Max-Planck-Institut für Mathematik in den Naturwissenschaften Leipzig

Superfast Fourier transform using QTT approximation

(revised version: March 2012)

by

Sergey Dolgov, Boris N. Khoromskij, and Dmitry Savostyanov

Preprint no.: 18 2011

Superfast Fourier transform using QTT approximation

Sergey Dolgov[⋆,†], Boris Khoromskij[†], Dmitry Savostyanov[⋆]

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

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

March 27, 2012

Abstract. We propose Fourier transform algorithms using QTT format for data-sparse approximate representation of one– and multi-dimensional vectors (m–tensors). Although the Fourier matrix itself does not have a low-rank QTT representation, it can be efficiently applied to a vector in the QTT format exploiting the multilevel structure of the Cooley-Tukey algorithm. The m–dimensional Fourier transform of an $n \times \ldots \times n$ vector with $n = 2^d$ has $\mathcal{O}(md^2R^3)$ complexity, where R is the maximum QTT–rank of input, output and all intermediate vectors in the procedure. For the vectors with moderate R and large n and m the proposed algorithm outperforms the $\mathcal{O}(n^m \log n)$ fast Fourier transform (FFT) algorithm and has asymptotically the same log-squared complexity as the superfast quantum Fourier transform (QFT) algorithm. By numerical experiments we demonstrate the examples of problems for which the use of QTT format relaxes the grid size constrains and allows the high-resolution computations of Fourier images and convolutions in higher dimensions without the 'curse of dimensionality'. We compare the proposed method with Sparse Fourier transform algorithms and show that our approach is competitive for signals with small number of randomly distributed frequencies and signals with limited bandwidth.
Keywords: High-dimensional problems, tensor train format, QTT, Fourier transform, convolution, sparse Fourier transform, quantum 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. 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, supported by the Promotionsstipendium of Max Planck Society. Part of this work was done during the Visiting Research Fellowship of D. Savostyanov (corresponding author) at the University of Chester, supported by the Leverhulme Trust.

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-dimensional data, see reviews [5, 39, 33] and monograph [18]. Recently proposed *tensor train* (TT) format [48, 43, 46], also known in quantum chemistry as matrix product states (MPS) [60], combines advances of both the canonical and Tucker formats: the number of representing parameters grows only linearly with the dimension and the approximation problem is stable and can be computed by algorithms based on singular value decomposition (SVD). For low-dimensional data, the tensor train format can be applied by substitution of the dimensions with large mode size by a larger number of dimensions with small mode size, resulting in a high-dimensional array. If the small mode size equals two, it becomes 'indivisible', i.e., can not be reduced further. We can say that such mode represents a *quant* or *bit* of information and call this representation of a vector the *quantized tensor train* (QTT) format, following [32]. Similarly, in quantum computations a large data vector is represented by the entangled quantum state of several *qubits* (quantum bits), which are systems with two quantum states.

The QTT format is rank-structured and the storage size is governed by QTT–ranks. The impressive approximation properties of the QTT format were discovered in [32] for a class of functions discretized on uniform grids. In particular, it was proven that the QTT–ranks of $\exp(\alpha x)$, $\sin(\alpha x)$, $\cos(\alpha x)$, x^p are uniformly bounded with respect to (w.r.t.) the grid size. For the functions $e^{-\alpha x^2}$, x^α, $\frac{\sin x}{x}$, $\frac{1}{x}$, etc., similar properties were found experimentally.

In this paper we propose algorithms for the discrete Fourier transform of one– and high-dimensional data represented or approximated in the QTT format. Our approach bases on a radix-2 recursion formula which reduces the Fourier transform to the one of half size and lies behind the well-known Cooley-Tukey FFT algorithm [11, 4]. Each step of the proposed QTT–FFT algorithm includes the approximation to reduce the storage size of the intermediate vectors adaptively to the prescribed level of accuracy. The complexity of m-dimensional $n \times \ldots \times n$ transform with $n = 2^d$ is bounded by $\mathcal{O}(md^2R^3)$, where R is the maximum QTT–rank of input, all intermediate vectors and output of the QTT–FFT algorithm. For vectors with moderate R, the QTT–FFT algorithm has *square logarithmic* scaling w.r.t. the total number of array entries and outperforms the Cooley-Tukey FFT algorithm, which has $\mathcal{O}(n^m \log n)$ complexity. Given an arbitrary input vector, it is not possible to predict the value of R and state if QTT–FFT algorithm is efficient, until the computation is done. This problem is solved partially in [53], where the class vectors with $R = 1$ is fully described and is also shown by numerical experiments that many vectors can be approximated by ones with moderate R. The QTT–ranks depend on the desired accuracy level, and transforms with lower accuracy are computed faster by the approximate QTT–FFT algorithm, in contrast to the exact FFT algorithm.

The QTT–FFT algorithm can be compared with the quantum Fourier transform (QFT) algorithm, widely utilized in quantum computations, such as eigenvalue estimation, order-finding, integer factorization, etc. A single operation in the quantum algorithm has the *exponential performance*, since it changes all 2^d components of the vector describing the entangled state of a quantum system. Remarkably, the superfast QFT algorithm [8] requires $\mathcal{O}(d^2)$ quantum operations. The QTT–FFT algorithm has asymptotically the same complexity for fixed R, that explains the word *superfast* in the title of this paper.

The QTT–FFT can be also compared with the Sparse Fourier transform. In the proposed method we use data-sparse rank-structured QTT format for data representation, instead of pointwise sparsity of the Fourier image exploited in the Sparse Fourier transform. The QTT–FFT algorithm requires the QTT representation of the input vector, which can be computed from a small number of vector elements (samples) using the TT–ACA algorithm [54]. Our method is deterministic, while the Sparse Fourier transforms algorithms are usually randomized with the probabilistic estimation of the accuracy.

This paper is organized as follows. We start from the overview of TT and QTT formats in Section 2. In Section 3 we present the Fourier transform algorithm for vectors in the QTT format. In Section 4 we explain how to keep the QTT–ranks moderate during this procedure and result in the QTT–FFT algorithm for the approximate computation of Fourier transform in the QTT format. In Section 5 we consider the real-valued transforms (convolution, cosine transform) and explain how to compute them using QTT–FFT algorithm. In Section 6 we develop the multi-dimensional Fourier transform algorithm in the QTT format. In Section 7 we give numerical examples illustrating that the use of QTT format relaxes the grid size constrains and allows high-resolution computations of Fourier images in higher dimensions without the 'curse of dimensionality'. In particular, our approach allows to compute one-dimensional and multi-dimensional Fourier images using $n = 2^{60}$ in 1D and $n = 2^{20}$ in 3D on a standard workstation, see Sec. 7.1 and Sec. 7.2. In Sec. 7.3 we compute the convolution transforms of data with strong cusps or singularities which occur in particular in quantum chemistry and require very fine grid. In Section 8 we compare our method with Sparse Fourier transform algorithms for exactly Fourier-sparse signals and signals with limited bandwidth.

2. Tensor train format

A *tensor* is an array with d indices (or *modes*)

$$\mathbf{X} = [x(k_1, \ldots, k_d)], \qquad k_p = 0, \ldots, n_p - 1, \quad p = 1, \ldots, d.$$

The tensor train (TT) format [43, 46] for the tensor $\mathbf{X}$ reads[1]

$$x(k_1, k_2, \ldots, k_d) = X^{(1)}_{k_1} X^{(2)}_{k_2} \ldots X^{(d)}_{k_d}, \tag{1}$$

where each $X^{(p)}_{k_p}$ is an $r_{p-1} \times r_p$ matrix. Usually the *border conditions* $r_0 = r_d = 1$ are imposed to make every entry $x(k_1, \ldots, k_d)$ a scalar. However, larger r_0 and r_d can be considered and every entry of a tensor $\mathbf{X} = [x(k_1, \ldots, k_d)]$ becomes an $r_0 \times r_d$ matrix. Values $r_0, \ldots, r_{d-1}$ are referred to as *TT–ranks* and characterize the *separation properties* of the tensor $\mathbf{X}$. Three-dimensional arrays $X^{(p)} = [X^{(p)}_{k_p}]$ are referred to as *TT–cores*. The terms 'cores' and 'ranks' were introduced in [43, 46] to reveal the analogy between TT and Tucker formats.

Definition 1 ([43]). The p–th *unfolding* of a $n_1 \times n_2 \times \ldots \times n_d$ tensor $\mathbf{X} = [x(k_1, \ldots, k_d)]$ is the $n_1 \ldots n_p \times n_{p+1} \ldots n_d$ matrix $X^{\{p\}} = [x^{\{p\}}(s, t)]$ with the following elements

$$x^{\{p\}}(s, t) = x(k_1, \ldots, k_p,\ k_{p+1}, \ldots, k_d),$$

$$s = k_1 + k_2 n_1 + \ldots + k_p \prod_{q=1}^{p-1} n_q, \qquad t = k_{p+1} + k_{p+2} n_{p+1} + \ldots + k_d \prod_{q=p+1}^{d} n_q.$$

[1] We will often write the equations in elementwise form, which assumes that all indices run through all possible values.

We will write $X^{\{p\}} = [x(k_1 \dots k_p, k_{p+1} \dots k_d)]$, assuming that comma separates row and column indices. Obviously, if (1) holds, then $r_p \geqslant \operatorname{rank} X^{\{p\}}$. In fact, $r_p = \operatorname{rank} X^{\{p\}}$.

Statement 1 ([43, 46]). Each tensor $\mathbf{X} = [x(k_1, \dots, k_d)]$ can be represented by the TT–format with TT–ranks equal to the ranks of unfoldings,

$$r_p = \operatorname{rank} X^{\{p\}} = \operatorname{rank}[x(k_1 \dots k_p, k_{p+1} \dots k_d)]. \tag{2}$$

For many tensors, unfoldings have large ranks, but can be approximated by the low rank matrices as follows

$$\|X^{\{p\}} - \tilde{X}^{\{p\}}\|_F \leqslant \varepsilon_p, \qquad \operatorname{rank} \tilde{X}^{\{p\}} = r_p.$$

The minimum r_p which satisfies this condition is referred to as the ε_p–rank of $X^{\{p\}}$.

Statement 2 ([43, 46]). If unfoldings $X^{\{p\}}$ of a tensor $\mathbf{X}$ have ε_p–ranks r_p, then $\mathbf{X}$ can be approximated by the tensor $\tilde{\mathbf{X}}$ with TT–ranks r_p and the following accuracy

$$\|\mathbf{X} - \tilde{\mathbf{X}}\|_F \leqslant \varepsilon, \qquad \varepsilon^2 = \varepsilon_1^2 + \ldots + \varepsilon_{d-1}^2.$$

The TT–format for $\tilde{\mathbf{X}}$, i.e., the *approximation* of a given tensor $\mathbf{X}$ in the TT–format with the prescribed accuracy ε, can be computed by the constructive SVD–based algorithm [46]. Here the Frobenius norm of a tensor is defined as follows

$$\|\mathbf{X}\|_F^2 \stackrel{\text{def}}{=} \sum_{k_1 \dots k_d} |x(k_1, \dots, k_d)|^2.$$

In the following we will omit the subscript and write $\|\cdot\| = \|\cdot\|_F$ for all vectors, matrices and tensors.

To apply the TT compression to low dimensional data, the idea of *quantization* was proposed [44, 32]. We will explain the idea for one-dimensional vector $x = [x(k)]_{k=0}^{n-1}$, restricting the discussion to $n = 2^d$. Define the binary notation[2] of index k as follows

$$k = \overline{k_1 \dots k_d} \stackrel{\text{def}}{=} \sum_{p=1}^{d} k_p 2^{p-1}, \qquad k_p = 0, 1. \tag{3}$$

The isomorphic mapping $k \leftrightarrow (k_1, \dots k_d)$ allows to *reshape* a vector $x = [x(k)]$ into the d–tensor $\dot{\mathbf{X}} = [\dot{x}(k_1, \dots, k_d)]$. The TT format (1) for the latter is called the *QTT format* and reads

$$x(k) = x(\overline{k_1 \dots k_d}) = \dot{x}(k_1, \dots, k_d) = X_{k_1}^{(1)} \dots X_{k_d}^{(d)}. \tag{4}$$

This idea appears in [44] in the context of matrix approximation. In [32] the TT format applied after the quantization of indices was called the *QTT format*. The impressive properties of the QTT approximation motivate the development of vector and tensor transforms in the QTT format.

By (2), the QTT–rank r_p of a vector x of size $n = 2^d$ is bounded by the sizes of the $2^p \times 2^{d-p}$ unfolding $X^{\{p\}}$, i.e., $1 \leqslant r_p \leqslant 2^{\min(p, d-p)}$.

[2]The order of bits in the binary notation can be different. The *big-endian* notation assumes that the most significant bit k_d goes first and the least significant bit k_1 goes last, similar to numbers written in the positional system. The *little-endian* notation uses reversed directions of bits, from k_1 to k_d, similar to numerals in the Arabic scripts. The little-endian ordering is consistent with the FORTRAN style of indexing for multidimensional arrays. In this paper we choose the little-endian notation since it allows more elegant and intuitive description of the main QTT–FFT algorithm. The little-endian notation is used also in [22].

Definition 2. We will call the vectors with QTT–ranks one the *rank-one vectors* and the vectors with QTT–ranks $r_p = 2^{\min(p,d-p)}$ the *full-rank* vectors.

A random tensor, like a random matrix, has full TT–ranks with probability one. In general, the QTT–ranks grow *exponentially* in d, making the QTT algorithms completely inefficient. Therefore, QTT methods are naturally limited to the class of problems where all data have moderate QTT–ranks. It is not possible yet to describe the whole class of such problems and vectors explicitly, but it is possible to justify the concept by a number of convincing examples. Some function-related examples are already mentioned, more examples of (piecewise) smooth functions and functions with singularities that have the low-rank QTT representation can be found in [16, 45, 37]. Exponential vector (i.e., function discretized on the uniform grid) deserves a special interest in this paper. It has the rank-one QTT representation [32]

$$\exp(\alpha k) = \exp(\alpha\, \overline{k_1 k_2 \dots k_d}) = \exp(\alpha k_1) \exp(2\alpha k_2) \dots \exp(2^{d-1} \alpha k_d), \tag{5}$$

which plays a key role for the efficient Fourier transform algorithm in the QTT format.

The QTT format can be applied not only to vectors, but also to matrices (see *TTM format* in [44]). Consider a $2^d \times 2^d$ matrix $A = [a(j,k)]_{j,k=0}^{2^d-1}$, use the binary notation (3) for indices $j = \overline{j_1 \dots j_d}$ and $k = \overline{k_1 \dots k_d}$, then permute binary indices and *reshape* A to the tensor $\dot{\mathbf{A}} = [\dot{a}(j_1 k_1, j_2 k_2, \dots, j_d k_d)]$. Finally, apply the TT format to $\dot{\mathbf{A}}$ as follows

$$a(j,k) = a(\overline{j_1 \dots j_d}, \overline{k_1 \dots k_d}) = \dot{a}(j_1 k_1, j_2 k_2, \dots, j_d k_d) = A^{(1)}_{j_1 k_1} A^{(2)}_{j_2 k_2} \dots A^{(d)}_{j_d k_d}, \tag{6}$$

where each $A^{(p)}_{j_p k_p}$ is an $r_{p-1} \times r_p$ matrix. Merging indices j_p and k_p in the pair-index i_p, we have the d–tensor $\dot{\mathbf{A}} = [\dot{a}(i_1, \dots, i_d)]$ and can use its unfoldings to find the TT–ranks of (6). In Sec 3.1 we show that the Fourier matrix is 'not compressible' using the QTT format, i.e., its QTT–ranks grow exponentially with d. Therefore, the efficient Fourier transform appears to be a nontrivial problem. In contrast, recent results on explicit representation for the inverse Laplacian and related matrices [25] and efficient convolution in the QTT format [26] are based on the low-rank QTT decompositions of certain matrices and tensors. There are more examples of high-dimensional problems which were efficiently solved using the QTT approximation [38, 37, 6] as well as H–Tucker format [17].

The TT/QTT algorithms are based on the basic linear algebra procedures like summation, multiplication and rank truncation (tensor rounding) maintaining the compressed format, i.e., the full data array is never computed. A comprehensive list of basic operations is given in [46]. We will need the Hadamard (elementwise) product,

$$z = x \odot y, \qquad \text{i.e.,} \qquad z(k) = x(k) y(k), \quad k = 0, \dots, 2^d - 1.$$

If x and y are given in the QTT format (4), the QTT format for z is the following

$$z(k) = z(\overline{k_1 \dots k_d}) = Z^{(1)}_{k_1} \dots Z^{(d)}_{k_d}, \qquad Z^{(p)}_{k_p} = X^{(p)}_{k_p} \otimes Y^{(p)}_{k_p}, \quad p = 1, \dots, d, \tag{7}$$

where $X \otimes Y$ denotes the Kronecker (tensor) product of matrices X and Y. The QTT–ranks of the Hadamard product z are the products of corresponding QTT–ranks of x and y.

3. Discrete Fourier transform in one dimension

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

$$y(j) = \frac{1}{2^{d/2}} \sum_{k=0}^{2^d-1} x(k)\omega_d^{jk}, \qquad \omega_d = \exp\left(-\frac{2\pi i}{2^d}\right), \quad i^2 = -1, \tag{8}$$

where $F_d = \frac{1}{2^{d/2}} \left[\omega_d^{jk}\right]_{j,k=0}^{2^d-1}$ is the unitary Fourier matrix. The inverse Fourier transform is written in the same way, with $\omega_d = \exp\left(\frac{2\pi i}{2^d}\right)$.

3.1. QTT decomposition of the Fourier matrix has full ranks

Unlike many matrices used in scientific computing (see, e.g., [44]), the Fourier matrix can not be compressed in the QTT format (6), i.e., its QTT–ranks grow exponentially with d. The proof is based on the properties of unfoldings of the Fourier matrix, defined as follows

$$F_d^{\{p\}} = [f_d^{\{p\}}(j'k', j''k'')], \qquad f_d^{\{p\}}(j'k', j''k'') = \frac{1}{2^{d/2}}\omega_d^{jk}, \qquad \text{where}$$
$$j = j' + 2^p j'', \quad k = k' + 2^p k'', \qquad j', k' = 0, \ldots, 2^p - 1, \quad j'', k'' = 0, \ldots, 2^{d-p} - 1.$$

Lemma 1. For $p \leqslant d/2$ the unfoldings $F_d^{\{p\}}$ have orthogonal rows, for $p \geqslant d/2$ they have orthogonal columns.

Proof. If $p \leqslant d/2$, elements of the unfolding $F_d^{\{p\}}$ are the following

$$f_d^{\{p\}}(j'k', j''k'') = \frac{1}{2^{d/2}}\omega_d^{jk} = \frac{1}{2^{d/2}}\omega_d^{j'k'}\omega_{d-p}^{j'k''}\omega_{d-p}^{k'j''}\omega_{d-2p}^{j''k''},$$

which in the matrix notation reads

$$F_p^{\{d\}} = \frac{2^{d-p}}{2^{d/2}}\Omega' \ (\Phi \otimes \Phi) \ \Omega'',$$

where Ω' and Ω'' are unitary diagonal matrices and Φ is the $2^p \times 2^{d-p}$ top submatrix of F_{d-p}. Since F_{d-p} is orthogonal, the matrix Φ also has orthogonal rows and the Gram matrix of $F_d^{\{p\}}$ writes as follows

$$\begin{aligned} F_d^{\{p\}}\left(F_d^{\{p\}}\right)^* &= 2^{d-2p}\Omega' \ (\Phi \otimes \Phi) \ \Omega'' (\Omega'')^* \ (\Phi \otimes \Phi)^* \ (\Omega')^* \\ &= 2^{d-2p}\Omega' \ (\Phi\Phi^*)^{\otimes 2} (\Omega')^* \\ &= 2^{d-2p}\Omega' (\Omega')^* \\ &= 2^{d-2p} I. \end{aligned}$$

The statement for the case $p \geqslant d/2$ is proved in the same way. □

The following theorem proves that the low-rank approximation of the Fourier matrix in the QTT format with the reasonable accuracy is not possible.

Theorem 1. If the Fourier matrix F_d is approximated by the matrix A with relative accuracy $\|F_d - A\| \leqslant \varepsilon \|F_d\|$, and A is given in the QTT format (6) with QTT–ranks $r_1, \ldots, r_{d-1}$, then

$$r_p \geqslant (1-\varepsilon) 4^{\min(p, d-p)}, \qquad p = 1, \ldots, d-1.$$

Proof. From the assumption of the theorem it follows that for every p the unfolding matrix $F_d^{\{p\}}$ is approximated by the rank–r_p matrix $A^{\{p\}}$ as follows,

$$\|F_d^{\{p\}} - A^{\{p\}}\| \leqslant \varepsilon \|F_d^{\{p\}}\|, \qquad A^{\{p\}} = [a^{\{p\}}(j'k', j''k'')],$$
$$a^{\{p\}}(j'k', j''k'') = A^{(1)}_{j_1 k_1} \ldots A^{(p)}_{j_p k_p} A^{(p+1)}_{j_{p+1} k_{p+1}} \ldots A^{(d)}_{j_d k_d}.$$

where $j' = \overline{j_1 \ldots j_p}$, $k' = \overline{k_1 \ldots k_p}$, $j'' = \overline{j_{p+1} \ldots j_d}$, $k'' = \overline{k_{p+1} \ldots k_d}$. Denote by $\tilde{F}_d^{\{p\}}$ the best rank–r_p approximation of $F_d^{\{p\}}$, then

$$\|F_d^{\{p\}} - \tilde{F}_d^{\{p\}}\| \leqslant \|F_d^{\{p\}} - A^{\{p\}}\|.$$

By Lemma 1, the $4^p \times 4^{d-p}$ unfolding $F_d^{\{p\}}$ has orthogonal rows/columns and the accuracy of the best rank–r_p approximation is exactly the following

$$\|F_d^{\{p\}} - \tilde{F}_d^{\{p\}}\| = \left(1 - \frac{r_p}{4^{\min(p,d-p)}}\right) \|F_d^{\{p\}}\|.$$

We have $1 - \frac{r_p}{4^{\min(p,d-p)}} \leqslant \varepsilon$, which completes the proof. □

Corollary 1. The QTT–ranks of the exact QTT–decomposition of the Fourier matrix are $r_p = 4^{\min(p,d-p)}$ and the storage size grows exponentially with d.

Remark 1. In contrast to the Fourier transform, the Hadamard transform matrix has QTT–ranks one, which follows directly from the definition,

$$H_d = H_1^{\otimes d}, \qquad H_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

This transform, also known as Walsh transform, arises in various applications, including quantum computing, etc. The Hadamard transform can be easily applied to a vector given in the QTT format and does not change the QTT–ranks. The Fourier transform can arbitrarily increase the QTT–ranks of a vector.

3.2. Radix-2 recursion formula in the QTT format

Since the Fourier matrix is not compressible in the QTT format, we can not compute $y = F_d x$ using matrix-vector multiplication algorithm in the TT format [46]. Nevertheless, we can *apply* the Fourier transform to the QTT vector efficiently. We define

$$\begin{array}{rclcll} j & = & \overline{j_1 j_2 \ldots j_d} & = & j_1 + 2j', & j' = \overline{j_2 \ldots j_d}, \\ k & = & \overline{k_1 \ldots k_{d-1} k_d} & = & k' + 2^{d-1} k_d, & k' = \overline{k_1 \ldots k_{d-1}} \end{array} \tag{9}$$

and split odd and even values of the result

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

We come to the well-known radix-2 recursion formula, the simplest and most common case of the Cooley-Tukey fast Fourier transform (FFT) algorithm [11, 4]. It reduces the full-size Fourier transform to the half-sized transforms as follows

$$
P_d F_d = \begin{bmatrix} F_{d-1} & \\ & F_{d-1} \end{bmatrix} \begin{bmatrix} I & \\ & \Omega_{d-1} \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} I & I \\ I & -I \end{bmatrix}. \tag{10}
$$

Here P_d is the *bit-shift* permutation which agglomerates even and odd elements of a vector, and $\Omega_{d-1} = \mathrm{diag}\{\omega_d^{k'}\}_{k'=0}^{2^{d-1}-1}$ is the matrix of *twiddle factors*. We will need the following general definitions later,

$$
(P_p y)(\overline{\underbrace{j_2\, j_3 \ldots j_p\, j_1}_{\text{bit-shift}}\; j_{p+1} \ldots j_d}) = y(\overline{j_1\, j_2 \ldots j_p\, j_{p+1} \ldots j_d}), \tag{11}
$$

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

Our goal is to compute $y = F_d x$ for the vector x given in the QTT format (4). First, we note that "top" and "bottom" half-vectors of x are the following

$$
\begin{aligned}
x_{\mathrm{top}}(k') &\stackrel{\mathrm{def}}{=} x(k') = x(\overline{k_1 \ldots k_{d-1} 0}) = X_{k_1}^{(1)} X_{k_2}^{(2)} \ldots X_{k_{d-1}}^{(d-1)} X_{k_d=0}^{(d)}, \\
x_{\mathrm{bot}}(k') &\stackrel{\mathrm{def}}{=} x(k'+2^{d-1}) = x(\overline{k_1 \ldots k_{d-1} 1}) = X_{k_1}^{(1)} X_{k_2}^{(2)} \ldots X_{k_{d-1}}^{(d-1)} X_{k_d=1}^{(d)},
\end{aligned} \tag{12}
$$

and their summation/subtraction affects only the last core.

$$
\hat{x} \stackrel{\mathrm{def}}{=} \frac{1}{\sqrt{2}} \begin{bmatrix} I & I \\ I & -I \end{bmatrix} x, \qquad \hat{x}(k) = X_{k_1}^{(1)} \ldots X_{k_{d-1}}^{(d-1)} \hat{X}_{k_d}^{(d)}, \qquad \begin{aligned} \hat{X}_0^{(d)} &= \tfrac{1}{\sqrt{2}} \left(X_0^{(d)} + X_1^{(d)} \right), \\ \hat{X}_1^{(d)} &= \tfrac{1}{\sqrt{2}} \left(X_0^{(d)} - X_1^{(d)} \right). \end{aligned}
$$

The multiplication by the diagonal matrix writes as the Hadamard multiplication

$$
z = \begin{bmatrix} I & \\ & \Omega_{d-1} \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} I & I \\ I & -I \end{bmatrix} x = \begin{bmatrix} I & \\ & \Omega_{d-1} \end{bmatrix} \hat{x} = w_d \odot \hat{x}, \tag{13}
$$

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

where vector $w_d = [w_d(k)]$ has the following rank-two QTT decomposition

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

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

$$\begin{aligned} z(k) = w_d(k)\hat{x}(k) = Z^{(1)}_{k_1} Z^{(2)}_{k_2} \dots Z^{(d)}_{k_d}, \qquad \text{with} \qquad Z^{(1)}_{k_1} = \begin{bmatrix} X^{(1)}_{k_1} & \omega_d^{k_1} X^{(1)}_{k_1} \end{bmatrix}, \\ Z^{(p)}_{k_p} = \begin{bmatrix} X^{(p)}_{k_p} & \\ & \omega_{d-p+1}^{k_p} X^{(p)}_{k_p} \end{bmatrix}, \quad p = 2, \dots, d-1, \qquad Z^{(d)}_{k_d} = \begin{bmatrix} (1-k_d)\hat{X}^{(d)}_{k_d} \\ k_d \hat{X}^{(d)}_{k_d} \end{bmatrix}. \end{aligned} \tag{14}$$

Note that the multiplication by the twiddle factors *doubles* the QTT–ranks.

The last step to implement (10) is to apply the half-size Fourier transform to the "top" and "bottom" parts of the vector z. We consider the following $n/2 \times 2r_{d-1}$ matrix

$$Z' = [z'(\overline{k_1 \dots k_{d-1}}, \alpha)], \quad \text{where} \quad z'(\overline{k_1 \dots k_{d-1}}, :) = Z^{(1)}_{k_1} \dots Z^{(d-1)}_{k_{d-1}},$$

and compute $F_{d-1}Z'$ in the QTT format, applying the radix-2 recursion subsequently.

Each radix-2 step applies the bit-shift permutation to the result. It is easy to notice 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 the QTT format *without any computations* by reversing the order of cores in the tensor train,

$$x(k) = x(\overline{k_1 \dots k_d}) = X^{(1)}_{k_1} \dots X^{(d)}_{k_d}, \qquad (R_d x)(\overline{k_d \dots k_1}) = \left(X^{(d)}_{k_d}\right)^T \dots \left(X^{(1)}_{k_1}\right)^T. \tag{15}$$

We add the bit-reverse step and summarize all the above in the QTT–FFT Algorithm 1.

4. Approximate Fourier transform

We do not benefit from the use of the QTT format if QTT–ranks are too large. However, in Algorithm 1 the QTT–ranks grow by a factor of two each time we multiply by twiddle factors on Line 4. After d steps they grow by $2^d = n$, which makes the QTT representation completely ineffective. To perform computations efficiently, it is necessary to truncate the QTT–ranks, i.e., to approximate the result by the QTT format with smaller values of ranks.

4.1. TT–rounding and TT–orthogonalization

We recall the most important properties of the *TT–rounding* (also *rank truncation* or *recompression*) algorithm proposed in [46]. We will use it to approximate a vector x given in the QTT–format (4) by vector $\tilde{x}$ in the following QTT form,

$$\tilde{x}(k) = \tilde{x}(\overline{k_1 \dots k_d}) = \tilde{X}^{(1)}_{k_1} \dots \tilde{X}^{(d)}_{k_d}, \quad \|\tilde{x}\| = \|x\|, \quad \|x - \tilde{x}\| \leqslant \varepsilon \|x\|, \tag{16}$$

where each $\tilde{X}_{k_p}$ is an $\tilde{r}_{p-1} \times \tilde{r}_p$ matrix, $\tilde{r}_0 = r_0$, $\tilde{r}_d = r_d$, $\tilde{r}_p \leqslant r_p$, $p = 1, \dots, d-1$. The approximation can be done up to some desired accuracy level or by bounding the values of QTT–ranks $\tilde{r}_p$ from above. If we want to prescribe the *relative accuracy* ε in (16), then the

Algorithm 1: QTT–FFT, exact computation

Input: Vector $x = [x(k)]_{k=0}^{2^d-1}$ in the QTT format $x(k) = x(\overline{k_1 \dots k_d}) = X_{k_1}^{(1)} \dots X_{k_d}^{(d)}$.

Output: Vector $y = F_d x = [y(j)]_{j=0}^{2^d-1}$ in the QTT format $y(j) = y(\overline{j_1 \dots j_d}) = Y_{j_1}^{(1)} \dots Y_{j_d}^{(d)}$.

1: Define $x_d = x$ and $X_{d,k_p}^{(p)} = X_{k_p}^{(p)}$.

2: **for** $D = d, d-1, \dots, 1$ **do**

3: $\hat{X}_{D,0}^{(D)} := \frac{1}{\sqrt{2}} \left(X_{D,0}^{(D)} + X_{D,1}^{(D)} \right), \quad \hat{X}_{D,1}^{(D)} := \frac{1}{\sqrt{2}} \left(X_{D,0}^{(D)} - X_{D,1}^{(D)} \right)$

{Now $\hat{x}_D$ has QTT form $\hat{x}_D(\overline{k_1 \dots k_D}) = X_{D,k_1}^{(1)} X_{D,k_2}^{(2)} \dots X_{D,k_{D-1}}^{(D-1)} \hat{X}_{D,k_D}^{(D)}$}

4: $Z_{k_1}^{(1)} := \begin{bmatrix} X_{k_1}^{(1)} & \omega_D^{k_1} X_{k_1}^{(1)} \end{bmatrix}$,

$$Z_{k_p}^{(p)} := \begin{bmatrix} X_{k_p}^{(p)} & \\ & \omega_D^{2^{p-1}k_p} X_{k_p}^{(p)} \end{bmatrix}, \quad p = 2, \dots, D-1, \qquad Z_{k_D}^{(D)} := \begin{bmatrix} (1-k_D)\hat{X}_{k_D}^{(D)} \\ k_D \hat{X}_{k_D}^{(D)} \end{bmatrix}.$$

{Now z_D has QTT form $z_D(\overline{k_1 \dots k_D}) = Z_{D,k_1}^{(1)} Z_{D,k_2}^{(2)} \dots Z_{D,k_D}^{(D)}$}

5: $X_{D-1,k_p}^{(p)} := Z_{D,k_p}^{(p)}$.

{Now x_{D-1} has QTT form $x_{D-1}(\overline{k_1 \dots k_{D-1}}) := X_{D-1,k_1}^{(1)} X_{D-1,k_2}^{(2)} \dots X_{D-1,k_{D-1}}^{(D-1)}$}

6: **end for**

7: Return $Y_{j_p}^{(p)} := (Z_{d-p+1,j_p}^{(d-p+1)})^T$, $p = 1, \dots, d$. {bit-reverse the output}

values of $\tilde{r}_p$ appear during the work of the TT–rounding procedure and in general can not be predicted. If $\tilde{r}_p$ are prescribed, then the approximation is quasi-optimal [49, 46], i.e.,

$$\|x - \tilde{x}\| \leqslant \sqrt{d-1} \|x - \tilde{x}^{\text{best}}\|,$$

where $\tilde{x}^{\text{best}}$ is the best approximation of x with TT–ranks $\tilde{r}_p$. However, the error $\|x - \tilde{x}\|$ generally can not be known in advance. The TT–rounding algorithm is based on QR and SVD algorithms for matrices, which are well established and included in many linear algebra packages (cf. LAPACK). We illustrate the work of TT–rounding on Fig. 1.

The key part of the TT–rounding algorithm is the *TT–orthogonalization*.

Definition 3. The TT-core $X^{(p)}$ is called *left–* or *right-orthogonal* if, respectively,

$$\sum_{k_p} \left(X_{k_p}^{(p)} \right)^* X_{k_p}^{(p)} = I \qquad \text{or} \qquad \sum_{k_p} X_{k_p}^{(p)} \left(X_{k_p}^{(p)} \right)^* = I.$$

If the TT-core $X^{(p)}$ is written as $r_{p-1} \times n_p \times r_p$ tensor, the left-orthogonality implies that the $r_{p-1} n_p \times r_p$ unfolding has orthogonal columns and the right-orthogonality means that the $r_{p-1} \times n_p r_p$ unfolding has orthogonal rows. A product of two left-orthogonal TT–cores is also a left-orthogonal array, as explained by the following statement.

Statement 3. [46] Consider $p \times q$ matrices A_i, $i = 0, \dots, m-1$, and $q \times r$ matrices B_j, $j = 0, \dots, n-1$. If 3-tensors $A = [A_i]$ and $B = [B_i]$ are left-orthogonal, i.e., $\sum_i A_i^* A_i = I$ and $\sum_j B_j^* B_j = I$, then the $p \times mn \times r$ tensor $C = [C_{ij}]$, $C_{ij} = A_i B_j$, is also left-orthogonal,

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

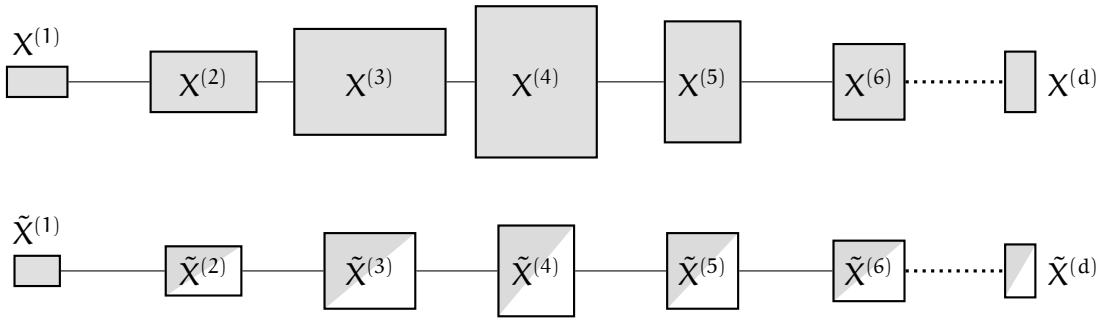

Figure 1. The TT–rounding algorithm: (top) input; (bottom) output. The rectangles represent the QTT cores, the sides of rectangles are proportional to the values of QTT–ranks.

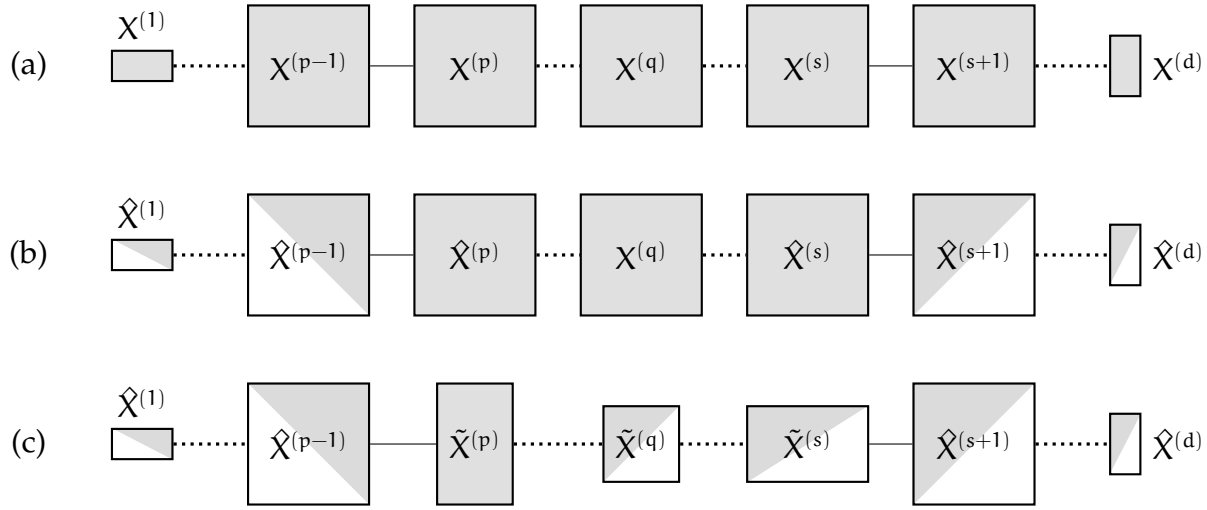

Figure 2. TT–orthogonalization. (a) Input QTT format for x; (b) TT–orthogonalization is applied to interfaces $X^{(1)} \dots X^{(p)}$ (left) and $X^{(s)} \dots X^{(d)}$ (right); (c) TT–rounding is applied to the subtrain $\hat{X}^{(p)}_{k_p} X^{(p+1)}_{k_{p+1}} \dots X^{(s-1)}_{k_{s-1}} \hat{X}^{(s)}_{k_s}$, giving $\tilde{X}^{(p)}_{k_p} \tilde{X}^{(p+1)}_{k_{p+1}} \dots \tilde{X}^{(s-1)}_{k_{s-1}} \tilde{X}^{(s)}_{k_s}$.

The same statement holds for right-orthogonal cores. It can be also generalized to a larger number of subsequent TT–cores.

Definition 4. A sequence of TT–cores $X^{(p)}, X^{(p+1)}, \dots, X^{(q-1)}, X^{(q)}$ will be referred to as a *subtrain*.

Statement 4. [46] If all TT–cores of the subtrain $X^{(1)}, \dots, X^{(p)}$ are left-orthogonal and $r_0 = 1$, then the following $(n_1 \dots n_p) \times r_p$ matrix X' has orthogonal columns,

$$X' = [x'(\overline{k_1 \dots k_p}, \alpha)], \quad \text{where} \quad x'(\overline{k_1 \dots k_p}, :) = X^{(1)}_{k_1} \dots X^{(p)}_{k_p}.$$

The left– or right-orthogonality of the particular subtrain can be achieved straightforwardly, using matrix orthogonalization procedure such as QR. The algorithm is called TT–orthogonalization and given in [46]. The 'structured orthogonality' of subtrains allows to perturb other cores and control the accuracy of the whole tensor. We illustrate this on Fig. 2. Given a vector x with QTT representation (4), we apply left TT–orthogonalization

Algorithm 2: QTT–FFT, approximation

Input: Vector x of size 2^d in the QTT format (4), accuracy ε or maximum ranks $R_1, \ldots, R_{d-1}$.
Output: Vector $y \approx F_d x$ in the QTT format $y(j) = y(\overline{j_1 \ldots j_d}) = Y^{(1)}_{j_1} Y^{(2)}_{j_2} \ldots Y^{(d)}_{j_d}$.
{Lines 1-4 as in Alg. 1}.
5: {Modify Line 5 of Alg. 1 as follows}
if accuracy criterion ε is given **then**
Use TT–rounding [46] to approximate z_D with relative accuracy ε as follows

$$\|z_D - \tilde{z}_D\| \leqslant \varepsilon \|z_D\|, \quad \|\tilde{z}_D\| = \|z_D\|, \quad \tilde{z}_D(\overline{k_1 \ldots k_D}) = \tilde{Z}^{(1)}_{D,k_1} \ldots \tilde{Z}^{(D)}_{D,k_D}.$$

else {maximum rank criterion $R_1, \ldots, R_{d-1}$ is given}
Use TT–rounding [46] to find quasi-optimal approximation

$$\tilde{z}_D \approx z_D, \quad \|\tilde{z}_D\| = \|z_D\|, \quad \tilde{z}_D(\overline{k_1 \ldots k_D}) = \tilde{Z}^{(1)}_{D,k_1} \ldots \tilde{Z}^{(D)}_{D,k_D}.$$

with QTT–ranks $\tilde{r}_p \leqslant R_p$.
end if
$X^{(p)}_{D-1,k_p} := \tilde{Z}^{(p)}_{D,k_p}$.
{Continue as in Alg. 1}
Return $Y^{(p)}_{j_p} := (\tilde{Z}^{(d-p+1)}_{d-p+1,j_p})^T$, $p = 1, \ldots, d$. {bit-reverse the output}

to the subtrain $X^{(1)} \ldots X^{(p)}$ and right TT–orthogonalization to $X^{(s)} \ldots X^{(d)}$. As a result, these cores are modified and we get another QTT representation for the same vector x, where cores $\hat{X}^{(1)}, \ldots, \hat{X}^{(p-1)}$ are left-orthogonal and $X^{(s+1)}, \ldots, X^{(d)}$ are right-orthogonal. We show left– and right–orthogonal cores as $\square$ and $\square$, respectively. Similar notation is used in [51].

After TT–orthogonalization is done, we apply TT–truncation to the subtrain $X^{(p)} \ldots X^{(s)}$ and reduce TT–ranks $r_p, \ldots, r_{s-1}$ to $\tilde{r}_p, \ldots, \tilde{r}_{s-1}$, introducing an approximation error or *perturbation* to the subtrain. The orthogonality of the left and right *interfaces* $X^{(1)} \ldots X^{(p-1)}$ and $X^{(s+1)} \ldots X^{(d)}$ guarantees that the same perturbation is introduced to the whole vector x. The orthogonality of interfaces is also used in DMRG algorithm [60].

Both the TT–rounding and TT–orthogonalization procedures for the QTT format have $\mathcal{O}(dR^3)$ complexity, where d is the length of the tensor train and $R = \max r_p$ is the maximum TT–rank. Note that the TT–rounding procedure can generate the approximation with orthogonal cores. In [46] the TT–rounding generates left-orthogonal tensor train, but we will assume that TT–rounding generates right-orthogonal QTT approximation, as shown on Fig. 1.

4.2. Accuracy of the QTT–FFT algorithm with approximation

We include the TT–rounding step in Algorithm 1 and result in the version of QTT–FFT with approximation, which we will call QTT–FFT in the following. It is given by Algorithm 2 and visualized on Fig. 3. Each step of Algorithm 2 includes an approximation which introduces an error to the active subtrain. In the following theorem we estimate the accuracy of the result returned by QTT–FFT.

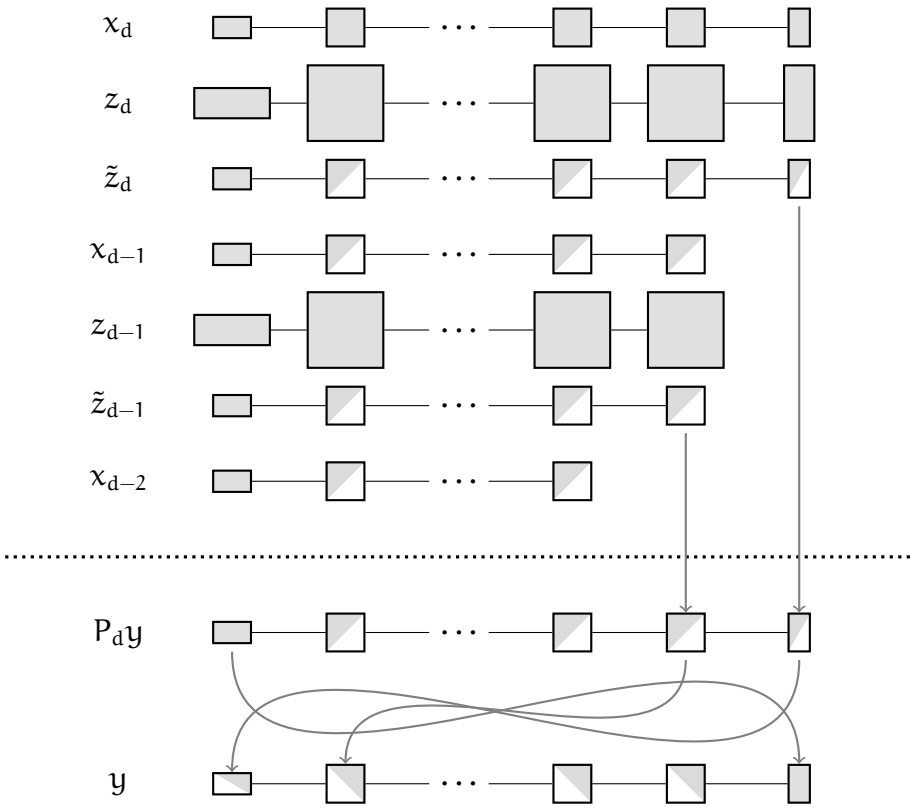

Figure 3. Visualization of the QTT–FFT Algorithm 2.

Theorem 2. If accuracy-based criterion ε is used on each approximation step in Algorithm 2, the accuracy of the result is the following

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

Proof. In the first step $D = d$ of the algorithm it holds $y = F_d x_d$ and

$$P_d y = (I_1 \otimes F_{d-1})\, z_d,$$

where I_p denotes the $2^p \times 2^p$ identity matrix and z_d is defined by (14). The approximation step gives $\|z_d - \tilde{z}_d\| \leqslant \varepsilon\|z_d\|$ and since P_d and $I_1 \otimes F_{d-1}$ are unitary matrices we have

$$y = \tilde{y}_d + \mu_d, \qquad \tilde{y}_d \stackrel{\text{def}}{=} P_d^{\mathrm{T}}\,(I_1 \otimes F_{d-1})\,\tilde{z}_d, \qquad \|\mu_d\| \leqslant \varepsilon\|y\|, \tag{17}$$

where $\tilde{y}_d$ is the approximation of the result and μ_d is the error at step $D = d$. At the next step the Fourier transform is applied to the x_{d-1} are $n/2 \times r_{d-1}(\tilde{z}_d)$ matrix x_{d-1}, where $r_{d-1}(\tilde{z}_d)$ is the corresponding QTT–rank of $\tilde{z}_d$. The matrix x_{d-1} has the following QTT decomposition

$$x_{d-1}(\overline{k_1 \dots k_{d-1}},\, :) := X^{(1)}_{d-1,k_1} X^{(2)}_{d-1,k_2} \dots X^{(d-1)}_{d-1,k_{d-1}}, \qquad X^{(p)}_{d-1,k_p} = \tilde{Z}^{(p)}_{d,k_p}, \quad p = 1, \dots, d-1.$$

If $y_{d-1} = F_{d-1} x_{d-1}$ is computed, the approximation $\tilde{y}_d$ writes in elementwise notation as follows

$$\tilde{y}_d(\overline{j_1 \dots j_d}) = y_{d-1}(\overline{j_2 \dots j_d})\tilde{Z}^{(d)}_{d,j_1}. \tag{18}$$

Note that the TT-core $\tilde{Z}_d^{(d)}$ is right-orthogonal after the TT–truncation, see Fig. 3. This means that any perturbation introduced to y_{d-1} results in the error of the same norm introduced to $\tilde{y}_d$. We will now estimate this error.

In the step $D = d-1$ of the algorithm we compute the Fourier transform of x_{d-1} using the same radix-2 step. We define z_{d-1} by Lines 3 and 4 of Algorithm 1 and have

$$P_{d-1} y_{d-1} = (I_1 \otimes F_{d-2})\, z_{d-1},$$

which is similar to what we had before. Approximation gives $\|\tilde{z}_{d-1} - z_{d-1}\| \leqslant \varepsilon \|z_{d-1}\|$. This leads to

$$y_{d-1} = \tilde{y}_{d-1} + \mu_{d-1}, \qquad \tilde{y}_{d-1} \stackrel{\text{def}}{=} P_{d-1}^{\mathsf{T}}\, (I_1 \otimes F_{d-2})\, \tilde{z}_{d-1}, \qquad \|\mu_{d-1}\| \leqslant \varepsilon \|y\|.$$

Substituting this equation to Eqs. (18) and (17), we have, in elementwise form,

$$y(\overline{j_1 \dots j_d}) = \tilde{y}_{d-1}(\overline{j_2 \dots j_d}) Z_{d,j_1}^{(d)} + \mu_{d-1}(\overline{j_2 \dots j_d}) Z_{d,j_1}^{(d)} + \mu_d(\overline{j_1 \dots j_d}).$$

Since the TT–core $Z_d^{(d)}$ is right-orthogonal, it does not change the Frobenius norm, and both error terms have norm bounded by $\varepsilon\|y\|$. It is easy to notice that every approximation step D introduces the perturbation $\|\mu_D\| \leqslant \varepsilon\|y\|$, and since all TT–cores in the right subtrain $Z_{D+1}^{(D+1)} \dots Z_d^{(d)}$ are right-orthogonal, the perturbation of the result has the same norm. After d approximation steps of the algorithm, the total error is not larger than $d\varepsilon\|y\|$. This completes the proof. □

Remark 2. The proof of the Theorem 2 actually shows that the norm of the error of Algorithm 2 is not larger than the sum of the norms of errors introduced on each approximation step. If for a certain input vector all TT–truncation steps are exact, the result returned by Algorithm 2 is precise. The examples of such vectors are given in [53].

Theorem 2 holds for any input vector x. However, the QTT–ranks of the intermediate vectors x_D, $D = d, \dots, 1$, depend both on the accuracy level ε and the properties of the vector x. It is not easy to describe explicitly the class of vectors x for which all x_D in Algorithm 2 will have QTT–ranks bounded by the desired value R for accuracy level ε. This problem is solved for $R = 1, \varepsilon = 0$ (QTT–rank-one vectors with QTT–rank-one Fourier images) in [53]. The numerical experiments provided in [53] show that the class of such vectors for larger R and non-zero ε can be 'not so small', even if ε is close to the machine threshold accuracy. For instance, the Fourier transform of a random rank-one vector typically (in the large set of numerical tests) is approximated by the vector with moderate QTT–ranks.

4.3. Complexity analysis

Now we estimate the asymptotic complexity of Algorithm 2 assuming that all QTT–ranks of vectors x_D on all steps of the algorithm are bounded by R, as well as QTT–ranks of input and output.

Theorem 3. If on levels $D = d, \dots, 1$ of Algorithm 2 the QTT–ranks of intermediate vectors x_D are not larger than R_D, the complexity is estimated as follows

$$\texttt{work} \leqslant \sum_{D=1}^{d} \mathcal{O}(D R_D^3) \leqslant \mathcal{O}(d^2 R^3), \qquad \text{where} \quad R = \max R_D. \tag{19}$$

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

1. alter the last block $\hat{X}^{(D)}_{0,1} = \frac{1}{\sqrt{2}} \left(X^{(D)}_0 \pm X^{(D)}_1 \right)$ in $\mathcal{O}(R^2_D)$ operations;
2. multiply the cores by twiddle factors, see Eq. (14) in $\mathcal{O}(R^2_D)$ operations;
3. compress the subtrain $z_D(k) = Z^{(1)}_{k_1} \cdots Z^{(D)}_{k_D}$ using the TT–truncation algorithm [46] of $\mathcal{O}(DR^3_D)$ complexity.

By summation we come to (19). □

The complexity of the QTT–FFT algorithm is bounded by $\mathcal{O}(d^2R^3)$, where R depends on the input vector x and the accuracy ε of the approximation. It can be compared with the complexity of the superfast quantum Fourier transform [8], which is $\mathcal{O}(d^2)$ *quantum* operations. In general (for arbitrary input vector x) the value of R can grow exponentially with d. If we use the 'brute-force' TT–truncation with the maximum rank criterion, the value of R does not grow and complexity is $\mathcal{O}(d^2)$ for all input vectors, but the accuracy of the result can be arbitrarily bad. If the accuracy-based criterion is used, Theorem 2 guarantees the accuracy of the result, but the value of R and the complexity can be arbitrarily large. Therefore, for a class of all possible input vectors the accuracy and the complexity of the QTT–FFT are related (informally speaking) by the uncertainty principle: *both can not be small*.

It is interesting (both theoretically and practically) to establish a special subclass of vectors x for which the QTT–FFT algorithm has both the square–logarithmic complexity and reasonable accuracy. This work is started in [53]. More examples are provided by the numerical experiments in Sec. 7.

5. Real–valued transforms using FFT–QTT algorithm

Many problems are formulated in real-valued arithmetic but can be efficiently solved using the Fourier transform. For example, consider the discrete *convolution* of two real–valued vectors, defined as follows

$$f(j) = \sum_{k=0}^{n-1} h(j-k)g(k), \qquad j = 0, \ldots, n-1.$$

In matrix-vector form $f = Tg$ where $T = [t(j,k)]^{n-1}_{j,k=0}$ is the Toeplitz matrix with elements $t(j,k) = h(j-k)$. Each $n \times n$ Toeplitz matrix is the leading submatrix of some $2n \times 2n$ circulant matrix

$$C = \begin{bmatrix} T & * \\ * & T \end{bmatrix}, \qquad C = [c(j,k)], \quad \text{where} \quad c(j,k) = \hat{c}(j-k \mod 2n).$$

To correspond with the matrix T, elements of C are defined as follows

$$\begin{array}{ll} \hat{c}(k) = h(k), & k = 0, \ldots, n-1, \\ \hat{c}(n) & \text{is arbitrary}, \\ \hat{c}(k) = h(k-2n), & k = n+1, \ldots, 2n-1. \end{array}$$

Circulant matrices are diagonalized by the unitary Fourier matrix [12] as follows

$$C = F^* \Lambda F, \qquad \Lambda = \sqrt{2n}\, \mathrm{diag}(F\hat{c}).$$

The multiplication by the Toeplitz matrix T can be performed as follows

$$\begin{bmatrix} f \\ * \end{bmatrix} = F^* \Lambda F \begin{bmatrix} g \\ 0 \end{bmatrix}, \tag{20}$$

which involves three Fourier transforms of size $2n$ and can be done 'at FFT speed'. The result is the complex–valued vector representing the real–valued convolution. In standard computations we can easily take the real or imaginary part of a vector. For vectors or tensors represented in the QTT format such operation is not straightforward. The algorithm is given by the constructive proof of the following theorem.

Theorem 4. If the complex-valued tensor $\mathbf{X} = [x(k_1, \ldots, k_d)]$ is represented by the tensor train format (1) with ranks $r_0, r_1, ..., r_{d-1}, r_d$, it can be represented by the following tensor train with TT–ranks $r_0, 2r_1, ..., 2r_{d-1}, r_d$, where all cores but one are real-valued.

$$x(k_1, \ldots, k_d) = \hat{X}^{(1)}_{k_1} \ldots \hat{X}^{(d)}_{k_d}$$

$$\hat{X}^{(1)}_{k_1} = \begin{bmatrix} B^{(1)}_{k_1} & C^{(1)}_{k_1} \end{bmatrix}, \quad \hat{X}^{(p)}_{k_p} = \begin{bmatrix} B^{(p)}_{k_p} & C^{(p)}_{k_p} \\ -C^{(p)}_{k_p} & B^{(p)}_{k_p}. \end{bmatrix}, \quad \hat{X}^{(d)}_{k_d} = \begin{bmatrix} B^{(d)}_{k_d} \\ -C^{(d)}_{k_d} \end{bmatrix} + i \begin{bmatrix} C^{(d)}_{k_d} \\ B^{(d)}_{k_d} \end{bmatrix}, \tag{21}$$

where $p = 2, \ldots, d-1$ and $B^{(q)}_{k_q} = \Re X^{(q)}_{k_q}$, $C^{(q)}_{k_q} = \Im X^{(q)}_{k_q}$ for $q = 1, \ldots, d$.

Proof. Start from

$$X^{(1)}_{k_1} = \begin{bmatrix} B^{(1)}_{k_1} & C^{(1)}_{k_1} \end{bmatrix} \begin{bmatrix} I \\ iI \end{bmatrix},$$

where I denotes the identity $r_1 \times r_1$ matrix. Define the real-valued core $\hat{X}^{(1)}_{k_1} = \begin{bmatrix} B^{(1)}_{k_1} & C^{(1)}_{k_1} \end{bmatrix}$ and multiply $\begin{bmatrix} I & iI \end{bmatrix}^T$ to the second core of tensor train as follows

$$\begin{bmatrix} I \\ iI \end{bmatrix} X^{(2)}_{k_2} = \begin{bmatrix} I \\ iI \end{bmatrix} \left(B^{(2)}_{k_2} + iC^{(2)}_{k_2} \right) = \begin{bmatrix} B^{(2)}_{k_2} + iC^{(2)}_{k_2} \\ -C^{(2)}_{k_2} + iB^{(2)}_{k_2} \end{bmatrix} = \begin{bmatrix} B^{(2)}_{k_2} & C^{(2)}_{k_2} \\ -C^{(2)}_{k_2} & B^{(2)}_{k_2} \end{bmatrix} \begin{bmatrix} I \\ iI \end{bmatrix} = \hat{X}^{(2)}_{k_2} \begin{bmatrix} I \\ iI \end{bmatrix},$$

where in the right-hand side $\hat{X}^{(2)}_{k_2}$ is the new real-valued core and I is $r_2 \times r_2$ identity matrix. Continue the process and establish (21). □

Corollary 2. The representation (21) allows to compute the real and imaginary parts of a tensor train (1) as follows

$$\Re x(k_1, \ldots, k_d) = \left(\prod_{p=1}^{d-1} \hat{X}^{(p)}_{k_p} \right) \Re \hat{X}^{(d)}_{k_d}, \qquad \Im x(k_1, \ldots, k_d) = \left(\prod_{p=1}^{d-1} \hat{X}^{(p)}_{k_p} \right) \Im \hat{X}^{(d)}_{k_d}.$$

Example 1. Obtain the QTT representation for cosine and sine vectors $[\cos \alpha k]_{k=0}^{2^d-1}$ and $[\sin \alpha k]_{k=0}^{2^d-1}$. They are the real and imaginary part of the exponential vector, which has rank-one QTT representation (5) with QTT–cores

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

We use (21) and form the real-valued cores $\hat{X}^{(1)}_{k_1} = \begin{bmatrix}\cos \alpha k_1 & \sin \alpha k_1\end{bmatrix}$ and

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

We result in

$$\begin{aligned} \cos \alpha j &= \begin{bmatrix} \cos \alpha j_1 \\ \sin \alpha j_1 \end{bmatrix}^T \cdots \begin{bmatrix} \cos 2^{p-1}\alpha j_p & \sin 2^{p-1}\alpha j_p \\ -\sin 2^{p-1}\alpha j_p & \cos 2^{p-1}\alpha j_p \end{bmatrix} \cdots \begin{bmatrix} \cos 2^{d-1}\alpha j_d \\ -\sin 2^{d-1}\alpha j_d \end{bmatrix}, \\ \sin \alpha j &= \begin{bmatrix} \cos \alpha j_1 \\ \sin \alpha j_1 \end{bmatrix}^T \cdots \begin{bmatrix} \cos 2^{p-1}\alpha j_p & \sin 2^{p-1}\alpha j_p \\ -\sin 2^{p-1}\alpha j_p & \cos 2^{p-1}\alpha j_p \end{bmatrix} \cdots \begin{bmatrix} \sin 2^{d-1}\alpha j_d \\ \cos 2^{d-1}\alpha j_d \end{bmatrix}. \end{aligned} \tag{22}$$

This is the *simultaneous representation* of cosine and sine vectors in the common QTT form. The QTT–ranks are two, which follows from the existence of the rank-one QTT representation for the exponential vector and the simultaneous representation of the real and imaginary part of complex tensor train, ensured by Theorem 4.

Example 2. Given a vector x, compute the orthogonal DCT-II (discrete cosine even type-2) transform, defined as follows

$$y(j) = \sum_{k=0}^{n-1} x(k) \cos\left(\frac{\pi}{n} j(k + 1/2)\right), \qquad j = 0, \ldots, n-1.$$

The history of this and related cosine transforms, including the applications to signal and image processing, can be found in a nice review [58]. The DCT-II can be computed through the Fourier transform of size $2n$ applied to the vector x expanded by zeros,

$$y(j) = \Re\left[\exp\left(-\frac{\pi i}{2n} j\right) \sum_{k=0}^{2n-1} \hat{x}(k) \exp\left(-\frac{2\pi i}{2n} jk\right)\right],$$

where $\hat{x}(0 : n-1) = x$ and $\hat{x}(n : 2n-1) = 0$. In computational practice double-sized vectors and transforms are usually avoided for the sake of efficiency. However, with QTT approach, double sized vector/transform does not require double storage/cost. If $n = 2^d$ and x is given in the QTT format (4), the vector $\hat{x}$ has the following QTT representation,

$$\hat{x}(k) = \hat{x}(\overline{k_1 \ldots k_d k_{d+1}}) = X^{(1)}_{k_1} \ldots X^{(d)}_{k_d}(1 - k_{d+1}),$$

which has the same QTT–ranks as x. To compute y, we need to apply QTT–FFT algorithm 2 to find $\hat{y} = F_{d+1}\hat{x}$, which requires $\mathcal{O}((d+1)^2 R^3)$ operations, asymptotically equal to the complexity of the transform of size 2^d. Then we need to consider "top" half-vector of $\hat{y}$ as explained by (12) and multiply the result pointwisely by the exponential vector. These operations do not change the QTT ranks. Finally, we apply Theorem 4 and obtain the real-valued cosine transform with the QTT–ranks twice as large as the ones of the Fourier transform.

Finally, we should mention that the proposed approach to the computation of convolution and cosine transform can be generalized to other types of trigonometric transforms as well as to the higher dimensions.

6. Discrete Fourier transform in higher-dimensional case

The m–dimensional normalized Fourier transform is defined as follows.

$$y(j_1,\ldots,j_m)=\sum_{k_1=0}^{n_1-1}\ldots\sum_{k_m=0}^{n_m-1}\frac{x(k_1,\ldots,k_m)}{\sqrt{n_1\ldots n_m}}\exp\left(-\frac{2\pi i}{n_1}j_1k_1\right)\ldots\exp\left(-\frac{2\pi i}{n_m}j_mk_m\right), \quad (23)$$

where $j_q=0,\ldots,n_q-1$ for $q=1,\ldots,m$. If $x=[x(k_1,\ldots,k_m)]$ and $y=[y(j_1,\ldots,j_m)]$ are considered as $n_1\ldots n_d$ vectors, the m-dimensional Fourier transform is the Kronecker product of m one-dimensional transforms, i.e., $F_{d_1,\ldots,d_m}=F_{d_m}\otimes F_{d_{m-1}}\otimes\ldots\otimes F_{d_2}\otimes F_{d_1}$. Using the tensor train representation (1), we have

$$\begin{aligned} x(k_1,k_2,\ldots,k_m)&=X^{(1)}_{k_1}X^{(2)}_{k_2}\ldots X^{(m)}_{k_m}, \qquad y(j_1,j_2,\ldots,j_m)=Y^{(1)}_{j_1}Y^{(2)}_{j_2}\ldots Y^{(m)}_{j_m},\\ Y^{(q)}_{j_q}&=\frac{1}{\sqrt{n_q}}\sum_{k_q=0}^{n_q-1}X^{(q)}_{k_q}\exp\left(-\frac{2\pi i}{n_q}j_qk_q\right), \qquad q=1,\ldots,m. \end{aligned} \quad (24)$$

We see that due to the perfect separation of variables, Fourier transform is independently applied to all TT–cores. The multidimensional Fourier transform does not change the TT–ranks. Therefore, Eq. (24) is the straightforward algorithm to compute the m–dimensional Fourier transform of a $n\times n\times\ldots\times n$ vector with $\mathcal{O}(mR^2n\log n)$ complexity, where R is the maximal TT–rank of a vector.

To reach the square–logarithmic complexity (of course, for a special type of input data), we assume $n_p=2^{d_p}$ and use binary notation (3) for all mode indices. The input array is represented as follows

$$x(k_1,\cdots,k_m)=x(\overline{k_{1,1}\ldots k_{1,d_1}},\cdots,\overline{k_{m,1}\ldots k_{m,d_m}})=\prod_{q=1}^{m}\prod_{p_q=1}^{d_q}X^{(q,p_q)}_{k_{q,p_q}}. \quad (25)$$

Then the QTT–FFT Algorithm 2 can be used to compute unimodular Fourier transforms subsequently. The QTT–FFT introduces the error, controlled by Theorem 2. The appropriate TT–orthogonalization should be used for interfaces to guarantee that this error will not be amplified in the whole m–dimensional vector. The multi-dimensional version of QTT–FFT is summarized by Algorithm 3 and visualized on Fig. 4. The accuracy and complexity are estimated similarly to the one-dimensional case.

Theorem 5. If the accuracy-based criterion ε is used on each approximation step in Algorithm 3, the accuracy of the result is the following

$$\|y-F_{d_1,\ldots,d_m}x\|\leqslant\sum_{q=1}^{m}d_q\varepsilon\|y\|.$$

Theorem 6. The complexity of Algorithm 3 for m–dimensional transform of an $n\times n\times\ldots\times n$ vector with $n=2^d$ is not larger than $\mathcal{O}(md^2R^3)$, where R is the maximum QTT–rank of all intermediate vectors at all steps of Algorithm 3 and all internal calls to Algorithm 2.

Proof. On Line 1 of Algorithm 3 the TT–orthogonalization of the tensor train with md cores is performed, which requires not more than $\mathcal{O}(mdR^3)$ operations. Then the Algorithm 2 and TT–orthogonalization are applied to m subtrains with d cores each, which costs not more than $m\mathcal{O}(d^2R^3)+m\mathcal{O}(dR^3)$. The last step does not require any operations. Therefore, we obtain $\mathcal{O}(md^2R^3)$ estimate, which completes the proof. □

Algorithm 3: Multi-dimensional QTT–FFT, approximation

Input: Vector x in the QTT format (25), accuracy ε or rank bounds $R_{1,1}, \ldots, R_{m,d_m-1}$
Output: Approximation $y \approx F_{d_1,\ldots,d_m} x$ of the Fourier transform (23) in the QTT format (26)

1: Orthogonalize input tensor train right-to-left using TT–orthogonalization [46]
2: **for** $q = 1, 2, \ldots, m$ **do**
3: Apply QTT–FFT Algorithm 2 *without* the last bit-reverse step to the subtrain $\prod_{p_q=1}^{d_q} X_{k_{q,p_q}}^{(q,p_1)}$. The result appears as $\prod_{p_q=1}^{d_q} \left(Y_{k_{q,d_q-p_q+1}}^{(q,d_q-p_q+1)}\right)^T$
4: If $q \neq m$, make cores of the subtrain $\prod_{p_q=1}^{d_q} \left(Y_{k_{q,d_q-p_q+1}}^{(q,d_q-p_q+1)}\right)^T$ left-orthogonal
5: **end for**
6: Reverse the tensor train $\prod_{q=1}^{m} \prod_{p_q=1}^{d_q} \left(Y_{k_{q,d_q-p_q+1}}^{(q,d_q-p_q+1)}\right)^T$ and obtain (26).

Note that due to the bit-reverse permutation issues, the order of mode indices in the tensor train on output is reversed as follows.

$$y(j_1, \cdots, j_m) = y(\overline{j_{1,1} \ldots j_{1,d_1}}, \cdots, \overline{j_{m,1} \ldots j_{m,d_m}}) = \prod_{q=m}^{1} \prod_{p_q=1}^{d_q} Y_{k_{q,p_q}}^{(q,p_q)}. \tag{26}$$

The reason is that every unimodular QTT transform is performed by Algorithm 2 with the bit-reverse permutation of the result. In the 1D case, the bit-reverse permutation is made without any computations by simple reordering of the transposed QTT cores. If we apply this reordering to each unimodular subtrain individually, the TT–ranks (ranks separating the physical variables) will not be consistent. Instead, we reverse the tensor train globally and obtain the result where the order of bits in the QTT representation is correct, but the order of frequency modes is reversed compared with the original ordering of space modes, as shown on Fig. 4. Fortunately, a wide class of applications (e.g. convolution, solution of discretized PDEs) requires *two* successive Fourier transforms. In this case the second transform will recover the original ordering.

This drawback of the multi-dimensional QTT–FFT can be removed using a one-dimensional QTT–FFT algorithm based on the self–sorting FFT [22].

7. Numerical examples

In this section we compare the accuracy and timings of the trigonometric transforms for one–, two– and three–dimensional functions with moderate QTT–ranks. From the proof of Theorem 3 we see that the complexity of the QTT–FFT depends on all QTT–ranks. The maximum QTT–rank used in (19) sometimes gives incorrect impression of "structure complexity" of particular data array. To account the distribution of all QTT–ranks $r_0, \ldots, r_d$, we define the *effective QTT–rank*[3] of a vector as the positive solution r of the quadratic equation $\mathtt{mem}(r_0, r_1, r_2, \ldots, r_{d-1}, r_d) = \mathtt{mem}(r_0, r, r, \ldots, r, r_d)$, where $\mathtt{mem}$ measures the storage for the QTT format as follows

$$\mathtt{mem}(r_0, r_1, r_2, \ldots, r_{d-1}, r_d) = r_0 r_1 + r_1 r_2 + \ldots + r_{d-1} r_d.$$

[3]This definition was proposed by E. E. Tyrtyshnikov

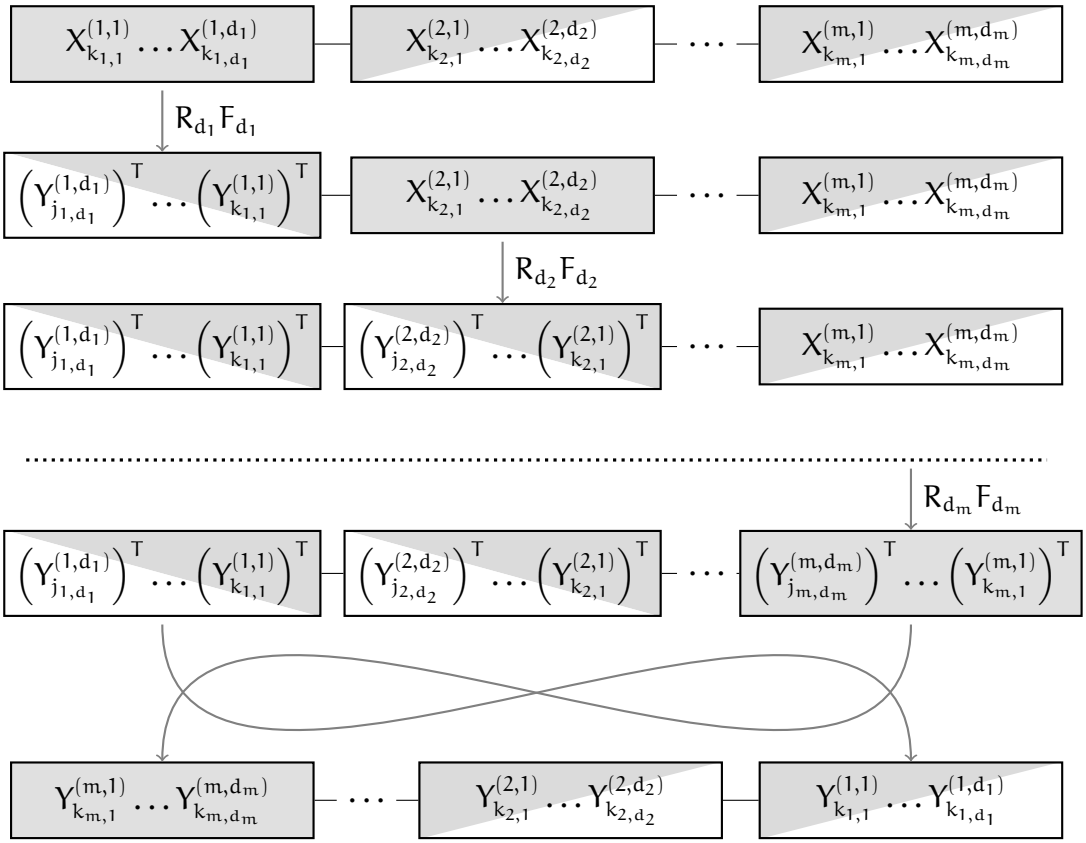

Figure 4. Visualization of the QTT–FFT Algorithm 3.

The effective QTT–rank is generally non-integer.

For experiments we use one Xeon E5504 CPU at 2.0 GHz with 72 GB of memory in the Institute of Numerical Mathematics, Russian Academy of Sciences, Russia. The TT subroutines and QTT–FFT algorithms are implemented in Fortran 90 by the third author. We use Intel Fortran Composer XE version 12.0 (64 bit) and BLAS/LAPACK and FFTW packages provided with MKL library.

7.1. Fourier images in 1D

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

$$\widehat{f}(\xi) = \int_{-\infty}^{+\infty} f(t)\exp(-2\pi i t\xi)dt. \tag{27}$$

We consider the *rectangle pulse* function, for which the Fourier transform is known,

$$\Pi(t) = \begin{cases} 0, & \text{if} \quad |t| > 1/2 \\ 1/2, & \text{if} \quad |t| = 1/2, \\ 1 & \text{if} \quad |t| < 1/2, \end{cases} \qquad \widehat{\Pi}(\xi) = \mathrm{sinc}(\xi) \stackrel{\text{def}}{=} \frac{\sin \pi\xi}{\pi\xi}. \tag{28}$$

Table 1. Time for QTT–FFT (in milliseconds) w.r.t. size $n = 2^d$ and accuracy ε. Here $\texttt{time}_{\text{QTT}}$ is the runtime of Algorithm 2, $\texttt{time}_{\text{FFTW}}$ is the runtime of the FFT algorithm from the FFTW library, and rank $\widehat{f}$ is the effective QTT–rank of the Fourier image.

$f = \Pi(t)$		$\varepsilon = 10^{-4}$		$\varepsilon = 10^{-8}$		$\varepsilon = 10^{-12}$	
d	$\texttt{time}_{\text{FFTW}}$	rank $\widehat{f}$	$\texttt{time}_{\text{QTT}}$	rank $\widehat{f}$	$\texttt{time}_{\text{QTT}}$	rank $\widehat{f}$	$\texttt{time}_{\text{QTT}}$
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
50	—	3.96	39.3	6.45	70.5	8.48	109
60	—	3.69	50.0	6.25	87.6	8.32	133

The Fourier integral is approximated by the rectangle rule. Since $f(t) = \Pi(t)$ is real and even, we write

$$\widehat{f}(\xi_j) = 2\Re \int_0^{+\infty} f(t)\exp(-2\pi i t\xi_j)\,dt \approx 2\Re \sum_{k=0}^{n-1} f(t_k)\exp(-2\pi i t_k \xi_j) h_t, \tag{29}$$

where $t_k = (k + 1/2)h_t$, $\xi_j = (j + 1/2)h_\xi$ and $k, j = 0, \ldots, n-1$. If $h_t h_\xi = 1/n$ and $n = 2^d$, the sum in the right-hand side can be computed by DFT as follows

$$\widehat{f} \approx \tilde{f} = 2^{d/2+1} h_t \Re\left(\omega_{d+2}\Omega_d F_d \Omega_d f\right), \qquad \widehat{f} = [\widehat{f}(\xi_j)]_{j=0}^{2^d-1}, \quad f = [f(t_k)]_{k=0}^{2^d-1}. \tag{30}$$

For $h_t = h_\xi = \frac{1}{2^{d/2}}$ and d even, the QTT representation of the rectangular pulse has QTT–ranks one, i.e.,

$$\Pi(t_k) = \Pi(h/2 + \overline{k_1 \ldots k_{d/2-1}}h + \overline{k_{d/2} \ldots k_d}/2) = (1 - k_{d/2}) \ldots (1 - k_d).$$

We can use QTT–FFT Algorithm 2 to compute DFT in (30). According to Theorem 2 the relative accuracy of the result is ε if the tolerance parameter is set to ε/d. The QTT–ranks and timings will grow for smaller ε. In Table 1 we compare the runtime of Algorithm 2 for different ε with the runtime of the FFT out-of-place algorithm from the FFTW library. The timings are rather small and were obtained by averaging the computation time over a large number of executions. We see that for the considered example the QTT–FFT algorithm outperforms the FFT algorithm from FFTW library for $n \geqslant 2^{20}$, even for very high accuracy. We also see that the use of moderate or low accuracy for QTT–FFT significantly reduces the QTT–ranks of result and speeds up the computation. That is not the case for the standard FFT algorithm.

The accuracy of the rectangle rule (30) increases for smaller step size h. For fixed frequency ξ we can expect $\mathcal{O}(h^2) = \mathcal{O}(n^{-1})$ convergence, but for very large frequencies ξ_j,

Table 2. Accuracy verification for the QTT–FFT Algorithm 2 and $f = \Pi(t)$. The desired accuracy is $\varepsilon = 10^{-13}$, grid size $h = 2^{-d/2}$, and acc denotes the relative accuracy in Frobenius norm of (30) for subvectors corresponding to $0 \leqslant \xi_j \leqslant 16$.

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
acc	2_{10}−5	5_{10}−6	1_{10}−6	3_{10}−7	8_{10}−8	2_{10}−8	2_{10}−11	2_{10}−13	2_{10}−13

$j \gtrsim n/2$, Eq. (29) is not accurate since the period of exponent $\exp(-2\pi x\xi_j)$ is less $2h$. Therefore, we check the accuracy of (30) for a subvectors corresponding to $0 \leqslant \xi_j \leqslant 16$. The results are shown in Table 2, where

$$\mathtt{acc} = \|\widehat{f} - \widetilde{f}\|_{[0,16]} / \|\widehat{f}\|_{[0,16]}, \qquad \text{where} \qquad \|f\|^2_{[0,16]} = \sum_{j:0\leqslant\xi_j\leqslant 16} |f(\xi_j)|^2.$$

The computations for $n \leqslant 2^{28}$ can be performed by FFTW or FFT–QTT. For $n > 2^{28}$ the one–processor subroutine from FFTW can not be used due to storage limitations and data–sparse QTT format is necessary to increase the accuracy of the computed Fourier image. For $d \leqslant 28$ the accuracy of QTT–FFT was verified by comparison with FFTW and the QTT–FFT error was always 'under control', i.e., less than the desired accuracy.

7.2. Fourier images in 3D

In larger dimensions the one-dimensional size of the array that can be processed by standard FFT algorithm is severely restricted by computing resources. Therefore, the discretization accuracy obtained on simple workstations is often low and multiprocessor parallel platforms have to be used for larger grids. As well as in 1D case, the use of data-structured QTT–FFT algorithm relaxes/removes the grid size restrictions and allows to reach higher accuracy of the computation using one CPU. As an example, we compute the following m–dimensional Fourier image from [57]

$$f(x) = \begin{cases} (1-|x|^2)^\delta, & \text{if} \quad |x| \leqslant 1, \\ 0 & \text{otherwise} \end{cases}, \quad \widehat{f}(\xi) = \int_{\mathbb{R}^m} f(x) e^{-2\pi i x\cdot\xi} d\xi = \frac{\Gamma(\delta+1) J_{m/2+\delta}(2\pi|\xi|)}{\pi^\delta |\xi|^{m/2+\delta}}, \tag{31}$$

where Γ is the gamma-function, δ is arbitrary real parameter, J_α is the Bessel function of the first kind of order α, and $x \cdot \xi$ denotes the scalar product. Introducing the uniform tensor product $n \times n \times \ldots \times n$ grids in space and frequency domains with step sizes $h_x = h_\xi = 1/\sqrt{n}$ and $n = 2^d$, we approximate (31) by the rectangle quadrature rule. The result can be computed by m-dimensional DFT.

We perform the experiments for the case $m = 3$, where DFT can be computed both by FFTW and QTT–FFT. The accuracy of Algorithm 3 is verified by comparison with the analytical answer (31). Since the comparison at all 2^{dm} points is unfeasible, the accuracy is defined as follows

$$\mathtt{acc} = \|\widehat{f} - \widetilde{f}\|_{\mathrm{diag}[0,8]^3} / \|\widehat{f}\|_{\mathrm{diag}[0,8]^3}, \qquad \text{where} \qquad \|f\|^2_{\mathrm{diag}[0,8]^3} = \sum_{j:0\leqslant\xi_j\leqslant 8} |f(\xi_j,\xi_j,\xi_j)|^2.$$

The results are shown in Table 3. For $d = 8$ the accuracy of QTT–FFT is verified also by comparison with the FFTW algorithm.

7.3. Convolution in 3D

In the scientific computing the Fourier transform is often used for the computation of convolutions. As a motivating example, consider the evaluation of the Newton potential,

$$V(x) = \frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{f(y)}{\|x-y\|} dy, \tag{32}$$

which plays an important role in the electronic structure calculations based on the Hartree-Fock equation. In many chemical modelling programs, e.g., PC GAMESS and MOLPRO, the *electron density* function $f(y)$ is represented as a sum of polynomially weighted 3D Gaussians, for which (32) 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 elements involved. The possible alternative may be the discretization on the $n \times n \times n$ tensor grids using standard Galerkin/collocation schemes with piecewise–constant or –linear basis elements. The electron density function has strong cusps in the positions of the nuclei, that motivates the use of very precise grids. The tensor-structured methods lead to almost linear or even logarithmic complexity w.r.t. n that make the use of very fine grids and high accuracy of computations possible without a special choice of basis. In this way, fast tensor-product convolution with the Newton kernel based on the canonical or/and Tucker tensor decompositions was developed in [34, 55, 31] and applied to the fully discrete grid-based solution of the Hartree-Fock equation in [36, 27]. These algorithms reduce the three-dimensional convolution to a number of one-dimensional ones, which are computed by FFT in $\mathcal{O}(n \log n)$ operations. The QTT representation of the canonical vectors reduces the complexity to the logarithmic scale w.r.t. n, see [29]. The multidimensional convolution of QTT-structured data can be computed also directly using the analytical QTT decomposition of a multilevel Toeplitz matrix [26]. We refer also to [30, 10, 52, 35, 47, 14, 56, 28] for the detailed discussions on recent progress in this field.

To discretize (32), we reduce the integration interval to $B \stackrel{\text{def}}{=} [-1/2, 1/2]^3$, assuming that $f(y)$ vanishes outside this cube, and introduce the uniform $n \times n \times n$ tensor-product grid in B. We consider two discretization methods: collocation and Nyström–type. Collocation method approximates $f(y)$ by a piecewise–constant function and leads to

$$V(x) \approx V_c(x) \stackrel{\text{def}}{=} \frac{1}{4\pi} \sum_k f(y_k) \int_{B_k} \frac{dy}{\|x-y\|}, \qquad V(x_j) \approx \nu_c(j) \stackrel{\text{def}}{=} V_c(x_j), \tag{33}$$

Table 3. Relative accuracy `acc` of 3D Fourier image (31)

	FFTW	QTT–FFT		
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	$8_{10}-7$	$8_{10}-7$	$1_{10}-8$	$2_{10}-10$

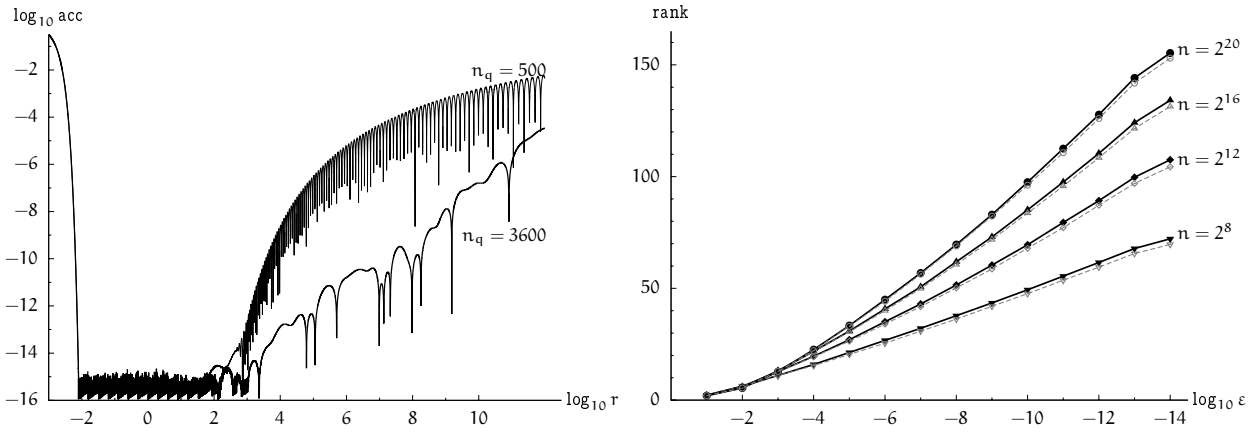

Figure 5. (left) Pointwise relative accuracy of the sinc–quadrature [19, eq. (5.3)] with n_q terms for r^{-1}. (right) QTT–ranks of the first column of Newton potential matrix for grid size n and relative accuracy ε. Solid lines — collocation method (33), dashed lines — Nyström–type method (34).

where $x_j = (h + 1/2)j$, $y_k = (h + 1/2)k$, B_k is $h \times h \times h$ cube centered at y_k, $j = (j_1, j_2, j_3)$, $k = (k_1, k_2, k_3)$ and $j_p, k_p = 0, \ldots, n-1$ for $p = 1, 2, 3$. As shown in [31], the pointwise accuracy of this approximation is

$$|V(x_j) - \nu_c(j)| \leqslant Ch^2.$$

Nyström–type convolution appears if we write (32) in points $x = x_j$ and then approximate integral by the rectangle quadrature rule over B as follows

$$V(x_j) \approx \nu_n(j) \stackrel{\text{def}}{=} \frac{1}{4\pi} \sum_k f(y'_k) \frac{h^3}{\|x_j - y'_k\|}, \tag{34}$$

where $y'_k = kh$ and j, k are defined as earlier. Since $y'_k \neq x_j$ for all j and k, the right-hand side in this equation is well defined. The standard accuracy estimate of the rectangle rule is not applicable to the singular functions and we did not find the accuracy estimate of this method in the literature. Surprisingly, it can be proven that the accuracy of the Nystöm–type convolution for the 3D Newton potential of smooth functions is estimated as

$$|V(x_j) - \nu_n(j)| \leqslant Ch^2 |\log h|,$$

which is almost the same order of convergence, as for the collocation method. We will report the proof elsewhere.

Note that the Nyström–type method 'shifts' the result, i.e., grids for $f(y)$ and $V(x)$ do not coincide. This can be a drawback for some applications. However, the coefficients of the convolution core in (34) are easier to compute than the ones in (33).

Both (33) and (34) represent the three–dimensional discrete convolution. By (20), it can be computed by three three–dimensional DFTs of size $(2n)^3$. This operation is highly restricted by the value of n and for $n \gtrsim 2^9$ is not feasible for full-sized vectors using a standard FFT algorithm on one processor. If $f(y)$ has moderate QTT–ranks, the Newton potential can be efficiently computed using QTT–FFT Algorithm 3. To do this, we need the

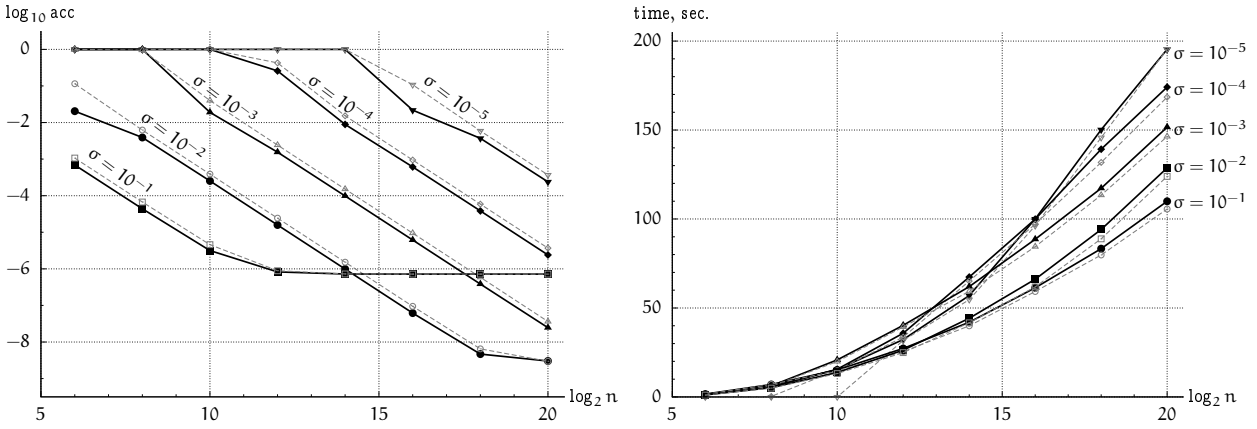

Figure 6. Accuracy (left) and computation time (right) for the Newton potential of the Gaussian charge density (35) with width σ on the uniform tensor–product $n \times n \times n$ grid. Solid lines — collocation method (33), dashed lines — Nyström–type method (34). Approximation accuracy for the QTT–FFT is $\varepsilon = 10^{-10}$.

QTT representation of the first column of the Newton potential matrix, which is different for the collocation and Nyström–type methods. Similar problem was discussed for the representation of Newton/Yukawa potential in canonical format [3], and was solved using the sinc–quadrature from [19], which expresses r^{-1} as the sum of Gaussian functions, i.e.,

$$r^{-1} \approx \sum_{q=1}^{n_q} b_q \exp(-a_q r^2) = \sum_{q=1}^{n_q} b_q \exp(-a_q x^2) \exp(-a_q y^2) \exp(-a_q z^2), \quad r^2 = x^2 + y^2 + z^2.$$

Gaussians are perfectly separable in physical modes and the QTT representation of each term can be constructed as the Kronecker product of one-dimensional Gaussians, for which we can use full-to-TT algorithm [44]. The QTT–ranks of a one-dimensional Gaussian are bounded by $\mathcal{O}(\log^{1/2} \varepsilon^{-1})$ [7].

The pointwise relative accuracy of the sinc-quadrature for different r is shown on Fig. 5(left). We see that for $\frac{r_{max}}{r_{min}} \approx n \lesssim 10^4$, the quadrature with $n_q = 500$ terms provides almost the machine threshold accuracy. For $n \leqslant 10^6$, the quadrature with $n_q = 3600$ terms has the pointwise relative accuracy $\varepsilon = 10^{-13}$, which is enough for our purposes. Since the potential should be constructed only once (there is no adaptation of grid, etc.), the large n_q is not restrictive. In Fig. 5(right) we show the QTT–ranks of the convolution cores (33) and (34) w.r.t grid size n and relative accuracy level ε.

To check the accuracy and speed of the proposed method, we compute the Hartree potential (32) for the Gaussian charge density $f = g_{0,\sigma}$ for different σ and compare the result with the analytical answer,

$$g_{a,\sigma}(y) = \frac{1}{(\sqrt{2\pi}\sigma)^3} \exp\left(-\frac{|y-a|^2}{2\sigma^2}\right), \qquad V(x) = \frac{1}{4\pi|x-a|} \operatorname{erf}\left(\frac{|x-a|}{\sqrt{2}\sigma}\right). \tag{35}$$

Here a, σ are the center and width of the Gaussian and $\operatorname{erf}(x) \stackrel{\text{def}}{=} \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt$ is the error–function. Accuracy and timings of the evaluation of (35) using 3D convolution are shown on Fig. 6. The relative accuracy in the Frobenius norm is measured over one line of grid points, $\{x_j\} \times \{x_{n/2}\} \times \{x_{n/2}\}$, $j = 0, \ldots, n-1$.

It is interesting, that on the single graph on Fig. 6(left) we can note three *error plateaus* which appear for different reasons. The right plateau for $\sigma = 10^{-1}$ appears on the level $\mathtt{acc} \approx 10^{-6}$ due to the error, introduced by the truncation of the integral away from the cube B. It is easy to notice, that on the boundary of B the Gaussian with $\sigma = 10^{-1}$ does not vanish to machine zero,

$$\exp\left(-\frac{x^2}{2\sigma^2}\right)\Big|_{x=1/2} \approx 10^{-6}.$$

Gaussian charges with smaller σ represent stronger cusps and do not have this error visible. Right plateau for the Gaussian with $\sigma = 10^{-2}$ appears due to the approximation error in QTT–FFT, which is set to $\varepsilon = 10^{-10}$. Finally, left plateau for $\sigma = 10^{-3}, \sigma = 10^{-4}$ and $\sigma = 10^{-5}$ appear since these cusps can not be properly described on coarse grids. Therefore, a precise evaluation of the Newton potential of such functions on the 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 QTT–FFT algorithm are very moderate and scale square-logarithmically w.r.t. d, see Fig. 6(right).

7.4. Signals with sparse Fourier images

Consider a sum of p plane waves in m–dimensional space, defined as follows

$$f(k) = \sum_{p=1}^{R} a_p \exp\left(\frac{2\pi i}{n} f_p \cdot k\right), \qquad f_p \in \mathbb{R}^m, \quad k = \{0, \ldots, n-1\}^m, \quad n = 2^d. \tag{36}$$

Each plane wave has QTT–ranks one (5), which do not depend on the accuracy and vector size. The QTT–ranks of $f = [f(k)]$ are not larger than R. If frequencies are *integer* numbers, $f_p \in \mathbb{Z}^m$, all intermediate vectors of Algorithm 3 also have QTT–ranks one [53]. Therefore, using plain waves we can construct a signal with prescribed QTT–ranks and complexity of the QTT–FFT algorithm, which allows to compare the speed of QTT–FFT and the standard FFT algorithm for different d and R. The results are given on Fig. 7.

We see that the QTT–FFT is asymptotically faster than the full–size FFT for all moderate R that we have tested. However, the *crossover point*, i.e., the minimum size of transform for which QTT–FFT outperforms the standard method, depends on R and m. For example, for $R = 8$ the crossover point for $m = 1$ is $d \approx 20$, for $m = 2$ is $d \approx 19$ and for $m = 3$ is $d \approx 18$. Therefore, the QTT–FFT algorithm performs better for high-dimensional problems than standard FFT algorithm and is not restricted by the problem size. However, the use of QTT–FFT is restricted by large R.

By Theorem 6, the complexity of multi-dimensional QTT–FFT is not larger than $\mathcal{O}(md^2R^3)$. Using signals (36) with different R, we can measure the actual runtime w.r.t. R. The results for signals (36) with $a_p = 1$ and random integer frequencies $f_p \in \mathbb{Z}^m$ are given by the fourth graph on Fig. 7. Surprisingly, the computational time grows *quadratically*, not cubically, w.r.t. parameter R for the large-size transforms. The time of smaller-size transforms (see $N = 2^{12}$) tends to constant for large R, since the QTT–ranks of vector (36) can not exceed the QTT–ranks of full-rank vectors. This constant level is, however, significantly larger than the time of corresponding transform of a dense vector. That shows that our current implementation of QTT–FFT is far not so optimized as the algorithms from FFTW library. We also see that for transforms of the same size $N = n^m$, i.e., with the same

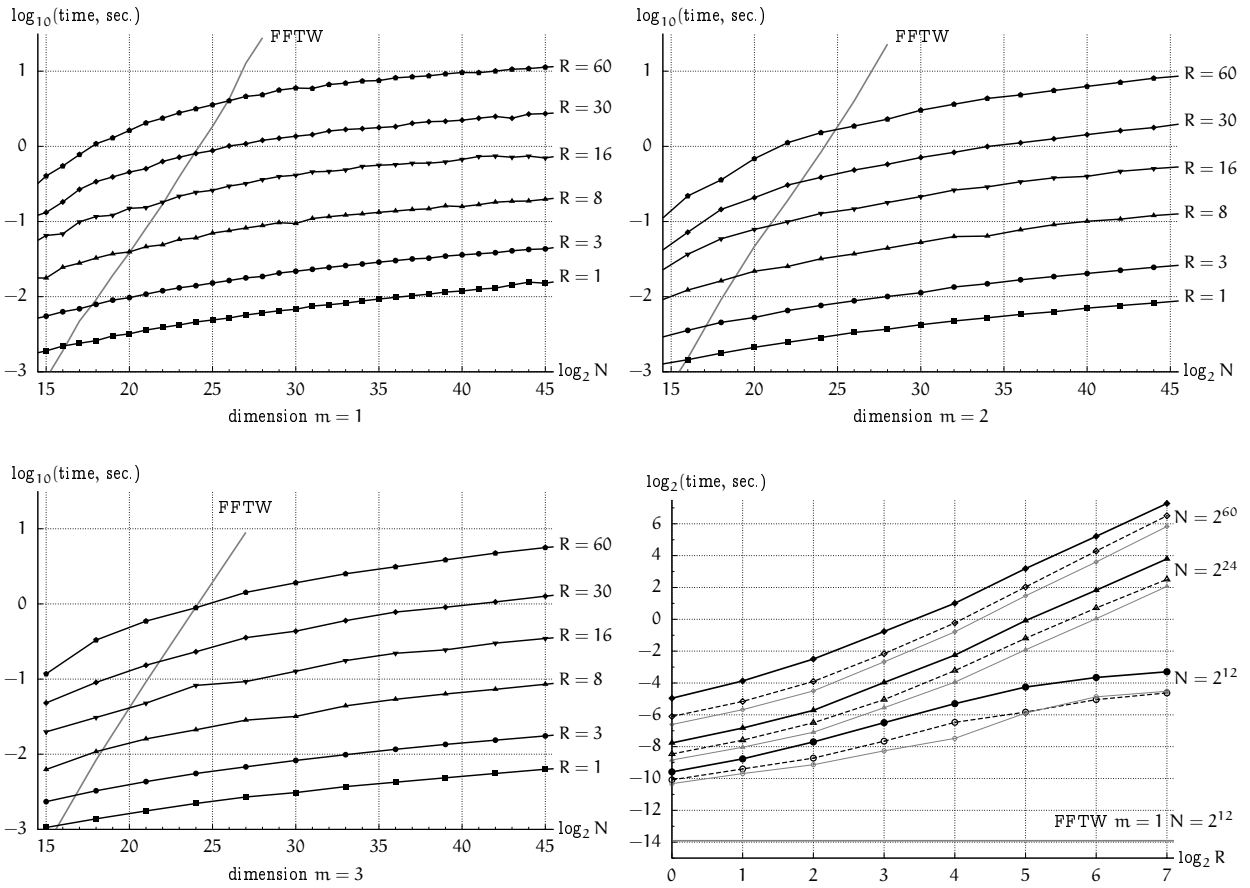

Figure 7. QTT–FFT and FFTW time for R plane waves in m dimensions. Graphs 1–3 show the runtime w.r.t. problem size $N = n^m$ for different R and m. Graph 4 shows the runtime w.r.t R for $m = 1$ (thick solid lines), $m = 2$ (dashed lines), $m = 3$ (thin gray solid lines).

$\log_2 N = m \log_2 n = md$, the transforms with larger m are faster, since the complexity is linear w.r.t. m and quadratic w.r.t. d.

Since the complexity of QTT–FFT is defined by parameters d, m and R of the input vector, the numerical results obtained for plane waves can be generalized to all vectors with QTT ranks bounded by R at all steps of the Cooley–Tukey algorithm. Therefore, the runtimes given on Fig. 7 can be considered as the 'typical' runtimes of QTT–FFT algorithm for signals characterized by m, n and R.

8. Comparison with Sparse Fourier transform

8.1. QTT–FFT and Sparse Fourier transform

The QTT representation is the example of a *data–sparse* representation, i.e., with number of representation parameters asymptotically smaller than the full number of array elements. The sum of plane waves (36) is the particular example of a signal with data–sparse QTT representation: the QTT–ranks of the sum of R plain waves does not exceed R. If $f_p \in \mathbb{Z}^m$, this signal has no more than R non-zero Fourier components and is both QTT–sparse and Fourier–sparse.

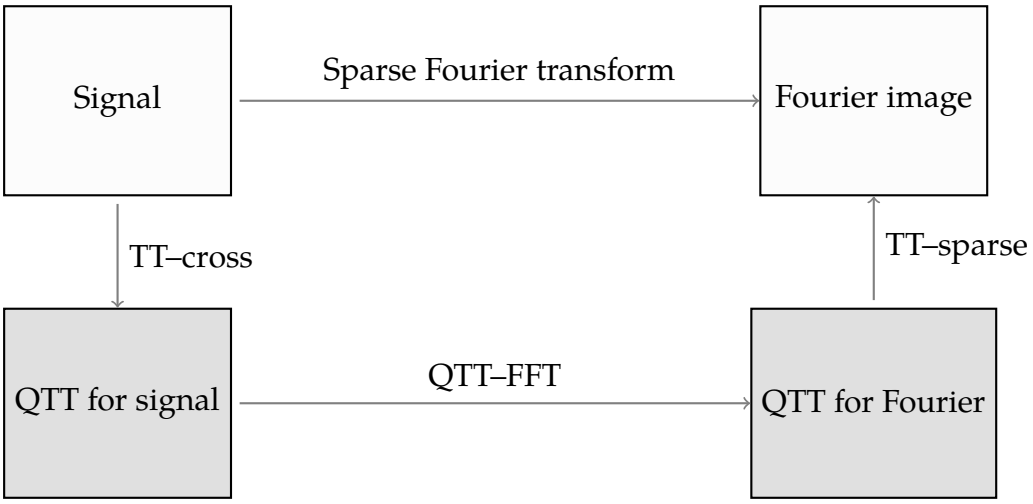

Figure 8. Sparse Fourier transform and QTT–FFT approach

The sparse Fourier transform problem is the following: given a signal of length n, estimate the R largest in magnitude coefficients of its Fourier transform. If we use all n elements of a given vector and the standard FFT algorithm, the answer can be trivially computed in $\mathcal{O}(n \log n)$ operations, which is both the complexity of FFT and sorting algorithm. To reduce the complexity, the heuristic algorithms are developed, which estimate the Fourier-sparse representation with given number of terms R using a subset of *samples* (vector elements), usually selected with a certain randomization. Sparse Fourier transform algorithms are most efficient for the vectors with $o(n)$ non–zero Fourier components. Note that the description of this class of vectors is not explicit, as well as the description of a class of 'suitable' vectors for the QTT–FFT algorithm. The Fourier-sparsity of a general input vector can not be predicted until the Fourier transform is computed.

To compare QTT–FFT with Sparse Fourier transform, we should first explain that the QTT–FFT solves actually the different problem: given a signal in data–sparse QTT form, compute a Fourier transform in the same data–sparse format. The similarity, which allows to compare the QTT–FFT approach with the Sparse Fourier transform, is the following: the class of QTT–sparse vectors includes all Fourier-sparse vectors, since each signal with exactly R non-zero Fourier coefficients has QTT ranks bounded by R (but not vice versa). The differences are the following: QTT–FFT algorithm requires input to be given already in the QTT format, and the output is again the QTT format, which is data–sparse, but not sparse. Two missing components are algorithm that computes the QTT representation of a given signal from a set of samples (TT–cross), and algorithm that converts the QTT format to a sparse vector (TT–sparse). This is illustrated by a scheme on Fig. 8.

Algorithms that reconstruct the tensor train format from samples of a given array are proposed in [49, 54]. They are based on the *maximum–volume* principle [15, 13], which selects proper sets of indices in the array for the interpolation procedure. Maximum–volume algorithm is heuristic, adaptive to the signal and originally non–random, but certain randomness can be included to check the accuracy and/or improve the robustness. The development of the maximum–volume concept to higher dimensions is still at the early stage and "TT–cross" algorithms should be further improved, as well as the background theory.

The "TT–sparse" algorithms were never considered previously, mostly due to the rea-

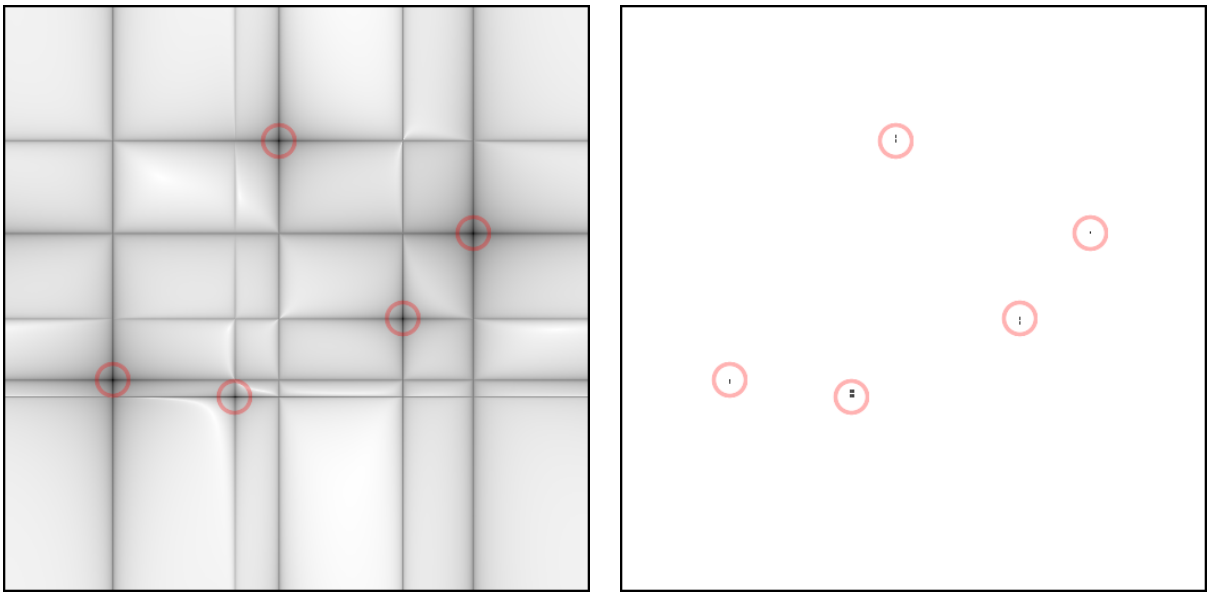

Figure 9. DFT image of 2D data set with $R = 5$ plane waves on $2^d \times 2^d$ grid, $d = 9$. Logarithm of amplitude is shown in 256 levels of gray (black = maximum, white = minimum). From the right, the image after sparsification in the QTT format is shown

son that data–sparse TT/QTT representations are "good enough" for the numerical work and we usually do not need to convert them into the pointwise–sparse format. In order to proof the concept, we can propose a very naive algorithm which sparsifies the TT cores of (1) *independently* as follows

$$A^{(p)}_{k_p} \to \tilde{A}^{(p)}_{k_p}, \quad \tilde{A}^{(p)}_{k_p}(i,j) = \begin{cases} A^{(p)}_{k_p}(i,j), & \text{if} \quad |A^{(p)}_{k_p}(i,j)| \geqslant \mu \max_{k_p} \max_{i,j} |A^{(p)}_{k_p}(i,j)|, \\ 0, & \text{elsewhere,} \end{cases} \tag{37}$$

where μ is the threshold parameter. The sparsified TT format can be easily "decompressed" into the sparse vector.

We show how "TT–cross $\to$ QTT–FFT $\to$ TT–sparse" scheme works for the particular two–dimensional signal, which is the sum of five plain waves (36) with unit amplitudes and randomly taken frequencies. Since frequencies are not integer, the discrete Fourier image of this signal (see Fig. 9, left) is not sparse, but has strong cusps near the positions of f_p. We apply the TT–ACA algorithm [54] (which is the DMRG–type TT–cross algorithm) to reconstruct the QTT representation of signal from a few samples. Then we apply Algorithm 3 and sparsify the DFT image by (37) with $\mu = 0.4$. The result is shown on Fig. 9 from the right and gives the correct positions of all five plane wave frequencies. This example shows that QTT–FFT approach (three–step scheme) can be helpful for the problems where Sparse Fourier transform algorithms are applied.

8.2. Comparison with Sparse Fourier transform algorithms

The history of the Sparse Fourier transform framework can be found in [20]. Since the Sparse Fourier transform is not a central topic of this paper, we will compare our approach only with recent algorithms developed in this field.

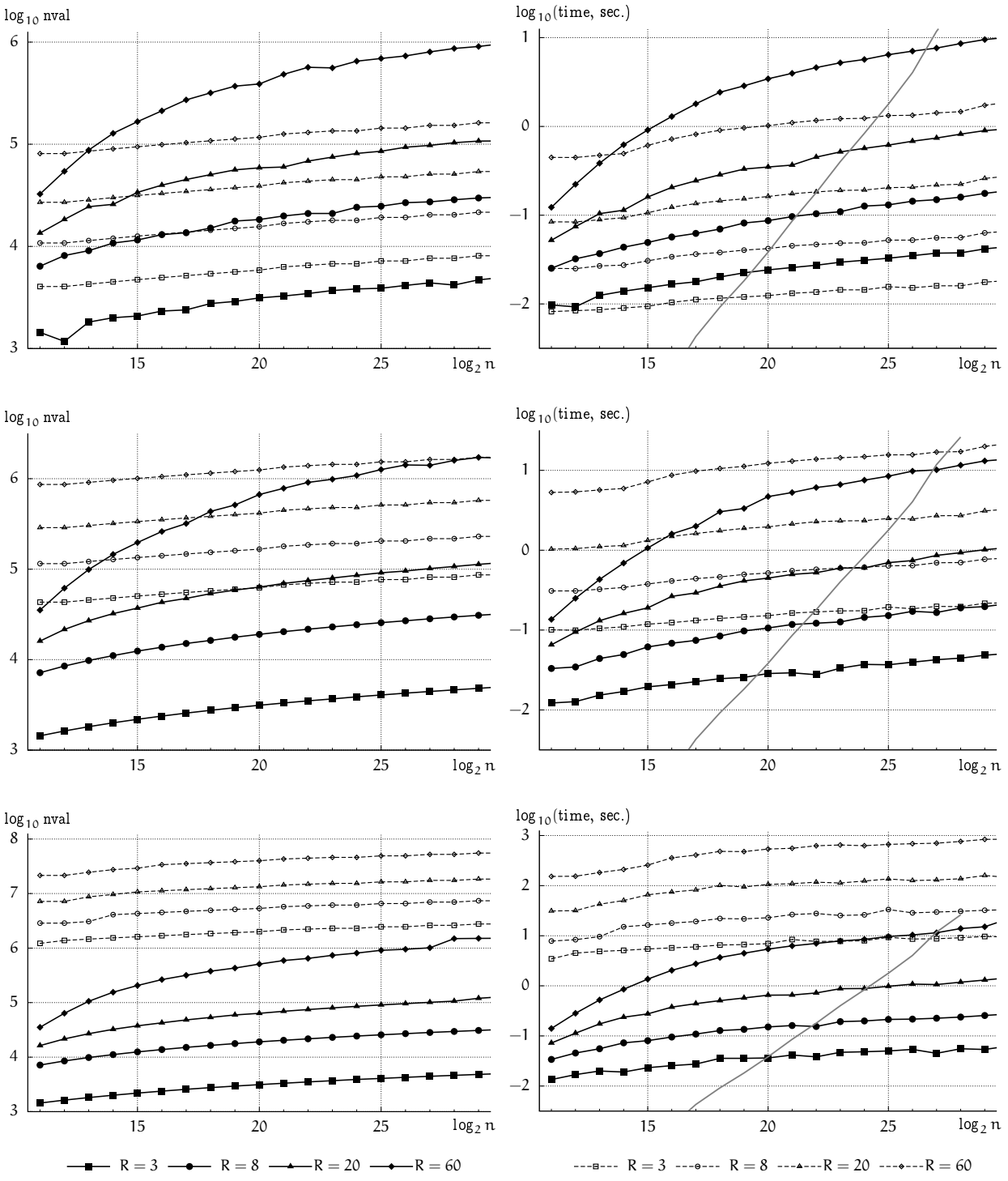

Figure 10. Comparison of QTT–FFT approach with AAFFT algorithm for plane waves example (36), $m = 1$. Left graphs — number of samples, right graphs — computation time. Top — integer frequencies $f_p \in \mathbb{Z}$, middle — real frequencies, $\varepsilon = 10^{-1}$, bottom — real frequencies, $\varepsilon = 10^{-2}$. Solid lines — TT–ACA + QTT–FFT, dashed lines — AAFFT, gray lines — FFTW.

8.2.1. RAlSFA. The randomized heuristic algorithm RAlSFA [63] finds a near-optimal R-term Fourier-sparse representation of a signal of length n with probability δ and quasi-optimality parameter α in time $\mathrm{poly}(r)\log n \log(1/\delta)/\alpha^2$, where the polynomial $\mathrm{poly}(r)$ is empirically found to be quadratic. The complexity of the QTT–FFT transform for such array is $\mathcal{O}(R^3 \log^2 n)$, that is larger than the RAlSFA complexity by $R \log n$ factor. The program code is not available in public domain, and we can compare the running time only using the information about crossover points with FFTW. The crossover point between RAlSFA and FFTW algorithms is reported for $R = 8$, some δ and α and dimensions $m = 1, 2, 3$ as $N \simeq 70000$ for 1D transforms, $N \simeq 900^2$ for 2D transforms and (estimated) as $N \simeq 210^3$ for 3D transforms. The crossover point of QTT–FFT and FFTW for $R = 8$ and $m = 1, 2, 3$ are $N \simeq 2^{20}$ for 1D problems, $N \simeq 2^{19}$ for 2D problems and $N \simeq 2^{18}$ for 3D problems, see Fig. 7. If we consider the runtime of TT–ACA algorithm together with QTT–FFT, the crossover point for 1D transform moves to $N = 2^{21}$, see Fig. 10(top right). For $m = 2, 3$ the runtime of TT–ACA also does not change the crosspoint significantly.

We can conclude that our approach is slower than RAlSFA for 1D problems, has almost the same speed for 2D problems and outperforms it in higher dimensions. The other advantage of the QTT–FFT method is that it does not require any choice of parameters, while the parameter tune in RAlSFA is quite tricky.

8.2.2. sFFT. The nearly optimal Sparse Fourier transform algorithm was recently proposed by MIT group [21, 20]. The complexity is reported to be $\mathcal{O}(R \log n \log(n/R))$, which is smaller than the complexity of QTT–FFT by a factor of R^2 for $n \gg R$. The sFFT algorithm is faster than FFTW for $n = 2^{22}$ and $R \leqslant 2^{17}$. From Fig. 10(top right) we see that "TT–ACA + QTT–FFT" approach is competitive with FFTW for $n = 2^{22}$ and $R \lesssim 2^3$, i.e., our approach is far less efficient for this problem. The program code for sFFT is currently not available in public domain.

8.2.3. AAFFT. The combinatorial sublinear Fourier transform algorithm [24] provides the approximation with high probability and has $\mathcal{O}(R \log^5 n)$ complexity. The deterministic version requires $\mathcal{O}(R^2 \log^4 n)$ operations. The complexity of QTT–FFT is $\mathcal{O}(R^3 \log^2 n)$ and TT–ACA algorithm [54] requires $\mathcal{O}(R^3 \log n)$ operations for each iteration/*sweep*. Therefore, we can expect that AAFFT is faster than our approach for larger R.

The program code of AAFFT algorithm (written in C++) is available in public domain [23] for the case $m = 1$. This allows us to perform the numerical comparison. Note that the measured runtime includes the time of *evaluation* of the value of signal on selected positions (samples), i.e., signal is not precomputed before the experiment. This allows us to use finer grids with $n \leqslant 2^{30}$ but increases the runtime of "sampling" part (TT–ACA and the corresponding part of AAFFT) by the factor of R. The results are reported on Fig. 10(top) in terms of number of samples required from given signal (left column) and runtime (right column). We see that our approach outperforms AAFFT for small n or very small R. For larger n and R the AAFFT algorithm is faster than "TT–ACA + QTT–FFT" scheme for the signal that is the sum of R one-dimensional plain waves (36) with integer frequencies $f_p \in \mathbb{Z}$.

8.3. Comparison for limited bandwidth signals

We have considered the signals (36) with integer frequencies, i.e., the exactly Fourier-sparse case. Such signals are the 'best examples' for the Sparse Fourier transform algo-

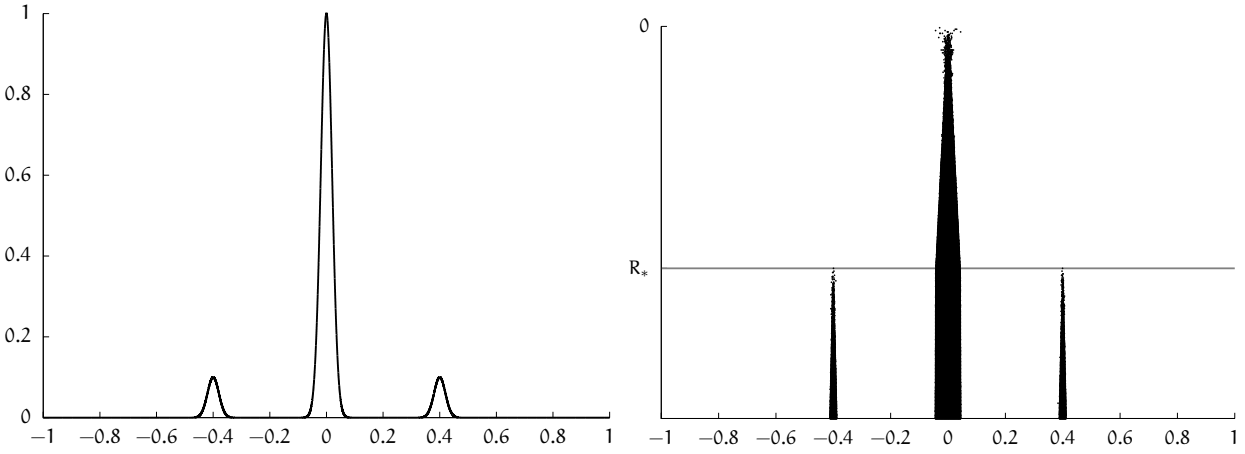

Figure 11. Example of limited bandwidth signal. Left — Fourier image, right — positions of frequencies of R–term Fourier-sparse representation computed by AAFFT w.r.t. R.

rithms. However, since our approach is applicable to a wider class of signals, it is interesting to provide comparison also for the signals which are not exactly Fourier-sparse. For instance, we consider the same plain waves example (36) with arbitrary real frequencies $f_p \in \mathbb{R}^m$. The Fourier images of such signals are not exactly sparse, for example see Fig. 9(left). The QTT ranks of (36) are bounded by the number of terms R, but the QTT ranks of Fourier image (as well as the complexity of QTT–FFT) are larger and depend on selected accuracy level ε, if $f_p \notin \mathbb{Z}^m$.

The number of Fourier terms required to represent the Fourier image of (36) with accuracy ε in sparse format is $o(n)$ but grows with ε. We found that for signals (36) with random frequencies and unit amplitudes the Sparse Fourier representation with about 32R terms provides the accuracy $\varepsilon = 10^{-1}$ and 1024R terms provide the accuracy $\varepsilon = 10^{-2}$. The corresponding number of samples and the runtime are shown on Fig. 10 on middle ($\varepsilon = 10^{-1}$) and bottom ($\varepsilon = 10^{-2}$) lines. We see that the QTT–ranks and runtime of our method grow very slowly with ε, while the runtime of AAFFT is proportional to the number of the Fourier terms required for the representation. Therefore, the sum of plane waves (36) with real frequencies is the example of signal for which our approach allows to compute the approximate Fourier image faster / more accurately than AAFFT Sparse Fourier algorithm.

Another example is the signal consisting of three components with limited bandwidths,

$$\widehat{f}(\xi) = \exp\left(-\frac{\xi^2}{2\sigma^2}\right) + \frac{1}{10}\exp\left(-\frac{(\xi-\xi_*)^2}{2\sigma^2}\right) + \frac{1}{10}\exp\left(-\frac{(\xi+\xi_*)^2}{2\sigma^2}\right), \qquad (38)$$

where $\sigma = 0.02$ and $\xi_* = 0.4$. The Fourier image is shown on Fig. 11(left). This signal models the AM (amplitude modulation) broadcast, where the amplitude of central carrier signal is one order of magnitude larger than the amplitude of sidebands. It is known that the spectrum of the signal belongs to $[-1, 1]$ interval and we can measure the signal at time domain in points $t_k = hk$ on the uniform grid of size $n = 2^d$. Suppose that our goal is to locate the positions of sidebands in frequency space using the small number of samples from the signal and/or the minimum runtime.

Table 4. Comparison of AAFFT and "TT–ACA + QTT–FFT" for limited bandwidth signal (38) of length $n = 2^d$. AAFFT section — R_* is minimal number of Fourier terms that allow to detect the sideband, `time` and `nval` are corresponding runtime (in milliseconds) and number of samples, ε is relative accuracy of the Fourier-sparse representation with R_* terms. TT–ACA + QTT–FFT sections — ε is desired accuracy, R is the maximum effective QTT rank seen in the computation, `time` and `nval` are corresponding runtime (in milliseconds) and number of samples.

	AAFFT $\varepsilon = 10^{-1}$			TT–ACA + QTT–FFT $\varepsilon = 10^{-2}$			TT–ACA + QTT–FFT $\varepsilon = 10^{-10}$		
d	R_*	time	nval	R	time	nval	R	time	nval
10	46	190	$7.36_{10}4$	3.00	5.00	$3.59_{10}3$	7.29	10.9	$4.74_{10}3$
12	183	$1.2_{10}3$	$3.29_{10}5$	2.78	6.72	$2.61_{10}3$	7.30	19.8	$4.92_{10}3$
14	710	$6.4_{10}3$	$1.42_{10}6$	2.58	9.00	$3.60_{10}3$	7.13	28.1	$5.00_{10}3$
16	2800	$8.4_{10}4$	$6.16_{10}6$	2.42	9.81	$3.83_{10}3$	6.78	36.1	$6.96_{10}3$
18	11100	$4.2_{10}5$	$2.67_{10}7$	2.30	11.2	$3.43_{10}3$	6.44	43.8	$5.52_{10}3$
20	—	—	—	2.19	12.9	$1.56_{10}4$	6.15	49.5	$5.88_{10}3$

Since the signal has many Fourier components which are almost zero, the problem can be naturally considered using the Sparse Fourier transform framework, i.e., compute the R–term Fourier-sparse representation of given signal and look at the positions of the computed frequencies. However the number of non-vanishing Fourier components of such signal is $\mathcal{O}(n)$, the ratio of the bandwidths of signal components to the whole interval where the spectrum is considered. This makes this problem particularly difficult for Sparse Fourier transform algorithms and can increase the complexity to $\mathcal{O}(n)$.

We try to locate the sidebands using the AAFFT algorithm with different R. Since the amplitude of carrier signal is larger than the amplitudes of sidebands, for $R < R_*$ all components of the obtained R–term representation belong to the carrier signal. In other words, even the *presence* of sidebands can not be detected until we set $R = R_*$ with certain *critical* number of terms R_*. This is shown on Fig. 11(right). Note that R_* grows with the signal size, since the frequency domain grid size decreases and more Fourier-sparse components belong to the support of each bandwidth limited component.

Our approach represents the signal by the QTT format (4), and large number of non–vanishing components does not mean large QTT–ranks and large complexity of the proposed method. In the experiment we note that QTT–ranks do not grow significantly w.r.t n, and grow very reasonably w.r.t. accuracy parameter ε. This leads to the square-logarithmic complexity of TT–ACA and QTT–FFT, i.e., $\mathcal{O}(R^3 d^2)$, where R is very moderate. The results are shown in Table 4. The sparse format with $R = R_*$ Fourier terms approximates the given signal with relative accuracy in Frobenius norm $\varepsilon \approx 10^{-1}$. Much more accurate QTT approximation is possible and can be obtained in sublinear time.

The considered limitation of the Sparse Fourier transform framework is general and does not (in our opinion) address specifically the AAFFT algorithm. We can conclude that for the considered signal with limited bandwidth our method outperforms the Sparse Fourier transform algorithms, even for small n.

9. Conclusion and future work

We propose algorithms for the approximate Fourier transform of one– and multi-dimensional vectors in the QTT format. The m–dimensional Fourier transform of an $n \times \ldots \times n$ array with $n = 2^d$ has $\mathcal{O}(md^2R^3)$ complexity where R is the maximum QTT rank of input vector, result, and all intermediate vectors of the Cooley-Tukey algorithm. The storage size is bounded by $\mathcal{O}(mdR^2)$. The proposed approach is efficient for vectors with moderate R. For vectors with moderate R and large n and m the proposed algorithm outperforms the $\mathcal{O}(n^m \log n)$ FFT algorithm. The complexity w.r.t. d corresponds to the complexity of the superfast quantum Fourier transform, which is $\mathcal{O}(d^2)$ quantum operations.

Using the 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 a 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 of Gaussian functions with strong cusps and width in the range from $\sigma = 10^{-1}$ to 10^{-5} by 3D convolution. We measure the runtime of algorithm applied to multidimensional functions ($m = 1, 2, 3$) with small number of Fourier coefficients, which can be used to estimate the runtime for vectors with similar m, d and R. Notice that in the case $m > 3$ the computation of high-resolution FFT and convolution transforms is practically infeasible without the use of a certain approximation.

Combining the QTT–FFT algorithm with a method that computes QTT representation from several elements (samples) of a given vector, we compare it with Sparse Fourier transform algorithms. By numerical examples we show that our approach can be competitive with the existing methods for the Fourier-sparse signals with randomly distributed components, especially for higher dimensions. Our approach is especially advantageous for the signals with limited bandwidth, which are not exactly Fourier-sparse. More detailed comparison with modern Sparse Fourier transform algorithms and links with the fast Fourier transform on hyperbolic cross points [9] and QTT–FFT or QTT cross interpolation schemes are postponed for the future work.

The proposed method can be applied in various scientific computing solvers working with multidimensional data in data-sparse tensor formats. The discrete sine/cosine and other trigonometric transforms can be implemented using the corresponding recursion formulae, following the structure of QTT–FFT algorithm. We hope that such algorithms can have an application to image and video processing, directly or with the use of WTT version of the format, following the ideas of [50]. Also, we hope that proposed method can help to construct the classical model of several quantum algorithms based on the QFT, at least for a certain class of input data.

References

[1] *N. Ahmed, T. Natarajan, and K. R. Rao*, Discrete cosine transform, *IEEE Trans. Computers*, C-23(1):90–93, 1974. doi: 10.1109/T-C.1974.223784.

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

[3] *C. Bertoglio and B. N. Khoromskij*, Low-rank quadrature-based tensor approximation of the Galerkin projected Newton/Yukawa kernels, *Computer Phys. Comm.*, 183(4):904–912, 2012. doi: 10.1016/j.cpc.2011.12.016.

[4] *J. W. Cooley and J. W. Tukey*, An algorithm for the machine calculation of complex Fourier series, *Math. Comput.*, 19:297–301, 1965. doi: 10.2307/2003354.

[5] *L. de Lathauwer*, A survey of tensor methods, in IEEE International Symposium on Circuits and Systems, May 2009, pp. 2773–2776. doi: 10.1109/iscas.2009.5118377.

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

[7] *S. Dolgov, B. N. Khoromskij, and I. V. Oseledets*, Fast solution of multi-dimensional parabolic problems in the TT/QTT–format with initial application to the Fokker-Planck equation, Preprint 80, MPI MIS, Leipzig, 2011. `http://www.mis.mpg.de/preprints/2011/preprint2011_80.pdf`.

[8] *A. Ekert and R. Jozsa*, Quantum algorithms: entanglement-enhanced information processing, *Phil. Trans. R. Soc. Lond.*, 356:1769–1782, 1998.

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

[10] *H.-J. Flad, B. N. Khoromskij, D. V. Savostyanov, and E. E. Tyrtyshnikov*, Verification of the cross 3D algorithm on quantum chemistry data, *Rus. J. Numer. Anal. Math. Model.*, 23(4):329–344, 2008. doi: 10.1515/RJNAMM.2008.020.

[11] *C. F. Gauss*, Nachlass: Theoria interpolationis methodo nova tractata, in Werke, vol. 3, Königliche Gesellschaft der Wissenschaften, Göttingem, 1866, pp. 265–330.

[12] *G. Golub and C. Van Loan*, Matrix computations, Johns Hopkins University Press, Baltimore, MD, 1996.

[13] *S. Goreinov, I. Oseledets, D. Savostyanov, E. Tyrtyshnikov, and N. Zamarashkin*, How to find a good submatrix, in Matrix Methods: Theory, Algorithms, Applications, V. Olshevsky and E. Tyrtyshnikov, eds., World Scientific, Hackensack, NY, 2010, pp. 247–256.

[14] *S. A. Goreinov, I. V. Oseledets, and D. V. Savostyanov*, Wedderburn rank reduction and Krylov subspace method for tensor approximation. Part 1: Tucker case, *SIAM J. Sci. Comput.*, 34(1):A1–A27, 2012. doi: 10.1137/100792056.

[15] *S. A. Goreinov and E. E. Tyrtyshnikov*, The maximal-volume concept in approximation by low-rank matrices, *Contemporary Mathematics*, 208:47–51, 2001.

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

[17] *W. Hackbusch*, Tensorisation of vectors and their efficient convolution, *Numerische Mathematik*, 119(3):465–488, 2011. doi: 10.1007/s00211-011-0393-0.

[18] *W. Hackbusch*, Tensor spaces and numerical tensor calculus, Springer–Verlag, Berlin, 2012.

[19] *W. Hackbusch and B. N. Khoromskij*, Low-rank Kronecker-product approximation to multi-dimensional nonlocal operators. I. Separable approximation of multi-variate functions, *Computing*, 76(3-4):177–202, 2006. doi: 10.1007/s00607-005-0144-0.

[20] *H. Hassanieh, P. Indyk, D. Katabi, and E. Price*, Nearly optimal sparse Fourier transform, Preprint arXiv:1201.2501 [cs.DS], 2012.

[21] ——, Simple and practical algorithm for sparse Fourier transform, in Proceedings of 23$^{\text{rd}}$ annual ACM-SIAM simposium on discrete mathematics, SIAM, 2012, pp. 1183–1194.

[22] *M. Hegland*, A self-sorting in-place fast Fourier transform algorithm suitable for vector and parallel processing, *Numerische Mathematik*, 68:507–547, 1994.

[23] *M. A. Iwen*, AAFFT (Ann Arbor Fast Fourier Transform), program code, 2008. `http://www.sourceforge.net/projects/aafftannarborfa/`.

[24] ——, Combinatorial sublinear–time Fourier algorithms, *Found. Comput. Matem.*, 10:303–338, 2010. doi: 10.1007/s10208-009-9057-1.

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

[26] *V. Kazeev, B. N. Khoromskij, and E. E. Tyrtyshnikov*, Multilevel Toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity, Tech. Rep. 36, MPI MIS, Leipzig, 2011. `http://www.mis.mpg.de/publications/preprints/2011/prepr2011-36.html`.

[27] *V. Khoromskaia*, Numerical solution of the Hartree-Fock equation by multilevel tensor-structured methods, PhD thesis, TU Berlin, 2010. `http://opus.kobv.de/tuberlin/volltexte/2011/2948/`.

[28] *V. Khoromskaia, D. Andrae, and B. N. Khoromskij*, Fast and accurate tensor calculation of the Fock operator in a general basis, Preprint 4, MPI MIS, Leipzig, 2012. `www.mis.mpg.de/preprints/2012/preprint2012_4.pdf`.

[29] *V. Khoromskaia, B. N. Khoromskij, and R. Schneider*, QTT representation of the Hartree and exchange operators in electronic structure calculations, *Comput. Meth. Appl. Math*, 11(3):327–341, 2011.

[30] *B. N. Khoromskij*, On tensor approximation of Green iterations for Kohn-Sham equations, *Computing and visualization in science*, 11(4-6):259–271, 2008. doi: 10.1007/s00791-008-0097-x.

[31] ——, Fast and accurate tensor approximation of multivariate convolution with linear scaling in dimension, *J. Comp. Appl. Math.*, 234(11):3122–3139, 2010. doi: 10.1016/j.cam.2010.02.004.

[32] ——, $\mathcal{O}(d \log n)$–Quantics approximation of N–d tensors in high-dimensional numerical modeling, *Constr. Appr.*, 34(2):257–280, 2011. doi: 10.1007/s00365-011-9131-1.

[33] ——, Tensor-structured numerical methods in scientific computing: survey on recent advances, *Chemometr. Intell. Lab. Syst.*, 110(1):1–19, 2012. doi: 10.1016/j.chemolab.2011.09.001.

[34] *B. N. Khoromskij and V. Khoromskaia*, Multigrid accelerated tensor approximation of function related multidimensional arrays, *SIAM J. Sci. Comput.*, 31(4):3002–3026, 2009. doi: 10.1137/080730408.

[35] *B. N. Khoromskij, V. Khoromskaia, S. R. Chinnamsetty, and H.-J. Flad*, Tensor decomposition in electronic structure calculations on 3D Cartesian grids, *J. Comput. Phys.*, 228(16):5749–5762, 2009. doi: 10.1016/j.jcp.2009.04.043.

[36] *B. N. Khoromskij, V. Khoromskaia, and H.-J. Flad.*, Numerical solution of the Hartree–Fock equation in multilevel tensor-structured format, *SIAM J. Sci. Comput.*, 33(1):45–65, 2011. doi: 10.1137/090777372.

[37] *B. N. Khoromskij and I. V. Oseledets*, DMRG+QTT approach to computation of the ground state for the molecular Schrödinger operator, Preprint 69, MPI MIS, Leipzig, 2010. `www.mis.mpg.de/preprints/2010/preprint2010_69.pdf`.

[38] ——, QTT-approximation of elliptic solution operators in high dimensions, *Rus. J. Numer. Anal. Math. Model.*, 26(3):303–322, 2011. doi: 10.1515/rjnamm.2011.017.

[39] *T. G. Kolda and B. W. Bader*, Tensor decompositions and applications, *SIAM Review*, 51(3):455–500, 2009. doi: 10.1137/07070111X.

[40] *B. G. Lee*, A new algorithm to compute the discrete cosine transform, *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-32(6):1243–1245, 1984. doi: 10.1109/TASSP.1984.1164443.

[41] *J. Makhoul*, A fast cosine transform in one and two dimensions, *IEEE Trans. Acoustics, Speech and Signal Processing*, 28(1):27–34, 1980. doi: 10.1109/TASSP.1980.1163351.

[42] *S. A. Martucci*, Symmetric convolution and the discrete sine and cosine transforms, *IEEE Trans. Sig. Proc.*, 42(5):1038–1051, 1994. doi: 10.1109/78.295213.

[43] *I. V. Oseledets*, A new tensor decomposition, *Doklady Math.*, 80(1):495–496, 2009. doi: 10.1134/S1064562409040115.

[44] ——, Approximation of $2^d \times 2^d$ matrices using tensor decomposition, *SIAM J. Matrix Anal. Appl.*, 31(4):2130–2145, 2010. doi: 10.1137/090757861.

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

[46] ——, Tensor-train decomposition, *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011. doi: 10.1137/090752286.

[47] *I. V. Oseledets, D. V. Savostyanov, and E. E. Tyrtyshnikov*, Cross approximation in tensor electron density computations, *Numer. Linear Algebra Appl.*, 17(6):935–952, 2010. doi: 10.1002/nla.682.

[48] *I. V. Oseledets and E. E. Tyrtyshnikov*, Breaking the curse of dimensionality, or how to use SVD in many dimensions, *SIAM J. Sci. Comput.*, 31(5):3744–3759, 2009. doi: 10.1137/090748330.

[49] ——, TT-cross approximation for multidimensional arrays, *Linear Algebra Appl.*, 432(1):70–88, 2010. doi: 10.1016/j.laa.2009.07.024.

[50] ——, Algebraic wavelet transform via quantics tensor train decomposition, *SIAM J. Sci. Comput.*, 33(3):1315–1328, 2011. doi: 10.1137/100811647.

[51] *T. Rohwedder, S. Holtz, and R. Schneider*, The alternation least squares scheme for tensor optimisation in the TT-format, Preprint DFG-Schwerpunktprogramm 1234 71, 2010.

[52] *D. V. Savostyanov*, Fast revealing of mode ranks of tensor in canonical format, *Numer. Math. Theor. Meth. Appl.*, 2(4):439–444, 2009. doi: 10.4208/nmtma.2009.m9006s.

[53] ——, QTT-rank-one vectors with QTT-rank-one and full-rank Fourier images, *Linear Alg. Appl.*, 436(9):3215–3224, 2012. doi: 10.1016/j.laa.2011.11.008.

[54] *D. V. Savostyanov and I. V. Oseledets*, Fast adaptive interpolation of multi-dimensional arrays in tensor train format, in Proceedings of nDS-2011 Conference, IEEE, 2011. doi: 10.1109/nDS.2011.6076873.

[55] *D. V. Savostyanov and E. E. Tyrtyshnikov*, Approximate multiplication of tensor matrices based on the individual filtering of factors, *J. Comp. Math. Math. Phys.*, 49(10):1662–1677, 2009. doi: 10.1134/s0965542509100029.

[56] *D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin*, Fast truncation of mode ranks for bilinear tensor operations, *Numer. Linear Algebra Appl.*, 19(1):103–111, 2012. doi: 10.1002/nla.765.

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

[58] *G. Strang*, The discrete cosine transform, *SIAM Review*, 41(1):135–147, 1999.

[59] *Z. Wang*, Fast algorithms for the discrete w transform and for the discrete fourier transform, *IEEE Trans. Acoustics, Speech and Signal Processing*, 32(4):803–816, 1984. doi: 10.1109/TASSP.1984.1164399.

[60] *S. R. White*, Density matrix formulation for quantum renormalization groups, *Physical Review Letters*, 69(19):2863–2866, 1992. doi: 10.1103/PhysRevLett.69.2863.

[61] *P. Yip and K. R. Rao*, On the computation and the effectiveness of discrete sine transform, *Computers and Electrical Engineering*, 7(1):45–55, 1980. doi: 10.1016/0045-7906(80)90018-X.

[62] ——, Fast decimation-in-time algorithms for a family of discrete sine and cosine transforms, *Circuits Systems Signal Process*, 3(4):387–408, 1984. doi: 10.1007/BF01599167.

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

A. Discrete cosine and sine transforms in QTT format

Discrete cosine transform (DCT), proposed in 1974 by N. Ahmed [1], is a power tool for signal and image processing [2]. It can be computed using real-to-complex Fourier transform [1, 41] or with purely real-valued arithmetics using the radix-2 recurrence [40, 62]. More details can be found in a nice review by G. Strang [58]. Discrete sine transform (DST) was later proposed by K. R. Rao [61]. The systematic factorisation method of Z. Wang [59] 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 [42]. We consider the even transforms

$$y(j) = \sum_{k=0}^{n-1} x(k) \cos \frac{\pi(j+j_0)(k+k_0)}{n}, \qquad z(j) = \sum_{k=0}^{n-1} x(k) \sin \frac{\pi(j+j_0)(k+k_0)}{n}, \tag{39}$$

where j_0 and k_0 define the particular transform and $j, k = 0, \ldots, n-1$. For $n = 2^d$ we propose algorithms to compute (39) for a signal given in QTT format (4), similarly to QTT–FFT Algorithm 2. This can be done using double-size Fourier transform and complex-to-real transformation, as explained by Theorem 4 and Example 2. In the following we will show how (39) can be computed in real-valued arithmetics, following the radix-2 scheme from Section 3.

Before we start, it should be mentioned that (39) is slightly different 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 $j, k = 0, \ldots, n$ for DCT-1 and $j, k = 1, \ldots, n-1$ for DST-1. In the QTT format, these corrections require the rank-one update of the result and can be implemented easily. In the following we silently omit these details and explain the general approach to the implementation of different radix-2 schemes in QTT format.

For $n = 2^d$ we use index transform (9) and write

$$\begin{aligned}\cos\frac{\pi(j+j_0)(k+k_0)}{2^d} &= \cos\frac{\pi(2J+j_0+j_1)(K+k_0+k_d 2^{d-1})}{2^d} \\ &= \tilde{c}c_J Cc'_K - \tilde{c}s_J Sc'_K - \tilde{s}c_J Cs'_K + \tilde{s}s_J Ss'_K - \tilde{c}c_J Ss'_K - \tilde{c}s_J Cs'_K - \tilde{s}c_J Sc'_K - \tilde{s}s_J Cc'_K,\end{aligned}$$

where

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

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 (22), 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 $y = C_d x$ and sine $z = S_d x$, defined as follows

$$y(j) = \sum_{k=0}^{n-1} x(k)\cos\frac{\pi jk}{n}, \qquad z(j) = \sum_{k=0}^{n-1} x(k)\sin\frac{\pi jk}{n}, \qquad n = 2^d. \tag{40}$$

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

$$\begin{aligned} P_d C_d x &= \begin{bmatrix} C_{d-1}x_{top} & + & J_1 C_{d-1} x_{bot} \\ C_{d-1}(c \odot x_{top}) - S_{d-1}(s \odot x_{top}) & - & J_1 C_{d-1}(s \odot x_{bot}) - J_1 S_{d-1}(c \odot x_{bot}) \end{bmatrix}, \\ P_d S_d x &= \begin{bmatrix} S_{d-1}x_{top} & + & J_1 S_{d-1} x_{bot} \\ C_{d-1}(s \odot x_{top}) + S_{d-1}(c \odot x_{top}) & + & J_1 C_{d-1}(c \odot x_{bot}) - J_1 S_{d-1}(s \odot x_{bot}) \end{bmatrix}, \end{aligned} \tag{41}$$

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

$$J_1 = \operatorname{diag}(\underbrace{+1, -1, +1, -1, \ldots + 1 - 1}_{2^{d-1} \text{ elements}}). \tag{42}$$

To get rid of P_d, we multiply both sides by the bit-reverse permutation R_{p-1} and since

$$R_{p-1}P_d = R_d, \qquad R_{d-1}(J_1 \times I) = (J_{d-1} \times I)R_{d-1},$$

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

we finally come to the following recurrence

$$\begin{aligned}
\tilde{C}_d x &= \begin{bmatrix} \tilde{C}_{d-1} x_{top} & + & J_{d-1}\tilde{C}_{d-1} x_{bot} \\ \tilde{C}_{d-1}(c \odot x_{top}) - \tilde{S}_{d-1}(s \odot x_{top}) & - & J_{d-1}\tilde{C}_{d-1}(s \odot x_{bot}) - J_{d-1}\tilde{S}_{d-1}(c \odot x_{bot}) \end{bmatrix}, \\
\tilde{S}_d x &= \begin{bmatrix} \tilde{S}_{d-1} x_{top} & + & J_{d-1}\tilde{S}_{d-1} x_{bot} \\ \tilde{C}_{d-1}(s \odot x_{top}) + \tilde{S}_{d-1}(c \odot x_{top}) & + & J_{d-1}\tilde{C}_{d-1}(c \odot x_{bot}) - J_{d-1}\tilde{S}_{d-1}(s \odot x_{bot}) \end{bmatrix},
\end{aligned} \tag{43}$$

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 4. 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 (41),(42). The QTT format for the multiples reads by (22)

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

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

Second, we go up through the tensor train and implement the recurrence (43). To explain Line 10, it is enough to mention that for all half-sized vectors in (43) the QTT representation with the same "lower-bit" structure $Q^{(1)}_{j_1} Q^{(2)}_{j_2} \dots Q^{(D-1)}_{j_{D-1}}$ exists at this moment. Therefore, for instance

$$\begin{aligned}
Q^{(1)}_{j_1} Q^{(2)}_{j_2} \dots Q^{(D-1)}_{j_{D-1}} Y^{(D)}_{j_D} U^{(D+1)}_0 \quad &\text{represents} \quad (\tilde{C}_D x_{top})(\overline{j_1 \dots j_D}), \\
Q^{(1)}_{j_1} Q^{(2)}_{j_2} \dots Q^{(D-1)}_{j_{D-1}} Z^{(D)}_{j_D} V^{(D+1)}_1 \quad &\text{represents} \quad (\tilde{S}_D(c \odot x_{bot}))(\overline{j_1 \dots j_D})
\end{aligned}$$

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 (42). Therefore, the recurrence (43) is implemented by proper combination of matrices $Y^{(D)}_{j_D}$ $Z^D_{j_D}$, that define the coefficients for cosine/sine transforms and $U^{(D+1)}_{j_{D+1}}$, $V^{(D+1)}_{j_{D+1}}$ and $W^{(D+1)}_{j_{D+1}}$, that represent bottom and top parts of vector x scaled by identity, cosine and sine multiplier, respectively.

Algorithm 4: QTT sine/cosine 1D, approximation

Input: $x(k) = X^{(1)}_{k_1} X^{(2)}_{k_2} \dots X^{(d)}_{k_d}$, accuracy ε 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 (40)

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

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

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: Use TT–SVD to recompress $T_D(k)$ using accuracy ε 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 (43) by

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

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 ε or rank R_D criterion.

12: **end for**

13: bit-reverse the output