Two-level Tucker-TT-QTT format for optimized tensor calculus

by

Sergey Dolgov, and Boris N. Khoromskij

Preprint no.: 19 2012
Two-level Tucker-TT-QTT format for optimized tensor calculus

Sergey Dolgov, Boris Khoromskij

† Max-Planck Institute for Mathematics in Sciences,
Germany, 04103 Leipzig, Inselstraße 22
[sergey.dolgov,bokh]@mis.mpg.de

AMS Subject Classification: 65N22, 65F50, 15A69, 33F05, 65F10, 65F30, 65N35

Keywords: Tensor formats, QTT-format, Tucker format, multilinear algebra, DMRG/ALS, higher dimensions, tensor methods, constructive tensor representations

Abstract. We propose a combined tensor format, which encapsulates the benefits of Tucker, Tensor Train (TT) and Quantized TT (QTT) formats. The structure is composed of subtensors in TT representations, so the approximation problem is proven to be stable. We describe all important algebraic and optimization operations, which are recast to the TT routines. Several examples on explicit function and operator representations are provided. The asymptotic storage complexity is at most cubic in the rank parameter, that is larger than for the QTT format, but the numerical examples manifest, that the ranks in the two-level format increase usually slower with the approximation accuracy than the QTT ones. In particular, we observe, that high rank peaks, which usually occur in the QTT representation, are significantly relaxed. Thus the reduced costs can be achieved.

1. Introduction

By a very general definition in mathematics, tensor is an array of values in \( \mathbb{C} \) with several indices \( A(i_1, \ldots, i_d) \), varying in some ranges \( 1, \ldots, n \). If all \( n_i \leq n \), one need at most \( n^d \) memory cells just to store all the elements. Such exponential growth with \( d \) is called the “curse of dimensionality” [1]. To get rid of this problem, the natural idea is not to keep all the elements explicitly, but rather approximate them via some low-parametric structured representation, which itself requires much less data to be stored. If only a few tensor entries are nonzeros, i.e. a tensor is (pointwise) sparse, the simplest approach is to store and use in computations only these elements [2,3].

Unfortunately, this is often not the case in practical problems (in particular, solution of multidimensional PDEs), and thus the development of some data-sparse techniques begun. Some of them are based on the idea of separation of variables from the functional analysis: the initial tensor is approximated with the sum of Kronecker products of one-dimensional vectors (so-called tensor formats). Others are based on properties of...
the discretization: the most important components, which can be represented with a low number of parameters are selected and smartly combined to eliminate the errors as far as possible (Sparse Grids, ANOVA decomposition [4, 5, 6, 7], wavelet and multiresolution analysis).

In this article, we consider only tensor formats. They are described by the splitting of the dimensions in the initial tensor (obviously, we can enumerate the same data arbitrarily), and the summation structure of the separable components. Even if the first issue is more or less fixed by the “physical” model, the second one gives us much more freedom, with totally different performance of numerical methods, based on a certain format.

During the history of tensor methods, different formats were proposed and analyzed, such as canonical [8, 9, 10, 11], Tucker [12, 13, 14, 15], Matrix Product States [16, 17, 18, 19] and MCTDH [20] in quantum physics community, HT (Hierarchical Tucker) [21, 22] and TT (Tensor Train) in numerical algebra community [23, 24, 25], and later the QTT (Quantized TT) [26, 27, 28, 29, 30]. Different comparisons of these formats were done, see the reviews [2, 31, 32, 33], and [34]. As the combined approach, the Tucker-Canonical format was applied to the solution of Hartree-Fock equation [35, 36].

In terms of the asymptotic complexity in n and d, the best one is the QTT format, with the storage demand \( O(d \log_2(n)r^2) \). However, the important question is, how large is the tensor rank bound \( r \) of a given tensor, and more important, of the tensors, arising during the solution of a certain problem. In a very general case, no theoretical answer is available.

In this paper we show, that in practice better results can be obtained with the smart combination of Tucker, TT and QTT formats (further called QTT-Tucker), rather then with only one of them. Namely, in many cases the Tucker and TT ranks are close. So the QTT approximation of the Tucker factors instead of the TT ones might require smaller ranks for the same accuracy, and moreover, avoid high rank peaks in the middle of Tensor Train. The latter property can be expected from the cyclic tensor networks (Tensor Chain), but approximation problems on them are usually unstable. The format proposed forms a closed manifold, as the Tucker, TT and HT formats [37, 38], hence it allows stable solution of optimization problems. The corresponding algorithms are described in this work.

The paper is organized as follows. In Section 2 we introduce the notations used, and briefly review the existing stable tensor formats. In Section 3, the new format, QTT-Tucker is introduced, and its basic advantages are discussed. In Section 4, we describe the TT-to-QTT-Tucker conversion, and provide the analytical examples, demonstrating the storage reduction possibilities of the new format. In Sections 5 and 6, the basic linear algebra operations, rounding (rank compression) and optimization methods are presented, the error analysis and complexity are discussed, with the use of the two-level Tucker-TT structure, and TT-structured subtensors. Finally, Sections 7 and 8 provide the numerical experiments to check the performance, and the conclusion.

2. Overview of tensor formats

2.1. Notations

Each element of the represented tensor is computed via sums over the rank indices. More complex tensor networks contain more ranks and sums. To write them explicitly
is too long and complicated. Instead, we will use the graphical notations, proposed in the quantum chemistry community [16,17], later in [39], and used also in [40]. The idea is very simple: the arrays are represented with rectangles, their indices are lines, and if a line connects two rectangles, it means that we multiply the elements of these arrays and perform the summation over the connecting index. For example, the following figures denote:

A matrix $A$ with two indices $i$, $j$: \[ \begin{array}{c|c} \text{i} & \text{j} \\ \hline A & \end{array} \]

A vector $[x]$: \[ \begin{array}{c|c} \text{j} & \end{array} x \]

The matrix-by-vector product $y = Ax = \left( \sum_j A_{ij} x_j \right)_{i=1}^n$ is now written as

\[ \begin{array}{c|c} \text{i} & \text{j} \\ \hline y & A \end{array} \]

Note that all blocks here are white, which means, that no restrictions are imposed. However, an important property we will need further is the orthogonality of cores.

**Definition 1.** An array $A(i_1, \ldots, i_p, j_1, \ldots, j_q)$ is said to be left-orthogonal, if

\[ \sum_{i_1, \ldots, i_p} \bar{A}(i_1, \ldots, i_p, k_1, \ldots, k_q)A(i_1, \ldots, i_p, j_1, \ldots, j_q) = \delta(k_1, j_1) \cdots \delta(k_q, j_q), \]

where $\bar{A}$ means the complex conjugation, and $\delta(k, j) = 1$ if $i = j$, and zero otherwise.

In a matrix case ($p = q = 1$) it means the well-known matrix orthogonality $A^*A = I$.

**Definition 2.** An array $A(i_1, \ldots, i_p, j_1, \ldots, j_q)$ is said to be right-orthogonal, if

\[ \sum_{j_1, \ldots, j_q} \bar{A}(m_1, \ldots, m_p, j_1, \ldots, j_q)A(i_1, \ldots, i_p, j_1, \ldots, j_q) = \delta(m_1, i_1) \cdots \delta(m_p, i_p). \]

In the graphical notation, the left/right orthogonality looks as follows:

The filled part of a rectangle emphasizes the indices, which are being convolved, give the identity tensor.

The orthogonality property is heavily utilized in the approximation and solution tensor routines, since it allows to control the introduced approximation error, and make orthogonal projections, see [24, 39, 41, 42].

We would like also to define some of array indices as parameters, to distinguish their purposes, and in consistency with the previous notations in MPS and TT community. Here are, for example, 3D array elements

\[ X(\alpha, i, \beta) = X_{\alpha,i,\beta} = X_{\alpha,\beta}(i), \quad \alpha, \beta = 1, \ldots, r, \quad i = 1, \ldots, n, \]

that could be written in the block form as

\[ X(i) = \begin{bmatrix} X_{1,1}(i) & \cdots & X_{1,r}(i) \\ \vdots & \ddots & \vdots \\ X_{r,1}(i) & \cdots & X_{r,r}(i) \end{bmatrix}, \]
or, if we need also to write explicitly the elements enumerated by \( i \) index,

\[
X(i) = \begin{bmatrix}
X_{1,1}(1) & X_{1,r}(1) \\
X_{1,1}(n) & X_{1,r}(n) \\
X_{r,1}(1) & X_{r,r}(1) \\
X_{r,1}(n) & X_{r,r}(n)
\end{bmatrix}.
\]

The latter is especially convenient for presenting the analytical structures in the format, since all the elements of a block are written in the same equation.

We emphasize here, that the same letters with the same superscripts denote the same arrays, and the structure of brackets or subscripts only specifies their certain permutation and reshaping. If a set of indices is written in brackets (like \( X(i) \)), all the rest indices are written in the subscript (\( X_{α,β}(i) \)), or omitted (\( X(i), X_{::}(i) \)) in the standard sense (omitted or substituted with “::” indices mean their full range, explicitly written - certain elements).

If there are only one or two subscripted indices, such object is considered as a matrix (vector), depending on indices in brackets as parameters, with row and column indices taken from the subscript. Any linear algebra operations (block concatenations, sums, scalar, Hadamard and contracted products, singular value decompositions, etc.) are then naturally defined.

2.2. Some existing formats

The oldest known closed tensor network (and, hence, the format with stable operations) is the Tucker format [12, 13, 14]:

\[
X(i_1, ..., i_d) = \sum_{γ_1, ..., γ_d} X^{(c)}(γ_1, ..., γ_d)X_{γ_1}^{(f_1)}(i_1) \cdots X_{γ_d}^{(f_d)}(i_d), \quad \text{or graphically,}
\]

```
  \(X^{(f_1)}\)  \(X^{(f_2)}\)  \(X^{(f_3)}\)  \(X^{(f_d)}\)
  \(i_1\)      \(γ_1\)      \(γ_2\)      \(γ_3\)      \(i_2\)
```

The \textit{mode} indices \( i_1, ..., i_d \) vary in ranges \( n_1, ..., n_d \), and the \textit{rank} indices \( γ_1, ..., γ_d \) - in ranges \( r_1, ..., r_d \). We see, that the storage of \( X^{(c)} \) still grows exponential in \( d \), though, if \( r ≪ n \), it is applicable to low-dimensional problems.

To avoid intrinsic curse of dimensionality, some new formats were proposed. Among
them are the Hierarchical (Tree) Tucker [21], or ML-MCTDH [20],

\[
X^{1234}_{\alpha_1 \beta_1, \gamma_1 \delta_1} \quad \cdots \quad X^{1234}_{\alpha_{d-1} \beta_{d-1}, \gamma_{d-1} \delta_{d-1}}
\]

and TT (MPS) [24, 17, 18],

\[
X^1_{\alpha_1} \quad X^2_{\alpha_2} \quad \cdots \quad X^d_{\alpha_{d-1}}
\]

In terms of parameter-dependent matrices it is written as \(X(i_1, ..., i_d) = X^1(i_1) \cdots X^d(i_d)\).

The mentioned above canonical tensor format is just a sum of rank-1 components (or the TT with diagonal blocks),

\[
X(i_1, ..., i_d) = \sum_{\gamma_1, ..., \gamma_d} X^{1}_{\gamma_1}(i_1) \cdots X^{d}_{\gamma_d}(i_d) = \prod_{k=2}^{d-1} \left[ X^k_{\gamma_k}(i_k) \right] \left[ X^d_{\gamma_d}(i_d) \right].
\]

Despite the simplicity, this format suffers from a serious drawback: it is not a closed tensor network, and the approximation problem is not stable [11].

3. Combining Tucker, TT and QTT into the two-level structure

The complexity of TT is \(O(dnr^2)\), but the complexity of the Tucker factors is \(O(dnr)\). To exploit the best properties of these two formats, in [43] was proposed to leave the Tucker factors “as is”, and compress into the TT only the core. This structure was named “extended TT decomposition”, and also reviewed in [33]. Thus, the storage of such representation is \(O(dnr + dr^3)\), i.e. asymptotically the same as in HT, but the structure is much simpler:

\[
X^{(c)1}_{i_1} \quad X^{(c)2}_{i_2} \quad \cdots \quad X^{(c)d}_{i_d}
\]

That is,

\[
X(i_1, ..., i_d) = \sum_{\gamma_1, ..., \gamma_d} [X^{(c)1}_{\gamma_1}(i_1) \cdots X^{(c)d}_{\gamma_d}(i_d)] X^{(f)1}_{\gamma_1}(i_1) \cdots X^{(f)d}_{\gamma_d}(i_d). \tag{1}
\]

Note, that the range of mode indices \(i_k\) can be formally even infinite, i.e. we store (say, as the analytical formulas) \(r_k\) functions in each factor \(X^{(f)k}\). But in the numerical calculus, the modes should be restricted to some finite values coming from the
discretization of functions. However, we might like to have so many grid points (e.g. several millions), that even linear complexity in \( n \) is prohibitive.

To use all the benefits from tensor decompositions for low-dimensional data, we use the QTT format \([26, 27, 28, 29, 44, 30, 45, 40]\). The idea behind quantization (hence the name QTT) is very simple. Each mode index \( i_k \) is further decomposed via the binary coding: 

\[
i_k = 1 + \sum_{p=1}^{L} (i_{k,p} - 1) \cdot 2^{p-1},
\]

where \( i_{k,p} \in \{1, 2\} \), so it varies in the minimal possible nontrivial range (“quant’’). If the \( k \)-th mode size is a power of 2, \( n = 2^L \), such coding enumerates all and no more values of \( i_k \). After that, the initial tensor is reshaped to \( 2 \times \cdots \times 2 \) (so its mode indices are \( i_{k,p} \)), and the TT approximation is applied.

The tucker factors are in fact the 1-dimensional functions, which are usually of the similar smoothness as the initial function. So we can apply the QTT for them:

\[
\begin{align*}
\gamma_k & \quad \Rightarrow \quad \gamma_k^{(f)k} \quad i_k^{(f)k} \quad \gamma_k \nonumber \\
\text{X}^{(f)k} (i_k) & \Rightarrow \quad \text{X}^{(f)k} (i_k^{(f)k}) \quad \cdots \quad \text{X}^{(f)k,1} (i_k^{(f)k,1}).
\end{align*}
\]

In the formulae notation

\[
X^{(f)k}(i_k) = X^{(f)k,L}(i_k,L) \cdots X^{(f)k,1}(i_k,1).
\]

The resulting network then looks as follows:

\[
\begin{align*}
\text{X}^{(e)1} (i_1,1) & \quad \alpha_1 \quad \text{X}^{(e)2} (i_2,1) \quad \cdots \quad \alpha_{d-1} \quad \text{X}^{(e)d} (i_d,1) \\
\gamma_1 & \quad \text{X}^{(f)1,1} (i_1,1) \quad \gamma_2 \quad \text{X}^{(f)2,1} (i_2,1) \quad \cdots \quad \gamma_{d-1} \quad \text{X}^{(f)d,1} (i_d,1) \\
\gamma_1 & \quad \cdots \quad \gamma_{d-1} \quad \gamma_d \quad \text{X}^{(f)d,L} (i_d,L) \\
\gamma_1,1 & \quad \cdots \quad \gamma_{d-1,1} \quad \gamma_{d,1} \quad \text{X}^{(f)d,L} (i_d,L)
\end{align*}
\]

Notice that here we write the same \( L \) for all factors for brevity, but there can be of course different \( L_k \). The formulae notation is more complicated: we have to rewrite (1), plugging the equation (2) instead of \( X_k^{(f)k}(i_k) \). That is why it is convenient to work in graphical notation with complex tensor structures.
Remark 1. The extended TT is closely related to HT format: in both cases the physical modes are separated in the Tucker factors, and the difference is only in the storage of the core. The TT format for the core is a certain realization of the tree sum structure from HT. However, we consider only TT variant due to its simplicity, and possibility of reduction of the algorithms to the TT-structured subtensors. The generalization to an arbitrary tree topology can be done straightforwardly with technical changes of the TT algorithms.

Moreover, the QTT representation of the Tucker factors introduces the new level of structure. A large tree in the HT format with small mode sizes but large dimension (e.g. applied to the tensor reshaped to $2 \times \cdots \times 2$) can give a serious overhead, see e.g. [46], spin system example.

In the following, to refer to such structure and its properties, we introduce our main definition.

**Definition 3.** (QTT-Tucker). The tensor format with the rank summation structure (3) is called the **QTT-Tucker** format. The parameter $d$ is called the physical, or core dimension, $L$ is the quantics, or factor dimension, $X^{c|k}$ is the $k$-th core block, $X^{f|k,p}$ is the $k,p$-th factor block, and the indices vary in the ranges:

- $i_{k,p} \leq n_{k,p} \leq n$ (mode size),
- $\alpha_k \leq r_{c,k} \leq r_c$ (core rank),
- $\gamma_k \leq r_{T,k} \leq r_T$ (Tucker rank), and
- $\gamma_{k,p} \leq r_{f,k,p} \leq r_f$ (factor rank).

Obviously, the extended TT decomposition is a certain case of the QTT-Tucker format, with $L = 1$.

The storage complexity is $O(\log_2(n) dr^2 + dr^3)$, which is formally even larger than in the standard QTT approach. For the approximation of discretized smooth functions, the benefit could be gained from the following consideration. Each TT block of a smooth function is usually also smooth. So, we can consider it as a set of $r_{c,k} - 1 \leq r_c \leq r_c$ one-dimensional vectors, and apply the QTT approximation to each of them. Suppose that each 1D vector has its own QTT rank bound $r_{\text{qtt}}$. Then, the maximal possible QTT rank of the whole tensor is estimated as $r_{\text{qtt}} r_c^2$.

If we apply the QTT to the Tucker factors, their QTT rank is bounded by

$$r_{\text{qtt}} \leq r_f \leq r_{\text{qtt}} r_T.$$  

Though in general case these equations give the same estimate (see Lemma [below]), most numerical experiments with the smooth functions manifest the Tucker rank of the same order, as the TT one, $r_f \sim r_c$. Thus, the rank estimate of the QTT-Tucker reads

$$r_{\text{qtt}} r_c$$

for the quantized factors, and

$$r_c$$
for the TT-approximation of the core. Recalling the number of elements in the TT representation, we obtain the memory estimates:

\[
\text{mem}(\text{QTT}) = O(d \log_2(n)r_{qtt}^2 r_c^3), \\
\text{mem}(\text{QTT-Tucker}) = O(d \log_2(n)r_{qtt}^2 r_T^2 + dr_c^3).
\]

It could give a significant reduction of the complexity, see the numerical experiments below. In the following, unless explicitly specified, we assume \(r_T \sim r_c\), e.g. the storage becomes \(O(d \log_2(n)r_{qtt}^2 r_c^3 + dr_c^3)\). It is usually the case in practical problems with smooth functions. However, in general the following rank bound holds:

**Lemma 1.** If the same tensor is represented in TT format with the rank bound \(r_c\), and in the Tucker format with the rank bound \(r_T\), they are connected as follows:

\[r_c \leq r_T \leq r_c^2.\]

The same result was also obtained for the HT format in [34]. The constructive proof is presented in the next section.

4. Explicit representations

4.1. TT to Tucker conversion

During the exploitation of various tensor formats, it is helpful to construct some simple and widely used objects explicitly, by specifying the format elements directly. For the QTT format this work started from [27, 29]. For example, consider the Laplace-like sum:

\[A = x \otimes y \otimes \cdots \otimes y + \cdots + y \otimes \cdots \otimes y \otimes x.\]

The exact rank-2 TT representation reads

\[A = \begin{bmatrix} x & y \\ y & x \end{bmatrix} \begin{bmatrix} 0^{d-2} \\ y \end{bmatrix}, \]

which is obtained simply by successive splitting linearly independent elements of the leading dimension to the current block, i.e. reconstructing the TT-SVD algorithm [23, 24] analytically. The Tucker decomposition can also be obtained by similar reconstruction of HOSVD [13]. However, here we also present the transformation of TT to extended TT (it is usually easier to compute first the analytical TT-SVD rather than HOSVD).

This method will be done in parallel over the TT blocks. Consider the current \(k\)-th TT block,

\[X^k(i_k) = \begin{bmatrix} X_{i_{1,1}}^{k}(i_k) & \cdots & X_{i_{r_k}}^{k}(i_k) \\ \vdots & \ddots & \vdots \\ X_{r_{k-1,1}}^{k}(i_k) & \cdots & X_{r_{k-1,r_k}}^{k}(i_k) \end{bmatrix}.\]
Now, find the minimal set of $R_k$ linearly independent elements $X^{(f)}_{k\alpha_{k-1,\alpha_k}}(i_k)$. Enumerate them $X^{(f)}_{\gamma_k}(i_k)$, where $\gamma_k = 1, \ldots, R_k$. Clearly, now $X^k$ can be represented as

$$\begin{bmatrix}
X^{(c)}_{1,1}(\gamma_k) & \cdots & X^{(c)}_{1,r_k}(\gamma_k) \\
\vdots & \ddots & \vdots \\
X^{(c)}_{r_k-1,1}(\gamma_k) & \cdots & X^{(c)}_{r_k-1,r_k}(\gamma_k)
\end{bmatrix}
\begin{bmatrix}
\alpha_{k-1} \\
\vdots \\
\alpha_k
\end{bmatrix}
= 
\begin{bmatrix}
X^{(f)}_{1}(i_k) & \cdots & X^{(f)}_{r_k}(i_k)
\end{bmatrix}
\begin{bmatrix}
\gamma_1 \\
\vdots \\
\gamma_{d-1} \\
\gamma_d
\end{bmatrix}
\begin{bmatrix}
x y \\
x y \end{bmatrix}
\begin{bmatrix}
i_1 \\
i_2 \\
\vdots \\
i_d
\end{bmatrix}
$$

where $X^{(c)}_{\alpha_{k-1,\alpha_k}}(\gamma_k)$ are the coefficients of $X^k_{\alpha_{k-1,\alpha_k}}(i_k)$ in the basis $X^{(f)}_{\gamma_k}(i_k)$. Obviously, the maximal number of linearly independent elements in $k$-th block is not greater than $r_{k-1}r_k$. On the other hand, if the rank of the initial block $X^k$ was not overestimated, the minimal number of linearly independent components is not smaller than $r_c$. This constructive algorithm confirms Lemma 1.

**Remark 2.** The extraction of the Tucker factor from the TT core can be also done approximately via the SVD. In this case, we choose not all, but the truncated amount of linearly independent principal components to ensure a certain accuracy.

The first Laplace-like example in this section contains only 2 linearly independent elements in each block, so its decomposition reads (we omit $i_2, \ldots, i_{d-1}$ for brevity):

$$\begin{bmatrix}
(1) & (0) \\
0 & (1)
\end{bmatrix}
\begin{bmatrix}
(0) & (0) & \cdots & (0) \\
(1) & (0) & \cdots & (0)
\end{bmatrix}^{d-2}
\begin{bmatrix}
(0) & (1) \\
1 & 0
\end{bmatrix}^T
\begin{bmatrix}
\gamma_1 \\
\gamma_2 \cdots \gamma_{d-1} \\
\gamma_d
\end{bmatrix}
\begin{bmatrix}
x y \\
x y
\end{bmatrix}
\begin{bmatrix}
i_1 \\
i_2 \\
\vdots \\
i_d
\end{bmatrix}
$$

Remarkable, we obtain all the same Tucker factors, and the core contains only zeros and ones to combine the factors correctly.

Other tensor networks can be converted first to the TT format (or a sum of TT tensors, for example, the Tensor Chain), and then into the QTT-Tucker. It could be helpful in quantum physical problems, if a model provides initial data in a complicated tensor network.

### 4.2. Operator example

Now, derive the quantics representation for $[x, y] = [\Delta_L, I_L]$, where $\Delta_L$ is the finite difference Laplacian with Dirichlet boundary conditions on a grid with $2^L$ nodes. In [29]
its explicit QTT representation was proven

\[
\Delta L = \begin{bmatrix}
\Delta_1 & -J & -J' \\
\end{bmatrix}
\]

where \(\Delta_1\) is a \(2 \times 2\) Laplacian matrix, \(I_1\) is the identity of size 2, and \(J = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}\) is a Jordan block. We note that the first row of the middle blocks always contains only the \(I_1\) at the first place, and zeros at the others, whereas the last block contains \(I_1\) as the first item as well. Therefore, the full identity matrix \(I_L\) can be represented using the same blocks as \(\Delta_L\) except the first:

\[
\begin{bmatrix}
\Delta_L \\
I_L \\
\end{bmatrix} = \gamma_1 \begin{bmatrix}
\Delta_1 & -J & -J' \\
I_1 & J' & (J) \\
\end{bmatrix}^{L-2} \gamma_{L-1} \begin{bmatrix}
I_1 & J' \\
J & (J') \\
\end{bmatrix}
\]

Plugging this decomposition into (4), we obtain the QTT-Tucker representation of the Laplace operator with factor ranks at most 3, and core ranks 2.

**Lemma 2.** The finite difference Laplace operator on uniform \(2^L \times \cdots \times 2^L\) grid with Dirichlet boundary conditions is exactly representable in the QTT-Tucker format with the ranks

\[
r_f = 3, \quad r_T = 2, \quad r_c = 2.
\]

Notice, that the crucial point here is to put the “enumerating” Tucker index \(\gamma\) in the lowest bit block. If we connect \(\gamma\) to the senior bit, the rank will be 4. Recall that in [29], the QTT decomposition of the multidimensional Laplacian was obtained with rank 4 for all blocks except one, since there are the TT rank indices on both sides of inner subtrains. This simplest example confirms the considerations in the previous section: instead of two TT rank indices in the “linear” QTT structure, the QTT-Tucker format requires only one Tucker rank index to be connected to the QTT representation of one-dimensional objects. Thus, choosing the proper bit order, we can reduce the rank of the common QTT basis.

\[\text{here we write the blocks, corresponding to lower bits on the left, i.e. with the Matlab and Fortran little-endian indexing, whereas in [29] they are written on the right}\]
4.3. Quadratic form example

In many applications we are to compute the following quadratic form:

\[ V(X', X) = \sum_{i,j=1}^{d} B_{ij} X'_i \otimes X_j, \quad \text{where} \]

\[ X'_i, X_j \text{ are the following rank-1 tensor product objects,} \]

\[ X'_i = e_1 \otimes e_2 \cdots \otimes x'_i \otimes \cdots e_d, \quad X_j = e_1 \otimes e_2 \cdots \otimes x_j \otimes \cdots e_d, \]

\( \otimes \) is distributive with \( \otimes \) (e.g. Hadamard or matrix product), \( e_k \) are the unities with respect to \( \otimes \), and \( B \) is a matrix of scalar numbers. The quadratic Lyapunov function\(^2\) is such a form, with \( e_k \) being vectors of all ones, \( x_k = x'_k \) are the vectors of grid points, and \( \otimes \) is the Hadamard product. For a general polynomial, it was investigated in [41], where the rank bound \( d + 2 \) was proven for the polynomial degree 2. Here, we give a refined result. An explicit construction of a low-rank representation of \( V \), provided that the basis elements \( e_k, x_k, x'_k, (x'_k \otimes x_k) \equiv x_k^2 \) are defined, gives the following

**Theorem 1.** Introduce the off-diagonal lower \( C^k = B_{k,k+1:d} \), upper \( \hat{C}^k = B_{1:k,k+1:d} \) submatrices. The quadratic form (5) possesses an exact TT decomposition with the ranks:

\[ r_{TT,k} = \text{rank}(C^k) + \text{rank}(\hat{C}^k) + 2 \leq d + 2. \] (6)

Moreover, if \( x_k = x'_k \) (symmetric case), the ranks reduce to

\[ r_{TT,k} = \text{rank} \left( \frac{(C^{k^T} + \hat{C}^k)}{2} \right) + 2 \leq d/2 + 2. \]

The proof is rather technical and thus is written in Appendix. The Tucker ranks equal 4 (3 in the symmetric case), since there are 4 (resp. 3) linearly independent elements in each block. The QTT ranks of the Tucker factors depend on given \( x_k \). In the case of a degree 2 polynomial on a uniform grid, they are equal to 3 [27, 30].

**Corollary 1.** The estimate (6) is applicable to the \( d \)-dimensional anisotropic elliptic operator \( \nabla^\top B \nabla \) with a constant matrix \( B \), by setting \( e_k = I_k, x_k = \nabla \nabla_k, x'_k = \nabla^\top \nabla_k, x_k^2 = \Delta_k \), and \( \otimes \) being the operator composition. A discussion on such structures was started in [29, 47].

**Remark 3.** The ranks of the off-diagonal blocks are the so-called *semiseparable* ranks. Theorem 1 establishes a connection between the matrix semiseparability and the TT structure. The rank estimate is numerically sharp. If the matrix is diagonal, i.e. \( C^{k^T} = \hat{C}^k = 0 \), then the TT ranks equal 2 (Laplace operator, harmonic oscillator potential).

5. Fast QTT-Tucker arithmetics

Very basic operations worth to have with a format, are the linear algebra operations: linear combinations, matrix products, etc. For all tensor formats, they are easily derived, using the following rules:

---

\(^2\)For example, it arises in calculation of the typical \( d \)-dimensional probability density \( \psi = \exp(-V) \)
• separable components are added, or multiplied independently in all variables,
• all rank sums (and hence, the ranks) are added in linear operations, and multiplied in bilinear ones.

During the iterative methods, the rank can grow very high due to multiplications. Hence, another important operation, which should be provided with the format, is the rank truncation.

The QTT-Tucker format consists from several connected Tensor Trains. So, it will be convenient in the following to split the description of a certain procedure to the underlying TTs, for which it is known.

5.1. Linear combination, matrix and Hadamard products

We recall, that the QTT-Tucker contains “physical” modes (i.e. the modes, which the initial operation is conducted on) only in factors. Therefore, we can recast the operations to the factors, describing only the proper combination of cores.

• **Linear combination** \( X = aX + bY \): add

\[
Z^{(f)} = aX^{(f)} + bY^{(f)}, \quad Z^{(f)k} = X^{(f)k} + Y^{(f)k}, \quad k > 1
\]
as tensor trains, with addition of the “tailing” (Tucker) rank \( r_T^X = r_T^Y + r_T^Y \) \[24\]; block-diagonal concatenation of core blocks

\[
Z^{(c)k} = \begin{bmatrix}
X^{(c)k}(\gamma^X_k) & Y^{(c)k}(\gamma^Y_k)
\end{bmatrix}.
\]

• **Matrix-by-matrix(vector) product** \( y = Ax \): multiply

\[
Y^{(f)k} = A^{(f)k}X^{(f)k}
\]
as tensor trains, with multiplication of the “tailing” (Tucker) rank \( r_T^A = r_T^X \cdot r_T^Y \) \[24\]; Kronecker product of core blocks

\[
Y^{(c)k} = A^{(c)k} \otimes X^{(c)k}.
\]

• **Hadamard** product is reducible to the matrix product: \( z = x \odot y \iff z = \text{diag}(x)y \).

• **Dot** product \( a = (x, y) \) requires more detailed description. First, we perform the dot products over factors:

\[
A^k = (X^{(f)k}, Y^{(f)k}) \in \mathbb{P}^{r_T^X \times r_T^Y}.
\]

It is the matrix of size \( r_T^X \times r_T^Y \), since it contains all combinations of 1D vectors in the Tucker factors. Now, the sum structure in dot product looks as follows:

\[
\begin{array}{cccc}
X^{(c)1} & X^{(c)2} & \cdots & X^{(c)d} \\
\gamma_1^X & \gamma_2^X & \cdots & \gamma_d^X \\
A^1 & A^2 & \cdots & A^d \\
\gamma_1^Y & \gamma_2^Y & \cdots & \gamma_d^Y \\
\end{array}
\]

12
Convolving the matrices $\Lambda^k$ to one of the cores, and taking their TT-dot product, we obtain the result.

5.2. QTT-Tucker rounding

The algorithm of rounding procedure should be written carefully to make the method stable. In [24] it was shown, that the only way to control the error introduced to some part of the format, is to impose the proper orthogonality on the other connected parts. We can reduce the QTT-Tucker rounding procedure to the structured variants of HOSVD and TT-rounding (for the definitions and descriptions of the TT-orthogonalization and TT-rounding we refer to [24]), but due to the TT-structure of the core, certain orthogonalizations will be written explicitly. We summarize the method in Algorithm 1.

**Algorithm 1:** QTT-Tucker round

**Input:** QTT-Tucker tensor $X$, accuracy bounds $\varepsilon_{k,p}$, or rank bounds $R_{k,p}$.

**Output:** QTT-Tucker tensor $Y$ with possibly smaller ranks, and $||Y - X||^2 \leq \sum \varepsilon_{k,p}$, or $R_{k,p} \leq R_{k,p}$.

1: for $k = 1, \ldots, d$ do
2: Make TT-tensor $X^{(\ell)}(i_k, \ldots, i_k) X^{(\ell)}(\gamma_k)$ left-orthogonal (i.e. the k-th factor connected with k-th core block, so that k-th factor becomes orthogonal.)
3: end for
4: Round the core $Y^{(c)} = \text{round}_{TT}(X^{(c)}, \varepsilon_k, R_k)$, using the TT-round procedure, so that $Y^{(c)}$ is right-orthogonal. [Complexity $O(d \ell^2)$]
5: for $k = 1, \ldots, d - 1$ do
6: Round the factor: $Y^{(c)}(i_k) \tilde{Y}^{(c)}(\gamma_k) = \text{round}_{TT}(X^{(c)}(i_k) Y^{(c)}(\gamma_k), \varepsilon_{k,p}, R_{k,p})$
7: Make the core block $\tilde{Y}^{(c)}(\gamma_k)$ left-orthogonal: [Complexity $O(d \ell^2)$]
8: Convolve the matrix $R$ to the next block: $Y^{(c)}(\gamma_{k+1}) := R Y^{(c)}(\gamma_{k+1})$.
9: end for
10: Round the last factor [Complexity $O(L \ell^2)$] $Y^{(c)}(i_d) Y^{(c)}(\alpha_{d-1}) = \text{round}_{TT}(X^{(c)}(i_d) Y^{(c)}(\alpha_{d-1}), \varepsilon_{d,p}, R_{d,p})$.

We emphasize here the operation of building the “factor-one-core-block” TT-tensor, by formally denoting the core rank indices $\alpha_{k-1}, \alpha_k$ as the new “physical” index, and considering $\gamma_k$ as the rank index, so that $Y^{(c)}(\alpha_{k-1}, \alpha_k) \rightarrow \gamma_k$ (with the same elements, but now with the rank index $\gamma_k$ consistent with the TT representation of factor.

**Definition 4.** The TT-tensor $X^{(c)}(i_k, \ldots, i_k) X^{(c)}(\gamma_k)$, with the last block obtained from the core block with the formal permutation of indices $\alpha_{k-1}, \gamma_k, \alpha_k \rightarrow \gamma_k, (\alpha_{k-1}, \alpha_k)$, is called the k-th extended factor.

This definition allows us to consider (and update) the Tucker ranks in the uniform way, by referring to the corresponding ranks of certain TT tensors. It will help us also in the next section.
It is also interesting, that the orthogonalization/rounding of the extended factors (lines 2, 6 and 10 of the algorithm) in fact mimics the Tucker HOSVD: the difference is in using only one block of the core instead of the whole core as the full array. Lines 4, 7, 8 take into account the TT structure of the core more carefully. The states of the QTT-Tucker tensor after certain lines of Algorithm 1 are shown on Fig. 1 (the sizes of the blocks are proportional to their ranks).

Figure 1. QTT-Tucker rounding steps in graphical notations

To finish with rounding, we provide the error analysis.

**Theorem 2.** Suppose, that each truncation in QTT-Tucker factors was done with the Frobenius error $\varepsilon_{k,p}$, $p > 0$, the Tucker ranks (ranges of $\gamma_k$ in extended factors) was determined with $\varepsilon_{k,0}$, and each core block is truncated with $\varepsilon_k$. Then, the Frobenius error in the whole tensor reads

$$||Y - X||^2 \leq \sum_k \varepsilon_{k,p}^2 + \sum_k \varepsilon_{k,0}^2 + \sum_k \varepsilon_k^2.$$  

*Proof.* We split the claim of the theorem to the corresponding estimates of the HOSVD and TT-round errors. First, suppose that each factor is approximated with the error $\delta_k$, and the core is exact. Then, from the HOSVD theorem in [13] we obtain

$$||Y - X||^2 \leq \sum_k \delta_k^2.$$  

Second, if the core is approximated with the error $\delta$, it is added to the estimate:

$$||Y - X||^2 \leq \sum_k \delta_k^2 + \delta^2. \quad (7)$$  

Finally, each $\delta_k$ and $\delta$ can be obtained from the TT-rounding theorem [24], since they arise from the TT-rounding procedure for extended factors and core, respectively. We have

$$\delta_k^2 \leq \sum_p \varepsilon_{k,p}^2 + \varepsilon_{k,0}^2, \quad \delta \leq \sum_k \varepsilon_k^2. \quad (8)$$  

Now, plugging (8) into (7), we obtain the statement of the theorem. $\square$

**Corollary 2.** If for all blocks the truncation error is fixed to the same value $\varepsilon$, the total accumulated error is not greater than

$$||Y - X|| \leq \sqrt{dL + d - 1} \varepsilon.$$  

In practical implementation, setting the local accuracy $\varepsilon_k = \varepsilon/\sqrt{2d}$ for the core truncation, and $\varepsilon_{k,p} = \varepsilon/\sqrt{2dL}$ for the factors provides the controlled accuracy in the whole tensor $\varepsilon$. 14
Remark 4. The Tucker, TT or extended TT are certain cases of the QTT-Tucker format. The error estimate above reduces to the corresponding estimates of simpler formats straightforwardly, by setting certain terms (being exactly represented) to zeros.

6. Optimization problems on the QTT-Tucker format

In numerical modeling, we will not be satisfied only with the data compression techniques. The problem of the same importance, is how to solve a certain equation keeping all the data in the format.

We are to optimize a functional \( J(x) \) with \( x \) in the format. We are interested in the following functionals:

- **error** \( J = \|x - y\|^2 \),
- **energy** \( J = (Ax, x) - 2(y, x) \),
- **residual** \( J = \|Ax - y\|^2 \), or
- **Rayleigh quotient** \( J = \frac{(Ax, x)}{(x, x)} \).

They correspond to the approximation, linear system solution and eigenvalue problems, and what is important, are reducible to the format parameters without changing the form, which allows the application of efficient matrix methods.

Provided with the fast algebra and rounding procedure, one can implement all the standard matrix-vector methods, see, for example, the Krylov methods in TT and HT [48, 49]. However, these methods are required to keep some intermediate vectors of the same size, as the initial tensor, thus they also must be approximated. Unfortunately, in many cases, the better the solution is, the worse structure (and larger ranks) these vectors have (e.g. residual, Krylov vectors, etc.), and though advanced techniques relax the complexity to some extent, much better way is to deduce a solver, exploiting the format structure.

Since the canonical format, one of such techniques was the Alternating Least Squares (ALS) method for the approximation and solution purposes. It uses the separation of variables naturally: all the factors except one are fixed, the minimization/solution problem is reduced only to its elements, then some other factor is updated (usually, the next), and so on. The similar approach is applicable to more complex formats, such as Tucker, TT, HT. However, it has several serious drawbacks:

- the ranks must be prescribed, and are not changed during the iterations, i.e. the method is not adaptive.
- the convergence might be very slow (e.g. millions of iterations)
- in some formats (e.g. canonical), the problem is unavoidable ill-conditioned and unstable [11].

Fortunately, for the Tucker, TT and HT formats, there exist modifications, which can treat these drawbacks. There is lack of convergence analysis (a sort of is given in [50], but almost inapplicable in practice), but most of numerical experiments with the practical problems manifest good performance.
6.1. From the Tucker ALS to the QTT-Tucker DMRG

For the Tucker format, the alternating scheme called TALS (Tucker ALS) (see [14] for the approximation problem case $J = \|x - y\|^2$) is conducted as described in Algorithm 2. We see, that the reduced optimization problem affects two connected blocks of the

Algorithm 2: Tucker ALS iteration

Input: The functional $J(x)$, initial guess $x$ (with orthogonal $X^{(f)k}$), accuracy $\varepsilon$ or rank $R$ tolerances.

Output: Improved solution $x$ in the Tucker format.

1: for $k = 1, \ldots, d$ do
2: Fix all the factors except $k$-th, optimize over this factor connected with core:
   $W^k = \arg\min_{\sum Y_{yk}X^{(c)}(\gamma_k, \ldots, :)} J(x) \in \mathbb{R}^{n \times r_{d-1}}$.
3: Split the updated factor and core:
   $[X^{(f)k}, S, X^{(c)}] = \text{svd} (W^k, \varepsilon, R)$, $X^{(c)}(\gamma_k, :)$.
4: end for

format, thus the rank can be updated on each step of the algorithm, which is similar to the DMRG algorithm [16, 41, 39, 42] for the MPS (TT) format.

We briefly review the TT-ALS and TT-DMRG algorithms here. For technical details, see [41, 39, 42].

The major difference between the Tucker ALS and the TT-ALS methods, is that the TT-ALS is not adaptive, and all ranks are fixed to some prescribed values. That is, if they are underestimated for the given accuracy, it can not be reached (however, the method usually converges to the optimal solution with given ranks). The TT-ALS method is written in Algorithm 3.

Algorithm 3: ALS_{TT} iteration

1: for $k = 1, \ldots, d, d - 1, \ldots, 1$ do
2: Fix all the blocks except $k$-th, optimize $X^k(i_k) = \arg\min_{X^k} J(x)$.
3: To make the reductions of the functional easier (see [39]), make $X^k$ left-orthogonal (if $k$ is increasing), or right-orthogonal (if $k$ is decreasing).
4: end for

The TT-DMRG ([16], MALS in [39], TT-solve in [42], DMRG MatVec in [51]) allows to determine the ranks adaptively at runtime. Instead of minimizing over each block, it merges two neighboring blocks into one larger with two mode indices, performs the ALS optimization step on such $d-1$-dimensional tensor, and then splits this large block back to two TT blocks via the SVD. On the latter step, the rank is updated, since the data was improved during optimization. See Algorithm 4.

What is nice, the Tucker ALS scheme generalizes to the QTT-Tucker case straightforwardly by meaning of the following substitutions:

- Instead of the factor-core connected block $W^k = X^{(f)k}X^{(c)}$, we can consider the extended factor (see def. [4]) of sizes $n \times r^2$, like in the QTT-Tucker rounding [1]
the core is stored in the full format, the method reduces to the simple Tucker ALS.

Input: The functional \( J(x) \), initial guess \( x \), accuracy \( \varepsilon \) or rank \( R \) tolerances.

Output: Improved solution \( x \) in the QTT-Tucker format.

1. for \( k = 1, \ldots, d - 1, d - 2, \ldots, 1 \) do
2. Fix all the blocks except \( k \) and \( k + 1 \), merge \( W^k(i_k, i_{k+1}) = X^k(i_k)X^{k+1}(i_{k+1}) \).
3. optimize \( W^k(i_k, i_{k+1}) = \arg \min_{W^k} J(x) \).
4. Perform the splitting: \( [X^k, S, X^{k+1}] = \text{svd} \left( \begin{bmatrix} W^k(1, 1) & \cdots & W^k(1, n) \\ W^k(n, 1) & \cdots & W^k(n, n) \end{bmatrix}, \varepsilon, R \right) \).
5. if \( k \) is increasing then
6. Leave \( X^k \) left-orthogonal, and \( X^{k+1}(i_{k+1}) := SX^{k+1}(i_{k+1}) \).
7. else
8. Leave \( X^{k+1} \) right-orthogonal, and \( X^k(i_k) := X^k(id)S \).
9. end if
10. end for

- If \( L > 1 \), instead of the full representation of the extended factor (and hence, optimization) we employ the TT-DMRG (MALS) algorithm 4 to optimize the TT representation \( X^{(c)}kL(i_{k,L}) \cdots X^{(c)}k1(i_{k,1})X^{(d)}k(\alpha_{k-1} \alpha_k) \).

- Since each factor optimization does not involve the whole core, we have to add the core optimization step (also via TT-DMRG method) explicitly.

Now, we can write the QTT-Tucker optimization algorithm 5. In the same way, as the QTT-Tucker rounding mimics the HOSVD, the QTT-Tucker DMRG algorithm implicitly performs the TALS. Note the similarity with the rounding algorithm 1. If \( L = 1 \), and

Algorithm 4: DMRG_{TT} iteration

1: for \( k = 1, \ldots, d \) do
2: Orthogonalize the extended factor \( X^{(f)}kL(i_{k,L}) \cdots X^{(f)}k1(i_{k,1})X^{(c)}k(\alpha_{k-1} \alpha_k) \) from the left to the right. {Complexity \( O(dLr^2) \)}
3: end for
4: Optimize the core \( X^{(c)} = \text{DMRG}_{TT}(J, X^{(c)}, \varepsilon, R) \), using the TT-DMRG 4 or ALS 3 procedures, so that \( X^{(c)} \) is right-orthogonal. {Complexity \( O(dr^2) \)}
5: for \( k = 1, \ldots, d - 1 \) do
6: Optimize the extended factor: {Complexity \( O(dLr^2) \)}
   \[
   X^{(f)}k(i_k)\tilde{X}^{(c)}k(\alpha_{k-1} \alpha_k) = \text{DMRG}_{TT}(J, X^{(f)}k(i_k)X^{(c)}k(\alpha_{k-1} \alpha_k), \varepsilon, R) 
   \]
   so that the factor is left-orthogonal.
7: Make the core block \( \tilde{X}^{(c)}k \) left-orthogonal: {Complexity \( O(dr^2) \)}
   \[
   \begin{bmatrix} X^{(c)}k(y_k = 1) & \cdots & X^{(c)}k(r_{ck}) \end{bmatrix}^T, R = \text{qr} \left( \begin{bmatrix} X^{(c)}k(y_k = 1) & \cdots & \tilde{X}^{(c)}k(r_{ck}) \end{bmatrix}^T \right) . \]
8: Convolve the matrix \( R \) to the next block: \( X^{(c)}k+1(y_{k+1}) := R X^{(c)}k+1(y_{k+1}) \).
9: end for
10: Optimize the last factor {Complexity \( O(Lr^2) \)}
   \[
   X^{(f)}d(i_d)X^{(c)}d(\alpha_{d-1}) = \text{DMRG}_{TT}(J, X^{(f)}d(i_d)X^{(c)}d(\alpha_{d-1}), \varepsilon, R) .
   \]

the core is stored in the full format, the method reduces to the simple Tucker ALS. If
we redefine \( r_1 := n, \ n := 1 \), the format reduces to the TT, and the only line 4 left in Algorithm 5 results in the TT-DMRG.

6.2. Some technical details on the QTT-Tucker DMRG method

An important operation, which is absent in Algorithm 5, is the reduction of the functional \( J(x) \) to the certain part of the format (extended factor, or core). For the TT case, it was shown \([39, 42]\), that it is equivalent to the projection of the functional gradient via the fixed TT blocks, and moreover, this projection is orthogonal, if the proper orthogonality constraints are imposed on fixed blocks. For the QTT-Tucker case the situation is the same, but we need to describe technical differences.

We consider in this section only the energy functional minimization. Its optimal condition is written as \( Ax = y \), as well as for error/residual functionals.

The first operation is the core optimization step 4. We need to prepare the core matrix in the TT format, i.e., the projection of the full QTT-Tucker MatVec \( Ax \) and right-hand side \( y \) on factors. The TT blocks of the projected matrix \( A^{(cp)k}_{\alpha_{k-1}^{A}, \alpha_{k}^{A}}(\gamma_{k}, \gamma_{k}^{x}) \) and r.h.s. \( Y^{(cp)k}_{\alpha_{k-1}^{U}, \alpha_{k}^{U}}(\tilde{Y}^{x}) \) are then written, respectively,

\begin{align*}
\tilde{\gamma}_{k}^{x} & \rightarrow \frac{\alpha_{k-1}^{A}}{\alpha_{k}^{A}} A^{(c)k}_{\alpha_{k}^{A}} \tilde{\gamma}_{k}^{x}, \\
\tilde{\gamma}_{k,1}^{x} & \rightarrow \frac{\alpha_{k-1}^{U}}{\alpha_{k}^{U}} V^{(c)k}_{\alpha_{k}^{U}} \tilde{\gamma}_{k,1}^{x},
\end{align*}

The linear system on the core then reads

\[ \sum_{\gamma_{1}^{x}, ..., \gamma_{d}^{x}} A^{(cp)}(\tilde{Y}_{1}^{x}, ..., \tilde{Y}_{d}^{x}) X^{(c)}(\gamma_{1}^{x}, ..., \gamma_{d}^{x}) = Y^{(cp)}(\tilde{Y}^{x}) \]

We see, that the projection is performed, by convolving the orthogonal factor blocks \( X^{(c)k} \) over the indices \( i_{k,p} \) with the parts of the linear system \( Ax \) and \( y \), and the Tucker ranks play the role of the mode sizes for the optimization over core. Note, that if \( A = I \) (the simple approximation problem is posed), the matrix projection is also identity due to the orthogonality, and we can skip the first equation in (9).

As soon as \( A^{(cp)} \) and \( Y^{(cp)} \) are computed, they can be easily used to reduce the problem to extended factors. Indeed, the step 6 of Algorithm 5 means, that all the factors, and all the core blocks except \( k \)-th are fixed. Thus, all the blocks of \( A^{(cp)} \) and \( Y^{(cp)} \) except \( k \)-th contain proper projections with factors. Now, we only need to perform their projection on all \( X^{(c)} \) blocks except \( k \)-th, in the same way, as it was done in the TT-ALS and DMRG routines \([39, 42]\). After that, we will have the matrix and right-hand side blocks for computing the last block of the extended factor \( X^{(c)k}_{\alpha_{k-1}^{A} \alpha_{k}^{A}} \).
6.3. Discussion on convergence and complexity

Despite the lack of theoretical convergence analysis of the ALS algorithms, in most numerical experiments they perform quite well. Sometimes, several technical tricks have to be added [42, 51], which are reasonable in certain cases. Initially, the DMRG algorithm was applied to the MPS format both in quantum chemistry and later numerical analysis community. Recently, the HT variant was developed [46], and here the QTT-Tucker one, all of them give satisfactory results for considered problems.

The complexity issues remain open even in practice. One of them, is the difference between the ALS and DMRG approach. In the ALS algorithm, we have to specify (usually, overestimated) ranks a priori, so the first iterations are as difficult as the last one (or even more, if the linear systems are ill-conditioned), and the convergence might be slow. The DMRG algorithm allows to start with a low-rank initial guess (first iterations are cheap), and, when the ranks become larger, we already have some reasonable solution approximation. Thus, in terms of iterations, the DMRG approach works better.

However, this consideration excludes the complexity of working with merged two blocks instead one, with the storage $O(r^2n^2)$ instead of $O(r^2n)$. In the QTT format, it increase the size of the local problem twice, which is not very restrictive. If we are to use larger mode sizes (even in the QTT-Tucker, for example, the core blocks have mode sizes equal to the Tucker ranks; as for the spectral discretizations, the quantization is inefficient, since the QTT rank of spectral matrices equals $n$), the overhead is much more significant. The truncation step involves SVD with even higher complexity $O(r^3n^3)$. Thus, the development of improved adaptive one-block techniques is planned for the future research.

7. Numerical examples

In this section we compare the standard Quantized-TT approximation, when the whole tensor is reshaped to $2 \times 2 \times \cdots \times 2$ and approximated via TT (called further as linear QTT), with the QTT-Tucker format, the Tucker splitting of the physical dimensions, and the QTT of each factor. The asymptotic storage complexity $O(d\log_2(n)r^2)$ of the linear QTT format is smaller, than $O(d\log_2(n)r^2 + dr^3)$ of the QTT-Tucker, but the ranks of the latter are usually much milder. Obviously, if we do not have the case of the exact low-rank tensors, the ranks should grow with the accuracy. In the following, we investigate the respective behavior of these formats in the approximation and the linear system problems.

7.1. Poisson equation

As the first example, consider the solution of a 5D Poisson equation

$$-\Delta u = f = 1 \quad \text{in } \Omega = [0, 1]^5,$$

$$u_{\partial\Omega} = 0,$$

discretized using the Finite Difference scheme on $n = 256$ grid points. To solve the tensor linear system, we use the DMRG TT-solver [42], Alg. 4 for the linear QTT, and also the DMRG-like method for the QTT-Tucker [5].
We compare both the storage complexity (maximal rank and the total memory cells used to keep the representation) and the solution timings and residuals versus the truncation accuracy $\varepsilon$ in Frobenius norm, see Figures 2-5.

The maximal rank is smaller in the QTT-Tucker format, but the memory plots have an intersection at $\varepsilon^* \sim 10^{-3}$. The very impressive thing is the “reversed” behavior of the memory plots: due to higher asymptotic complexity of the QTT-Tucker, we might expect it to be less efficient for smaller tolerances and hence larger ranks. On the contrary, the ranks of the linear QTT grow so fast, that the QTT-Tucker outperforms the linear QTT for high accuracies. If the ranks are almost the same (lower accuracies), the work and storage complexities of the linear QTT are better.

One may note, that for both formats the memory requirements are still mild: we need only $10^4 - 10^5$ cells to approximate the tensor with $256^5 \sim 10^{12}$ elements with single precision.

Now, let us consider the solution times and the residual (Fig. 4, 5). Here again,
as for the memory, the CPU times have an “reversed” intersection, but at the larger accuracy $\varepsilon^* \sim 5 \cdot 10^{-5}$. We recall here, that the solution complexity is higher than storage: the DMRG method for the linear QTT requires $O(d \log_2(n)r^3)$ operations, and for the QTT-Tucker $O(d \log_2(n)r^3 + dr_c^6)$ (however, $r_c$ is smaller than $r_l$). So, the “practical” complexity is much more optimistic than the theoretical.

It is worth to check the residuals, since it is the only available measure of the “correctness” of the solution. From Fig. 5 we see, that the methods manifest almost equal convergence, and the QTT-Tucker is even slightly better.

**Remark 5.** One might see, that the relative residual is much larger than the relative rounding accuracy, due to significantly large constants of equivalence ($c_1$, $c_2$),

$$\frac{1}{c_1} \frac{||A(x - y)||}{||Ax||} \leq \frac{||x - y||}{||x||} \leq c_2 \frac{||A(x - y)||}{||Ax||},$$

depending on the condition number of the matrix.

### 7.2. Generalized Gaussian function - solution of high dimensional Fokker-Planck equation

One interesting application of tensor methods is the Fokker-Planck equation, which is usually high-dimensional. It models the joint probability density distribution of noisy dynamical system configurations (e.g. positions of particles). The (stochastic) system ODE reads

$$\frac{dx}{dt} = -A(x) + G\eta \in \mathbb{R}^d,$$

where $<\eta> = 0$, $<\eta_i\eta_j> = \delta_{ij}$. The probability to find the configurations in some volume $x^* + dx$ is written as follows:

$$P \left( x \in B_{dx}(x^*) \right) = \psi(x^*) dx,$$

and the deterministic PDE on the probability density is the Fokker-Planck equation:

$$\frac{\partial \psi}{\partial t} = \frac{\partial}{\partial x} \cdot (A(x)\psi) + \frac{1}{2} \frac{\partial}{\partial x} \left( D \frac{\partial \psi}{\partial x} \right), \quad D = GG^T.$$

It was proven for example in [52], that for a linear system $A(x) = Ax$, the (unnormalized) steady probability density is given by the generalized Gaussian

$$\psi|_{t \to \infty} = \exp(-x^TBx),$$

(10)

where $B$ is the solution of the following Lyapunov equation:

$$AB^{-1} + B^{-1}A^T = -2D = -2GG^T.$$

So, we would like at least to compute and store the generalized Gaussians in a tensor format.
7.2.1. 12D example. In this test, we consider the 3D Hookean model for a polymer chain \([53, 54, 55, 56, 57]\) with 4 springs in the shear flow regime, which yields us the 12-dimensional problem, with the matrix

\[
A = I_{4 \times 4} \otimes \begin{bmatrix} 0 & \beta & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{2} D, \quad D = \frac{1}{2} \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \otimes I_{3 \times 3}, \quad \beta = 0.2.
\]

This problem was also considered in \([47]\), with attempt to solve the corresponding Fokker-Planck equation in the linear QTT format. It was shown, that the ranks grow linearly with the shear flow rate \(\beta\), and to get a satisfactory approximation of the stationary solution requires several hours of CPU time. The exponential argument \(V = x^T B x\) admits an exact low-rank TT decomposition, see Thm. \([\text{I}]\). To construct the stationary solution \((10)\), we use the scaling and squaring method \([58]\) to compute the pointwise exponential in the QTT or QTT-Tucker formats.

On figures \([6, 7]\) we show respectively the memory cells required to store the FPE solution, and the CPU time of the scaling and squaring method versus the Frobenius rounding accuracy in different formats and methods. We compare the linear QTT, QTT-Tucker (with mode size 2), and also the Extended TT (ETT) formats \((n = 256, L = 1)\). The most consuming operation is the squaring of the approximant, and there are two approaches: exact Hadamard product + rounding (round), and the DMRG-based approximation of the result (dmrg). We failed to proceed with the former approach for high accuracies due to the memory limitations: the ranks of the product are squared, thus we need \(O(dLn r_t^4 + dr_t^6)\) storage, and the rounding complexity is \(O(dLn r_t^6 + dr_t^8)\). By the same reasons, we do not consider the TT format with \(n = 256\). The DMRG complexity is \(O(dLn^2 r_t^4 + dr_t^6)\), and the storage is \(O(dLn^2 r_t^6 + dr_t^8)\). It is also interesting, that the QTT-Tucker and ETT formats use asymptotically the same amount of memory (which shows that the largest rank is in the Tucker core), but the \(n^2\)-overhead makes the DMRG procedure faster for the Q-format.
In this example, the complexity of the QTT-Tucker is more than 10 times smaller than of the QTT format for 6-7 digits of accuracy. It shows the advantages of the new format in solution of high-dimensional problems.

8. Conclusion

A combined tensor representation was considered. It encapsulates the benefits of the Tucker, TT and QTT formats. As its predecessors, the new format is a closed manifold. All basic algebraic operations, as well as rounding and alternating optimization methods are available and stable. Despite the worse asymptotic storage and computation complexity, it usually performs better than the “standard” linear-structured QTT, due to much lower ranks required to fix the same accuracy. Moreover, they are distributed more uniformly (similarly to the Tucker ranks), on the contrary to the linear QTT, which normally keeps a “peak” rank at the middle of the tensor train.

From that point of view, the QTT-Tucker format can be considered as a “rank equalizing” structure. In terms of TT, an attempt to implement such an idea resulted in the so-called Tensor Chain [30] format, which is the TT with the summation over the connected first and the last ranks not equal to 1. Such representation was expected to manifest uniform ranks distribution, since none of them are “boundary” or “middle”. Unfortunately, the TC manifold is not closed, and optimization problems are ill-posed. The QTT-Tucker format reduces the peak rank values by making the effective “length” of the involved tensor trains smaller ($d$ or $L$ versus $dL$), but without loosing the stability. Moreover, any TC representation can be easily converted to the QTT-Tucker format.

However, the rank overhead of the DMRG optimization algorithms can be significantly worse in the Tucker-based formats rather than the TT, since the effective mode sizes are now $r$ but not 2, and DMRG techniques require merging of two blocks. To achieve the same rank complexity as in the rounding procedure, robust one-block alternating methods have to be developed. This is the aim for the future work.

References


First of all, we compute the off-diagonal skeleton decompositions

\[
\begin{align*}
C^k &= W^k U^k, \quad W^k \in \mathbb{R}^{d-k \times r_k}, \quad U^k \in \mathbb{R}^{r_k \times k}, \\
\hat{C}^k &= \hat{U}^T \hat{W}^T, \quad \hat{W}^k \in \mathbb{R}^{d-k \times \hat{r}_k}, \quad \hat{U}^k \in \mathbb{R}^{\hat{r}_k \times k},
\end{align*}
\]

with orthonormal \(U^k\), \(\hat{U}^k\). Now, we separate the TT blocks recursively.

On the first step we have

\[
V = B_{11} X_1^2 + \sum_{i=2}^d B_{1i} X_i' \otimes X_1 + \sum_{j=2}^d B_{i1} X_i' \otimes X_j + V^{[2]},
\]

where

\[
V^{[k]} = \sum_{i,j=k}^d B_{ij} X_i' \otimes X_j.
\]

So,

\[
V = \left[ B_{11} X_1^2 \quad X_1' \quad x_1 \quad e_1 \right] \left[ E^{[2]} \quad \sum_{j=2}^d \hat{B}_{1j} X_j^{[2]} \quad \sum_{i=2}^d B_{i1} X_i^{[2]} \quad V^{[2]} \right]^T,
\]

where \(E^{[k]} = e_K \otimes \cdots \otimes e_{d'} X_i^{[k]} = e_k \otimes x_j \otimes e_{j+1} \otimes \cdots \otimes e_d\). The first term in (11) is the first TT block of \(V\). On the other hand, we can represent it using the skeleton factors for the first row and column:

\[
V = \left[ B_{11} X_1^2 \quad \hat{U}^1 X_1' \quad \hat{U}^1 x_1 \quad e_1 \right] \left[ E^{[2]} \quad \sum_{j=2}^d \hat{W}_{1j} X_j^{[2]} \quad \sum_{i=2}^d W_{i1} X_i^{[2]} \quad V^{[2]} \right]^T.
\]

Now, we need to derive the recurrent representation for the second term.
Applying the first step to $V[k]$, obtain

$$
\tilde{V} = \left[ E[k] \sum_{j=1}^{d} \hat{X}_j^{[k]} W_{j,k}^{k-1} \cdots \sum_{j=1}^{d} X_j^{[k]} \hat{W}_{j,k}^{k-1} \sum_{i=1}^{d} X_i^{[k]} W_{i,k}^{k-1} \cdots \sum_{i=1}^{d} X_i^{[k]} \hat{W}_{i,k}^{k-1} V[k] \right]^T. \tag{12}
$$

Suppose we have

$$
\tilde{V} = \begin{bmatrix}
    x_k \hat{W}_{k,k}^{k-1} & x_k'U_{k,k} & \cdots & x_k'U_{k,1} & e_k
\end{bmatrix}
\begin{bmatrix}
    \sum_{j=1}^{d} X_j^{[k]} \hat{W}_{j,k}^{k-1} & \cdots & \sum_{j=1}^{d} X_j^{[k]} W_{j,k}^{k-1} \cdots & \sum_{j=1}^{d} X_j^{[k]} \hat{W}_{j,k}^{k-1} & \sum_{i=1}^{d} X_i^{[k]} W_{i,k}^{k-1} \cdots \sum_{i=1}^{d} X_i^{[k]} \hat{W}_{i,k}^{k-1} V[k]
\end{bmatrix}
= \begin{bmatrix}
    E[k+1] & \cdots & e_k
\end{bmatrix}, \quad I_{r_{k-1}} = \begin{bmatrix}
    e_k
\end{bmatrix}
$$

or, separating the scalar coefficients from $X$-related data in the last column,

$$
\tilde{V} = \begin{bmatrix}
    x_k \hat{W}_{k,k}^{k-1} & x_k'U_{k,k} & \cdots & x_k'U_{k,1} & e_k
\end{bmatrix}
\begin{bmatrix}
    \hat{W}_{k,k+1:d,k} & \cdots & \hat{W}_{k,k+1:d,d} & \cdots & \hat{W}_{k,k+1:d,1}
\end{bmatrix}
\begin{bmatrix}
    E[k+1] & \cdots & e_k
\end{bmatrix}, \quad I_{r_{k-1}} = \begin{bmatrix}
    e_k
\end{bmatrix}
$$

For brevity, denote the second matrix in this statement $R_k$. Now, recall that $\hat{W}_{k-1}^{k-1} = \hat{U}_{k-1}^{k-1} B_{1:k-1,k+1:d}$, $W_{k-1}^{k-1} = B_{k+1:d,k+1:k-1} \hat{U}_{k-1}^{k-1}^T$:

$$
R_k = \begin{bmatrix}
    1 & \hat{U}_{k-1}^{k-1} & \hat{U}_{k-1}^{k-1} & 1 & 1
\end{bmatrix}
\begin{bmatrix}
    B_{1:k-1,k+1:d} & B_{k+1:d,k+1:k-1} & B_{k+1:d,k+1:k-1} & B_{k+1:d,k+1:k-1}
\end{bmatrix}
$$

For the second term we use the $k$-th skeleton decomposition:

$$
R_k = \begin{bmatrix}
    1 & \hat{U}_{k-1}^{k-1} & \hat{U}_{k-1}^{k-1} & 1 & 1
\end{bmatrix}
\begin{bmatrix}
    \hat{U}_{k-1}^{k-1} & \hat{U}_{k-1}^{k-1} & \hat{U}_{k-1}^{k-1} & \hat{U}_{k-1}^{k-1} & \hat{U}_{k-1}^{k-1} & \hat{U}_{k-1}^{k-1}
\end{bmatrix}
\begin{bmatrix}
    1 & \hat{W}_{k,k+1:d,k} & \cdots & \hat{W}_{k,k+1:d,k} & \cdots & \hat{W}_{k,k+1:d,k}
\end{bmatrix}
\begin{bmatrix}
    E[k+1] & \cdots & e_k
\end{bmatrix}
$$

The last term contains only $W$-coefficients, i.e. yields the form of $\tilde{V}$ (12). Multiplying
the first two terms, and plugging into (13), we get

\[
\hat{V} = \begin{bmatrix}
    e_k & 0_{k^2} & 0_{k^2} & \ldots & 0_{k^2} \\
    x_k \hat{W}_{k^2} & k^2 & \ldots & \ldots & \ldots \\
    x_k' \hat{W}_k & \ldots & \ldots & \ldots & \ldots \\
    B_{kk} x_k^2 & \ldots & \ldots & \ldots & \ldots \\
\end{bmatrix}
\begin{bmatrix}
    e_k \\
    x_k \\
    x_k' \\
    B_{kk} x_k^2 \\
\end{bmatrix}
\begin{bmatrix}
    e_k U^{k-1} U_{1:k,1:k-1}^T \\
    x_k U_{1:k,1:k-1}^T \\
    x_k' U_{1:k,1:k-1}^T \\
    B_{kk} x_k^2 \\
\end{bmatrix}
\begin{bmatrix}
    E^{[k+1]} \\
    \sum_{j=k+1}^d X_j^{[k+1]} W_{1:j,1:k}^T \\
    \sum_{j=k+1}^d \chi_j^{[k+1]} W_{1:j,1:k}^T \\
    V^{[k+1]} \\
\end{bmatrix}
\].

(14)

The first term is nothing else than the k-th TT block of \( V \), since it contains only \( x_k, e_k \). It has the sizes \((1 + \hat{r}_{k-1} + r_{k-1} + 1) \times (1 + \hat{r}_k + r_k + 1)\), which confirms the first statement of the theorem. With the second term we can proceed by recursion, and the last TT block reads (since \( V^{[d]} \) is just a one-dimensional \( B_{dd} x_d^2 \))

\[
V_d = \begin{bmatrix}
    e_d & x_d' \hat{W}_{d,1:d-1} & x_d' W_{d,1:d-1} & B_{dd} x_d^2 \\
\end{bmatrix}^T.
\]

(15)

If the first-order term is presented with unique object \( x_k = x_k' \), the ranks are reduced as follows. First,

\[
V_1 = [B_{11} x_1^2 \ 2 x_1 \ 1] \begin{bmatrix}
    1 \\
    0.5 \\
    0.5 \\
\end{bmatrix},
\]

Reassigning \( \hat{V}_1 = [B_{11} x_1^2 \ 2 x_1 \ 1] \), we convolve the rest matrix with the middle blocks, obtaining

\[
\hat{V}_k = \begin{bmatrix}
    e_k & 0_{k^2} & 0_{k^2} & \ldots & 0_{k^2} \\
    x_k \hat{W}_{k^2} & k^2 & \ldots & \ldots & \ldots \\
    x_k' \hat{W}_k & \ldots & \ldots & \ldots & \ldots \\
    B_{kk} x_k^2 & \ldots & \ldots & \ldots & \ldots \\
\end{bmatrix}
\begin{bmatrix}
    e_k \\
    x_k \\
    x_k' \\
    B_{kk} x_k^2 \\
\end{bmatrix}
\begin{bmatrix}
    e_k U^{k-1} U_{1:k,1:k-1}^T \\
    x_k U_{1:k,1:k-1}^T \\
    x_k' U_{1:k,1:k-1}^T \\
    B_{kk} x_k^2 \\
\end{bmatrix}
\begin{bmatrix}
    E^{[k+1]} \\
    \sum_{j=k+1}^d X_j^{[k+1]} W_{1:j,1:k}^T \\
    \sum_{j=k+1}^d \chi_j^{[k+1]} W_{1:j,1:k}^T \\
    V^{[k+1]} \\
\end{bmatrix}
\].

We note here, that the constraint \( x_k = x_k' \) yields \( V \equiv 0 \) if \( B = -B^T \), so that only the symmetric part \( B := 0.5 (B + B^T) \) is relevant. Thus, \( \hat{W}^k = W^k, \hat{U}^k = U^k \). The rest TT blocks are simplified as follows:

\[
\begin{bmatrix}
    E^{[k+1]} \\
    \sum_{j=k+1}^d X_j^{[k+1]} W_{1:j,1:k}^T \\
    \sum_{j=k+1}^d \chi_j^{[k+1]} W_{1:j,1:k}^T \\
    V^{[k+1]} \\
\end{bmatrix}
= \begin{bmatrix}
    1 & 0_{d-1} \\
    1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
    E^{[k+1]} \\
    \sum_{j=k+1}^d X_j^{[k+1]} W_{1:j,1:k}^T \\
    \sum_{j=k+1}^d \chi_j^{[k+1]} W_{1:j,1:k}^T \\
    V^{[k+1]} \\
\end{bmatrix}
\].

Convolving the first matrix from the right with \( \hat{V}_k \), obtain the reduced block

\[
\hat{V}_k = \begin{bmatrix}
    e_k & 0_{k^2} & 0_{k^2} & \ldots & 0_{k^2} \\
    x_k \hat{W}_{k^2} & k^2 & \ldots & \ldots & \ldots \\
    x_k' \hat{W}_k & \ldots & \ldots & \ldots & \ldots \\
    B_{kk} x_k^2 & \ldots & \ldots & \ldots & \ldots \\
\end{bmatrix}
\begin{bmatrix}
    e_k \\
    x_k \\
    x_k' \\
    B_{kk} x_k^2 \\
\end{bmatrix}
\begin{bmatrix}
    e_k U^{k-1} U_{1:k,1:k-1}^T \\
    x_k U_{1:k,1:k-1}^T \\
    x_k' U_{1:k,1:k-1}^T \\
    B_{kk} x_k^2 \\
\end{bmatrix}
\begin{bmatrix}
    E^{[k+1]} \\
    \sum_{j=k+1}^d X_j^{[k+1]} W_{1:j,1:k}^T \\
    \sum_{j=k+1}^d \chi_j^{[k+1]} W_{1:j,1:k}^T \\
    V^{[k+1]} \\
\end{bmatrix}
\]

of size \((r_{k-1} + 2) \times (r_k + 2)\).

Remark 6. The proof gives a constructive routine for fast assembly of a quadratic form in TT format. Indeed, performing the SVDs of submatrices of \( B \) (which are in the order of tens) and building the blocks from (11), (14) (or (16) in symmetric case), (15) we get the analytical TT representation.