Institute of Numerical Mathematics
Russian Academy of Sciences
119333, Moscow, Gubkina 8
www.inm.ras.ru

P U B
I N M
R A S
pub.inm.ras.ru

I.V. Oseledets

Tensors inside of matrices give logarithmic complexity

Preprint 2009-04

April 2, 2009

Moscow 2009

Tensors inside of matrices give logarithmic complexity

I.V. Oseledets

Institute of Numerical Mathematics, Russian Academy of Sciences,
Russia, 119333 Moscow, Gubkina 8,
ivan.oseledets@gmail.com

April 2, 2009

[$]This work was supported by RFBR grants 08-01-00115, 09-01-00565 and RFBR/DFG grant 09-01-91332

Abstract

A new method for structured representation of matrices and vectors is presented. The method is based on the representation of a matrix as a d-dimensional tensor and applying the TT-decomposition proposed recently. It turned out that for many important cases the number of parameters to represent an $n \times n$ matrix falls down to $\mathcal{O}(\log^{\alpha} n)$, giving a logarithmic storage. It is shown that this format can be used not only for storage reduction, but also for linear algebra operations. Possible applications include differential and integral equations, data and image compression.

1. Introduction

Consider an $n \times n$ matrix. How many parameters are needed to represent it? One can imagine that a matrix is defined by $\mathcal{O}(n)$ parameters: Toeplitz matrices, sparse matrices, low-rank matrices and matrices with rank structure [1]. It is possible to have matrices that are described by $\mathcal{O}(\sqrt{n})$ parameters, if the tensor structure is used. For example, many important Block Toeplitz Toeplitz Block (BTTB) can be represented as a sum of tensor products of form [2, 3]

$$A \approx \sum_{i=1}^{r} A_i \times B_i,$$

with factors A_i and B_i being Toeplitz. This is related to two-dimensional problems defined on tensor grids, with some "one-dimensional" structure in small factor matrices. The tensor product anzatz can be extended to the general case of $d > 2$ dimensions giving for many physically significant operators and their discretization matrices a nice representation which contains only $\mathcal{O}(n^{1/d})$ parameters. However, the title of this article suggests a class of matrices containing only $\mathcal{O}(\log^{\alpha} n)$ parameters [1], and this class is not related to high-dimensional problems directly. It contains well-known matrices. For example, one-dimensional Laplace operator,

$$\triangle = \text{tridiag}[-1, 2, -1],$$

is in this class, 2D and 3D Laplace operators are also there. The reader may say, that $\triangle$ is defined not by logarithmic number of parameters but by only three, being a banded Toeplitz matrix. However, the inverse of a banded Toeplitz matrix is not a banded Toeplitz matrix and the known structure of the inverse (semiseparable matrix) gives $\mathcal{O}(n)$ parameters, thus the class is "narrow". Note that $\mathcal{O}(n)$ is excellent for many applications, but in this case we can do even better. Another example is the Hilbert matrix defined as

$$A_{ij} = \frac{1}{i - j + \frac{1}{2}},$$

how to efficiently represent it? And what about the *inverse* of the 2D Laplace operator, it can be approximated by a small sum of tensor products [4], but it is still $\mathcal{O}(\sqrt{n})$ parameters at its best [5]. We hope the reader is intrigued in this point already and are ready to describe the new approach. It starts with relating the $n \times n$ matrix with some d-dimensional tensor, since the tensor decomposition in d dimensions gives awesome compression rates. An d-dimensional array (tensor) $\mathbf{A}$ is often approximated by the canonical decomposition of form

$$A(i_1, i_2, \ldots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) U_2(u_2, \alpha) \ldots U_d(i_d, \alpha), \quad 1 \leqslant i_k \leqslant n_k, \tag{1}$$

[1] When talking about asymptotic complexity, it is implicitly assumed that we have not one but a sequence of matrices A_n, related, for example, to finer discretizations.

where r is called *tensor rank*, and $n_k \times r$ matrices with elements $U_k(i_k, \alpha)$ are called *canonical factors*. This decomposition gives an excellent compression rate provided that it exists and r is small, but it is often hard and unstable to compute, and we need a robust algorithm. Instead we use the *TT-format*, introduced in [6] (the previous version, which is based on hierarchical approach was presented in [7]

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

where the summation over α_k in (2) goes from 1 to r_k. The values r_k play a crucial role in complexity and storage estimates and are called *compression ranks*. The decomposition is defined by three-dimensional tensors (hence the name "TT") G_k. It was shown that the computation of (2) reduces to a sequence of SVD decompositions, is simple to compute by standard algorithms from LAPACK, and often contains fewer parameters than the canonical decomposition (1). The compression procedure of [6] is our workhorse and the main "experimental tool". We take an array and test, whether it has a "good" TT-decomposition, giving us food for thought and theorems to prove. All of the examples presented in this papers were discovered by applying this subroutine (it will be presented shortly after)

There is still one link missing: from d-dimensional tensor to an ordinary "1D" matrix like the Laplace operator. And here is the main idea of the paper. From now on we suppose that n, the dimension of the matrix, is not arbitrary but a power of 2:

$$n = 2^d,$$

and A is a $2^d \times 2^d$ matrix. In line with the above discussion we reshape it into a $2d$-dimensional array. It can be written that

$$\mathbf{A} = A(i_1, i_2, \ldots, i_d, j_1, j_2, \ldots, j_d), \tag{3}$$

where i_k and j_k take values 0 or 1 only, and the rows of the matrix A are indexed by a d-tuple $(i_1, i_2, \ldots, i_d)$ and the columns of A are indexed by $(j_1, j_2, \ldots, j_d)$. i_k are simply digits in the binary form of the integer i. And the question is that if this tensor really admits a low-parametric decomposition. Suppose it is true and all ranks r_k are bounded by r. Then the TT-format has at most $4dr^2$ parameters, and $d = \log n$. by the definition. If r depends on n logarithmically, then we have found what we sought. The compression procedure is available, so experiments can be performed. However, direct application of the TT-decomposition to the tensor $\mathbf{A}$ failed, even for the the 1D dimensional Laplace operator the ranks were not small. What is the remedy? The problem is that we have taken all from a tensor, but not all from a matrix. By the analogy with the tensor product of matrices, reorder the $2d$-dimensional tensor $\mathbf{A}$ into a $2d$-dimensional tensor $\mathbf{B}$ with elements

$$B(i_1, j_1, i_2, j_2, \ldots, i_d, j_d) = A(i_1, i_2, \ldots, i_d, j_1, \ldots, j_d),$$

i.e. with i_k and j_k indices interleaved. In two dimensions, it is the idea of Van Loan and Pitsianis [8] for transforming Kronecker product approximation into a low rank approximation of a "shuffled matrix". In the d-dimensional case, such permutation of the dimensions is equivalent to the treatment of the matrix A as a d-level matrix with 2×2 blocks in each level, in a simplest case when all ranks are equal to 1 the TT-decomposition and the canonical decomposition coincide and A is a tensor product of 2×2 matrices. For this new tensor the TT-decomposition is used and now all the ranks remain small for many interesting examples. For the matrix $A_{ij} = \frac{1}{i-j+\frac{1}{2}}$ the ranks are bounded by 8 for the accuracy $\varepsilon = 10^{-6}$, for 1D-dimensional Laplacian they are at maximum 3, for two-dimensional Laplacian — maximum 4, for the inverse of the 1D Laplacian they are 5, for the inverse of two-dimensional Laplacian they seem to grow logarithmically in n and are about 54 for accuracy $\varepsilon = 10^{-6}$ and matrix size $10^6 \times 10^6$. So, the compression method looks very promising, allowing us to think about grids with millions (two-dimensional) and billions of unknowns (three-dimensional) problems, reducing the time to solve them significantly. The application of TT-decomposition to a specially permuted tensor, obtained from the given matrix gives a new matrix decomposition and a new matrix format. We will call it *TTM* format, from "TT-format" and "Matrix".

2. Notations and basic facts about TT-decomposition

Let us introduce some notations and recall some basic facts about the TT-format. The TT-decomposition of a d-dimensional tensor $\mathbf{A}$ with ranks r_k and core tensors G_k of sizes $n_k \times r_{k-1} \times r_k$ for $2 \leqslant k \leqslant d-1$ and $n_1 \times r_1$ for $k = 1$ and $n_k \times r_{k-1}$ for $k = d$ is defined as

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

where α_k goes from 1 to r_k. The most important property about the TT-decomposition is that the values of r_k can be readily estimated as the ranks of the *unfolding matrices* defined as

$$A_k = A(i_1 i_2 \ldots i_k; i_{k+1} \ldots i_d), \tag{5}$$

where the first k indices enumerate the rows of A_k and the last $d-k$ — the columns of it. Then it holds [6] that there exists a TT-decomposition such that

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

The most successfull decomposition of a d-dimensional tensor is the canonical decomposition (also known as CANDECOMP/PARAFAC model) [9, 10, 11, 12, 13] which has the form

$$A(i_1, i_2, \ldots, i_d) = \sum_{\alpha=1}^{r} W_1(i_1, \alpha) W_2(i_2, \alpha) \ldots W_d(i_d, \alpha), \tag{6}$$

and r is called *canonical rank*. If a tensor $\mathbf{A}$ admits canonical rank-r decomposition, then $\operatorname{rank} A_k \leqslant r$ and r_k is bounded by r. However, despite the progress for the algorithms computing the canonical decomposition [14, 12] there are no robust algorithms for obtaing it. Moreover, the problem of the best canonical approximation is often unstable and can be ill-posed [15, 16]. The TT decomposition is free of these redundancies and can be computed by using standard SVD and QR decompositions. Another approach based on the hierarchical splitting of the modes was presented in [7], but the nonrecursive variant is more simple and intuitive that is why it used in this paper.

In practice r_k can be much smaller than r, and the number of parameters in TT representation is less then in the canonical format [6]. The TT representation of a tensor can be obtained from a canonical representation of a tensor by using a recompression procedure, which is based on a sequence of QR and SVD decompositions and is robust, with guaranteed precision. These algorithms are also described in [6]. It is interesting to note that the compression ranks depend on the ordering of the dimensions, i.e. the tensor can be "compressible" (i.e. r_k are small) for one permutation of the dimensions and not compressible for another. This is an experimental evidence that *there is no canonical decomposition* with small ranks for such a tensor, since r_k are bounded from above by r for any permutation of the dimensions.

When a tensor is given as a full array (for example in our case, the tensor is obtained from an $n \times n$ matrix), an approximation algorithm is also needed. It is not hard to design an algorithm for full-to-TT compression. Following [6] we compute TT-decomposition step-by-step. First, take the unfolding matrix A_1 defined as

$$A_1 = A(i_1; i_2 \dots i_d),$$

and compute its truncated singular value decomposition

$$A_1 \approx U \Lambda V^\top, \tag{7}$$

where U is $n_1 \times r_1$ orthogonal matrix, Λ is $r_1 \times r_1$ diagonal matrix and a matrix V can be treated as $d-1$ dimensional tensor $\mathbf{V}$ with elements enumerated as

$$V(\alpha_1 i_2, i_3, \dots, i_d),$$

i.e. index α_1 which varies from 1 to r_1 is incorporated into the first dimension of the tensor V. To see that it can be done write (7) in the index form:

$$A(i_1, i_2, \dots, i_d) = \sum_{\alpha_1=1}^{r_1} U(i_1, \alpha_1) V(\alpha_1, i_2, \dots i_d).$$

The matrix U becomes G_1, the first core of the TT decomposition. It can be proven that the tensor V has a good TT approximation, and it can be decomposed analogously as the tensor $\mathbf{A}$ itself (by taking the first unfolding matrix):

$$V_1 \approx \widehat{U} \widehat{\Lambda} \widehat{V}^\top,$$

but now the rows of a small matrix $\widehat{U}$ are naturally indexed as

$$\widehat{U}(i_2, \alpha_1, \alpha_2),$$

and this is the second core G_2 of the decomposition. The process continues until the last dimension is reached. To prevent error accumulation at each SVD truncation (there will be $(d-1)$ of them) it is a good idea to transfer the diagonal matrix Λ of the SVD to the right factor, in this case it is easy to see that if we allow an absolute error ε at each step, the total accumulated error would be not higher than $d\varepsilon$, which is acceptable. The procedure is summarized in the Algorithm 1. The Matlab reshape and permute functions are utilized heavily for multidimensional array operations and have the same effect as their Matlab counterparts. For example, if A is a vector of length $n_1 n_2 n_3$ then reshape$(A, [n_1, n_2, n_3])$ gives a three-dimensional tensor $\mathbf{B}$ of size $n_1 \times n_2 \times n_3$ and permute$(\mathbf{B}, [2, 1, 3])$ gives a three-dimensional tensor $\mathbf{C}$ of size $n_2 \times n_1 \times n_3$ with elements given by

$$C(i_2, i_1, i_3) = B(i_1, i_2, i_3),$$

i.e the dimensions of a tensor are permuted according to the permutation vector.

Algorithm 1: Full-to-TT compression algorithm

Input: $n_1 \times n_2 \ldots \times n_d$ tensor $\mathbf{A}$, required accuracy ε.

Output: The cores $G_k, k = 1, \ldots, d.$ of the TT decomposition.

1: Compute nrm $= \|\mathbf{A}\|_F$.
2: Unfoldings size: $N_l = n_1, N_r = \prod_{k=2}^{d} n_k$.
3: Temporary tensor: $\mathbf{B} = \mathbf{A}$.
4: First unfolding: $M =$ reshape$(\mathbf{B}, [N_l, N_r])$.
5: Compute truncated SVD: $M \approx U\Lambda V^\top$, and set r to be the approximate rank of M.
6: Set $G_1 := U$, $M := \Lambda V^\top, r_1 = r$.
7: {Process other modes}
8: **for** $k = 2$ to $d - 1$ **do**
9: Set dimensions: $N_l := n_k, N_r := \frac{N_r}{n_k}$.
10: Construct unfolding: $M :=$ reshape$(M, [rN_l, N_r])$.
11: Compute truncated SVD: $M \approx U\Lambda V^\top$, and set $r_k = r$ to be the approximate rank of M.
12: Reshape and permute matrix U into a tensor:

$$G_k := \text{reshape}(U, [r_{k-1}, n_k, r_k]), G_k := \text{permute}(G_k, [2, 1, 3]).$$

13: Recompute M: $M := \Lambda V^\top$.
14: **end for**
15: $G_d = M^\top$.

The running time of Algorithm 1 can be roughly estimated as follows. Suppose that all mode dimensions n_k are equal to n, and the compression ranks are bounded by

r. The SVD of the first unfolding requires an SVD of a $n \times n^{d-1}$ matrix which costs $\mathcal{O}(n^{d+1})$ operations. If a rank-revealing method is used, for example cross approximation methods [17, 18] and the rank r is small, this cost can be reduced to $\mathcal{O}(n^{d-1}r)$, where r is the rank bound for A_1. For $k \geqslant 2$ the matrix has size $nr \times n^{d-k}$ and the number of operations is

$$\mathcal{O}((nr)^2 n^{d-k}) = \mathcal{O}(r^2 n^{d-k+2}),$$

if $nr \leqslant n^{d-k}$ and

$$\mathcal{O}(nr(n^{d-k})^2$$

otherwise. Denote the "turning" (i.e. when the index k when the matrix has more rows than columns) by b. From the inequality

$$nr \leqslant n^{d-b}$$

we have

$$(d-b) \geqslant \log_n r.$$

The total number of operations is estimated as

$$\mathcal{O}(C(n,r)),$$

where

$$C(n,r) = r^2 n^{d+2} \sum_{\alpha=2}^{b} n^{-k} + \sum_{\alpha=b+1}^{d-1} nrn^{2d-2k},$$

evaluating two geometrical progressions and separating the leading term we have

$$C(n,r) = \mathcal{O}(r^2 n^d \frac{n}{n+1}) = \mathcal{O}(n^d r^2).$$

If $N = n^d$ is the total number of elements in the multidimensional array, then the cost of the full-to-TT compression is asymptotically

$$\mathcal{O}(Nr^2).$$

If the cross approximation techniques are used, the factor n can be saved. To summarize, for the SVD-based approach for the computation of the TT-decomposition for a full array we have the following

Theorem 1. *Suppose that* $n_1 \times n_2 \times \ldots \times n_d$ d*-dimensional array* $\mathbf{A}$ *has TT-decomposition with compression ranks* r_k *and* $n_k \leqslant n, k = 1, \ldots d$ *and* $r_k \leqslant r, k = 1, \ldots, d-1$. *Then the TT-decomposition can be computed by means of* $(d-1)$ *SVD decompositions in*

$$\mathcal{O}(n^d r^2) \text{ operations.} \tag{8}$$

3. TTM decomposition: TT-decomposition of a matrix-produced tensor

Now let us go back to matrices. For an $2^d \times 2^d$ matrix A its *TTM decomposition*, or is defined as a TT-decomposition of a $4 \times 4 \times \ldots \times 4$ tensor $\mathbf{A}$ obtained from A in three steps. First, reshape it into a $2d$-dimensional tensor $\mathbf{B}$

$$\mathbf{B} = \text{reshape}(A, [2, 2, \ldots, 2]) = B(i_1, i_2, \ldots, i_d, j_1, \ldots, j_d),$$

then perform the interleaving of the dimensions into a tensor $\mathbf{C}$ by formulae

$$\mathbf{C} = \text{permute}(\mathbf{B}, [1, \frac{n}{2} + 1, 2, \frac{n}{2} + 2, \ldots, \frac{n}{2}, n]),$$

or in the index form

$$C(i_1, j_1, i_2, j_2, \ldots, i_d, j_d) = B(i_1, i_2, j_1, j_2, \ldots, i_d, j_d).$$

and treat (i_k, j_k) as a "long index" of length 4. In the end, we have the required tensor $\mathbf{A}$. Another interpretation is the following. We treat the initial matrix A as d-level matrix with rows indexed by a multiindex $(i_1, i_2, \ldots, i_d)$ and the column indexed by a multiindex $(j_1, j_2, \ldots, j_d)$. This d-level matrix admits a natural TT-decomposition with modes indexed not by one index, but with two indices, (i_k, j_k) treated as one, and the cores are indexed as

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

for $k \neq 1, d$ and $G_1(i_1, j_1, \alpha_1)$ and $G_d(i_d, j_d, \alpha_{d-1})$ for $k = 1, d$ respectively. This "virtual level" interpretation gives a clear way to the basic operations in linear algebra for matrices in the TTM format, namely matrix-by-vector product and matrix-by-matrix product by using the algorithms from [6] for d-level products directly. For example, to multiply two matrices in this format with compression ranks bounded by r_1 and r_2 one would need $\mathcal{O}(dr_1^2 r_2^2)$ operations and the ranks of the product would be bounded by $r_1 r_2$ in each mode. Obviously, $d = \log n$ and this is the logarithmic complexity.

For the TT-decomposition, the compression ranks are estimated as matrix ranks of unfoldings of a tensor. For the TTM-decomposition, there is an analogous relation of the compression ranks with the *tensor ranks* of auxiliary matrices A_k which correspond to different block splittings of A. Recall, that a tensor rank of a $nm \times nm$ matrix A (which can be treated as a 4-dimensional array $A(i_1, i_2, j_1, j_2)$) is the rank of the rearranged $n^2 \times m^2$ matrix B with elements

$$B(i_1, j_1, i_2, j_2) = A(i_1, i_2, j_1, j_2).$$

The tensor rank of a matrix depends on a splitting of the matrix dimension into a product $N = nm$. We will denote a tensor rank of a matrix A as $\text{trank}_{n,m} A$, and the lower index will be omitted when it is clear from the context what is the actual splitting of the matrix dimension. Any skeleton (dyadic) decomposition of B of form

$$B \approx UV^\top,$$

(U is $n^2 \times r$ and V is $m^2 \times r$), gives a tensor approximation of the initial matrix A. Each column of U (denote it by $u_i, i = 1, \ldots, r$) can be reshaped into a $n \times n$ matrix U_i, and the same for the corresponding columns of V. Then [8] A has a low-tensor rank approximation of form

$$A \approx \sum_{\alpha=1}^{r} U_i \times V_i,$$

where $\times$ is the usual tensor (Kronecker) product of matrices. It is not difficult to see, that the k-th unfolding matrix of a permuted tensor $\mathbf{A}$ is the matrix the matrix obtained from the initial matrix A by considering it as block matrix with $2^k \times 2^k$ blocks and applying the "Van Loan-Pitsianis" reordering of the matrix elements. Therefore, it is simple to estimate the compression ranks for the TTM representation of a matrix A.

Theorem 2. *For a $2^d \times 2^d$ matrix A its TTM compression ranks r_k satisfy*

$$r_k \leqslant \operatorname{trank}_{2^k, 2^{d-k}} A.$$

Therefore, the estimation of r_k is reduced to the estimation of the tensor ranks a matrix A for different splittings of it. Another interpretation of a tensor rank of a matrix, which may be useful, is the following. If A is $N \times N$ and $N = nm$, then A can be treated as $n \times n$ block matrix with $m \times m$ blocks. Denote these blocks by B_{ij} then $\operatorname{trank}_{n,m} A$ is equal to the dimension of a linear space spanned by matrices B_{ij}, i.e. is equal to the number of linearly independent blocks in A. This property of the tensor rank proves to be useful in obtaining theoretical estimates for the tensor ranks of matrices (sometimes it is enough to count the number of different blocks in a matrix).

The computation of the TTM decomposition is performed via reshaping the matrix into a d-dimensional tensor and applying Algorithm 1, with complexity $\mathcal{O}(n^2 r^2)$ (see Theorem 1).

The "virtual level" interpretation of the TTM format is very useful. For example, if a matrix is given as a tensor product of two matrices, A and B, then the complexity for obtaining the representation can be significantly reduced without any additional cost. If

$$C = A \times B,$$

and A and B are in the TTM-format, then the TTM representation of C can be computed by merging the representations of A and B. Specifically, if the cores for A are A_k, $k = 1, \ldots, d_1$ and for B they are B_k, $k = 1, \ldots, d_2$ then there are $d_1 + d_2$ cores C_k for C:

$$C_k = \begin{cases} A_k, & 1 \leqslant k < d_1 \\ X & k = d_1, \\ Y & k = d_1 + 1, \\ B_{k-d_1} & d_1 + 1 < k \leqslant d_2. \end{cases} \tag{9}$$

where X is a $n_x \times r^{(A)}_{d_1-1} \times 1$ "tensor" obtained from A_d by adding a fictions third dimension and Y is a $n_y \times 1 \times r^{(B)}_1$ tensor obtained from B_1 by adding a fictious second dimension

of size 1, and n_x, n_y are corresponding mode sizes. In short, the tensor product of two TTM matrices requires no operations, just stacking the cores "together".

4. Operations in TTM-format

4.1. Approximate matrix inversion

The TTM-format can be used to implement fast linear algebra. Everything is rather straightforward: a $2^d \times 2^d$ matrix is treated as a d-level matrix and the algorithms from [6] can be used. For example, we can compute an approximate inverse of a matrix in the TTM-format by applying the Newton iteration method:

$$X_{k+1} = 2X_k - X_k A X_k, \tag{10}$$

$k = 0, 1, \ldots$, where the initial approximation X_0 satisfies

$$\rho(AX_0 - I) < 1.$$

This requirement can be always satisfied by taking $X_0 = \alpha A^\top$ with sufficiently small α. For a symmetric positive definite matrix A we can take $X_0 = \alpha I$ with sufficiently small α. Our iterations requires projection of the approximate inverse onto a set of structured matrices. If the inverse matrix possesses a nice TTM structure, then the Newton iteration retains the quadratic convergence [19, 20, 21]. The iterations (10) require matrix-by-matrix products and compression of X_k at each step. For these two tasks the algorithms of [6] are utilized. As a test consider 2D Laplace operator, of form

$$A = I \times \triangle + \triangle \times I,$$

where $\triangle$ is discrete analogue of the 1D Laplace operator:

$$\triangle = \text{tridiag}[-1, 2, -1].$$

Some statistics is presented in Table 1. The approximate inversion was performed with $\varepsilon = 10^{-4}$ until the relative residue $\|I - AX_k\|_F / \|I\|_F$ is smaller than $5 \cdot 10^{-2}$. This approximate inverse is an excellent preconditioner for a linear system with a TTM matrix. In can be seen that the time to compute the approximate inverse does not change too much if the dimension of a matrix is multiplied by 4 and that is amazing.

n	Number of iterations	Final residue	Time
32^2	11	$2.2 \cdot 10^{-2}$	1 sec
64^2	12	$3.0 \cdot 10^{-2}$	3 sec
128^2	13	$3.3 \cdot 10^{-2}$	4 sec
256^2	15	$2.4 \cdot 10^{-2}$	14 sec
512^2	16	$2.0 \cdot 10^{-2}$	17 sec
1024^2	17	$2.5 \cdot 10^{-2}$	18 sec
2048^2	18	$2.6 \cdot 10^{-2}$	23 sec
4096^2	19	$2.6 \cdot 10^{-2}$	24 sec

Table 1. Approximate inversion of the 2D Laplace operator

4.2. Matrix-by-vector product

In the applications it is possible to have a structured $n \times n$ matrix A but unstructured (i.e. random) vector x, where $n = 2^d$. In that case A can be approximated in a TTM format, some of its characteristics (i.e., spectrum, norm) can be computed efficiently, as well as some some matrix functions can be computed fast. However, if the matrix-by-vector product with a full vector is needed, then a new algorithm has to be presented. It is not difficult to give such an algorithm. A matrix A is specified by its cores $A_k(i_k, j_k, \alpha_{k-1}, \alpha_k)$, a vector x can be treated as a full d-dimensional array $X = X(j_1, j_2, \ldots, j_d)$ and we want to compute

$$Y(i_1, i_2, \ldots, i_d) = \sum_{j_1, j_2, \ldots, j_d} A(i_1, j_1, \ldots, i_d, j_d) X(j_1, j_2, \ldots, j_d). \tag{11}$$

For generality suppose that the mode dimensions of $\mathbf{A}$ are equal to m (for TTM they are 2). The convolution can be performed step-by-step. First, convolve over j_1. This includes the computation of

$$Y_1(\alpha_1 i_1, j_2, \ldots, j_d) = \sum_{j_1} A_1(i_1, j_1, \alpha_1) X(j_1, j_2, \ldots, j_d),$$

which can be realized as a matrix-by-matrix product. If all ranks are bounded by r, then such convolution requires $\mathcal{O}(m^{d+1}r)$ operations. The TT format contains summation over $d-1$ indices, $\alpha_1, \ldots, \alpha_{d-1}$, and the summations over j_k and α_k can be combined. For the second core the summation is

$$Y_2(\alpha_2, i_1, i_2, j_3, \ldots, j_d) = \sum_{j_2 \alpha_1} A_2(i_2, j_2, \alpha_1, \alpha_2) Y_1(\alpha_1, i_1, j_2, \ldots, j_d),$$

which also need one matrix-by-matrix product and costs $\mathcal{O}(m^{d+1}r^2)$, for other cores the complexity will be the same. The total cost is

$$\mathcal{O}(dm^{d+1}r^2),$$

and for the TTM format it is

$$\mathcal{O}(nr^2 \log n).$$

for an $n \times n$ matrix, i.e. almost linear in n. However it should be pointed again that the best complexity can be obtained if the structure is imposed not only in the matrix, but in the vector also. In this case the logarithmic complexity is attained.

5. Numerical experiments

It is interesting to figure are out if there is any TTM-structure in some matrices. We have considered several examples. The compression rate depends on values of r_k, and their behavior is very interesting and requires a separate study for almost every example. But in all examples considered, they were bounded by a small number. The timings

were performed in Matlab, on a Intel Core2 notebook. We are interested in the values of ranks, r_k. There are a lot of them, so in most of the Tables only the maximal rank r_{max} and the *compression rate*, with is defined as

$$\frac{\mathrm{mem}(\mathrm{TTM})}{\mathrm{mem}(A)},$$

where $\mathrm{mem}(\mathrm{TTM})$ is the total number of elements in all core tensors.

5.1. 1D Laplace operator

As a first example, consider the discretization of one-dimensional Laplace operator with Dirichlet boundary conditions:

$$A = \mathrm{tridiag}[-1, 2, -1].$$

In Table 2 we report time needed to compute the TTM decomposition, the maximal value of r_k for different r. In this case, the decomposition appears to be exact.

n	r_{max}	Compression	Time
128	3	$1.2 \cdot 10^{-2}$	0.02 sec
256	3	$3.7 \cdot 10^{-3}$	0.05 sec
512	3	$1.1 \cdot 10^{-3}$	0.11 sec
1024	3	$3.0 \cdot 10^{-4}$	0.5 sec
2048	3	$8.3 \cdot 10^{-5}$	2 sec
4096	3	$2.3 \cdot 10^{-5}$	8 sec

Table 2. Timings and ranks for the 1D Laplace operator

5.2. Inverse of the Laplace operator

The second example is the inverse of 1D Laplace operator, considered previously. The results are given in Table 3.

n	r_{max}	Compression	Time
128	5	$3 \cdot 10^{-2}$	0.03 sec
256	5	$9 \cdot 10^{-3}$	0.06 sec
512	5	$2.6 \cdot 10^{-3}$	0.13 sec
1024	5	$7.5 \cdot 10^{-4}$	0.9 sec
2048	5	$2 \cdot 10^{-4}$	3.8 sec
4096	5	$5 \cdot 10^{-5}$	16 sec

Table 3. Timings and ranks for the inverse of a 1D Laplace operator

It appears that all r_k are equal to 5 exactly.

5.3. Hilbert matrix

The third example is the Hilbert matrix with elements

$$A_{ij} = \frac{1}{i - j + \frac{1}{2}}.$$

Now we can talk only about the approximation, so two experiments were performed. The first one is with fixed accuracy $\varepsilon = 10^{-6}$ and n is varying, the results are presented in Table 4, and the second experiment studies the dependence of the ranks from the accuracy parameter ε for a fixed $n = 1024$, the results are given in Table 5

n	r_{max}	Compression	Time
128	8	$5.4 \cdot 10^{-2}$	0.02 sec
256	9	$1.8 \cdot 10^{-2}$	0.06 sec
512	9	$5.8 \cdot 10^{-3}$	0.12 sec
1024	9	$1.7 \cdot 10^{-3}$	0.8 sec
2048	9	$5.0 \cdot 10^{-4}$	3.7 sec
4096	9	$1.4 \cdot 10^{-4}$	15 sec

Table 4. Timings and ranks for the Hilbert matrix, $\varepsilon = 10^{-6}$

ε	r_{max}	Compression	Time
10^{-3}	6	$7.8 \cdot 10^{-4}$	0.7 sec
10^{-4}	7	$1.1 \cdot 10^{-3}$	0.7 sec
10^{-5}	8	$1.4 \cdot 10^{-3}$	0.8 sec
10^{-6}	9	$1.7 \cdot 10^{-3}$	0.8 sec
10^{-7}	10	$1.9 \cdot 10^{-3}$	0.8 sec
10^{-8}	10	$2.2 \cdot 10^{-3}$	0.8 sec
10^{-9}	11	$2.6 \cdot 10^{-3}$	0.8 sec
10^{-10}	12	$2.9 \cdot 10^{-3}$	0.8 sec
10^{-11}	13	$3.3 \cdot 10^{-3}$	0.8 sec
10^{-12}	14	$3.6 \cdot 10^{-3}$	0.8 sec

Table 5. Timings and ranks for the Hilbert matrix, $n = 1024$.

5.4. 2D Laplace operator

Two dimensional Laplace operator, defined as

$$\triangle_2 = I \times \triangle + \triangle \times I,$$

possesses a natural two-level structure. However, if we introduce additional levels and use the TTM decomposition, the number of parameters is greatly reduced. We can

compute a TTM decomposition of $\triangle_2$ by using the tensor product of the representations of $\triangle$ and I via (9), and sum them and compress using the compression algorithm from [6]. For one-dimensional grid size $n = 1024$ the decomposition with following ranks was obtained.

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

Table 6. Compression ranks for different modes for a 2D Laplacian

Note that $r_{10} = 2$ since the tensor rank of the 2D Laplacian is 2.

5.5. Inverse of 2D Laplace operator

Another example is the inverse of the two-dimensional Laplace operator. To compute the approximation and evaluate the rank we used the following scheme. First, we computed the low-tensor rank approximation to A^{-1},

$$A^{-1} \approx \sum_{\alpha=1}^{r} X_i \times Y_i,$$

by using well-known approximations (see, for example [22, 4]) and then computed TTM decomposition of $n \times n$ factor matrices X_i and Y_i, then approximated for each summand, computed the sum and then compressed the result. For the accuracy parameter $\varepsilon = 10^{-6}$ the ranks are presented in Table 7. The compression ranks are higher for this case then in other examples, but from numerical experiments it clearly follows that $r_{max} = C \log n$, thus the total number of parameters to store the approximation is $\mathcal{O}(dr^2) = \mathcal{O}(\log^3 n)$. The constant hidden in $\mathcal{O}(\cdot)$ depends on the accuracy parameter ε. To study this dependence experimentally, we took fixed $n = 1024^2$ and computed the TTM-approximation for different ε. The results are presented in Table 8.

n	r_{max}	Compression
128^2	39	$1.1 \cdot 10^{-4}$
256^2	44	$1.0 \cdot 10^{-5}$
512^2	47	$8.7 \cdot 10^{-7}$
1024^2	51	$6.9 \cdot 10^{-8}$
2048^2	55	$5.2 \cdot 10^{-9}$

Table 7. Compression for the inverse of a 2D Laplace operator

ε	r_{max}	Compression
10^{-3}	19	$9.0 \cdot 10^{-9}$
10^{-4}	29	$2.2 \cdot 10^{-8}$
10^{-5}	40	$4.3 \cdot 10^{-8}$
10^{-6}	51	$6.9 \cdot 10^{-8}$
10^{-7}	62	$9.9 \cdot 10^{-8}$
10^{-8}	73	$1.3 \cdot 10^{-7}$
10^{-9}	83	$1.7 \cdot 10^{-7}$
10^{-10}	93	$2.1 \cdot 10^{-7}$
10^{-11}	102	$2.5 \cdot 10^{-7}$
10^{-12}	113	$2.9 \cdot 10^{-7}$

Table 8. Compression for the inverse of 2D Laplace operator, $n = 1024^2$.

The maximal r_k grows as $\log \varepsilon^{-1}$, but the r_k are rather "non-uniform", and the compression rate grows mildly. The whole rank pattern is presented in Table 9 for $\varepsilon = 10^{-12}$.

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
r_k	4	12	28	46	62	77	94	113	110	31	106	98	87	76	64	49	30	14	4

Table 9. Compression ranks for different modes for the inverse of a 2D Laplacian, $n = 1024^2, \varepsilon = 10^{-12}$.

5.6. Matrix-by-vector product

The matrix-by-vector product was also studied. The results are given in Table 10 and the accuracy is set to $\varepsilon = 10^{-6}$.

Size	Time
$n = 128^2$	0.15 sec
$n = 256^2$	0.89 sec
$n = 512^2$	4.6 sec
$n = 1024^2$	22.4 sec

Table 10. Matrix-by-vector product timings, $\varepsilon = 10^{-6}$.

6. Conclusions and future work

A new approach to matrix approximation is presented. Many familiar matrices, coming from integral and differential equations can be approximated by a low number of parameters, usually $\mathcal{O}(n)$ which is considered as good. This article shows that these

matrices possess a deep "inner structure" that allows to reduce the number of parameters significantly in some cases to even $\mathcal{O}(\log^{\alpha} n)$ parameters. The idea is to create additional dimensions and to treat a matrix as a d-dimensional tensor and then apply the TT-decomposition to this d-dimensional tensor (or, equivalently, d-level matrix). The d-level matrices can be also multiplied by a vector fast, and Newton method can be used to compute the approximate inverse and hence the preconditioner. The new format combining matrices and tensors (TTM format) looks very promising. A lot of work has to be done, both on mathematical and programming sides.

A close study is required for determining the actual class of matrices that can be casted in the TTM format with good accuracy and small compression ranks. Based on the experiments, the conjecture that this class contains discretizations of all physically important differential and integral operators on tensor grids looks not too bold.

Another question concerns more complicated regions in two and three dimensions, for example, triangle. Is it possible to apply these techniques for the discrete analogues of differential or integral operators, acting on such domain?

An interesting idea is to use the TTM format in many dimensions. For an d-dimensional operator considered on a grid with 2^k points in each dimension a combined format with an dk-dimensional operator can be proposed. Preliminary experiments on the discretization of the Schroedinger operator show that this approach is very effective, and the results will be reported in the forthcoming papers.

References

[1] *Delvaux S., Van Barel M.* A Givens-weight representation for rank structured matrices // *SIAM J. Matrix Anal. Appl.* 2007. V. 29, № 4. P. 1147-1170.

[2] *Kamm J., Nagy J. G.* Optimal Kronecker product approximation of block Toeplitz matrices // *SIAM J. Matrix Anal. Appl.* 2000. V. 22. P. 155-172.

[3] *Olshevsky V., Oseledets I. V., Tyrtyshnikov E. E.* Tensor properties of multilevel Toeplitz and related matrices // *Liner Algebra Appl.* 2006. V. 1. P. 1-21.

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

[5] *Olshevsky V., Oseledets I. V., Tyrtyshnikov E. E.* Superfast inversion of two-level Toeplitz matrices using Newton iteration and tensor-displacement structure // *Operator Theory: Advances and Applications*. 2008. V. 179. P. 229-240.

[6] *Oseledets I. V.* Compact matrix form of the d-dimensional tensor decomposition: Preprint 2009-01: INM RAS, 2009. — Mar. http://pub.inm.ras.ru/preprint-2009-01.

[7] *Oseledets I. V., Tyrtyshnikov E. E.* Breaking the curse of dimensionality, or how to use SVD in many dimensions // *SIAM J. Sci. Comp.* http://pub.inm.ras.ru/preprint-2009-02.

[8] *Van Loan C. F., Pitsianis N.* Approximation with Kronecker products // Linear algebra for large scale and real-time applications (Leuven, 1992). — Dordrecht: Kluwer Acad. Publ., 1993. — V. 232 of *NATO Adv. Sci. Inst. Ser. E Appl. Sci.* — P. 293–314.

[9] *Carroll J. D., Chang J. J.* Analysis of individual differences in multidimensional scaling via n-way generalization of Eckart-Young decomposition // *Psychometrika*. 1970. V. 35. P. 283-319.

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

[11] *de Lathauwer L., de Moor B., Vandewalle J.* A multilinear singular value decomposition // *SIAM J. Matrix Anal. Appl.* 2000. V. 21. P. 1253–1278.

[12] *de Lathauwer L., de Moor B., Vandewalle J.* Computing of Canonical decomposition by means of a simultaneous generalized Schur decomposition // *SIAM J. Matrix Anal. Appl.* 2004. V. 26. P. 295-327.

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

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

[15] *de Silva V., Lim L.-H.* Tensor rank and the ill-posedness of the best low-rank approximation problem // *SIAM J. Matrix Anal. Appl.* 2008. V. 30, № 3. P. 1084–1127.

[16] *Bader B., Kolda T.* Tensor decompositions and applications // *SIAM Review*. 2009. — Sep. V. 51, № 3. to appear.

[17] *Tyrtyshnikov E. E.* Incomplete cross approximation in the mosaic–skeleton method // *Computing*. 2000. V. 64, № 4. P. 367-380.

[18] *Bebendorf M.* Approximation of boundary element matrices // *Numer. Math.* 2000. V. 86, № 4. P. 565–589.

[19] *Oseledets I. V., Tyrtyshnikov E. E.* Approximate inversion of matrices in the process of solving a hypersingular integral equation // *Comp. Math. and Math. Phys.* 2005. V. 45, № 2. P. 302-313.

[20] *Hackbusch W., Khoromskij B. N., Tyrtyshnikov E. E.* Approximate iterations for structured matrices // *Numer. Mathematik*. 2008. V. 109, № 3.

[21] *Pan V. Y.*, *Rami Y.*, *Wang X.* Structure matrices and Newton's iteration: unified approach // *Linear Algebra Appl.* 2002. V. 343-344. P. 232-265.

[22] *Beylkin G.*, *Mohlenkamp M. J.* Numerical operator calculus in higher dimensions // *Proc. Nat. Acad. Sci. USA*. 2002. V. 99, № 16. P. 10246-10251.