \setminus t$: $\overline{\langle \mathcal{X}, \mathcal{X} \rangle_{t^c}}$, for which the matricization is given by (14), see also Figure 9 for an illustration. It can be seen from the figure that Gramians are closely related to contraction. In particular, this relation implies that G_t can be calculated by a sequence of matrix-tensor and tensor-tensor products, similar to Algorithm 2.

Typically, not the reduced Gramian at one node t is required, but reduced Gramians G_t for all nodes $t \in \mathcal{T}$. These can be computed simultaneously by exploiting relations between different reduced Gramians. The relationship between the reduced Gramians G_t and G_{t_r}, where t_r is the right child node of t, is illustrated in Figure 10 for the case of a general non-leaf node t. From the tensor network diagram, it can be seen that

$$\begin{aligned} G_{t_r} &= (B_t^{(\{1,3\})})^H (G_t \otimes U_{t_l}^H U_{t_l}) B_t^{(\{1,3\})}, \\ G_{t_l} &= (B_t^{(\{2,3\})})^H (G_t \otimes U_{t_r}^H U_{t_r}) B_t^{(\{2,3\})}. \end{aligned} \tag{15}$$

Formally setting $G_{t_{\text{root}}} = 1$, this defines a recursive algorithm for the calculation of all reduced Gramians, see Algorithm 3. An efficient algorithm for calculating $M_t = U_t^H U_t$ is already

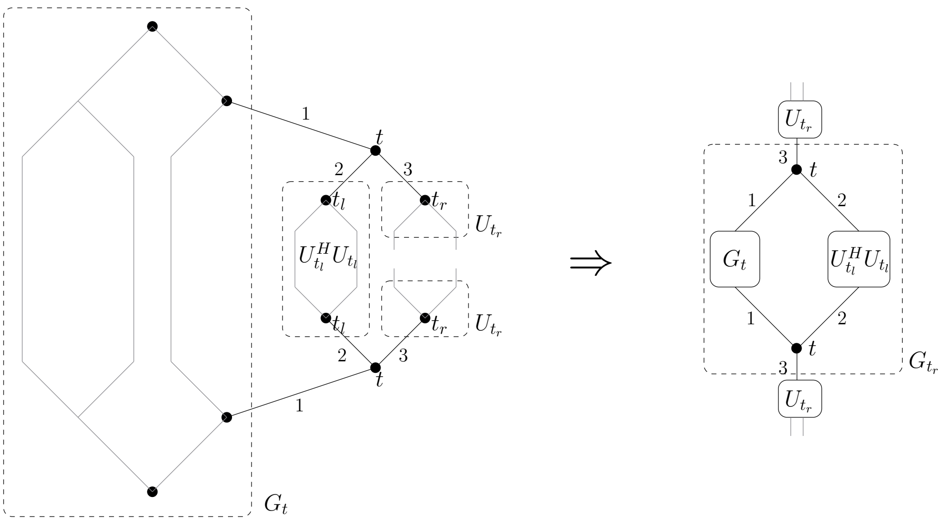


Figure 10: How to calculate G_{t_r} from G_t, U_{t_l} and B_t, where t is a general non-leaf node. The gray lines represent arbitrary subtrees.

described in Section 5.2and given in Algorithm 2. Note that all M_t are identity matrices for a tensor in orthogonalized HTD.

For a general HTD, Algorithm 3 requires $O(dnr^2 + (d-2)r^4)$ operations. For an orthogonalized HTD, this reduces to $O((d-2)r^4)$ operations (but orthogonalization itself requires $O(dnr^2 + (d-2)r^4)$ operations).

`G = gramians(x);` Reduced Gramians of $\mathcal{X}$ in cell array `G`, orthogonalizing the HTD of $\mathcal{X}$ if necessary.
`G = gramians_nonorthog(x);` Reduced Gramians of $\mathcal{X}$ in cell array `G`, without orthogonalizing.
`sv = singular_values(x);` Singular values of $\mathcal{X}$ in cell array `sv`.

Algorithm 3 Reduced Gramians of a tensor in HTD

```
Input: Basis matrices $U_t$ and transfer tensors $\mathcal{B}_t$ defining a general HTD of a tensor $\mathcal{X}$.
Output: Reduced Gramians $G_t$ for all $t \in \mathcal{T}$.
  for $t \in \mathcal{L}(\mathcal{T})$ do
    Form $M_t = (U_t)^H U_t$.
  end for
  for $t \in \mathcal{N}(\mathcal{T})$ (visiting both child nodes before the parent node) do
    Form $M_t = (B_t)^H \big(M_{t_r} \otimes M_{t_l}\big)(B_t)$.
  end for
  Set $G_{t_{\mathrm{root}}} = 1$.
  for $t \in \mathcal{N}(\mathcal{T})$ (visiting parent nodes before their children) do
    Form $G_{t_r} = (B_t^{(\{1,3\})})^H (G_t \otimes M_{t_l}) B_t^{(\{1,3\})}$.
    Form $G_{t_l} = (B_t^{(\{2,3\})})^H (G_t \otimes M_{t_r}) B_t^{(\{2,3\})}$.
  end for
```

6 Truncation of tensors

Truncation of tensors to HTD is one of the most important and most frequently used operations in `htucker`.

6.1 Truncation of explicit tensors

We start with the truncation of an explicitly given tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$. Although this situation is limited to small dimensions/sizes, it provides a gentle introduction and illustration of the general concepts. Truncation to HTD is done by successive projections to the subspaces $\operatorname{span}(W_t)$, which typically represent approximations to the column spaces of $X^{(t)} \in \mathbb{C}^{n_t \times n_{t^c}}$. For a subset $t \subset \{1, \ldots, d\}$, we define $n_t := \prod_{\mu \in t} n_\mu$. We require $W_t \in \mathbb{C}^{n_t \times r_t}$ to have orthonormal columns and define the orthogonal projections

$$\pi_t := W_t W_t^H. \tag{16}$$

In the following, we use the shorthand notation $\pi_t \mathcal{X}$ for $\pi_t \circ_t \mathcal{X}$. As shown in [19, Lemma 3.15], applying these projections in the correct order leads to a tensor in HTD, with hierarchical ranks bounded by r_t.

6.1.1 Root-to-leaves truncation

The simplest way to construct the projections π_t in (16) is to let each matrix W_t contain the r_t dominant left singular vectors of the corresponding matricization $X^{(t)}$. To obtain a tensor in HTD, $\widetilde{\mathcal{X}} \in \mathcal{H}\text{-Tucker}\big((r_t)_{t \in \mathcal{T}}\big)$, the projections need to be applied from the root node to the leaves. This is illustrated by the following example for order 4:

$$\begin{aligned} \operatorname{vec}(\widetilde{\mathcal{X}}) &= (W_4 W_4^H \otimes W_3 W_3^H \otimes W_2 W_2^H \otimes W_1 W_1^H)(W_{34} W_{34}^H \otimes W_{12} W_{12}^H) \operatorname{vec}(\mathcal{X}) \qquad (17) \\ &= (W_4 \otimes W_3 \otimes W_2 \otimes W_1)(\underbrace{[W_4^H \otimes W_3^H] W_{34}}_{=:B_{34}} \otimes \underbrace{[W_2^H \otimes W_1^H] W_{12}}_{=:B_{12}}) \underbrace{([W_{34}^H \otimes W_{12}^H] \operatorname{vec}(\mathcal{X}))}_{=:B_{1234}}. \end{aligned}$$

The computation for the general case is described in Algorithm 4. Note that the HTD of the resulting tensor is not orthogonalized, only the matrices in the leaf nodes have orthonormal columns. Setting $n = \max_t n_t$, the computational complexity of root-to-leaves truncation is of order $dn^{3d/2}$ in the case of a balanced tree. An efficient way to calculate W_t is through an eigenvalue decomposition of the Gramian: $X^{(t)}(X^{(t)})^H = W_t \Sigma^2 W_t^H$. The resulting tensor $\widetilde{\mathcal{X}}$ satisfies the following error bound [19, Theorem 3.11]:

$$\big\|\mathcal{X} - \widetilde{\mathcal{X}}\big\|_2 \le \sqrt{\sum_{t \in \mathcal{T}'} \delta_{r_t}(X^{(t)})^2} \le \sqrt{2d-3}\, \big\|\mathcal{X} - \mathcal{X}_{\text{best}}\big\|_2, \tag{18}$$

where $\mathcal{X}_{\text{best}}$ represents the best approximation of $\mathcal{X}$ in $\mathcal{H}\text{-Tucker}((r_t)_{t \in \mathcal{T}})$, $\mathcal{T}' := \mathcal{T} \backslash \{t_{\text{root}}, t_{\text{child}}\}$ where t_{child} is a child of the root node t_{root}, and

$$\delta_{r_t}(X^{(t)})^2 := \sum_{j=r_t+1}^{n_t} \sigma_j(X^{(t)})^2. \tag{19}$$

Algorithm 4 Root-to-leaves truncation of a tensor

```
Input: Tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, dimension tree $\mathcal{T}$ and desired hierarchical ranks $(r_t)_{t\in\mathcal{T}}$ of the truncated
   tensor.
Output: Tensor $\widetilde{\mathcal{X}}$ in HTD, with $\operatorname{rank}(\widetilde{X}^{(t)}) \le r_t$ for all $t \in \mathcal{T}$.
   for $t \in \mathcal{T}$ (visiting both child nodes before the parent node) do
      if $t \in \mathcal{L}(\mathcal{T})$ then
         Compute singular value decomposition $X^{(t)} =: \widehat{U}_t \widehat{\Sigma}_t \widehat{V}_t^H$.
         Set $U_t := \widehat{U}_t(:, 1 : r_t)$.
      else if $t$ is the root node then
         Form $B_t := (U_{t_r}^H \otimes U_{t_l}^H) \operatorname{vec}(\mathcal{X})$.
      else
         Compute singular value decomposition $X^{(t)} =: \widehat{U}_t \widehat{\Sigma}_t \widehat{V}_t^H$.
         Set $U_t := \widehat{U}_t(:, 1 : r_t)$.
         Form $B_t := (U_{t_r}^H \otimes U_{t_l}^H) U_t$.
      end if
   end for
```

Remark 6.1. *The error bound (18) allows us to choose the hierarchical ranks $(r_t)_{t\in\mathcal{T}}$ such that a certain error bound is satisfied:*

$$\big\|\mathcal{X} - \widetilde{\mathcal{X}}\big\|_2 \le \epsilon_{abs}, \quad \textit{choose } r_t \textit{ s.t. } \delta_{r_t}(X^{(t)}) \le \frac{\epsilon_{abs}}{\sqrt{2d-3}} \quad \forall t \in \mathcal{T} \setminus \{t_{root}\},$$

$$\big\|\mathcal{X} - \widetilde{\mathcal{X}}\big\|_2 \le \epsilon_{rel} \|\mathcal{X}\|_2, \quad \textit{choose } r_t \textit{ s.t. } \delta_{r_t}(X^{(t)}) \le \frac{\epsilon_{rel} \|\mathcal{X}\|_2}{\sqrt{2d-3}} \quad \forall t \in \mathcal{T} \setminus \{t_{root}\}.$$

Similar adaptive choices of the hierarchical ranks are possible for all other truncation methods discussed in the following.

`opts.max_rank = 10;` maximal rank at truncation, mandatory argument.
`opts.rel_eps = 1e-6;` maximal relative truncation error, optional argument.
`opts.abs_eps = 1e-6;` maximal absolute truncation error, optional argument.
Condition `max_rank` takes precedence over `rel_eps` and `abs_eps`.
`y = htensor.truncate_rtl(x, opts);` takes a MATLAB multidimensional array and returns the truncation to lower rank HTD.

6.1.2 Leaves-to-root truncation

Root-to-leaves truncation is very costly, the most expensive part being the computation of the singular value decomposition of every $X^{(t)} \in \mathbb{C}^{n_t \times n_{t^c}}$, where both n_t and n_{t^c} can become very large. Leaves-to-root truncation can be considerably faster. To illustrate the idea, consider a fourth order tensor. For each leaf node t, we define W_t to contain the r_t dominant left singular vectors of $X^{(t)}$ and set

$$\operatorname{vec}(\mathcal{C}_1) := (W_4^H \otimes W_3^H \otimes W_2^H \otimes W_1^H) \operatorname{vec}(\mathcal{X}), \quad \mathcal{C}_1 \in \mathbb{C}^{r_1 \times r_2 \times r_3 \times r_4}.$$

In the next step, we consider the nodes $t = \{1,2\}, t = \{3,4\}$, and define S_t to contain the r_t dominant left singular vectors of $C_1^{(t)}$:

$$\operatorname{vec}(\mathcal{C}_0) = (S_{34}^H \otimes S_{12}^H) \operatorname{vec}(\mathcal{C}_1), \quad \mathcal{C}_0 \in \mathbb{C}^{r_{12} \times r_{34}}.$$

The resulting tensor is in HTD:

$$\operatorname{vec}(\widetilde{\mathcal{X}}) = (W_4 \otimes W_3 \otimes W_2 \otimes W_1)(S_{34} \otimes S_{12}) \operatorname{vec}(\mathcal{C}_0).$$

For the case of a general tensor, see Algorithm 5. Note that $h(\mathcal{T})$ denotes the height of $\mathcal{T}$, and the set of all nodes with distance ℓ ($\ell = 0, \ldots, h(\mathcal{T})$) to the root node is denoted by $\mathcal{T}_\ell$.

Algorithm 5 can be interpreted in terms of projections $W_t W_t^H$, with the definition $W_t = (W_{t_r} \otimes W_{t_l}) S_t$. As the subspaces defined by W_t are nested, it can be seen that all projections π_t commute, see also Lemma B.1. Combined with [19, Lemma 3.15], this shows that the resulting tensor is in $\mathcal{H}$-Tucker$((r_t)_{t\in\mathcal{T}})$. The computational complexity of leaves-to-root truncation is $O(dn^{d+1})$, which is a significant reduction compared to the root-to-leaves method, while the error bound (18) still holds, see Lemma B.2. Moreover, the resulting tensor $\widetilde{\mathcal{X}}$ is in orthogonalized HTD.

Algorithm 5 Leaves-to-root truncation of a tensor

```
Input: Tensor $\mathcal{X} \in \mathbb{C}^{n_1\times\cdots\times n_d}$, dimension tree $\mathcal{T}$ and desired hierarchical ranks $(r_t)_{t\in\mathcal{T}}$ of the truncated
  tensor.
Output: Tensor $\widetilde{\mathcal{X}}$ in HTD, with $\operatorname{rank}(\widetilde{X}^{(t)}) \le r_t$ for all $t \in \mathcal{T}$.
  for $t \in \mathcal{L}(\mathcal{T})$ do
    Compute singular value decomposition $X^{(t)} =: \widehat{U}_t \widehat{\Sigma}_t \widehat{V}_t^H$.
    Set $\widetilde{U}_t := \widehat{U}_t(:, 1 : r_t)$.
  end for
  Form $\mathcal{C}_{L-1} := (\widetilde{U}_d^H \otimes \cdots \otimes \widetilde{U}_1^H)\mathcal{X}$.
  for $\ell = h(\mathcal{T}) - 1, \ldots, 1$ do
    for $t \in \mathcal{T}_\ell \setminus \mathcal{L}(\mathcal{T})$ do
      Compute singular value decomposition $C_\ell^{(t)} =: \widehat{S}_t \widehat{\Sigma}_t \widehat{V}_t^H$.
      Set $B_t := \widehat{S}_t(:, 1 : r_t)$.
    end for
    Form $\mathcal{C}_{\ell-1} := \Big(\prod_{t\in\mathcal{T}_\ell} B_t^H \circ_t\Big)\mathcal{C}_\ell$.
  end for
  Set $B_{t_{\mathrm{root}}} := \operatorname{vec}(\mathcal{C}_0)$.
```

`x = htensor.truncate_ltr(x, opts);` takes a MATLAB multidimensional array and returns the truncation to lower rank HTD.

6.2 Truncation of $\mathcal{H}$-Tucker decomposition to lower rank

The truncation of a tensor which is already given in HTD to a tensor in HTD of lower rank is an essential operation in most algorithms based on this format. In Section 6.2.1, we describe an efficient method for performing such a truncation. This will be the method of choice for general tensors. However, for structured tensors resulting, e.g., from the addition of several tensors in HTD, a different approach described in Section 6.2.2 is preferable.

6.2.1 Truncation of a tensor in HTD

Truncation of a tensor $\mathcal{X}$ in HTD can be performed by a fairly straightforward adaptation of the root-to-leaves method. For this purpose, we recall that Section 5.4 describes an efficient method for computing the reduced Gramians G_t in the decomposition

$$X^{(t)}(X^{(t)})^H = U_t G_t U_t^H,$$

where U_t has orthonormal columns and is implicitly represented as illustrated in Figure 10. After orthogonalizing the HTD of $\mathcal{X} \in \mathcal{H}\text{-Tucker}\big((k_t)_{t\in\mathcal{T}}\big)$ and calculating the reduced Gramians, we compute an orthonormal basis $S_t \in \mathbb{C}^{k_t\times r_t}$ for the r_t dominant eigenvectors of the symmetric matrix G_t. As above, we define $W_t := U_t S_t$ and obtain the truncated tensor $\widetilde{\mathcal{X}}$ from subsequent application of the projections $\pi_t = W_t W_t^H$.

To illustrate how these projections can be applied to a tensor in HTD, let us consider the example of a tensor $\mathcal{X}$ of order 4:

$$\begin{aligned}
\operatorname{vec}(\widetilde{\mathcal{X}}) &= (W_4W_4^H \otimes W_3W_3^H \otimes W_2W_2^H \otimes W_1W_1^H)(W_{34}W_{34}^H \otimes W_{12}W_{12}^H)\operatorname{vec}(\mathcal{X}) \\
&= (U_4S_4 \otimes U_3S_3 \otimes U_2S_2 \otimes U_1S_1)\cdots \\
&\qquad \cdots\big(\underbrace{(S_4^H \otimes S_3^H)B_{34}S_{34}}_{=:\widetilde{B}_{34}} \otimes \underbrace{(S_2^H \otimes S_1^H)B_{12}S_{12}}_{=:\widetilde{B}_{12}}\big)\underbrace{(S_{34}^H \otimes S_{12}^H)B_{1234}}_{=:\widetilde{B}_{1234}}.
\end{aligned}$$

Hence, an HTD for $\widetilde{\mathcal{X}}$ is obtained by updating the leaf matrices $\widetilde{U}_1 := U_1S_1, \ldots, \widetilde{U}_4 := U_4S_4$, and the transfer matrices. Note that the matrices W_t are never calculated explicitly.

This update can be extended to the general case in a direct way:

$$\begin{array}{rcll}
\widetilde{U}_t &:=& U_tS_t & \forall t \in \mathcal{L}(\mathcal{T}), \\
\widetilde{B}_t &:=& (S_{t_r}^H \otimes S_{t_l}^H)B_tS_t & \forall t \in \mathcal{N}(\mathcal{T}).
\end{array}$$

Note that the update for the root node $t \equiv t_{\text{root}}$ simplifies to $\widetilde{B}_t := (S_{t_r}^H \otimes S_{t_l}^H)B_t$.

Algorithm 6 Truncation of a tensor in HTD

```
Input: Tensor $\mathcal{X}$ in HTD and desired hierarchical ranks $(r_t)_{t\in\mathcal{T}}$ of the truncated tensor.
Output: Tensor $\widetilde{\mathcal{X}}$ in HTD, with $\operatorname{rank}(\widetilde{X}^{(t)}) \le r_t \quad \forall t \in \mathcal{T}$.
  Orthogonalize $\mathcal{X}$ (as described in Algorithm 1).
  Calculate reduced Gramians $G_t$ (as described in Algorithm 3).
  for $t \in \mathcal{T} \setminus \{t_{\text{root}}\}$ do
    Compute symmetric eigenvalue decomposition $G_t =: \widehat{S}_t\widehat{\Sigma}_t^2\widehat{S}_t^H$.
    Set $S_t := \widehat{S}_t(:, 1 : r_t)$.
  end for
  $S_{t_{\text{root}}} = 1$
  for $t \in \mathcal{L}(\mathcal{T})$: Set $\widetilde{U}_t := U_tS_t$.
  for $t \in \mathcal{N}(\mathcal{T})$: Set $\widetilde{B}_t := (S_{t_r}^H \otimes S_{t_l}^H)B_tS_t$.
```

Algorithm 6, which describes this procedure, is implemented in `truncate_std(x, opts)`, and requires $O(dnr^2 + dr^4)$ operations. As this algorithm is mathematically identical to the explicit root-to-leaves algorithm described in Section 6.1.1, the error bound (18) holds. Note that the resulting tensor $\widetilde{\mathcal{X}}$ is not in orthogonalized HTD.

`xt = truncate_std(x, opts);` takes an `htensor` $\mathcal{X}$ and returns a truncated `htensor` $\widetilde{\mathcal{X}}$.

6.2.2 Truncation of a tensor in HTD without initial orthogonalization

The method of truncation introduced in the last section represents the default method for truncating a tensor in HTD. However, in certain situations, it can be beneficial to exploit additional structure in the HTD. For example, a tensor resulting from addition of tensors in HTD has block diagonal transfer tensors. In the standard method of truncation, such structures are immediately destroyed by the initial orthogonalization step. In the following, we discuss a method that avoids this step.

In a first step, the reduced Gramians G_t in the decomposition

$$X^{(t)}(X^{(t)})^H = U_t G_t U_t^H$$

are calculated without the initial orthogonalization. Note, however, that the singular value decomposition of $X^{(t)}$ cannot be computed directly from G_t, as the columns of U_t are not orthonormal.

In a second step, the proposed method successively orthonormalizes the matrices U_t. Let us first consider the leaf nodes t, for which we compute the r_t dominant left singular vectors of $X^{(t)}$ as follows: Compute the QR decomposition $U_t =: Q_t R_t$, and determine the matrix S_t containing the r_t dominant eigenvectors of $R_t G_t R_t^H$. Then the projection $\pi_t = W_t W_t^H$, with $W_t = Q_t S_t$, is applied to $\mathcal{X}$. Note that the updated leaf nodes $\widetilde{U}_t := Q_t S_t$ are orthonormal.

Non-leaf nodes are processed in a similar manner with a recursive algorithm, traversing the tree such that every parent node is visited after its child nodes. Assume we are at node t, and let $\widetilde{U}_t$ account for all updates from previous operations on the descendants of t. Based on the original decomposition $X^{(t)} = U_t V_t^H$, we set $\widetilde{X}_t^{(t)} := \widetilde{U}_t V_t^H$ and observe that the corresponding Gramian takes the form

$$\widetilde{X}_t^{(t)}(\widetilde{X}_t^{(t)})^H = \widetilde{U}_t G_t \widetilde{U}_t^H.$$

To orthogonalize $\widetilde{U}_t = (\widetilde{U}_{t_r} \otimes \widetilde{U}_{t_l})\widetilde{B}_t$, it is sufficient to calculate the QR decomposition of $\widetilde{B}_t = (S_{t_r}^H R_{t_r} \otimes S_{t_l}^H R_{t_l})B_t$, as the columns of $\widetilde{U}_{t_l}$ and $\widetilde{U}_{t_r}$ are orthonormal. Then, we calculate the r_t dominant left singular vectors of $\widetilde{X}_t^{(t)}$ as in the case of the leaf nodes.

A more detailed description of truncation to HTD without initial orthogonalization can be found in Algorithm 7. The result of this algorithm satisfies practically the same error bound as in (18), see also Lemma B.3.

The computational complexities of Algorithm 6 and Algorithm 7 are the same, but the latter requires more operations.

`xt = truncate_nonorthog(x, opts);` takes an `htensor` $\mathcal{X}$ and returns a truncated `htensor` $\widetilde{\mathcal{X}}$.

6.3 Combined Addition and Truncation

As explained in Section 4.2, the addition of tensors in HTD leads to a significant growth of the hierarchical ranks. For example, the sum of s tensors of hierarchical ranks r has hierarchical ranks sr. Truncation of this tensor back to hierarchical rank r requires $O(dns^2r^2 + ds^4r^4)$ operations, which is too expensive unless s is very small.

Algorithm 7 Truncation of a tensor in HTD without initial orthogonalization

```
Input: Tensor $\mathcal{X}$ in HTD and desired hierarchical ranks $(r_t)_{t\in\mathcal{T}}$ of the truncated tensor.
Output: Tensor $\widetilde{\mathcal{X}}$ in HTD, with $\operatorname{rank}(\widetilde{X}^{(t)}) \le r_t$.
  Calculate reduced Gramians $G_t$ (as described in Algorithm 3).
  for $t \in \mathcal{L}(\mathcal{T})$ do
    Compute QR decomposition $U_t =: Q_t R_t$.
    Compute symmetric eigenvalue decomposition $R_t G_t R_t^H =: \widehat{S}_t \widehat{\Sigma}_t^2 \widehat{S}_t^H$.
    Set $S_t := \widehat{S}_t(:, 1 : r_t)$.
    Form $U_t := Q_t S_t$.
  end for
  for $t \in \mathcal{N}(\mathcal{T}) \setminus \{t_{\text{root}}\}$ (visiting both child nodes before the parent node) do
    Compute QR decomposition $(S_{t_r}^H R_{t_r} \otimes S_{t_l}^H R_{t_l}) B_t =: Q_t R_t$.
    Compute symmetric eigenvalue decomposition $R_t G_t R_t^H =: \widehat{S}_t \widehat{\Sigma}_t^2 \widehat{S}_t^H$.
    Set $S_t := \widehat{S}_t(:, 1 : r_t)$.
    Form $B_t := Q_t S_t$.
  end for
  Form $B_{t_{\text{root}}} := (S_{t_r}^H R_{t_r} \otimes S_{t_l}^H R_{t_l}) B_{t_{\text{root}}}$ with child nodes $t_l$ and $t_r$ of $t_{\text{root}}$.
```

A cheaper alternative is to add the s tensors successively and truncate immediately after each addition. After setting $\widetilde{\mathcal{Y}}_1 := \mathcal{X}_1$, we compute for $j = 1, \dots, s-1$:

$$\begin{aligned} &\text{Form } \mathcal{Y}_{j+1} := \widetilde{\mathcal{Y}}_j + \mathcal{X}_j. \\ &\text{Truncate } \mathcal{Y}_{j+1} \text{ to } \widetilde{\mathcal{Y}}_{j+1}. \end{aligned} \tag{20}$$

However, one can easily construct examples for which this scheme suffers from severe cancellation (see `example_cancellation.m` in the toolbox).

To avoid cancellation and still increase efficiency, we propose to apply truncation without initial orthogonalization (Section 6.2.2) directly to the sum of tensors and exploit the block diagonal structures illustrated in Figure 5. This results in significant savings when calculating the reduced Gramians. Computing the reduced Gramians G_t at a non-leaf node requires only $O(s^2r^4)$ instead of $O(s^4r^4)$ operations. Hence, the computational cost of the whole addition and truncation process reduces to $O(dns^2r^2 + ds^2r^4 + ds^3r^3)$.

With a numerical experiment we examine the execution time required for the addition and truncation of s random tensors of order $d = 5$, with size $n = 500$ and rank $r = 20$. The number of summands s varies between 2 and 10 (see Figure 11). This numerical experiment was performed in Matlab, version 7.6, on an Intel Xeon CPU E31225 with 3 GHz and 6MB L2 Cache and 8GB RAM. The execution time of the new method increases proportionally with s^2, indicating that the term s^3r^3 does not dominate the cost for this rather typical setting. Note that the execution time of the new method is relatively high for small s. However, this only reflects the additional overhead of this method in a Matlab implementation; even for $s = 2$, the operation count of the new method is smaller compared to applying standard truncation to the sum of tensors.

`xt = truncate_sum({x1, x2, x3}, opts);` takes `htensor` objects $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3$ and returns a truncated `htensor` $\widetilde{\mathcal{X}} \approx \mathcal{X}_1 + \mathcal{X}_2 + \mathcal{X}_3$.

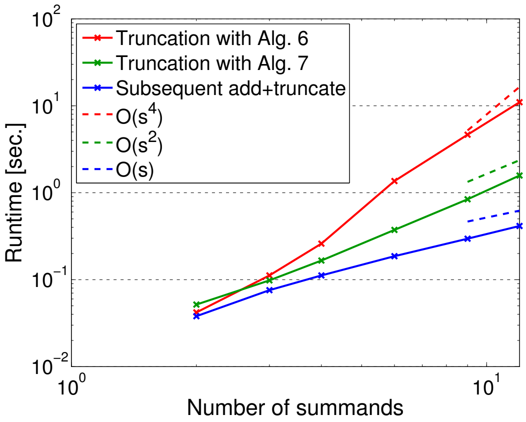


Figure 11: Execution times for truncating a sum of tensors in HTD. Red: Standard truncation of the sum. Green: Truncation of the sum with using combined addition and truncation, as described in Section 6.3. Blue: Successive addition and truncation, see (20).

7 Elementwise multiplication

The elementwise multiplication of two tensors is an important operation in connection with function-related tensors and can be performed efficiently for tensors in HTD. In particular, this operation corresponds to the pointwise multiplication $f(\xi_1,\dots,\xi_d) = f_1(\xi_1,\dots,\xi_d)f_2(\xi_1,\dots,\xi_d)$ of the functions f_1, f_2 represented by the two tensors.

For illustration, we first consider the elementwise multiplication of two low-rank matrices $X = U^x S^x (V^x)^H$ and $Y = U^y S^y (V^y)^H$, with $S^x, S^y \in \mathbb{C}^{r\times r}$. Then the elementwise product (also called Hadamard product) can be written as

$$\begin{aligned}(X \star Y)_{i,j} := X_{ij}Y_{ij} &= \sum_{\alpha,\beta,\gamma,\delta} U^x_{i,\alpha}U^y_{i,\gamma}S^x_{\alpha,\beta}S^y_{\gamma,\delta}V^x_{j,\beta}V^y_{j,\delta} \\ &= (U^x \odot^T U^y)(S^x \otimes S^y)(V^x \odot^T V^y)^H,\end{aligned}$$

where $\odot^T$ denotes a transposed variant of the Khatri-Rao product [32]. More specifically, for a matrix $A \in \mathbb{C}^{n\times k}$ with rows a_j^T and a matrix $B \in \mathbb{C}^{n\times r}$ with rows b_j^T, we define

$$A \odot^T B = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{bmatrix} \odot^T \begin{bmatrix} b_1^T \\ b_2^T \\ \vdots \\ b_n^T \end{bmatrix} := \begin{bmatrix} a_1^T \otimes b_1^T \\ a_2^T \otimes b_2^T \\ \vdots \\ a_n^T \otimes b_n^T \end{bmatrix} \in \mathbb{C}^{n\times kr}.$$

Note that $A \odot^T B = (A^T \odot B^T)^T$, where $\odot$ is the usual Khatri-Rao product.

For the elementwise multiplication of two tensors $\mathcal{X}, \mathcal{Y}$ in HTD, the same technique can be used to construct the leaf matrices U_t and transfer tensors $\mathcal{B}_t$ for an HTD of $\mathcal{X} \star \mathcal{Y}$:

$$U_t = U_t^x \odot^T U_t^y \text{ for leaf nodes } t \text{ and } \mathcal{B}_t = \mathcal{B}_t^x \otimes \mathcal{B}_t^y \text{ for non-leaf nodes } t,$$

where $\otimes$ represents a direct generalization of the Kronecker product to tensors: For two tensors $\mathcal{X} \in \mathbb{C}^{n_1\times\cdots\times n_d}, \widehat{\mathcal{X}} \in \mathbb{C}^{\widehat{n}_1\times\cdots\times\widehat{n}_d}$ we define $(\mathcal{X} \otimes \widehat{\mathcal{X}})_{J_1,\dots,J_d} := \mathcal{X}_{i_1,\dots,i_d}\widehat{\mathcal{X}}_{j_1,\dots,j_d}$, with $J_\mu = (i_\mu -$

$1)\widehat{n}_\mu + j_\mu$. As a consequence, the hierarchical rank r_t of $\mathcal{X} \star \mathcal{Y}$ is the product $r_t^x r_t^y$ of the hierarchical ranks of $\mathcal{X}$ and $\mathcal{Y}$.

It would be useful to avoid the rank growth of $\mathcal{X} \star \mathcal{Y}$ and directly calculate a truncated version, similarly as for the sum of s tensors in Section 6.3. Unfortunately, it is not clear how to transfer the ideas from Section 6.3 to the elementwise product; the Kronecker structure of the transfer tensors does not lead in an obvious way to a reduction of computational or storage cost. However, we can exploit the fact that the elementwise product is contained in the Kronecker product. More specifically, there is a $(0,1)$-matrix $J_n \in \mathbb{R}^{n^2 \times n}$ with orthonormal columns such that

$$a \star b = J_n^H (a \otimes b),$$

for any two vectors $a, b \in \mathbb{C}^n$. This extends in a direct fashion to tensors:

$$J^H(\mathcal{X} \otimes \mathcal{Y}) := (J_{n_d}^H \otimes \cdots \otimes J_{n_1}^H)(\mathcal{X} \otimes \mathcal{Y}) = \mathcal{X} \star \mathcal{Y}. \tag{21}$$

Hence, we can implicitly form the Kronecker product $\mathcal{X} \otimes \mathcal{Y}$ in HTD and extract the elementwise product after truncation.

The HTD of the Kronecker product $\mathcal{X} \otimes \mathcal{Y}$ of two tensors $\mathcal{X}, \mathcal{Y}$ in HTD is particularly simple:

$$\begin{array}{rcll} U_t & := & U_t^x \otimes U_t^y & \quad \forall t \in \mathcal{L}(\mathcal{T}), \\ B_t & := & B_t^x \otimes B_t^y & \quad \forall t \in \mathcal{N}(\mathcal{T}). \end{array}$$

This implies that the reduced Gramians have Kronecker structure, $G_t = G_t^x \otimes G_t^y$, as well as their singular value decompositions used in the standard truncation to lower rank. Consequently, this allows for a particularly efficient HTD truncation $\mathcal{Z}$ of $\mathcal{X} \otimes \mathcal{Y}$. Using (21), the extracted tensor $J^H \mathcal{Z}$ represents an approximation of $\mathcal{X} \star \mathcal{Y}$ satisfying the error bound

$$\left\| \mathcal{X} \star \mathcal{Y} - J^H \mathcal{Z} \right\|_2 = \left\| J^H(\mathcal{X} \otimes \mathcal{Y} - \mathcal{Z}) \right\|_2 \leq \left\| \mathcal{X} \otimes \mathcal{Y} - \mathcal{Z} \right\|_2 \leq \epsilon_{abs}.$$

Although the hierarchical ranks of $J^H \mathcal{Z}$ are typically much smaller compared to $\mathcal{X} \star \mathcal{Y}$, the error bound above is far from being sharp. It is therefore recommended to truncate $J^H \mathcal{Z}$ again after the extraction.

`z = x .* y` elementwise product of $\mathcal{X}$ and $\mathcal{Y}$.
`z = elem_mult(x, y, opts)` approximate elementwise product, with `opts` defined as in `truncate`.

8 HTD of linear operators on tensors

The use of tensors in PDE-related applications often requires the efficient storage and application of a linear operator to tensors. In many cases, such a linear operator can be written as a short sum of Kronecker products:

$$\operatorname{vec}(\mathcal{A}(\mathcal{X})) = \sum_{j=1}^{R} \left(A_j^{(d)} \otimes \cdots \otimes A_j^{(1)} \right) \operatorname{vec}(\mathcal{X}), \qquad \text{with } A_j^{(\mu)} \in \mathbb{C}^{m_\mu \times n_\mu}. \tag{22}$$

For example, a discretized Laplace operator in d dimensions takes this form with $R = d$, see Example 8.1 below. The operation (22) can be implemented by first applying μ-mode matrix products (`ttm`) and then using an algorithm for computing a sum of tensors in HTD, see Section 6.3. In general, this is a reasonable approach. However, for particular linear operators, a much more efficient scheme can be devised.

This scheme is based on interpreting a linear operator as a tensor, an idea which goes back to the computational physics community [40]. For example, the operator in (22) can be vectorized into

$$\widetilde{\mathcal{A}} = \sum_{j=1}^{R} \operatorname{vec}\left(A_j^{(d)}\right) \otimes \cdots \otimes \operatorname{vec}\left(A_j^{(1)}\right).$$

Note that this format is a CP decomposition. More generally, there is an isomorphism

$$\Psi : \mathfrak{L}\big(\mathbb{C}^{n_1\times\cdots\times n_d}, \mathbb{C}^{m_1\times\cdots\times m_d}\big) \to \mathbb{C}^{n_1 m_1\times\cdots n_d m_d},$$

which takes the matrix representation $\mathcal{A}_M \in \mathbb{C}^{(m_1\cdots m_d)\times(n_1\cdots n_d)}$ of a linear operator $\mathcal{A}$, and permutes and reshapes its entries into a tensor $\widetilde{\mathcal{A}} = \Psi(\mathcal{A})$ of order d.

The tensor $\widetilde{\mathcal{A}} = \Psi(\mathcal{A})$ can now be approximated in HTD by the methods described in this paper. When applying a linear operator implicitly represented as a tensor in HTD with leaf bases $U_t^{\mathcal{A}} \in \mathbb{C}^{m_t n_t \times s_t}$ and transfer tensors $\mathcal{B}_t^{\mathcal{A}}$, it is convenient to reinterpret the columns of the leaf bases as matrices $A_t^{(j)}$:

$$U_t^{\mathcal{A}}(:,j) = \operatorname{vec}\big(A_t^{(j)}\big).$$

Then the application of $\mathcal{A}$ to a tensor $\mathcal{X}$ of conforming size in HTD with hierarchical ranks r_t again results in an HTD with

$$U_t = \left[A_t^{(1)} U_t^x, \ldots, A_t^{(s_t)} U_t^x\right], \quad \mathcal{B}_t = \mathcal{B}_t^{\mathcal{A}} \otimes \mathcal{B}_t^{\mathcal{X}}.$$

Hence, the hierarchical ranks grow to $s_t r_t$, which illustrates the importance of keeping the hierarchical ranks s_t of $\mathcal{A}$ low.

The sesquilinear product $\langle \mathcal{X}, \mathcal{Y}\rangle_{\mathcal{A}}$ can be computed without applying $\mathcal{A}$ to one of the tensors, by interpreting the product as a tensor network and contracting the network. This amounts to a computational complexity of $O(d(sn^2r + snr^2 + 3s^2r^4 + s^3r^2))$, where $\mathcal{X}, \mathcal{Y}$ are in HTD with sizes n and hierarchical ranks r, while $\widetilde{\mathcal{A}}$ is in HTD with hierarchical ranks s.

The composition of two linear operators (i.e., the multiplication of the corresponding matrix representations) in HTD can be calculated in a similar way as the application of a linear operator in HTD.

`y = apply_mat_to_vec(A, x)` returns $\mathcal{Y} = \mathcal{A}(\mathcal{X})$ for a linear operator $\mathcal{A}$ in HTD.
`s = innerprod_mat(x, y, A)` returns $s = \langle \mathcal{X}, \mathcal{Y}\rangle_{\mathcal{A}}$.
`C = apply_mat_to_mat(A, B, p)` returns $\mathcal{C} = \mathcal{A} \circ \mathcal{B}$ for two linear operators $\mathcal{A} \in \mathfrak{L}\big(\mathbb{C}^{p_1\times\cdots\times p_d}, \mathbb{C}^{n_1\times\cdots\times n_d}\big)$ and $\mathcal{B} \in \mathfrak{L}\big(\mathbb{C}^{m_1\times\cdots\times m_d}, \mathbb{C}^{p_1\times\cdots\times p_d}\big)$.

Note that a truncation of a linear operator $\mathcal{A}$ to HTD produces a quasi-optimal approximation in the Frobenius norm. Therefore, its effect on the smallest eigenvalues of $\mathcal{A}$ is likely to be significant and its use for the direct solution of linear systems questionable. However, this approximation may still be useful in the construction of preconditioners, and there are some notable cases for which an exact representation in HTD of low rank is possible.

Example 8.1. *A discretized Laplace-like operator of the form*

$$A^{(d)} \otimes I \otimes \cdots \otimes I \;+\; I \otimes A^{(d-1)} \otimes I \otimes \cdots \otimes I \;+\; \cdots \;+\; I \otimes \cdots \otimes I \otimes A^{(1)} \tag{23}$$

can be represented exactly in HTD with hierarchical rank 2 *for any dimension tree* $\mathcal{T}$*:*

$$U_t = \left[\operatorname{vec}(I), \operatorname{vec}(A^{(t)})\right] \quad \forall t \in \mathcal{L}(\mathcal{T})$$

$$B_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \forall t \in \mathcal{N}(\mathcal{T}) \setminus \{t_{root}\}, \qquad B_{t_{root}} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

A similar decomposition has been proposed for the TT decomposition in [27].

9 Examples

In the following, we will show two examples for the use of the described `htucker` toolbox. One relatively simple example is concerned with a tensor containing function samples, and another example is concerned with a tensor containing solutions to a parameter-dependent partial differential equation.

Example 9.1. The tensor $\mathcal{X}$ is defined to contain all function values of the d-variate function

$$f(\xi_1, \ldots, \xi_d) = \frac{1}{\xi_1 + \cdots + \xi_d}$$

on a uniform tensor grid in $[1, 10]^d$. The following commands create this tensor as a standard Matlab multidimensional array (see `examples/example_sum_reciproc.m`):

```
n = 50; d = 4;
xi = linspace(1, 10, n)';
xil = xi*ones(1, n^(d-1)); xil = reshape(xil, n*ones(1, d));
xisum = xil;
for ii=2:d
  xisum = xisum + permute(xil, [ii, 2:ii-1, 1, ii+1:d]);
end
x = 1./xisum;
```

We then truncate this full tensor $\mathcal{X}$ to HTD:

```
opts.max_rank = 10;  opts.rel_eps = 1e-5;
x_ht = truncate(x, opts);
rel_err = norm(x(:) - x_ht(:))/norm(x(:))
> 1.3403e-06
```

Note that this approach is limited to small values of d, as $\mathcal{X}$ is constructed explicitly. An alternative approach relies on the following identity

$$\frac{1}{\xi_1 + \cdots + \xi_d} = \int_0^\infty \exp(-t \cdot (\xi_1 + \cdots + \xi_d))dt \approx \sum_{j=-M}^{M} \omega_j \prod_{\mu=1}^{d} e^{-\alpha_j \xi_\mu}.$$

Suitable coefficients α_j, ω_j are described in [17, 20]. Sampling the function on the right-hand side directly results in a CP decomposition of rank $2M + 1$. This can then be converted to the HTD format, where it can be truncated further, resulting in much smaller ranks; from tensor rank 51 to hierarchical rank 5 in our example:

```
M = 25; j = (-M:M);
xMin = d*min(xi);
hst = pi/sqrt(M);
alpha = -2*log(exp(j*hst)+sqrt(1+exp(2*j*hst)))/xMin;
omega = 2*hst./sqrt(1+exp(-2*j*hst))/xMin;

x_cp = cell(1, d);
for ii=1:d, x_cp{ii} = exp(xi*alpha); end
x_cp{1} = x_cp{1}*diag(omega);
x_cp = htensor(x_cp);
rel_err = norm(x(:) - x_cp(:))/norm(x(:))
> 1.4848e-06
x_trunc_cp = truncate(x_cp, opts);
rel_err = norm(x(:) - x_trunc_cp(:))/norm(x(:))
> 2.0001e-06
```

The approach above is limited to functions of a very specific structure. A more generally applicable method relies on a Newton-Schultz iteration for finding the elementwise reciprocal of a tensor in HTD, see `examples/elem_reciprocal.m` in the `htucker` toolbox. In the context of low-rank tensors, such an iteration was already proposed by Oseledets in [36].

Construction of $\mathcal{X}$ using Newton-Schultz iteration

```
xisum_cp = cell(1, d);
for ii=1:d
  xisum_cp{ii} = ones(n, d); xisum_cp{ii}(:, ii) = xi;
end

opts.elem_mult_max_rank = 50; opts.elem_mult_abs_eps = 1e-2;
opts.max_rank = 50; opts.rel_eps = 1e-5;
x0 = htenones(size(x)) / ( d*max(xi) );
x_rec = elem_reciprocal(htensor(xisum_cp), opts, x0 );
rel_err = norm(x(:) - x_rec(:))/norm(x(:))
> 1.3595e-06
```

Note that, independent of the choice of the three methods above, the obtained `htensor` objects `x_ht`, `x_trunc_cp`, `x_rec` all have hierarchical ranks 5, and very similar singular value

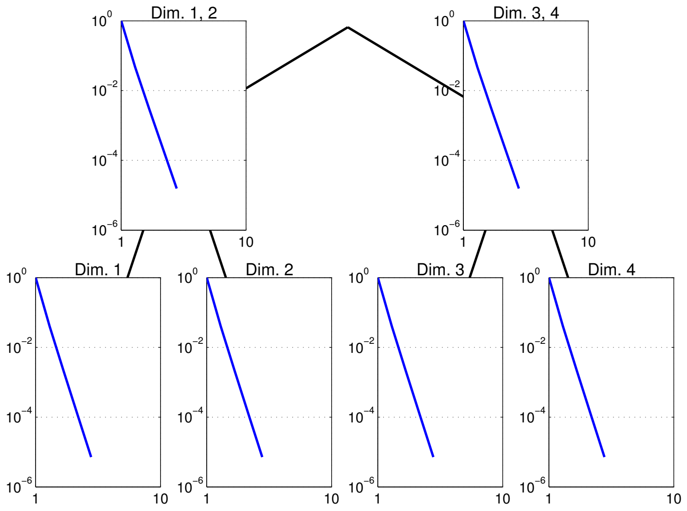


Figure 12: Singular value tree of tensors `x_ht`, `x_trunc_cp`, `x_rec` in HTD (Example 9.1), produced by the function `plot_sv`.

decay (see Figure 12). Having obtained an approximation of the sampled tensor through any of the methods described above, we will now show how to use this approximation to evaluate an integral of the form

$$\int_1^{10} \cdots \int_1^{10} \frac{1}{\xi_1 + \cdots + \xi_d} \mathrm{d}\xi_1 \cdots \mathrm{d}\xi_d.$$

We use Simpson's rule in each variable to perform numerical quadrature on the tensor grid.

Quadrature using approximation in HTD

```
% Construction of quadrature weights
h = 9/(n-1);
w = 4*ones(n, 1); w(3:2:end-2) = 2;  w(1) = 1;  w(end) = 1;
w = h/3*w;

% Inner product between weights and function values by repeated contraction
for ii=1:d, w_cell{ii} = w; end
ttv(x_ht, w_cell)
```

◇

Example 9.2. In the following, we consider an example from [34] concerning the solution of parameter-dependent linear systems. More specifically, let $x(\alpha)$ with $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ denote the solution of

$$\left(A_0 + \sum_{\mu=1}^{4} \alpha_\mu A_\mu \right) x(\alpha) = b. \qquad (24)$$

Then we take m samples $\{\alpha_1^{(\mu)}, \ldots, \alpha_m^{(\mu)}\}$ for each parameter α_μ and stack the sampled solutions into a “snapshot” tensor $\mathcal{X} \in \mathbb{R}^{n\times m\times m\times m\times m}$ as follows:

$$\mathcal{X}(:, i_1, i_2, i_3, i_4) = x\Big(\alpha^{(1)}_{i_1}, \alpha^{(2)}_{i_2}, \alpha^{(3)}_{i_3}, \alpha^{(4)}_{i_4}\Big), \qquad i_\mu = 1, \dots, m.$$

As explained in [34], this tensor can be interpreted as the solution of a (huge) symmetric positive definite linear system $\mathcal{A}(\mathcal{X}) = \mathcal{B}$. This allows us to approximate the solution in HTD by applying a low-rank variant of the preconditioned CG method to $\mathcal{A}(\mathcal{X}) = \mathcal{B}$, see `examples/cg_tensor.m`.

Our specific example from [34, Sec. 4] is the stationary heat equation on a square domain with Dirichlet boundary conditions. The heat conductivity coefficient $\sigma(\xi)$ is piecewise constant and depends on the four parameters as follows:

$$\sigma(\xi) = \begin{cases} 1 + \alpha_\mu & \text{for } \xi \in \Omega_\mu,\ \mu = 1, \dots, 4, \\ 1 & \text{for } \xi \notin \bigcup_{\mu=1}^{4} \Omega_\mu, \end{cases}$$

where $\Omega_1, \dots, \Omega_4$ are mutually disjoint discs inside the domain. A finite element discretization results in a parameter-dependent linear system (24) of size $n = 1580$. We choose the samples $\{\alpha^{(\mu)}_1, \dots, \alpha^{(\mu)}_m\} = \{0, 1, \dots, 100\}$ and hence $m = 101$. The matrices $A_0, \dots, A_4$ as well as the vector b are contained in the file `examples/cookies_matrices_2x2.mat`, and the following code can be found in `examples/example_cookies.m`.

```
load cookies_matrices_2x2
A_handle = handle_lin_mat(A, {[], 0:100, 0:100, 0:100, 0:100});
M_handle = handle_inv_mat(A{1});
e = ones(101, 1); b_cell = {b, e, e, e, e};
b_tensor = htensor(b_cell);

opts.max_rank = 30;   opts.rel_eps = 1e-10;
opts.maxit = 50;      opts.tol = 0;
[x, norm_r] = cg_tensor(A_handle, M_handle, b_tensor, opts);
```

From the resulting tensor $\mathcal{X} \in \mathbb{R}^{1580\times 101\times 101\times 101\times 101}$, we calculate the sample mean and variance of x, see also Figure 13.

```
x_mean = full(ttv(x, {e,e,e,e}, [2 3 4 5])) / 101^4;
x_diff = x - htensor({x_mean,e,e,e,e});
x_var = diag(full(ttt(x_diff, x_diff, [2 3 4 5]))) / ( 101^4 - 1 );
```

◇

10 Conclusions and ongoing work

The main goal of this work was to provide a convenient way to work with tensors in HTD. Several extensions of this work are possible and currently under consideration.

The most expensive operations of the `htucker` toolbox are typically calls to level 3 BLAS and LAPACK functions, and hence there is a simple to gain performance by, e.g., linking to multi-threaded BLAS. However, in order to address challenging applications that feature high

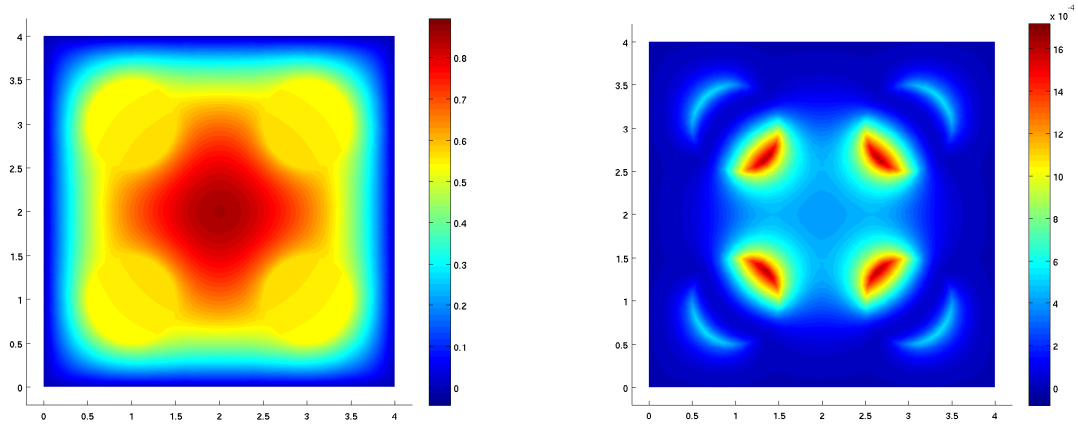


Figure 13: Sample mean and variance of the parameter-dependent stationary heat equation (Example 9.2).

ranks, there is clearly a need for a more advanced implementation fine-tuned to modern high-performance and parallel machines.

One of the most challenging aspects in using low-rank tensor decompositions, such as HTD, is the a priori choice of a suitable decomposition. This also includes the choice of the dimension tree in HTD, which at the moment is ad hoc. The development of rational criteria or effective heuristics for these choices is an important open question.

The work on low-rank tensor decompositions is quickly expanding. As new methods emerge and mature, they will be added to future versions of the `htucker` toolbox. Candidates include dynamical low-rank methods for HTD [3, 35] and low-rank tensor cross approximation for HTD [6, 8]. For both, preliminary implementations are already available see `http://anchp.epfl.ch/htucker`.

References

[1] E. Acar, D. M. Dunlavy, and T. G. Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *J. Chemometrics*, 25(2):67–86, 2011.

[2] C. A. Andersson and R. Bro. The N-way toolbox for MATLAB. *Chemometrics and Intelligent Laboratory Systems*, 52(1):1–4, 2000. Available from `http://www.models.life.ku.dk/nwaytoolbox`.

[3] A. Arnold and T. Jahnke. On the approximation of high-dimensional differential equations in the hierarchical Tucker format. Technical report, KIT, Karlsruhe, Germany, 2012.

[4] B. W. Bader and T. G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Trans. Math. Software*, 32(4):635–653, 2006. Available from `http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/`.

[5] B. W. Bader and T. G. Kolda. Efficient MATLAB computations with sparse and factored tensors. *SIAM J. Sci. Comput.*, 30(1):205–231, 2007.

[6] J. Ballani. *Fast evaluation of near-field boundary integrals using tensor approximations.* Dissertation, Universität Leipzig, 2012.

[7] J. Ballani and L. Grasedyck. A projection method to solve linear systems in tensor format. *Numer. Linear Algebra Appl.*, 20(1):27–43, 2013.

[8] J. Ballani, L. Grasedyck, and M. Kluge. Black box approximation of tensors in hierarchical Tucker format. Technical report, RWTH Aachen, Germany, 2012. To be published, available at `http://dx.doi.org/10.1016/j.laa.2011.08.010`.

[9] B. Bauer et al. The ALPS project release 2.0: open source software for strongly correlated systems. *J. Stat. Mech.*, 5, 2011.

[10] J. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35:283–319, 1970.

[11] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000.

[12] M. Espig and W. Hackbusch. A regularized Newton method for the efficient approximation of tensors represented in the canonical tensor format. *Numer. Math.*, 122(3):489–525, 2012.

[13] M. Espig, W. Hackbusch, S. Handschuh, and R. Schneider. Optimization problems in contracted tensor networks. *Computing and Visualization in Science*, 14:271–285, 2011.

[14] M. Espig, W. Hackbusch, T. Rohwedder, and R. Schneider. Variational calculus with sums of elementary tensors of fixed rank. *Numer. Math.*, 122(52):469–488, 2012.

[15] M. Espig, M. Schuster, A. Killaitis, N. Waldren, P. Wähnert, S. Handschuh, and H. Auer. TensorCalculus library, 2012. Available from `http://gitorious.org/tensorcalculus`.

[16] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore (MD), third edition, 1996.

[17] L. Grasedyck. Existence and computation of low Kronecker-rank approximations for large linear systems of tensor product structure. *Computing*, 72(3-4):247–265, 2004.

[18] L. Grasedyck. Hierarchical low rank approximation of tensors and multivariate functions, 2010. Lecture notes of Zürich summer school on "Sparse Tensor Discretizations of High-Dimensional Problems".

[19] L. Grasedyck. Hierarchical singular value decomposition of tensors. *SIAM J. Matrix Anal. Appl.*, 31(4):2029–2054, 2010.

[20] W. Hackbusch. Approximation of $1/x$ by exponential sums. Available from `http://www.mis.mpg.de/scicomp/EXP_SUM/1_x/tabelle`. Retrieved August 2008.

[21] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus.* Springer, 2012.

[22] W. Hackbusch, B. N. Khoromskij, S. A. Sauter, and E. E. Tyrtyshnikov. Use of tensor formats in elliptic eigenvalue problems. *Numer. Linear Algebra Appl.*, 19(1):133–151, 2012.

[23] W. Hackbusch and S. Kühn. A new scheme for the tensor representation. *J. Fourier Anal. Appl.*, 15(5):706–722, 2009.

[24] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

[25] S. Holtz, T. Rohwedder, and R. Schneider. The alternating linear scheme for tensor optimization in the Tensor Train format. *SIAM J. Sci. Comput.*, 34(2):A683–A713, 2012.

[26] T. Huckle, K. Waldherr, and T. Schulte-Herbrüggen. Computations in quantum tensor networks. *Linear Algebra Appl.*, 438(2):750–781, 2013.

[27] V. A. Kazeev and B. N. Khoromskij. Low-rank explicit QTT representation of Laplace operator and its inverse. *SIAM J. Matrix Anal. Appl.*, 33(3):742–758, 2012.

[28] B. N. Khoromskij and I. V. Oseledets. Quantics-TT collocation approximation of parameter-dependent and stochastic elliptic PDEs. *Comput. Methods Appl. Math.*, 10(4):376–394, 2010.

[29] B. N. Khoromskij and I. V. Oseledets. QTT approximation of elliptic solution operators in higher dimensions. *Russian J. Numer. Anal. Math. Modelling*, 26(3):303–322, 2011.

[30] B. N. Khoromskij and Ch. Schwab. Tensor-structured Galerkin approximation of parametric and stochastic elliptic PDEs. *SIAM J. Sci. Comput.*, 33(1):364–385, 2011.

[31] O. Koch and C. Lubich. Dynamical tensor approximation. *SIAM J. Matrix Anal. Appl.*, 31(5):2360–2375, 2010.

[32] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[33] D. Kressner and C. Tobler. Krylov subspace methods for linear systems with tensor product structure. *SIAM J. Matrix Anal. Appl.*, 31(4):1688–1714, 2010.

[34] D. Kressner and C. Tobler. Low-rank tensor Krylov subspace methods for parametrized linear systems. *SIAM J. Matrix Anal. Appl.*, 32(4):1288–1316, 2011.

[35] Ch. Lubich, T. Rohwedder, R. Schneider, and B. Vandereycken. Dynamical approximation of hierarchical Tucker and tensor-train tensors. Technical report, July 2012.

[36] I. V. Oseledets. MATLAB TT-Toolbox Version 1.0, May 2009. See `http://spring.inm.ras.ru/osel/?page_id=24`.

[37] I. V. Oseledets. MATLAB TT-Toolbox Version 2.1, May 2011. See `http://spring.inm.ras.ru/osel/?page_id=24`.

[38] I. V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011.

[39] I. V. Oseledets and B. N. Khoromskij. DMRG+QTT approach to high-dimensional quantum molecular dynamics. Preprint 69/2010, Max-Planck-Institut für Mathematik in den Naturwissenschaften, 2010.

[40] U. Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326, 2011.

[41] C. Tobler. *Low-rank Tensor Methods for Linear Systems and Eigenvalue Problems.* PhD thesis, ETH Zürich, Switzerland, 2012.

[42] L. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

A List of Matlab functions

Tables 1, 2, and 3 give an overview of the complete functionality of our Matlab toolbox `htucker`. More details for the use of each function can be obtained using the command `help`.

Construction of `htensor` objects.	
`htensor`	Construct a tensor in HTD and return `htensor` object.
`define_tree`	Define dimension tree.

Basic functionality	
`cat`	Concatenate two `htensor` objects.
`change_dimtree`	Change dimension tree of htensor.
`change_root`	Change root of the dimension tree.
`check_htensor`	Check internal consistency of `htensor`.
`conj`	Complex conjugate of `htensor`.
`disp`	Command window display of dimension tree of `htensor`.
`display`	Command window display of dimension tree of `htensor`.
`disp_all`	Command window display of `htensor`.
`end`	Last index in one mode of `htensor`.
`equal_dimtree`	Compare dimension trees of two `htensor` objects.
`full`	Convert `htensor` to a (full) tensor.
`full_block`	Return subblock of `htensor` as a (full) tensor.
`full_leaves`	Convert leaf matrices U_t to dense matrices.
`ipermute`	Inverse permute dimensions of `htensor`.
`isequal`	Check whether two htensors are equal.
`mrdivide (/)`	Scalar division for `htensor`.
`mtimes (*)`	Scalar multiplication for `htensor`.
`ndims`	Order (number of dimensions) of `htensor`.
`ndofs`	Number of degrees of freedom in `htensor`.
`norm`	Norm of `htensor`.
`norm_diff`	Norm of difference between `htensor` and full tensor.
`nvecs`	Dominant left singular vectors for matricization of `htensor`.
`permute`	Permute dimensions of `htensor`.
`plot_sv`	Plot singular value tree of `htensor`.
`rank`	Hierarchical ranks of `htensor`.
`singular_values`	Singular values for matricizations of `htensor`.
`size`	Size of `htensor`.
`sparse_leaves`	Convert leaf matrices U_t to sparse matrices.
`spy`	Plot sparsity pattern of the nodes of `htensor`.
`squeeze`	Remove singleton dimensions from `htensor`.
`subsasgn`	Subscripted assignment for `htensor`.
`subsref`	Subscripted reference for `htensor`.
`subtree`	Return all nodes in the subtree of a node.
`uminus`	Unary minus (-) of `htensor`.
`uplus`	Unary plus for `htensor`.

Table 1: List of functions in `htucker` toolbox (part 1).

Operations with htensor objects.	
elem_mult	Approximate element-by-element multiplication for htensor.
innerprod	Inner product for htensor.
minus (-)	Binary subtraction for htensor.
plus (+)	Binary addition for htensor.
power (.^2)	Element-by-element square for htensor.
times (.*)	Element-by-element multiplication for htensor.
ttm	N-mode multiplication of htensor with matrix.
ttt	Tensor-times-tensor for htensor.
ttv	Tensor-times-vector for htensor.

Orthogonalization and truncation.	
gramians	Reduced Gramians of htensor in orthogonalized HTD.
gramians_cp	Reduced Gramians of CP tensor.
gramians_nonorthog	Reduced Gramians of htensor.
gramians_sum	Reduced Gramians for sum of htensor objects.
left_svd_gramian	Left singular vectors and values from Gramian.
left_svd_qr	Left singular vectors and values.
orthog	Orthogonalize HTD of htensor.
trunc_rank	Return rank according to user-specified parameters.
truncate	Truncate full tensor/htensor/CP to htensor.
truncate_cp	Truncate CP tensor to lower-rank htensor.
truncate_ltr	Truncate full tensor to htensor, leaves-to-root.
truncate_nonorthog	Truncate htensor to lower-rank htensor.
truncate_rtl	Truncate full tensor to htensor, root-to-leaves.
truncate_std	Truncate htensor to lower-rank htensor.
truncate_sum	Truncate sum of htensor objects to lower-rank htensor. .

Linear Operators.	
apply_mat_to_mat	Applies an operator in HTD to another operator in HTD.
apply_mat_to_vec	Applies an operator in HTD to htensor.
full_mat	Full matrix represented by an operator in HTD.
innerprod_mat	Weighted inner product for htensor.

Interface with Tensor Toolbox.	
ktensor_approx	Approximation of htensor by ktensor.
mttkrp	Building block for approximating htensor by ktensor.
ttensor	Convert htensor to a Tensor Toolbox ttensor.

Table 2: List of functions in htucker toolbox (part 2).

Auxiliary functions for full tensors.	
`dematricize`	Determine (full) tensor from matricization.
`diag3d`	Return third-order diagonal tensor.
`isindexvector`	Check whether input is index vector.
`khatrirao_aux`	Khatri-Rao product.
`khatrirao_t`	Transposed Khatri-Rao product.
`matricize`	Matricization of (full) tensor.
`spy3`	Plot sparsity pattern of order-3 tensor.
`ttm`	N-mode multiplication of (full) tensor with matrix.
`ttt`	Tensor times tensor (full tensors).

Example tensors.	
`gen_invlaplace`	htensor for approx. inverse of Laplace-like matrix.
`gen_laplace`	`htensor` for Laplace-like matrix.
`gen_sin_cos`	Function-valued `htensor` for sine and cosine.
`htenones`	`htensor` with all elements one.
`htenrandn`	Random `htensor`.
`laplace_core`	Core tensor for Laplace operator.
`reciproc_sum`	Function-valued tensor for $1/(\xi_1 + \cdots + \xi_d)$.

Examples	
`cg_tensor`	Truncated Conjugate Gradient method for htensor.
`demo_basics`	Demonstration of basic htensor functionality.
`demo_constructor`	Demonstration of htensor constructors.
`demo_elem_reciprocal`	Demonstration of element-wise reciprocal.
`demo_function`	Demonstration of htensor function approximation.
`demo_invlaplace`	Demonstration of approximate inverse Laplace.
`demo_operator`	Demonstration of operator-HTD format.
`elem_reciprocal`	Iterative computation of elementwise reciprocal for htensor.
`example_cancellation`	Cancellation in $\tan(x) + 1/x - \tan(x)$.
`example_cancellation2`	Cancellation in $\exp(-x^2) + \sin(x)^2 + \cos(x)^2$.
`example_cookies`	Apply CG method to a parametric PDE.
`example_maxest`	Example for computing element of maximal absolute value.
`example_spins`	Demonstration of operator-HTD for 1D spin system.
`example_sum_reciproc`	Apply element-wise reciprocal method to sum of tensors.
`example_truncation`	Comparison of speed for different truncation methods.
`handle_inv_mat`	Function handle to Kronecker structured matrix multiplication.
`handle_lin_mat`	Function handle to Kronecker structured matrix multiplication.
`maxest`	Approximate element of maximal absolute value.

Table 3: List of functions in `htucker` toolbox (part 3).

B Proofs of approximation results

In this appendix, we provide and prove bounds on the approximation error of the leaves-to-root method (Section 6.1.2) and of truncation in HTD without initial orthogonalization (Section 6.2.2). The proofs will be based on the following basic result.

Lemma B.1. *Consider a dimension tree $\mathcal{T}$ and orthogonal projections $\pi_t = W_t W_t^H$ for $t \in \mathcal{T}$. If the projections are nested:*

$$\mathit{span}(W_t) \subset \mathit{span}(W_{t_r} \otimes W_{t_l}) \qquad \text{for all } t \in \mathcal{N}(\mathcal{T}),$$

then all projections π_s and π_t commute.

Proof. Any two nodes $s, t \in \mathcal{T}$ are either disjoint ($s \cap t = \emptyset$) or nested (w.l.o.g., $s \subset t$):

1. $s \cap t = \emptyset$: In this case, the statement of the lemma follows directly from

   $$W_s W_s^H \circ_s W_t W_t^H \circ_t \mathcal{X} = W_t W_t^H \circ_t W_s W_s^H \circ_s \mathcal{X},$$

   for any tensor $\mathcal{X}$.

2. $s \subset t$: Without loss of generality, we may assume that the modes contained in node s are the leading modes of t. Then $\mathrm{span}(W_t) \subset \mathrm{span}(I \otimes W_s)$ and the statement is an immediate consequence of the obvious fact that projections onto nested subspaces commute.

□

To simplify the presentation, we require some additional notation. All truncation methods involve a sequence of projections to some subspaces $\mathcal{R}(W_t)$ for nodes $t \in \mathcal{T}$. Note that there is no projection associated with the root node. Moreover, the truncations for both children t_l, t_r of the root node arise from essentially the same singular value decomposition, as $X^{(t_l)} = (X^{(t_r)})^T$. Thus, it is sufficient to consider only one of them for the error bound. We therefore define the set $\mathcal{T}'$ to contain all nodes of $\mathcal{T}$, except for the root node and one of its children.

The leaves-to-root method described in Algorithm 5 can be written recursively in terms of projections as

$$\widetilde{\mathcal{X}}_L := \mathcal{X}, \quad \widetilde{\mathcal{X}}_{\ell-1} = \prod_{t \in \mathcal{T}'_\ell} \pi_t \widetilde{\mathcal{X}}_\ell, \quad \widetilde{\mathcal{X}} := \widetilde{\mathcal{X}}_0,$$

with $\mathcal{T}'_L = \mathcal{L}(\mathcal{T}) \cap \mathcal{T}'$ and $\mathcal{T}'_\ell = (\mathcal{T}_\ell \cap \mathcal{T}') \setminus \mathcal{L}(\mathcal{T})$, $\ell = 1, \dots, L-1$. Defining the matrix W_t to contain the k_t dominant left singular vectors of $\widetilde{X}_\ell^{(t)}$ for $t \in \mathcal{T}'_\ell$ and $\ell \in \{1, \dots, L\}$, we set $\pi_t = W_t W_t^H$.

Lemma B.2. *The truncation error of the leaves-to-root method described in Algorithm 5 satisfies*

$$\|\mathcal{X} - \widetilde{\mathcal{X}}\|_2 \le \sqrt{\sum_{\ell=1}^{L} \sum_{t \in \mathcal{T}'_\ell} \delta_{k_t}(\widetilde{X}_\ell^{(t)})^2} \le \sqrt{2d-3}\, \|\mathcal{X} - \mathcal{X}_{best}\|_2,$$

where $\mathcal{X}_{best}$ is a best approximation of $\mathcal{X}$ in $\mathcal{H}$-Tucker$((k_t)_{t \in \mathcal{T}})$, and the low-rank approximation error δ_{k_t} is defined as in (19).

Proof. We expand $\mathcal{X} - \widetilde{\mathcal{X}}$ into a sum:

$$\mathcal{X} - \widetilde{\mathcal{X}} = \sum_{\ell=1}^{L} \widetilde{\mathcal{X}}_\ell - \widetilde{\mathcal{X}}_{\ell-1} = \sum_{\ell=1}^{L} (I - \pi_\ell)\pi_{\ell+1} \cdots \pi_L \mathcal{X}, \qquad \text{with } \pi_\ell := \prod_{t \in \mathcal{T}'_\ell} \pi_t.$$

As the projections π_t commute, each summand is orthogonal to all subsequent summands and therefore $\|\mathcal{X} - \widetilde{\mathcal{X}}\|_2^2 = \sum_{\ell=1}^{L} \|\widetilde{\mathcal{X}}_\ell - \pi_\ell \widetilde{\mathcal{X}}_\ell\|_2^2$. A similar reasoning for the projections on level ℓ leads to $\|\widetilde{\mathcal{X}}_\ell - \pi_\ell \widetilde{\mathcal{X}}_\ell\|_2^2 \le \sum_{t \in \mathcal{T}'_\ell} \|\widetilde{\mathcal{X}}_\ell - \pi_t \widetilde{\mathcal{X}}_\ell\|_2^2$, showing the first inequality of the lemma.

For the second inequality, we will prove that $\|\widetilde{\mathcal{X}}_\ell - \pi_t \widetilde{\mathcal{X}}_\ell\|_2 \le \|\mathcal{X} - \mathcal{X}_{\text{best}}\|_2$ for all nodes t. We start by defining $\mathcal{X}_{t,\text{best}}$ to be a best approximation of $\mathcal{X}$ under the condition that the rank of the t-matricization is not larger than k_t. Clearly, $\left\|\mathcal{X} - \mathcal{X}_{t,\text{best}}\right\|_2 \le \left\|\mathcal{X} - \mathcal{X}_{\text{best}}\right\|_2$. Furthermore, we define the set

$$\mathcal{S}_t := \left\{ \mathcal{Y} \in \mathbb{C}^{n_1 \times \cdots \times n_d} \,\middle|\, \operatorname{rank}(Y^{(t)}) \le k_t \text{ and } \operatorname{span}(Y^{(s)}) \subset \operatorname{span}(W_s) \ \forall s \in \bigcup_{j=\ell+1}^{L} \mathcal{T}'_j \right\}.$$

Note that $\pi_t \widetilde{\mathcal{X}}_\ell$ is a minimizer of $\|\widetilde{\mathcal{X}}_\ell - \mathcal{Y}\|_2$ on $\mathcal{S}_t$, as π_t is based on the SVD of $\widetilde{X}_\ell^{(t)}$. Furthermore, $\pi_{\ell+1} \cdots \pi_L \mathcal{X}_{t,\text{best}}$ is a member of $\mathcal{S}_t$ (from [19, Lemma 3.15]). In conclusion,

$$\left\|\widetilde{\mathcal{X}}_\ell - \pi_t \widetilde{\mathcal{X}}_\ell\right\|_2 \le \left\|\pi_{\ell+1} \cdots \pi_L \mathcal{X} - \pi_{\ell+1} \cdots \pi_L \mathcal{X}_{t,\text{best}}\right\|_2 \le \left\|\mathcal{X} - \mathcal{X}_{t,\text{best}}\right\|_2 \le \left\|\mathcal{X} - \mathcal{X}_{\text{best}}\right\|_2.$$

□

Truncation in HTD without initial orthogonalization is described in Algorithm 7 and can also be written recursively in terms of projections: $\widetilde{\mathcal{X}} := \prod_{t \in \mathcal{T}'} \pi_t \mathcal{X}$, where π_t is the orthogonal projection onto the subspace spanned by the k_t dominant left singular vectors of

$$\widetilde{U}_t V_t^H = \Big(\prod_{s \in \mathcal{T}', s \subsetneq t} \pi_s \Big) X^{(t)} =: \widetilde{X}_t^{(t)}. \tag{25}$$

Thus, we can equivalently define $\widetilde{\mathcal{X}} = \widetilde{\mathcal{X}}_{t_{\text{root}}}$.

Lemma B.3. *The truncation error of HTD without initial orthogonalization described in Algorithm 7 satisfies:*

$$\|\mathcal{X} - \widetilde{\mathcal{X}}\|_2 \ \le \ \sqrt{\sum_{t \in \mathcal{T}'} \delta_{k_t}(\widetilde{X}_t^{(t)})^2} \ \le \ \sqrt{2d-3}\, \|\mathcal{X} - \mathcal{X}_{best}\|_2,$$

where $\mathcal{X}_{best}$ is a best approximation of $\mathcal{X}$ in $\mathcal{H}$-Tucker$((k_t)_{t \in \mathcal{T}})$.

Proof. We define the projections $\widetilde{\pi}_t$ by the recursion

$$\widetilde{\pi}_t = \begin{cases} \pi_t & \text{if } t \text{ is a leaf node,} \\ \widetilde{\pi}_{t_l} \widetilde{\pi}_{t_r} \pi_t & \text{otherwise,} \end{cases}$$

where we have formally set $\pi_{t_{\text{root}}}$ to the identity. Note that the projections $\widetilde{\pi}_t$ commute. Let us now consider $\widetilde{\mathcal{X}}_t = \widetilde{\pi}_{t_l} \widetilde{\pi}_{t_r} \mathcal{X}$ for a non-leaf node t:

$$\begin{aligned} \|\mathcal{X} - \widetilde{\pi}_t \mathcal{X}\|_2^2 &= \|\mathcal{X} - \widetilde{\pi}_{t_l} \mathcal{X} + \widetilde{\pi}_{t_l} \mathcal{X} - \widetilde{\pi}_{t_l} \widetilde{\pi}_{t_r} \mathcal{X} + \widetilde{\pi}_{t_l} \widetilde{\pi}_{t_r} \mathcal{X} - \pi_t \widetilde{\pi}_{t_l} \widetilde{\pi}_{t_r} \mathcal{X}\|_2^2 \\ &\le \|\mathcal{X} - \widetilde{\pi}_{t_l} \mathcal{X}\|_2^2 + \|\mathcal{X} - \widetilde{\pi}_{t_r} \mathcal{X}\|_2^2 + \|\widetilde{\mathcal{X}}_t - \pi_t \widetilde{\mathcal{X}}_t\|_2^2. \end{aligned}$$

Successive application of this inequality shows the first inequality of the lemma:

$$\|\mathcal{X} - \widetilde{\mathcal{X}}\|_2^2 = \|\mathcal{X} - \widetilde{\pi}_{t_{\mathrm{root}}}\mathcal{X}\|_2^2 \leq \sum_{t \in \mathcal{T}'} \|\widetilde{\mathcal{X}}_t - \pi_t \widetilde{\mathcal{X}}_t\|_2^2.$$

For the second inequality, the proof is analogous to the one of Lemma B.2. □