Institute of Numerical Mathematics

Compact matrix form of the d-dimensional tensor decomposition [$]

I.V. Oseledets [*]

[*] *Institute of Numerical Mathematics RAS, 119333 Moscow, Gubkina 8,*
ivan.oseledets@gmail.com,http://spring.inm.ras.ru/osel

Preprint 09-01

Abstract

A simple nonrecursive form of the tensor decomposition in d dimensions is presented. It does not suffer from the curse of dimensionality, it has asymptotically the same number of parameters as the canonical decomposition, but it is stable and its computation is based on low-rank approximation of auxiliary *unfolding matrices*. The new form gives a clear and convenient way to implement all basic operations efficiently. A fast recompression procedure is presented, as well as basic linear algebra operations. Examples showing the benefits of the decomposition are given and the efficiency is demonstrated by the computation of the smallest eigenvalue of a 19-dimensional operator.

1. Introduction

Tensors are natural multidimensional generalizations of matrices and they have attracted a tremendous interest in the last years. Multilinear algebra, tensor analysis and the theory of tensor approximations play increasingly important role in the computational mathematics and numerical analysis [1, 2, 3, 4, 5]. An efficient representation of a tensor (by tensor we mean only an array with d indices) with small number of parameters may give us an opportunity and ability to work with d-dimensional problems with d being as high as 10, 100 or even 1000. The problems of such sizes can not be handled by standard numerical methods due to the *curse of dimensionality*, since everything (memory, amount of operations) grows exponentially in d. There is an effective way to represent a large class of important d-dimensional tensors by using a canonical decomposition of a given tensor $\mathbf{A}$ [6, 7]:

$$\mathbf{A} = A(i_1, i_2, \ldots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) U_2(i_2, \alpha) \ldots U_d(i_d, \alpha), \tag{1}$$

where r is called *tensor rank* and $n_i \times r$ matrices $U_k = [U_k(i_k, \alpha)]$ are called *canonical factors*. For large d tensor $\mathbf{A}$ is never formed explicitly but represented in some low-parametric format. Canonical decomposition (1) is a good candidate for such format. However, it suffers from several drawbacks. First, it can be unstable, i.e. for some tensors $\mathbf{A}$ the approximation with a fixed r does not exist, and the numerical algorithms for computing an approximate representation often fail. Also, even the most successful existing algorithms [8, 9, 10] for computing best low-tensor rank approximation are not guaranteed to work well. It is often the case that they encounter local minima and are stuck there. That is why it is a good idea to look at alternatives for the canonical format, which may have a larger number of parameters but are much more well suited for the numerical treatment. Preliminary attempts to present such a format where independently made in [11] and [12] using very different approaches. Both of these approaches rely on a hierarchical tree structure and reduce the storage of the d-dimensional arrays to the storage of the auxiliary three-dimensional ones. The number of parameters in principle can be larger than for the canonical format, but they are based entirely on the SVD. In [11] an algorithm which computes the tree-like decomposition of a d-dimensional array by a recursive splitting was presented and convincing numerical experiments we are given. The process goes from the top of the tree to the bottom of it. In [12] the construction is entirely different, since it goes from bottom to top, and from our point of view is less suited for the numerical implementation, the authors only present a concept but do not present any numerical experiments.

However, the tree decomposition depends on the splitting of spatial indices and leads to the "nonsymmetric decomposition", where different array indices are treated unequally. By carefully looking at the parameters defining the decomposition we found that in fact it leads to a new extremely simple, but powerful matrix form of the decomposition. This is a new tensor decomposition that we are going to present here.

For a given tensor $\mathbf{A}$ we approximate it by an object of form

$$\mathbf{A} = A(i_1, i_2, \ldots, i_d) = G_1^{(i_1)} G_2^{(i_2)} \ldots G_d^{(i_d)}, \tag{2}$$

where $G_1^{(i_1)}$ is a $1 \times r_1$ row vector, matrices $G_k^{(i_k)}$ are $r_{k-1} \times r_k$ for $k = 2, \ldots, d-1$ and $G_d^{(i_d)}$ is $l_{d-1} \times 1$ column vector. The indices $i_1, \ldots, i_d$ now enumerate small $r_{k-1} \times r_k$ matrices or vectors for border values of k.

This form is very compact and simple. The defining parameters, r_k are easily computable by rank detection. To illustrate that we will use a 5-dimensional array. The case $d = 5$ is reduced to

$$A(i_1, i_2, i_3, i_4, i_5) = \sum_{\alpha_1 \alpha_2 \alpha_3 \alpha_4} G_1(i_1, \alpha_1) G_2(i_2, \alpha_1, \alpha_2) G_3(i_3, \alpha_2, \alpha_3) G_4(i_4, \alpha_3, \alpha_4) G_5(i_5, \alpha_4). \tag{3}$$

Note an important difference between (3), (2) and (1): the indices $i_1, i_2 \ldots i_d$ are connected only pairwise through auxiliary indices, which appear only in two places, while in the canonical format index α is everywhere. Since the auxiliary indices appear only twice, the corresponding dimensions are bounded from above only by the rank of the corresponding *unfolding matrix* of a tensor (we will prove it rigorously shortly after). For example, to estimate the dimension in α_2 one should take an unfolding

$$A_2 = A(i_1, i_2; i_3, i_4, i_5),$$

estimate its rank

$$r_2 = \operatorname{rank} A_2,$$

and the dimension in α_2 is not higher than r_2. In the general case, the sizes of of the auxiliary dimensions r_k are just the ranks of the matrices

$$A_k = A(i_1, i_2, \ldots . . i_k; i_{k+1}, \ldots, i_d).$$

From a Lemma in [11] we now know that all these ranks are bounded from above by r, in fact they can be sufficiently smaller, since they are bounded by so-called effective or border rank [13]. The effective rank can be much smaller even for very simple examples, like the discretization of the d-dimensional Laplacian [11, 10].

The ranks r_k can be estimated by SVD or any suitable rank-revealing method. The new decomposition (we will leave the name TT*) and it gives a straightforward way to perform basic operations with multidimensional arrays. The most important one is the recompression procedure, which allows us to reduce the values of ranks when some representation of form (2) is given. This procedure is described in Section 3 and is based on a sequence of "rolling" QR decompositions and the SVD decompositions of small auxiliary matrices. The addition of tensors is also easy, as well as the scalar product and the Frobenius norm of a tensor. These are described in Sections 4.2. We can trivially

*Initially, TT stood for "Tree Tucker" recursive decomposition of [11], now there is nor tree neither Tucker in it. However, since the format is still uniquely defined by Three-dimensional Tensors, we leave the same name for it

convert the canonical representation to the TT format and then compress, examples borrowed from [11] show that the number of parameters can be indeed much smaller than in the canonical representation. In Section 4.3 the matrix-by-vector procedure where both a matrix and a vector are given in the TT format is described and numerical examples are presented in Section 5.

2. Notations and a more detailed description of the decomposition

Let us give a more formal definition of the new decomposition and specify the parameters that define it. A d-dimensional $n_1 \times n_2 \ldots \times n_k$ tensor $\mathbf{A}$ is represented as two matrices of sizes $n_1 \times r_1$ and $n_d \times r_{d-1}$ denoted by $G_1(i_1, \alpha_1)$ and $G_d(i_d, \alpha_{d-1})$ and $(d-2)$ three-dimensional tensors of sizes $n_k \times r_{k-1} \times r_k$ denoted by $G_k(i_k, \alpha_{k-1}, \alpha_k)$. The TT-decomposition is defined by formula

$$\mathbf{A} = A(i_1, \ldots, i_d) = \sum_{\alpha_1, \ldots \alpha_{d-1}} G_1(i_1, \alpha_1) G_2(i_2, \alpha_1, \alpha_2) G_3(i_3, \alpha_2, \alpha_3) \ldots G_{d-1}(i_{d-1}, \alpha_{d-2}, \alpha_{d-1}), \tag{4}$$

where α_k goes from 1 to r_k. There is a useful equivalent matrix form of (4):

$$A(i_1, \ldots, i_d) = G_1^{(i_1)} G_2^{(i_2)} \ldots G_d^{(i_d)},$$

where $G_1^{(i_1)}, G_d^{(i_d)}$ are row and column vectors respectively, and $G_k^{(i_k)}$ are $r_{k-1} \times r_k$ matrices.

The parameters defining the TT decomposition are the three dimensional arrays

$$G_k(i_k, \alpha_{k-1}, \alpha_k).$$

We will call them *core tensors* of the TT decomposition analogously to the Tucker core of the Tucker decomposition. The ranks r_k will be called *compression ranks*. It is easy to get a bound on them. Each r_k appears only twice in the equation (4), it is bounded from above by them rank of the following *unfolding matrix* of $\mathbf{A}$:

$$A_k = A_k(i_1, \ldots, i_k; i_{k+1} \ldots i_d) = A(i_1, \ldots, i_d), \tag{5}$$

i.e. with the first k indices enumerate the rows of A_k and the last $d-k$ the columns of A_k.

The following theorem is true.

Theorem 1. *If for each unfolding matrix A_k of form (5) of d-dimensional tensor $\mathbf{A}$*

$$\operatorname{rank} A_k = r_k, \tag{6}$$

then there exists a decomposition (4) with compression ranks not higher than r_k.

Proof. Consider index i_1 and the unfolding matrix A_1. Its rank is equal to r_1, therefore it admits dyadic (skeleton) decomposition

$$A_1 = UV^\top.$$

or in the index form

$$A_1(i_1; i_2, \ldots, i_d) = \sum_{\alpha_1=1}^{n_1} U(i_1, \alpha_1) V(\alpha_1, i_2, \ldots, i_d).$$

We can treat V again as a d-dimensional tensor with the first index replaced by α_1. However, we will treat it as a $d-1$ dimensional tensor

$$V(\alpha_1 i_2, i_3, i_4, \ldots, i_d),$$

i.e. with α_1 and i_2 considered as a one "long" index. For this tensor again take unfolding matrices $V_2, \ldots V_d$. It holds that

$$\operatorname{rank} V_k \leqslant r_k.$$

To prove that express V as

$$V = A_1^\top U^\top (U^\top U)^{-1} = A_1^\top W,$$

or in the index form

$$V(\alpha_1 i_2, \ldots, i_d) = \sum_{i_1=1}^{n_1} A(i_1, \ldots, i_d) W(i_1, \alpha).$$

For the k-th mode the compression rank is equal to r_k, therefore $\mathbf{A}$ can be represented as

$$A(i_1, \ldots, i_d) = \sum_{\beta=1}^{r_k} F(i_1, \ldots, i_k, \beta) G(\beta, i_{k+1}, \ldots, i_d).$$

Using that we obtain

$$\begin{aligned} V_k = V(\alpha_1 i_2, \ldots, i_k; i_{k+1}, \ldots, i_d) &= \sum_{i_1=1}^{n_1} \sum_{\beta=1}^{r_k} W(i_1, \alpha_1) F(i_1, \ldots, i_k, \beta) G(\beta, i_{k+1}, \ldots, i_d) = \\ &= \sum_{\beta=1}^{r_k} H(\alpha_1, i_2, \ldots, i_k, \beta) G(\beta, i_{k+1}, \ldots, i_d), \end{aligned}$$

where

$$H(\alpha_1 i_2, \ldots, i_k, \beta) = \sum_{i_1=1}^{n_1} F(i_1, \ldots, i_k, \beta) W(i_1, \alpha_1).$$

The row and column indices are now separated and

$$\operatorname{rank} V_k \leqslant r_k.$$

Therefore we can continue the process with V, separating index $(i_2 \alpha_1)$ from the others. This yields a core tensor $G_2(i_2, \alpha_1, \alpha_2)$ and so on, finally giving the TT representation. □

Our proof is constructive and shows how the representation can be computed. In is reduced to d successive singular value decompositions of unfolding matrices.

Corollary 1. *If* $\mathbf{A}$ *admits canonical rank-*r *representation, then the number of parameters in* (4) *is not higher than*

$$(d-2)nr^2 + 2nr,$$

where $n = \max(n_1, n_2, \ldots, n_d)$ *is the maximum mode size of* $\mathbf{A}$.

Proof. To prove that it is sufficient to see that [11]

$$\operatorname{rank} A_k \leqslant r,$$

and count the number of elements in the core tensors G_k. $\square$

Remark. The number of parameters in the tree format of [11] as well as for the Φ-system in [12] is estimated as.

$$\mathcal{O}(dnr + (d-2)r^3).$$

It is easy to modify TT decomposition to reduce $(d-2)nr^2 + 2nr$ to $dnr + (d-2)r^3$ by using auxiliary Tucker decomposition of the core tensors G_k. G_k for $2 \leqslant k \leqslant (d-1)$ are

$$n_k \times r_{k-1} \times r_k$$

tensors and it is easy to prove that the mode-1 rank of it is not higher than t_k, where t_k is the Tucker (mode rank) of $\mathbf{A}$ along the k-th mode. Therefore each G_k will be replaced by a $n_k \times t_k$ factor matrix and a $t_k \times r_{k-1} \times r_k$ auxiliary three-dimensional array. However, for the simplicity of the presentation we omit this step from our decomposition, but the places will be pointed out where it can be used to reduce computational complexity.

Throughout the paper we use a tensor-by-matrix multiplication referred to as the mode-k contraction or mode-k multiplication. Given an array (tensor) $\mathbf{A} = [a_{i_1 \ldots i_d}]$ and a matrix $U = [u_{i_k \alpha}]$, we define the mode-k multiplication result as a new tensor $\mathbf{B} = [b_{i_1 \ldots \alpha \ldots i_d}]$ (α is on the k-th place) obtained by the convolution over the k-th axis:

$$b_{i_1 \ldots \alpha \ldots i_d} = \sum_{\beta_k = 1}^{n} a_{i_1 \ldots i_k \ldots i_d} u_{i_k \alpha}.$$

We denote this operation as follows:

$$\mathbf{B} = \mathbf{A} \times_k U.$$

3. Recompression from TT to TT.

Given a tensor in a TT format,

$$\mathbf{A} = G_1^{(i_1)} G_2^{(i_2)} G_3^{(i_3)} \ldots G_d^{(i_d)},$$

we want to estimate the true value of ranks $r'_k \leqslant r_k$ while maintaining the prescribed accuracy ε. These ranks are equal to the ε-ranks of the unfoldings of the tensors. First,

try to compute r_1' and reduce this rank. The corresponding unfolding matrix A_1 can be written as a product

$$A_1 = UV^\top, \tag{7}$$

where

$$U(i_1, \alpha_1) = G_1(i_1, \alpha_1),$$

and

$$V(i_2, i_3, \ldots, i_d; \alpha_{d-1}) = \sum_{\alpha_2, \ldots, \alpha_{d-1}} G_2(i_2, \alpha_1, \alpha_2) \ldots G_d(i_d, \alpha_{d-1}), \tag{8}$$

where the rows of U are indexed by i_1, whereas the rows of V are indexed by a multiindex $(i_2, \ldots, i_d)$.

A standard way to compute the singular value decomposition (SVD) of A_1 using the representation of form (7) is the following. First, compute QR-decompositions of U and V:

$$U = Q_u R_u, \quad V = Q_v R_v,$$

assemble a small $r \times r$ matrix

$$P = R_u R_v^\top,$$

compute its reduced SVD:

$$P = XDY^\top,$$

where D is a $\widehat{r} \times \widehat{r}$ diagonal matrix and X and Y are $r \times \widehat{r}$ matrices with orthonormal columns. $\widehat{r}$ is the ε-rank of D (which is equal to the ε-rank of B). Finally,

$$\widehat{U} = Q_u X, \quad \widehat{V} = Q_v Y,$$

are the matrices of dominant singular vectors of the full matrix B. In our case, columns of U and V are very long and correspond to multidimensional arrays, so it is prohibitive to compute full QR-decompositions. However, these decompositions can be computed in a structured way very efficiently. The U matrix for the A_1 is small, so we can compute QR-decomposition of it directly. The V matrix, however, is very large and something else has to be done. Let us describe the procedure to compute the QR-decomposition of V and then prove that it indeed provides the desired decomposition.

For the core G_d we can compute its QR-decomposition:

$$G_d = Q_d R_d,$$

where $R_d = R_d(\alpha_{d-1}, \alpha_{d-1}')$ is the R-factor of the decomposition. The index α_{d-1} appears only twice in the expression for V, therefore we can convolve G_d and G_{d-1} over α_{d-1}, and obtain another decomposition of form (8):

$$V(i_2, \ldots, i_d, \alpha_1) = \sum_{\alpha_2, \ldots, \alpha_{d-1}'} G_2(i_2, \alpha_1, \alpha_2) \ldots G_{d-1}'(i_{d-1}, \alpha_{d-2}, \alpha_{d-1}') G_d'(i_d, \alpha_{d-1}'),$$

where G_d' is equal to Q_d, the Q-factor of the QR-decomposition of G_d. The core G_{d-1}' is computed as

$$G_{d-1}' = G_{d-1} \times_3 R_d'.$$

We can continue this process of orthogonalization. Reshape the core

$$G'_{d-1} = G'_{d-1}(i_{d-1}, \alpha_{d-2}, \alpha'_{d-1})$$

into a $(n_{d-1}r_{d-1}) \times r_{d-2}$ matrix T_{d-1} with elements

$$T_{d-1}(i_{d-1}\alpha'_{d-1}; \alpha_{d-2}) = G'_{d-1}(i_{d-1}, \alpha_{d-2}, \alpha'_{d-1}).$$

It amounts to the permutation of the dimensions and reshaping a tensor into a matrix. Then orthogonalize it columns. The Q-factor will be again a $(n_{d-1}r_{d-2}) \times r_{d-1}$ matrix, and it can be reshaped into a three-dimensional tensor and permuted back to a three-dimensional array $G''_{d-1}(i_d, \alpha'_{d-2}, \alpha'_{d-1})$. Also a $r_{d-2} \times r_{d-2}$ R-factor is computed. Denote it by R_{d-1}. Now continue this process: convolve G_{d-2} with R_{d-1} over the third index, obtain a three-dimensional tensor, union the first and the third index into one long index, compute the QR-decomposition of the obtained matrix, put Q-factor instead of the G_{d-2} and continue with mode $d-3$ and so on. Each step of the algorithm is an equivalent transformation of the TT representation, being some orthogonalization procedure. It will stop at mode 2, where we will have the decomposition

$$V(i_2, \ldots, i_d; \alpha_1) = \sum_{\alpha'_1} R_2(\alpha_1, \alpha'_1) Z(i_2, \ldots, i_d, \alpha'_1),$$

and the tensor Z is expressed as

$$Z(i_2, \ldots, i_d; \alpha'_1) = \sum_{\alpha'_2, \ldots, \alpha'_{d-1}} Q_2(i_2, \alpha'_1, \alpha'_2) \ldots Q_d(i_d, \alpha'_{d-1}), \tag{9}$$

where the cores Q_k have the following *orthogonality property*:

$$\sum_{i_k \alpha'_k} Q_k(i_k, \alpha'_{k-1}, \alpha'_k) Q_k(i_k, \widehat{\alpha}'_{k-1}, \alpha'_k) = \delta(\alpha'_{k-1}, \widehat{\alpha}'_{k-1}), \tag{10}$$

for $k = 2, \ldots, d-1$ and

$$\sum_{i_d} Q_d(i_d, \alpha'_{d-1}) Q_d(i_d, \widehat{\alpha}'_{d-1}) = \delta(\alpha'_{d-1}, \widehat{\alpha}'_{d-1}). \tag{11}$$

This property follows directly from the construction. Then the following Lemma is true.

Lemma 1. *If Z is specified by (9) and the cores Q_k satisfy orthogonality properties (10),(11), then Z, treated as a "long matrix" has orthonormal columns:*

$$\sum_{i_2, \ldots, i_d} Z(i_2, \ldots, i_d; \alpha'_1) Z(i_2, \ldots, i_d; \widehat{\alpha}'_1) = \delta(\alpha_1, \widehat{\alpha}_1). \tag{12}$$

Proof. The summation over i_d in (12) touches only two cores, yielding a delta function:

$$\sum_{i_k} Q_d(i_d, \alpha'_{d-1}) Q_d(i_d, \widehat{\alpha}'_{d-1}) = \delta(\alpha'_{d-1}, \widehat{\alpha'}_{d-1}).$$

Therefore the summation over $\widehat{\alpha}'_{d-1}$ vanishes giving us the following summation over i_{d-1}:

$$\sum_{i_{d-1}\,\alpha'_{d-1}} Q_{d-1}(i_{d-1}, \alpha'_{d-2}, \alpha'_{d-1}) Q_{d-1}(i_{d-1}, \widehat{\alpha}'_{d-2}, \alpha'_{d-1}),$$

where $\widehat{\alpha'}_{d-1}$ was replaced by α'_{d-1} due to the Kronecker delta symbol, coming from the d-th core. This sum is equal to $\delta(\alpha'_{d-2}, \widehat{\alpha}'_{d-2})$ due to the orthogonality condition (10). Going from $d-1$ to $d-2$ and so on to 2, we find that Z indeed has orthonormal columns. $\square$

We have presented a way to compute QR-decomposition of V using only the cores G_k of the TT-decomposition of $\mathbf{A}$, and the Q factor was computed in a structured form. To perform the compression, we compute the compressed SVD and convolve two cores containing α_1 with two small matrices. After the first mode was compressed, we can do the same thing for each mode, since for arbitrary k we can use the same algorithm to compute structured QR-decompositions of the U and V factors (the algorithm for U is the same with slight modifications), matrices R_u and R_v, singular values, the reduced rank and the matrices X and Y which perform the dimensionality reduction. However, we can avoid making these decompositions every time for every mode from scratch by using information obtained from the previous mode. For example, after we reduced the rank for A_1, we modify cores G_1 and G_2, cores $G_3, \ldots, G_d$ stay the same and they satisfy orthogonality conditions (10),(11). Therefore, to compress in the second mode, we just have to orthogonalize G_1 and G_2. This can be realized by storing the R-matrix that appears during the orthogonalization algorithm. In fact we do the following: for A_1 we compute the reduced decomposition of form

$$A_1 = U_1 V_1^\top,$$

where U_1 is orthogonal, and compress V_1, and so on. In that way the error of the approximation can not increase by more than ε in each step, therefore if the singular values are cut at ε, the total accumulated error can not exceed $\mathcal{O}(d\varepsilon)$.

The formal description of the algorithm is given in Algorithm 1.

Let us estimate the number of operations required by the algorithm. For simplicity, assume that $r_k \sim r$. The right-to-left sweep requires successive QR-decompositions of $nr \times r$ matrices which cost $\mathcal{O}(nr^3)$ operations each, in total $\mathcal{O}(dnr^3)$ operations. The compression step requires SVD of $(nr) \times r$ matrices, which need $\mathcal{O}(nr^3)$ for each mode The final estimate is

$$\mathcal{O}(dnr^3)$$

operations for the full compression procedure. By using additionally Tucker factors and applying the TT decomposition only to the core of it, we can reduce the complexity to

$$\mathcal{O}(dnr^2 + dr^4),$$

where the first term is just the price of d sequential QR decompositions of $n \times r$ matrices of Tucker factors.

Algorithm 1: TT recompression

Input: $\mathbf{A}$ in the TT format specified by core tensors $G_k(i_k, \alpha_{k-1}, \alpha_k)$, accuracy ε

Output: $\mathbf{A}'$ in the TT format specified by core tensors $G'_k(i_k, \alpha'_{k-1}, \alpha'_k)$, with smallest possibles compression ranks r_k such that

$$\|A - A'\| \leqslant \varepsilon \|A\|.$$

1: {Right-to-left sweep}
2: R_v is the R-factor of $G(i_d, \alpha_{d-1})$, G_d is replaced by Q factor of it.
3: **for** $k = d - 1$ to 2 **do**
4: Convolve $G_k(i_k, \alpha_{k-1}, \alpha_k)$ over α_k with the matrix R_v:

$$T(i_k, \alpha_{k-1}, \alpha'_k) = G_k \times_3 R_v.$$

5: Set R_v to the R-factor of an $(n_k r_k) \times r_{k-1}$ matrix T with elements

$$T(i_k \alpha'_k; \alpha_{k-1}) = G_k(i_k, \alpha_{k-1}, \alpha'_k),$$

and Q_k is the Q-factor of T.
6: Reshape Q_k into an $n_k \times r_k \times r_{k-1}$ tensor, and set G_k to be a permutation of it:

$$G_k(i_k, \alpha_{k-1}, \alpha_k) = Q_k(i_k, \alpha_k, \alpha_{k-1}).$$

7: **end for**
8: {Mode-1 compression and the initialization of R_u}
9: $G_1 := G_1 R_v$.
10: {Compute truncated SVD of G_1}
11: $G_1 = U \Lambda V^\top$.
12: Set $G_1 = U$, $R_u = V\Lambda$.
13: {Compression of other modes}
14: **for** $k = 2$ to $d - 1$ **do**
15: Convolve with R_u: $G_k = G_k \times_2 R_u$.
16: Reshape G_k into an $(n_k r_{k-1}) \times r_k$ matrix T_k compute its reduced SVD

$$T_k = U_k \Lambda_k V_k^\top,$$

and set G_k to be an $n_k \times r_{k-1} \times \widehat{r}_k$ tensor which is a reshaping of U_k.
17: Set $R_u := V_k \Lambda_k$.
18: **end for**
19: $G_d := G_d R_u$.

3.1. From canonical to TT

The conversion from canonical decomposition to the TT format is trivial. For tree structure of [11], it presented some difficulties, requiring a recursive algorithm based on the computation of Gram matrices. Here we just have to rewrite the canonical format of form

$$\mathbf{A} = \sum_{\alpha} U_1(i_1, \alpha) \dots U_d(i_d, \alpha)$$

in the TT format by using Kronecker delta symbols:

$$\mathbf{A} = \sum_{\alpha_1 \alpha_2 \dots \alpha_{d-1}} U_1(i_1, \alpha_1)\delta(\alpha_1, \alpha_2)U_2(i_2, \alpha_2)\delta(\alpha_2, \alpha_3)\dots\delta(\alpha_{d-2}, \alpha_{d-1})U_d(i_d, \alpha_{d-1}).$$

After that we can compress it by using the recompressing procedure described above. For example, for the discretization of the d-dimensional Laplace operator of form

$$\triangle_d = \triangle \times I \times \dots \times I + \dots + I \times \dots \times \triangle, \tag{13}$$

all compression ranks are equal to 2, since $\triangle$ can be approximated by a matrix of tensor rank 2 with arbitrary precision. We implemented the algorithm in Matlab and the timings are presented below. The structure of the tensor does not depend on the actual matrices $\triangle$ and I, so we can consider the "Laplace-like" tensor

$$A = a \times b \times \dots \times b + \dots + b \times \dots \times a, \tag{14}$$

where a, b are vectors of length n. The computational timings depend only on n and d. Moreover, to compute the decomposition for a general n, we can take some vectors a and b of length 2 and compress the $2 \times 2 \dots \times 2$ core tensor, and use it for the decomposition of an arbitrary Laplace-like operator. The Laplace operator is often encountered in the applications so it may be useful to derive a special algorithm for it. In Table 1 we give timings for our straightforward implementation applied to the Laplace-like operator of form (14). This is a prototype implementation but the timings still look very nice. For large n we could not take d larger that 32 because of memory restrictions during the naive canonical to TT conversion step (which requires storing full $n \times d \times d$ matrices) which can be of course improved by a special variant of the compression procedure applied directly to the tensor in the canonical format. This will be reported in a separate paper.

The compression time to compress a 32-dimensional operator is less than a second and taking into account that for Laplace-like operators we need to compress only a $2 \times 2 \times \dots \times 2$ core tensor this dimension can be as high as 128. The only restriction at this point is a memory restriction (not for the TT format but for storing intermediate arrays) and can be passed by using a machine with a larger amount of memory. As in [11] all ranks involved are equal to 2, and the tensor is represented by a set of $(d-2)$ array of sizes $n \times 2 \times 2$ and two $n \times 2$ matrices. If we use Tucker factors additionally, the number of parameters will be reduced to

$$\mathcal{O}(2dn + 8(d-2))$$

$n = 2$		$n = 1024$	
$d = 4$	$8.0 \cdot 10^{-4}$	$d = 4$	$2.3 \cdot 10^{-3}$
$d = 8$	$1.6 \cdot 10^{-3}$	$d = 8$	$2.2 \cdot 10^{-2}$
$d = 16$	$3.8 \cdot 10^{-3}$	$d = 16$	$2.4 \cdot 10^{-1}$
$d = 32$	$1.0 \cdot 10^{-2}$	$d = 32$	$2.3 \cdot 10^{0}$
$d = 64$	$5.1 \cdot 10^{-2}$	—	—
$d = 128$	$4.4 \cdot 10^{-1}$	—	—

Table 1. Compression timings (in seconds) for d-dimensional Laplace-like tensor, by "—" we denote cases where we did not have enough memory for canonical to TT compression.

for a d-dimensional Laplace-like operator.

Another interesting example is the discretization of the second-order differential operator of form

$$\mathcal{L}P = \sum_{i<j} \sigma_{ij} \frac{\partial^2 P}{\partial x_i \partial x_j}. \tag{15}$$

The natural discretization of this operator has canonical rank $d(d-1)/2$. However the experiments in [11] showed that the splitting rank of the tree format [11] behaved like $d/2 + 2$. Here we see the same pattern. For the compression ranks we have even more interesting pattern that surely exhibits a "thin structure" of the unfolding matrices. The natural discretization of (15) gives us a matrix of form

$$A = \sum_{i<j} \sigma_{ij} W_i W_j, \tag{16}$$

where W_i is acting only in the i-th mode:

$$W_i = I \times \ldots \times \underbrace{W_i}_{i} \times \ldots \times I.$$

For $\sigma_{ij} = 1$ this a electron-type potential considered in [10] and it was proved there that this matrix can be approximated by a matrix of tensor rank 3 with arbitrary precision. The estimates for the case of general σ_{ij} are unknown by now. But we can test it by our compressing subroutine and the results are quite interesting. It appears that the ranks do not depend on the actual "one-dimensional" matrices W_i and even the identity matrix I can be replaced by an arbitrary matrix. We will call the matrices of form (16) *Scholes-like* matrices since that the appear in the Black-Scholes equation for multi-asset option pricing. For example, for $d = 19$ the ranks are given in Table 2. The σ_{ij} were taken at random, and we did not observe any dependence on them (there are special cases where the rank is smaller, but for the general case these ranks should be the same.) The canonical rank was 171, and the compression is indeed very good. It is conjectured that the highest rank (11 here) is $\mathcal{O}(d/2)$. This conjecture is supported by numerical examples. The Tucker ranks are equal to 2 (only two matrices in each

Mode	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Rank	2	4	5	6	7	8	9	10	11	11	10	9	8	7	6	5	4	2	2

Table 2. Compression ranks for different modes

mode), therefore an estimate for the storage of the Scholes-like operator is

$$\mathcal{O}(dn) + \mathcal{O}(d^2)$$

instead of the

$$\mathcal{O}(d^3 n)$$

parameters for the canonical format.

This interesting behavior of compression ranks requires rigorous explanation and such an explanation will be reported in the forthcoming paper.

4. Basic operations

4.1. Additions and multiplication by a number

Arithmetic operations in the TT format can be readily implemented. The addition of two vectors,

$$\mathbf{A} = A_1^{(i_1)} \dots A_2^{(i_d)},$$
$$\mathbf{B} = B_2^{(i_2)} \dots B_2^{(i_d)}.$$

is reduced to the merge of cores, and for each mode the sizes of auxiliary dimensions are summed. If for an index k we have two cores, $A_k(i_k, \alpha_{k-1}, \alpha_k), B(i_k, \alpha_{k-1}, \alpha_k)$, of sizes $n_k \times r_1 \times r_2$ and $n_k \times r_3 \times r_4$ respectively, then the result C_k will have size $n_k \times (r_1 + r_3) \times (r_2 + r_4)$ and its elements, in Matlab notation, are given by

$$C_k(1 : n_k, 1 : r_1, 1 : r_2) = A_k, \quad C_k(1 : n_k, (r_1 + 1) : (r_1 + r_3), (r_2 + 1) : (r_2 + r_4)) = B_k,$$

and all other elements of C_k are equal to zero. The multiplication by a number α is trivial, we just scale one of the cores by it. The addition of two tensors is a good test for the recompression procedure. If we sum a vector t in TT format with itself, the ranks are doubled, but the result should be compressed to $2t$ with the same ranks as for t. In our experiments, such recompression was performed with accuracy which is of order of the machine precision. The addition of two vectors requires virtually no operations. However, the recompression that is almost always needed afterwards requires $\mathcal{O}(nr^3 d)$ operations. If we used an auxiliary Tucker decomposition of the tensor and kept only the core of the decomposition in the TT format, then we can reduce the computational complexity of the recompression step to

$$\mathcal{O}(nr^2 d + dr^4).$$

4.2. Multidimensional convolution, scalar products and norms

The operation which is very nice in the TT format is the multidimensional convolution, i.e. evaluation of the expression of form

$$W = \sum_{i_1 \ldots i_d} \mathbf{A}(i_1, \ldots, i_d) u_1(i_1) \ldots u_d(i_d),$$

and u_k are vectors of length n_k. This is a discrete analogue to the high-dimensional integration. If $\mathbf{A}$ is in the TT format, then

$$\mathbf{A} = G_1^{(i_1)} \ldots G_d^{(i_d)},$$

and the convolution is done in two steps. At the first step, each core $G_k(i_k, \alpha_{k-1}, \alpha_k)$ is convolved over i_k with u_k yielding a matrix:

$$\Gamma_k(\alpha_{k-1}, \alpha_k) = \sum_{i_k=1}^{n_k} G_k(i_k, \alpha_{k-1}, \alpha_k), u_k(i_k).$$

Such convolution can be implemented as a single matrix-by-vector product in $\mathcal{O}(nr^2)$ operations. For border cases, $k = 1$ and $k = d$ we have not a matrix but a vector (denote them by v_1 and v_d). Finally, we have to evaluate the product

$$W = v_1^\top \Gamma_2 \ldots \Gamma_k v_d,$$

which is just a sequence of matrix-by-vector products. In the end, the 3 product is computed. The complexity of this step is $\mathcal{O}(dr^2)$ The total number of arithmetic operations required is $\mathcal{O}(dnr^2 + dr^2)$. Again Tucker format can be used to reduce the number of operations if $r < n$. The implementation is rather straightforward and requires

$$\mathcal{O}(dnr + dr^3)$$

operations for a single convolution.

Another important operation encountered in many iterational methods is the scalar (dot) product of two tensors. For two tensors $\mathbf{A}, \mathbf{B}$ it is just a scalar product of their unfoldings:

$$\langle A, B \rangle = \sum_{i_1, \ldots, i_d} A(i_1, \ldots, i_d) B(i_1, \ldots, i_d).$$

How does this operation look like in the TT format, i.e. in terms of cores? Everything goes just like in the case of the multidimensional convolution but with minor modifications. For two cores $A_k(i_k, \alpha_{k-1}, \alpha_k)$ and $B_k(i_k, \alpha'_{k-1}, \alpha'_k)$ we construct a four-dimensional array by convolving over i_k:

$$C_k(\alpha_{k-1}, \alpha'_{k-1}, \alpha_k, \alpha'_k) = \sum_{i_k=1}^{n_k} A_k(i_k, \alpha_{k-1}, \alpha_k) B_k(i_k, \alpha'_{k-1}, \alpha'_k).$$

(For $k = 1$ and $k = d$ the number of indices involved is smaller, but everything is similar). This is equivalent to the matrix multiplication and costs $\mathcal{O}(nr^4)$ arithmetic

operations (if ranks are approximately the same) for each pair of cores, in total $\mathcal{O}(dnr^4)$ operations. For border cases, modes 1 and d, we have not four-dimensional but two-dimensional arrays. After that, we have to compute literally the same expression as for the case of multidimensional convolution, but where each index α_k is replaced by a pair of indices $\alpha_k\alpha'_k$. From algorithmic point of view there is no difference between them and the same program can be used. At all, the second step ("combing") costs $\mathcal{O}(dr^4)$ operations and is reduced to $(d-2)$ matrix-by-vector products. Using dot product, Frobenius norm

$$\|\mathbf{A}\| = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle},$$

and a distance between two tensors:

$$\|\mathbf{A} - \mathbf{B}\|^2 = \|\mathbf{A}\|^2 - 2\langle \mathbf{A}, \mathbf{B} \rangle + \|\mathbf{B}\|^2,$$

can be computed. The number of operations required is

$$\mathcal{O}(dnr^4 + dr^4),$$

operations for the direct implementation and

$$\mathcal{O}(dnr^2 + dr^5),$$

operations if the Tucker format is used additionally.

Algorithm 2: Multidimensional convolution

Input: Tensor $\mathbf{A}$ in the TT format with cores A_k, and vectors $u_1, \ldots, u_d$.
Output: $W = \mathbf{A} \times_1 u_1^\top \ldots \times_d u_d^\top$.

1: {Mode contractions}
2: $v_1(\alpha_1) = \sum_{i_1=1}^{n_1} A_1(i_1, \alpha_1) u_1(i_1)$,
3: $v_d(\alpha_{d-1}) = \sum_{i_d=1}^{n_d} A_d(i_d, \alpha_{d-1}) u_d(i_d)$,
4: **for** $k = 2$ to $d-1$ **do**
5: $\quad \Gamma_k(\alpha_{k-1}, \alpha_k) = \sum_{i_k=1}^{n_k} A_k(i_k, \alpha_{k-1}, \alpha_k) u_k(i_k)$.
6: **end for**
7: $v := v_1$.
8: **for** $k = 2$ to $d-1$ **do**
9: $\quad v := v\Gamma_k$.
10: **end for**
11: $W = v^\top v_d$.

4.3. Matrix-by-vector product

The most important operation in linear algebra is probably a matrix-by-vector product. When both matrix and vector are given in the TT format then the natural question is how to compute their product. The only difference between a TT format for a matrix is that the core tensors have two "spatial" indices instead of one: $G_k = G_k(i_k j_k, \alpha_{k-1}, \alpha_k)$.

Algorithm 3: Dot product

Input: Tensor $\mathbf{A}$ in the TT format with cores A_k, and tensor B in the TT format with cores B_k

Output: $W = \langle \mathbf{A}, \mathbf{B} \rangle$.

1: {Mode contractions}
2: $v_1(\alpha_1, \alpha'_1) = \sum_{i_1=1}^{n_1} A_1(i_1, \alpha_1) B_1(i_1, \alpha'_1)$,
3: $v_d(\alpha_{d-1}, \alpha'_{d-1}) = \sum_{i_d=1}^{n_d} A_d(i_d, \alpha_{d-1}) B_d(i_d, \alpha'_{d-1})$.
4: **for** $k = 2$ to $d - 1$ **do**
5: $\quad \Gamma_k(\alpha_{k-1}\alpha'_{k-1}, \alpha_k\alpha'_k) = \sum_{i_k=1}^{n_k} A_k(i_k, \alpha_{k-1}, \alpha_k) B_k(i_k, \alpha'_{k-1}, \alpha'_k)$.
6: **end for**
7: $v := v_1$.
8: **for** $k = 2$ to $d - 1$ **do**
9: $\quad v := v\Gamma_k$.
10: **end for**
11: $W = v^\top v_d$.

The matrix-by-vector product is the convolution over j_k and it can be performed solely in terms of core tensors. Each pair of core tensors corresponding to the same index is treated independently. If the core for the matrix is $M_k(i_k j_k, \alpha_{k-1}, \alpha_k)$ and the core for the vector is $v_k(j_k, \beta_k, \beta_{k+1})$ with sizes $(n_k n_k) \times m_{k-1} \times m_k$ and $n_k \times r_{k-1} \times r_k$ respectively, then the resulting core tensor Y_k has size $n_k \times (m_{k-1} r_{k-1}) \times (m_k r_k)$ and its elements are equal to

$$Y_k(i_k, \alpha_{k-1}\beta_{k-1}, \alpha_k\beta_k) = \sum_{j_k=1}^{n_k} M_k(i_k j_k, \alpha_{k-1}, \alpha_k) v_k(j_k, \beta_{k-1}, \beta_k).$$

Such convolution can be implemented as a product of a $(m_{k-1} m_k n_k) \times n_k$ matrix and a matrix of size $n_k \times (r_{k-1} r_k)$. If we assume that all compression ranks for the matrix are approximately the same and are equal to m, the spatial dimensions are of size n, and the compression ranks for the vector are all equal to r, that operation will cost us $\mathcal{O}(n^2 m^2 r^2)$ operations. And it should be done for all core tensors, therefore the total computation needs $\mathcal{O}(d n^2 m^2 r^2)$ operations, which is linear in the dimension d. After that, the compression has to be performed and it will cost $\mathcal{O}(d m^4 r^4)$ operations. If also $m \approx r$ then the final complexity estimate is

$$\mathcal{O}(d n^2 r^4 + d r^8)$$

which is quite well (we have no exponential dependence from d) but still there is a room for improvement in the part of dependence from n and r, as it was done for three-dimensional tensors [14]. This can be done since all the work performed is with three-dimensional objects and the same techniques can be used.

The $\mathcal{O}(d n^2 r^4)$ term can be reduced to at most $\mathcal{O}(d n^2 r^2)$ by using an auxiliary Tucker decomposition of both matrix and vector tensors. That means that we preliminary compute the Tucker factors and store only the core with small mode sizes (but still of

order r^d when stored as a full array). The Tucker cores are stored in the TT format. Then the multiplication is first performed with factors, in each mode we have to multiply t_k matrices by r_k vectors, having dt_kr_k matrix-by-vector products (Here we can also try to use the additional structure of the Tucker factors, i.e. sparse or Toeplitz to perform multiplication faster than $\mathcal{O}(n^2)$), yielding d Tucker factors of size $n_k \times (t_k r_k)$ for the products. Then we have to compute QR-decomposition of each factor, obtain d matrices and convolve them with the Tucker core of the product each along its specified mode. This step is essentially the same as for the naive algorithm for matrix-by-vector product except for the fact that n_k is now replaced by $t_k r_k$, i.e. the Tucker rank of the product. With this approach, the complexity is

$$\mathcal{O}(dn^2r^2 + dr^8),$$

which is much less, especially for large r. Also the second term can be made better, in the fashion of [14], but this is beyond the scope of the current paper.

Algorithm 4: Matrix-by-vector product

Input: Matrix M in the TT format with cores M_k, vector v in the TT format with cores v_k.

Output: Vector $y = Mv$ in the TT format with cores Y_k.

1: {Compute convolutions}

2: **for** $k = 2$ to $d - 1$ **do**

3:

$$Y_k(i_k, \alpha_{k-1}\beta_{k-1}, \alpha_k\beta_k) = \sum_{j_k=1}^{n_k} M_k(i_kj_k, \alpha_{k-1}, \alpha_k)v_k(j_k, \beta_{k-1}, \beta_k),$$

by using array reshaping, index permutation and matrix multiplication.

4: **end for**

5: {Cases $k = 1$ and $k = d$}

6: If $k = 1$, compute

$$Y_1(i_1, \alpha_1, \beta_1) = \sum_{j_1=1}^{n_1} M_1(i_1j_1, \alpha_1)v_1(j_1, \beta_1).$$

7: If $k = d$, compute

$$Y_d(i_d, \alpha_{d-1}, \beta_d) = \sum_{j_d=1}^{n_d} M_d(i_dj_d, \alpha_{d-1})v_d(j_d, \beta_d).$$

---

5. A numerical example

As an example consider the computation of the smallest eigenvalue in magnitude of the following operator

$$H = \triangle_d + c_v \sum_i \cos(x - x_i) + c_w \sum_{i<j} \cos(x_i - x_j),$$

the one considered in [10],[9]. We have chosen simplest possible discretization (3-point discretization of one-dimensional Laplacian with zero boundary conditions on $[0, 1]$). This is just an illustration of how the method works. As a starting guess for the initial vector we have chosen the eigenvector corresponding to the smallest eigenvalue in magnitude of the d-dimensional Laplace operator $\triangle_d$. This eigenvector has tensor rank-1 and is easily computable. After that we applied a simple power iteration to the shifted matrix

$$\widehat{H} = cI - H,$$

where the shift c was chosen to make the smallest eigenvalue of H the largest eigenvalue in magnitude of the shifted matrix. We have taken $d = 19$ and one-dimensional grid size $n = 8, 16, 32, 64$, therefore the maximal mode size for the matrix has been $n^2 = 4096$. The matrix was compressed by the canonical-to-TT compression algorithm, and then the power iteration

$$v := Hv; \quad v = \frac{v}{\|v\|},$$

was performed. This is surely not the best method for the computation of the smallest eigenpair, it was used just to test the recompression procedure and the matrix-by-vector subroutine. The parameters c_v, c_w were set to $c_v = 100$, $c_w = 5$. The computed smallest eigenvalues for different n are given in Table 3. By "time for one iteration" we mean the total time required for the matrix-by-vector product and for the recompression with the parameter $\varepsilon = 10^{-6}$. δ is the estimated error of the eigenvalue of the operator (i.e., model error), where for the exact eigenvalue we take the eigenvalue computed for the largest n (here it is $n = 64$). We can see that the eigenvalue stabilizes. To detect the convergence of the iterative process for the discrete problem we used the scaled residual, $\|Ax - \lambda x\|/|\lambda|$ and stopped when it was smaller than 10^{-5}. The number of iterations for the power method was of order $1000 - 2000$, we do not present it here. The compressing ranks for the vector where not higher than 4 in all cases. Note that for

n	8	16	32	64
λ_{min}	$2.41 \cdot 10^3$	$2.51 \cdot 10^3$	$2.56 \cdot 10^3$	$2.58 \cdot 10^3$
δ	$6 \cdot 10^{-2}$	$2.7 \cdot 10^{-2}$	$7 \cdot 10^{-3}$	—
Average time for one iteration (sec)	$3 \cdot 10^{-2}$	$3.6 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$6.2 \cdot 10^{-2}$

Table 3. Results for the computation of the minimal eigenvalue

these small values of n the timings for one iteration grow very mildly when increasing n, it is interesting to explain the nature of this behavior.

6. Conclusion and future work

We have presented a new tensor decomposition. In some sense, our main decomposition (4) is just another form of writing the tree format of [11] or the subspace approach of [12]: the same three-dimensional tensors as defining parameters, the same complexity estimates. However, the almost-symmetric form of the TT-decomposition gives a big advantage over the approaches mentioned above. It gives a nice and clear way for the stable and fast recompression procedures. It is based entirely on a sequence of QR and SVD decompositions of matrices, and does not require any recursion. Its implementation required only about 150 lines of Matlab code [@] compared to the several thousands code lines in C and Fortran for the recursive TT format. It also allows fast and intuitive implementation of the basic linear algebra operations: matrix-by-vector multiplication, addition, dot product and norm. We showed how to apply these subroutines to compute the smallest eigenvalue of a high-dimensional operator. This is a rather simplified example, but everything works fine.

There is a great room for improvement and further development. The dependence from d is now linear and the curse of dimensionality no longer threatens us. However, the dependence from n and r still poses some limit on the problems we can tackle if we want to solve problems in $d = 1000$ or, for example, invert matrices in 100-dimensional space. We should incorporate Tucker factors into existing code, reducing the complexity of almost all important operations by an order of magnitude. This requires only a slight modification of the code. The techniques applicable for three-dimensional problems [14] can be applied for the d-dimensional case.

The approximation procedure is based on standard QR and SVD decompositions, and is robust. It can be applied to any suitable d-dimensional array to detect any "hidden" structure in it. We already discovered a new structure in the Scholes-like d-dimensional operator and the theoretical treatment will be reported in the forthcoming paper. Another idea is to apply TT-format for data analysis and compression. Our preliminary experiments show that this high-dimensional decomposition can be applied to the image compression problem giving a new compression algorithm.

And the last but not the least goal we want to mention are the applications. There are several important multidimensional equations: Schroedinger equation in quantum chemistry, Black-Scholes equation in multi-asset modelling, Fokker-Planck equation, stochastic partial differential equations. The new decomposition gives a straightforward way for the approximate solution of such equations for the dimensions that were impossible to other methods.

References

[1] *L. De Lathauwer, B. De Moor, J. Vandewalle*. A multlinear singular value decomposition // *SIAM J. Matrix Anal. Appl.* 2000. Vol. 21. Pp. 1253–1278.

[@] Matlab codes are available by request or at http://spring.inm.ras.ru/osel

[2] *P. Comon*. Tensor decomposition: State of the Art and Applications // IMA Conf. Math. in Signal Proc., Warwick, UK. 2000. "— December.

[3] *L. De Lathauwer, B. De Moor, J. Vandewalle*. On best rank-1 and rank-$(R_1, R_2, ..., R_N)$ approximation of high-order tensors // *SIAM J. Matrix Anal. Appl.* 2000. Vol. 21. Pp. 1324–1342.

[4] *Richard Bro*. PARAFAC: Tutorial and applications // *Chemometrics and Intelligent Laboratory Systems*. 1997. Vol. 38, no. 2. Pp. 149–171.

[5] *L. Grasedyck*. Existence and computation of low kronecker-rank approximations for large systems in tensor product structure // *Computing*. 2004. Vol. 72. Pp. 247–265.

[6] *R.A. Harshman*. Foundations of the Parafac procedure: models and conditions for an explanatory multimodal factor analysis // *UCLA Working Papers in Phonetics*. 1970. Vol. 16. Pp. 1–84.

[7] *J.D. Caroll, J.J Chang*. Analysis of individual differences in multidimensional scaling via n-way generalization of Eckart-Young decomposition // *Psychometrica*. 1970. Vol. 35. Pp. 283–319.

[8] *M. Espig*. Effiziente Bestapproximation mittels Summen von Elementartensoren in hohen Dimensionen: Ph.D. thesis. 2007.

[9] *G. Beylkin, M. J. Mohlenkamp*. Algorithms for numerical analysis in high dimensions. // *SIAM J. Sci. Comput*. 2005. Vol. 26, no. 6. Pp. 2133–2159.

[10] *G. Beylkin, M. J. Mohlenkamp*. Numerical operator calculus in higher dimensions. // *Proc. Natl. Acad. Sci. USA*. 2002. Vol. 99, no. 16. Pp. 10246–10251.

[11] *I.V. Oseledets, E.E. Tyrtyshnikov*. Breaking the curse of dimensionality, or how to use SVD in many dimensions: Research Report 09-03. Kowloon Tong, Hong Kong: ICM HKBU, 2009. www.math.hkbu.edu.hk/ICM/pdf/09-03.pdf.

[12] *W. Hackbusch, S. Kühn*. A new scheme for the tensor representation: Preprint 2. Leipzig: MPI MIS, 2009. www.mis.mpg.de/preprints/2009/preprint2009_2.pdf.

[13] *I.V. Oseledets, E. E. Tyrtyshnikov*. Recursive decomposition of multidimensional tensors // *Doklady Math.* 2009.

[14] *I.V. Oseledets, D.V. Savostyanov, E.E. Tyrtyshnikov*. Linear algebra for tensor problems // *Computing*. 2009. www.math.hkbu.edu.hk/ICM/pdf/08-03.pdf.