Max-Planck-Institut
für Mathematik
in den Naturwissenschaften
Leipzig

Generalized Cross Approximation for 3d-tensors

by

Kishore Kumar Naraparaju, and Jan Schneider

Preprint no.: 29 2010

Generalized Cross Approximation for 3d-tensors

Kishore Kumar Naraparaju, Jan Schneider

Max Planck Institute for Mathematics
in the Sciences, Leipzig

Abstract

In this article we present a generalized version of the Cross Approximation for 3d- tensors. The given tensor $a \in \mathbb{R}^{n\times n\times n}$ is represented as a matrix of vectors and 2d adaptive Cross Approximation is applied in a nested way to get the tensor decomposition. The main focus lies on theoretical issues of the construction such as the desired interpolation property or the explicit formulas for the vectors in the decomposition. The computational complexity of the proposed algorithm is shown to be linear in n.

AMS Subject Classification: 41A80,41A63,15A69
Keywords: Adaptive Cross Approximation, tensor decompositon, pivot strategy, maximum norm

1 Introduction

The approximation of multidimensional arrays (tensors) generated by the evaluation of functions by some tensor product structures is nowadays one of the major issues in numerical analysis. It has many applications in areas like chemometrics, electronical engineering, quantum chemistry, etc. A thorough survey by T. G. Kolda and B. W. Bader in [9] provides an overview of higher order tensor decompositions. The general aim when working with these large objects is to reduce the computational complexity significantly. For the latest developements in this area, one can look at [3, 6, 7, 8, 11].

The main focus of this paper is on the method of Cross Approximation. It is one of the methods that proved its success in low dimensions, see [2] also for references to other methods. It was introduced for $d = 2$ by Bebendorf in [1] as *adaptive cross approximation* (ACA), with forerunners by Tyrtyshnikov and others (see for instance [5]). It uses only a small portion of the original array entries for the corresponding approximation and constructs a sum of products of one-dimensional restrictions of a two-dimensional function

$$f(x, y) \sim \sum_{i,j=1}^{k} c_{ij} f(x_i, y) f(x, y_j)$$

with points x_i, y_j and coefficients c_{ij}. This procedure has a lot of advantages, for instance the so-called rank and interpolation properties, see [12], where the algorithm was called CA2D. Therefore, it is desirable to extend the method to functions of more than two variables. This was done in different ways, see for instance [3, 10]. A natural extension of the bivariate method to functions of three or four variables is proposed in [2] and can be seen as nested ACA.

In the present article we will investigate another version of three-dimensional Cross Approximation. Let us briefly explain the basic idea of our approach to approximate a 3d-tensor, which also works in case $d = 4$, where it was originally born, but that is outside the scope of this paper.

We consider a tensor $a \in \mathbb{R}^{n\times n\times n}$ generated by a function $f(x, y, z)$ defined on $[0, 1]^3$, where

the grid in each direction consists of n points, i.e., we have $a_{ijk} = f(x(i), y(j), z(k))$ for $1 \leq i, j, k \leq n$. The main motivation is to save as much as possible from the nice properties of CA2D, such as rank and interpolation properties mentioned above. We follow the simple idea of interpreting a as matrix

$$a = \begin{pmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & & \vdots \\ A_{n1} & \cdots & A_{nn} \end{pmatrix},$$

where all the entries $A_{\mu\nu} = (a_{\mu\nu 1}, \ldots, a_{\mu\nu n})$ are now vectors of length n for all $1 \leq \mu, \nu \leq n$. Looking at a as matrix to approximate, we can immediately apply CA2D. Since the entries are now vectors, we have to use an appropriate kind of multiplication, so we rely our method on the pointwise (or Hadamard) product, denoted by $\odot$. We choose A_{uv} as our first pivot and define the first approximation to $a = a^{(0)}$ by

$$a^{(1)} = \begin{pmatrix} A^{(1)}_{11} & \cdots & A^{(1)}_{1n} \\ \vdots & & \vdots \\ A^{(1)}_{n1} & \cdots & A^{(1)}_{nn} \end{pmatrix} \quad \text{with } A^{(1)}_{\mu\nu} = A_{u\nu} \odot A_{\mu v} \odot \text{ inv}(A_{uv}) \tag{1}$$

for all $1 \leq \mu, \nu \leq n$, where $\text{inv}(A)$ denotes the vector of pointwise inverted entries of A. This, of course, is only allowed, if

$$\min_{1 \leq k \leq n} |a_{uvk}| > 0,$$

or in other words: there must not a single entry be equal 0. In the language of functions formula (1) reads as

$$f^{(1)}(x, y, z) = \frac{f(x_1, y, z) f(x, y_1, z)}{f(x_1, y_1, z)} \tag{2}$$

for fixed pivot coordinates (x_1, y_1) with the restriction $f(x_1, y_1, z) \neq 0$ for all $z \in [0, 1]$. That is a strong restriction for the pivot element and we treat it within our discussion concerning the pivot strategy in section 3. One can easily see that the remainder

$$r^{(1)} = a - a^{(1)} \tag{3}$$

contains a cross of 0-vectors. That is the first step on the way to the interpolation property. Let's go on with this idea in a more compact notation and define recursively for $1 \leq t \leq T$

$$a^{(t)} = \begin{pmatrix} A^{(t)}_{11} & \cdots & A^{(t)}_{1n} \\ \vdots & & \vdots \\ A^{(t)}_{n1} & \cdots & A^{(t)}_{nn} \end{pmatrix} \quad \text{with } A^{(t)}_{\mu\nu} = R^{(t-1)}_{u_t\nu} \odot R^{(t-1)}_{\mu v_t} \odot \text{ inv}(R^{(t-1)}_{u_t v_t})$$

for all $1 \leq \mu, \nu \leq n$ (R is here always a vector and denotes an entry in r, which is the error matrix of vectors), where we always chose $R^{(t-1)}_{u_t v_t}$ as t-th pivot with

$$\min |R^{(t-1)}_{u_t v_t}| = \min_{1 \leq k \leq n} |r^{(t-1)}_{u_t v_t k}| > 0$$

in order to ensure pointwise multiplication. That gives the remainder

$$r^{(T)} = a - s^{(T)} = a - \sum_{t=1}^{T} a^{(t)},$$

which in entry-wise notation looks like

$$R_{\mu\nu}^{(T)} = A_{\mu\nu} - \sum_{t=1}^{T} R_{u_t\nu}^{(t-1)} \odot R_{\mu v_t}^{(t-1)} \odot \text{ inv}(R_{u_t v_t}^{(t-1)}) \tag{4}$$

and because of its recursive character we have

$$R_{\mu\nu}^{(T)} = R_{\mu\nu}^{(T-1)} - R_{u_T\nu}^{(T-1)} \odot R_{\mu v_T}^{(T-1)} \odot \text{ inv}(R_{u_T v_T}^{(T-1)}). \tag{5}$$

On the level of functions this looks like

$$r^{(T)}(x,y,z) = r^{(T-1)}(x,y,z) - \frac{r^{(T-1)}(x_T,y,z)r^{(T-1)}(x,y_T,z)}{r^{(T-1)}(x_T,y_T,z)}, \tag{6}$$

where we can see that the scheme follows the same structure as CA2D. More general, we realize that (6) is an example for *incremental approximation* as it is called in [2]. This procedure results in the approximation

$$f(x,y,z) \sim \sum_{t=1}^{T} g_t(x,z)h_t(y,z), \tag{7}$$

with special functions g_t, h_t that are still two-dimensional and therefore another approximation by CA2D is required to get the complete decomposition (i.e to represent f as a sum of products of one dimensional functions). This is done with a linear complexity in n. Explicit formulas are derived for the vectors in the tensor decomposition. The details of the complete algorithm are described in section 3.

In Section 2, basics of the adaptive cross approximation in two dimensions are discussed. The explicit formulas for the vectors in the decomposition of a 3d-tensor for the existing algorithm from [2] are also deduced. In Section 3, the proposed algorithm is derived and its computational complexity is discussed. The details of the pivot strategy are shown. Finally, in Section 4 we present a few numerical examples to show the efficiency of the method.

2 Preliminaries

In this section we want to settle the theoretical background for our method. For simplicity we start again with the two-dimensional case. Let $f : [0,1]^2 \longrightarrow \mathbb{R}$, then the recursion $R_0(x,y) = f(x,y)$ and

$$R_k(x,y) = R_{k-1}(x,y) - \frac{R_{k-1}(x,y_k)R_{k-1}(x_k,y)}{R_{k-1}(x_k,y_k)} \quad \text{for } k \in \mathbb{N}, \tag{8}$$

with points x_k, y_k chosen such that $R_{k-1}(x_k,y_k) \neq 0$, is the heart of CA2D, compare [1, 12]. It gives the remainder after k iteration steps of the approximation $S_k(x,y) = f(x,y) - R_k(x,y)$. If we set $u_k(x) = R_{k-1}(x,y_k)$ and $v_k(y) = R_{k-1}(x_k,y)$, then using (8) we get

$$\begin{aligned} S_k(x,y) &= f(x,y) - R_k(x,y) \\ &= f(x,y) - \left(R_{k-1}(x,y) - \frac{u_k(x)}{u_k(x_k)} v_k(y) \right) \\ &\vdots \\ &= f(x,y) - \left(R_0(x,y) - \frac{u_1(x)}{u_1(x_1)} v_1(y) - \cdots - \frac{u_k(x)}{u_k(x_k)} v_k(y) \right) \\ &= \sum_{i=1}^{k} \frac{u_i(x)}{u_i(x_i)} v_i(y), \end{aligned} \tag{9}$$

where we realize that it has the desired tensor product structure of separated variables. By a similar calculation one can even derive the explicit formulas

$$u_k(x) = f(x, y_k) - \sum_{l=1}^{k-1} \frac{u_l(x)}{u_l(x_l)} v_l(y_k) \tag{10}$$

and

$$v_k(y) = f(x_k, y) - \sum_{l=1}^{k-1} \frac{u_l(x_k)}{u_l(x_l)} v_l(y), \tag{11}$$

see also [2].

Now we turn to the three-dimensional case. As mentioned in the Introduction, in [2] Bebendorf proposed a natural way to extend the above procedure to functions $f(x, y, z)$. Here we restrict the domain of definition to $[0, 1]^3$. Let us start with the following recursion to describe his method: $R_0(x, y, z) = f(x, y, z)$ and

$$R_k(x, y, z) = R_{k-1}(x, y, z) - \frac{R_{k-1}(x, y_k, z_k)}{R_{k-1}(x_k, y_k, z_k)} R_{k-1}(x_k, y, z) \quad \text{for } k \in \mathbb{N}, \tag{12}$$

where the analogy to (8) is obvious and the points x_k, y_k, z_k are chosen such that $R_{k-1}(x_k, y_k, z_k) \neq 0$. We set $u_k(x) = R_{k-1}(x, y_k, z_k)$ and $V_k(y, z) = R_{k-1}(x_k, y, z)$, so we get analogously to the above calculation

$$\begin{aligned} S_k(x, y, z) &= f(x, y, z) - R_k(x, y, z) \\ &= f(x, y, z) - \left(R_0(x, y, z) - \frac{u_1(x)}{u_1(x_1)} V_1(y, z) - \cdots - \frac{u_k(x)}{u_k(x_k)} V_k(y, z) \right) \\ &= \sum_{i=1}^{k} \frac{u_i(x)}{u_i(x_i)} V_i(y, z), \end{aligned} \tag{13}$$

where in contrast to (9) the functions V_i are still two-dimensional. We will now investigate the explicit formulas for u_k and V_k. In analogy to (10) and (11) we get

$$u_k(x) = f(x, y_k, z_k) - \sum_{l=1}^{k-1} \frac{u_l(x)}{u_l(x_l)} V_l(y_k, z_k) \tag{14}$$

and

$$V_k(y, z) = f(x_k, y, z) - \sum_{l=1}^{k-1} \frac{u_l(x_k)}{u_l(x_l)} V_l(y, z). \tag{15}$$

Since V_k are bivariate functions the complexity can not be reduced to be linear in n while working with them directly. That is why Bebendorf ([2]) uses the following recursion: $R_0(x, y, z) = f(x, y, z)$ and

$$R_k(x, y, z) = R_{k-1}(x, y, z) - \frac{R_{k-1}(x, y_k, z_k)}{R_{k-1}(x_k, y_k, z_k)} \mathcal{A}(R_{k-1})(x_k, y, z) \quad \text{for } k \in \mathbb{N}, \tag{16}$$

where he substitutes the functions $V_k(y, z) = R_{k-1}(x_k, y, z)$ by their approximations via ACA, denoted by $\mathcal{A}(R_{k-1})(x_k, y, z)$. In each iteration step $k = 1, 2, \ldots$ of the recursion (16) he performs a two-dimensional Cross Approximation for the function V_k with k' steps in the inner loop. Since that separates the variables y and z it gives the desired tensor product structure of the

approximant. Now we also investigate the explicit formulas for this relation. For the inner recursion we introduce the following notation: $r_0^{(k)}(y,z) = V_k(y,z)$ and

$$r_{k'}^{(k)}(y,z) = r_{k'-1}^{(k)}(y,z) - \frac{r_{k'-1}^{(k)}(y,z_{k'}^{(k)})}{r_{k'-1}^{(k)}(y_{k'}^{(k)},z_{k'}^{(k)})} r_{k'-1}^{(k)}(y_{k'}^{(k)},z) \quad \text{for } k' \in \mathbb{N}, \tag{17}$$

with points $y_{k'}^{(k)}, z_{k'}^{(k)}$ chosen such that $r_{k'-1}^{(k)}(y_{k'}^{(k)}, z_{k'}^{(k)}) \neq 0$. Then, following the usual scheme, we set $v_{k'}^{(k)}(y) = r_{k'-1}^{(k)}(y, z_{k'}^{(k)})$ and $w_{k'}^{(k)}(z) = r_{k'-1}^{(k)}(y_{k'}^{(k)}, z)$ and get

$$\begin{aligned} s_{k'}^{(k)}(y,z) &= V_k(y,z) - r_{k'}^{(k)}(y,z) \\ &= \sum_{\mu=1}^{k'} \frac{v_\mu^{(k)}(y)}{v_\mu^{(k)}(y_\mu^{(k)})} w_\mu^{(k)}(z) \end{aligned} \tag{18}$$

for the inner approximation of V_k. Also here we can get the following explicit formulas:

$$v_{k'}^{(k)}(y) = V_k(y, z_{k'}^{(k)}) - \sum_{j=1}^{k'-1} \frac{v_j^{(k)}(y)}{v_j^{(k)}(y_j^{(k)})} w_j^{(k)}(z_{k'}^{(k)}) \tag{19}$$

and

$$w_{k'}^{(k)}(z) = V_k(y_{k'}^{(k)}, z) - \sum_{j=1}^{k'-1} \frac{v_j^{(k)}(y_{k'}^{(k)})}{v_j^{(k)}(y_j^{(k)})} w_j^{(k)}(z). \tag{20}$$

But these equalities take only the ongoing inner approximation into account and not all the previous steps, but that can be merged and be described explicitly: Starting with the formulas (14), (19) and (20) and substituting V_k with its representation (15) while the functions V_l from all the previous steps are substituted with its inner approximations (18), we end up with the final constructive formulas for the functions $u_k(x)$, $v_{k'}^{(k)}(y)$ and $w_{k'}^{(k)}(z)$

$$u_k(x) = f(x, y_k, z_k) - \sum_{l=1}^{k-1} \frac{u_l(x)}{u_l(x_l)} \sum_{\mu=1}^{k'} \frac{v_\mu^{(l)}(y_k)}{v_\mu^{(l)}(y_\mu^{(l)})} w_\mu^{(l)}(z_k) \tag{21}$$

$$v_{k'}^{(k)}(y) = f(x_k, y, z_{k'}^{(k)}) - \sum_{l=1}^{k-1} \frac{u_l(x_k)}{u_l(x_l)} \sum_{\mu=1}^{k'} \frac{v_\mu^{(l)}(y)}{v_\mu^{(l)}(y_\mu^{(l)})} w_\mu^{(l)}(z_{k'}^{(k)}) - \sum_{j=1}^{k'-1} \frac{v_j^{(k)}(y)}{v_j^{(k)}(y_j^{(k)})} w_j^{(k)}(z_{k'}^{(k)}) \tag{22}$$

$$w_{k'}^{(k)}(z) = f(x_k, y_{k'}^{(k)}, z) - \sum_{l=1}^{k-1} \frac{u_l(x_k)}{u_l(x_l)} \sum_{\mu=1}^{k'} \frac{v_\mu^{(l)}(y_{k'}^{(k)})}{v_\mu^{(l)}(y_\mu^{(l)})} w_\mu^{(l)}(z) - \sum_{j=1}^{k'-1} \frac{v_j^{(k)}(y_{k'}^{(k)})}{v_j^{(k)}(y_j^{(k)})} w_j^{(k)}(z), \tag{23}$$

which can be used directly to implement the method. Putting (18) also into (13), we finally see that the original function f is approximated by

$$f(x,y,z) \sim \sum_{i=1}^{k} \frac{u_i(x)}{u_i(x_i)} \sum_{\mu=1}^{k'} \frac{v_\mu^{(i)}(y)}{v_\mu^{(i)}(y_\mu^{(i)})} w_\mu^{(i)}(z) \tag{24}$$

within this concept.

3 Our approach

3.1 The outer loop

As indicated in the Introduction (formula (6)) we follow a different way to generalize the recursion (8) for three-dimensional functions. To describe this we start with the recursion for the outer loop, it reads as $R_0(x,y,z) = f(x,y,z)$ and

$$R_k(x,y,z) = R_{k-1}(x,y,z) - \frac{R_{k-1}(x,y_k,z)R_{k-1}(x_k,y,z)}{R_{k-1}(x_k,y_k,z)}, \tag{25}$$

for $k \in \mathbb{N}$. This again is a possible analogue to (8), where the points x_k, y_k are chosen such that $R_{k-1}(x_k,y_k,z) \neq 0$, for all $z \in [0,1]$. Setting $U_k(x,z) = R_{k-1}(x,y_k,z)$ and $V_k(y,z) = R_{k-1}(x_k,y,z)$ and following the same kind of calculation we have already done several times before, we get

$$S_k(x,y,z) = f(x,y,z) - R_k(x,y,z) = \sum_{i=1}^{k} \frac{U_i(x,z)}{U_i(x_i,z)} V_i(y,z) \tag{26}$$

with

$$U_k(x,z) = f(x,y_k,z) - \sum_{l=1}^{k-1} \frac{U_l(x,z)}{U_l(x_l,z)} V_l(y_k,z) \tag{27}$$

and

$$V_k(y,z) = f(x_k,y,z) - \sum_{l=1}^{k-1} \frac{U_l(x_k,z)}{U_l(x_l,z)} V_l(y,z). \tag{28}$$

Now both U_k and V_k are still two-dimensional and have to be approximated by CA2D again in order to ensure linear complexity in the problem size n. This will be explained in the next subsection.

Now, for a moment, we keep the point of view, that calculations with the matrices U_k, V_k would not be to expensive and we are focusing on the accuracy of the approximation (26), in other words, we use the exact matrices and not their approximations to end up with a structure like (7). Then we can preserve some of the nice properties of CA2D (compare [12]), mentioned in the Introduction and state their analogues here. We start with the interpolation property using the vector-wise notation introduced before:

Proposition 3.1 *Whenever $\mu \in \{u_1, \ldots, u_T\}$ or $\nu \in \{v_1, \ldots, v_T\}$ we have*

$$R_{\mu\nu}^{(T)} = 0.$$

Proof For $T = 1$ formula (3) gives the assertion. Now, by induction, assume that $R_{\mu\nu}^{(T-1)} = 0$, whenever $\mu \in \{u_1, \ldots, u_{T-1}\}$. Then we find by (5)

$$R_{u_t\nu}^{(T)} = R_{u_t\nu}^{(T-1)} - R_{u_T\nu}^{(T-1)} \odot R_{u_tv_T}^{(T-1)} \odot \operatorname{inv}(R_{u_Tv_T}^{(T-1)}) = 0 - 0 \tag{29}$$

for $1 \leq t \leq T-1$. Furthermore, we have

$$\begin{aligned} R_{u_T\nu}^{(T)} &= R_{u_T\nu}^{(T-1)} - R_{u_T\nu}^{(T-1)} \odot R_{u_Tv_T}^{(T-1)} \odot \operatorname{inv}(R_{u_Tv_T}^{(T-1)}) \\ &= R_{u_T\nu}^{(T-1)} - R_{u_T\nu}^{(T-1)} = 0. \end{aligned}$$

Since the same arguments hold for ν instead of μ the proposition is proved.

□

Remark 3.2 *In the language of functions this looks more familiar to us and also more readable, i.e.,* $R_T(x,y,z) = f(x,y,z) - S_T(x,y,z) = 0$ *for all* $z \in [0,1]$ *whenever* $x \in \{x_1,\dots,x_T\}$ *or* $y \in \{y_1,\dots,y_T\}$*, compare with Proposition 3.1 in [12].*

This interpolation property is the point we are most interested in preserving within our concept. We will come back to this task in the next subsection.
Now we want to prove the analogue to the so-called rank property of CA2D as it is stated in Proposition 3.2 in [12].

Proposition 3.3 *If the function* f *can be written in the form*

$$f(x,y,z) = \sum_{t=1}^{T} g_t(x,z)h_t(y,z), \tag{30}$$

with arbitrary functions g_t, h_t*, then the above procedure gives* $f - S_T = 0$ *on the whole cube* $[0,1]^3$.

In the case, where T in the formula above is the smallest possible number of summands for such a representation, we say that f has (x,y)-separation rank T.

Proof We will prove that $R_\theta = f - S_\theta$ has (x,y)-separation rank $T-\theta$ for $\theta \in \{0,1,\dots,T\}$ by induction. For $\theta = 0$ there is nothing to prove, so let for $\theta < T$ the function R_θ have (x,y)-separation rank $T-\theta$. For fixed $z \in [0,1]$ we define

$$V = V_z = \operatorname{span}\{R_\theta(\cdot,y,z) : y \in [0,1]\}$$

and

$$V' = V'_z = \operatorname{span}\{R_{\theta+1}(\cdot,y,z) : y \in [0,1]\}.$$

We know that $\dim V = T-\theta$ and want to show $\dim V' = T-(\theta+1)$. For each $y \in [0,1]$ we have by construction

$$R_{\theta+1}(\cdot,y,z) = R_\theta(\cdot,y,z) - \frac{R_\theta(\cdot,y_{\theta+1},z)R_\theta(x_{\theta+1},y,z)}{R_\theta(x_{\theta+1},y_{\theta+1},z)}$$

and verify that both terms on the right hand-side belong to V, hence $V' \subset V$. In particular $R_\theta(\cdot,y_{\theta+1},z) \in V$, but because its value at the point $x_{\theta+1}$ does not vanish, whereas $R_{\theta+1}(x_{\theta+1},y,z) = 0$ for all $y \in [0,1]$, there is no representation of $R_\theta(\cdot,y_{\theta+1},z)$ as a linear combination of functions $R_{\theta+1}(\cdot,y,z)$, hence $R_\theta(\cdot,y_{\theta+1},z) \notin V'$. It follows $\dim V' < \dim V$ and because those dimensions can differ at most by one, we get $\dim V' = \dim V - 1 = T-\theta-1$. Now we know that for all $y \in [0,1]$ we have for $R_{\theta+1}(\cdot,y,z) \in V'$ a representation

$$R_{\theta+1}(x,y,z) = \sum_{t=1}^{T-\theta-1} \alpha_t(y,z)\varphi_t(x,z) \tag{31}$$

with coefficients α_t and basis functions φ_t. So we conclude that $R_{\theta+1}$ has (x,y)-separation rank $T-(\theta+1)$.

□

Remark 3.4 *It is obvious that condition* (30) *is crucial for some kind of rank property because it coincides with the structure of the approximant given in* (26).

These properties could easily be verified by numerical experiments. When implementing those kind of methods one always has to think about an appropriate pivot strategy, see [1, 12]. In this outer loop we have to fix the coordinates x_k, y_k in each step, compare (25). We choose them completely randomly, except for the first one, where we always set $x_1 = 1/2, y_1 = 1/2$. Of course, for all $k = 1, 2, \ldots$ we have to ensure that

$$R_{k-1}(x_k, y_k, z) \neq 0 \quad \text{for all } z \in [0, 1], \tag{32}$$

but if that accidently fails we just try another random point. If this condition cannot be satisfied by any choice of pivots, the algorithm stops. As a consequence of this random pivot strategy the number of possible iterations may vary. Still we mainly focus on the accuracy of the method, therefore we test it on small grids only ($n = 129$). Table 3.1 illustrates that the outcome is not always satisfactory.

f		g		h	
err	iter	err	iter	err	iter
$5,82e-13$	12	$1,24e-02$	10	$2,84e-01$	1
$2,37e-13$	10	$2,39e-02$	10	$2,84e-01$	1
$5,01e-13$	10	$1,78e-02$	10	$2,84e-01$	1
$2,08e-13$	12	$2,09e-02$	11	$2,84e-01$	1
$8,94e-13$	12	$2,15e-02$	11	$2,84e-01$	1

Table 3.1: Outer loop with random pivots

Here we used the following notation:
$f(x, y, z) = 1/(1 + x + y + z)$,
$g(x, y, z) = \exp(-10\sqrt{(x - 1/2)^2 + (y - 1/2)^2 + (z - 1/2)^2})$ and
$h(x, y, z) = \exp(-xyz)$.

The column labelled "err" shows the maximum error

$$\max_{x,y,z \in [0,1]^3} |R_k(x, y, z)|$$

after the corresponding number of iterations, which column is labelled by "iter". We see that for f the method produces a very accurate approximant. For g the number of possible steps is too small to reach a better accuracy, where for h the algorithm stops after the first iteration already, because there are no points x_2, y_2 such that $R_1(x_2, y_2, z) \neq 0$ for all $z \in [0, 1]$. Therefore, we have to find a way to circumvent condition (32). Our way to do that is very simple. Since we are interested in accuracy up to $\varepsilon = 10^{-16}$, we just substitute all entries in the pivot vector having absolute value smaller than ε by ε. Of course, we do that only, if no pivot that fulfills condition (32) can be found. In that case we choose as pivot the vector with the largest maximum norm and substitute all entries that are too small. Now the algorithm stops when a prescribed maximum error $\varepsilon_0 = 10^{-13}$ is reached. Table 3.2 shows the satisfactory consequence of the supplemented pivot strategy, where we used the same notation as before.
Here the accuracy is fine in all three cases because the algorithm can run long enough to reach a small error. The deterministic outcome for h grounds on the fact that by the above strategy the pivots after the first step are always found as the vector with the largest entry.

f		g		h	
err	iter	err	iter	err	iter
$8,82e-14$	14	$1,50e-14$	24	$3,12e-15$	9
$8,68e-14$	14	$8,29e-15$	24	$3,12e-15$	9
$7,02e-14$	13	$7,44e-14$	23	$3,12e-15$	9
$4,82e-14$	11	$6,53e-14$	23	$3,12e-15$	9
$6,58e-14$	11	$2,60e-14$	24	$3,12e-15$	9

Table 3.2: Outer loop with refined pivot strategy

In terms of accuracy and stability we could be satisfied with the results shown in Table 3.2, but of course the question arises, what happens if we increase the grid size enormously. Then working with the exact matrices generated by U_k and V_k is not cheap anymore because the complexity involves n^2. That is why we add in each step of the outer loop an inner loop which approximates the corresponding matrix by a sum of products of vectors via CA2D, as will be described in the sequel.

3.2 Adding the inner loops

The modified recursion for the outer loop now reads as $R_0(x,y,z) = f(x,y,z)$ and

$$R_k(x,y,z) = R_{k-1}(x,y,z) - \frac{\mathcal{A}(R_{k-1})(x,y_k,z)\mathcal{A}(R_{k-1})(x_k,y,z)}{\mathcal{A}(R_{k-1})(x_k,y_k,z)}, \tag{33}$$

for $k \in \mathbb{N}$, where the points x_k, y_k are chosen such that

$$|\mathcal{A}(R_{k-1})(x_k,y_k,z)| \geq c_{\min}^{(k)} > 0, \quad \text{for all } z \in [0,1], \tag{34}$$

and $\mathcal{A}(g)$ denotes again the approximation of the two-dimensional function g by CA2D. In contrast to (25) we perform here in each step $k = 1, 2, \ldots$ the approximation of the bivariate error functions $R_{k-1}(x,y_k,z)$ and $R_{k-1}(x_k,y,z)$ within k' steps of an inner loop. For simplicity we keep k' to be constant for our theoretical investigations.
One of our main questions was about the interpolation property. How much of it is preserved now in this nested application of CA2D ? The answer is given by the following Proposition.

Proposition 3.5 *Let $J \subset \{1,\ldots,k'\}$. If $z_j^{(i)} = \hat{z}_j^{(i)} = z_j^*$ for all $i = 1,\ldots,k$ and all $j \in J$, then*

$$R_i(x,y,z_j^*) = 0 \quad \text{for } i = 1,\ldots,k \text{ and } j \in J,$$

whenever $x \in \{x_1,\ldots,x_i\}$ or $y \in \{y_1,\ldots,y_i\}$.

This interesting result tells us the following: If one chooses for all the outer loops some of the pivot coordinates (maybe all) in z-direction to be equal in both inner loops , say equal to z_j^*, then one gets vanishing errors on the crosses of the outer pivot coordinates on the faces determined by those z_j^*.

Proof The proof follows the same idea as used for Proposition 3.1. First we check that for $i = 1$ we have

$$\begin{aligned} R_1(x_1,y,z_j^*) &= f(x_1,y,z_j^*) - \frac{\mathcal{A}(f)(x_1,y_1,z_j^*)\mathcal{A}(f)(x_1,y,z_j^*)}{\mathcal{A}(f)(x_1,y_1,z_j^*)} \\ &= f(x_1,y,z_j^*) - \mathcal{A}(f)(x_1,y,z_j^*) = 0 \end{aligned}$$

because of the $2d$-interpolation property in the inner loop. Then, by induction, we assume that $R_{i-1}(x, y, z_j^*) = 0$ for all $x \in \{x_1, \ldots, x_{i-1}\}$. Now for those x we find

$$\begin{aligned} R_i(x, y, z_j^*) &= R_{i-1}(x, y, z_j^*) - \frac{\mathcal{A}(R_{i-1})(x, y_i, z_j^*)\mathcal{A}(R_{i-1})(x_i, y, z_j^*)}{\mathcal{A}(R_{i-1})(x_i, y_i, z_j^*)} \\ &= 0 - R_{i-1}(x, y_i, z_j^*)\frac{R_{i-1}(x_i, y, z_j^*)}{R_{i-1}(x_i, y_i, z_j^*)} = 0 \end{aligned}$$

because of our induction assumption. Here we used the $2d$-interpolation property again for the inner approximations. It remains to check the case $x = x_i$, but that is easy, since

$$\begin{aligned} R_i(x_i, y, z_j^*) &= R_{i-1}(x_i, y, z_j^*) - \frac{\mathcal{A}(R_{i-1})(x_i, y_i, z_j^*)\mathcal{A}(R_{i-1})(x_i, y, z_j^*)}{\mathcal{A}(R_{i-1})(x_i, y_i, z_j^*)} \\ &= R_{i-1}(x_i, y, z_j^*) - R_{i-1}(x_i, y, z_j^*) = 0, \end{aligned}$$

by the same $2d$-interpolation argument. Since the analogue way is possible for y instead of x the proof is finished.

□

Remark 3.6 *Proposition 3.5 seems to suggest a preferred pivoting strategy in z-direction for the inner loops, namely, to keep as many z-coordinates for the pivots as possible from the first inner loops to all the succeeding loops. But that might not always be the best choice. So, what happens if we violate the condition in the previous result ? It turns out that the following weaker version is still true:*
For $i = 1, \ldots, k$ let $J_i \subset \{1, \ldots, k'\}$. If $z_j^{(i)} = \hat{z}_j^{(i)}$ for each particular $i = 1, \ldots, k$ and all $j \in J$, then

$$R_i(x, y, z_j^{(i)}) = 0 \quad \text{for } i = 1, \ldots, k \text{ and } j \in J_i,$$

whenever

$$x = x_\mu \in \{x_1, \ldots, x_i\} \text{ with } z_j^{(i)} \in J_\mu$$

or

$$y = y_\nu \in \{y_1, \ldots, y_i\} \text{ with } z_j^{(i)} \in J_\nu.$$

Propostion 3.5 and Remark 3.6 show that it is at least partially possible to preserve the interpolation property of CA2D. What about the rank property ? As indicated in Remark 3.4 it is determined by the structure of the approximant constructed in our method. This structure is given by the following Theorem.

Theorem 3.7 *Let the function $f(x, y, z)$ be given on $[0, 1]^3$. Then the above procedure generated by the recursion* (33) *produces after k steps in the outer and k' steps in each inner loop the approximation*

$$S_k(x, y, z) = \sum_{i=1}^{k} \sum_{\mu_2=1}^{k'} \sum_{\mu_1=1}^{k'} \frac{u_{\mu_1}^{(i)}(x)}{u_{\mu_1}^{(i)}(x_{\mu_1}^{(i)})} \cdot \frac{v_{\mu_2}^{(i)}(y)}{v_{\mu_2}^{(i)}(y_{\mu_2}^{(i)})} \cdot \frac{w_{\mu_1}^{(i)}(z)\hat{w}_{\mu_2}^{(i)}(z)}{\sum_{\nu=1}^{k'} \frac{u_\nu^{(i)}(x_i)}{u_\nu^{(i)}(x_\nu^{(i)})} w_\nu^{(i)}(z)}, \tag{35}$$

where

$$u_{\mu_1}^{(i)}(x) = f(x, y_i, z_{\mu_1}^{(i)}) - \sum_{l=1}^{i-1} \frac{\sum_{m=1}^{k'} \frac{u_m^{(l)}(x)}{u_m^{(l)}(x_m^{(l)})} w_m^{(l)}(z_{\mu_1}^{(i)})}{\sum_{\nu=1}^{k'} \frac{u_\nu^{(l)}(x_l)}{u_\nu^{(l)}(x_\nu^{(l)})} w_\nu^{(l)}(z_{\mu_1}^{(i)})} \cdot \sum_{\mu=1}^{k'} \frac{v_\mu^{(l)}(y_i)}{v_\mu^{(l)}(y_\mu^{(l)})} \hat{w}_\mu^{(l)}(z_{\mu_1}^{(i)}) - \sum_{j=1}^{\mu_1 - 1} \frac{u_j^{(i)}(x)}{u_j^{(i)}(x_j^{(i)})} w_j^{(i)}(z_{\mu_1}^{(i)}), \tag{36}$$

$$v_{\mu_2}^{(i)}(y) = f(x_i, y, \hat{z}_{\mu_2}^{(i)}) - \sum_{l=1}^{i-1} \frac{\sum_{m=1}^{k'} \frac{u_m^{(l)}(x_i)}{u_m^{(l)}(x_m^{(l)})} w_m^{(l)}(\hat{z}_{\mu_2}^{(i)})}{\sum_{\nu=1}^{k'} \frac{u_\nu^{(l)}(x_l)}{u_\nu^{(l)}(x_\nu^{(l)})} w_\nu^{(l)}(\hat{z}_{\mu_2}^{(i)})} \cdot \sum_{\mu=1}^{k'} \frac{v_\mu^{(l)}(y)}{v_\mu^{(l)}(y_\mu^{(l)})} \hat{w}_\mu^{(l)}(\hat{z}_{\mu_2}^{(i)}) - \sum_{j=1}^{\mu_2-1} \frac{v_j^{(i)}(y)}{v_j^{(i)}(y_j^{(i)})} \hat{w}_j^{(i)}(\hat{z}_{\mu_2}^{(i)}), \quad (37)$$

$$w_{\mu_1}^{(i)}(z) = f(x_{\mu_1}^{(i)}, y_i, z) - \sum_{l=1}^{i-1} \frac{\sum_{m=1}^{k'} \frac{u_m^{(l)}(x_{\mu_1}^{(i)})}{u_m^{(l)}(x_m^{(l)})} w_m^{(l)}(z)}{\sum_{\nu=1}^{k'} \frac{u_\nu^{(l)}(x_l)}{u_\nu^{(l)}(x_\nu^{(l)})} w_\nu^{(l)}(z)} \cdot \sum_{\mu=1}^{k'} \frac{v_\mu^{(l)}(y_i)}{v_\mu^{(l)}(y_\mu^{(l)})} \hat{w}_\mu^{(l)}(z) - \sum_{j=1}^{\mu_1-1} \frac{u_j^{(i)}(x_{\mu_1}^{(i)})}{u_j^{(i)}(x_j^{(i)})} w_j^{(i)}(z) \quad (38)$$

and

$$\hat{w}_{\mu_2}^{(i)}(z) = f(x_i, y_{\mu_2}^{(i)}, z) - \sum_{l=1}^{i-1} \frac{\sum_{m=1}^{k'} \frac{u_m^{(l)}(x_i)}{u_m^{(l)}(x_m^{(l)})} w_m^{(l)}(z)}{\sum_{\nu=1}^{k'} \frac{u_\nu^{(l)}(x_l)}{u_\nu^{(l)}(x_\nu^{(l)})} w_\nu^{(l)}(z)} \cdot \sum_{\mu=1}^{k'} \frac{v_\mu^{(l)}(y_{\mu_2}^{(l)})}{v_\mu^{(l)}(y_\mu^{(l)})} \hat{w}_\mu^{(l)}(z) - \sum_{j=1}^{\mu_2-1} \frac{v_j^{(i)}(y_{\mu_2}^{(i)})}{v_j^{(i)}(y_j^{(i)})} \hat{w}_j^{(i)}(z) \quad (39)$$

for $\mu_1, \mu_2 = 1, \ldots, k'$ *and the inner pivots* $x_{\mu_1}^{(i)}, y_{\mu_2}^{(i)}, z_{\mu_1}^{(i)}, \hat{z}_{\mu_2}^{(i)}$.

Remark 3.8 *From formula* (35) *we see that the total rank of the approximation is* $k(k')^2$*, provided that* k' *is constant for each inner loop. In our numerical examples we will see, that it is possible to decrease the number of inner iterations for higher outer loops.*

Proof The proof follows the same idea as for the explicit formulas at the end of section 2. Formula (33) has the structure of (25) where we now substitute $U_i(x,z) = R_{i-1}(x, y_i, z)$ and $V_i(y,z) = R_{i-1}(x_i, y, z)$ in each outer loop $i = 1, \ldots, k$ by their approximations via CA2D in an inner loop with k' steps. First we concentrate on the inner loop to decompose U_i. For the corresponding recursion we introduce the following notation $r_0^{(i)}(x,z) = U_i(x,z)$ and

$$r_{\mu_1}^{(i)}(x,z) = r_{\mu_1-1}^{(i)}(x,z) - \frac{r_{\mu_1-1}^{(i)}(x, z_{\mu_1}^{(i)})}{r_{\mu_1-1}^{(i)}(x_{\mu_1}^{(i)}, z_{\mu_1}^{(i)})} r_{\mu_1-1}^{(i)}(x_{\mu_1}^{(i)}, z) \quad \text{for } \mu_1 = 1, \ldots, k'. \quad (40)$$

Setting $u_{\mu_1}^{(i)}(x) = r_{\mu_1-1}^{(i)}(x, z_{\mu_1}^{(i)})$ and $w_{\mu_1}^{(i)}(z) = r_{\mu_1-1}^{(i)}(x_{\mu_1}^{(i)}, z)$ we get for the inner approximation of U_i

$$s_{\mu_1}^{(i)}(x,z) = U_i(x,z) - r_{\mu_1}^{(i)}(x,z) = \sum_{m=1}^{\mu_1} \frac{u_m^{(i)}(x)}{u_m^{(i)}(x_m^{(i)})} w_m^{(i)}(z) \quad (41)$$

with

$$u_m^{(i)}(x) = U_i(x, z_m^{(i)}) - \sum_{j=1}^{m-1} \frac{u_j^{(i)}(x)}{u_j^{(i)}(x_j^{(i)})} w_j^{(i)}(z_m^{(i)}) \quad (42)$$

and

$$w_m^{(i)}(z) = U_i(x_m^{(i)}, z) - \sum_{j=1}^{m-1} \frac{u_j^{(i)}(x_m^{(i)})}{u_j^{(i)}(x_j^{(i)})} w_j^{(i)}(z). \quad (43)$$

Now we do the same calculations for the decomposition of the function V_i and end up with

$$\hat{s}_{\mu_2}^{(i)}(y,z) = V_i(y,z) - \hat{r}_{\mu_2}^{(i)}(y,z) = \sum_{\mu=1}^{\mu_2} \frac{v_\mu^{(i)}(y)}{v_\mu^{(i)}(y_\mu^{(i)})} \hat{w}_\mu^{(i)}(z) \quad (44)$$

for $\mu_2 = 1, \ldots, k'$, where $\hat{r}^{(i)}_{\mu_2}(y,z)$, $v^{(i)}_{\mu}(y)$ and $\hat{w}^{(i)}_{\mu}(z)$ are defined analog to (40), furthermore,

$$v^{(i)}_{\mu}(y) = V_i(y, \hat{z}^{(i)}_{\mu}) - \sum_{j=1}^{\mu-1} \frac{v^{(i)}_j(y)}{v^{(i)}_j(y^{(i)}_j)} \hat{w}^{(i)}_j(\hat{z}^{(i)}_{\mu}) \tag{45}$$

and

$$\hat{w}^{(i)}_{\mu}(z) = V_i(y^{(i)}_{\mu}, z) - \sum_{j=1}^{\mu-1} \frac{v^{(i)}_j(y^{(i)}_{\mu})}{v^{(i)}_j(y^{(i)}_j)} \hat{w}^{(i)}_j(z). \tag{46}$$

Now we just have to put everything together. Starting with formula (26) and inserting for U_i and V_i their approximations $s^{(i)}_{\mu_1}$ with $\mu_1 = k'$ from (41) and $\hat{s}^{(i)}_{\mu_2}$ with $\mu_2 = k'$ from (44) respectively, we arrive at (35). To get formulas (36) and (38) we start with (42) and (43) with $m = \mu_1$, insert for U_i its representation (27) with $i = k$ and substitute there the functions U_l and V_l from the former steps by its approximations $s^{(i)}_{\mu_1}$ with $i = l$ and $\mu_1 = k'$ from (41) and $\hat{s}^{(i)}_{\mu_2}$ with $i = l$ and $\mu_2 = k'$ from (44) respectively. Analogously we get (37) and (39), by taking (45) and (46), inserting (28) and finally substitute U_l and V_l as before. That finishes the proof. □

Remark 3.9 *Looking at formula* (35) *shows us, that in order to prove a rank property, as indicated in Remark 3.4, the condition would be very restrictive. So, in contrast to the interpolation property we don't claim now that something worth to be called rank property holds here.*

The complete procedure is described in the following Algorithm.

Algorithm 1 (The Generalized Cross Approximation)

1: Choose the pivot strategy for the outer loop (i.e x_i, y_i for $i = 1, .., k$)
2: do $nk = 1, ..., k$ (outer loop)
3: for $nk = 1$
3a: Consider U_1, V_1 and obtain $u^{(1)}_j, w^{(1)}_j, v^{(1)}_j$and $\hat{w}^{(1)}_j$ for $j = 1, \ldots, k'$ by applying CA2D.
4: for $nk \neq 1$
5: do $mk = 1, ..., k'$ (inner loop)
5a: for $mk = 1$
Choose pivot points in the inner loop ($x^{(nk)}_1, y^{(nk)}_1, z^{(nk)}_1$ and $\hat{z}^{(nk)}_1$).
Obtain the vectors $u^{(nk)}_1, w^{(nk)}_1, v^{(nk)}_1$and $\hat{w}^{(nk)}_1$.
5b: for $mk \neq 1$
Choose pivot points in the inner loop ($x^{(nk)}_j, y^{(nk)}_j, z^{(nk)}_j$ and $\hat{z}^{(nk)}_j$ for $j = 2, .., k'$).
Obtain the vectors $u^{(nk)}_j, w^{(nk)}_j, v^{(nk)}_j$and $\hat{w}^{(nk)}_j$ for $j = 2, .., k'$.
6: end do (inner loop).
7: end do (outer loop).

The pivot strategy for the outer loop and inner loops in the Step 1, Step 5a and 5b is described later in this section. Since we know the pivot points x_1 and y_1 and the pivot points in the first inner loop, the vectors $u^{(1)}_j, w^{(1)}_j, v^{(1)}_j$and $\hat{w}^{(1)}_j$ in the step 3a of the Algorithm 1 can be obtained easily by using CA2D with linear complexity in n (look at (10) and (11)). The

vectors $u_i^{(nk)}, w_i^{(nk)}, v_i^{(nk)}$and $\hat{w}_i^{(nk)}$ in step 5a and 5b can be easily obtained from the equations (34),(35),(36) and (37). Since these equations involve U_{nk}, V_{nk} (U_{nk}, V_{nk} to be evaluated at some fixed points), we need the information from all the previous outer loops. These can be implemented efficiently and inexpensively in the algorithm during the computation. The computational complexity of the algorithm is discussed below.

Error analysis

In [2] Bebendorf developed a systematic machinery to analyze the error behavior in the framework of incremental approximations gradually, even for nested algorithms. The fundamental relation for all these methods is the recursion $r_0[f] = f$ and

$$r_k[f] = r_{k-1}[f] - \frac{r_{k-1}[f](x_k)}{l_k(x_k)} l_k$$

with functions l_k at our disposal such that $l_k(x_k) \neq 0$. The remainder $r_k[f]$ can be connected to the error

$$\mathcal{E}_k[f] = f - \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_k) \end{bmatrix}^T \Psi_k$$

of a linear approximation in an arbitrary system $\Psi_k = [\psi_1, \ldots, \psi_k]^T$ of continuous functions.
Fortunately, our method fits into this framework and we can exploit his results, where we partly adapt the notation now.
For fixed $y, z \in [0,1]^2$ we set $r_i[f_{yz}] = R_i(\cdot, y, z)$ (from formula (25)), $l_i = r_{i-1}[f_{yz}]$ for $i = 1, \ldots, k$. Furthermore, we define

$$M_k = \begin{pmatrix} l_1(x_1) & \cdots & \cdots & l_1(x_k) \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & l_k(x_k) \end{pmatrix} \quad \text{and} \quad \xi^{(k)} = M_k^{-1} \begin{bmatrix} l_1 \\ \vdots \\ l_k \end{bmatrix}.$$

The terms entering the estimates are finally set as

$$\sigma_k = \sup_{x \in [0,1]} \sum_{i=1}^{k} |\xi_i^{(k)}(x)| \quad \text{and} \quad \tilde{\sigma}_{i,j} = \sup_{x \in [0,1]} \sum_{\nu=i}^{j} |\tilde{\xi}_\nu^{(j)}(x)| \qquad (\tilde{\sigma}_{1,0} = 0)$$

for $i, j = 1, \ldots, k$, where the $\tilde{\xi}$'s and corresponding $\tilde{M}$'s are given by substituting l with its approximation $\tilde{l}$ in the lines above, for more explanation see [2].
After all these notational adaptions we can state:

Theorem 3.10 *Let $\varepsilon > 0$ be sufficiently small and we assume*

$$\sup_{(x,z) \in [0,1]^2} |R_{i-1}(x, y_i, z) - \mathcal{A}(R_{i-1})(x, y_i, z)| \leq \varepsilon \tag{47}$$

$$\sup_{(y,z) \in [0,1]^2} |R_{i-1}(x_i, y, z) - \mathcal{A}(R_{i-1})(x_i, y, z)| \leq \varepsilon \tag{48}$$

for $i = 1, \ldots, k$. Then

$$|R_k(x, y, z)| \leq (1 + \sigma_k) \max_{\tau \in \{x, x_1, \ldots, x_k\}} \left\| \mathcal{E}_k[f_\tau] \right\|_{\infty, [0,1]^2} + c_k \varepsilon \tag{49}$$

for $(x, y, z) \in [0, 1]^3$. *Here*

$$c_k = \tilde{\sigma}_{1,k} + 2\sum_{j=1}^{k}(\tilde{\sigma}_{1,j-1}+1)\prod_{i=j}^{k}(c_{piv}^{(i)}+1)(\tilde{\sigma}_{i,k}+1), \quad with \;\; c_{piv}^{(i)} = \sup_{y,z}\frac{|\mathcal{A}(R_{i-1})(x_i, y, z)|}{c_{\min}^{(i)}}.$$

The proof follows the same lines as for Theorems 2 and 3 in [2] with further adaptions according to the notation there.

Pivot strategy

The accuracy and efficiency of the Cross Approximation rely on the pivot strategy. There are different ways of choosing the pivot points and we did not make to much effort to find something optimal. As explained from the beginning the choice of pivot points are such that condition (34) holds, where we follow the same substituting strategy as explained before Table 3.2. While in section 3.1 the algorithm produced completely random pivots during the performance for the outer loop, here we updated the error on the diagonal after each step k of the outer loop and found the position of the maximum entry in modulus for the coordinates x_{k+1}, y_{k+1}. In the inner loops we basically follow the partial pivoting strategy, compare [1] or [12]. For the matrix U_1 we choose $x_1^{(1)}, \ldots, x_{k'}^{(1)}$ randomly and find the positions of the maximum error entry after each iteration in the corresponding z-depending vectors. According to Proposition 3.5 we keep those $z_1^{(1)}, \ldots, z_{k'}^{(1)}$ as inner pivot coordinates for all the succeeding matrices to approximate. In the other direction (either x or y) again we find the position of the maximum error entry in the corresponding vectors as next pivot coordinate. These error vectors have to be updated at each iteration step, which can be implemented in the algorithm with a linear complexity in n.

Computational complexity

As explained earlier, the approximation has been constructed in two steps. k' inner loops are involved in each of the k outer loops. The first outer loop requires $O(n(k')^2)$ operations (Step 3a: Only two adaptive Cross Approximations in two dimensions are involved). Further, k' inner loops require $O(n(l-1)(k')^2)$ arithmetic operations in each outer loop $l = 2, 3, ..k$ (Step 5a and 5b). Therefore $O(nk^2(k')^2)$ arithmetic operations are required to construct the complete approximation. So the complexity is linear in n.

4 Numerical results

To show the performance of the proposed algorithm, here we present a few numerical examples, where we always used a AMD Opteron 8220 with 2800 MHz and 256 GB RAM where the code was written in Fortran 90.

In contrast to our theoretical investigations in the previous section, now we allow the number k' of inner steps to vary from one outer iteration to another and denote it by $k'_1, \ldots k'_k$. The stopping criterion for the inner loop now reads as

$$\left\| \frac{u_{k'_i}^{(i)}}{u_{k'_i}^{(i)}(x_{k'_i}^{(i)})} \right\|_\infty \cdot \|w_{k'_i}^{(i)}\|_\infty < \varepsilon^{(i)},$$

where the inner precision $\varepsilon^{(i)}$ is up to our choice. Mainly because of symmetry of our test functions for $i = 1, \ldots, k$ we always run k_i' inner iterations to decompose both U_i and V_i.
Let us consider $n = 501$ uniformly divided points on the interval $[0, 1]$ in each direction. A tensor is generated by the function on the uniform grid (tensor product of the n grid points in each direction on the interval $[0, 1]$). Our pivot strategy is described in the previous section. The approximation is obtained and the error of the approximation is calculated in the maximum norm. As explained earlier, we rely our algorithm on the pointwise (or Hadamard) product and therefore we are sometimes forced to use a substituting strategy. In those cases we replaced the entries which are less than or equal to 5.0e-16 by 9.0e-16.
The tables below show (from left to right) the iteration step k, the corresponding error bound for the stopping criterion of the inner loop $\varepsilon^{(k)}$, the resulting numbers of inner iterations $k_1', \ldots k_k'$, the relative approximation error in the maximum norm and the time (measured in seconds) needed to compute the tensor decomposition plus the time taken for partial pivoting.

Example 1: Consider the smooth three dimensional function

$$f(x, y, z) = \frac{1}{1 + x + y + z} \quad \text{on } [0, 1]^3$$

from section 3.1. As expected the result is very satisfactory and one can observe a significant decrease in the number of inner iterations, which results in a smaller rank of the final decomposition compared with $k(k')^2$.

k	$\varepsilon^{(k)}$	$k_1', \ldots, k_k'$	error	time
1	9e-02	2	8.333e-02	0
2	5e-03	3,2	2.32129e-03	0.009
3	5e-04	4,3,2	4.03437e-05	0.01
4	1e-05	5,4,3,2	7.942e-07	0.029
5	1e-06	6,5,4,3,1	1.6383e-07	0.05
6	1e-07	6,5,5,4,2,2	3.4534e-09	0.069
7	1e-09	7,7,6,5,4,3,1	4.2992e-11	0.119
8	5e-12	9,8,8,6,6,5,3,2	1.76025e-13	0.25

Table 4.3: Approximation error of f for different k

Example 2: Now we consider

$$h(x, y, z) = e^{-xyz} \quad \text{on } [0, 1]^3$$

as it was called in section 3.1. There we had to use the substituting strategy after the first iteration in each step to get a good approximation. The following table shows that it is also working very well here. The decreasing inner ranks are also visible.

k	$\varepsilon^{(k)}$	$k'_1, \ldots, k'_k$	error	time
1	0.1	3	0.284	0
2	5e-02	3,2	6.778e-02	0.009
3	5e-03	4,3,2	5.55114e-04	0.019
4	5e-04	4,4,3,1	6.8267e-05	0.038
5	1e-05	5,4,4,3,1	3.2136e-06	0.049
6	1e-07	6,5,5,4,4,2	4.445e-08	0.088
7	1e-09	7,6,6,5,4,4,1	3.297e-10	0.149
8	2.5e-11	8,7,7,7,5,6,4,1	5.133e-11	0.248

Table 4.4: Approximation error of h for different k

Example 3: Now we consider the function

$$g(x, y, z) = \exp(-10\sqrt{(x - 1/2)^2 + (y - 1/2)^2 + (z - 1/2)^2}) \quad \text{on } [0, 1]^3,$$

using the same notation as in section 3.1. Here we see that for better approximations one needs large ranks because of the bad smoothness behavior in the neighborhood of the center of the domain, which also results in a more irregular behavior of the inner ranks.

k	$\varepsilon^{(k)}$	$k'_1, \ldots, k'_k$	error	time
1	0.1	3	0.12685	0
2	5e-02	3,2	6.016e-02	0.009
3	5e-03	6,4,2	2.931249e-02	0.019
4	3e-03	6,4,4,6	1.2158e-02	0.04
5	4e-05	12,11,7,9,11	6.312e-03	0.25
6	5e-06	12,11,7,9,11,4	3.2983e-03	0.32

Table 4.5: Approximation error of g for different k

Example 4: Now we consider the function

$$\psi(x, y, z) = \frac{1}{0.01 + x + y + z} \quad \text{on } [0, 1]^3,$$

which shows how the method works close to singularities. Here again one needs larger ranks for better precision and the decrease of the inner ranks is very slow.

k	$\varepsilon^{(k)}$	$k'_1, \ldots, k'_k$	error	time
1	0.9	4	9.01662e-02	0
2	0.4	5,3	2.533e-02	0.009
3	0.3	6,3,3	8.4734e-03	0.019
4	4e-03	9,7,7,7	4.3482e-03	0.089
5	3e-03	9,7,7,7,7	1.091e-03	0.139
6	5e-04	10,7,7,7,7,7	6.222e-04	0.26
7	3e-05	12,11,10,9,10,9,9	3.455e-04	0.76

Table 4.6: Approximation error of ψ for different k

Conclusions

As mentioned in the very beginning, the original idea was born in four dimensions, where one can interpret a 4d-tensor as matrix of matrices and apply CA2D to this matrix. The resulting approximation would be a sum of products of 3d-tensors and one would have preserved the corresponding interpolation and rank properties again. But since linear complexity with respect to the system size n has highest priority, now even two more levels of nested Cross Approximations would be neccessary. A simpler way was proposed by Bebendorf in [2] (which is the four-dimensional analog to the method discussed in section 2), where he even presented an error analysis. The explicit formulas for the building blocks in his method are also possible to derive by our procedure used in sections 2 and 3.

The quality of our numerical results is of the same kind as those presented in [2] and [10]. But we believe that our more symmetric format together with the obtained theoretical results give another perspective for the extension to higher dimensional problems and applications, see [7].

As described in section 3, to ensure that the pointwise product with $1/R_{k-1}(x_k, y_k, z)$ for all $z \in [0, 1]$ is well defined, we had to impose a substituting strategy, which proved to work fine in practice. To find a theoretical justification for it is still an open problem.

Acknowledgements. The authors thank Prof. W. Hackbusch (MPI for Mathematics in the Siences, Leipzig) and Prof. M. Bebendorf (University of Bonn) for valuable discussions.

References

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

[2] Bebendorf, M.: Adaptive Cross Approximation of Multivariate Functions. to appear in Constr. Approx. 2010.

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

[4] Espig, M.: Effiziente Bestapproximation mittels Summen von Elementartensoren in hohen Dimension. Ph.D thesis, University of Leipzig, Germany 2007.

[5] Goreinov, S.A., Tyrtyshnikov E.E. and Zamarashkin, N.L.: A theory of pseudoskeleton approximations. Lin. Alg. Appl. 261, 1-21 (1997).

[6] Hackbusch, W. and Kühn, S.: A new scheme for the tensor representation. Preprint 2/2009, MPI MIS Leipzig, Germany, 2009.

[7] Khoromskij, B.N.: Tensors-structured Numerical Methods in Scientific Computing: Survey on Recent Advances. Preprint 21/2010, MPI MIS Leipzig, Germany, 2010.

[8] Khoromskij, B.N. and Khoromskaia, V.: Multigrid tensor approximation of function related tensors. SIAM J. Sci. Computing, 31(4), 3002-3026, 2009.

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

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

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

[12] Schneider, J.: Error estimates for two-dimensional Cross Approximation. to appear in Jour. Approx. Theo. 2010.