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

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

I.V. Oseledets, E.E. Tyrtyshnikov

Algebraic wavelet transform via quantics tensor train decomposition

Preprint 2010-03

August 12, 2010

Moscow 2010

Algebraic wavelet transform via quantics tensor train decomposition

I.V. Oseledets, E.E. Tyrtyshnikov

Institute of Numerical Mathematics, Russian Academy of Sciences,
Russia, 119333 Moscow, Gubkina 8
ivan.oseledets@gmail.com
tee@bach.inm.ras.ru

August 12, 2010

[$]This work was supported by RFBR grants 08-01-00115, 09-01-00565 and RFBR/DFG grant 09-01-91332, Russian Goverment Contracts P1178, P940, P1112, Russian President Grant MK-127.2009.1

Abstract

In this paper we show that recently introduced QTT (quantics tensor train) decomposition can be considered as an algebraic wavelet transform with adaptively determined filters. The main algorithm for obtaining QTT decomposition can be reformulated as a method seeking for "good subspaces" or "good bases" and considered as a parameterized transformation of initial tensor into a sparse tensor. This interpretation allows us to introduce a modification of the TT-SVD algorithm to make it work in cases where original algorithm does not work, it results in the new wavelet-like transforms called WTT (wavelet tensor train transform). Properties of WTT transforms are studied numerically, a theoretical conjecture on the number of vanishing moments is proposed. It is shown that WTT transforms are orthogonal by construction, and the efficiency of WTT is compared with and often outperforms Daubechies wavelet transforms on certain classes of function-related vectors and matrices.

1. Introduction

Discrete wavelet transform is a well-known approach to efficient representation of discrete analogues of functions or signals. For a function $f(x)$ of one argument $x \in [a.b]$ the procedure looks as follows. Given a grid with the nodes $x_k \in [a, b]$, $k = 1, \ldots, n$, consider a vector v of the values of f:

$$v_k = f(x_k), \quad k = 1, \ldots, n.$$

For simplicity, suppose the grid is uniform. For the vector v to be stored, n memory cells are required. If v possesses no additional structure, nothing can be done. However, if f has some smoothness properties, then one can expect that the number of parameters to store v can be smaller. To compress v using a discrete wavelet transform (DWT) [1], a suitable wavelet transform matrix W is constructed and a new representation of the same vector

$$w = Wv.$$

is computed. As a simple example, *Haar wavelet transform* can be considered: suppose $n = 2^d$, then one level of the Haar transform reads

$$\begin{aligned} w_{2k} = (Wv)_{2k} &= \frac{1}{\sqrt{2}}(v_{2k} + v_{2k-1}), \\ w_{2k+1} = (Wv)_{2k} &= \frac{1}{\sqrt{2}}(v_{2k} - v_{2k-1}), \quad k = 1, \ldots, 2^{d-1}, \end{aligned}$$

and then the same transform is applied recursively to vector $w' \in \mathbb{R}^{2^{d-1}}$, where $w'_k = w_{2k}$. There are many other DWT's which are more effficient, for example Daubechies wavelet transforms. From algebraic point of view, wavelet transforms can be characterized as follows:

(a) They are linear transforms

$$w = Wv,$$

where Wv and $W^{-1}w$ can be computed in $\mathcal{O}(n)$ operations

(b) They transform a certain set of "smooth" non-sparse vectors into pseudo-sparse vectors, i.e. many elements of the transformed vector are very small.

(c) Locality property: if v is changed in few places, then $w = Wv$ is also changed only just few places.

Usually, the sparsity requirement is implemented as *zero moments condition*: there exists a number p such that for all $k = 0, 1, \ldots, p-1$, the vectors corresponding to functions $f(x) = x^k$ are transformed into sparse vectors *exactly*. For example, the Haar wavelet transform [1] for the vector v with components $v_i = 1$ makes all elements

except one equal to zero. This means that this transform has *one zero moment*. The Daubechies transform family [2] has transforms with higher number of zero moments, and that leads to higher compression. However, polynomial zero moments may not be the best for a particular function. For example, for a very smooth function $f = e^{\lambda x}$ these transforms lead only to pseudo-sparsity of the transformed vector, but no exact zeros. Are there any other (and preferably simple) ways to design an *algebraic wavelet transform*, i.e. determine the matrix W adaptively for a given function?

The goal of this paper is to show connections between recently introduced *quantics tensor train* (QTT) *representation of vectors* [3, 4, 5] and wavelet transforms. Note that QTT is a special form (with quantized mode sizes) of tensor-train (TT) decomposition [6], for basic related facts and a generalization of skeleton (dyadic) decomposition from matrices to tensors see [7] and also [8] where it is proved that any scheme of recursive reduction of dimensionality of tensors [9] results in some TT decomposition. The very concept of artificially increasing the number of modes (in other words, inserting fictitious modes) in tensors was earlier considered in [10]. In this paper, it will be shown that the QTT (*quantics tensor train*) format with certain modifications can be considered as an *algebraic wavelet transform* which we further call WTT (*wavelet tensor train*).

The idea behind QTT is simple. For a given vector v of size $n = 2^d$, we consider it as a d-dimensional array, or tensor:

$$\mathbf{V}(i_1, \ldots, i_d) = v(i),$$

where $i_k = 0$ or 1 are just binary digits of the integer i, more precisely,

$$i/2^d = i_1/2 + i_2/2^2 + \ldots + i_d/2^d.$$

Then, a *tensor train* (TT) *approximation* [6, 7] is computed for this d-dimensional array. The TT decomposition is a special representation of d-dimensional tensors:

$$V(i_1, \ldots, i_d) \approx \sum_{\alpha_1, \alpha_2, \ldots, \alpha_{d-1}} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_d(\alpha_{d-1}, i_d), \tag{1}$$

where each α_k runs from 1 to r_k. The quantities r_k are called compression ranks, or simply TT-ranks. Arrays G_k are called *carriages* (that suits the name of *tensor train*) or *cores* of the TT decomposition. This decomposition was introduced and studied in [6, 7]. The main benefit of this decomposition is that it can be computed (it means that TT-ranks and carriages are found) through d sequential singular value decompositions. The corresponding algorithm called TT-SVD is presented in [7] and will be recollected in Section 2. The TT-SVD algorithm is stable and fast, which makes such decomposition very appealing. Moreover, if TT-ranks $r_k \leqslant r$ are small, then the total memory required to store the TT decomposition is $\mathcal{O}(dr^2)$, and the dependence on the dimensionality d is linear!

The cases where ranks are small were already considered in [3, 11]. It was proved that for the exponential function all ranks are equal to 1, and for polynomials of degree

p ranks are bounded by $p+1$, so if a function can be approximated by a sum of exponents and polynomials, then ranks are bounded independent of the grid size n, and the memory to store the QTT decomposition is logarithmic in n.

The goal of this paper is to show that decomposition (1) can be considered as a wavelet transform with adaptively determined filter banks, and show what is the difference between the two approaches. Indeed, they can be incorporated into one common framework, which can be considered as *algebraic wavelet transform*. We will call it *wavelet tensor train* or WTT, and show that there are cases when direct application of QTT does not work, but WTT works perfectly. This is especially clear for two-dimensional functions.

2. Basic facts about TT decomposition and TT-SVD algorithm

Basic objects used in this paper are multidimensional arrays, called tensors. They will be denoted by boldface letters, e.g. $\mathbf{A}$. Elements of a $n_1 \times n_2 \ldots \times n_d$ tensor $\mathbf{A}$ are denoted as $A(i_1, \ldots, i_d)$, and n_k are referred to as *mode sizes*. Since tensors belong to a linear space, standard linear operations (addition, multiplication by a number) are naturally defined. The Frobenius norm of a tensor is defined as

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i_1,\ldots,i_d} A(i_1,\ldots,i_d)^2}.$$

An important operation is *tensor-by-matrix multiplication* over the mode k (also called mode-product or contracted product). It is defined as

$$\mathbf{B} = \mathbf{A} \times_k U \quad \rightarrow \quad B(i_1,\ldots,i'_k,\ldots,i_d) = \sum_{i_k=1}^{n_k} A(i_1,\ldots,i_d)U(i_k,i'_k).$$

Tensors can be transformed into matrices in various ways. We adopt the following notation: given a tensor $\mathbf{A} = A(i_1, i_2, \ldots, i_d)$, denote by

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

its *k-th unfolding matrix*, the first k indices enumerating its rows and the last $d-k$ counting its columns.

A detailed review of tensors and their application can be found in [12, 13].

A multidimensional array (tensor) $\mathbf{A} = A(i_1, \ldots, i_d)$, with indices $1 \leqslant i_k \leqslant n_k$ is said to be in the TT (tensor train) format, if it is represented as

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

where α_k varies from 1 to r_k, and G_k are called *carriages* or *cores* of the TT-decompositions, and r_k are called compression ranks, or simply TT-ranks. G_1 and

G_d are matrices. To make this decomposition more symmetric, it is natural to consider its *extended form*:

$$A(i_1, i_2, \ldots, i_d) = \sum_{\alpha_0, \alpha_1, \ldots, \alpha_{d-1}, \alpha_d} G_1(\alpha_0, i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_d(\alpha_{d-1}, i_d, \alpha_d), \qquad (3)$$

where two dummy indices α_0 and α_d are equal to one. This form simplifies the description of algorithms, always with $r_0 = r_d = 1$.

Also, the k-th rank of the TT decomposition of $\mathbf{A}$ will be denoted by $r_k(\mathbf{A})$. The TT-ranks are bounded from below by ranks of the unfolding matrices A_k,

$$r_k \geqslant \operatorname{rank} A_k, \quad k = 1, \ldots, d.$$

The unfolding procedure is also called *matrization* of a tensor [12]. It should be emphasized, however, that the unfolding matrices considered in [12] arise in the Tucker decomposition, and they differ from those of the TT decomposition. Only the first and the last are the same; thus TT-ranks do not coincide with the Tucker ranks in general. It is always guaranteed that a TT decomposition with the ranks $r_k = \operatorname{rank} A_k$ exists [6], and, more than that, can be computed by d singular value decompositions (SVD) of auxiliary matrices. Moreover, this procedure is stable: if the TT unfolding matrices are of approximate rank r_k:

$$A_k = R_k + E_k, \quad \text{where} \quad \operatorname{rank} R_k = r_k \quad \text{and} \quad \|E_k\|_F = \varepsilon_k.$$

Then the TT approximation $\mathbf{B}$ computed by a sequence of SVD decompositions satisfies

$$\|\mathbf{A} - \mathbf{B}\|_F \leqslant \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}.$$

This confirms the stability of the approximation procedure (called TT-SVD later on, since it can be considered as a generalization to tensors of the SVD algorithm for matrices) [7]. An important case is when the input tensor to be compressed is given in the TT format with possibly large TT-ranks while the output tensor is sought in the same TT format with smaller ranks. An efficient and robust algorithm for such cases is proposed in [6, 14] and can be called *TT rounding algorithm*.

If all ranks are equal to r and all mode dimensions are equal to n, then the TT format requires $\mathcal{O}(dnr^2)$ memory cells. Hence, the storage is linear in d and quadratic in r. To represent a d-dimensional array, a standard format that was in the focus of several decades of research is the *canonical format*:

$$A(i_1, \ldots, i_d) \approx \sum_{\alpha=1}^{R} U_1(i_1, \alpha) \ldots U_d(i_d, \alpha). \qquad (4)$$

The minimal possible R in exact representations in the canonical format is denoted by $\operatorname{rank}(\mathbf{A})$, it gives the estimate for the TT-ranks of $\mathbf{A}$ from above:

$$\operatorname{rank}(\mathbf{A}) \geqslant r_k, \quad k = 1, \ldots, d.$$

The canonical format requires $\mathcal{O}(dnR)$ memory cells, however, even when R is sufficiently small, it suffers from certain drawbacks. Despite recent progress, there are no robust algorithms to compute canonical decomposition numerically, and the approximation by a canonical tensor with a fixed rank can be ill-posed [15]. In contrast, computation of the best TT approximation with fixed TT-ranks is a well-posed problem, and quasioptimal approximation can be computed by means of the TT-SVD algorithm which uses standard LAPACK procedures [7]. By this reason, the TT format is preferable in numerical computations. The TT-SVD algorithm plays a crucial role, so we recall it in Algorithm 1. A MATLAB code for this algorithm is a part of the TT-Toolbox.

Algorithm 1: TT-SVD

Input: d-dimensional tensor $\mathbf{A}$, prescribed accuracy ε.

Output: Carriages (cores) $G_1, \ldots, G_d$ of the TT approximation $\mathbf{B}$ to $\mathbf{A}$ in the TT format with smallest possibles compression ranks $\widehat{r}_k$ such that

$$\|\mathbf{A} - \mathbf{B}\|_F \leqslant \varepsilon \|\mathbf{A}\|_F,$$

{Initialization}
Compute truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}} \|\mathbf{A}\|_F$.
Temporary tensor: $\mathbf{C} = \mathbf{A}$.
$N = \mathrm{numel}(A)$, $r_0 = 1$.
for $k = 1$ to $d-1$ **do**
  $C := \mathrm{reshape}(C, [r_{k-1} n_k, \frac{N}{r_{k-1} n_k}])$.
  Compute δ-truncated SVD: $C = USV + E, \|E\|_F \leqslant \delta, \quad r_k = \mathrm{rank}_\delta(C)$.
  New core: $G_k := \mathrm{reshape}(U, [r_{k-1}, n_k, r_k])$.
  $C := SV^\top$.
  $N := \frac{N r_k}{n_k r_{k-1}}$.
end for
$G_d = C$.
Return tensor $\mathbf{B}$ in TT-format with carriages (cores) $G_1, \ldots, G_d$.

The complexity of TT-SVD can be estimated as $\mathcal{O}(Nr^2)$, where N is the total number of entries in the tensor, and r is the maximal rank that appeared during computation.

3. TT decomposition as a subspace approach

3.1. Another view of the TT-SVD algorithm

Algorithm 1 is a very simple algorithm to compute a TT decomposition of a given tensor. Let us focus on the QTT decomposition, which means merely that $n_k = 2$ and the tensor $\mathbf{V}$ is a reshaped version of the vector $v_k = f(x_k)$. Then, the TT-SVD algorithm can be interpreted in the following way. In the first step, the given vector v

is reshaped into a $2 \times n/2$ matrix C, and its SVD is computed:

$$C = USV,$$

where U is 2×2, S is 2×2 and V is $2 \times n/2$. Then,

$$C' = SV = U^\top C,$$

since U is an orthogonal matrix. The norms (Euclidean lengths) of rows of C' are equal to the singular values of C. If there is a small singular value, then the corresponding row of C' has a small norm and can be neglected and omitted in further compression steps. That is exactly what is happening in the TT-SVD algorithm: in the matrix C', rows of small norm are omitted, the remaining part is reshaped again and compressed. For the QTT representation, the matrix C at the first step is of the form

$$C = \begin{pmatrix} v_1 & v_3 & \cdots & v_{2k-1} & \cdots & v_{2n-1} \\ v_2 & v_4 & \cdots & v_{2k} & \cdots & v_{2n} \end{pmatrix},$$

and multiplication by $U^\top$ from the left is a "local operation":

$$U^\top C = \begin{pmatrix} u_{11}v_1 + u_{21}v_2 & u_{11}v_3 + u_{21}v_4 & \cdots & u_{11}v_{2k-1} + u_{21}v_{2k} & \cdots & u_{11}v_{2n-1} + u_{21}v_{2n} \\ u_{12}v_1 + u_{22}v_2 & u_{12}v_3 + u_{22}v_4 & \cdots & u_{12}v_{2k-1} + u_{22}v_{2k} & \cdots & u_{12}v_{2n-1} + u_{22}v_{2n} \end{pmatrix},$$

i.e. *one and the same linear filter* U *is applied* to each subvector of length 2 . Furthermore, the second row has smaller norm, and if, for example, $f(x) = e^{\lambda x}$, it will be zero exactly, so the new matrix C that will be compressed further on will be constructed only from the first row of $U^\top C$. Of course, the multiplication by $U^\top$ means a change of basis, the columns of U being the new basis to represent the subvectors. This is exactly the way how a wavelet transform works: it exploits a specifically designed linear filter (basis) U to represent each subvector. For example, if we select

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix},$$

then this will be one level of the Haar wavelet transform. Note that if we take $f(x) = 1$ then the matrix U and the transform computed during the TT-SVD algorithm will coincide with the Haar wavelet transform. There are two principal differences between wavelets and tensor trains:

(1) For a wavelet transform, the matrix U is fixed once-and-for-all, whereas for QTT it is computed adaptively, and

(2) For wavelet transforms the ranks r (i.e. the number of rows to be kept in the matrix C, i.e. in the further compression steps) is fixed, and the omitted rows may not be of small norm, instead they are usually *pseudo-sparse*.

The second difference can be turned into advantage yielding a *wavelet tensor train transform* (WTT). Suppose that we have computed the matrix U via SVD and let

$$C' = U^\top C.$$

The singular values may not be as small as required, but the singular vectors can be *pseudo-sparse*, so the procedure is now as follows. We choose some parameter r, leave r rows for further compression while the remaining rows are approximated by sparse rows and stored as a sparse vector. In the same fashion, a matrix U_2 is computed from the first r rows of C', and so on, and these matrices $(U_1 = U, U_2, \dots)$ define a sequence of linear filters. In the end, they define a linear transformation

$$w = W(U_1, \dots, U_d)v,$$

of the initial vectors. If the number r, now a parameter, is independent of n, then $\mathcal{O}(dr^2) = \mathcal{O}(\log n)$ memory cells are required to store the transform, and vector w is sparse or pseudo-sparse (in many interesting cases). How to select $r_1, \dots, r_{d-1}$ is a different story, which requires a separate study.

3.2. Computation of filters and application of the transform

So long as the rank selection strategy is fixed, the algorithm for computing the filter matrices $U_1, \dots, U_d$ along with the transform look as follows. The computation of linear filters is presented in Algorithm 2, and the algorithm implementing the transform is given in Algorithm 3. Since the algorithms do not rely on the fact that arrays are of size $2 \times 2 \times \dots \times 2$ and can be applied to a general d-dimensional array, we present both algorithms for general tensors with mode sizes n_k, $k = 1, \dots, d$.

Algorithm 2: WTT-filter

Input: Tensor v of size $n_1 \times n_2 \times \dots \times n_d$, rank parameters $r_1, \dots, r_d$.
Output: Linear filters $U_1, \dots, U_{d-1}$ defining the WTT transform.

1: {Initialization}
2: $N := n_1 \dots n_d$, a temporary matrix $A := \text{reshape}(v, [n_1, N/n_1])$, $r_0 = 1$.
3: **for** $k = 1$ to $d - 1$ **do**
4: $[m, p] := \text{size}(A)$, the rank estimate: $r := \min(m, p, r_k)$.
5: Compute SVD: $A = USV$, and

$$U_k := U.$$

6: Number of elements in the further compressed part: $N := \frac{Nr}{n_{k-1} r_{k-1}}$.
7: $A := U^\top A$, $A := A(1 : r, :)$, $A := \text{reshape}(A, [n_k r, \frac{N}{n_k r}])$.
8: $r_k := r$.
9: **end for**

Algorithm 3: WTT-application

Input: Vector v of size $n = \prod_{k=1}^{d} n_k$, linear filters $U_1, \ldots, U_{d-1}$ defining the WTT transform.

Output: Transformed vector w.

1: {Initialization}
2: $N := n$, $w = \text{reshape}(v, [n_1, N/n_1])$, $r_0 = 1$.
3: $w = U_k^\top w$.
4: {Recursion}
5: $\widehat{w} := w(1 : r_k, :)$, reshape it into a vector and apply WTT with filters $U_{k+1}, \ldots, U_d$ to $\widehat{w}$:

$$\widehat{w} := \text{WTT}(\widehat{w}, U_{k+1}, \ldots, U_d)$$

6: Put $\widehat{w}$ on the place of $w(1 : r_k, :)$.

3.3. Transform properties

Algorithm 3 describes a *linear transform*. It is easy to see that its complexity is $\mathcal{O}(nr^2)$, where $r = \max r_k$. Also, the transform is orthogonal by construction, and the following lemma holds true.

Lemma 1. *The transform with matrices U_k obtained from Algorithm 2 and implemented by Algorithm 3 is orthogonal.*

Proof. It is sufficient to show that the norm of the vector is conserved after the transform. Since the matrices U_k come from the SVD, they are orthogonal. Therefore, at each step of WTT transform the norm of the vector w after multiplication by $U_k^\top$ does not change, and in the next step some subtensor is transformed and its norm is not changing as well, thus the whole transform is orthogonal.

3.4. Rank selection strategy

An important problem is the selection strategy for the ranks $r_1, \ldots, r_{d-1}$. Two options can be used:

(a) ε-strategy: at each step the singular values are truncated at threshold set by ε. This is equivalent to QTT decomposition of the vector.

(b) Limited rank truncation: maximal possible rank r is preset and only r rows are kept. This can be non-optimal in certain cases, but it is equivalent to prescribing the "number of zero moments" in standard wavelet transforms.

More complicated rank selection strategies may include a combination of ε-strategy and the limited rank requirement, or based on the analysis of decay of singular values, or on the sparsity structure of singular vectors, or on both. This is a separate topic, and we leave it for future research.

4. Application to oscillating functions

In one dimension, the adaptive wavelet transform is not always a favourite for compression. In order to store linear filters $U_1, \ldots, U_d$, one needs dr^2 elements. If $d = 10$ and $r = 4$, then it amounts to 160 elements, compared to $2^d = 1024$ elements in the original vector, and a compression using standard wavelet approaches where filters are considered *precomputed* is often better. For example, for a simple sum of exponents

$$f(x) = e^x + e^{1.5x} - 2e^{-2x} + 7e^{3x}$$

defined on the interval $[-1, 1]$, ranks are exactly equal to 4 and the transformed vector has only 4 nonzero elements. If the Daubechies transform with 4 vanishing moments is used, then thresholding at the level of 10^{-8} leaves only 184 non-zero elements and this number is independent from d, whereas the memory required to store adaptive linear filters grows linearly in d. However, there are cases where WTT is remarkably better. Consider two examples.

Example 1. *Consider a sine function*

$$f(x) = \sin(100x), \; x \in [0, 1].$$

For Example 1 it is known, that all TT-ranks are exactly equal to 2, whereas Daubechies transforms leave large amount of nonzero elements (see Table 1). In this experiment d is set to 20.

ε	nnz(WTT)	Memory(U_k)	nnz(D4)	nnz(D8)
10^{-4}	2	152	3338	880
10^{-6}	2	152	19696	2010
10^{-8}	2	152	117575	6570
10^{-10}	2	152	845869	15703
10^{-12}	2	152	1046647	49761

Table 1. Sine function, $n = 2^d$, $d = 20$

Example 2. *This example is similar to Example 1, we test Bessel function with high frequency:*

$$f(x) = J_1(100x), \; x \in [0, 1].$$

In this case, QTT decomposition (and thus, WTT decomposition) are only approximate, so the memory for filters $U_1, \ldots, U_d$ increases with increased accuracy ε.

The results for Example 2 are presented in Table 2

ε	nnz(WTT)	Memory(U_k)	nnz(D4)	nnz(D8)
10^{-4}	4	640	1777	543
10^{-6}	4	946	10641	1712
10^{-8}	4	1254	66280	4005
10^{-10}	4	1636	448011	12797
10^{-12}	4	2016	1039004	31327

Table 2. Bessel function, $n = 2^d$, $d = 20$

5. Using WTT to create new wavelet transforms

5.1. WTT for polynomial functions

We can also use WTT to create new wavelet transforms. For some fixed function we calculate filters $U_1, \ldots, U_d$ and use them as filters for other functions. A good idea is to use polynomials,

$$f(x) = x^k.$$

Obtained transforms are automatically orthogonal, and moreover it appears that they also have *zero moments*. We observed this experimentally and based on these experimental facts the following conjecture can be formulated:

Conjecture 1. Suppose that a uniform grid with 2^d points is used on the interval $[a, b]$, and filters $U_1, \ldots, U_d$ are computed by Algorithm 2 for a function $f(x) = x^k$, $k \geqslant 0$. Then the resulting WTT tranform has $k+1$ zero moments, i.e. for the vectors $v^{(s)}$, $s = 0, \ldots, k$, with elements

$$v_i^{(s)} = (x_i)^s, \quad i = 1, \ldots, 2^d,$$

the transformed vector

$$w^{(s)} = Wv^{(s)}$$

has no more than $k + 1$ non-zero elements.

The transform is then uniquely defined by the function f, interval $[a, b]$ and number of grid points determined by d. Note that filters $U_1, \ldots, U_d$ are not necessary the same, but anyway they can be precomputed and stored. We will denote the transform parameters computed for a function f as

$$W = \mathrm{WTT}(f),$$

where the grid size and interval are supposed to be fixed (otherwise, if they are not specified, notation $W = \mathrm{WTT}(f, [a, b], d)$ can be used).

So, polynomials x^k give rise to *orthogonal wavelet transforms* with $(k + 1)$ zero moments. It is an open question, whether these transforms coincide with some known ones, and this requires a separate study.

5.2. Numerical comparison with Daubechies transform

In this paper we just compare this transform with Daubechies transform of the same order for the matrix compression. A given $2^d \times 2^d$ matrix A is transformed as

$$\widehat{A} = WAW^\top,$$

where W is a suitable wavelet transform, and sparsity or pseudo-sparsity of $\widehat{A}$ is studied.

Example 3. *Consider a matrix*

$$A_{ij} = \begin{cases} \frac{1}{i-j}, & i \neq j \\ 0, & i = j \end{cases}, \quad 1 \leqslant i, j \leqslant 2^d.$$

This example was also considered in [16]. The results are presented in Table 3. Four transforms are compared: WTT(x) (with two zero moments), WTT(x^3) (with four zero moments), D4 (with two zero moments), and D8 (with four zero moments). The elements of transformed matrices are thresholded at parameter ε, i.e. all elements smaller in modulus than $\varepsilon \max(A_{ij})$ are set to zero, and the number of non-zero elements in this new matrix is calculated. For relatively large ε, the Daubechies transforms are better, but when ε becomes smaller, a clear advantage of WTT transforms (with the same number of zero moments) is observed.

ε	nnz(WTT(x))	nnz(WTT(x^3))	nnz(D4)	nnz(D8)
10^{-4}	43732	46546	54002	35456
10^{-6}	95898	71356	194188	66740
10^{-8}	196518	116238	588914	141794
10^{-10}	376378	182716	814022	291732
10^{-12}	623212	283972	926328	549962

Table 3. Comparison of WTT and Daubechies transforms for Example 3

Example 4. *The second example, also taken from [16], is*

$$A_{ij} = \begin{cases} \frac{\log|i-n/2| - \log|j-n/2|}{i-j}, & i \neq j, i \neq n/2, j \neq n/2 \\ 0, & \textit{otherwise.} \end{cases}$$

The results are presented in Table 4, and in this case the WTT transform also performs better than the Daubechies transforms.

ε	nnz(WTT(x))	nnz(WTT(x^3))	nnz(D4)	nnz(D8)
10^{-4}	5798	5388	16110	35456
10^{-6}	28384	17654	97734	40962
10^{-8}	93402	43438	481866	95140
10^{-10}	235168	83842	749978	220416
10^{-12}	485312	147922	851230	491856

Table 4. Comparison of WTT and Daubechies transforms for Example 4

6. Two-dimensional case

6.1. How to apply WTT in two dimensions

Algorithms 2, 3 can be applied to any d-dimensional tensors, not only ones that come from reshaping of the values of a one-dimensional function on a uniform grid. It computes a compressed representation of d-dimensional array, which consists of a set of matrices $U_1, \ldots, U_d$ and a supposedly sparse tensor. Thus, it is a good idea to try to extend this approach to other cases for which we will consider the same matrix compression problem as the one studied in the previous Section. Suppose that a bivariate function $f(x, y)$ is given and a square matrix $A = [A_{ij}]$ of order $n = 2^d$ consists of the values of this function on a rectangular grid. Wavelet compression of such matrices, as already mentioned in the previous section, usually consists in the application of one-dimensional transforms to the columns and rows of A:

$$\widehat{A} = WAW^\top,$$

with the hope that $\widehat{A}$ is pseudo-sparse. If A is nonsingular, the number of nonzeros in A can not be smaller than $\mathcal{O}(n)$. However, at least for the examples considered in the previous section, this can be reduced greatly by using ideas presented in [5].

Given a $2^d \times 2^d$ matrix A, multi-indices $(i_1, i_2, \ldots, i_d)$ and $(j_1, j_2, \ldots, j_d)$ can be used to point to rows of and columns of A, then A is reshaped into a $2d$-dimensional tensor $\mathbf{A}$ as follows:

$$A(i_1, i_2, \ldots, i_d; j_1, j_2, \ldots, j_d).$$

In the Matlab notation,

$$A = \mathrm{reshape}(A, [2, \ldots, 2]).$$

We need a more suitable tensor to represent A. To get it, we permute the indices in the Matlab style so that each pair (i_k, j_k) is treated as one long index. The resulting tensor $\mathbf{B}$ is defined by the assignments

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

where $\mathbf{B}$ is a d-dimensional tensor of size $4 \times 4 \times \ldots \times 4$. Then, a TT decomposition (initially called TTM [4]) is constructed for this tensor:

$$B(i_1 j_1, i_2 j_2, \ldots, i_d, j_d) \approx \sum_{\alpha_1, \ldots, \alpha_{d-1}} G_1(i_1, j_1, \alpha_1) G_2(\alpha_1, i_2, j_2, \alpha_2) \ldots G_{d-1}(\alpha_{d-1}, i_d, j_d),$$

the carriages (cores) containing 4 indices instead of 3. This can be justly called *tenzorization of a matrix*, as in [11]. The permutation is necessary to get a good compression for nonsingular matrices.

Algorithms 2 and 3 are now applied to the d-dimensional array $\mathbf{B}$. First, consider the same matrices as in Examples 3 and 4 and opt for ε-rank selection strategy. The memory in this case is to store the elements of $U_1, \ldots, U_d$, and comparison with the Daubechies transform is presented in Table 5 (for Example 3) and Table 6 (for Example 4). Here the value of ε is different from one in the previous section, WTT transforms are performed with $\varepsilon = 10^{-8}$. The results of compression for matrices from Example 4 are presented in Table 6.

$n = 2^d$	Memory(WTT)	nnz(D4)	nnz(D8)	nnz(D20)
2^5	388	992	992	992
2^6	752	4032	3792	3348
2^7	1220	15750	13246	8662
2^8	1776	59470	41508	20970
2^9	2260	213392	102078	45638
2^{10}	2744	780590	215738	95754
2^{11}	3156	1538944	306880	176130

Table 5. Comparison of two-dimensional WTT and Daubechies transforms for Example 3, $\varepsilon = 10^{-8}$

$n = 2^d$	Memory(WTT)	nnz(D4)	nnz(D8)	nnz(D20)
2^5	1568	1024	1024	1024
2^6	7200	4096	3822	3836
2^7	20256	15750	13418	10090
2^8	38264	54814	37004	23330
2^9	58688	194520	77642	48790
2^{10}	81300	732326	164218	100246
2^{11}	104856	2397430	299776	196424

Table 6. Comparison of two-dimensional WTT and Daubechies transforms for Example 4, $\varepsilon = 10^{-8}$

As seen from Table 6, WTT is worse for small n (up to 1024 if compared to D20 transform), but the compression becomes better when n increases. Moreover, in both examples the memory behaves linear in d, or logarithmically in n. This is impossible to achieve when using tensor-product two-dimensional transforms.

6.2. Cases where ε-strategy does not work

There is a simple case where ε-strategy does not work. Consider a matrix A_{ij} which is a discretized version of "disk on a gray background":

$$A_{ij} = \begin{cases} 2, & (i - n/2)^2 + (j - n/2)^2 \leqslant \frac{n^2}{4} \\ 1, & \text{otherwise.} \end{cases} \tag{5}$$

Exact rank of matrices appearing during the TT-SVD algorithm can be characterized by the following theorem [5].

Theorem 1. *The TT-ranks r_k are equal to the tensor (Kronecker-product) ranks of A for different block splittings of A:*

$$r_k = \operatorname{trank}_{(2^k, 2^{d-k})}(A), k = 1, \ldots, 2^d,$$

or in other words, if A is partitioned into $2^k \times 2^k$ blocks $B_{pq}, p, q = 1, \ldots, 2^{d-k}$, then r_k is equal to the dimension of linear space spanned by matrices B_{pq}.

For this example, it is easy to see that each spliiting into $2^k \times 2^k$ blocks produces $\mathcal{O}(n)$ different linearly independent blocks, therefore

$$r_k = \mathcal{O}(n),$$

and the memory to store matrices $U_1, \ldots, U_d$ in this case is $\mathcal{O}(n^2)$, thus no compression is provided for this simple example. However, WTT transform with limited maximal ranks works well. For this example, there is a natural accuracy level related to discrete representation of a circle on a rectangular grid, rougly $\frac{1}{n}$, and that is confirmed by experiment. If the threshold parameter ε is set to 10^{-2}, the number of large elements is small, whereas for $\varepsilon = 10^{-4}$ it increases quite a lot. Such matrices appear in image processing, and this kind of accuracy is more than enough in these applications.

ε	nnz(WTT2), $r_{max} = 2$	nnz(WTT2), $r_{max} = 4$	D4
10^{-2}	131	86	107
10^{-4}	9524	15129	11561

Table 7. Comparison of WTT and Daubechies transforms for (5), $n = 1024$

6.3. How to improve adaptive wavelet transform

From Table 7 it is seen that the selection of filters by using Algorithm 2 which is based on SVD is non-optimal for this matrix. An open question is how to select these filters optimally. The selection of the filter U_k at each step is based on the SVD decomposition, which tries to minimize the norms of rows in $U_k^\top A$ at each step: rows with larger indices are made as small as possible in norm, but their norms are ordered, so the first row

has largest norm (first singular vector), second one the second in magnitude (second singular vector) and so on. It is not clear, why this method makes *pseudo-sparse singular vectors*. A more natural objective is the following: find an orthogonal matrix U_k so that it makes the matrix $U_k^\top A$ as sparse as possible.

However, it is also not the full truth: some rows of $U_k^\top A$ will be compressed further on, and they should not be very sparse, so the detailed formulation of the objective function to be minimized is still an open question and a topic for future research. In any case, surprisingly, the SVD gives *a very good approximation* to the matrix U_k that makes $U_k^\top A$ pseudo-sparse, and it can be used as an initial guess for other methods. Surely, ideas of compressed sensing and L_1-norm minimization seem to be very helpful.

7. Conclusion and future research

In this paper we showed that recently intoduced QTT decomposition has a direct connection with discrete wavelet transform algorithms. QTT can be reformulated as an algebraic wavelet transform, which tries to adapt to a given function in constructing a sequence of linear filters. These transforms are orthogonal by construction. Moreover, experiments show that the filters constructed using the TT-SVD algorithm for polynomial function x^k give rise to orthogonal discrete wavelet transforms with $k+1$ zero moments (this is still a conjecture to be proved), which is comparable to the Daubechies transform. The WTT is a general approach for compression of a d-dimensional array, so it can be applied also to tensors arising from tensorization of matrices, and it can be much more efficient than tensor-product wavelet transforms. In certain cases (like a "disk on a gray background image"), ε-strategy for selecting ranks is useless because of large ranks, and instead, a mixed wavelet-QTT idea is used: despite the fact that singular values are not small, smallest singular vectors are sparse, and the WTT transform gives a sparse vector after transformation with very small ranks. The main question for future research is the correct *objective functional* that should be optimized in construction of adaptive filters. In this case, quite surprisingly, the singular value decomposition gives adequate solutions that can be used as initial guesses. None the less, the SVD does not lead to the sparsest possible solution, in this regard the techniques of compressed sensing may look as useful.

The authors are grateful to Gilbert Strang for his interest in the paper and useful comments that helped to improve the presentation.

References

[1] *Strang G., Nguyen T.* Wavelets and filter banks. — Wellesley Cambridge Pr, 1996.

[2] *Daubechies I.* Ten lectures on wavelets. — Society for Industrial Mathematics, 1992.

[3] *Khoromskij B. N.* $\mathcal{O}(d \log N)$-Quantics Approximation of Nd Tensors in High-Dimensional Numerical Modeling. 2009. http://en.scientificcommons.org/52484259.

[4] *Oseledets I. V.* Approximation of matrices with logarithmic number of parameters // *Doklady Math.* 2009. V. 80, № 2. P. 653–654. http://www.springerlink.com/index/10.1134/S1064562409050056.

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

[6] *Oseledets I. V.* A new tensor decomposition // *Doklady Math.* 2009. V. 80, № 1. P. 495–496.

[7] *Oseledets I. V., Tyrtyshnikov E. E.* TT-cross approximation for multidimensional arrays // *Linear Algebra Appl.* 2010. — . V. 432, № 1. P. 70–88. http://linkinghub.elsevier.com/retrieve/pii/S0024379509003747.

[8] *Oseledets I. V., Tyrtyshnikov E. E.* Tensor-tree decomposition does not need a tree // Linear Algebra Appl. — 2010.

[9] *Oseledets I. V., Tyrtyshnikov E. E.* Breaking the Curse of Dimensionality, Or How to Use SVD in Many Dimensions // *SIAM J. Sci. Comp.* 2009. V. 31, № 5. P. 3744. http://link.aip.org/link/SJOCE3/v31/i5/p3744/s1&Agg=doi.

[10] *Tyrtyshnikov E. E.* Tensor approximations of matrices generated by asymptotically smooth functions // *Sbornik: Mathematics.* 2003. — . V. 194, № 6. P. 941–954. http://stacks.iop.org/1064-5616/194/i=6/a=A09?key=crossref.759fa240378703afa76f11293a94de57.

[11] *Grasedyck L.* Polynomial Approximation in Hierarchical Tucker Format by Vector-Tensorization. 2010. http://www.dfg-spp1324.de/download/preprints/preprint043.pdf.

[12] *Kolda T. G., Bader B. W.* Tensor Decompositions and Applications // *SIAM Review.* 2009. V. 51, № 3. P. 455. http://link.aip.org/link/SIREAD/v51/i3/p455/s1&Agg=doi.

[13] *Khoromskij B. N.* Tensors-structured Numerical Methods in Scientific Computing: Survey on Recent Advances. 2010. — .

[14] *Oseledets I. V.* Compact matrix form of the d-dimensional tensor decomposition. 2009.

[15] *de Silva V., Lim L.-H.* Tensor Rank and the Ill-Posedness of the Best Low-Rank Approximation Problem // *SIAM J. Matrix Anal. Appl.* 2008. V. 30, № 3. P. 1084. http://link.aip.org/link/SJMAEL/v30/i3/p1084/s1&Agg=doi.

[16] *Beylkin G., Coifman R., Rokhlin V.* Fast wavelet transforms and numerical algorithms I // *Comm. Pure Appl. Math.* 1991. V. 44, № 2. P. 141–183.