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

Sergey Dolgov, Boris Khoromskij, Dmitry Savostyanov

Multidimensional Fourier transform in logarithmic complexity using QTT approximation

Preprint 2011-01 28-04-2011

Moscow 2011

Multidimensional Fourier transform in logarithmic complexity using QTT approximation

Sergey Dolgov, Boris Khoromskij, Dmitry Savostyanov

Institute of Numerical Mathematics, Russian Academy of Sciences,
Russia, 119333 Moscow, Gubkina 8
[sergey.v.dolgov,dmitry.savostyanov]@gmail.com

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

April 28, 2011

Abstract. We propose algorithms for the Fourier transform and related sine/cosine transforms, using QTT format for data-sparse approximate representation of one– and multi–dimensional vectors (m-tensors). Although a Fourier matrix itself does not possess a low-rank QTT representation, it can be efficiently applied to m–tensor in the QTT format exploiting the multilevel structure of the Cooley-Tuckey algorithm. The m–dimensional Fourier transform of an $n \times \ldots \times n$ vector with $n = 2^d$ has $\log^2(\text{volume}) = \mathcal{O}(m^2 d^2 r^3)$ complexity, where r is the maximum QTT rank of input, output and all intermediate vectors during the procedure. The storage size in QTT format is bounded by $\mathcal{O}(mdr^2)$. For vectors with moderate r and large n and m the proposed algorithm outperforms the $\mathcal{O}(n^m \log n)$ FFT algorithm of almost volume complexity. The numerical experiments show that the use of QTT format relaxes the grid size constrains and allow high-resolution computations of Fourier images and convolutions in high dimensions without "curse of dimensionality". In the case of m-dimensional signals with small number of frequencies the proposed approach is competitive with a Sparse Fourier transform, even for moderate m.

Keywords: Multidimensional arrays, QTT, FFT, sine transform, cosine transform, convolution, sparse Fourier transform
AMS classification: 15A23, 15A69, 65T50, 65F99

$This work was supported by RFBR grants 09-01-12058, 10-01-00757, 11-01-00549, RFBR/DFG grant 09-01-91332, Russian Federation Gov. contracts No. П1112 and П940, 14.740.11.0345 and Promotion-stipendium of Max-Planck Gesellschaft. Part of this work was done during the stay of S. Dolgov and D. Savostyanov in Max-Plank Institute for Mathematics in Sciences, Leipzig, Germany.

1. Introduction

For high-dimensional problems, data-sparse representation schemes allow to overcome the so-called *curse of dimensionality* and perform computations efficiently. Among many low-parametric formats, *tensor decomposition* methods appear to be the most promising for high-dimension data [1, 2, 3]. Recently proposed *tensor train* (TT) format [4, 5] combines advances of both the canonical [6, 7, 8] and Tucker [9] formats: the number of representing parameters grows only linearly with dimension and the approximation problem is stable and can be computed by SVD-based algorithms. For low-dimensional data, the tensor train format can be applied by substitution of the dimensions with large mode sizes by a larger number of dimensions with small (say 2) mode size, resulting in a high-dimensional array. Following [10] we call such a representation a *QTT format*. For data in the QTT format, basic linear algebra procedures like summation, multiplication and rank truncation (tensor rounding) are implemented maintaining the compressed format, i.e. the full data array is never computed.

In this paper we present algorithms for trigonometric transforms (discrete Fourier, sine and cosine transform) of one– and high–dimensional data represented (approximated) in the QTT format. For fixed QTT structure, the complexity m-dimensional $n \times \ldots \times n$ transform is bounded by $\mathcal{O}(m^2 \log^2 n)$ and has *square logarithmical* scaling w.r.t. volume n^m of an array. Our approach bases on a radix-2 recurrence that links full and half-sized Fourier transforms and lays behind a widely used Cooley-Tuckey FFT algorithm [11, 12], that has $\mathcal{O}(n^m \log n)$ complexity. Each level of the proposed Cooley-Tuckey-like algorithm includes a rank truncation approximation step to optimize QTT ranks of intermediate vectors adaptive to prescribed level of accuracy. Similar method applies for sine and cosine transforms of vectors in the QTT format, using the inherit multilevel structure of this representation.

For a *tensor*, i.e. array with many indices $\mathbf{A} = [a(j_1, \ldots, j_d)]$, the TT format reads

$$a(j_1, \ldots, j_d) = A^{(1)}_{j_1} \ldots A^{(d)}_{j_d}, \tag{1}$$

where each $A^{(p)}_{j_p}$ is an $r_{p-1} \times r_p$ matrix. Usually $r_0 = r_d = 1$ and every entry $a(j_1, \ldots, j_d)$ is a scalar. However, larger r_0 and r_d can be considered and every entry of a tensor $\mathbf{A} = [a(j_1, \ldots, j_d)]$ becomes an $r_0 \times r_d$ matrix.

This format is known in quantum chemistry as matrix product states (MPS) [13]. It was introduced to scientific computing community with a purely algebraic SVD-based decomposition/approximation algorithm [14, 5]. For given $\mathbf{A} = [a(j_1, \ldots, j_d)]$ it computes approximation $\tilde{\mathbf{A}} = [\tilde{a}(j_1, \ldots, j_d)]$ with a prescribed accuracy ε

$$a(j_1, \ldots, j_d) \approx \tilde{a}(j_1, \ldots, j_d) = A^{(1)}_{j_1} \ldots A^{(d)}_{j_d}, \qquad \|\mathbf{A} - \tilde{\mathbf{A}}\|_F \leqslant \varepsilon \|\mathbf{A}\|_F,$$

where the Frobenius norm of a tensor is defined by $\|\mathbf{A}\|_F^2 \overset{\text{def}}{=} \sum_{j_1 \ldots j_d} |a(j_1, \ldots, j_d)|^2$. The TT ranks $r_1, \ldots, r_{d-1}$ depend on the chosen ε and intrinsic *structure properties* of a tensor $\mathbf{A}$.

To apply the TT compression to low dimensional data, the idea of *quantization* was proposed. We illustrate it for one-dimensional vector $a = [a(j)]_{j=0}^{n-1}$, restricting the discussion to $n = 2^d$. First, define binary code of index j by

$$j = \overline{j_1 \dots j_d} \overset{\text{def}}{=} \sum_{p=1}^{d} j_p 2^{p-1}, \qquad j_p = 0, 1, \tag{2}$$

and note that it is an isomorphic mapping $j \leftrightarrow (j_1, \dots j_d)$, that allows to *reshape* a vector $[a(j)]$ into d–tensor $[a(j_1, \dots, j_d)]$. The TT format (1) for the latter is called a *QTT format* and reads

$$a(j) = a(\overline{j_1 \dots j_d}) = A^{(1)}_{j_1} \dots A^{(d)}_{j_d}.$$

This idea appears in [15] in the context of matrix approximation. In [10] it was given the name and applied for some univariate functions with small *quantics*-TT (QTT) ranks, that were established theoretically ($\exp x$, $\sin x$, $\cos x$, x^p) or experimentally (x^α, $e^{-\alpha x^2}$, $\frac{\sin x}{x}$, $\frac{1}{x}$ et al).

An analytical rank-one QTT representation of one-dimensional exponential function discretised on the uniform grid,

$$\exp(\alpha j) = \exp(\alpha\, \overline{j_1 \dots j_d}) = \exp(\alpha j_1) \exp(2\alpha j_2) \exp(2^2 \alpha j_3) \dots \exp(2^{d-1} \alpha j_d), \tag{3}$$

is a key tool for the efficient Fourier transform algorithm maintaining the QTT format.

Recent development of the QTT format includes the theoretical estimates for the ranks of the *tensorisations* [10, 16], constructive representation for function-related vectors [17], explicit representation for the inverse Laplacian and related matrices [18] and some links with the hierarchical wavelet transform [19]. The QTT format was already applied to the solution of several high-dimensional problems [20, 21, 22] as well as H–Tucker format [23]. It is worth to note that the efficient QTT representation of the Fourier transform appears to be a nontrivial problem since the Fourier matrix has full QTT ε–ranks (see Section 2.1).

This paper is organized as follows. In Section 2 we present the Fourier transform algorithm for vectors in the QTT format. In Section 3 we explain how to keep the QTT ranks moderate during this procedure. In Section 4 we develop the multi-dimensional Fourier transform algorithm. In Section 5 we give numerical examples illustrating that the use of QTT format relaxes the grid size constrains and allow high-resolution computations of Fourier images in high dimensions without "curse of dimensionality". In particular, due to $\mathcal{O}(m^2 \log^2 n)$ scaling, our approach allows to compute one– and multi–dimensional Fourier images using $n = 2^{60}$ in 1D and $n = 2^{20}$ in 3D on a standard workstation, see Sec. 5.2 and Sec. 5.3. Fourier transform can be also applied for m–dimensional convolution. With our approach we compute the convolution transforms of data with strong cusps or singularities that requires very fine grid, see Sec. 5.4. Such data occur in particular in quantum chemistry computations [24, 25, 26, 27]. Note that the "curse of dimensionality" can be also relaxed using the fast Fourier transform on hyperbolic cross points [28]. Finally, we apply the proposed method to m-dimensional signals with small number of frequencies and show that it is competitive with the Sparse Fourier transform [29], see Sec. 5.5.2. In Appendix A we propose sine and cosine transforms in the QTT format using either purely real-value arithmetics or real-to-complex and complex-to-real transforms in the QTT format.

2. Discrete Fourier transform in one dimension

For $n = 2^d$, the discrete Fourier transform (DFT) reads

$$y(j) = \sum_{k=0}^{2^d-1} x(k)\omega_d^{jk}, \qquad \omega_d = \exp\left(-\frac{2\pi i}{2^d}\right), \quad i^2 = -1, \tag{4}$$

where $F_d = [\omega_d^{jk}]_{j,k=0}^{2^d-1}$ is the Fourier matrix.

2.1. QTT decomposition of the Fourier matrix has full ranks

Fast linear transforms in the QTT format are often based on explicit low-rank QTT representation of a transformation matrix, cf. [15, 18]. For the Fourier matrix, however, this is not the case.

The QTT format for a matrix $A = [a(j,k)]_{j,k=1}^{2^d}$ (defined as *TTM format* in [15]) formally appears as the TT format (1) after the *quantization* of indices and *reshape* of tensor elements

$$a(i,j) = a(\overline{j_1 \dots j_d}, \overline{k_1 \dots k_d}) = \dot{a}(j_1k_1, j_2k_2, \dots, j_dk_d) = A^{(1)}_{j_1k_1} A^{(2)}_{j_2k_2} \dots A^{(d)}_{j_dk_d}. \tag{5}$$

Here, the d-tensor $\dot{\mathbf{A}}$ contains elements of A in the reshaped order and matrices $A^{(p)}_{j_pk_p}$ have size $r_{p-1} \times r_p$ where r_p are the QTT ranks. The following theorem proves that the QTT representation of the Fourier matrix has *full* ranks.

Theorem 1. QTT decomposition (5) of the Fourier matrix F_d has ranks $r_k = 2^{\min(k,d-k)}$.

Proof. The TT ranks r_p are closely related to the ranks of *unfolding* or *matricisation* of a given tensor. Define bi-index $i_p \overset{\text{def}}{=} j_pk_p$ and consider the matrix $A^{\{p\}} = [\dot{a}(i_1i_2 \dots i_p, i_{p+1} \dots i_d)]$, where comma separates column and row indices. Obviously, $r_p \geqslant \operatorname{rank} A^{\{p\}}$.[1] Therefore, it is enough to show that all QTT matricisations of the Fourier matrix $F_d = [\omega_d^{jk}]_{j,k=0}^{2^d-1}$ have full ranks. The p-th matricisation $F_d^{\{p\}} = [f_d^{\{p\}}(j'k', j''k'')]$ appears after the following index splitting

$$j = j' + 2^pj'', \quad k = k' + 2^pk'', \qquad j', k' = 0, \dots, 2^p - 1, \quad j'', k'' = 0, \dots, 2^{d-p} - 1.$$

First, we assume $p \leqslant d/2$ and write

$$f_d^{\{p\}}(j'k', j''k'') = \omega_d^{jk} = \omega_d^{j'k'}\omega_{d-p}^{j'k''}\omega_{d-p}^{k'j''}\omega_{d-2p}^{j''k''}.$$

[1] The decomposition with $r_p = \operatorname{rank} A_p$ always exists. This important result is proved in [14]

In the matrix form,

$$F_d^{\{p\}} = \Omega' \left(F_{d-p}^{\square} \otimes F_{d-p}^{\square}\right) \Omega'', \tag{6}$$

where Ω' and Ω'' are diagonal matrices, $F_{d-p}^{\square}$ is the submatrix of first 2^p rows of F_{d-p} and "$\otimes$" denotes the Kronecker product. Since the Fourier matrix is unnormalised orthogonal, $F_q^* F_q = 2^q I$, the same holds for its rows, $F_{d-p}^{\square}(F_{d-p}^{\square})^* = 2^{d-p} I$. Since all elements on the diagonals of Ω' and Ω'' are unit in modulus, the matricisation (6) has orthogonal rows and hence has a full rank. For $p > d/2$ the result follows similarly. $\square$

Remark 1. The low-rank approximation of the Fourier matrix in the QTT format with reasonable accuracy is not possible.

Proof. The matricisation $F_d^{\{p\}}$ contains orthogonal rows (for $p \leqslant \frac{d}{2}$) or columns (for $p \geqslant \frac{d}{2}$), scaled by a constant. Assume $p \leqslant \frac{d}{2}$) and write the Gram matrix

$$F_d^{\{p\}} \left(F_d^{\{p\}}\right)^* = \Omega' \left(F_{d-p}^{\square} \otimes F_{d-p}^{\square}\right) \Omega'' (\Omega'')^* \left(F_{d-p}^{\square} \otimes F_{d-p}^{\square}\right)^* (\Omega')^* = (2^{d-p})^2 I.$$

All singular values of $F_d^{\{p\}}$ are the same, therefore its low-rank approximation with reasonable accuracy is not possible. It follows that the approximation of the Fourier matrix in the QTT format (5) with relative accuracy $\varepsilon \leqslant 2^{-\min(p,d-p)/2}$ has full ranks $r_k = 2^{\min(k,d-k)}$. $\square$

2.2. Radix-2 recurrence in the QTT format

Now we explain how to *apply* the Fourier transform to the QTT vector efficiently. We define

$$\begin{array}{rclcl} j &=& \overline{j_1 j_2 \ldots j_d} &=& j_1 + 2j', \qquad j' = \overline{j_2 \ldots j_d}, \\ k &=& \overline{k_1 \ldots k_{d-1} k_d} &=& k' + 2^{d-1} k_d, \qquad k' = \overline{k_1 \ldots k_{d-1}} \end{array} \tag{7}$$

and split odd and even values of the result

$$\begin{aligned} y(0+2j') &= \sum_{k'=0}^{2^{d-1}-1} x(k')\omega_d^{2j'k'} + \sum_{k'=0}^{2^{d-1}-1} x(k'+2^{d-1})\omega_d^{2j'(k'+2^{d-1})} \\ &= \sum_{k'=0}^{2^{d-1}-1} \left(x(k') + x(k'+2^{d-1})\right) \omega_d^{2j'k'} \\ y(1+2j') &= \sum_{k'=0}^{2^{d-1}-1} x(k')\omega_d^{(1+2j')k'} + \sum_{k'=0}^{2^{d-1}-1} x(k'+2^{d-1})\omega_d^{(1+2j')(k'+2^{d-1})} \\ &= \sum_{k'=0}^{2^{d-1}-1} \left(x(k') - x(k'+2^{d-1})\right) \omega_d^{k'} \omega_d^{2j'k'}. \end{aligned}$$

We come to the well-known radix-2 Fourier recurrence, the simplest and most common case of the Cooley-Tuckey fast Fourier transform (FFT) algorithm [11, 12]. It reduces full-size Fourier transform to the half-sized ones,

$$P_d F_d = \begin{bmatrix} F_{d-1} & \\ & F_{d-1} \end{bmatrix} \begin{bmatrix} I & \\ & \Omega_{d-1} \end{bmatrix} \begin{bmatrix} I & I \\ I & -I \end{bmatrix}. \tag{8}$$

Here P_d is a *bit-shift* permutation, that agglomerates even and odd elements of a vector, and $\Omega_{d-1} = \mathrm{diag}\{\omega_d^{k'}\}_{k'=0}^{2^{d-1}-1}$ is a matrix of *twiddle factors*. We define them for other indices as follows

$$(P_p y)(\overline{\underbrace{j_2 j_3 \ldots j_p j_1}_{\text{bit-shift}}\ j_{p+1} \ldots j_d}) = y(\overline{j_1 j_2 \ldots j_p j_{p+1} \ldots j_d}), \tag{9}$$

$$\Omega_p = \mathrm{diag}(1, \omega_{p+1}, \omega_{p+1}^2, \ldots, \omega_{p+1}^{2^p-1}).$$

Our goal is to compute $y = F_d x$ for the vector x given in the QTT format

$$x(k) = x(\overline{k_1 \ldots k_d}) = X_{k_1}^{(1)} \ldots X_{k_d}^{(d)}. \tag{10}$$

First, we note that "top" and "bottom" half-vectors of x read

$$\begin{aligned} x_{\mathrm{top}}(K) &\overset{\mathrm{def}}{=} x(K) = x(\overline{k_1 \ldots k_{d-1} 0}) = X_{k_1}^{(1)} X_{k_2}^{(2)} \ldots X_{k_{d-1}}^{(d-1)} X_{k_d=0}^{(d)}, \\ x_{\mathrm{bot}}(K) &\overset{\mathrm{def}}{=} x(K + 2^{d-1}) = x(\overline{k_1 \ldots k_{d-1} 1}) = X_{k_1}^{(1)} X_{k_2}^{(2)} \ldots X_{k_{d-1}}^{(d-1)} X_{k_d=1}^{(d)}, \end{aligned}$$

and their summation/subtraction affects only the last core.

$$\hat{x} \overset{\mathrm{def}}{=} \begin{bmatrix} I & I \\ I & -I \end{bmatrix} x, \qquad \hat{x}(k) = X_{k_1}^{(1)} \ldots X_{k_{d-1}}^{(d-1)} \hat{X}_{k_d}^{(d)}, \qquad \begin{aligned} \hat{X}_0^{(d)} &= X_0^{(d)} + X_1^{(d)}, \\ \hat{X}_1^{(d)} &= X_0^{(d)} - X_1^{(d)}. \end{aligned}$$

The multiplication by the diagonal matrix writes as a Hadamard multiplication

$$z = \begin{bmatrix} I & \\ & \Omega_{d-1} \end{bmatrix} \begin{bmatrix} I & I \\ I & -I \end{bmatrix} x = \begin{bmatrix} I & \\ & \Omega_{d-1} \end{bmatrix} \hat{x} = w_d \odot \hat{x}, \tag{11}$$

$$w_d^T \overset{\mathrm{def}}{=} [\ \underbrace{1 \ldots 1}_{2^{d-1}\ \text{elements}}\ \ \underbrace{1\ \omega_d\ \omega_d^2 \ldots\ \omega_d^{2^{d-1}-1}}_{2^{d-1}\ \text{elements}}\ ],$$

where vector w_d has the following rank-two QTT decomposition

$$w_d(k) = w_d(\overline{k_1 \ldots k_d}) = \begin{bmatrix} 1 & \omega_d^{k_1} \end{bmatrix} \begin{bmatrix} 1 & \\ & \omega_d^{2k_2} \end{bmatrix} \cdots \begin{bmatrix} 1 & \\ & \omega_d^{2^{d-2}k_{d-1}} \end{bmatrix} \begin{bmatrix} 1 - k_d \\ k_d \end{bmatrix}.$$

The Hadamard product $\mathbf{C} = \mathbf{A} \odot \mathbf{B}$ of two tensors in the TT format (1) writes easily in the TT format with the ranks being the product of ones of $\mathbf{A}$ and $\mathbf{B}$, namely

$$c(i_1, \ldots, i_d) = C_{i_1}^{(1)} \ldots C_{i_d}^{(d)}, \qquad C_{i_k}^{(k)} = A_{i_k}^{(k)} \otimes B_{i_k}^{(k)}.$$

Therefore, vector $z = w_d \odot \hat{x}$ has the following QTT decomposition

$$\begin{aligned} &z(k) = w_d(k)\hat{x}(k) = Z_{k_1}^{(1)} Z_{k_2}^{(2)} \ldots Z_{k_d}^{(d)}, \qquad \text{with} \qquad Z_{k_1}^{(1)} = \begin{bmatrix} X_{k_1}^{(1)} & \omega_d^{k_1} X_{k_1}^{(1)} \end{bmatrix}, \\ &Z_{k_p}^{(p)} = \begin{bmatrix} X_{k_p}^{(p)} & \\ & \omega_d^{2^{p-1}k_p} X_{k_p}^{(p)} \end{bmatrix}, \quad p = 2, \ldots, d-1, \qquad Z_{k_d}^{(d)} = \begin{bmatrix} (1 - k_d)\hat{X}_{k_d}^{(d)} \\ k_d \hat{X}_{k_d}^{(d)} \end{bmatrix}, \end{aligned} \tag{12}$$

and, shortly, *multiplication by twiddle factor doubles the QTT ranks.*

Algorithm 1: FFT-QTT, exact computation

```
Input: $x(k) = X^{(1)}_{k_1} X^{(2)}_{k_2} \dots X^{(d)}_{k_d}$
Output: $y(j) = \sum_{k=0}^{2^d-1} x(k) \omega_d^{jk} = Y^{(1)}_{j_1} Y^{(2)}_{j_2} \dots Y^{(d)}_{j_d}$
1: Define $x_d(k) = x(k) = X^{(1)}_{d,k_1} X^{(2)}_{d,k_2} \dots X^{(d)}_{d,k_d}$ with $X^{(p)}_{d,k_p} = X^{(p)}_{k_p}$.
2: for $D = d, d-1, \dots, 1$ do
3:     $k := \overline{k_1 \dots K_D}$, $k = 0, \dots, 2^D - 1$.
4:     $\hat{x}_D(k) := X^{(1)}_{D,k_1} X^{(2)}_{D,k_2} \dots X^{(D-1)}_{D,k_{D-1}} \hat{X}^{(D)}_{D,k_D}$ with $\hat{X}^{(D)}_{D,0} = X^{(D)}_{D,0} + X^{(D)}_{D,1}$,  $\hat{X}^{(D)}_{D,1} = X^{(D)}_{D,0} - X^{(D)}_{D,1}$
5:     Compute $z_D(k) := w_D(k) \hat{x}_D(k) = Z^{(1)}_{D,k_1} Z^{(2)}_{D,k_2} \dots Z^{(D)}_{D,k_D}$ by (12)
6:     $x_{D-1}(\overline{k_1 \dots k_{D-1}}) := X^{(1)}_{D-1,k_1} X^{(2)}_{D-1,k_2} \dots X^{(D-1)}_{D-1,k_{D-1}}$ with $X^{(p)}_{D-1,k_p} := Z^{(p)}_{D,k_p}$.
7: end for
8: $Y^{(p)}_{j_p} = (Z^{(d-p+1)}_{j_p})^T$, $p = 1, \dots, d$. {bit-reverse the output}
```

The last step to implement (8) is to apply the half-size Fourier transform to the "top" and "bottom" parts of vector z, i.e. to compute $(F_{d-1} \times I_1) z$, where I_p is identity matrix of size 2^p. Clearly, this operation does not affect the last core of the QTT representation. We are only to apply the half-size Fourier transform to the "subtrain" Z', that is composed from first $d-1$ cores of z,

$$z'(\overline{k_1 \dots k_{d-1}}, :) = Z^{(1)}_{k_1} \dots Z^{(d-1)}_{k_{d-1}}.$$

Here Z' is $n/2 \times 2r_{d-1}$ matrix and all rows $z'(\overline{k_1 \dots k_{d-1}}, :)$ are simultaneously represented in the QTT format with border ranks 1 and $2r_{d-1}$. Finally, we notice that all the described steps are valid if we substitute vector x by the $n \times r$ matrix given in QTT format, so we can apply the radix-2 recurrence to Z' by repeating the previously described steps.

Each radix-2 step applies the bit-shift permutation to the result. It is easy to see that $P_1 P_2 \dots P_d = R_d$, which is a *bit-reverse* permutation $(R_d y)(\overline{j_d j_{d-1} \dots j_1}) = y(\overline{j_1 \dots j_d})$. It can be nicely implemented in QTT format *without any computations* by reversing the order of cores in the tensor train,

$$x(k) = x(\overline{k_1 \dots k_d}) = X^{(1)}_{k_1} \dots X^{(d)}_{k_d}, \qquad (R_d x)(\overline{k_d \dots k_1}) = \left(X^{(d)}_{k_d}\right)^T \dots \left(X^{(1)}_{k_1}\right)^T. \tag{13}$$

Therefore, we add the bit-reverse step and result in Algorithm 1.

3. Approximate Fourier transform

The advances of tensor train format disappear if TT ranks are too large. However, in Algorithm 1 the TT ranks grow twice each time we multiply by twiddle factors on Line 5. After d steps they grow by factor $2^d = n$, which makes the TT representation completely ineffective. To keep the moderate values of ranks and perform computation efficiently, it is necessary to truncate the TT ranks, i.e. to approximate the result by the TT format with smaller values of ranks. The approximation can be done up to

Algorithm 2: [5] Orthogonalisation in TT format (left-to-right)

```
Input: Tensor $\mathbf{A}$ in the QTT format (1)
Output: Tensor $\mathbf{A}$, with orthogonal structured columns, see (14)
1: $\begin{bmatrix} B_0^{(1)} & B_1^{(1)} \end{bmatrix} = \begin{bmatrix} A_0^{(1)} & A_1^{(1)} \end{bmatrix}$
2: for $p = 1, \dots, d-1$ do
3:     Orthogonalise {for instance, by QR}
       $$\begin{bmatrix} B_0^{(p)} \\ B_1^{(p)} \end{bmatrix} =: \begin{bmatrix} C_0^{(p)} \\ C_1^{(p)} \end{bmatrix} R_{p+1}, \qquad \begin{bmatrix} C_0^{(p)} \\ C_1^{(p)} \end{bmatrix}^* \begin{bmatrix} C_0^{(p)} \\ C_1^{(p)} \end{bmatrix} = \sum_{j_p} (C_{j_p}^{(p)})^* C_{j_p}^{(p)} = I.$$
4:     $\begin{bmatrix} B_0^{(p+1)} & B_1^{(p+1)} \end{bmatrix} := R_{p+1} \begin{bmatrix} A_0^{(p+1)} & A_1^{(p+1)} \end{bmatrix}$
5: end for
6: $A_{j_p}^{(p)} := C_{j_p}^{(p)}, \quad p = 1, \dots, d.$
```

some fixed accuracy bound, or by setting the maximum possible TT ranks. In this section we give a short description of SVD-based algorithm from [5], incorporate it for the recompression of vectors in Algorithm 1 and give the error and complexity analysis of the resulted method.

3.1. Orthogonalisation and rank truncation in the TT format

First, we introduce the orthogonalisation algorithm in the TT format. Consider tensor $\mathbf{A}$ in TT format (1) with left border rank $r_0 = 1$. The Algorithm 2 modifies the cores $A_{j_p}^{(p)}$ of the decomposition (1) without changing the elements of $\mathbf{A}$. However, the "structured columns" of every matrix A'_p for $p = 1, \dots, d-1$, become orthogonal as follows

$$A'_p = [a'_p(\overline{j_1 \dots j_p}, \alpha_p)], \qquad a'_p(\overline{j_1 \dots j_p}, :) = A_{j_1}^{(1)} \dots A_{j_p}^{(p)}$$

$$(A'_p)^* A'_p = I, \qquad \sum_{j_1 \dots j_p} \bar{a}'_p(\overline{j_1 \dots j_p}, \alpha_p) a'_p(\overline{j_1 \dots j_p}, \beta_p) = \delta_{\alpha_p, \beta_p} \overset{\text{def}}{=} \begin{cases} 1, & \alpha_p = \beta_p, \\ 0, & \alpha_p \neq \beta_p. \end{cases}$$

In the description of Algorithm 2 we assume $n_p = 2$ (QTT case), but the generalisation to larger mode sizes is straightforward.

The last equation allows to generalize the orthogonalisation scheme to the case $r_0 > 1$ if we define the "structured columns" as ones of the matrix

$$A'_p = [a'(\alpha_0 \overline{j_1 \dots j_p}, \alpha_p)], \qquad a'(\alpha_0 \overline{j_1 \dots j_p}, \alpha_p) = A_{j_1}^{(1)}(\alpha_0, :) A_{j_2}^{(2)} \dots A_{j_{p-1}}^{(p-1)} A_{j_p}^{(p)}(:, \alpha_p),$$

where the left border index $\alpha_0 = 1, \dots, r_0$ and mode index $\overline{i_1 \dots i_p}$ are merged in the single multiindex. In this case the left-to-right orthogonalisation reads

$$\begin{gathered} \sum_{\alpha_0, j_1 \dots j_p} \bar{a}(\alpha_0, \overline{j_1 \dots j_p}, \alpha_p) a(\alpha_0, \overline{j_1 \dots j_p}, \beta_p) = \delta_{\alpha_p, \beta_p}, \\ \sum_{j_1 \dots j_p} A_{j_1 \dots j_p}^* A_{j_1 \dots j_p} = I, \qquad \text{where} \quad A_{j_1 \dots j_p} = A_{j_1}^{(1)} \dots A_{j_p}^{(p)}. \end{gathered} \tag{14}$$

Algorithm 3: [5] TT rank truncation (right-to-left)

Input: Tensor $\mathbf{A}$ in the QTT format (1), accuracy ε or ranks $R_1, \dots, R_{d-1}$

Output: Tensor $\tilde{\mathbf{A}}$ such that $\|\mathbf{A}-\tilde{\mathbf{A}}\|_F \leqslant \varepsilon \|\mathbf{A}\|_F$ or TT ranks $\tilde{r}_p \leqslant R_p, \quad p = 1, \dots, d-1$.

```
1: Apply orthogonalisation Algorithm 2 and ensure (14)
2: $\begin{bmatrix} B_0^{(d)} & B_1^{(d)} \end{bmatrix} = \begin{bmatrix} A_0^{(d)} & A_1^{(d)} \end{bmatrix}$
3: for $p = d, d-1, \dots, 2$ do
4:     Compute truncated SVD
           $\begin{bmatrix} B_0^{(p)} & B_1^{(p)} \end{bmatrix} \approx: \begin{bmatrix} \tilde{B}_0^{(p)} & \tilde{B}_1^{(p)} \end{bmatrix} = U_{p-1} S_{p-1} \begin{bmatrix} C_0^{(p)} & C_1^{(p)} \end{bmatrix}$
           $U_{p-1}^* U_{p-1} = I, \qquad \begin{bmatrix} C_0^{(p)} & C_1^{(p)} \end{bmatrix} \begin{bmatrix} C_0^{(p)} & C_1^{(p)} \end{bmatrix}^* = I,$
       filtering small singular values using accuracy criterion
           $\left\| \begin{bmatrix} B_0^{(p)} & B_1^{(p)} \end{bmatrix} - \begin{bmatrix} \tilde{B}_0^{(p)} & \tilde{B}_1^{(p)} \end{bmatrix} \right\|_F \leqslant \frac{\varepsilon}{\sqrt{d}} \left\| \begin{bmatrix} B_0^{(p)} & B_1^{(p)} \end{bmatrix} \right\|_F$
       or rank criterion $\operatorname{rank}([\tilde{B}_0^{(p)} \ \tilde{B}_1^{(p)}]) \leqslant R_p$.
5:     Modify
           $\begin{bmatrix} B_0^{(p-1)} \\ B_1^{(p-1)} \end{bmatrix} := \begin{bmatrix} A_0^{(p-1)} \\ A_1^{(p-1)} \end{bmatrix} U_{p-1} S_{p-1}$
6: end for
7: $\tilde{A}_{j_1}^{(1)} := B_{j_1}^{(1)}$ and $\tilde{A}_{j_p}^{(p)} := C_{j_p}^{(p)}, \quad p = 2, \dots, d.$
```

To explain, why Algorithm 2 ensures (14) one can apply the following lemma subsequently for $A'_2, \dots, A'_p$.

Lemma 1. Consider $p \times r$ matrices A_i and $r \times q$ matrices B_j, such that $\sum_i A_i^* A_i = I$ and $\sum_j B_j^* B_j = I$. Then for $C_{ij} = A_i B_j$ it holds

$$\sum_{ij} C_{ij}^* C_{ij} = \sum_{ij} (A_i B_j)^* (A_i B_j) = \sum_j B_j^* \left(\sum_i A_i^* A_i \right) B_j = \sum_j B_j^* B_j = I.$$

Based on the orthogonalisation Algorithm 2, the TT compression (rank truncation) Algorithm 3 was proposed in [5]. On step p the core $A^{(p)}$ of given TT format undergoes the approximation and the TT rank r_{p-1} is reduced using accuracy threshold and/or maximum rank criterion. The orthogonalisation steps proposed in Algorithm 3 ensure that 'interfaces' U and V^T in the decomposition

$$a(j_1, j_2, \dots, j_d) = \underbrace{A_{j_1}^{(1)} \dots A_{j_{p-1}}^{(p-1)}}_{=u(\overline{j_1 \dots j_{p-1}}, :)} A_{j_p}^{(p)} \underbrace{A_{j_{p+1}}^{(p+1)} \dots A_{j_d}^{(d)}}_{v(:, \overline{j_{p+1} \dots j_d})}$$

have orthogonal columns, and therefore an approximation error of $A^{(p)}$ introduces the *same* perturbation to the whole tensor. This allow to control the overall accuracy in

Algorithm 4: FFT-QTT, approximation

```
Input: $x(k) = X^{(1)}_{k_1} X^{(2)}_{k_2} \dots X^{(d)}_{k_d}$, accuracy $\varepsilon$ or ranks $R_1, \dots, R_{d-1}$
Output: $y(j) = \sum_{k=0}^{2^d-1} x(k)\omega_d^{jk} = Y^{(1)}_{j_1} Y^{(2)}_{j_2} \dots Y^{(d)}_{j_d}$
1: Define $x_d(k) = x(k) = X^{(1)}_{d,k_1} X^{(2)}_{d,k_2} \dots X^{(d)}_{d,k_d}$ with $X^{(p)}_{d,k_p} = X^{(p)}_{k_p}$.
2: for $D = d, d-1, \dots, 1$ do
3:     $k := \overline{k_1 \dots k_D}$, $k = 0, \dots, 2^D - 1$.
4:     $\hat{x}_D(k) := X^{(1)}_{D,k_1} X^{(2)}_{D,k_2} \dots X^{(D-1)}_{D,k_{D-1}} \hat{X}^{(D)}_{D,k_D}$ with $\hat{X}^{(D)}_{D,0} = X^{(D)}_{D,0} + X^{(D)}_{D,1}$,   $\hat{X}^{(D)}_{D,1} = X^{(D)}_{D,0} - X^{(D)}_{D,1}$
5:     Compute $z_D(k) := w_D(k)\hat{x}_D(k) = Z^{(1)}_{D,k_1} Z^{(2)}_{D,k_2} \dots Z^{(D)}_{D,k_D}$ by (12)
6:     Apply Algorithm 3 to approximate $z_D(k) \approx: \tilde{z}_D(k) = \tilde{Z}^{(1)}_{D,k_1} \dots \tilde{Z}^{(D)}_{D,k_D}$ and reduce
       the TT ranks using accuracy $\varepsilon$ or maximum rank $R_1, \dots, R_{D-1}$ criterion.
7:     $x_{D-1}(\overline{k_1 \dots k_{D-1}}) := X^{(1)}_{D-1,k_1} X^{(2)}_{D-1,k_2} \dots X^{(D-1)}_{D-1,k_{D-1}}$ with $X^{(p)}_{D-1,k_p} := \tilde{Z}^{(p)}_{D,k_p}$.
8: end for
9: $Y^{(p)}_{j_p} = (\tilde{Z}^{(d-p+1)}_{d-p+1,j_p})^T$, $p = 1, \dots, d$. {bit-reverse the output}
```

Algorithm 3. The detailed proof can be found in [5]. Including the approximation step in the Fourier decomposition Algorithm 1 we result in Algorithm 4.

3.2. Error analysis

It is important to show how the error introduced on the approximation step in Algorithm 4 affects the whole tensor.

If the truncation step is done without any error, than on each step $D = d, \dots, 1$ of Algorithm 4 it holds $R_D y = R_{D-1}(F_{D-1} \times I_{d-D+1})\mathbf{z}_D$, where I_p is identity $2^p \times 2^p$ matrix and

$$\mathbf{z}_D(k) = \mathbf{z}_D(\overline{k_1 \dots k_d}) = z_D(\overline{k_1 \dots k_D}) \prod_{p=D+1}^{d} \tilde{Z}^{(p)}_{p,k_p}.$$

The right "interface" $\prod_{p=D+1}^{d} \tilde{Z}^{(p)}_{p,k_p}$ is orthogonal, since every factor $Z^{(p)}_p$ comes from the TT rank truncation Algorithm 3 and has right-to-left orthogonalisation property. Therefore, if z_D undergoes the approximation $\|z_D - \tilde{z}_D\| \leqslant \varepsilon \|z_D\|$ then $\|\mathbf{z}_D - \tilde{\mathbf{z}}_D\| \leqslant \varepsilon \|\mathbf{z}_D\|$. Permutation matrices R_D and R_{D-1} are orthogonal, Fourier transform matrix F_p is unnormalised orthogonal, $F_p^* F_p = 2^p I_p$, and the same holds for matrix $A = (F_{D-1} \times I_{d-D+1})$, i.e. $A^*A = 2^p I_d$. Therefore,

$$\text{if} \quad R_D \tilde{y}_D = R_{D-1}(F_{D-1} \times I_{d-D+1})\tilde{\mathbf{z}}_D, \quad \text{then} \quad \|y - \tilde{y}_D\| \leqslant \varepsilon \|y\|.$$

We come to the following theorem.

Theorem 2. If the approximation step in Algorithm 4 is performed with the relative error ε, than the accuracy of the output can be estimated as

$$\|y - F_d x\| \leqslant d\varepsilon \|y\|.$$

3.3. Complexity analysis

Now we estimate the asymptotic complexity of Algorithm 4 in a special case when QTT ranks of vectors x_D are bounded on all steps of the algorithm (as well as ranks of input and output). We start from the following theorem from [4] that evaluates the complexity of Algorithms 2 and 3.

Theorem 3. For d-tensor X given in the TT format with ranks $r_p \leqslant r$ and mode sizes $n_p \leqslant n$,. the complexity of both the orthogonalisation Algorithm 2 and rank truncation Algorithm 3 is bounded as $\mathtt{work} \lesssim dnr^3$.

Using this theorem, we can estimate the complexity of proposed algorithm.

Theorem 4. If on levels $D = d, \dots, 1$ of the FFT-QTT Algorithm 4 the QTT ranks of the *transition* vectors x_D are bounded by r_D, then the overall complexity is

$$\mathtt{work} \lesssim \sum_{D=1}^{d} Dr_D^3 \leqslant d^2 r^3, \qquad \text{where} \quad r = \max r_D. \tag{15}$$

Proof. On step $D = d, \dots, 1$ we do the following.

1. Alter the last block $\hat{X}_0^{(D)} = X_0^{(D)} + X_1^{(D)}$, $\hat{X}_1^{(D)} = X_0^{(D)} - X_1^{(D)}$, that requires $\mathcal{O}(r_D^2)$ operations.
2. Multiply the cores by twiddle factors, see Eq. (12), that requires $\mathcal{O}(r_D^2)$ operations.
3. Compress the "subtrain" $z_D(k) = Z_{k_1}^{(1)} \cdots Z_{k_D}^{(D)}$ using algorithm 3, that requires $\mathcal{O}(Dr_D^3)$ operations.

By summation of the complexity of all the steps, we come to (15). $\square$

Now we can discuss two possible strategies of rank truncation on Line 6 of Algorithm 4. If maximum rank criterion is used with $R_k \leqslant r$, then the QTT ranks of transition vectors x_D remain bounded during the procedure and we can predict the overall cost of the algorithm by Theorem 4. However such a "brute-force" rank truncation generally introduces an unpredictable error to the result and we can not rely on the Theorem 2. Vise versa, if an ε–truncation strategy is used, the Theorem 2 provides us with the accuracy of result, but the complexity of QTT-FFT given by Theorem 4 could not be predicted, since $r = \max r_D$ is generally not know in advance. In the following we give a simple example when we can estimate both the complexity and accuracy of the proposed scheme.

Theorem 5. Consider input vector $x = [\exp(\frac{2\pi i}{2^d} fk)]_{k=0}^{2^d-1}$ that has the QTT rank-one representation by (3). If "frequency" f is integer, the QTT ranks of transition vectors x_D on all steps $D = d, \dots, 1$ of the QTT-FFT Algorithm 4 are equal to one.

Proof. Without loss of generality we can consider $0 \leqslant f \leqslant 2^d - 1$, so $f = \overline{f_1, \dots, f_d}$ with $f_p = 0, 1$, $p = 1, \dots, d$. The input vector $x_d = x$ has rank-one QTT decomposition by (3),

$$x_d(k) = \exp(fk) = \prod_{p=1}^{d} x^{(p)}_{d,k_p}, \qquad x^{(p)}_{d,k_p} = \exp\left(\frac{2\pi i}{2^{d-p+1}} f k_p\right).$$

On the first step of Algorithm 4,

$$\hat{x}_d(k) = \left(\prod_{p=1}^{d-1} x^{(p)}_{k_p}\right) \hat{x}^{(d)}_{k_d}, \qquad \hat{x}^{(d)}_0 = x^{(d)}_0 + x^{(d)}_1, \quad \hat{x}^{(d)}_1 = x^{(d)}_0 - x^{(d)}_1.$$

Note that

$$x^{(d)}_{k_d} = \exp\left(\frac{2\pi i}{2} k_d f\right) = \exp\left(\frac{2\pi i}{2}\, k_d\, \overline{f_1 \dots f_d}\right) = (-1)^{k_d f_1},$$

and so the multiplication by the twiddle factors (11) writes for f even and odd as follows

- for $f_1 = 0$, $x^{(d)} = [1 \quad 1]$, $\hat{x}^{(d)} = [2 \quad 0]$ (bottom half of $\hat{x}_d$ is zero) and $z_d = \hat{x}_d$,
- for $f_1 = 1$, $x^{(d)} = [1 \; -1]$, $\hat{x}^{(d)} = [0 \quad 2]$ (top half of $\hat{x}_d$ is zero) and $z_d = \Omega_{d-1}\hat{x}_d$.

In both cases the QTT ranks of z_d are equal to one, $z_d(k) = \left(\prod_{p=1}^{d-1} z^{(p)}_{k_p}\right) \hat{x}^{(d)}_{k_d}$. For f even,

$$z^{(p)}_{k_p} = x^{(p)}_{k_p} = \exp\left(\frac{2\pi i}{2^{d-p+1}} f k_p\right),$$

for f odd,

$$z^{(p)}_{k_p} = x^{(p)}_{k_p} \omega_d^{2^{p-1} k_p} = \exp\left(\frac{2\pi i}{2^{d-p+1}} f k_p\right) \exp\left(-\frac{2\pi i}{2^{d-p+1}} k_p\right) = \exp\left(\frac{2\pi i}{2^{d-p+1}} (f-1) k_p\right).$$

The approximation step can be skipped, since we have exact rank-one QTT representation for z_d that can not be further compressed. Finally,

$$x_{d-1}(k') = x_{d-1}(\overline{k_1 \dots k_{d-1}}) = \prod_{p=1}^{d-1} \exp\left(\frac{2\pi i}{2^{d-p}} f' k_p\right) = \exp\left(\frac{2\pi i}{2^{d-1}} f' k'\right),$$

where $k' = 1, \dots, 2^{d-1} - 1$, $\quad f' = \lfloor f/2 \rfloor = \overline{f_2 \dots f_d}$. The proof continues recursively for the half-sized vector x_{d-1} which is again the discretised exponent with integer frequency. Since the bit-reverse permutation (13) does not change the QTT ranks, the proof is complete. $\square$

Corollary 1. For input vector $x = [x(k)]$ being a sum of r plane waves $x(k) = \sum_{p=1}^{r} \exp(\frac{2\pi i}{2^d} f_p k)$ with integer "frequencies" f_p, the transition vectors x_D on all steps of Algorithm 4 have exact QTT representation with ranks not larger than r. This implies the complexity bound (15) given by Theorem 4.

4. Discrete Fourier transform in higher-dimensional case

In addition to the QTT structure for one-dimensional vectors in Fourier transform, we may apply tensor structures to compute the higher-dimensional Fourier transform

$$\begin{gathered} y(j_1,\ldots,j_M) = \sum_{k_1=0}^{n_1-1} \ldots \sum_{k_M=0}^{n_M-1} x(k_1,\ldots,k_M)\omega_{d_1}^{j_1k_1}\ldots\omega_{d_M}^{j_Mk_M}, \\ \omega_d = \exp\left(-\frac{2\pi i}{2^d}\right), \quad i^2=-1, \qquad j_p = 0,\ldots,n_p-1, \quad p=1,\ldots,M. \end{gathered} \tag{16}$$

The TT approximations (1) for input and output tensors read

$$x(k_1,\ldots,k_M) = X_{k_1}^{(1)}\ldots X_{k_M}^{(M)}, \qquad y(j_1,\ldots,j_M) = Y_{j_1}^{(1)}\ldots Y_{j_M}^{(M)}.$$

Therefore the M-dimensional transform splits to independent unimodal transforms

$$Y^{(p)} = F_{d_p}X^{(p)}, \qquad Y_{j_p}^{(p)} = \sum_{k_p=0}^{n_p-1} X_{k_p}^{(p)}\omega_{d_p}^{j_pk_p}, \qquad p=1,\ldots,M.$$

To perform the latter, we quantize all mode indices k_p by (2) and obtain tensors in the QTT format

$$X_{k_p}^{(p)} = X^{(p)}(\overline{k_{p,1},\ldots,k_{p,d_p}}) = X_{k_{p,1}}^{(p,1)}X_{k_{p,2}}^{(p,2)}\cdots X_{k_{p,d_p}}^{(p,d_p)} = \prod_{q_p=1}^{d_p} X_{k_{p,q_p}}^{(p,q_p)}, \tag{17}$$

that are then composed in the tensor train for the whole array

$$x(\overline{k_{1,1},\ldots,k_{1,d_1}},\ldots,\overline{k_{M,1},\ldots,k_{M,d_M}}) = \prod_{p=1}^{M}\prod_{q_p=1}^{d_p} X_{k_{p,q_p}}^{(p,q_p)}. \tag{18}$$

We can apply the 1D Fourier transform Algorithm 4 to the “one dimension” sub-trains (17) independently, but with two modifications.

First, we should take care of the approximation error, introduced to the whole tensor. To control this, we need to require the appropriate orthogonality conditions while approximating the Fourier transform of p-th dimension train (17): all cores of the left interface should be orthogonalised left-to-right by Algorithm 2 fulfilling (14) and in the right interface the right-to-left orthogonalisation is required. The latter can be done using subsequently the bit-reverse permutation (13), left-to-right orthogonalisation Algorithm 2 and using bit-reverse again.

Second, the radix-2 scheme provides an output vector with reversed bits in the binary coding. In the 1D case, the bit-reverse can be easily done via the permutation of transposed TT cores in the reverse order, as the first and the last ranks $r_0 = r_d = 1$. But it does not hold for 1D parts in a multidimensional TT tensor, so we can not apply the bit-reverse operation just for every 1D part, as we obtain inconsistent ranks between the physical variables: for example, we would have to multiply $(X_{k_{1,1}}^{(1,1)})^T$ with

Algorithm 5: FFTn-QTT, approximation

```
Input: $x(k_1, ..., k_M) = \prod_{p=1}^{M} \prod_{q_p=1}^{d_p} X_{k_{p,q_p}}^{(p,q_p)}$, accuracy $\varepsilon$ or rank bounds $R_{1,1}, \ldots, R_{M,d_M-1}$
Output: $y(j_1, ..., j_M) = \prod_{p=1}^{M} \prod_{q_p=1}^{d_p} X_{k_{p,q_p}}^{(p,q_p)}$, defined by (16)
1: Orthogonalise $x(k_1, ..., k_M)$ right-to-left applying (13), Alg. 2 and (13) again.
2: for $p = 1, 2, \ldots, M$ do
3:     Apply FFT-QTT Algorithm 4 without the last bit-reverse operation to tensor $\prod_{q_p=1}^{d_p} X_{k_{p,q_p}}^{(p,q_p)}$, and store the result as $\prod_{q_p=1}^{d_p} Y_{k_{p,d-q_p+1}}^{(p,d-q_p+1)}$
4:     If $p \neq M$, orthogonalise tensor $\left(\prod_{q_p=1}^{d_p} Y_{k_{p,d-q_p+1}}^{(p,d-q_p+1)}\right) X_{k_{p+1,1}}^{(p+1,1)}$ left-to-right
5: end for
6: If necessary, permute the bits of $y$ in the proper order via the matrices (9) and compressions.
```

the sizes $r_{1,1} \times 1$ and $(X_{k_{2,d_2}}^{(2,d_2)})^T$ with the sizes $r_{2,d_2} \times r_{2,d_2-1}$. Fortunately, a wide class of applications (e.g. convolution, solution of discretised PDEs) requires *two* successive Fourier transforms, usually, forward and backward ones. In this case, no bit permutation is actually required: after the second transform, we will have the proper bit order. The only thing to care about, is to conduct all the operations between two transforms in the reverse bit ordering. In the case when we need only the forward transform, one should use the subsequent applications of the bit-permutation matrices (9) and compressions using Algorithm 3.

We summarize this in Algorithm 5.

5. Numerical examples

5.1. Preliminary remarks

In this section we compare accuracy and timings of the trigonometric transforms for one–, two– and three–dimensional functions with moderate QTT ranks. From (15) we see that the complexity of the FFT-QTT depends crucially of the upper rank bound. However, the maximum TT rank sometimes gives incorrect impression of "structure complexity" of particular data array. We define the *effective TT rank* r that accounts the distribution of all TT ranks $r_0, \ldots, r_d$, measuring the storage for TT format,

$$\mathtt{mem} = r_0 n_1 r_1 + r_1 n_2 r_2 + \ldots + r_{d-1} n_d r_d.$$

Therefore, *effective rank* rank $\mathbf{A}$ of tensor $\mathbf{A}$ with mode sizes n_p and TT ranks r_p is a positive solution of the quadratic equation $\mathtt{mem} = r_0 n_1 r + r n_2 r + \ldots + r n_d r_d$, i.e.

$$\operatorname{rank} A = r, \qquad \text{such that} \quad \left(\sum_{p=2}^{d-1} n_p\right) r^2 + (r_0 n_1 + n_d r_d)\, r - \sum_{p=1}^{d} r_{p-1} n_p r_p = 0. \quad (19)$$

Definitely, the effective rank is generally non-integer.

For experiments we use one Xeon E5504 CPU at 2.0 GHz with 72 GB of memory in Institute of Numerical Mathematics RAS, Russia. The TT algorithms are implemented in Fortran 90 and compiled by Intel Fortran Composer XE version 12.0 (64 bit) using BLAS/LAPACK from MKL library. For comparison we use the FFT algorithm from FFTW version 3.2.2 library provided with MKL.

5.2. Fourier images in 1D

5.2.1. Definitions. We consider several well-known examples of one-dimensional integral Fourier transforms and compute the corresponding DFTs. The functions considered are *rectangular pulse*

$$\operatorname{rect}(t) = \Pi(t) = \begin{cases} 0, & \text{if} \quad |t| > 1/2 \\ 1/2, & \text{if} \quad |t| = 1/2, \\ 1 & \text{if} \quad |t| < 1/2, \end{cases} \tag{20}$$

triangular function

$$\operatorname{tri}(t) = \Lambda(t) = \begin{cases} 1 - |t|, & \text{if} \quad |t| < 1, \\ 0 & \text{otherwise}, \end{cases} \tag{21}$$

and *Slater* function

$$s(t) = \exp(-|t|). \tag{22}$$

For integrable function $f(x)$, the Fourier transform, or *image* $\hat{f}(\xi)$ is defined for every real ξ by

$$\mathcal{F} f = \hat{f}(\xi) = \int_{-\infty}^{+\infty} f(t) \exp(-2\pi i t \xi) dt. \tag{23}$$

For functions considered, the Fourier images are known analytically

$$\hat{\Pi} = \operatorname{sinc}(\xi), \qquad \hat{\Lambda} = \operatorname{sinc}^2(\xi), \qquad \hat{s} = \frac{2}{1 + 4\pi^2\xi^2}. \tag{24}$$

where $\operatorname{sinc}(\xi) = \frac{\sin \pi\xi}{\pi\xi}$. Using uniform time and frequency grids $t_k = kh$ and $\xi_j = j\hat{h}$ with $k, j = -n/2, \ldots, n/2 - 1$, we discretise (23) using piecewise-constant basis functions,

$$\hat{f}(\xi_j) \approx \tilde{f}(\xi_j) = \frac{1}{n} \sum_{k=-n/2}^{n/2-1} f(t_k) \exp(-2\pi i t_k \xi_j) = \frac{1}{n} \sum_{k=-n/2}^{n/2-1} f(t_k) \exp(-2\pi i h \hat{h} k j). \tag{25}$$

If $h\hat{h} = 1/n$, the right-hand side is a DFT $\hat{f} = 1/n F_d f$. Here, vectors $f = [f_k]_{k=0}^{n-1}$ and $\hat{f} = [\hat{f}_j]_{j=0}^{n-1}$ contain elements $f(t_k)$ and $\tilde{f}(\xi_j)$ in a "modulo n" order,

$$f_{(k+n \mod n)} = f(t_k), \quad \hat{f}_{(j+n \mod n)} = \tilde{f}(\xi_j), \qquad j, k = -n/2, \ldots, n/2 - 1.$$

Alternatively, we can define grid points t_k and ξ_j by

$$t_k = \begin{cases} kh, & \text{for} \quad k = 0, \ldots, n/2 - 1, \\ (k-n)h, & \text{for} \quad k = n/2, \ldots, n-1; \end{cases} \qquad \xi_j = \begin{cases} j\hat{h}, & \text{for} \quad j = 0, \ldots, n/2 - 1, \\ (j-n)\hat{h}, & \text{for} \quad j = n/2, \ldots, n-1; \end{cases} \tag{26}$$

and use straight indexing $f_k = f(t_k)$, $\hat{f}_j = \hat{f}(\xi_j)$ (cf. Tab. 1). Using Fourier images (24), we can check the accuracy of (25) comparing $\hat{f}(\xi_j)$ with $\hat{f}_j$. This allows to test the accuracy of FFT-QTT for very large n.

Table 1. Binary indices, vector indices and grid points for $d = 4$, $h = 1/\sqrt{n}$

k_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
k_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
k_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
k_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t_k	0	$1/4$	$1/2$	$3/4$	1	$1\frac{1}{4}$	$1\frac{1}{2}$	$1\frac{3}{4}$	-2	$-1\frac{3}{4}$	$-1\frac{1}{2}$	$-1\frac{1}{4}$	-1	$-3/4$	$1/2$	$-1/4$

5.2.2. QTT representation of input. We take $n = 2^d$ and choose $h = \hat{h} = 1/\sqrt{n}$ to have the same discretisation accuracy at time and frequency domain. We are now to write low-rank QTT representations of discretised rectangular, triangular and Slater function exactly in the case of d even. First, note the correspondence (26) between vector index k and grid points t_k (see Table 1) and especially the special role of k_d, that switches between nonnegative ($k_d = 0$) and negative ($k_d = 1$) arguments t_k. So, if we naively define a QTT vector $[a_k]_{k=0}^{2^d-1}$ by formula for the exponential (3),

$$a_k = \exp(-hk) = \exp(-hk_1)\exp(-2hk_2)\dots\exp(-2^{d-2}hk_{d-1})\exp(-2^{d-1}hk_d), \qquad (a)$$

the result will look like on Fig. 1a, that is not what we supposed. However, we can "clear" all values of a_k corresponding to negative argument t_k by setting them to zero as follows

$$b_k = \exp(-hk_1)\exp(-2hk_2)\dots\exp(-2^{d-2}hk_{d-1})\exp(-2^{d-1}hk_d)(1-k_d), \qquad (b)$$

that is shown on Fig. 1b, Then, reflect this function about the y–axis,

$$c_k = \exp(-h(1-k_1))\dots\exp(-2^{d-2}h(1-k_{d-1}))\exp(-2^{d-1}h(1-k_d))k_d. \qquad (c)$$

and define $d = b + e^{-h}c$ to get the QTT rank-two representation of Slater function, see Fig. 1d.

The characteristic function for the continuous set of points can be defined as follows (see Fig. 1e)

$$e_k = (1-k_{d/2})\dots(1-k_d). \qquad (e)$$

By summation with the reflected function we get the vector

$$f_k = k_{d/2}\dots k_d + (1-k_{d/2})\dots(1-k_d), \qquad (f)$$

that is represented on Fig. 1f. It differs from the rectangle function in points $t_k = \pm 1$, i.e. the QTT representation for $\mathrm{rect}(t)$ can be constructed with ranks bounded by 4.

An arithmetic progression has QTT rank-2 representation

$$\alpha k + \beta = \begin{bmatrix} 2 & \beta + \alpha k_1 \end{bmatrix} \begin{bmatrix} 2 & \alpha k_2 \\ 0 & 1 \end{bmatrix} \dots \begin{bmatrix} 2 & \alpha k_{d-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha k_d \\ 1 \end{bmatrix}.$$

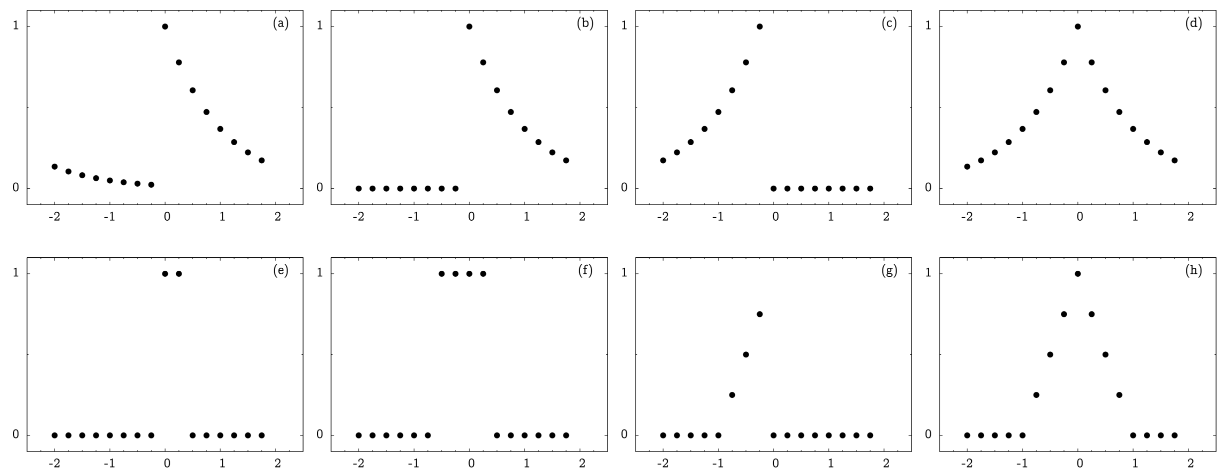
Table 2. Time for FFT-QTT (in milliseconds) versus size $n = 2^d$ and accuracy ε. Here $\texttt{time}_{\texttt{QTT}}$ if for FTT-QTT Algorithm 4 and $\texttt{time}_{\texttt{FFTW}}$ is for the FFT algorithm from FFTW library.

$f = \mathrm{rect}(t)$		$\varepsilon = 10^{-4}$		$\varepsilon = 10^{-8}$		$\varepsilon = 10^{-12}$	
d	$\texttt{time}_{\texttt{FFTW}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$
16	1.7	4.66	7.9	6.85	13.8	8.85	20.0
18	8.9	4.70	9.7	6.86	16.7	8.82	23.4
20	42.5	4.75	11.3	6.85	19.8	8.86	30.6
22	180	4.77	13.1	6.83	23.3	8.89	36.4
24	810	4.74	15.0	6.72	26.3	8.94	41.7
26	4100	4.62	17.0	6.76	30.0	8.89	46.5
28	26300	4.57	18.9	6.80	33.0	8.88	51.2
30	—	4.72	20.3	6.78	36.2	8.84	57.0
40	—	4.20	29.1	6.59	53.6	8.78	83.2
60	—	3.69	50.0	6.25	87.6	8.32	133

$f = \mathrm{tri}(t)$		$\varepsilon = 10^{-4}$		$\varepsilon = 10^{-8}$		$\varepsilon = 10^{-12}$	
d	$\texttt{time}_{\texttt{FFTW}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$
16	1.6	4.29	8.4	6.63	14.0	8.80	20.0
18	8.9	4.20	10.1	6.59	16.9	8.74	25.2
20	37.2	4.02	11.8	6.55	20.5	8.77	31.0
22	170	3.87	13.1	6.43	23.8	8.69	36.1
24	815	3.79	14.9	6.34	27.0	8.60	41.5
26	4000	3.68	16.8	6.22	30.7	8.48	46.9
28	26000	3.58	19.2	6.14	34.0	8.37	52.6
30	—	3.49	20.8	6.05	37.2	8.36	56.6
40	—	3.18	29.4	5.39	52.3	7.68	82.2
60	—	2.77	52.1	4.56	86.7	6.60	127

$f = s(t)$		$\varepsilon = 10^{-4}$		$\varepsilon = 10^{-8}$		$\varepsilon = 10^{-12}$	
d	$\texttt{time}_{\texttt{FFTW}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$	rank $\hat{f}$	$\texttt{time}_{\texttt{QTT}}$
16	1.7	4.50	11.1	7.27	15.1	9.39	23.0
18	8.9	4.39	10.4	7.20	18.9	9.44	28.7
20	38.5	4.19	12.1	7.13	22.7	9.47	34.4
22	177.2	3.99	13.8	7.01	26.4	9.41	40.5
24	840.6	3.86	15.9	6.96	29.8	9.24	47.2
26	4110	3.79	17.6	6.80	33.3	9.14	53.9
28	27000	3.64	19.9	6.68	37.1	8.99	60.0
30	—	3.57	22.5	6.49	40.9	8.86	69.4
40	—	3.23	32.2	5.78	57.0	8.18	91.7
60	—	2.85	54.8	4.92	92.3	6.98	138.6

Figure 1. Function-related vectors with low-rank QTT representation



Using this, we define the left part of the triangular function (see Fig. 1g) as follows

$$g_k = \begin{bmatrix} 2 & hk_1 \end{bmatrix} \begin{bmatrix} 2 & hk_2 \\ 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} 2 & hk_{d/2-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} hk_{d/2} \\ 1 \end{bmatrix} k_{d/2+1} \ldots k_d, \qquad (g)$$

and add the reflection, shifted by $\beta = h$, to get the rank-4 QTT representation of $\mathrm{tri}(t)$, see Fig. 1h. In [17] there are more examples of analytical QTT low-rank representations of function-related vectors.

5.2.3. Timings. In Table 2 we compare timings for the FFT-QTT Algorithm 4 and for FFT out-of-place algorithm from the FFTW library. All timings were obtained by averaging the computation time over a number of executions in a period of 128 seconds.

We see that for considered examples the FFT-QTT algorithm outperforms the FFT algorithm from FFTW library for $n > 2^{20}$.

We also note that in practical computations we do not see the square complexity w.r.t. d, like we have in the complexity estimate (15). The reason is that QTT ranks change on every step of the QTT procedure and the actual computation time is smaller that is prescribed by the estimate with the maximum rank bound.

5.2.4. Accuracy. To verify the accuracy of Algorithm 4 we compare its output with the results given by FFTW library. This can be done for $n \leqslant 2^{30}$. The accuracy parameter of Algorithm 4 was set to ε/d, where ε is the desired accuracy of the result, by Theorem 2. The results shown is Table 3, where we see that FTT-QTT algorithm holds the desired accuracy in our experiments (maximum error over all tested examples is shown). We also can compute the accuracy of (25), i.e. the distance between the integral Fourier transform and the DFT. This is done by pointwise comparison on $n = 2^d$ grid points for $d \leqslant 30$ and on 2^{30} points randomly selected in $[0 : 8]$ subset. The relative error in the Frobenius norm is given in Table 3 by "acc" lines and can be compared with the

Table 3. Accuracy verification for the FFT-QTT Algorithm 4 and $f = \mathrm{tri}(t)$. The $\varepsilon = 10^{-12}$ is desired accuracy, h is grid size, $\mathtt{err}$ is accuracy of FFT-QTT verified against FFTW library, $\mathtt{acc}$ denotes the relative accuracy of (25) in Frobenius norm

d	20	22	24	26	28	30	40	50	60
h	$1_{10}-3$	$5_{10}-4$	$2_{10}-4$	$1_{10}-4$	$6_{10}-5$	$3_{10}-5$	$1_{10}-6$	$3_{10}-8$	$9_{10}-10$
$\mathtt{acc}\,\mathrm{tri}(t)$	$9_{10}-6$	$3_{10}-6$	$1_{10}-6$	$4_{10}-7$	$1_{10}-7$	$4_{10}-8$	$2_{10}-10$	$2_{10}-11$	$2_{10}-12$
$\mathtt{acc}\,s(t)$	$6_{10}-6$	$2_{10}-6$	$8_{10}-7$	$3_{10}-7$	$1_{10}-7$	$5_{10}-8$	$5_{10}-10$	$4_{10}-12$	$2_{10}-12$
$\mathtt{err}/\varepsilon$	0.30	0.40	0.45	0.43	0.40	0.41	—	—	—

grid size h. We note that the use of FFT-QTT algorithm allows us to use finer grids and to compute the Fourier integral more accurately that by standard FFT algorithm.

5.3. Fourier images in 3D

In larger dimensions the one-dimensional size of the array that can be processed by full-sized FFT algorithm is severely restricted by computing resources and the discretisation accuracy is usually very low. As well as in 1D case, the use of data-structured FFT-QTT algorithm almostly removes the grid size restrictions and allows to reach higher accuracy of the computation. As an example, we compute the following m–dimensional Fourier image from [30]

$$f(x) = \begin{cases} (1-|x|)^\delta, & \text{if} \quad |x| \leqslant 1, \\ 0 & \text{otherwise} \end{cases}, \qquad \hat{f}(\xi) = \int_{\mathbb{R}^m} f(x) e^{-2\pi i x\cdot\xi} d\xi = \frac{\Gamma(\delta+1) J_{m/2+\delta}(2\pi|\xi|)}{\pi^\delta |\xi|^{m/2+\delta}}, \tag{27}$$

where Γ is gamma-function and J_α is Bessel function of the first kind of order α. Like in the 1D case, we introduce a uniform space and frequency domain grids with equal step sizes $h_x = h_\xi = 1/\sqrt{n}$, where $n = 2^d$ is one-dimensional grid size and approximate (27) by rectangle quadrature rule, that can be computed by m-dimensional DFT. The accuracy is computed by comparing the DFT $\hat{f} = (\underbrace{F_d \times \ldots \times F_d}_{m \text{ times}})f$ with analytical answer (27) on the grid points in area $[0:8]^d$.

5.4. Convolution in 3D

In scientific computing Fourier transforms are used often for the computation of convolutions. As a motivating example, consider the evaluation of the Hartree potential,

$$V_H[f](x) = \int_{\mathbb{R}^3} \frac{f(y)}{\|x-y\|} dy, \tag{28}$$

that plays an important role in electronic structure calculations using the Hartree-Fock equations. In many chemical modelling programs, e.g. PC GAMESS and MOLPRO,

Table 4. Relative accuracy of 3D Fourier image (27) computed by the DFT

	FFTW	FFT-QTT		
δ	$d=8$	$d=8$	$d=10$	$d=12$
0.5	$2_{10}{-3}$	$2_{10}{-3}$	$3_{10}{-4}$	$7_{10}{-5}$
1.5	$8_{10}{-5}$	$8_{10}{-5}$	$7_{10}{-6}$	$5_{10}{-7}$
2.5	$9_{10}{-6}$	$9_{10}{-6}$	$3_{10}{-7}$	$1_{10}{-8}$
3.5	$8_{10}{-7}$	$8_{10}{-7}$	$1_{10}{-8}$	$2_{10}{-10}$

the *electron density* function $f(y)$ is represented as a sum of polynomially weighted 3D Gaussians, for which (28) is evaluated analytically. For larger molecules, the proper choice of Gaussian-type orbitals (GTO) basis requires significant efforts, and sometimes computations become infeasible due to the huge number of basis element involved. The possible alternative may be the use $n \times n \times n$ tensor grids and Galerkin/collocation schemes using standard piecewise–constant or –linear basis elements. Since the use of tensor-structured methods leads to the almost linear or even logarithmic complexity w.r.t. n, very fine grids and high accuracy of computations is possible without a special choice of basis. For recent progress in this field see [31, 24, 32, 33, 25, 26, 27].

The electron density function has a strong cusps in the positions of the nuclei, that motivates the use of very precise grids. Using the piecewise-constant elements on $n \times n \times n$ uniform tensor grid, we compute the Hartree potential (28) for the Gaussian charge density in a box $[-0.5 : 0.5]^3$ and compare it with the analytical answer,

$$g_{a,\sigma}(y) = \frac{1}{(\sqrt{2\pi}\sigma)^3} \exp\left(-\frac{|y-a|^2}{2\sigma^2}\right), \qquad V_H[g_{a,\sigma}](x) = \frac{1}{4\pi|x-a|} \operatorname{erf}\left(\frac{|x-a|}{\sqrt{2}\sigma}\right), \quad (29)$$

where a and σ give the center and the width of Gaussian function and $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x \exp(-t^2)dt$ is error–function.

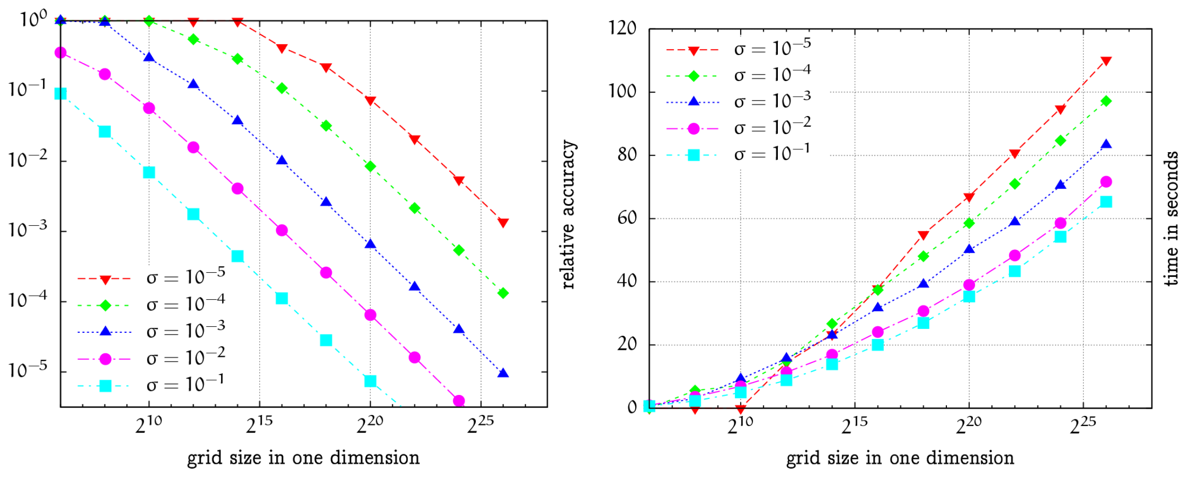
The discrete convolution writes as a multiplication by 3-level Toeplitz matrix,

$$f = Tg, \qquad \text{i.e.} \qquad f_{ijk} = h^3 \sum_{pqs=-n}^{n} \frac{g_{a,\sigma}(y_{pqs})}{\|x_{ijk} - y_{pqs}\|},$$

$$\begin{bmatrix} f \\ * \end{bmatrix} = C \begin{bmatrix} g \\ 0 \end{bmatrix}, \qquad C = \frac{1}{(2n)^3}(F \times F \times F)\Lambda(F \times F \times F), \quad \Lambda = \operatorname{diag}((F \times F \times F)c),$$

where C is three-level circulant $(2n)^3 \times (2n)^3$ matrix defined by the coefficients of T, c is first column of C and F is Fourier $2n \times 2n$ matrix. Thus, 3D convolution can be rendered via three FFTs and a Hadamard multiplication of Fourier images. For $n = 2^d$ with $d \geqslant 9$ these operations require large memory and computation resources and are not feasible for full-sized vectors using a standard FFT algorithm. For such grids we perform the computations in the QTT format using QTT-FFT algorithm. The QTT representation for a Gaussian function is constructed as a Kronecker product of

Figure 2. Accuracy and time of 3D convolution of Hartree potential with the Gaussian charge density



one-dimensional Gaussians, for which we can use full-to-TT algorithm from [15]. The QTT representation for a Newton potential is constructed using the *sinc*–quadrature from [34], that represents $^1/_r$ by a sum of Gaussians.

Accuracy and timings of the evaluation of (29) using 3D convolution are shown at Fig. 2. We put Gaussians in a center of the $[0:1]^3$ box and estimate the relative l_2 error of the Hartree potential on the subset $[0:1] \times \{^1/_2\} \times \{^1/_2\}$. Gaussian charges with smaller σ represent stronger cusps. We see that the precise evaluation of the Hartree potential of such functions on uniform tensor product grids requires larger grid sizes and can be feasible only using QTT approximation. Timings for the evaluation of 3D convolution using the FFT-QTT algorithm are very moderate and scale logarithmically w.r.t. d, see Fig. 2(right). Fast tensor-product convolution with the Newton kernel based on canonical and Tucker tensor decompositions were also considered and applied in electronic structure calculations in [27, 25, 26]. These methods have $\mathcal{O}(n \log n)$ complexity.

The multidimensional convolution of QTT-structured data can be computed also directly using the analytical QTT decomposition of a multilevel Toeplitz matrix [35].

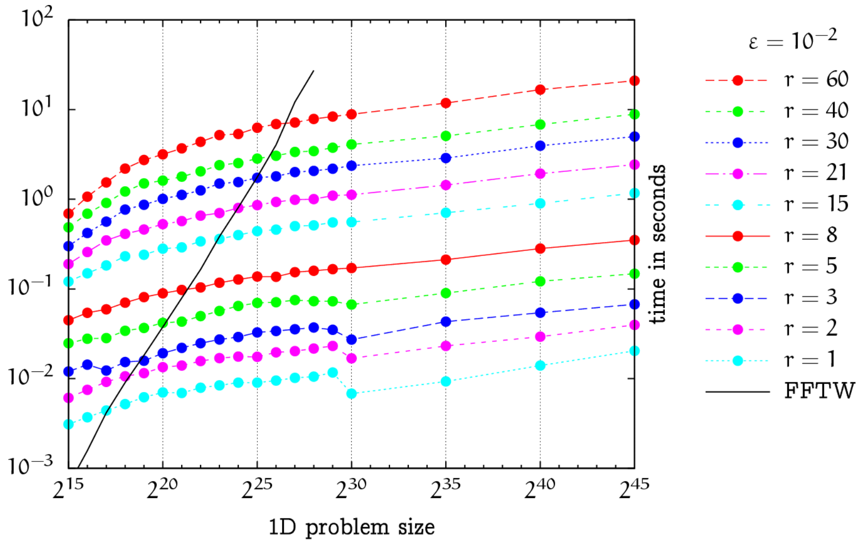
5.5. Plane waves and Sparse Fourier transform

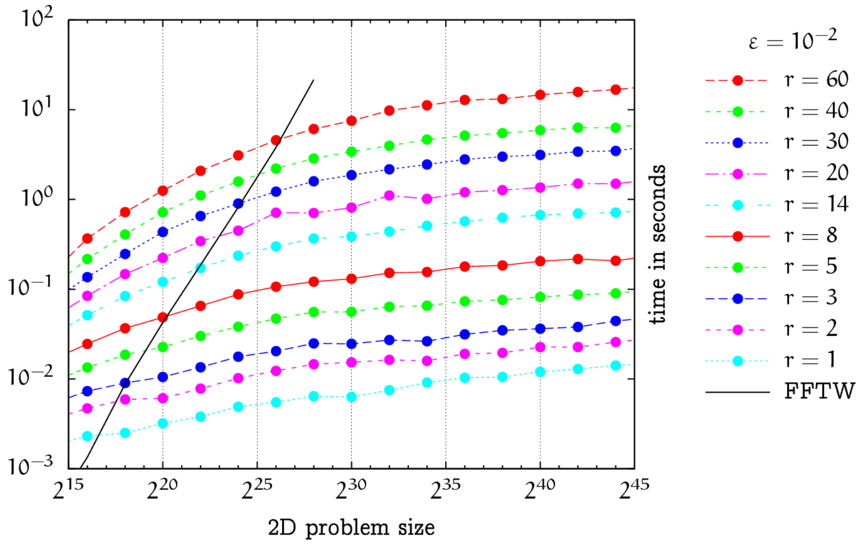
We consider a sum of r plane waves in m–dimensional space,

$$f(x) = \sum_{p=1}^{r} a_p \exp(2\pi i f_p \cdot x), \qquad f_p \in [0:1)^m, \quad x \in [0:2^d)^m, \tag{30}$$

where $f_p \cdot x$ denotes a scalar product. On the uniform tensor grid $\{x_k\} = \{x_{k_1} \times x_{k_2} \times \ldots \times x_{k_m}\}$ these functions have the QTT representation with rank bounded by r, that is constructed via (3). We may be interested to compute the m–dimensional Fourier transform of (30) to estimate frequencies f_p from the "measured" data $f(x_k)$.

Table 5. FFT-QTT and FFTW time versus problem size





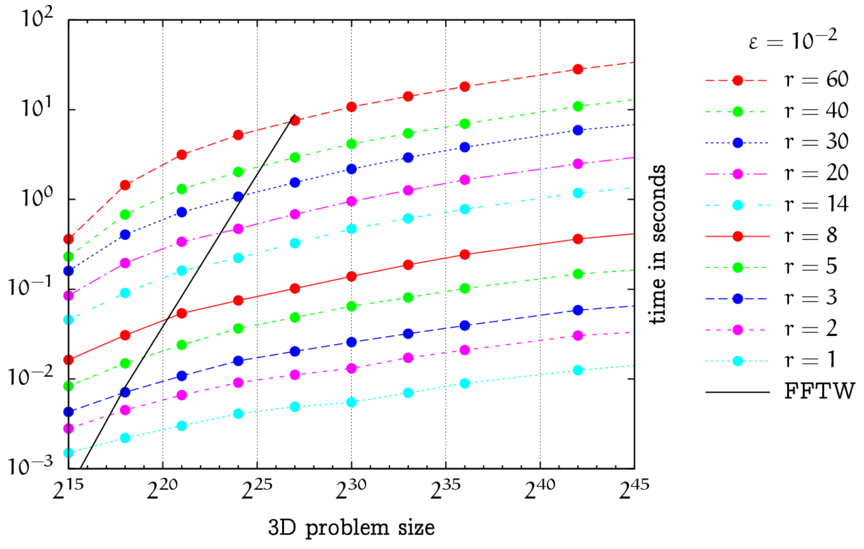
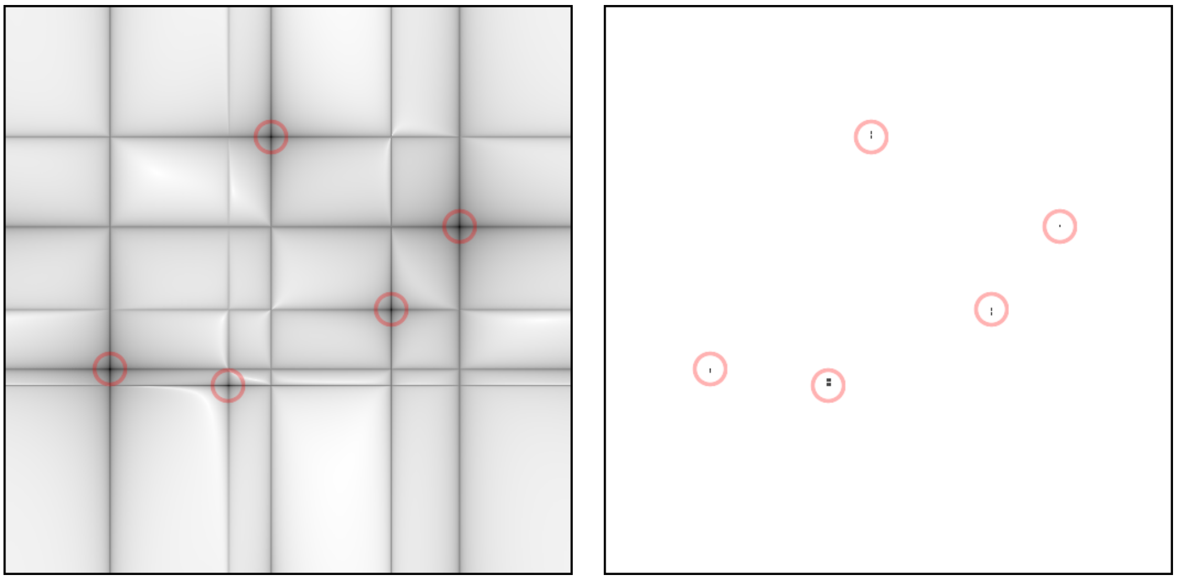


Figure 3. DFT image of 2D data set with $p = 5$ exponentials on $2^d \times 2^d$ grid, $d = 9$. Logarithm of amplitude is shown in 256 levels of gray (black is maximum, white is minimum). From the right, the image after sparsification in QTT format is shown



We define the array of grid values $f(x_k)$ as $f(k_1, \ldots, k_d) = f(x_{k_1}, \ldots, x_{k_d})$, using the same symbol f. If $f_p = j_p/n$ is integer multiple of $^1\!/_n$, the DFT reads

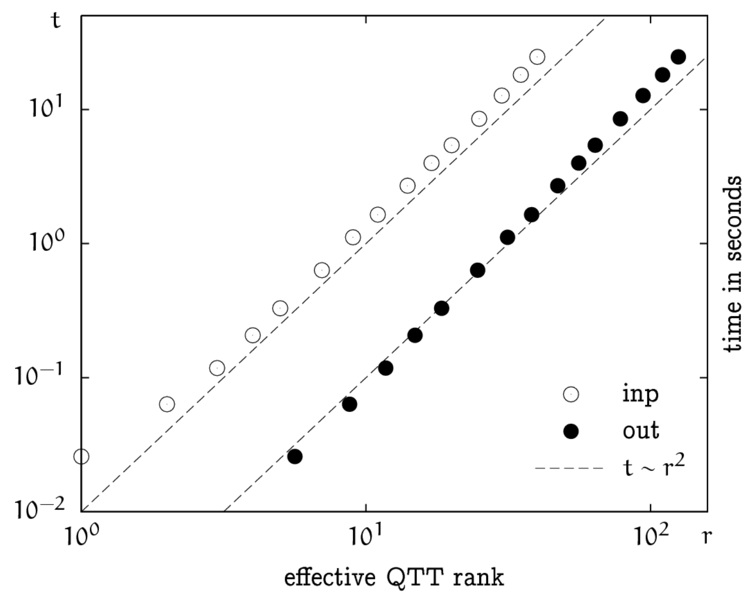
$$\hat{f} = \underbrace{(F_d \times \ldots \times F_d)}_{m \text{ times}} f = n^{md} \sum_{p=1}^{r} a_p e_{j_p},$$

where vector e_j has all zero elements but one unit in position $j = (j_1, \ldots, j_m)$. For frequencies, that are not integer multiple of $^1\!/_n$ the DFT image of each term from (30) is not a single point. The example of DFT image for the sum of five exponentials with random amplitudes a_p and frequencies f_p is shown on Figure 3 from the left.

5.5.1. Time versus rank. Timings for 1D, 2D and 3D Fourier transforms for different number of plane waves r and different problem sizes $n = 2^{md}$ are given on graphs on Fig. 5. We compare them with timings of full-size FFT and can easily see the cross points for different r, i.e. the problem size where the FFT-QTT becomes faster than FFTW. It depends on the QTT ranks, that grow with accuracy ε and number of exponentials r.

Using different r, we can check the dependence of time on the rank of QTT format. The results are given at Fig. 4 and show that instead of the cubic dependence, given by (15), the computational time in real experiments grow almost quadratically w.r.t. effective QTT ranks of both input and output arrays. Again, this is an example where the actual computational time has slower asymptotics than the upper bound (15).

Figure 4. Time of FFT-QTT w.r.t. effective QTT ranks of input (white circles) and output (black circles) arrays for functions given by the sum of exponentials (30). Dashed lines show quadratic dependence $t \sim r^2$. Accuracy $\varepsilon = 10^{-6}$.



5.5.2. Comparison with Sparse Fourier transform. The QTT representation is one of *data-sparse* representations, i.e. with number of representation parameters asymptotically smaller than the full number of array elements. If necessary, the number of parameters can be reduced by the TT-SVD approximation Algorithm 3 using the accuracy or maximum rank criterion. Therefore, the QTT representation can be an efficient tool for the data analysis of discrete signals with small number of frequencies. In the framework of Sparse Fourier transform a sparse representation of a discrete signal by r-term Fourier sum is seen. The randomized heuristic algorithm RAlSFA proposed in [29] finds a near-optimal r-term Sparse representation for a signal on length n with probability δ and quasi-optimality parameter α in time $\text{poly}(r)\log(n)\log(1/\delta)/\alpha^2$, where the polynomial $\text{poly}(r)$ is empirically found to be quadratic. Note, that any Sparse r-term Fourier representation belongs to the class of low-rank QTT representations since it has QTT ranks r. By the Theorem 4 and Corollary 1 to the Theorem 5 the complexity for the FFT-QTT transform for such array is $\mathcal{O}(r^3(\log n)^2)$, that overcomes the RAlSFA complexity by $r \log n$ factor. The crossover point between RAlSFA and FFTW algorithms for $r = 8$ and some δ and α is reported to be $n \simeq 70000$ for 1D and $n \simeq 900$ for 2D transforms and is estimated as $n \simeq 210$ for 3D transforms. For $\varepsilon = 10^{-2}$ and $r = 8$ the crossover point between FFT-QTT and FFTW is $d \simeq 21$ for 1D problems, $2d \simeq 20$ for 2D problems and $3d \simeq 20$ for 3D problems, see Fig. 5. That corresponds to $n \simeq 2 \cdot 10^6$, $n \simeq 1024$ and $n \simeq 100$ respectively. We see that the FFT-QTT is slower than RAlSFA for 1D problems, has almost the same speed for 2D problems and outperforms it in higher dimensions. The other advance of FFT-QTT is that it does not require any choice of parameters, while the parameter tune in RAlSFA is quite tricky.

Finally, we can discuss a method to convert a *data-sparse* QTT format to a *pointwise-sparse* format, with actually small number of non-zero entries. The accurate solution of this problem requires to develop a rank-truncation algorithm for the TT format that in governed by Chebyshëv (pointwise) norm. This is a non-trivial problem and we postpone it for future. Instead, we apply a very naive algorithm that sparsifies the TT cores of (1) *independently*, using the threshold parameter μ as follows

$$A^{(p)}_{k_p} \to \tilde{A}^{(p)}_{k_p}, \qquad \tilde{A}^{(p)}_{k_p}(i,j) = \begin{cases} A^{(p)}_{k_p}(i,j), & \text{if} \quad |A^{(p)}_{k_p}(i,j)| \geqslant \mu |A^p|, \\ 0, & \text{elsewhere}, \end{cases} \tag{31}$$

where $|A_p| = \max_{k_p} \max_{i,j} |A^{(p)}_{k_p}(i,j)|$. The sparsified DFT image for $\mu = 0.4$ is show on Fig. 3 from the right. It shows correct positions of all five plane wave frequencies for this example.

6. Conclusion and future work

We propose algorithms for the Fourier transform of multidimensional vectors in the QTT format. The m–dimensional Fourier transform of an $n \times \ldots \times n$ m-tensor with $n = 2^d$ has $\log^2(\text{volume}) = \mathcal{O}(m^2 d^2 r^3)$ complexity where r depends on the complexity of QTT structure. The storage size is bounded by $\mathcal{O}(mdr^2)$. For vectors with moderate r and large n and m the proposed algorithm outperforms the $\mathcal{O}(n^m \log n)$ FFT algorithm of almost volume complexity. Using QTT format, the 1D FFT transform of size $n = 2^d$ for vectors with QTT ranks $r \lesssim 10$ can be computed for $n \leqslant 2^{60}$ in less than second. The 3D FFT transform for $n \times n \times n$ array with about the same QTT ranks can be computed for $n \leqslant 2^{20}$ in one second. This allows to use very fine grids and accurately compute the Hartree potential in a box $[0 : 1]^3$ for the Gaussian functions, $e^{-\frac{x^2}{2\sigma^2}}$, with strong cusps and width in the range $\sigma = 10^{-1} \div 10^{-5}$ by 3D convolution. Also, our approach applies to the multidimensional functions with small number of Fourier coefficients, i.e. the sum of few plane waves. Using the proposed method we compute the Fourier images in 1D, 2D and 3D and show that our method is competitive with the Sparse Fourier transform, especially for dimensions two and three. Notice that in the case $m > 3$ the computation of high–resolution FFT and convolution transforms is practically infeasible without the use of QTT approximation.

The proposed method can be applied in various scientific computing solvers working with multidimensional data in data-sparse tensor formats. It can have also an application to image and video processing, but in this direction we should think about the WTT version of the algorithm, following the ideas from [19]. The efficient application to Sparse Fourier transform requires to develop the TT truncation method in l_1 norm. The application of the proposed method to exponential fitting will be considered in a forthcoming work. It is also might be interesting to establish links and compare between the fast Fourier transform on hyperbolic cross points [28] and QTT-FFT or QTT cross interpolation schemes [36].

References

[1] *de Lathauwer L.* A survey of tensor methods // *IEEE International Symposium on Circuits and Systems*. 2009, May. — P. 2773-2776.

[2] *Kolda T. G., Bader B. W.* Tensor Decompositions and Applications // *SIAM Review*. 2009. V. 51, № 3. P. 455–500. doi: 10.1137/07070111X.

[3] *Khoromskij B. N.* Tensors-structured numerical methods in scientific computing: Survey on recent advances: Preprint 21. — Leipzig: MPI MIS, 2010. www.mis.mpg.de/preprints/2010/preprint2010_21.pdf.

[4] *Oseledets I. V., Tyrtyshnikov E. E.* Breaking the curse of dimensionality, or how to use SVD in many dimensions // *SIAM J. Sci. Comp.* 2009. V. 31, № 5. P. 3744-3759. doi: 10.1137/090748330.

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

[6] *Hitchcock F. L.* The expression of a tensor or a polyadic as a sum of products // *J. Math. Phys.* 1927. V. 6, № 1. P. 164–189.

[7] *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.

[8] *Caroll 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. doi: 10.1007/BF02310791.

[9] *Tucker L. R.* Some mathematical notes on three-mode factor analysis // *Psychometrika*. 1966. V. 31. P. 279-311. doi: 10.1007/BF02289464.

[10] *Khoromskij B. N.* $\mathcal{O}(\mathrm{d}\log\mathrm{N})$–Quantics approximation of N–d tensors in high-dimensional numerical modeling // *J. Constr. Appr.* (2011). To appear. www.mis.mpg.de/preprints/2009/preprint2009_55.pdf.

[11] *Gauss C. F.* Nachlass: Theoria interpolationis methodo nova tractata // Werke. — Königliche Gesellschaft der Wissenschaften, Göttingem, 1866. — V. 3. — P. 265-330.

[12] *Cooley J. W., Tuckey J. W.* An algorithm for the machine calculation of complex Fourier series // *Math. Comput.* 1965. V. 19. P. 297-301. doi: 10.2307/2003354.

[13] *White S. R.* Density matrix formulation for quantum renormalization groups // *Physical Review Letters*. 1992. V. 69, № 19. P. 2863-2866. doi: 10.1103/PhysRevLett.69.2863.

[14] *Oseledets I.* A new tensor decomposition // *Doklady Math.* 2009. V. 80, № 1. P. 495-496. doi: 10.1134/S1064562409040115.

[15] *Oseledets I. V.* Approximation of $2^d \times 2^d$ matrices using tensor decomposition // *SIAM J. Matrix Anal. Appl.* 2010. V. 31, № 4. P. 2130-2145. doi: 10.1137/090757861.

[16] *Grasedyck L.* Polynomial approximation in hierarchical Tucker format by vector-tensorization: DFG-SPP1324 Preprint 43. — Marburg: Philipps-Univ., 2010. http://www.dfg-spp1324.de/download/preprints/preprint043.pdf.

[17] *Oseledets I.* Constructive representation of functions in tensor formats: Preprint 2010-04. — Moscow: INM RAS, 2010. http://pub.inm.ras.ru.

[18] *Kazeev V., Khoromskij B. N.* Explicit low-rank QTT representation of Laplace operator and its inverse: Preprint 75. — Leipzig: MPI MIS, 2010. www.mis.mpg.de/preprints/2010/preprint2010_75.pdf.

[19] *Oseledets I. V.* Algebraic wavelet transform via quantics tensor train decomposition: Preprint 2010-03. — Moscow: INM RAS, 2010. http://pub.inm.ras.ru.

[20] *Khoromskij B. N., Oseledets I. V.* Quantics-TT approximation of elliptic solution operators in high dimensions // *Rus. J. Numer. Math. Math. Phys.* (2011). To appear.

[21] *Khoromskij B. N., Oseledets I. V.* DMRG + QTT approach to high-dimensional quantum molecular dynamics: Preprint 68. — Leipzig: MPI MIS, 2010. www.mis.mpg.de/preprints/2010/preprint2010_68.pdf.

[22] *Dolgov S., Khoromskij B., Oseledets I., Tyrtyshnikov E.* Tensor structured iterative solution of elliptic problems with jumping coefficients: Preprint 55. — Leipzig: MPI MIS, 2010. www.mis.mpg.de/preprints/2010/preprint2010_55.pdf.

[23] *Hackbusch W.* Tensorisation of vectors and their efficient convolution: DFG-SPP1324 Preprint 65. — Marburg: Philipps-Univ., 2010. http://www.dfg-spp1324.de/download/preprints/preprint065.pdf.

[24] *Flad H.-J., Khoromskij B. N., Savostyanov D. V., Tyrtyshnikov E. E.* Verification of the cross 3D algorithm on quantum chemistry data // *Rus. J. Numer. Anal. Math. Model.* 2008. V. 23, № 4. P. 329-344. doi: 10.1515/RJNAMM.2008.020.

[25] *Khoromskij B. N.* Fast and accurate tensor approximation of multivariate convolution with linear scaling in dimension // *J. Comp. Appl. Math.* 2010. V. 234, № 11. P. 3122-3139. doi: 10.1016/j.cam.2010.02.004.

[26] *Khoromskij B. N., Khoromskaia V., Flad. H.-J.* Numerical solution of the Hartree–Fock equation in multilevel tensor-structured format // *SIAM J. Sci. Comp.* 2011. V. 33, № 1. P. 45-65. doi: 10.1137/090777372.

[27] *Khoromskaia V.* Numerical Solution of the Hartree-Fock Equation by Multilevel Tensor-structured methods: Ph.D. thesis / TU Berlin. — 2010. http://opus.kobv.de/tuberlin/volltexte/2011/2948/.

[28] *Fenn M., Kunis S., Potts D.* Fast evaluation of trigonometric polynomials from hyperbolic crosses // *Num. Algorithms*. 2006. V. 41. P. 339-352. doi: 10.1007/s11075-006-9017-7.

[29] *Zou J., Gilbert A., Strauss M., Daubechies I.* Theoretical and experimental analysis of a randomized algorithm for Sparse Fourier transform analysis // *J. Comp. Phys.* 2006. V. 211. P. 572-595. doi: 10.1016/j.jcp.2005.06.005.

[30] *Stein E., Weiss G.* Introduction to Fourier Analysis on Euclidean Spaces. — Princenton, N.J.: Princeton University Press, 1971.

[31] *Khoromskij B. N.* On tensor approximation of Green iterations for Kohn-Sham equations // *Computing and visualization in science*. 2008. V. 11, №4-6. P. 259-271. doi: 10.1007/s00791-008-0097-x.

[32] *Chinnamsetty S. R., Flad H.-J., Khoromskaia V., Khoromskij B. N.* Tensor decomposition in electronic structure calculations on 3D Cartesian grids // *J. Comp. Phys.* 2009. V. 228, № 16. P. 5749-5762. doi: 10.1016/j.jcp.2009.04.043.

[33] *Oseledets I. V., Savostyanov D. V., Tyrtyshnikov E. E.* Cross approximation in tensor electron density computations // *Numer. Lin. Alg. Appl.* 2010. V. 17, № 6. P. 935-952. doi: 10.1002/nla.682.

[34] *Hackbusch W., Khoromskij B. N.* Low-rank Kronecker-product approximation to multi-dimensional nonlocal operators. Part 1. Separable approximation of multivariate functions // *Computing*. 2006. V. 76, №3-4. P. 177-202. doi: 10.1007/s00607-005-0144-0.

[35] *Kazeev V., Khoromskij B. N., Tyrtyshnikov E. E.* Tensor structure of multilevel Toeplitz–circulant matrices leads to convolution with logarithmic complexity: Preprint. — Leipzig: MPI MIS, 2011. — In preparation.

[36] *Oseledets I. V., Tyrtyshnikov E. E.* TT-cross algorithm for the approximation of multidimensional arrays // *Linear Algebra Appl.* 2010. V. 432, № 1. P. 70-88. doi: 10.1016/j.laa.2009.07.024.

[37] *Ahmed N., Natarajan T., Rao K. R.* Discrete cosine transform // *IEEE Trans. Computers*. 1974. V. C-23, № 1. P. 90-93. doi: 10.1109/T-C.1974.223784.

[38] *Ahmed N., Rao K. R.* Orthogonal transforms for digital signal processing. — Berlin and New York: Springer-Verlag, 1975.

[39] *Makhoul J.* A fast cosine transform in one and two dimensions // *IEEE Trans. Acoustics, Speech and Signal Processing*. 1980. V. 28, № 1. P. 27-34. doi: 10.1109/TASSP.1980.1163351.

[40] *Lee B. G.* A new algorithm to compute the discrete cosine transform // *IEEE Trans. Acoustics, Speech and Signal Processing*. 1984. V. ASSP-32, № 6. P. 1243-1245. doi: 10.1109/TASSP.1984.1164443.

[41] *Yip P., Rao K. R.* Fast decimation-in-time algorithms for a family of discrete sine and cosine transforms // *Circuits Systems Signal Process*. 1984. V. 3, № 4. P. 387-408. doi: 10.1007/BF01599167.

[42] *Strang G.* The discrete cosine transform // *SIAM Review*. 1999. V. 41, № 1. P. 135-147.

[43] *Yip P., Rao K. R.* On the computation and the effectiveness of discrete sine transform // *Computers and Electrical Engineering*. 1980. V. 7, № 1. P. 45-55. doi: 10.1016/0045-7906(80)90018-X.

[44] *Wang Z.* Fast algorithms for the discrete W transform and for the discrete Fourier transform // *IEEE Trans. Acoustics, Speech and Signal Processing*. 1984. V. 32, № 4. P. 803-816. doi: 10.1109/TASSP.1984.1164399.

[45] *Martucci S. A.* Symmetric convolution and the discrete sine and cosine transforms // *IEEE Trans. Sig. Proc.* 1994. V. 42, № 5. P. 1038-1051. doi: 10.1109/78.295213.

A. Discrete cosine and sine transforms in QTT format

A.1. Introduction

The discrete cosine transform (DCT), proposed in 1974 by N. Ahmed [37], is a power tool for signal and image processing [38]. It can be computed using real-to-complex Fourier transform [37, 39] or with purely real-valued arithmetics using the radix-2 recurrence [40, 41]. More details can be found in a nice review by G. Strang [42]. Also, a discrete sine transform (DST) was proposed by K. R. Rao [43] and a systematic factorisation method was developed by Z. Wang [44], that leads to fast real-valued algorithms for the whole family of discrete trigonometric transforms. Sixteen trigonometric transforms (sine or cosine, odd or even, type 1, 2, 3 or 4) are usually considered in literature and eight are usually implemented in numerical libraries [45]. We consider only even transforms

$$y(j) = \sum_{k=0}^{n-1} x(k) \cos \frac{\pi(j+j_0)(k+k_0)}{n}, \qquad z(j) = \sum_{k=0}^{n-1} x(k) \sin \frac{\pi(j+j_0)(k+k_0)}{n}, \qquad (32)$$

where j_0 and k_0 define the particular transform and $j, k = 0, \dots, n-1$. This form differs slightly from the common definition: we omit the factor $1/2$ for the $j = 0$ coefficient of DCT-1 and DST-3 transform and indices j, k run as $j, k = 0, \dots, n$ for DCT-1 and $j, k = 1, \dots, n-1$ for DST-1. In the QTT format, these corrections require the rank-one update of the result and can be implemented easily. However, in this section we give only the basic idea how implement (32) in the QTT format, using real-to-complex Fourier transform or purely real-valued arithmetics, and the detailed algorithm for different types of sine and cosine transforms will be reported elsewhere.

A.2. DCT and DST using real-to-complex and complex-to-real transforms

The cosine/sine transforms (32) can be written as real/imaginary part of complex vector

$$y(j) = \Re f(j), \quad z(j) = -\Im f(j), \qquad f(j) = \sum_{k=0}^{n-1} x(k) \exp \frac{-\pi i (j + j_0)(k + k_0)}{n},$$

that can be computed using the Fourier transform as follows

$$f = \exp\left(\frac{-\pi i j_0 k_0}{n}\right) D_{k_0} I_{2n\times n}^T F I_{2n\times n} D_{j_0} x, \qquad D_p = \operatorname{diag}\left\{\exp\left(\frac{-\pi i p q}{n}\right)\right\}_{q=0}^{n-1},$$

where F is $2n \times 2n$ Fourier matrix and $I_{2n\times n}$ consists of the first n columns of $2n \times 2n$ identity matrix. Suppose $n = 2^d$ and x is given in the QTT format (10). The pre-multiplication by the diagonal matrix D_{j_0} does not change the QTT ranks, since it is the Hadamard multiplication by the exponential vector (3) with QTT-ranks one. The application of $I_{2n\times n}$ prolongates the vector $D_{j_0} x$ with $n = 2^d$ zeroes, adding one extra core to the QTT format. If x is given by (10) then $\hat{x} = I_{2n\times n} D_{j_0} x$ reads

$$\hat{x}(\overline{k_1 \dots k_d k_{d+1}}) = \hat{X}^{(1)}_{k_1} \dots \hat{X}^{(d)}_{k_d} (1 - k_{d+1}), \qquad \hat{X}^{(p)}_{k_p} = X^{(p)}_{k_p} \exp\left(\frac{-\pi i j_0 k_p}{2^{d-p+1}}\right).$$

Then we apply $F = F_{d+1}$ using Alg. 4, select the first half-vector (and remove the last core of QTT format) by choosing $j_{d+1} = 0$, multiply again by the twiddle factors D_{k_0} and result in vector f. Finally, we are to explain how to select real and imaginary part of the complex-valued vector in the QTT format. We need the following theorem.

Theorem 6. The complex-valued tensor train $a(j_1, \dots, j_d) = A^{(1)}_{j_1} \dots A^{(d)}_{j_d}$, with ranks $r_0, r_1, \dots, r_{d-1}, r_d$ can be represented as tensor train $a(j_1, \dots, j_d) = \hat{A}^{(1)}_{j_1} \dots \hat{A}^{(d)}_{j_d}$, with ranks $r_0, 2r_1, \dots, 2r_{d-1}, r_d$, where the cores $\hat{A}^{(p)}_{j_p}$, $p = 1, \dots, d-1$ are real-valued,

$$\begin{gathered}\hat{A}^{(1)}_{j_1} = \begin{bmatrix} \Re A^{(1)}_{j_1} & \Im A^{(1)}_{j_1} \end{bmatrix}, \quad \hat{A}^{(p)}_{j_p} = \begin{bmatrix} \Re A^{(p)}_{j_p} & \Im A^{(p)}_{j_p} \\ -\Im A^{(p)}_{j_p} & \Re A^{(p)}_{j_p}. \end{bmatrix}, \qquad p = 2, \dots, d-1, \\ \hat{A}^{(d)}_{j_d} = \begin{bmatrix} \Re A^{(d)}_{j_d} \\ -\Im A^{(d)}_{j_d} \end{bmatrix} + i \begin{bmatrix} \Im A^{(d)}_{j_d} \\ \Re A^{(d)}_{j_d} \end{bmatrix}\end{gathered} \tag{33}$$

Proof. The proof is constructive and provides an algorithm for such representation. Let $A^{(p)}_{j_p} = B^{(p)}_{j_p} + i C^{(p)}_{j_p}$, with real-valued $B^{(p)}_{j_p} = \Re A^{(p)}_{j_p}$ and $C^{(p)}_{j_p} = \Im A^{(p)}_{j_p}$. Then

$$A^{(1)}_{j_1} = \begin{bmatrix} B^{(1)}_{j_1} & C^{(1)}_{j_1} \end{bmatrix} \begin{bmatrix} I \\ iI \end{bmatrix},$$

where I is an identity $r_1 \times r_1$ matrix. Define real-valued core $\hat{A}^{(1)}_{j_1} = \begin{bmatrix} B^{(1)}_{j_1} & C^{(1)}_{j_1} \end{bmatrix}$ and multiply $\begin{bmatrix} I & iI \end{bmatrix}^T$ to the right, like we do in the orthogonalisation Algorithm 2.

$$\begin{bmatrix} I \\ iI \end{bmatrix} A^{(2)}_{j_2} = \begin{bmatrix} I \\ iI \end{bmatrix} \left(B^{(2)}_{j_2} + iC^{(2)}_{j_2} \right) = \begin{bmatrix} B^{(2)}_{j_2} + iC^{(2)}_{j_2} \\ -C^{(2)}_{j_2} + iB^{(2)}_{j_2} \end{bmatrix} = \begin{bmatrix} B^{(2)}_{j_2} & C^{(2)}_{j_2} \\ -C^{(2)}_{j_2} & B^{(2)}_{j_2} \end{bmatrix} \begin{bmatrix} I \\ iI \end{bmatrix} = \hat{A}^{(2)}_{j_2} \begin{bmatrix} I \\ iI \end{bmatrix},$$

where in the right-hand side $\hat{A}^{(2)}_{j_2}$ is a new real-valued core and I is $r_2 \times r_2$ identity matrix. We continue the process and establish (33). The last core reads

$$\hat{A}^{(d)}_{j_d} = \begin{bmatrix} I \\ iI \end{bmatrix} A^{(d)}_{j_d} = \begin{bmatrix} I \\ iI \end{bmatrix} \left(B^{(d)}_{j_d} + iC^{(d)}_{j_d} \right) = \begin{bmatrix} B^{(d)}_{j_d} + iC^{(d)}_{j_d} \\ -C^{(d)}_{j_d} + iB^{(d)}_{j_d} \end{bmatrix} = \begin{bmatrix} B^{(d)}_{j_d} \\ -C^{(d)}_{j_d} \end{bmatrix} + i \begin{bmatrix} C^{(d)}_{j_d} \\ B^{(d)}_{j_d} \end{bmatrix}, \qquad (34)$$

that completes the proof. □

Corollary 2. The representation (33) allows to compute real and imaginary parts of a tensor train $a(j_1, \ldots, j_d)$,

$$\Re a(j_1, \ldots, j_d) = \left(\prod_{p=1}^{d-1} \hat{A}^{(p)}_{j_p} \right) \Re \hat{A}^{(d)}_{j_d}, \qquad \Im a(j_1, \ldots, j_d) = \left(\prod_{p=1}^{d-1} \hat{A}^{(p)}_{j_p} \right) \Im \hat{A}^{(d)}_{j_d}.$$

Example 1. Obtain the QTT representation for sine and cosine vectors by extracting the real and imaginary parts of (3). Each core of the exponential vector has the form

$$A^{(p)}_{j_p} = \exp(i 2^{p-1} \alpha j_p) = \cos 2^{p-1} \alpha j_p + i \sin 2^{p-1} \alpha j_p.$$

We use (33) and form real-valued cores $\hat{A}^{(1)}_{j_1} = \begin{bmatrix} \cos \alpha j_1 & \sin \alpha j_1 \end{bmatrix}$ and

$$\hat{A}^{(p)}_{j_p} = \begin{bmatrix} \cos 2^{p-1} \alpha j_p & \sin 2^{p-1} \alpha j_p \\ -\sin 2^{p-1} \alpha j_p & \cos 2^{p-1} \alpha j_p \end{bmatrix}, \qquad p = 2, \ldots, d-1.$$

Using (34), we result in

$$\begin{bmatrix} \cos \alpha j \\ \sin \alpha j \end{bmatrix}^T = \begin{bmatrix} \cos \alpha j_1 \\ \sin \alpha j_1 \end{bmatrix}^T \begin{bmatrix} \cos 2\alpha j_2 & \sin 2\alpha j_2 \\ -\sin 2\alpha j_2 & \cos 2\alpha j_2 \end{bmatrix} \cdots \begin{bmatrix} \cos 2^{d-1} \alpha j_d & \sin 2^{d-1} \alpha j_d \\ -\sin 2^{d-1} \alpha j_d & \cos 2^{d-1} \alpha j_d \end{bmatrix}. \qquad (35)$$

Therefore, the nice property of *simultaneous representation* of sine and cosine in the common QTT basis (i.e. all the cores except one are the same) with ranks 2 follows from the existence of the rank-one QTT representation for the exponential and the simultaneous representation of the real and imaginary part of complex tensor train, ensured by Theorem 6.

A.3. Sine and cosine radix-2 recurrence in QTT format

If we want to compute (32) using only real-valued arithmetics, we can follow the radix-2 scheme from Section 2. For $n = 2^d$ we use index transform (7) and write

$$\begin{aligned}\cos\frac{\pi(j+j_0)(k+k_0)}{2^d} &= \cos\frac{\pi(2J+j_0+j_1)(K+k_0+k_d2^{d-1})}{2^d} \\ &= \tilde{c}c_JCc'_K - \tilde{c}s_JSc'_K - \tilde{s}c_JCs'_K + \tilde{s}s_JSs'_K - \tilde{c}c_JSs'_K - \tilde{c}s_JCs'_K - \tilde{s}c_JSc'_K - \tilde{s}s_JCc'_K,\end{aligned}$$

where

$$\begin{gathered}\tilde{c} = \cos\frac{\pi j'k'}{2^d}, \quad c_J = \cos\frac{\pi Jk'}{2^{d-1}}, \quad C = \cos\frac{\pi JK}{2^{d-1}}, \quad c'_K = \cos\frac{\pi j'K}{2^d}, \\ \tilde{s} = \sin\frac{\pi j'k'}{2^d}, \quad s_J = \sin\frac{\pi Jk'}{2^{d-1}}, \quad S = \sin\frac{\pi JK}{2^{d-1}}, \quad s'_K = \cos\frac{\pi j'K}{2^d}, \\ j' = j_0 + j_1, \qquad k' = k_0 + k_d n/2.\end{gathered}$$

The same recurrence also writes for sine function. Therefore, the evaluation of sine and cosine transforms performs in four steps.

1. Hadamard multiplication of top and bottom parts of vector x by certain sine and cosine vectors,
2. half-size sine and cosine transforms with doubled frequency ω,
3. Hadamard multiplication of the odd and even elements of the result by another sine and cosine vectors,
4. summation of the result according to the recurrence relation.

Obviously, the same recurrence can be written for half-sized and all subsequent transforms. Since both sine and cosine vectors are of low QTT ranks (35), these steps can be implemented maintaining the QTT format of the vectors, like we do for FFT. Therefore, sine/cosine transforms can be efficiently implemented in QTT format, providing the ranks of the vectors in the procedure remain bounded.

After the first radix-2 steps the even DCT and DST writes through two basic transforms, cosine $y = C_dx$ and sine $z = S_dx$, defined as follows

$$y(j) = \sum_{k=0}^{n-1} x(k)\cos\frac{\pi jk}{n}, \qquad z(j) = \sum_{k=0}^{n-1} x(k)\sin\frac{\pi jk}{n}, \qquad n = 2^d. \tag{36}$$

In the following we restrict the discussion to these two transforms. Applying the radix-2 scheme, we write in the block form

$$\begin{aligned} P_dC_dx &= \begin{bmatrix} C_{d-1}x_{top} & + & J_1C_{d-1}x_{bot} \\ C_{d-1}(c\odot x_{top}) - S_{d-1}(s\odot x_{top}) & - & J_1C_{d-1}(s\odot x_{bot}) - J_1S_{d-1}(c\odot x_{bot}) \end{bmatrix}, \\ P_dS_dx &= \begin{bmatrix} S_{d-1}x_{top} & + & J_1S_{d-1}x_{bot} \\ C_{d-1}(s\odot x_{top}) + S_{d-1}(c\odot x_{top}) & + & J_1C_{d-1}(c\odot x_{bot}) - J_1S_{d-1}(s\odot x_{bot}) \end{bmatrix}, \end{aligned} \tag{37}$$

where $c = [\cos \frac{\pi k}{2^d}]_{k=0}^{2^{d-1}-1}$, $s = [\sin \frac{\pi k}{2^d}]_{k=0}^{2^{d-1}-1}$, and

$$J_1 = \operatorname{diag}(\underbrace{+1, -1, +1, -1, \dots +1 - 1}_{2^{d-1} \quad \text{elements}}). \tag{38}$$

To get rid of P_d, we multiply both sides by bit-reverse permutation R_{p-1} and since $R_{p-1}P_d = R_d$ and $R_{d-1}(J_1 \times I) = (J_{d-1} \times I)R_{d-1}$, where

$$J_{d-1} = \operatorname{diag}(\underbrace{+1, +1, \dots +1}_{2^{d-2} \text{ elements}} \ \underbrace{-1, -1, \dots -1}_{2^{d-2} \text{ elements}}),$$

we finally come to the following recurrence

$$\begin{aligned}
\tilde{C}_d x &= \begin{bmatrix} \tilde{C}_{d-1}x_{top} & + & J_{d-1}\tilde{C}_{d-1}x_{bot} \\ \tilde{C}_{d-1}(c \odot x_{top}) - \tilde{S}_{d-1}(s \odot x_{top}) & - & J_{d-1}\tilde{C}_{d-1}(s \odot x_{bot}) - J_{d-1}\tilde{S}_{d-1}(c \odot x_{bot}) \end{bmatrix}, \\
\tilde{S}_d x &= \begin{bmatrix} \tilde{S}_{d-1}x_{top} & + & J_{d-1}\tilde{S}_{d-1}x_{bot} \\ \tilde{C}_{d-1}(s \odot x_{top}) + \tilde{S}_{d-1}(c \odot x_{top}) & + & J_{d-1}\tilde{C}_{d-1}(c \odot x_{bot}) - J_{d-1}\tilde{S}_{d-1}(s \odot x_{bot}) \end{bmatrix},
\end{aligned} \tag{39}$$

where $\tilde{S}_p$ and $\tilde{C}_p$ are sine and cosine transforms of size 2^p with bit-reversed output. It is important to note that the final form of the recurrence can be implemented only by the operations with the two upper cores of tensor train (number $d-1$ and d), that considerably simplifies the algorithm.

The discrete sine and cosine transforms in QTT format are summarised in Algorithm 6. Basically, it consists of the two parts. First, we go down through the tensor train, detach upper cores and multiply the rest of the train by sine, cosine and identity factor as proposed by (37),(38). The QTT format for the multiples reads by (35)

$$\begin{bmatrix} 1 \\ \cos \frac{\pi k}{2^{d+1}} \\ \sin \frac{\pi k}{2^{d+1}} \end{bmatrix}^T = \begin{bmatrix} 1 \\ \cos \frac{\pi k_1}{2^{d+1}} \\ \sin \frac{\pi k_1}{2^{d+1}} \end{bmatrix}^T \begin{bmatrix} 1 & & \\ & \cos \frac{\pi k_2}{2^d} & \sin \frac{\pi k_2}{2^d} \\ & -\sin \frac{\pi k_2}{2^d} & \cos \frac{\pi k_2}{2^d} \end{bmatrix} \cdots \begin{bmatrix} 1 & & \\ & \cos \frac{\pi k_d}{2} & \sin \frac{\pi k_d}{2} \\ & -\sin \frac{\pi k_d}{2} & \cos \frac{\pi k_d}{2} \end{bmatrix}.$$

The Hadamard multiplication by these function is implemented on Line 4 of Algorithm 6.

Second, we go up through the tensor train and implement the recurrence (39). To explain Line 10, it is enough to mention that for all half-sized vectors in (39) the QTT representation with the same "lower-bit" structure $Q_{j_1}^{(1)}Q_{j_2}^{(2)} \dots Q_{j_{D-1}}^{(D-1)}$ exists at this moment. Therefore, for instance

$$\begin{array}{lll} Q_{j_1}^{(1)}Q_{j_2}^{(2)} \dots Q_{j_{D-1}}^{(D-1)}Y_{j_D}^{(D)}U_0^{(D+1)} & \text{represents} & (\tilde{C}_D x_{top})(\overline{j_1 \dots j_D}), \\ Q_{j_1}^{(1)}Q_{j_2}^{(2)} \dots Q_{j_{D-1}}^{(D-1)}Z_{j_D}^{(D)}V_1^{(D+1)} & \text{represents} & (\tilde{S}_D(c \odot x_{bot}))(\overline{j_1 \dots j_D}) \end{array}$$

and so on, where x_{top} and x_{bot} are top and bottom halves of the corresponding 2^{D+1} vector and c and s are appropriate sine and cosine multipliers, like we have in (38). Therefore, the recurrence (39) is implemented by proper combination of matrices $Y_{j_D}^{(D)}$ $Z_{j_D}^{D}$,

Algorithm 6: QTT sine/cosine 1D, approximation

```
Input: $x(k) = X^{(1)}_{k_1} X^{(2)}_{k_2} \dots X^{(d)}_{k_d}$, accuracy $\varepsilon$ or ranks $R_1, \dots, R_{d-1}$
Output: $[y(j)\ z(j)] = Q^{(1)}_{j_1} Q^{(2)}_{j_2} \dots Q^{(d-1)}_{j_{d-1}} [Y^{(d)}_{j_d}\ Z^{(d)}_{j_d}]$, where $y, z$ are defined by (36)
 1: Orthogonalized $X^{(1)}_{k_1} X^{(2)}_{k_2} \dots X^{(d)}_{k_d}$ right-to-left
 2: for $D = d-1, d-2, \dots, 2$ do {Go down along the train}
 3:     Define index $k = \overline{k_1 \dots k_D}$ and $x_D(k) = X^{(1)}_{D,k_1} \dots X^{(D)}_{D,k_D}$
 4:     Compute $y_D(k) = x_D(k) \cos \frac{\pi k}{2^{D+1}}$ and $z_D(k) = x_D(k) \sin \frac{\pi k}{2^{D+1}}$ by

        $$T_D(k) \overset{\text{def}}{=} [x_D(k)\ y_D(k)\ z_D(k)] = \hat{X}^{(1)}_{D,k_1} \hat{X}^{(2)}_{D,k_2} \dots \hat{X}^{(D-1)}_{D,k_{D-1}} \left[ \hat{X}^{(D)}_{D,k_D}\ \hat{Y}^{(D)}_{D,k_D}\ \hat{Z}^{(D)}_{D,k_D} \right],$$

        where $\hat{X}^{(1)}_{D,k_1} := X^{(1)}_{D,k_1} \otimes \begin{bmatrix} 1 & \cos \frac{\pi k_1}{2^{D+1}} & \sin \frac{\pi k_1}{2^{D+1}} \end{bmatrix},$

        $$\hat{X}^{(p)}_{D,k_p} := X^{(p)}_{D,k_p} \otimes \begin{bmatrix} 1 & & \\ & \cos \frac{\pi k_p}{2^{D+2-p}} & \sin \frac{\pi k_p}{2^{D+2-p}} \\ & -\sin \frac{\pi k_2}{2^{D+2-p}} & \cos \frac{\pi k_2}{2^{D+2-p}} \end{bmatrix}, \qquad p = 2, \dots, D-1$$

        $$\hat{X}^{(D)}_{D,k_D} := X^{(D)}_{D,k_D}, \quad \hat{Y}^{D}_{D,k_D} := X^{(D)}_{D,k_D} \otimes \begin{bmatrix} 0 \\ \cos \frac{\pi k_p}{2^2} \\ -\sin \frac{\pi k_p}{2^2} \end{bmatrix}, \quad \hat{Z}^{D}_{D,k_D} := X^{(D)}_{D,k_D} \otimes \begin{bmatrix} 0 \\ \sin \frac{\pi k_p}{2^2} \\ \cos \frac{\pi k_p}{2^2} \end{bmatrix}.$$

 5:     Recompress $T_D(k)$ by Algorithm 3 using accuracy $\varepsilon$ or rank $R_1, \dots, R_{D-1}$ criterion

        $$T_D(k) \approx \tilde{T}_D(k) =: X^{(1)}_{D-1,k_1} X^{(2)}_{D-1,k_2} \dots X^{(D-1)}_{D-1,k_{D-1}} \left[ U^{(D)}_{k_D}\ V^{(D)}_{k_D}\ W^{(D)}_{k_D} \right].$$

 6: end for
 7: $Y^{(1)}_{j_1} := \sum_{k_1=0}^{1} X^{(1)}_{1,k_1} \cos \pi k_1 j_1/2$, $Z^{(1)}_{j_1} := \sum_{k_1=0}^{1} X^{(1)}_{1,k_1} \sin \pi k_1 j_1/2$
 8: for $D = 1, 2, \dots, d-1$ do {Go up along the train}
 9:     Define index $k = \overline{k_1 \dots k_D}$ and $[y_D(j)\ z_D(j)] = Q^{(1)}_{j_1} Q^{(2)}_{j_2} \dots Q^{(D-1)}_{j_{D-1}} [Y^{(D)}_{j_D}\ Z^{(D)}_{j_D}]$
10:     Implement recurrence step (39) by

        $C^{(D)}_{j_D, j_{D+1}=0} := Y^{(D)}_{j_D} U^{(D+1)}_0 + (-1)^{j_D} Y^{(D)}_{j_D} U^{(D+1)}_1,$
        $C^{(D)}_{j_D, j_{D+1}=1} := Y^{(D)}_{j_D} V^{(D+1)}_0 - Z^{(D)}_{j_D} W^{(D+1)}_0 - (-1)^{j_D} Y^{(D)}_{j_D} W^{(D+1)}_1 - (-1)^{j_D} Z^{(D)}_{j_D} V^{(D+1)}_1,$
        $S^{(D)}_{j_D, j_{D+1}=0} := Z^{(D)}_{j_D} U^{(D+1)}_0 + (-1)^{j_D} Z^{(D)}_{j_D} U^{(D+1)}_1,$
        $S^{(D)}_{j_D, j_{D+1}=1} := Y^{(D)}_{j_D} W^{(D+1)}_0 + Z^{(D)}_{j_D} V^{(D+1)}_0 + (-1)^{j_D} Y^{(D)}_{j_D} V^{(D+1)}_1 - (-1)^{j_D} Z^{(D)}_{j_D} W^{(D+1)}_1.$

11:     Perform low-rank decomposition $\left[ C^{(D)}_{j_D, j_{D+1}}\ S^{(D)}_{j_D, j_{D+1}} \right] \approx: Q^{(D)}_{j_D} \left[ Y^{(D+1)}_{j_{D+1}}\ Z^{(D+1)}_{j_{D+1}} \right]$ with
        $\sum_{j_D} (Q^{(D)}_{j_D})^* (Q^{(D)}_{j_D}) = I$ using accuracy $\varepsilon$ or rank $R_D$ criterion.
12: end for
13: bit-reverse the output
```

that define the coefficients for cosine/sine transforms and $U_{j_{D+1}}^{(D+1)}$, $V_{j_{D+1}}^{(D+1)}$ and $W_{j_{D+1}}^{(D+1)}$, that represent bottom and top parts of vector x scaled by identity, cosine and sine multiplier, respectively.

It is also worth to mention, that to control the perturbation introduced to the whole tensor during each particular approximation it is necessary to keep the orthogonality of *interfaces* (i.e. parts of the whole tensor, that are not approximated on this step). By *orthogonality* of the left-size interface we mean the orthogonality of structured columns (14) that can be ensured by Algorithm 2. The orthogonality for the right interface assumes the similar condition to be ensured for the structured rows. The latter can be done using subsequently the bit-reverse permutation (13), left-to-right orthogonalisation Algorithm 2 and applying bit-reverse again. Note that in the following description of the algorithms we will use left-to-right and right-to-left orthogonalisation freely assuming that the output is written over the input tensors.