# A note on approximation in Tensor Chain format

(revised version: September 2013)

by

*Mike Espig, Kishore Kumar Naraparaju, and Jan Schneider*

Preprint no.: 16 2012

# A note on Tensor Chain approximation

Mike Espig,
Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany

Kishore Kumar Naraparaju,
Indian Institute of Technology, Gandhinagar, India

Jan Schneider,
Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany

**Abstract**

This paper deals with the approximation of $d$-dimensional tensors, as discrete representations of arbitrary functions $f(x_1, \ldots, x_d)$ on $[0,1]^d$, in the so-called Tensor Chain format. The main goal of this paper is to show that the construction of a Tensor Chain approximation is possible using Skeleton/Cross Approximation type methods. The complete algorithm is described, computational issues are discussed in detail and the complexity of the algorithm is shown to be linear in $d$. Some numerical examples are given to validate the theoretical results.

**AMS Subject Classification:** 15A69, 41A63, 65F30

**Keywords:** Cross Approximation, Skeleton Decomposition, Tensor Chain format, Singular Value Decomposition

## 1 Introduction

Multiparametric arrays appear in many applications. The direct numerical treatment of these arrays leads to serious problems like memory requirements and the complexity of basic operations (they grow exponentially in $d$). In the last decade the approximation of multiparametric arrays has become a central issue in approximation theory and numerical analysis. These arrays, from now on called tensors, often arise from the discretizations of multidimensional functions that are involved in the numerical treatment of complex problems in many different areas of natural, financial or social sciences. In [1] one can find an overview on applications of several formats of such tensor decompositions, where for more recent developments one may look in [2, 8, 12, 21, 25, 34].

The main idea of the approximation of a tensor is decomposing the given tensor as sums of outer products of vectors. In the language of functions, it is an approximation of multivariable functions by sums of products of univariate functions.

In the matrix case (tensor of order 2) the low rank approximation is well studied. The original $n \times n$ matrix $A$ can be reconstructed through a product of $U$ and $V$. Namely, $A \approx U_{n\times r} V_{r\times n}$, where $r$ is the effective rank of $A$. Singular value decomposition (SVD) gives the best decomposition with given rank. However the algorithm is very expensive ($O(n^3)$), especially when the matrix dimensions are large [22].

Various inexpensive techniques of low rank approximation based on skeleton/cross approximation are available in the literature. In the skeleton decomposition [15] the matrix $A$ is approximated by $A \approx UGV^T$, where $U, V^T$ consists of selected columns and rows of $A$ and $G = M^{-1}$, a $r \times r$ submatrix on the intersection of crosses formed by selected rows and columns. The accuracy of this depends on the choice of $M$. A good choice for $M$ is the maximum volume submatrix, i.e, the one with maximum modulus of determinant among all $r \times r$ submatrices. Since the search for this submatrix is a $NP-$complex problem, it is not feasible even for moderate values of $n$ and $r$. In practise, such a submatrix can be replaced by matrices that can be computed by the techniques described in [3, 32]. In [3, 4] adaptive Cross Approximation (ACA) has been proposed. This technique gives the above decomposition adaptively and the columns and rows are iteratively added until an error criterion is reached. In pseudo-skeleton decomposition the matrix $G$ is not necessary equal to $M^{-1}$ and even not necessarily nonsingular [18, 19]. For example $G$ can be chosen as pseudo inverse of $M$. The computational complexity of these techniques scales linear in $n$.

For $d > 2$, there are many tensor formats like Canonical, Tucker (for complete list of references look at [1]), Higher order Singular Value Decomposition (HOSVD) [9, 10], Hierarchical format [23] and Tensor Train format [34, 35] etc. available in the literature. For the details of all these formats and their applications in various fields one can look at the recent book by Hackbusch [24]. Adaptive Cross Approximation is generalized to higher dimensions ($d$=3,4) in [4]. In [33] matrix Cross Approximation method is generalized to three dimensional tensors (see also [11, 30]) and in [16] the results for pseudo-skeleton methods have been generalized to $d$ dimensional tensors (compare also [7, 28, 14]).

In [34], Tensor Train (TT) approximation is proposed for multidimensional arrays using skeleton decomposition, similar to what we are going to present here. The TT approximation of a 3-dimensional tensor $A$ can be written as

$$A = (a_{ijk}) \approx \sum_{m_1=1}^{r_1} \sum_{m_2=1}^{r_2} \quad u_i^{m_1} v_j^{(m_1,m_2)} w_k^{m_2},$$

where $u_i, v_j, w_k$ depend only on one variable. The summation indices $m_1, m_2$ are referred to as auxiliary indices. TT approximation is computed by a sequence of skeleton decomposition. Further developements in TT approximation can be found in [35, 31].

The Tensor Chain (TC) approximation of a three dimensional array $A$ looks similar:

$$A = (a_{ijk}) \approx \sum_{m_1=1}^{r_1} \sum_{m_2=1}^{r_2} \sum_{m_3=1}^{r_3} u_i^{(m_1,m_2)} v_j^{(m_2,m_3)} w_k^{(m_3,m_1)}.$$

One can easily see that the Tensor Train (TT) format can be considered as a Tensor Chain with one summation rank equal to one (i.e summation index $m_3$ with $r_3 = 1$).

The natural question arised whether skeleton/Cross Approximation type methods would also be able to construct the TC format with the ranks $r_1, r_2, r_3$ all different from 1. But there was for some time the believe in the community that this is not possible. The main aim of this paper is to show that such an approximation is possible, i.e, it is possible to approximate a multidimensional array in Tensor Chain format with ranks $r_1, r_2, ..., r_d$, which are all different from 1, using skeleton/Cross Approximation type methods.

In this paper we focus on the construction of Tensor Chain approximation and the developement of the algorithm. The rigorous study of the quality of approximation is ongoing work. We emphasize that we do not claim TC to be the format of choice having better properties than other formats. We are aware of the results in [27], concerning the closedness of the TC format.

The basic idea is explained for $d = 3$ in section 2, the implementation issues are discussed

in section 3, also for higher dimensions. The computational complexity of the algorithm is shown to be linear in $d$ and $n$. Two numerical experiments are shown to support the theory in section 4.
We will use a lot of terms and notation introduced in the literature mentiond above (especially [34] and [36]). When working with arrays we often use matlab notation.

## 2 The idea

We are given a function $f(x, y, z)$ on $[0,1]^3$. After discretization of the unit cube with a uniform grid of $n$ points in each direction we represent $f$ by the tensor

$$A = (a_{ijk}), \qquad \text{where} \qquad a_{ijk} = f\left(\frac{i-1}{n-1}, \frac{j-1}{n-1}, \frac{k-1}{n-1}\right)$$

for indices $1 \leq i, j, k \leq n$. The task is now to find an approximation of $A \in \mathbb{R}^{n\times n\times n}$ in the form

$$A = (a_{ijk}) \approx \sum_{m_1=1}^{r} \sum_{m_2=1}^{r} \sum_{m_3=1}^{r} u_i^{(m_1,m_2)} v_j^{(m_2,m_3)} w_k^{(m_3,m_1)}, \tag{1}$$

where $r \in \mathbb{N}$ and for fixed numbers $1 \leq m_1, m_2, m_3 \leq r$ the vectors $u, v, w$ depend only on one variable. The right-hand side of (1) is known as the Tensor Chain format, where this name first appeared in [26]. It can be visualized by the following tensor network graph.

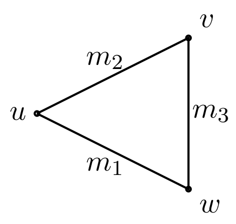

Figure 1: Tensor Chain graph for $d = 3$

Here the nodes symbolize the spatial variables and the edges show the summation indices. For a more detailed discussion of tensor network graphs see [12]. By the help of those graphs we can illustrate the meaning of each step in the approximation scheme we treat in this article.

Step 1
Consider the unfolding $A_1 = A(i; jk) \in \mathbb{R}^{n\times n^2}$ and apply Cross Approximation with $r^2$ steps to $A_1$ (if the rank of A is smaller than $r^2$, the algorithm would stop having found this rank, see [3]). The resulting approximation can be written as

$$A_1 \approx \sum_{l=1}^{r^2} u_i^{(l)} B_{jk}^{(l)}, \tag{2}$$

where the vectors $u^{(l)} \in \mathbb{R}^n$ depend only on the first variable $i$ and the long vectors $B^{(l)} \in \mathbb{R}^{n^2}$ still depend on the second and third variables $j, k$. In terms of our graphs, we established the following connection:

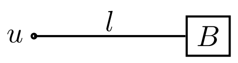

Figure 2: Graph after first step

Here the box means, that the object $B$ is still depending on more than one spatial variable. Now we come to a small trick that already gives us a Chain like look of the graph. We split the index $l = r(m_1 - 1) + m_2$, which means we just rename the chosen crosses by using the pairs $(m_1, m_2)$, and get

$$A_1 \approx \sum_{m_1=1}^{r} \sum_{m_2=1}^{r} u_i^{(m_1,m_2)} B_{jk}^{(m_1,m_2)}, \tag{3}$$

where the right-hand sides of (2) and (3) are exactly the same just carrying more complicated indices now. The graph looks like:

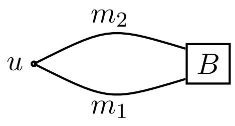

Figure 3: Graph after index splitting

Step 2

For varying $m_1, m_2$ the object $B$ can be interpreted as a 4-dimensional tensor. To achieve the final structure of (1) we have to combine the right pairs of parameters and seperate them from the others. So we reshape $B$ into the matrix $B(jm_2; km_1) \in \mathbb{R}^{rn \times rn}$ and apply Cross Approximation with $r$ steps now. That gives

$$B \approx \sum_{m_3=1}^{r} v_j^{(m_2,m_3)} w_k^{(m_3,m_1)}, \tag{4}$$

which inserted into (3) leads to the desired structure with a network graph identical to Figure 1.

It is always possible to obtain the Tensor chain representation using singular value decomposition at each step (i.e. applying SVD to $A_1$ in Step 1 and to $B$ in Step 2). Though this is very expensive, we will show some numerical tests for the approximation quality at the end of this section.

The above steps roughly describes the idea how to use Cross Approximation techniques to construct a Tensor Chain for an arbitrarily given 3d-tensor. Here for simplicity we balanced the Chain in the sense that all ranks equal $r$. When it comes to the implementation we will allow different ranks $r_1, r_2, r_3$.

The outcome of this procedure we formulate in the language of functions as the following theorem.

**Theorem 2.1** *Given an arbitrary function $f(x, y, z)$ defined on $[0, 1]^3$, the steps described above results in an approximant $F$ having the following Tensor Chain structure*

$$F(x, y, z) = \sum_{m_1=1}^{r} \sum_{m_2=1}^{r} \sum_{m_3=1}^{r} u^{(m_1,m_2)}(x) v^{(m_2,m_3)}(y) w^{(m_3,m_1)}(z), \tag{5}$$

*where*

$$u^{(m_1,m_2)}(x) = \sum_{m_1'=1}^{r} \sum_{m_2'=1}^{r} c_1 f(x, (y, z)^{(m_1',m_2')}), \tag{6}$$

$$v^{(m_2,m_3)}(y) = \sum_{m_3'=1}^{r} c_2 f(x^{(m_1^{(m_3')},m_2)}, y, z^{(m_3')}) \tag{7}$$

$$\textit{and} \quad w^{(m_3,m_1)}(z) = f(x^{(m_1,m_2^{(m_3)})}, y^{(m_3)}, z). \tag{8}$$

*Here $(y,z)^{(m_1',m_2')}$ denote the pivot coordinates of the first step in "second" direction, $y^{(m_3)}, z^{(m_3')}$ denote the pivot coordinates of the second step and $x^{(m_1^{(m_3')},m_2)}, x^{(m_1,m_2^{(m_3)})}$ denote some of the pivot coordinates taken in the first step in "first" direction selected by $m_3$ or $m_3'$ in the second step. The numbers $c_1 = c_1(f,m_1,m_2,m_1',m_2')$ and $c_2 = c_2(f,m_3,m_3')$ are real constants with respect to $(x,y,z)$.*

**Remark 2.2** *Although we just called the output of the procedure an approximant, it is not clear yet that the procedure approximates our input function $f$ at all. This will be justified by the numerical examples in section 4.*

**Proof** In the first step the algorithm treats the function $f_1(x,\eta) = f(x,y,z)$ with $\eta = (y,z)$ as a function of two variables and applies Cross Approximation with $r^2$ steps, which gives

$$f_1(x,\eta) \approx \sum_{l=1}^{r^2}\sum_{l'=1}^{r^2} c_1 f_1(x,\eta^{(l')}) f_1(x^{(l)},\eta), \tag{9}$$

with pivot points $(x^{(l)},\eta^{(l)})$ for $l = 1,\dots,r^2$ and coefficients $c_1$, defined for example by using the so-called Fredholm minors of $f_1$, i.e.

$$c_1 = c_1(l,l') = (-1)^{l+l'} \frac{f_1\binom{x^{(1)},\dots,x^{(l-1)},x^{(l+1)},\dots,x^{(r^2)}}{\eta^{(1)},\dots,\eta^{(l-1)},\eta^{(l+1)},\dots,\eta^{(r^2)}}}{f_1\binom{x^{(1)},\dots,x^{(r^2)}}{\eta^{(1)},\dots,\eta^{(r^2)}}},$$

(see [29, 36]). These coefficients are nothing but the entries of the inverse of the matrix consisting of the function values at all the intersection points of pivot rows and columns. Now we split the index $l = r(m_1-1)+m_2$ (analog also for $l'$) such that there is a one-to-one correspondence between $\{1,\dots,r^2\}$ and $\{1,\dots,r\}\times\{1,\dots,r\}$. Then formula (9) becomes

$$f_1(x,\eta) \approx \sum_{m_1=1}^{r}\sum_{m_2=1}^{r} \underbrace{\sum_{m_1'=1}^{r}\sum_{m_2'=1}^{r} c_1 f_1(x,\eta^{(m_1',m_2')})}_{=:u^{(m_1,m_2)}(x)} f_1(x^{(m_1,m_2)},\eta), \tag{10}$$

which gives (6).
Now for each of the $r^2$ pairs $(m_1,m_2)$ we can interpret $f_1(x^{(m_1,m_2)},\eta) = f(x^{(m_1,m_2)},y,z)$ as a distinct function. For the second step the algorithm considers all those functions at the same time by the definition

$$f_2(\mu,\nu) := f(x^{(m_1,m_2)},y,z), \text{ with } \mu = (m_2,y) \text{ and } \nu = (m_1,z).$$

This is formally again a two-dimensional function, so we apply Cross Approximation with $r$ steps and get

$$f_2(\mu,\nu) \approx \sum_{m_3=1}^{r} \underbrace{\sum_{m_3'=1}^{r} c_2 f_2(\mu,\nu^{(m_3')})}_{=:v^{(m_2,m_3)}(y)} \underbrace{f_2(\mu^{(m_3)},\nu)}_{w^{(m_3,m_1)}(z)}, \tag{11}$$

with pivot points $(\mu^{(m_3)},\nu^{(m_3)})$ for $m_3 = 1,\dots,r$ and coefficients $c_2$, defined analogously to $c_1$ now for the function $f_2$. If we insert formula (11) for the last term in (10) we get

(5). After re-substitution we find

$$\begin{aligned} v^{(m_2,m_3)}(y) &= \sum_{m_3'=1}^{r} c_2 f_2(\mu, \nu^{(m_3')}) \\ &= \sum_{m_3'=1}^{r} c_2 f_2((m_2,y),(m_1,z)^{(m_3')}) \\ &= \sum_{m_3'=1}^{r} c_2 f(x^{(m_1^{(m_3')},m_2)}, y, z^{(m_3')}), \end{aligned}$$

which is (7), and

$$w^{(m_3,m_1)}(z) = f_2(\mu^{(m_3)},\nu) = f_2((m_2,y)^{(m_3)},(m_1,z)) = f(x^{(m_1,m_2^{(m_3)})}, y^{(m_3)}, z),$$

which proves (8) and finishes the proof.

□

One of the best properties of skeleton/Cross Approximation techniques is that they interpolate the original object on prescribed positions (see [3, 18, 32]). In this spirit we can formulate the following result.

**Proposition 2.3** *(Interpolation)*
*We have $f(x,y,z) = F(x,y,z)$ whenever there exist $m_1, m_2, m_3$ satisfying*

$$\begin{aligned} & x = x^{(m_1,m_2)} \textit{ and } y \textit{ is such that } (m_2,y) = (m_2,y)^{(m_3)} && (12) \\ \textit{or} \quad & x = x^{(m_1,m_2)} \textit{ and } z \textit{ is such that } (m_1,z) = (m_1,z)^{(m_3)} && (13) \\ \textit{or} \quad & (y,z) = (y,z)^{(m_1,m_2)} \textit{ with } (m_2,y) = (m_2,y)^{(m_3)} && (14) \\ \textit{or} \quad & (y,z) = (y,z)^{(m_1,m_2)} \textit{ with } (m_1,z) = (m_1,z)^{(m_3)}. && (15) \end{aligned}$$

**Proof** We look at the proof of the above Theorem and denote by $F_1(x,\eta)$ the right-hand side of (10) and by $F_2(\mu,\nu)$ the right-hand side of (11). Because of the interpolation property (see [3] again) we have $f_1(x,\eta) = F_1(x,\eta)$ if we choose a pivot row $x = x^{(m_1,m_2)}$ for a pair $(m_1,m_2) \in \{1,\ldots,r\}^2$, or a pivot column $\eta = \eta^{(m_1',m_2')}$ for $(m_1',m_2') \in \{1,\ldots,r\}^2$. Our procedure approximates parts of $F_1$ further on, so the interpolation property can only hold where this second approximation is exact. That is, $f_2(\mu,\nu) = F_2(\mu,\nu)$ if $\mu = \mu^{(m_3)}$ or $\nu = \nu^{(m_3')}$ for $m_3, m_3' \in \{1,\ldots,r\}$. Now we put those conditions together.
First suppose we choose $x = x^{(m_1,m_2)}$ for some fixed $(m_1,m_2)$. Then the interpolation property holds either if $y$ is such that the coordinate $(m_2,y)$ was chosen as a pivot coordinate in the second approximation, i.e. $\mu = (m_2,y) = (m_2,y)^{(m_3)} = \mu^{(m_3)}$ for some $m_3$ (condition (12), interpolation on a $z$-fiber), or if $z$ is such that the coordinate $(m_1,z)$ was chosen as a pivot coordinate in the second approximation, i.e. $\nu = (m_1,z) = (m_1,z)^{(m_3)} = \nu^{(m_3)}$ for some $m_3$ (condition (13), interpolation on a $y$-fiber).
In the other case, suppose we choose $\eta = (y,z) = (y,z)^{(m_1,m_2)}$ for fixed $(m_1,m_2)$. Then the procedure interpolates on a $x$-fiber if for some $m_3$ either $(m_2,y) = (m_2,y)^{(m_3)}$ (condition (14)) or $(m_1,z) = (m_1,z)^{(m_3)}$ (condition (15)) was chosen as a pivot coordinate in the second step.

□

**Remark 2.4** *Conditions* (12)-(15) *look rather technical, but the philosophy behind is very simple: The procedure reproduces the function exact if the interpolation conditions of both Cross Approximation procedures together are fulfilled.*

The question is how to implement this scheme with linear complexity in $n$. That means we have to avoid to store the long vectors $B^{(l)} \in \mathbb{R}^{n^2}$ from Step 1 and the large matrix $B(jm_2; km_1) \in \mathbb{R}^{rn\times rn}$ from Step 2. This is possible and we give the detailed description in the next section. There we will modify the basic idea described above at many places according to numerical stability, efficiency and low rank accuracy. The structure of the factors $u, v, w$ in (5) will also change a lot as well as the exact form of the involved constants so that the Theorem and the Proposition above would have to be adapted to the implemented procedure. But the main property will stay untouched, namely that each factor in the decomposition is depending on one spatial variable only. We come back to this again in the summary at the end of section 3.2.
To provide numerical evidence that the idea works at all (also for $d > 3$), we present some small experiments using the expensive but optimal (and much easier to implement) SVD instead of Cross Approximation in the procedure described above, as mentioned after Step 2. Here our underlying function is

$$f(x_1, \ldots, x_d) = \frac{1}{\sqrt{1 + x_1^2 + \ldots + x_d^2}}$$

defined on $[0, 1]^d$. We discretize the unit cube with $n$ points in each direction by an uniform grid. In Step 1 by SVD we get $A_1 = U\Sigma V^T$ and truncate the factors $A_1 \approx \tilde{U}\tilde{\Sigma}\tilde{V}^T$ such that only singular values $|\sigma_j| > 1e-09$ (determining the rank $R$) are taken into account for the sequel. Then we take the rounded up square root of $R$ as splitted rank $r$, which is similarly described in Step 1. From Step 2 on, the truncation after each SVD (also for $d > 3$) determines the rank $r$ as smallest number such that the sum of all $\sigma_j^2$ with $j > r$ is smaller than $\varepsilon^2$. This tolerance $\varepsilon$, the outcoming ranks and the relative approximation error measured in the Frobenius norm are shown in Table 2.1[1].

| $d$ | $n$ | $\varepsilon$ | ranks | error |
|---|---|---|---|---|
| 3 | 129 | 0.1 | 3,3,3 | 1.4e-05 |
| | | 1e-05 | 3,3,15 | 6.8e-09 |
| 4 | 33 | 0.1 | 3,3,3,3 | 4.0e-05 |
| | | 1e-05 | 3,3,15,19 | 6.6e-09 |
| 5 | 30 | 0.1 | 3,3,5,6,7 | 4.3e-05 |
| | | 1e-05 | 3,3,17,21,24 | 6.0e-09 |
| | 20 | 0.1 | 3,3,3,3,3 | 8.3e-05 |
| | | 1e-05 | 3,3,15,21,23 | 8.7e-09 |
| 6 | 20 | 0.1 | 3,3,5,6,6,6 | 1.8e-03 |
| | | 1e-05 | 3,3,17,22,26,44 | 3.3e-09 |
| | 10 | 0.1 | 3,3,3,3,3,3, | 2.3e-04 |
| | | 1e-05 | 3,3,15,20,25,29 | 8.0e-09 |
| 7 | 10 | 0.1 | 3,3,4,5,6,6,6 | 4.8e-03 |
| | | 1e-05 | 3,3,15,22,29,87,30 | 1.2e-08 |
| 8 | 10 | 0.1 | 3,3,5,6,6,6,6,6 | 5.8e-03 |

Table 2.1: Outcome for the SVD-based version

[1]These results and the implementation in C++ of the SVD-based procedure were produced by our colleague Stefan Handschuh, MPI MIS Leipzig.

## 3 Tensor Chain approximation using Cross Approximation

As we have seen in the section 2, the size of the unfolding matrices are like $n \times m$, where $m \sim n^q$ for some natural $q$. The adaptive Cross Approximation method (ACA) [3] is expensive in these cases. So we use pseudo-skeleton approximation to decompose the unfolding matrices. Here we describe it briefly.

### 3.1 Pseudo-skeleton approximation revisited

Before going to the details of pseudo-skeleton approximation, we first review the skeleton approximation method. Consider a matrix $A \in \mathbb{R}^{n \times m}$, where $m \sim n^q$ for some natural $q$. As explained in the introduction the matrix $A$ is approximated by $A \approx CGR$, where $C$ and $R$ contains the selected columns and rows respectively and $G = M^{-1}$, a submatrix on the intersections of the selected columns and rows. To construct a skeleton approximation, say of rank $r$, first one needs to compute $r$ columns of $A$ given by the indices $J = (j^{(1)}, \dots, j^{(r)})$ and store them as $C = A(:, J) \in \mathbb{R}^{n \times r}$. To find a good row matrix one can use the maxvol procedure (see [17, 20]) that gives $r$ row indices $I = (i^{(1)}, \dots, i^{(r)})$ such that the corresponding intersection matrix $M = A(I, J) \in \mathbb{R}^{r \times r}$ has almost maximal volume.
In the matrix case ($q = 1$) one could now also store the row matrix $R = A(I, :) \in \mathbb{R}^{r \times n}$ and achieved already a skeleton approximation $A \approx CM^{-1}R$. If the column indices $J$ at the beginning were badly chosen (maybe because of some random strategy), this approximation might not be very good and the inversion of $M$ might even be unstable (due to over estimation of ranks and random choice of columns). To overcome this one can now use the maxvol procedure for the rows $R$ to optimize the choice of the columns and even alternate further the optimization between rows and columns until the determinant of $M$ stays almost constant and the approximation is fine (row-column alternating algorithm [32, 34]).
Since we are not in the matrix case ($q > 1$) we can not follow this alternating strategy because the object $R$ is still too large to handle. Therefore we use pseudo-skeleton approximation. In the pseudo-skeleton approximation the inverse of the degenerate or badly conditioned matrix $M$ is replaced by the pseudo inverse of $M$ [19, 37].
We choose our starting number $k$ (instead of $r$) of randomly chosen columns to be rather large (so that $r$ important rows and columns are embedded in it) and after finding good rows and the intersection matrix $M$ by the maxvol procedure, we reduce the rank to $r \le k$ in the following way:
We perform a SVD of $M = USV'$ (from now on the transposed of a matrix $A$ will be denoted by $A'$) and truncate $S$ such that all its remaining singular values $\sigma_l$ satisfy a certain condition, for example, $|\sigma_l| > eps_{CPU}$, $l = 1, \dots, r$ (other conditions are possible). Correspondingly we also truncate $U \in \mathbb{R}^{k \times r}, V' \in \mathbb{R}^{r \times k}$ and our pseudo-skeleton approximation reads as $A \approx CVS^{-1} \cdot U'R$, where the inversion of the small diagonal matrix $S$ is now very easy. The factor $CVS^{-1}$ can be stored for the final approximation, where the still large object $U'R$ can be treated in further steps as a $d-1$-dimensional tensor. This describes the basic idea for each step in our algorithm that we explain now in detail.

## 3.2 The practical algorithm in $d = 3$

For an efficient implementation of the idea described at the beginning of section 2 we use in each step the pseudo-skeleton procedure from above. Before we discuss in detail how to incorporate it for our needs, we formulate the complete algorithm, which takes the initial rank $k$ and an error bound $\varepsilon$ as input to produce a Tensor Chain approximation for a tensor $A \in \mathbb{R}^{n\times n\times n}$ by using Cross Approximation techniques. The output of the algorithm will be the vectors $u, v, w$ from (1).

**Algorithm 1** (Tensor Chain approximation, $d = 3$)

```
Step 1
1: Choose $k$ random pivot indices $\bar{\eta} = (\eta^{(1)}, \dots, \eta^{(k)})$ between 1 and $n^2$.
2: Compute the $k$ corresponding $x$-fibers of $A$ and store them as $U \in \mathbb{R}^{n\times k}$.
3: Use maxvol procedure to find $k$ row indices $\bar{x} = (x^{(1)}, \dots, x^{(k)})$ in $U$ and store the
   submatrix $M \in \mathbb{R}^{k\times k}$.
4: Decompose $M = U^M S^M (V^M)'$ by SVD. Set $r$ as number of singular values larger
   than $eps_{CPU}$. Truncate $U^M, V^M \in \mathbb{R}^{k\times r}$ and $S^M \in \mathbb{R}^{r\times r}$.
5: Manipulate $U = UV^M (S^M)^{-1} \in \mathbb{R}^{n\times r}$.
6: Split $r = r_1 r_2$ with $r_1, r_2 \in \mathbb{N}$.
Step 2
7: Choose $k$ random pivot indices $\bar{y} = (y^{(1)}, \dots, y^{(k)})$ between 1 and $r_1 n$.
8: Compute the $k$ corresponding vectors of length $r_2 n$ with values in $(U^M)'(A_1|_{\bar{x}})$ and
   store them as $V \in \mathbb{R}^{r_2 n\times k}$.
9: Use maxvol procedure to find $k$ row indices $\bar{z} = (z^{(1)}, \dots, z^{(k)})$ in $V$ and store the
   submatrix $\tilde{M} \in \mathbb{R}^{k\times k}$.
10: Compute the $k$ corresponding vectors of length $r_1 n$ with values in $(U^M)'(A_1|_{\bar{x}})$ and
   store them as $W \in \mathbb{R}^{k\times r_1 n}$.
11: Decompose $\tilde{M} = U^{\tilde{M}} S^{\tilde{M}} (V^{\tilde{M}})'$ by SVD. Set $\tilde{r_3}$ as number of singular values larger
   than $eps_{CPU}$. Truncate $U^{\tilde{M}}, V^{\tilde{M}} \in \mathbb{R}^{k\times \tilde{r_3}}$ and $S^{\tilde{M}} \in \mathbb{R}^{\tilde{r_3}\times \tilde{r_3}}$.
12: Manipulate $V = VV^{\tilde{M}} \in \mathbb{R}^{r_2 n\times \tilde{r_3}}$ and $W = (U^{\tilde{M}})' W \in \mathbb{R}^{\tilde{r_3}\times r_1 n}$.
Step 3 (rank reduction)
13: Take $QR$-decompositions of $V = Q^V R^V$ and $W' = Q^W R^W$.
   Set $\tilde{S} = R^V (S^{\tilde{M}})^{-1} (R^W)'$.
14: Decompose $\tilde{S} = U^{\tilde{S}} S^{\tilde{S}} (V^{\tilde{S}})'$ by SVD with singular values $\sigma_j$. Set $r_3$ minimal with
   $\sum_{j=r_3+1}^{\tilde{r_3}} (\sigma_j)^2 < \varepsilon^2$. Truncate $U^{\tilde{S}}, V^{\tilde{S}} \in \mathbb{R}^{\tilde{r_3}\times r_3}$ and $S^{\tilde{S}} \in \mathbb{R}^{r_3\times r_3}$.
15: Manipulate $V = Q^V U^{\tilde{S}} S^{\tilde{S}} \in \mathbb{R}^{r_2 n\times r_3}$ and $W = (V^{\tilde{S}})'(Q^W)' \in \mathbb{R}^{r_3\times r_1 n}$.
```

**Detailed description**

We should always keep in mind that before we run our procedure we have no information about $f$ (or the corresponding $A$), just the possibility to compute function values (or entries in $A$), which might be expensive (Black Box). In addition to the two steps of the algorithm roughly described in the previous section, here we have a third step to reduce the rank of the last edge corresponding to the summation over $m_3$.

Step 1

1-2: We start by choosing $k$ $x$-dependent fibers of $A$ randomly out of all $n^2$ possibilities and store these vectors of length $n$ in the matrix $U \in \mathbb{R}^{n\times k}$.

3: Now we use the maxvol procedure ([17]) to find a "good" submatrix with almost maximal determinant and store it as $M \in \mathbb{R}^{k\times k}$, the corresponding $k$ row indices we store in the vector $\bar{x}$.

4-5: Actually we would have to invert $M$ for the approximation of the unfolding $A_1 \in \mathbb{R}^{n\times n^2}$ by the skeleton method, but because of the random initial choice of $k$ columns this might be an unstable operation. Therefore, we perform a SVD of $M = U^M S^M (V^M)'$ and take only the "good" part of it, i.e., we truncate its singular values to machine epsilon, call the new rank $r \leq k$, update our final $U = UV^M(S^M)^{-1} \in \mathbb{R}^{n\times r}$ and store $U^M \in \mathbb{R}^{k\times r}$ for later use instead of inverting the full $M$.
6: Finally we split the rank $r$ to establish the two edges according to the summations over $m_1$ and $m_2$. In the previous section the initial rank was $r^2$, so this splitting was just $r^2 = rr$ to balance the chain. This might not be optimal, but since we are in a Black Box situation this seemed natural. Nevertheless, now our $r$ is unlikely to be a square, so we split $r = r_1r_2$ such that $|r_1 - r_2|$ is minimal. Again, this is very artificial, just think of $r$ being a prime, then the splitting is $r = r \cdot 1$. One can certainly try different splitting strategies, see the discussion at the end of section 4.
Step 2
In step 2 we are working only with entries of the unfolded restriction of $A$ to the $(y, z)$-slices chosen in Step 1, denoted by $A_1|_{\bar{x}} \in \mathbb{R}^{k\times n^2}$. We virtually reshape the entries of $(U^M)'(A_1|_{\bar{x}})$ to a matrix, say $B \in \mathbb{R}^{r_2n\times r_1n}$ and basically follow the same strategy as in step 1.
7-8: That means we randomly choose $k$ pivot columns of $B$ and store them in $V \in \mathbb{R}^{r_2n\times k}$.
9-10: We use maxvol again to find a submatrix $\tilde{M} \in \mathbb{R}^{k\times k}$ and store the corresponding $k$ rows of $B$ in $W \in \mathbb{R}^{k\times r_1n}$.
11: For this second approximation we use the same stabilization trick as before avoiding the inversion of $\tilde{M}$. We decompose $\tilde{M} = U^{\tilde{M}} S^{\tilde{M}} (V^{\tilde{M}})'$ by SVD and truncate the singular values to machine epsilon with rank $\tilde{r_3} \leq k$.
12: We overwrite $V = VV^{\tilde{M}} \in \mathbb{R}^{r_2n\times \tilde{r_3}}$ and $W = (U^{\tilde{M}})'W \in \mathbb{R}^{\tilde{r_3}\times r_1n}$ and could actually stop working.
Step 3
But since $V$ and $W$ are quite small objects now, we can afford a further reduction of $\tilde{r_3}$.
13: In this last step we take $QR$-decompositions of $V = Q^V R^V$ and $W' = Q^W R^W$ and set $\tilde{S} = R^V (S^{\tilde{M}})^{-1} (R^W)'$ (see [5]).
14: Now we perform a SVD for $\tilde{S} = U^{\tilde{S}} S^{\tilde{S}} (V^{\tilde{S}})'$ and set the final rank $r_3 \leq \tilde{r_3}$ as the minimal number such that $\sum_{j=r_3+1}^{\tilde{r_3}} (\sigma_j)^2 < \varepsilon^2$, where $\sigma_j$ are the singular values of $\tilde{S}$ and $\varepsilon$ is the input error bound. Then we truncate $U^{\tilde{S}}$, $S^{\tilde{S}}$ and $V^{\tilde{S}}$ according to the new rank $r_3$. 15: We can now update our final $V = Q^V U^{\tilde{S}} S^{\tilde{S}} \in \mathbb{R}^{r_2n\times r_3}$ and $W = (V^{\tilde{S}})'(Q^W)' \in \mathbb{R}^{r_3\times r_1n}$ and finish.

**Comments**
Algorithm 1 solves the task from the following point of view: Given an arbitrary function $f(x, y, z)|[0, 1]^3$, we want to find the building blocks $u, v, w$ for the Tensor Chain approximation (1) of its discrete representation with a desired Frobenius norm accuracy $\varepsilon$ using at most rank $k$ for each summation.
We also implemented a slightly different algorithm in terms of the rank reduction, let's call it Algorithm 2. The modification takes place in step 11 of Algorithm 1. There the truncation criterion is such that all the singular values being smaller in modulus than

machine epsilon are omitted. In Algorithm 2 we now omit already all singular values such that the sum of their squares is still smaller than $\varepsilon^2$ (similar to step 14 of Algorithm 1). As an effect, the additional $QR$-decompositions in step 3 to reduce the rank have no further effect, therefore the whole step 3 is omitted in Algorithm 2. We will compare these two variants in our numerical results.

**Computational Cost**
The overall computational cost of Algorithm 1 is

$$\mathcal{O}(k^3 + \tilde{r}_3^3 + n(k(r + (r_1 + r_2)\tilde{r}_3^2)),$$

where we need $\mathcal{O}(nk(r_1 + r_2) + k(k + r + \tilde{r}_3))$ units of storage and $\mathcal{O}(nk(r_1 + r_2))$ tensor entry evaluations. The costs of all relevant steps are listed in Table 3.2.

| step | numerical cost | storage | tensor value calls |
|---|---|---|---|
| 2: | - | $nk$ | $nk$ |
| 3: | - | $k^2$ | - |
| 4: | $\mathcal{O}(k^3)$ | $\mathcal{O}(k(k+r))$ | - |
| 5: | $\mathcal{O}(nkr)$ | - | - |
| 8: | - | $r_2nk$ | $r_2nk$ |
| 9: | - | $k^2$ | - |
| 10: | - | $r_1nk$ | $r_1nk$ |
| 11: | $\mathcal{O}(k^3)$ | $\mathcal{O}(k(k+\tilde{r}_3))$ | - |
| 12: | $\mathcal{O}((r_1+r_2)nk\tilde{r}_3)$ | - | - |
| 13: | $\mathcal{O}((r_1+r_2)n\tilde{r}_3^2+\tilde{r}_3^3)$ | - | - |
| 14: | $\mathcal{O}(\tilde{r}_3^3)$ | $\tilde{r}_3^2$ | - |
| 15: | $\mathcal{O}((r_1+r_3)nr_3\tilde{r}_3)$ | - | - |

Table 3.2: Computational cost of Algorithm 1

**Summary**
As mentioned at the end of section 2 the practical algorithm described above draws many modifications compared with Theorem 2.1 or the subsequent Proposition. Our interest now is to give the true structure of the approximant which our algorithm produces using the discrete setting.
If we reshape the output matrices $V, W$ in line 15 of Algorithm 1 such that $V \in \mathbb{R}^{n\times r_2 r_3}$, $W \in \mathbb{R}^{n\times r_3 r_1}$ and index the columns of $U, V, W$ by the pairs $(m_1, m_2), (m_2, m_3), (m_3, m_1)$ respectively, we can state the following: Given an arbitrary tensor $A \in \mathbb{R}^{n\times n\times n}$, Algorithm 1 generates an approximant of the form

$$A \sim \sum_{m_1=1}^{r_1}\sum_{m_2=1}^{r_2}\sum_{m_3=1}^{r_3} U^{(m_1,m_2)} \otimes V^{(m_2,m_3)} \otimes W^{(m_3,m_1)}, \tag{16}$$

where

$$U = A_1|_{\bar{\eta}} V^M (S^M)^{-1}, \quad V = Q\left[B|_{\bar{y}} V^{\tilde{M}}\right] U^{\tilde{S}} S^{\tilde{S}}, \quad W = (V^{\tilde{S}})' \left(Q\left[\left((U^{\tilde{M}})' B|_{\bar{z}}\right)'\right]\right)', \tag{17}$$

where $Q[\cdot]$ denotes the corresponding factor of a $QR$-decomposition and all the other notation was introduced before. Although the explicit formulas in (17) are rather terrifying, we can see that the columns of $U, V, W$ are linear combinations of original fibers of $A$ in the corresponding directions with coefficients depending on the pivots.

### 3.3 The case $(d > 3)$

It is possible to generalize the above procedure to arbitrary dimensions $d \in \mathbb{N}$. We will describe now how such a generalization should look like even though we implemented it for $d = 3, 4$ only. The basic idea is the same as for the three-dimensional case and it is again useful to visualize the meaning of each step by the corresponding graph.

Given a function $f(x_1, \dots, x_d)$ defined on $[0,1]^d$ we wish to approximate the corresponding tensor $A \in \mathbb{R}^{n_1 \times \dots \times n_d}$ (arising after discretization) by using Cross Approximation techniques applied to successive unfoldings of $A$. The desired form is the following

$$A \sim \sum_{m_1=1}^{r_1} \cdots \sum_{m_d=1}^{r_d} U_1^{(m_1,m_2)} \otimes U_2^{(m_2,m_3)} \otimes \ldots \otimes U_{d-1}^{(m_{d-1},m_d)} \otimes U_d^{(m_d,m_1)}, \tag{18}$$

where the $U_t \in \mathbb{R}^{r_t \times n_t \times r_{t+1}}$ are 3-tensors for all $t = 1, \dots, d$. The corresponding graph is shown in figure 4.

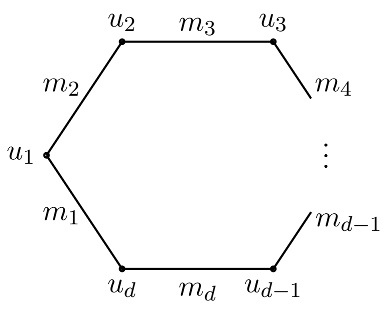

Figure 4: Tensor Chain graph for $d \in \mathbb{N}$

For simplicity of the description we assume from now on that $n_1 = \ldots = n_d = n$, so first we have to consider the unfolding $A_1 \in \mathbb{R}^{n \times n^{d-1}}$.

**Opening step**

1: Choose $k$ random pivot indices $J_1 = (j_1^{(1)}, \dots, j_1^{(k)})$ between 1 and $n^{d-1}$.
2: Compute the $k$ corresponding $x_1$-fibers of $A$ and store them as $U_1 \in \mathbb{R}^{n \times k}$.
3: Use maxvol procedure to find $k$ row indices $I_1 = (i_1^{(1)}, \dots, i_1^{(k)})$ in $U_1$ and store the submatrix $M_1 \in \mathbb{R}^{k \times k}$.
4: Decompose $M_1 = U^{M_1} S^{M_1} (V^{M_1})'$ by SVD. Set $r$ as number of singular values larger than $eps_{CPU}$. Truncate $U^{M_1}, V^{M_1} \in \mathbb{R}^{k \times r}$ and $S^{M_1} \in \mathbb{R}^{r \times r}$.
5: Manipulate $U_1 = U_1 V^{M_1} (S^{M_1})^{-1} \in \mathbb{R}^{n \times r}$.
6: Split $r = r_1 r_2$ with $r_1, r_2 \in \mathbb{N}$.

In terms of our network graph we established the following connections with $l = 1, \dots, k$, $m_1 = 1, \dots, r_1$ and $m_2 = 1, \dots, r_2$:

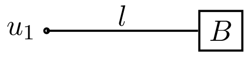

Figure 5: Graph before index splitting

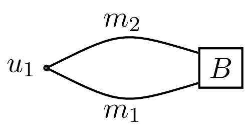

Figure 6: Graph after index splitting

Here $B$ just denotes the remaining object to approximate still depending on all other spatial variables not separated yet. Since we already produced a chain like graph we can proceed very similar to the tensor train method proposed in [34]. Therefore, in each step $t$ we consider the unfolded remainder $A_t \in \mathbb{R}^{r_t n \times r_1 n^{d-t}}$.

**Intermediate steps** ($d-3$ many)
0: Set counter $t = 2, \dots, d-2$.
1: Choose $k$ random pivot indices $J_t = (j_t^{(1)}, \dots, j_t^{(k)})$ between 1 and $r_1 n^{d-t}$.
2: Compute the $k$ corresponding $x_t$-fibers of $A$ (after manipulation with the corresponding $U^{M_1}, \dots, U^{M_{t-1}}$ from the previous steps) and store them as $U_t \in \mathbb{R}^{r_t n \times k}$.
3: Use maxvol procedure to find $k$ row indices $I_t = (i_t^{(1)}, \dots, i_t^{(k)})$ in $U_t$ and store the submatrix $M_t \in \mathbb{R}^{k \times k}$.
4: Decompose $M_t = U^{M_t} S^{M_t} (V^{M_t})'$ by SVD. Set $r_{t+1}$ as number of singular values larger than $eps_{CPU}$. Truncate $U^{M_t}, V^{M_t} \in \mathbb{R}^{k \times r_{t+1}}$ and $S^{M_t} \in \mathbb{R}^{r_{t+1} \times r_{t+1}}$.
5: Manipulate $U_t = U_t V^{M_t} (S^{M_t})^{-1} \in \mathbb{R}^{r_t n \times r_{t+1}}$.

After $d-3$ of those steps the chain graph is almost complete, it looks as in figure 7. Again, the box means that the remaining object $Z$ still depends on more than one spatial variables.

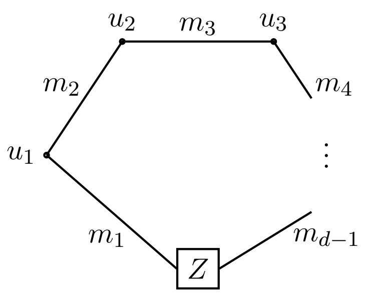

Figure 7: Tensor Chain graph for $d \in \mathbb{N}$

As we described in the Comments after Algorithm 1, here we also considered two different truncation strategies.
The point of view for the first one is to gather as much information about the tensor as possible in each step in order to assure good approximation quality. That means we omit the singular values with modulus smaller than machine epsilon (see point 4 in the intermediate step). Only in the very last step (closing step), which will be described below, we take the desired accuracy $\varepsilon$ into account and truncate with respect to the Frobenius norm to reduce the rank of the last edge.
Another point of view could be that one is only interested in a final approximation accuracy of order $\varepsilon$ anyway. So, why not truncating with respect to the Frobenius norm (as in point 8 below) in every step already in order to keep the ranks smaller ? That is exactly our second truncation strategy, previously called Algorithm 2 in case $d = 3$ (also here it results in the redundancy of steps 7,8,9 below). The opening step is untouched by this,

there we always cut down to machine epsilon because we need to split the outcoming rank making it small enough. We will discuss this issue again, when it comes to the experiments.

For the last step we are left to approximate the unfolded remainder $A_{d-1} \in \mathbb{R}^{r_{d-1}n \times r_1 n}$. This is a fairly small object and so we can store columns and rows needed for Cross Approximation to produce the last two factors in (18).

**Closing step**

1: Choose $k$ random pivot indices $J_{d-1} = (j_{d-1}^{(1)}, \dots, j_{d-1}^{(k)})$ between 1 and $r_1 n$.
2: Compute the $k$ corresponding $x_{d-1}$-fibers of $A$ (after manipulation with the corresponding $U^{M_1}, \dots, U^{M_{d-2}}$ from the previous steps) and store them as $U_{d-1} \in \mathbb{R}^{r_{d-1}n \times k}$.
3: Use maxvol procedure to find $k$ row indices $I_{d-1} = (i_{d-1}^{(1)}, \dots, i_{d-1}^{(k)})$ in $U_{d-1}$ and store the submatrix $M_{d-1} \in \mathbb{R}^{k \times k}$.
4: Compute the $k$ corresponding vectors of length $r_1 n$ ($x_d$-fibers of $A$ after manipulation) and store them as $U_d \in \mathbb{R}^{k \times r_1 n}$
5: Decompose $M_{d-1} = U^{M_{d-1}} S^{M_{d-1}} (V^{M_{d-1}})'$ by SVD. Set $\tilde{r_d}$ as number of singular values larger than $eps_{CPU}$. Truncate $U^{M_{d-1}}, V^{M_{d-1}} \in \mathbb{R}^{k \times \tilde{r_d}}$ and $S^{M_{d-1}} \in \mathbb{R}^{\tilde{r_d} \times \tilde{r_d}}$.
6: Manipulate $U_{d-1} = U_{d-1} V^{M_{d-1}} \in \mathbb{R}^{r_{d-1}n \times \tilde{r_d}}$ and $U_d = (U^{M_{d-1}})' U_d \in \mathbb{R}^{\tilde{r_d} \times r_1 n}$.
(final rank reduction :)
7: Take $QR$-decompositions of $U_{d-1} = Q_{d-1} R_{d-1}$ and $U_d' = Q_d R_d$. Set $\tilde{S} = R_{d-1} (S^{M_{d-1}})^{-1} R_d'$.
8: Decompose $\tilde{S} = U^{\tilde{S}} S^{\tilde{S}} (V^{\tilde{S}})'$ by SVD with singular values $\sigma_j$. Set $r_d$ minimal with $\sum_{j=r_d+1}^{\tilde{r_d}} (\sigma_j)^2 < \varepsilon^2$. Truncate $U^{\tilde{S}}, V^{\tilde{S}} \in \mathbb{R}^{\tilde{r_d} \times r_d}$ and $S^{\tilde{S}} \in \mathbb{R}^{r_d \times r_d}$.
9: Manipulate $U_{d-1} = Q_{d-1} U^{\tilde{S}} S^{\tilde{S}} \in \mathbb{R}^{r_{d-1}n \times r_d}$ and $U_d = (V^{\tilde{S}})' Q_d' \in \mathbb{R}^{r_d \times r_1 n}$.

Finally, we produced all factors $U_1, \dots, U_d$ needed for the approximant in (18) having the network graph as in figure 4.

**Computational Cost**

The computational cost for the case $d > 3$ is given in Table 3.3.
The overall computational cost is summed up to

$$\mathcal{O}\left(d(k^3 + r^2 nk) + nkr^3\right),$$

where $r = \max_{1 \le t \le d} r_t$. The memory requirement is $\mathcal{O}\left(d(k^3 + rnk)\right)$, where we need $\mathcal{O}(drnk)$ tensor entry evaluations.

**Error behavior**

Each step introduces two types of errors to the approximation. In step $t$ the unfolding $A_t$ is first approximated by a skeleton $U_t (M_t)^\dagger V_t$ (where $(M_t)^\dagger$ is the pseudo inverse of $M_t$) with an accuracy $\delta_t$ (always measured in the Frobenius norm). This error we do not have under control, it heavily depends on the random pivot choice. But experience with Cross Approximation methods shows that if the underlying function $f$ is "nice" (smooth, slowly oscillating, no peaks, etc.) this error will be very small. The second error $\varepsilon_t$ always appears in the truncation of $M_t = U^{M_t} S^{M_t} (V^{M_t})'$ to smaller ranks. This error is controlled by the input parameter $\varepsilon$ (in different ways according to Algorithm 1 or 2). If we denote the

| step | numerical cost | storage | tensor value calls |
|---|---|---|---|
| Opening step | | | |
| 2: | - | $nk$ | $nk$ |
| 3: | - | $k^2$ | - |
| 4: | $\mathcal{O}(k^3)$ | $\mathcal{O}(k(k+r))$ | - |
| 5: | $\mathcal{O}(nkr)$ | - | - |
| Intermediate step | $d-3$ times | $d-3$ times | $d-3$ times |
| 2: | - | $r_t nk$ | $r_t nk$ |
| 3: | - | $k^2$ | - |
| 4: | $\mathcal{O}(k^3)$ | $\mathcal{O}(k(k+r_{t+1}))$ | - |
| 5: | $\mathcal{O}(r_t r_{t+1} nk)$ | - | - |
| Closing step | | | |
| 2: | - | $r_{d-1}nk$ | $r_{d-1}nk$ |
| 3: | - | $k^2$ | |
| 4: | - | $r_1 nk$ | $r_1 nk$ |
| 5: | $\mathcal{O}(k^3)$ | $\mathcal{O}(k(k+\tilde{r}_d))$ | - |
| 6: | $\mathcal{O}((r_1+r_{d-1})nk\tilde{r}_d)$ | - | - |
| 7: | $\mathcal{O}((r_1+r_{d-1})n\tilde{r}_d^2+\tilde{r}_d^3)$ | - | - |
| 8: | $\mathcal{O}(\tilde{r}_d^3)$ | $\mathcal{O}(\tilde{r}_d^2)$ | - |
| 9: | $\mathcal{O}((r_1+r_{d-1})nr_d\tilde{r}_d)$ | - | - |

Table 3.3: Computational cost of the Algorithm for $d > 3$

right-hand side of (18) by $X$, then after using triangle inequality for the approximations and truncations in each step, we end up with something like

$$\|A - X\| \leq \sum_{i=1}^{d-1} \alpha_i(\delta_i + \beta_i \varepsilon_i),$$

where the $\alpha$'s and $\beta$'s are amplification factors depending on the norms of chosen rows and column matrices. For "nice" $f$ and normalized factors in the approximant, $\varepsilon$ plays the dominant role for the final accuracy which can be observed in the experiments.
In contrast to that, the error behavior is quite different, for example, if

$$g(x_1, \ldots, x_d) = \exp\left(-10 * \sqrt{\sum_{i=1}^{d}(x_i - 0.5)^2}\right)$$

on $[0,1]^d$. Then the $\delta_t$'s are dominating and the choice of $\varepsilon$ plays only a minor role. A rigorous error analysis is still desirable and ongoing work. The most promising approach is in the spirit of the recent work in [6], where error bounds for skeleton decompositions of matrices are given with certain probabilities and are based on the notion of coherence.

## 4 Numerical results

To show the quality of our approximation scheme we present the results of some numerical experiments with our favourite function

$$f(x_1, \ldots, x_d) = \frac{1}{\sqrt{1 + x_1^2 + \ldots + x_d^2}}$$

defined on $[0,1]^d$. Obviously this function is very smooth and should not cause any kind of problems in the approximation. But since our goal was to show whether the approximation in the TC format is possible by our method, we found this example sufficient. In case $d = 3$ we took $n = 129$ points in each direction of an uniform grid and show the outcome in table 4.4. Table 4.5 presents the results for $d = 4$, where only $n = 33$ is taken because of the waiting time needed to test the final error.

| $k$ | $\varepsilon$ | QR | $r_1, r_2, r_3$ | error | time |
|---|---|---|---|---|---|
| 10 | 0.1 | + | 3,3,3 (70%) | 1e-05 | 0.01 |
| | | | 2,5,4 (25%) | 1e-04 | |
| | | | 2,4,4 (5%) | 8e-05 | |
| | | - | 3,3,2 (65%) | 3e-04 | |
| | | | 2,5,2 (30%) | 4e-04 | |
| | | | 2,4,2 (5%) | 4e-04 | |
| 10 | 1e-05 | + | 3,3,9 (40%) | 1e-08 | 0.01 |
| | | | 3,3,10 (30%) | 5e-07 | |
| | | | 2,5,9 (20%) | 5e-08 | |
| | | - | 3,3,5 (45%) | 5e-06 | |
| | | | 3,3,6 (35%) | 2e-06 | |
| | | | 2,5,6 (10%) | 3e-06 | |
| 20 | 1e-10 | + | 13,1,9 (65%) | 5e-13 | 0.03 |
| | | | 2,7,16 (15%) | 5e-13 | |
| | | | 3,4,19 (10%) | 6e-12 | |
| | | - | 13,1,7 (60%) | 1e-11 | 0.02 |
| | | | 13,1,6 (15%) | 1e-08 | |
| | | | 3,4,14 (10%) | 7e-11 | |
| 30 | 1e-16 | + - | 19,1,27* (50%) | 3e-14 | 0.06 |
| | | | 3,6,29* (40%) | 6e-13 | |
| | | | 4,5,29* (10%) | 8e-14 | |

Table 4.4: Results for $f$ in $d = 3$

| $k$ | $\varepsilon$ | QR | $r_1, r_2, r_3, r_4$ | error | time |
|---|---|---|---|---|---|
| 10 | 0.1 | + | 3,3,10,3 (70%) | 2e-05 | 0.02 |
| | | | 2,4,10,3 (20%) | 3e-04 | |
| | | | 2,5,10,3 (10%) | 2e-04 | |
| | | - | 3,3,2,2 (75%) | 7e-04 | |
| | | | 2,4,2,2 (20%) | 8e-04 | |
| | | | 2,5,2,2 (5%) | 7e-04 | |
| 10 | 1e-05 | + | 3,3,10,9 (30%) | 5e-07 | 0.02 |
| | | | 2,4,10,9 (30%) | 8e-08 | |
| | | | 3,3,10,10 (25%) | 2e-07 | |
| | | - | 3,3,4,4 (35%) | 2e-05 | |
| | | | 2,4,6,6 (20%) | 6e-06 | |
| | | | 3,3,5,6 (15%) | 9e-06 | |
| 20 | 1e-10 | + | 3,4,20,19 (30%) | 1e-10 | 0.1 |
| | | | 13,1,15,10 (15%) | 5e-12 | 0.2 |
| | | | 3,4,20,20 (15%) | 5e-11 | 0.1 |
| | | - | 13,1,7,8 (35%) | 1e-10 | 0.2 |
| | | | 11,1,7,8 (15%) | 8e-11 | 0.2 |
| | | | 3,4,14,15 (10%) | 2e-10 | 0.1 |
| 30 | 1e-16 | + - | 3,6,29*,29* (55%) | 1e-12 | 0.2 |
| | | | 17,1,25*,27* (25%) | 1e-13 | 0.7 |
| | | | 19,1,25*,28* (15%) | 5e-11 | 0.8 |

Table 4.5: Results for $f$ in $d = 4$

The meaning of the columns in the tables is as follows: Obviously, $k$ is the starting rank (number of randomly chosen columns in each step), $\varepsilon$ is the desired accuracy of the final approximation and the time needed to construct all factors of the approximant measured in seconds is shown in the last column.
The third column specifies the truncation method, as mentioned in the comments after Algorithm 1 and before the closing step in the previous section: '+' means that we perform additional $QR$-decompositions after the closing step and always truncate to machine epsilon in all previous steps and '$-$' refers to the fact that we always truncate with respect to the initial $\varepsilon$ in the Frobenius norm (except for the first step) and can therefore omit the $QR$'s for final reduction.
The main results are displayed in columns 4 and 5: $r_1, \dots, r_d$ shows for fixed parameters $k$ and $\varepsilon$ the three rank distributions appearing most often (within 20 runs) with its cumulative percentage in brackets and the final (averaged) approximation error measured in the Frobenius norm is given. In the rows for $k = 30$ the superscript $*$ at some given rank numbers indicate that these are averaged and the true rank may slightly differ.
The experiments were performed on a Sun/Oracle SPARC Enterprise M3000 Server (shared with other users) with 32GB memory and a SPARC64-VII processor at 2,5 GHz. The code was written in matlab.

**Discussion**

The comparison between the two variants (+ and $-$) follows our expectation. The variant with the $QR$'s at the end is always a bit more accurate, because of the weaker truncation to machine epsilon in all the steps, where without the $QR$'s the procedure leads to a more balanced chain in terms of ranks (best visible in case $k = 10$). In both cases the desired

accuracy $\varepsilon$, that the algorithm gets as an input, is almost always reached. If one needs really precise calculations ($\varepsilon \leq 1e-10$), then larger ranks have to be allowed. Therefore, the starting rank $k$ is chosen to be 20 or 30, but there is no obvious strategy to choose $k$, it is up to heuristics.
The appearance of 1's for $r_2$ is because of the splitting $r = r_1 r_2$ with $r_1 \geq r_2$ and $|r_1 - r_2|$ minimal. If $r$ happens to be a prime, $r_2 = 1$ will be chosen. As mentioned before this is artificial and one could try to modify this. In later work ([13] is not yet published) the splitting is more natural. If $r \leq k$ is the number of singular values of the intersection matrix $M$ that are larger than $eps_{CPU}$, we truncate the size of the involved factors to $\hat{r} = ([\sqrt{r}] + 1)^2$ such that $r \leq \hat{r} \leq k$, where now the splitting for a balanced chain is easy. In case $\varepsilon = 1e-16$ the two variants do not differ much, as expected.
Although we do not compete with other formats, we shortly compare our results with the well-established TT-toolbox of Oseledets, due to a wish of an anonymous referee. In case $d = 3$, $n = 129$ for an accuracy of $1e-06$ (measured in relative Frobenius norm) the TT-ranks for our function $f$ are $(4, 4)$ and for an error of $1e-10$ the toolbox gives $(7, 7)$. For $d = 4$, $n = 33$ the TT-ranks are analogue ($(4, 4, 4)$ for $1e-06$ and $(7, 7, 7)$ for $1e-10$). These results are comparable to the ranks produced by our Algorithm without final $QR$-decomposition (indicated with $-$ in the tables above).

**Special example**
To investigate the important question whether our algorithm is able to reproduce tensors that are given in the TC format already, let us consider an explicit example here. This will also shed some light on the connection between TT ranks and TC ranks.
Let $\{e_j \in \mathbb{R}^r : 1 \leq j \leq r\}$ be the canonical basis of $\mathbb{R}^r$. Let us further define

$$u^{(i,j)} := v^{(i,j)} := w^{(i,j)} := e_i \otimes e_j \in \mathbb{R}^{r^2}, \quad (1 \leq i, j \leq r)$$

and

$$A := \sum_{m_1=1}^{r} \sum_{m_2=1}^{r} \sum_{m_3=1}^{r} u^{(m_1,m_2)} \otimes v^{(m_2,m_3)} \otimes w^{(m_3,m_1)}.$$

Obviously, the ranks of $A$ in the TC format are bounded by $r$. Now let's check the ranks in the TT format, which are given by the ranks of the matricizations (unfoldings) of $A$. Because of symmetry it is sufficient to consider only the first unfolding of $A$

$$\mathcal{M}_1(A) = UW^T$$

where the the matrices $U \in \mathbb{R}^{r^2 \times r^2}$ and $W \in \mathbb{R}^{r^4 \times r^2}$ are defined as follows:

$$\begin{aligned} U &:= \left(u^{(m_1,m_2)} \,:\, 1 \leq m_1, m_2, \leq r\right), \\ W &:= \left(\sum_{m_3=1}^{r} v^{(m_2,m_3)} \otimes w^{(m_3,m_1)} \,:\, 1 \leq m_1, m_2, \leq r\right). \end{aligned}$$

Since $U$ is invertible and $W^T W = r \cdot \mathbf{Id}_{\mathbb{R}^{r\times r}} \otimes \mathbf{Id}_{\mathbb{R}^{r\times r}}$, i.e. $\operatorname{rank}(W) = r^2$, we have $\operatorname{rank}(\mathcal{M}_1(A)) = r^2$. Therefore, all the TT ranks of $A$ are equal to $r^2$.
In such a case where the TC ranks are much smaller than the ranks in the TT format, it would be even more desirable that Algorithm 1 reproduces $A$ exactly. Because of our general Black Box assumption, the position of the columns that carry useful information

in this degenerate situation is not known in advance, so it's of course unlikely to find them here (also in the right order) by a random choice and the approximation often fails then. Consequently, a recovery property can not be expected in general. Still there is some hope that in case of more input information about our underlying function this property could hold. Namely, if we use the expensive SVD-based procedure described at the end of section 2 for this specific example, the original $A$ will be reconstructed perfectly with the correct ranks in many cases. What goes wrong sometimes is the index reordering after the rank splitting in the first step. So if, in addition, we ensure that in Step 1 the left-singular vectors form $U$ as the identity matrix, then we recover $A$ in all cases, which has to be exploited in future work. Here we present the outcome for the procedure described at the end of section 2 with the tolerance $\varepsilon = 1e - 05$, where the third column indicates whether we forced $U$ to be the identity or not.[2]

| $d$ | $n = r^2$ | $U \overset{!}{=} I$ | ranks | error |
|---|---|---|---|---|
| 3 | 4 | + | 2,2,2 | 5.8e-16 |
| | | - | 2,2,4 | 1.7e-16 |
| | 9 | + | 3,3,3 | 2.7e-16 |
| | | - | 3,3,21 | 2.6e-15 |
| | 16 | + | 4,4,4 | 1.6e-16 |
| | | - | 4,4,4 | 1.6e-16 |
| | 25 | + | 5,5,5 | 2.7e-16 |
| | | - | 5,5,5 | 2.7e-16 |
| | 36 | + | 6,6,6 | 1.4e-15 |
| | | - | 6,6,96 | 1.8e-15 |
| | 49 | + | 7,7,7 | 7.4e-16 |
| | | - | 7,7,7 | 7.4e-16 |
| | 64 | + | 8,8,8 | 2.0e-15 |
| | | - | 8,8,120 | 2.5e-15 |
| | 81 | + | 9,9,9 | 1.8e-15 |
| | | - | 9,9,9 | 1.8e-15 |
| | 100 | + | 10,10,10 | 4.9e-16 |
| | | - | 10,10,10 | 4.9e-16 |
| 4 | 4 | + | 2,2,2,2 | 1.6e-16 |
| | | - | 2,2,2,2 | 1.6e-16 |
| | 9 | + | 3,3,3,3 | 3.6e-15 |
| | | - | 3,3,3,3 | 3.6e-15 |
| | 16 | + | 4,4,4,4 | 6.4e-16 |
| | | - | 4,4,20,20 | 1.1e-15 |
| | 25 | + | 5,5,5,5 | 8.0e-16 |
| | | - | 5,5,5,5 | 8.0e-16 |
| | 36 | + | 6,6,6,6 | 1.6e-15 |
| | | - | 6,6,6,6 | 1.6e-15 |
| | 49 | + | 7,7,7,7 | 6.9e-16 |
| | | - | 7,7,7,7 | 6.9e-16 |
| | 64 | + | 8,8,8,8 | 9.2e-16 |
| | | - | 8,8,8,8 | 9.2e-16 |
| | 81 | + | 9,9,9,9 | 6.9e-16 |
| | | - | 9,9,9,9 | 6.9e-16 |
| 5 | 4 | + | 2,2,2,2,2 | 7.9e-16 |
| | | - | 2,2,4,4,4 | 8.4e-16 |
| | 9 | + | 3,3,3,3,3 | 9.5e-16 |
| | | - | 3,3,3,3,3 | 9.5e-16 |
| | 16 | + | 4,4,4,4,4 | 1.3e-15 |
| | | - | 4,4,4,4,4 | 1.3e-15 |
| | 25 | + | 5,5,5,5,5 | 1.5e-15 |
| | | - | 5,5,25,25,25 | 3.3e-15 |
| | 36 | + | 6,6,6,6,6 | 1.2e-15 |
| | | - | 6,6,6,6,6 | 1.2e-15 |
| 6 | 4 | + | 2,2,2,2,2,2 | 3.1e-15 |
| | | - | 2,2,8,8,8,8 | 1.0e-15 |
| | 9 | + | 3,3,3,3,3,3 | 1.2e-15 |
| | | - | 3,3,3,3,3,3 | 1.2e-15 |
| | 16 | + | 4,4,4,4,4,4 | 2.4e-15 |
| | | - | 4,4,4,4,4,4 | 2.4e-15 |
| 7 | 4 | + | 2,2,2,2,2,2,2 | 2.0e-15 |
| | | - | 2,2,8,8,8,8,8 | 2.2e-15 |
| | 9 | + | 3,3,3,3,3,3,3 | 2.9e-15 |
| | | - | 3,3,3,3,3,3,3 | 2.9e-15 |
| 8 | 4 | + | 2,2,2,2,2,2,2,2 | 1.4e-15 |
| | | - | 2,2,2,2,2,2,2,2 | 1.4e-15 |
| | 9 | + | 3,3,3,3,3,3,3,3 | 8.0e-15 |
| | | - | 3,3,3,3,3,3,3,3 | 8.0e-15 |
| 9 | 4 | + | 2,2,2,2,2,2,2,2,2 | 2.7e-15 |
| | | - | 2,2,2,2,2,2,2,2,2 | 2.7e-15 |

Table 4.6: Recovering of TC-ranks for different $d$

## 5 An Outlook

Now that we know how to construct tensor trains and tensor chains by Cross Approximation methods, we believe that it is possible to tackle much more general tensor networks

[2]These results and the implementation in C++ of the SVD-based procedure were produced by our colleague Stefan Handschuh, MPI MIS Leipzig.

by similar techniques. It might just be the question of a clever decomposition of the graph. We postpone this as future work but would like to mention the rough idea here.
Suppose we are given a fixed format characterized by a network graph, say

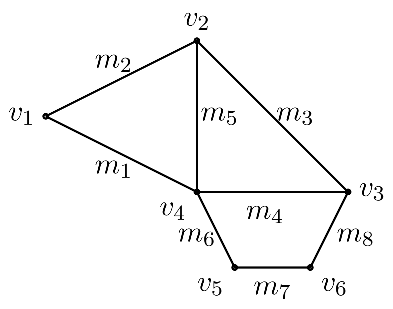

Figure 8: Arbitrary tensor network graph

The task is to approximate a tensor $A_f$ generated as uniform discretization of the function $f(x_1, \dots, x_6)$ within the format given by Figure 8, i.e.

$$A_f(i_1, \dots, i_6) \sim \sum_{m_1=1}^{r_1} \cdots \sum_{m_8=1}^{r_8} v_1(i_1)^{(m_1,m_2)} v_2(i_2)^{(m_2,m_3,m_5)} \dots v_6(i_6)^{(m_7,m_8)} \tag{19}$$

using only a few function values (linear in $n$). The idea is similar to the method treated above, using a pseudo-skeleton approximation successively to certain unfoldings of $A_f$. The figures below illustrate the development of the network graph during the first few steps of the procedure we have in mind.

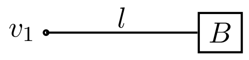

Figure 9: Graph after first step

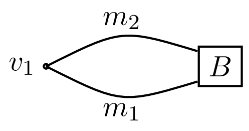

Figure 10: Graph after first index splitting

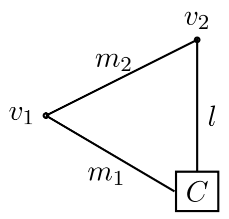

Figure 11: Graph after second step

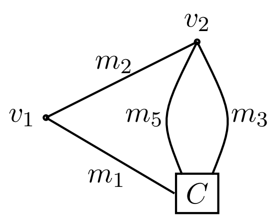

Figure 12: Graph after second index splitting

Recent results ([13]) show that by this strategy it is basically possible to construct any tensor network structure. This structure is coded in a matrix $S$ determining what nodes are connected (in graph theory, this is called adjacency matrix). $S$ is then given as input to the algorithm, which is now able to treat different networks including the Tensor Train and the Tensor Chain formats, where the approximation quality is similar to the examples shown before. In this sense the present text is an important building block.

## References

[1] Bader B.W. and Kolda T.G.: Tensor decomposition and applications. SIAM Review, 51(3), 2009.

[2] Ballani J., Grasedyck L. and Kluge M.: Black Box Approximation of Tensors in Hierarchical Tucker Format. MPI MIS Preprint: 57/2010, accepted for Lin. Alg. Appl.

[3] Bebendorf M.: Approximation of boundary element matrices. Numer. Math. 86: 565-589, 2000.

[4] Bebendorf M.: Adaptive Cross Approximation of Multivariate Functions. Constr. Approx. 34:149-179, 2011.

[5] Bebendorf M.: Hierarchical Matrices. Lectures in computational science and engineering, Vol 63, Springer.

[6] Chiu, J. and Demanet, L.: Sublinear randomized algorithms for skeleton decompositions. arXiv:1110.4193v2 [math.NA].

[7] Ciafa C.F. and Cichocki A.: Generalizing the column-row matrix decomposition to multi-way arrays. Linear Alg. and appl., 433(3), 557-573, 2010.

[8] Dolgov S., Khoromskij B.N. and Oseledets I.V.: Fast solution of multi-dimensional parabolic problems in the TT/QTT-format with initial application to the Fokker-Planck equation. MPI MIS Preprint: 80/2011.

[9] de Lathauwer L., de Moor B. and Vandewalle J.: A multilinear singular value decompostion. SIAM J. Matr. Anal. Appl., 21, 1253-1278, 2000.

[10] de Lathauwer L., de Moor B. and Vandewalle J.: On best rank-1 and rank-$(R_1, R_2, .., R_N)$ approximation of higher-order tensors. SIAM J. Matr. Anal. Appl., 21, 1324-1342, 2000.

[11] Espig M., Grasedyck L. and Hackbusch W.: Black Box low tensor-rank approximation using fiber-crosses. Constr. Approx. 30, 557-597, 2009.

[12] Espig M., Hackbusch W., Handschuh S. and Schneider R.: Optimization Problems in Contracted Tensor Networks. MPI MIS Preprint No. 66, Leipzig 2011.

[13] Espig M., Handschuh S., Khachatryan A, Naraparaju K.K. and Schneider J.: Construction of arbitrary tensor networks. in preparation at MPI MIS Leipzig.

[14] Friedland S., Mehrmann V., Miedlar A. and Nkengla M.: Fast lower rank approximations of matrices and tensors. Electronic Journal of Linear Algebra, V0ol 22, 1031-1048, 2011.

[15] Gantmacher F.R.: Theory of Matrices. Chelsea, Newyork, 1959.

[16] Goreinov S. A.: On cross approximation of multi-index array. Dokaldy Math., 420(4), 404-406, 2008.

[17] Goreinov S. A., Oseledets I. V., Savostyanov D. V. et al.: How to find a good submatrix. Research Report 08-10, Kowloon Tong, Hong Kong: ICM HKBU, 2008.

[18] Goreinov S.A., Tyrtyshnikov E.E. and Zamarashkin N.L.: A theory of pseudo-skeleton approximations. Lin. Alg. Appl. 261, 1-21, 1997.

[19] Goreinov S.A., Zamarashkin N.L. and Tyrtyshnikov E.E.: Pseudo-skeleton approximations by matrices of maximal volume. Mathematical Notes, vol 62, No. 4, 1997.

[20] Goreinov S. A. and Tyrtyshnikov E. E.: The maximal-volume concept in approximation by low-rank matrices. Contemporary Mathematics, Vol. 208, 47-51, 2001.

[21] Grasedyck L.: Hierarchical Singular Value Decomposition of Tensors. SIAM J. Matrix Anal. Appl. 31, 2029-2054, 2010.

[22] Golub G. and Van Loan C.: Matrix Computations. -3 edition-John Hopkins Univ. Press., 1996.

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

[24] Hackbusch W.: Tensor Spaces and Numerical Tensor Calculus. Springer Series in Computational Mathematics, Vol. 42.

[25] Khoromskaia V., Andrae D. and Khoromskij B.N.: Fast and Accurate Tensor Calculation of the Fock Operator in a General Basis. MPI MIS Preprint 4/2012.

[26] Khoromskij B.N.: $O(d \log N)$-Quantics Approximation of N-d Tensors in High-Dimensional Numerical Modeling. Constr. Approx. 34(2), 257-280, 2011.

[27] Landsberg J. M., Yang Qi and Ke Ye: On the geometry of tensor network states. arXiv:1105.4449 [math.AG], 2011.

[28] Mahoney M. W., Maggioni M. and Drineas P.: Tensor -CUR decompositions for tensor based data. SIAM J. Matr. Anal. Appl., 30(3), 957-987, 2008.

[29] Micchelli C.A. and Pinkus A.: Some Problems in the Approximation of functions of Two Variables and $n$-Widths of Integral Operators. Jour. Approx. Theo. **24**, 51-77 (1978).

[30] Naraparaju K.K. and Schneider J.: Generalized Cross Approximation for 3d-tensors. Comput Visual Sci 14(3), 105-115, 2011.

[31] Savostyanov D. V. and Oseledets I.V: Fast adaptive interpolation of multidimensional arrays in tensor train format. In proc. of 7th Intern. Workshop on Multidim. Systems (nDS). IEEE, 2011.

[32] Tyrtyshnikov E.E: Incomplete Cross Approximation in the Mosaic-Skeleton method. Computing, 64(4), 367-380, 2000.

[33] Oseledets I.V., Savostianov D.V. and Tyrtyshnikov E.E.: Tucker dimensionality reduction of three-dimensional arrays in linear time. SIAM J. Matrix Anal. Appl. 30(3), 939-956, 2008.

[34] Oseledets I.V and Tyrtyshnikov E.E.: TT-Cross approximation for multidimensional arrays. Lin. Alg. Appl. 432(1), 2010, 70-88.

[35] Oseledets I.V.: Tensor-Train Decomposition. SIAM J. Sci. Comp, 33 (5), 2295-2317, 2011.

[36] Schneider J.: Error estimates for two-dimensional Cross Approximation. J. Approx. Theory 162 (9), 1685-1700, 2010.

[37] Zhu X. and Lin W.: Randomised pseudo-skeleton approximation and its application in electromagnetics, Electronics Letters, Vol 47, NO. 10, May 2011.