SIAM J. SCI. COMPUT.
Vol. 33, No. 5, pp. 2295–2317

TENSOR-TRAIN DECOMPOSITION*

I. V. OSELEDETS†

Abstract. A simple nonrecursive form of the tensor decomposition in d dimensions is presented. It does not inherently suffer from the curse of dimensionality, it has asymptotically the same number of parameters as the canonical decomposition, but it is stable and its computation is based on low-rank approximation of auxiliary *unfolding matrices*. The new form gives a clear and convenient way to implement all basic operations efficiently. A fast rounding procedure is presented, as well as basic linear algebra operations. Examples showing the benefits of the decomposition are given, and the efficiency is demonstrated by the computation of the smallest eigenvalue of a 19-dimensional operator.

Key words. tensors, high-dimensional problems, SVD, TT-format

AMS subject classifications. 15A23, 15A69, 65F99

DOI. 10.1137/090752286

1. Introduction.

Tensors are natural multidimensional generalizations of matrices and have attracted tremendous interest in recent years. Multilinear algebra, tensor analysis, and the theory of tensor approximations play increasingly important roles in computational mathematics and numerical analysis [8, 7, 9, 5, 14]; see also the review [26]. An efficient representation of a tensor (by tensor we mean only an array with d indices) by a small number of parameters may give us an opportunity and ability to work with d-dimensional problems, with d being as high as 10, 100, or even 1000 (such problems appear in quantum molecular dynamics [28, 40, 27], stochastic partial differential equations [1, 2], and financial modelling [35, 41]). Problems of such sizes cannot be handled by standard numerical methods due to the *curse of dimensionality*, since everything (memory, amount of operations) grows exponentially in d. There is an effective way to represent a large class of important d-dimensional tensors by using the canonical decomposition of a given tensor $\mathbf{A}$ with elements $A(i_1, \dots, A_d)$ [19, 6]:[1]

$$A(i_1, i_2, \dots, i_d) = \sum_{\alpha=1}^{r} U_1(i_1, \alpha) U_2(i_2, \alpha) \dots U_d(i_d, \alpha). \tag{1.1}$$

The minimal number of summands r required to express $\mathbf{A}$ in form (1.1) is called the *tensor rank* (or the canonical rank). The matrices $U_k = [U_k(i_k, \alpha)]$ are called *canonical factors*. For large d the tensor $\mathbf{A}$ is never formed explicitly but represented in some low-parametric format. The canonical decomposition (1.1) is a good candidate for such a format. However, it suffers from several drawbacks. The computation of the canonical rank is an NP-hard problem [20], and the approximation with a

*Submitted to the journal's Methods and Algorithms for Scientific Computing section March 10, 2009; accepted for publication (in revised form) June 19, 2011; published electronically September 22, 2011. This work was supported by RFBR grant 09-01-00565 and RFBR/DFG grant 09-01-91332, by Russian Government contracts П940, П1178, and П1112, by Russian President grant MK-140.2011.1, and by Priority Research Program OMN-3.

http://www.siam.org/journals/sisc/33-5/75228.html

†Institute of Numerical Mathematics, Russian Academy of Sciences, Gubkina Street 8, Moscow, Russia (ivan.oseledets@gmail.com).

[1] In this paper, tensors are denoted by boldface letters, i.e. $\mathbf{A}$; their elements by a normal letter with MATLAB-like notation, i.e., $A(i_1, i_2, \dots, i_d)$; and matricizations of a tensor by a normal letter with a suitable index.

fixed canonical rank in the Frobenius norm can be ill-posed [10]; thus the numerical algorithms for computing an approximate representation in such cases might fail. Also, even the most successful existing algorithms [12, 4, 3] for computing the best low-tensor-rank approximation are not guaranteed to work well even in cases where a good approximation is known to exist. It is often the case that they encounter local minima and are stuck there. That is why it is a good idea to look at the alternatives for the canonical format, which may have a larger number of parameters but are much better suited for the numerical treatment.

The *Tucker format* [36, 8] is stable but has exponential in d number of parameters, $\mathcal{O}(dnr+r^d)$. It is suitable for "small" dimensions, especially for the three-dimensional case [22, 30, 23]. For large d it is not suitable.

Preliminary attempts to present such new formats were independently made in [32] and [18] using very different approaches. Both of these approaches rely on a hierarchical tree structure and reduce the storage of d-dimensional arrays to the storage of auxiliary three-dimensional ones. The number of parameters in principle can be larger than for the canonical format, but these formats are based entirely on the *singular value decomposition* (SVD). In [32] an algorithm which computes a tree-like decomposition of a d-dimensional array by recursive splitting was presented, and convincing numerical experiments were given. The process goes from the top of the tree to its bottom. In [18] the construction is entirely different; since it goes from the bottom of the tree to its top, the authors presented only a concept and did not present any numerical experiments. Convincing numerical experiments were presented in [15] half a year after, and that justifies that new tensor formats are very promising. The tree-type decompositions [32, 18, 15] depend on the splitting of spatial indices and require recursive algorithms which may complicate the implementation. By carefully looking at the parameters, defining the decomposition, we found that it can be written in a simple but powerful matrix form.

We approximate a given tensor $\mathbf{B}$ by a tensor $\mathbf{A} \approx \mathbf{B}$ with elements

$$A(i_1, i_2, \ldots, i_d) = G_1(i_1)G_2(i_2)\ldots G_d(i_d), \tag{1.2}$$

where $G_k(i_k)$ is an $r_{k-1} \times r_k$ matrix. The product of these parameter-dependent matrices is a matrix of size $r_0 \times r_d$, so "boundary conditions" $r_0 = r_d = 1$ have to be imposed. Compare (1.2) with the definition of a rank-1 tensor: it is a quite straightforward block generalization of the rank-1 tensor. As will be shown in this paper, one of the differences between (1.2) and the canonical decomposition (1.1) is that the ranks r_k can be computed as the ranks of certain auxiliary matrices. Let us write (1.2) in the index form. Matrix $G_k(i_k)$ is actually a three-dimensional array, and it can be treated as an $r_{k-1} \times n_k \times r_k$ array with elements $G_k(\alpha_{k-1}, n_k, \alpha_k) = G_k(i_k)_{\alpha_{k-1}\alpha_k}$.

In the index form the decomposition is written as[2]

$$A(i_1, \ldots, i_d) = \sum_{\alpha_0,\ldots,\alpha_{d-1},\alpha_d} G_1(\alpha_0, i_1, \alpha_1)G_2(\alpha_1, i_2, \alpha_2)\ldots G_d(\alpha_{d-1}, i_d, \alpha_d). \tag{1.3}$$

Since $r_0 = r_d = 1$ this decomposition can also be represented graphically by a *linear tensor network* [21, 39], which is presented in Figure 1.1 for $d = 5$. This graphical

[2]We will make abuse of the notation: by $G_k(i_k)$ we denote an $r_{k-1} \times r_k$ matrix, present in the definition of the tensor train format, depending on the integer parameter i_k. Along the same lines, by $G_k(\alpha_{k-1}, i_k, \alpha_k)$ we will denote the elements of the matrix $G_k(i_k)$. The precise meaning will be clear from the context.

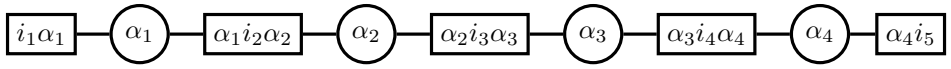

FIG. 1.1. *Tensor-train network.*

representation means the following. There are two types of nodes. Rectangles contain spatial indices (i.e., the indices i_k of the original tensor) and some auxiliary indices α_k, and a tensor with these indices is associated with such kind of nodes. Circles contain only the auxiliary indices α_k and represent a *link*: if an auxiliary index is present in two cores, we connect it. The summation over the auxiliary indices is assumed; i.e., to evaluate an entry of a tensor, one has to multiply all tensors in the rectangles and then perform the summation over all auxiliary indices. This picture looks like a train with carriages and links between them, and that justifies the name *tensor train decomposition*, or simply *TT-decomposition*. The ranks r_k will be called compression ranks or TT-ranks, three-dimensional tensors G_k—cores of the TT-decomposition (analogous to the core of the Tucker decomposition). There are more general types of tensor networks, represented as graphs; however, only few of them possess good numerical properties. The TT-format (also known in other areas as a linear tensor network (LTN) or a matrix product state (MPS); cf. [21, 39]) has several features that distinguish it from the other types of networks, and the corresponding numerical algorithms will be presented in this paper. Our main goal is to represent tensors in the TT-format and perform operations with them efficiently. Not only are the exact decompositions of interest, but also the approximations (which are more common in scientific computing) with a prescribed accuracy ε. (This means replacing the initial tensor $\mathbf{A}$ with its approximation $\mathbf{B}$ in the TT-format such that $||A - B||_F \leq \varepsilon ||B||_F$ holds.)

Thus, approximate operations have to be performed with such tensors which reduce the storage while maintaining the accuracy. To do that, we need to answer the following questions:

- How to compute the ranks r_k (or approximate ranks with a prescribed accuracy ε) for a given tensor $\mathbf{A}$?
- If a tensor is already in the TT-format, how to find the optimal TT-ranks r_k, given the required accuracy level ε? (This is similar to *rounding* in the finite-precision computer arithmetic, but instead of digits we have a nonlinear low-parametric approximation of a tensor.)
- How to implement basic linear algebra (addition, scalar product, matrix-by-vector product, and norms) in the TT-format?
- How to convert from other tensor formats, like the canonical decomposition?

2. Definition of the format and compression from the full array to the TT-format. Let us establish basic properties of the TT-format. A d-dimensional $n_1 \times n_2 \times \cdots \times n_d$ tensor $\mathbf{A}$ is said to be in the TT-format with cores G_k of size $r_{k-1} \times n_k \times r_k$, $k = 1, \ldots, d, r_0 = r_d = 1$, if its elements are defined by formula (1.3). It is easy to get a bound on r_k. Each α_k appears only twice in (1.3), and thus it is bounded from below by the rank of the following *unfolding matrix* of $\mathbf{A}$:

$$A_k = A_k(i_1, \ldots, i_k; i_{k+1} \ldots i_d) = A(i_1, \ldots, i_d); \tag{2.1}$$

i.e., the first k indices enumerate the rows of A_k, and the last $d - k$ the columns of A_k. (On the left side of (2.1) there is an element of A_k in row $(i_1, \ldots, i_k)$ and column $(i_{k+1}, \ldots, i_d)$, whereas on the right side there is an element of $\mathbf{A}$ in position

$(i_1, \ldots, i_d)$.) The size of this matrix is $(\prod_{s=1}^{k} n_s) \times (\prod_{s=k+1}^{d} n_s)$, and in MATLAB it can be obtained from the tensor $\mathbf{A}$ by a single call to the **reshape** function:

$$A_k = \text{reshape}\left(\mathbf{A}, \prod_{s=1}^{k} n_s, \prod_{s=k+1}^{d} n_s\right).$$

Moreover, these ranks are achievable, as shown by the following theorem, which also gives a constructive way to compute the TT-decomposition.

THEOREM 2.1. *If for each unfolding matrix A_k of form* (2.1) *of a d-dimensional tensor* $\mathbf{A}$

$$\text{rank}\, A_k = r_k, \tag{2.2}$$

then there exists a decomposition (1.3) *with TT-ranks not higher than r_k.*

Proof. Consider the unfolding matrix A_1. Its rank is equal to r_1; therefore it admits a dyadic (skeleton) decomposition

$$A_1 = UV^\top,$$

or in the index form

$$A_1(i_1; i_2, \ldots, i_d) = \sum_{\alpha_1=1}^{r_1} U(i_1, \alpha_1) V(\alpha_1, i_2, \ldots, i_d).$$

The matrix V can be expressed as

$$V = A_1^\top U (U^\top U)^{-1} = A_1^\top W,$$

or in the index form

$$V(\alpha_1, i_2, \ldots, i_d) = \sum_{i_1=1}^{n_1} A(i_1, \ldots, i_d) W(i_1, \alpha_1).$$

Now the matrix V can be treated as a $(d-1)$-dimensional tensor $\mathbf{V}$ with $(\alpha_1 i_2)$ as one long index:

$$\mathbf{V} = V(\alpha_1 i_2, i_3, \ldots, i_d).$$

Now consider its unfolding matrices $V_2, \ldots, V_d$. We will show that $\text{rank}\, V_k \le r_k$ holds. Indeed, for the kth mode the TT-rank is equal to r_k; therefore $\mathbf{A}$ can be represented as

$$A(i_1, \ldots, i_d) = \sum_{\beta=1}^{r_k} F(i_1, \ldots, i_k, \beta) G(\beta, i_{k+1}, \ldots, i_d).$$

Using that, we obtain

$$\begin{aligned} V_k &= V(\alpha_1 i_2, \ldots, i_k; i_{k+1}, \ldots, i_d) \\ &= \sum_{i_1=1}^{n_1} \sum_{\beta=1}^{r_k} W(i_1, \alpha_1) F(i_1, \ldots, i_k, \beta) G(\beta, i_{k+1}, \ldots, i_d) \\ &= \sum_{\beta=1}^{r_k} H(\alpha_1 i_2, \ldots, i_k, \beta) G(\beta, i_{k+1}, \ldots, i_d), \end{aligned}$$

where

$$H(\alpha_1 i_2, \ldots, i_k, \beta) = \sum_{i_1=1}^{n_1} F(i_1, \ldots, i_k, \beta) W(i_1, \alpha_1).$$

Row and column indices of V_k are now separated and

$$\operatorname{rank} V_k \le r_k.$$

The process can be continued by induction. Consider $\mathbf{V}$ and separate the index (α_1, i_2) from others:

$$V(\alpha_1 i_2, i_3, \dots, i_d) = \sum_{\alpha_2=1}^{r_2} G_2(\alpha_1, i_2, \alpha_2) V'(\alpha_2 i_3, i_4, \dots, i_d).$$

This yields the next core tensor $G_2(\alpha_1, i_2, \alpha_2)$ and so on, up to $G_d(\alpha_{d-1}, i_d)$, finally giving the TT-representation. ∎

Low-rank matrices rarely appear in practical computations. Suppose that the unfolding matrices are of low rank only approximately, i.e.,

$$A_k = R_k + E_k, \quad \operatorname{rank} R_k = r_k, \quad ||E_k||_F = \varepsilon_k, \quad k = 1, \dots, d-1. \tag{2.3}$$

The proof of the Theorem 2.1 is constructive and gives an algorithm for computing the TT-decomposition using d sequential SVDs of auxiliary matrices. This algorithm will be called the *TT-SVD algorithm*. It can be modified to the approximate case, when instead of exact low-rank decomposition, the best rank-r_k approximation via the SVD is computed. Then, the introduced error can be estimated.

THEOREM 2.2 (see [29]). *Suppose that the unfoldings A_k of the tensor $\mathbf{A}$ satisfy* (2.3). *Then TT-SVD computes a tensor $\mathbf{B}$ in the TT-format with TT-ranks r_k and*

$$||\mathbf{A} - \mathbf{B}||_F \le \sqrt{\sum_{k=1}^{d-1} \varepsilon_k^2}. \tag{2.4}$$

Proof. The proof is by induction. For $d = 2$ the statement follows from the properties of the SVD. Consider an arbitrary $d > 2$. Then the first unfolding A_1 is decomposed as

$$A_1 = U_1 \Sigma V_1 + E_1 = U_1 B_1 + E_1,$$

where U_1 is of size $n_1 \times r_1$, has orthonormal columns, and $||E_1|| = \varepsilon_1$. The matrix B_1 is naturally associated with a $(d-1)$-dimensional tensor $\mathbf{B}_1$ with elements $B(\alpha_1 i_2, i_3, \dots, i_d)$, which will be decomposed further in the TT-SVD algorithm. This means that B_1 will be approximated by some other matrix $\widehat{B}_1$. From the properties of the SVD it follows that $U_1^\top E_1 = 0$, and thus

$$\begin{aligned} ||\mathbf{A} - \mathbf{B}||_F^2 &= ||A_1 - U_1 \widehat{B}_1||_F^2 = ||A_1 - U_1(\widehat{B}_1 + B_1 - B_1)||_F^2 \\ &= ||A_1 - U_1 B_1||_F^2 + ||U_1(B_1 - \widehat{B}_1)||_F^2, \end{aligned}$$

and since U_1 has orthonormal columns,

$$||\mathbf{A} - \mathbf{B}||_F^2 \le \varepsilon_1^2 + ||B_1 - \widehat{B}_1||_F^2. \tag{2.5}$$

The matrix B_1 is easily expressed from A_1,

$$B_1 = U_1^\top A_1,$$

and thus it is not difficult to see from the orthonormality of columns of U_1 that the distance of the kth unfolding ($k = 2, \dots, d-1$) of the $(d-1)$-dimensional tensor $\mathbf{B}_1$ to the r_kth rank matrix cannot be larger then ε_k. Proceeding by induction, we have

$$||B_1 - \widehat{B}_1||_F^2 \le \sum_{k=2}^{d-1} \varepsilon_k^2,$$

and together with (2.5), this completes the proof. ∎

From Theorem 2.2 two corollaries immediately follow [29].

COROLLARY 2.3. *If a tensor* $\mathbf{A}$ *admits a canonical approximation with* R *terms and accuracy* ε*, then there exists a TT-approximation with TT-ranks* $r_k \le R$ *and accuracy* $\sqrt{d-1}\,\varepsilon$.

COROLLARY 2.4. *Given a tensor* $\mathbf{A}$ *and rank bounds* r_k*, the best approximation to* $\mathbf{A}$ *in the Frobenius norm with TT-ranks bounded by* r_k *always exists (denote it by* $\mathbf{A}^{best}$*), and the TT-approximation* $\mathbf{B}$ *computed by the TT-SVD algorithm is quasi-optimal:*

$$||\mathbf{A} - \mathbf{B}||_F \le \sqrt{d-1}||\mathbf{A} - \mathbf{A}^{best}||_F.$$

Proof. Let $\varepsilon = \inf_{\mathbf{C}} ||\mathbf{A} - \mathbf{C}||$, where the infimum is taken over all tensor trains with TT-ranks bounded by r_k. Then, by the definition of the infimum, there exists a sequence of tensor trains $\mathbf{B}^{(s)}$ ($s = 1, 2, \dots$) with the property $\lim_{s\to\infty} ||\mathbf{A} - \mathbf{B}^{(s)}||_F = \varepsilon$. All elements of the tensors $\mathbf{B}^{(s)}$ are bounded; hence some subsequence $\mathbf{B}^{(s_t)}$ converges elementwise to some tensor $\mathbf{B}^{(\min)}$, and unfolding matrices also converge: $B_k^{(s_t)} \to B_k^{(\min)}$, $1 \le k \le d$. Since the set of matrices of rank not higher than r_k is closed and $\operatorname{rank} B^{(s_t)} \le r_k$, thus $\operatorname{rank} B_k^{(\min)} \le r_k$. Moreover, $||\mathbf{A} - \mathbf{B}^{(\min)}||_F = \varepsilon$, so $\mathbf{B}^{(\min)}$ is the minimizer. It is now sufficient to note that $\varepsilon_k \le \varepsilon$, since each unfolding can be approximated with at least accuracy ε. The quasioptimality bound follows directly from (2.4). ∎

From Theorem 2.2 it immediately follows that if singular values of unfolding matrices are truncated at δ, the error of the approximation will be $\sqrt{d-1}\delta$, and to obtain any prescribed accuracy ε the threshold δ has to be set to $\frac{\varepsilon}{\sqrt{d-1}}$. Finally, an algorithm for constructing the TT-approximation with prescribed (relative) accuracy is given as Algorithm 1 below. The computed TT-ranks are actually δ-ranks[3] of the unfoldings, where to achieve the required relative accuracy ε one has to select $\delta = \frac{\varepsilon}{\sqrt{d-1}}||A||_F$.

Remark. The number of parameters in the tree format of [32] as well as for the $\mathcal{H}$-Tucker format in [18, 15] is estimated as

$$\mathcal{O}(dnr + (d-2)r^3).$$

It is easy to modify the TT-decomposition to reduce $(d-2)nr^2 + 2nr$ to $dnr + (d-2)r^3$ by using an auxiliary Tucker decomposition [36] of the core tensors G_k. G_k is an $r_{k-1} \times n_k \times r_k$ tensor, and it is not difficult to prove that its mode-2 rank is not higher than t_k, where t_k is the Tucker rank (mode rank) [36] of $\mathbf{A}$ along the kth mode. Therefore each G_k can be replaced by an $n_k \times t_k$ factor matrix and an $r_{k-1} \times t_k \times r_k$ auxiliary three-dimensional array. However, for the simplicity of the presentation we

[3] For a given matrix A its δ-rank is defined as the minimum of rank B over all matrices B satisfying $||A - B||_F \le \delta$.

Algorithm 1. TT-SVD.

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

Ensure: Cores $G_1, \dots, G_d$ of the TT-approximation $\mathbf{B}$ to $\mathbf{A}$ in the TT-format with TT-ranks $\widehat{r}_k$ equal to the δ-ranks of the unfoldings A_k of $\mathbf{A}$, where $\delta = \frac{\varepsilon}{\sqrt{d-1}}||A||_F$. The computed approximation satisfies

$$||\mathbf{A} - \mathbf{B}||_F \leq \varepsilon ||\mathbf{A}||_F.$$

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

omit this step from our decomposition, but places will be pointed out where it can be used to reduce the computational complexity.

Throughout the paper we use the tensor-by-matrix multiplication referred to as the mode-k contraction or the mode-k multiplication. Given an array (tensor) $\mathbf{A} = [A(i_1, i_2, \dots, i_d)]$ and a matrix $U = [U(i_k, \alpha)]$, we define the mode-k multiplication result as a new tensor $\mathbf{B} = [B(i_1, \dots, \alpha, \dots, i_d)]$ (α is on the kth place) obtained by the contraction over the kth axis:

$$B(i_1, \dots, \alpha, \dots, i_d) = \sum_{i_k=1}^{n_k} A(i_1, i_2, \dots, i_d) U(\alpha, i_k).$$

We denote this operation as follows:

$$\mathbf{B} = \mathbf{A} \times_k U.$$

3. Rounding in TT-format. A full (dense) tensor can be converted into the TT-format with help of the TT-SVD algorithm described in the previous section. However, even computing all entries of the tensor is an expensive task for high dimensions. If the tensor is already in some structured format, this complexity can be reduced. An important case is the case when the tensor is already is given in the TT-format, but with suboptimal ranks r_k. Such tensors can appear in the following context. As will be shown later, many basic linear algebra operations with TT-tensors (addition, matrix-by-vector product, etc.) yield results also in the TT-format, but with increased ranks. To avoid rank growth one has to reduce ranks while maintaining accuracy. Of course, this can be done by the TT-SVD algorithm. But if the tensor is already in the TT-format, its complexity is greatly reduced. Suppose that $\mathbf{A}$ is in the TT-format,

$$A(i_1, i_2, \dots, i_d) = G_1(i_1) G_2(i_2) \dots G_d(i_d),$$

but with increased ranks r_k. We want to estimate the true values of ranks $r'_k \leq r_k$ while maintaining the prescribed accuracy ε. Such a procedure will be called *rounding* (it can be also called *truncation* or *recompression*), since it is analogous to the rounding when working with floating point numbers, but instead of digits and mantissa we have a low-parametric representation of a tensor. First, try to compute r'_1 and reduce this rank. The corresponding unfolding matrix A_1 can be written as a product

$$A_1 = UV^\top, \tag{3.1}$$

where

$$U(i_1, \alpha_1) = G_1(i_1, \alpha_1), \quad V(i_2, i_3, \ldots, i_d; \alpha_1) = G_2(\alpha_1, i_2)G_3(i_3)\ldots G_d(i_d), \tag{3.2}$$

where the rows of U are indexed by i_1, whereas the rows of V are indexed by a multi-index $(i_2, \ldots, i_d)$.

A standard way to compute the SVD of A_1 using the representation of form (3.1) is the following. First, compute QR-decompositions of U and V,

$$U = Q_u R_u, \quad V = Q_v R_v,$$

assemble a small $r \times r$ matrix

$$P = R_u R_v^\top,$$

and compute its reduced SVD:

$$P = XDY^\top,$$

where D is an $\widehat{r} \times \widehat{r}$ diagonal matrix and X and Y are $r \times \widehat{r}$ matrices with orthonormal columns. $\widehat{r}$ is the ε-rank of D (which is equal to the ε-rank of A_1). Finally,

$$\widehat{U} = Q_u X, \quad \widehat{V} = Q_v Y,$$

are matrices of dominant singular vectors of the full matrix A_1.

The U matrix for A_1 is small, so we can compute its QR-decomposition directly. The V matrix, however, is very large, and something else has to be done. We will prove that the QR-decomposition of V can be computed in a structured way, with the Q-factor in the TT-format (and R is small, and can be stored explicitly). The following lemma shows that if the TT-decomposition cores satisfy certain orthogonality properties, then the corresponding matrix has orthogonal columns.

LEMMA 3.1. *If a tensor* $\mathbf{Z}$ *is expressed as*

$$Z(\alpha_1, i_2, \ldots, i_d) = Q_2(i_2)Q_3(i_3)\ldots Q_d(i_d), \tag{3.3}$$

where $Q_k(i_k)$ *is an* $r_{k-1} \times r_k$ *matrix,* $k = 2, \ldots, d, r_d = 1$ *(for fixed* $i_k, k = 2, \ldots, i_d$*, the product reduces to a vector of length* r_1*, which is indexed by* α_1*), and the matrices* $Q_k(i_k)$ *satisfy orthogonality conditions*

$$\sum_{i_k} Q_k(i_k)Q_k^\top(i_k) = I_{r_{k-1}} \tag{3.4}$$

(by I_s we denote an $s \times s$ identity matrix), then $\mathbf{Z}$ *considered as an* $r_1 \times \prod_{k=2}^{d} n_k$ *matrix* Z *with orthonormal rows, i.e.,*

$$(ZZ^\top)_{\alpha_1,\widehat{\alpha}_1} = \sum_{i_2,\dots,i_d} Z(\alpha_1, i_2, \dots, i_d) Z(\widehat{\alpha}_1, i_2, \dots, i_d) = \delta(\alpha_1, \widehat{\alpha}_1).$$

Proof. It is sufficient to see that

$$\begin{aligned} ZZ^\top &= \sum_{i_1,\dots,i_d} \Big(Q(i_1)Q_2(i_2)\dots Q_d(i_d)\Big)\Big(Q(i_1)Q_2(i_2)\dots Q_d(i_d)\Big)^\top \\ &= \sum_{i_1,\dots,i_{d-1}} \Big(Q_1(i_1)\dots Q_{d-1}(i_{d-1})\Big) \underbrace{\Big(\sum_{i_d} Q_d(i_d)Q_d^\top(i_d)\Big)}_{I_{r_d}} \Big(Q_{d-1}^\top(i_{d-1})\dots Q_1^\top(i_1)\Big) \\ &= \sum_{i_1,\dots,i_{d-1}} \Big(Q_1(i_1)\dots Q_{d-1}(i_{d-1})\Big)\Big(Q_{d-1}^\top(i_{d-1})\dots Q_1^\top(i_1)\Big) \\ &= \dots = \sum_{i_1} Q_1(i_1)Q_1^\top(i_1) = I_{r_1}; \end{aligned}$$

i.e., summations over i_k vanish due to the orthogonality conditions (3.4). ∎

Using Lemma 3.1, we can design a fast algorithm for the structured QR-decomposition of the matrix V from (3.1) in the TT-format. The algorithm is a single right-to-left sweep through all cores. The matrix V can be written as

$$V(i_2, \dots, i_d) = G_2(i_2)\dots G_d(i_d).$$

Equivalent transformations of this representation have to be performed to satisfy orthogonality conditions. First, $G_d(i_d)$ is represented as

$$G_d(i_d) = R_d Q_d(i_d),$$

where $Q_d(i_d)$, considered as an $r_{d-1} \times n_d$ matrix (recall that $r_d = 1$), has orthonormal rows. This can be done by considering G_d as an $r_{d-1} \times n_d$ matrix and orthogonalizing its rows. Then,

$$V(i_2, \dots, i_d) = G_2(i_2)\dots G'_{d-1}(i_{d-1})Q_d(i_d),$$

where

$$G'_{d-1}(i_{d-1}) = G_{d-1}(i_{d-1})R_d.$$

Suppose that we already have a representation of form

$$V(i_2, \dots, i_d) = G_2(i_2)\dots G_k(i_k)Q_{k+1}(i_{k+1})\dots Q_d(i_d),$$

where matrices $Q_s(i_s)$ satisfy orthogonality conditions (3.4) for $s = k+1, \dots d$, and we want to transform this representation into an equivalent one that satisfies (3.4) for $s = k$. In order to do that, $G'_k(i_k)$ is represented as a product

$$G'_k(i_k) = R_k Q_k(i_k), \tag{3.5}$$

with some matrix R_k that is independent of i_k and

$$\sum_{i_k} Q_k(i_k)Q_k^\top(i_k) = I_{r_k}. \tag{3.6}$$

Equations (3.5), and (3.6) can be written in the index form:

$$G'_k(\alpha_{k-1}, i_k, \alpha_k) = \sum_{\beta_k} R(\alpha_k, \beta_k) Q_k(\beta_{k-1}, i_k, \alpha_k)$$

and

$$\sum_{i_k, \alpha_k} Q_k(\beta_{k-1}, i_k, \alpha_k) Q_k(\widehat{\beta}_{k-1}, i_k, \alpha_k) = \delta(\beta_k, \widehat{\beta}_k).$$

Thus, Q_k and R_k can be computed via the orthogonalization of the rows of the matrix G obtained from the reshaping of the tensor G_k with elements $G_k(\alpha_{k-1}, i_k, \alpha_k)$ into a matrix of size $r_{k-1} \times (n_k r_k)$, since the second equation is just the orthogonality of the rows of the same reshaping of the tensor Q_k. After this decomposition has been computed, the core $G_{k-1}(i_{k-1})$ is multiplied from the right by R_k, yielding the required representation.

We have presented a way to compute the QR-decomposition of V using only the cores G_k of the TT-decomposition of $\mathbf{A}$, and the Q-factor was computed in a structured form. To perform the compression, we compute the compressed SVD and contract two cores containing α_1 with two small matrices. After the first mode was compressed, we can do the same thing for each mode, since for an arbitrary k we can use the same algorithm to compute the structured QR-decompositions of the U and V factors (the algorithm for U is the same with slight modifications), matrices R_u and R_v, singular values, the reduced rank, and matrices X and Y which perform the dimensionality reduction. However, we can avoid making these decompositions every time for every mode from scratch by using information obtained from previous steps. For example, after we reduced the rank for A_1, we modify cores G_1 and G_2, but cores $G_3, \ldots, G_d$ stay the same and satisfy orthogonality conditions (3.4). Therefore, to compress in the second mode, we just have to orthogonalize G_1 and G_2. This can be realized by storing the R-matrix that appears during the orthogonalization algorithm. In fact we do the following: for A_1 we compute the reduced decomposition of form

$$A_1 = U_1 V_1^\top,$$

where the matrix U_1 has orthonormal columns, and compress V_1, and so on. Since this is equivalent to the TT-SVD algorithm applied to a structured tensor, the singular values have to be cut off at the same threshold $\delta = ||A||_F \frac{\varepsilon}{\sqrt{d-1}}$ as in the full tensor case. The only thing that is left is an estimate of the Frobenius norm. That can be computed from the tensor directly in the TT-format, and we will show how to compute it in the next sections. The formal description of the algorithm is presented in Algorithm 2. A MATLAB code for this algorithm is a part of the TT-Toolbox. By SVD_δ in Algorithm 2 we denote the SVD with singular values that are set to zero if smaller than δ, and by $\mathrm{QR}_{\mathrm{rows}}$ we denote the QR-decomposition of a matrix, where the Q-factor has orthonormal rows. The procedure $\mathrm{SVD}_\delta(A)$ returns three matrices U, Λ, V of the decomposition $A \approx U \Lambda V^\top$ (as MATLAB **svd** function), and the procedure $\mathrm{QR}_{\mathrm{rows}}$ returns two: the Q-factor and the R-factor. The notation $G_k(\beta_k; i_k \beta_k)$ means that the tensor G_k is treated as a matrix with β_{k-1} as a row index and $(i_k \beta_k)$ as a column index. In MATLAB it can be done via a single call to the **reshape** function.

Let us estimate the number of operations required by the algorithm. For simplicity, assume that $r_k \sim r$, $n_k \sim n$. The right-to-left sweep requires successive QR-decompositions of $nr \times r$ matrices which cost $\mathcal{O}(nr^3)$ operations each, in total

Algorithm 2. TT-rounding.

Require: d-dimensional tensor $\mathbf{A}$ in the TT-format, required accuracy ε

Ensure: $\mathbf{B}$ in the TT-format with TT-ranks $\widehat{r}_k$ equal to the δ-ranks of the unfoldings A_k of $\mathbf{A}$, where $\delta = \frac{\varepsilon}{\sqrt{d-1}}||A||_F$. The computed approximation satisfies

$$||\mathbf{A} - \mathbf{B}||_F \leq \varepsilon ||\mathbf{A}||_F.$$

Let G_k, $k = 1, \ldots, d$, be cores of $\mathbf{A}$.
{Initialization}
Compute truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}}||\mathbf{A}||_F$////////////////.
{Right-to-left orthogonalization}
for $k = d$ to 2 step -1 **do**
  $[G_k(\beta_{k-1}; i_k\beta_k), R(\alpha_{k-1}, \beta_{k-1})] := \mathrm{QR}_{\mathrm{rows}}(G_k(\alpha_{k-1}; i_k\beta_k))$.
  $G_{k-1} := G_k \times_3 R$.
end for
{Compression of the orthogonalized representation}
for $k = 1$ to $d - 1$ **do**
  {Compute δ-truncated SVD}
  $[G_k(\beta_{k-1}i_k; \gamma_k), \Lambda, V(\beta_k, \gamma_k)] := \mathrm{SVD}_\delta[G_k(\beta_{k-1}i_k; \beta_k)]$.
  $G_{k+1} := G_{k+1} \times_1 (V\Lambda)^\top$.
end for
Return G_k, $k = 1, \ldots, d$, as cores of $\mathbf{B}$.

$\mathcal{O}(dnr^3)$ operations. The compression step requires SVDs of $(nr) \times r$ matrices, which need $\mathcal{O}(nr^3)$ for each mode. The final estimate is

$$\mathcal{O}(dnr^3)$$

operations for the full compression procedure. By additionally using the Tucker format and applying the TT-decomposition only to its core, we can reduce the complexity to

$$\mathcal{O}(dnr^2 + dr^4),$$

where the first term is just the price of d sequential QR-decompositions of Tucker factors.

3.1. From canonical to TT. The conversion from the canonical decomposition to the TT-format is trivial. The tree structure of [32] led to some difficulties, requiring a recursive algorithm based on the computation of Gram matrices. Here we just have to rewrite the canonical format of form

$$A(i_1, \ldots, i_d) = \sum_\alpha U_1(i_1, \alpha) \ldots U_d(i_d, \alpha)$$

in the TT-format by using Kronecker delta symbols:

$$\begin{aligned} &A(i_1, \ldots, i_d) \\ &= \sum_{\alpha_1\alpha_2\ldots\alpha_{d-1}} U_1(i_1, \alpha_1)\delta(\alpha_1, \alpha_2)U_2(i_2, \alpha_2)\delta(\alpha_2, \alpha_3) \ldots \delta(\alpha_{d-2}, \alpha_{d-1})U_d(i_d, \alpha_{d-1}). \end{aligned}$$

TABLE 3.1
Compression timings (in seconds) for d-dimensional Laplace-like tensor.

$n = 2$		$n = 1024$	
$d = 4$	$8.0 \cdot 10^{-4}$	$d = 4$	$2.3 \cdot 10^{-3}$
$d = 8$	$1.6 \cdot 10^{-3}$	$d = 8$	$2.2 \cdot 10^{-2}$
$d = 16$	$3.8 \cdot 10^{-3}$	$d = 16$	$2.4 \cdot 10^{-1}$
$d = 32$	$1.0 \cdot 10^{-2}$	$d = 32$	$2.3 \cdot 10^{0}$
$d = 64$	$5.1 \cdot 10^{-2}$	Out of memory	Out of memory
$d = 128$	$4.4 \cdot 10^{-1}$	Out of memory	Out of memory

In the matrix form it looks like

$$A(i_1, \dots, i_d) = \Lambda_1(i_1)\Lambda_2(i_2) \dots \Lambda_d(i_d),$$

where

$$\Lambda_k(i_k) = \mathrm{diag}(U(i_k, :)), k = 2, \dots, d-1, \quad \Lambda_1(i_1) = U(i_1, :), \quad \Lambda_d(i_d) = \Big(U(i_d, :)\Big)^\top.$$

$\Lambda_k(i_k)$ are diagonal matrices for each fixed i_k, except for $k = 1$ and $k = d$. Then we can compress the resulting TT-tensor by using the rounding procedure described above. For example, consider a discretization of the d-dimensional Laplace operator of form

$$\Delta_d = \Delta \otimes I \otimes \dots \otimes I + \dots + I \otimes \dots \otimes \Delta, \tag{3.7}$$

where $\otimes$ is the Kronecker product of matrices and Δ is a standard second-order discretization of the one-dimensional Laplace operator with the Dirichlet boundary conditions (up to a scaling constant which is not important for us):

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

Now let describe how we use the TT-format here. The rows of the matrix Δ_d can be naturally indexed by a multi-index $(i_1, i_2, \dots, i_d)$ and its columns by a multi-index $(j_1, j_2, \dots, j_d)$. To make it a tensor, each pair (i_k, j_k) is treated as a one long index, and (3.7) transforms into a rank-d canonical representation. This tensor is a tensor from $\otimes^d V$, where $\dim V$ is a two-dimensional vector space. Because all two-dimensional vector spaces are isomorphic, the computations can be done in the space $\otimes^d \mathbb{R}^2$ for "Laplace-like" tensors of form

$$\mathbf{A} = a \otimes b \otimes \dots \otimes b + \dots + b \otimes \dots \otimes a. \tag{3.8}$$

For such tensors all TT-ranks are equal to 2, since they can be can be approximated by a tensor of rank 2 with arbitrary precision [3].

The Laplace operator is often encountered in the applications, so it may be useful to derive a special algorithm for it. To approximate the Laplace operator, we do the following: for a tensor of form (3.8), derived from the Laplace operator in d dimensions, the TT-representation with TT-ranks $r_k = d$ is obtained by using the canonical-to-TT transformation. Then Algorithm 2 is run. The results are presented in Table 3.1. Not surprisingly, the approximation error here is of the order of machine precision, since all TT-ranks are equal to 2. The computational timings depend only on n and d but not on actual vectors a and b. Note that in Table 3.1 the case $n = 1024$

is treated directly, without exploring the isomorhism to $n = 2$, to illustrate the numerical complexity while working with large mode sizes. In practical computations this should not be done, and the problem should be reduced to the case $n = 2$ directly.

The time to compress a 32-dimensional operator is of the order of a second, and taking into account that for Laplace-like operators we need to compress only a $2\times 2\times\cdots\times 2$ core tensor, this dimension can be as high as 128. The only restriction at this point is a memory restriction (not for the TT-format but for storing intermediate arrays), and it can be passed by using a machine with a larger amount of memory. As in [32], all ranks involved are equal to 2, and the tensor is represented by a set of $(d-2)$ arrays of sizes $2\times n\times 2$ and two $n\times 2$ matrices. If Tucker format is used additionally, the number of parameters will be reduced to $\mathcal{O}(2dn+8(d-2))$ for a d-dimensional Laplace-like operator (compare to $\mathcal{O}(d^2n)$ in the canonical format[4]). Another interesting example is the discretization of the second-order differential operator of form

$$\mathcal{L}P = \sum_{i,j=1,i<j}^{d} \sigma_{ij} \frac{\partial^2 P}{\partial x_i \partial x_j}, \tag{3.9}$$

$$A = \sum_{i,j=1,i<j}^{d} \sigma_{ij} W_i W_j, \tag{3.10}$$

where W_i is acting only in the ith mode:

$$W_i = I \otimes \cdots \times \underbrace{B_i}_{i} \otimes \cdots \otimes I.$$

The matrix B_i is a discrete analogue of the gradient operator. If B_i is an $m \times m$ matrix, then the tensor product of matrices gives an $m^d \times m^d$ matrix, which is then transformed into an $m^2\times m^2\times\cdots\times m^2$ d-dimensional tensor, just as in the Laplace-like case. The general form of such tensors can be written as

$$\mathbf{A} = \sum_{i,j=1,i<j}^{d} \sigma_{ij} \left(c \otimes \cdots \otimes \underbrace{a}_{i} \otimes c \otimes \cdots \otimes \underbrace{b}_{j} \otimes c \otimes \cdots \otimes c \right), \tag{3.11}$$

where a, b, c are some vectors of length n. For any a, b, c, $\mathbf{A}$ is a tensor of canonical rank at most $\frac{d(d-1)}{2}$. For $\sigma_{ij} = 1$ this is an electron-type potential considered in [3], and it was proved there that such a tensor can be approximated by a rank-3 tensor with arbitrary precision. Analogous estimates for the case of general σ_{ij} are unknown currently, but we can provide experimental results and give a bound on the TT-ranks of such tensors. The results are quite interesting: It appears that the ranks depend only on d. We will call matrices of form (3.10) *Scholes-like* matrices (and corresponding tensors of form (3.11). Scholes-like tensors) since they appear in the Black–Scholes equation for multiasset option pricing [33].

For example, for $d = 19$ the ranks are given in Table 3.2. The coefficients σ_{ij} were taken at random, and we did not observe any dependence on them. (There are special cases where the rank is smaller, but for the general case these ranks should be the same, since we observe that decompositions are exact.) The initial canonical

[4]Of course, to store the Laplace-like tensor only $2n$ parameters are needed for the vectors a and b. However, the TT-format is intended for performing fast arithmetic operations with such tensors. In the arithmetic operations the ranks (canonical or TT) are crucial, since the special structure of factors will be destroyed.

TABLE 3.2
TT-ranks for different modes.

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

rank was 171, so the TT-ranks are much smaller than the canonical rank. Based on numerical experiments, we can conjecture that the highest rank is $\approx \frac{d}{2}$. To prove this conjecture, consider the unfolding A_k of the Scholes-like tensor: The matrix

$$\left[c \otimes \cdots \otimes c, \sum_{i,j=1, i<j}^{k} \sigma_{ij} c \otimes \cdots \otimes a \otimes \cdots \otimes b \otimes \cdots \otimes c, \right.$$
$$\left. a \otimes c \cdots \otimes c, \ldots, c \otimes \cdots \otimes c \otimes a\right]$$

spans the columns of this unfolding, and therefore the rank is bounded by $r_k \leq 2 + k$. A similar reasoning for the rows of each unfolding leads to $r_k \leq 2 + \min\{k, d-k\}$, which is a sharp bound for the observed ranks. The maximum is obtained when $k \approx d - k$, i.e., $r_k \approx \frac{d}{2}$.

All Tucker ranks are equal to 3 (only three basis vectors in each mode); therefore an estimate for the storage of the Scholes-like operator is $\mathcal{O}(dn) + \mathcal{O}(d^2)$ instead of the $\mathcal{O}(d^3 n)$ parameters for the canonical format and $O(dn) + O(d^3)$ for the combined CP and Tucker format. (The situation is the same as for the Laplace-like tensors: The storage of the canonical format reduces to to $\frac{d(d-1)}{2} + 3n$ if identical vectors are stored only once, but this special structure can be destroyed during the subsequent arithmetic operations with such tensors.)

4. Basic operations.

4.1. Addition and multiplication by a number. Arithmetic operations in the TT-format can be readily implemented. The addition of two tensors in the TT-format,

$$\mathbf{A} = A_1(i_1) \ldots A_d(i_d), \quad \mathbf{B} = B_1(i_1) \ldots B_d(i_d),$$

is reduced to the merge of cores, and for each mode, sizes of auxiliary dimensions are summed. The cores $C_k(i_k)$ of the sum $\mathbf{C} = \mathbf{A} + \mathbf{B}$ are defined as

$$C_k(i_k) = \begin{pmatrix} A_k(i_k) & 0 \\ 0 & B_k(i_k) \end{pmatrix}, \quad k = 2, \ldots, d-1,$$

and

$$C_1(i_1) = \begin{pmatrix} A_1(i_1) & B_1(i_1) \end{pmatrix}, \quad C_d(i_d) = \begin{pmatrix} A_d(i_d) \\ B_d(i_d) \end{pmatrix},$$

for border cores. Indeed, by direct multiplication,

$$C_1(i_1) C_2(i_2) \ldots C_d(i_d) = A_1(i_1) A_2(i_2) \ldots A_d(i_d) + B_1(i_1) B_2(i_2) \ldots B_d(i_d).$$

The multiplication by a number α is trivial; we just scale one of cores by it. The addition of two tensors is a good test for the rounding procedure. If we sum a vector t given in the TT-format with itself, the ranks are doubled, but the result should be

compressed to $2t$ with the same ranks as for t. In our experiments, such rounding was performed with an accuracy which is of the order of the machine precision. The addition of two vectors requires virtually no operations, but it increases the TT-ranks. If the addition has to be done many times, then rounding is needed. If the rounding is applied after each addition (to avoid rank growth), it costs $\mathcal{O}(nr^3d)$ operations for each addition. If an auxiliary Tucker decomposition of the tensor is used and only the core of the decomposition is in the TT-format, then the computational complexity of the rounding step is reduced to

$$\mathcal{O}(dnr^2 + dr^4).$$

4.2. Multidimensional contraction, Hadamard product, scalar product, and norm. In the TT-format many important operations can be implemented in a complexity linear in d. Consider the multidimensional contraction, i.e., evaluation of an expression of the form

$$W = \sum_{i_1,\dots,i_d} A(i_1,\dots,i_d)u_1(i_1)\dots u_d(i_d),$$

where $u_k(i_k)$ are vectors of length n_k. This is a scalar product of $\mathbf{A}$ with a canonical rank-1 tensor:

$$W = \langle \mathbf{A}, \otimes_{i=1}^{d} u_j \rangle.$$

Note that such summation appears when an integral of a multivariate function is computed via a tensor-product quadrature. In this case, the tensor $\mathbf{A}$ consists of function values on a tensor grid, and u_k are (one-dimensional) quadrature weights. Let $\mathbf{A}$ be in the TT-format,

$$\mathbf{A} = G_1(i_1)\dots G_d(i_d).$$

Then,

$$W = \left(\sum_{i_1} u_1(i_1)G_1(i_1)\right)\left(\sum_{i_2} u_2(i_2)G_2(i_2)\right)\dots\left(\sum_{i_d} u_d(i_d)G_d(i_d)\right).$$

Introduce matrices

$$\Gamma_k = \sum_{i_k} u_k(i_k)G_k(i_k).$$

The matrix Γ_k is an $r_{k-1} \times r_k$ matrix, and

$$W = \Gamma_1 \dots \Gamma_d.$$

Since Γ_1 is a row vector and Γ_d is a column vector, evaluating W reduces to the computation of matrices Γ_k and evaluating d matrix-by-vector products. The total number of arithmetic operations required is $\mathcal{O}(dnr^2)$. Again the Tucker format can be used to reduce the number of operations if $r < n$. The implementation is rather straightforward and requires

$$\mathcal{O}(dnr + dr^3)$$

operations for a single contraction. If we want to compute the elementwise (Hadamard) product of two tensors $\mathbf{A}$ and $\mathbf{B}$,

$$\mathbf{C} = \mathbf{A} \circ \mathbf{B},$$

i.e., elements of $\mathbf{C}$ are defined as

$$C(i_1, \dots, i_d) = A(i_1, \dots, i_d) B(i_1, \dots, B_d),$$

the result will be also in the TT-format, with TT-ranks of $\mathbf{A}$ and $\mathbf{B}$ multiplied. Indeed,

$$\begin{aligned} C(i_1, \dots, i_d) &= A_1(i_1) \dots A_d(i_d) B_1(i_1) \dots B_d(i_d) \\ &= \Big(A_1(i_1) \dots A_d(i_d)\Big) \otimes \Big(B_1(i_1) \dots B_d(i_d)\Big) \\ &= \Big(A_1(i_1) \otimes B_1(i_1)\Big)\Big(A_2(i_2) \otimes B_2(i_2)\Big) \dots \Big(A_d(i_d) \otimes B_d(i_d)\Big). \end{aligned}$$

This means that the cores of $\mathbf{C}$ are just

$$C_k(i_k) = A_k(i_k) \otimes B_k(i_k), \quad k = 1, \dots, d.$$

Using the Hadamard product, one can compute the scalar product of two tensors, which is important in many applications. For two tensors $\mathbf{A}, \mathbf{B}$ it is defined as

$$\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{i_1, \dots, i_d} A(i_1, \dots, i_d) B(i_1, \dots, i_d) = \sum_{i_1, \dots, i_d} C(i_1, \dots, i_d),$$

where $\mathbf{C} = \mathbf{A} \circ \mathbf{B}$. Thus, the scalar product can be computed by taking the Hadamard product and then by computing the contraction with vectors of all ones, i.e., $u_k(i_k) = 1$. The ranks of the product are $\mathcal{O}(r^2)$; thus the complexity is equal to $\mathcal{O}(dnr^4)$. However, it can be reduced. Recall that the computation of the contraction is reduced to the computation of the product

$$W = \Gamma_1 \dots \Gamma_d,$$

where in this case

$$\Gamma_k = \sum_{i_k} A_k(i_k) \otimes B_k(i_k).$$

Since Γ_1 is a row vector, W can be sequentially computed by a sequence of matrix-by-vector products:

$$v_k = v_{k-1} \Gamma_k, \quad k = 2, \dots, d, \quad v_1 = \Gamma_1.$$

Here v_k is a row vector of length $r_k^{(A)} r_k^{(B)}$. Consider the computation of v_k when v_{k-1} is known:

$$v_k = v_{k-1} \Gamma_k = v_{k-1} \sum_{i_k} A_k(i_k) \otimes B_k(i_k) = \sum_{i_k} p_k(i_k),$$

where

$$p_k(i_k) = v_{k-1} \Big(A_k(i_k) \otimes B_k(i_k)\Big)$$

is a vector of length $r_k^{(A)} r_k^{(B)}$. If all TT-ranks involved are of order r, then for each i_k the computation of $p_k(i_k)$ can be done in $\mathcal{O}(r^3)$ operations due to the special structure of the matrix $A_k(i_k) \otimes B_k(i_k)$; thus v_k can be computed in $\mathcal{O}(nr^3)$ operations, and the cost of the scalar product is $\mathcal{O}(dnr^3)$. If the Tucker format is used for both of the operands, the complexity is

$$\mathcal{O}(dnr^2 + dr^4).$$

Using the dot product, the Frobenius norm

$$||\mathbf{A}||_F = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle}$$

and the distance between two tensors

$$||\mathbf{A} - \mathbf{B}||_F$$

can be computed. Indeed, it is sufficient to subtract two tensors $\mathbf{A}$ and $\mathbf{B}$ (this would yield a tensor with TT-ranks equal to sum of TT-ranks of $\mathbf{A}$ and $\mathbf{B}$) and compute its norm. The complexity is also $\mathcal{O}(dnr^3)$ for the TT-format, and $\mathcal{O}(dnr^2 + dr^4)$ if the Tucker format is used. Algorithm 3 contains a formal description of how the multidimensional contraction is performed, and Algorithm 4 contains a formal description of how the dot product is computed in the TT-format.

Algorithm 3. Multidimensional contraction.

Require: Tensor $\mathbf{A}$ in the TT-format with cores A_k and vectors $u_1, \ldots, u_d$.
Ensure: $W = \mathbf{A} \times_1 u_1^\top \ldots \times_d u_d^\top$.

```
for k = 1 to d do
    Γ_k = Σ_{i_k} A_k(i_k) u_k(i_k).
end for
v := Γ_1.
for k = 2 to d do
    v := v Γ_k.
end for
W = v.
```

Algorithm 4. Dot product.

Require: Tensor $\mathbf{A}$ in the TT-format with cores A_k, and tensor $\mathbf{B}$ in the TT-format with cores B_k.
Ensure: $W = \langle \mathbf{A}, \mathbf{B} \rangle$.

```
v := Σ_{i_1} A_1(i_1) ⊗ B_1(i_1).
for k = 2 to d do
    p_k(i_k) = v(A_k(i_k) ⊗ B_k(i_k)).
    v := Σ_{i_k} p_k(i_k).
end for
W = v.
```

4.3. Matrix-by-vector product. The most important operation in linear algebra is probably the matrix-by-vector product. When both the matrix and the vector are given in the TT-format then the natural question is how to compute their product. When talking about "vector in the TT-format" we implicitly assume that a vector of

length $N = n_1 \dots n_d$ is treated as a d-dimensional tensor with mode sizes n_k, and this tensor is represented in the TT-format. Matrices acting on such vectors of length N should be of size $M \times N$; for simplicity assume that $M = N$. Elements of such matrices can be indexed by $2d$-tuples $(i_1, \dots, i_d, j_1, \dots, j_d)$, where $(i_1, \dots, i_d)$ enumerate the rows of M and $(j_1, \dots, j_d)$ enumerate its columns. A matrix M is said to be in the TT-format if its elements are defined as

$$M(i_1, \dots, i_d, j_1, \dots, j_d) = M_1(i_1, j_1) \dots M_d(i_d, j_d), \tag{4.1}$$

where $M_k(i_k, j_k)$ is an $r_{k-1} \times r_k$ matrix. i.e. (i_k, j_k) is treated as one "long index." Such permutation of dimensions is standard in the compression of high-dimensional operators [37, 17, 16]. It is motivated by the following observation, first mentioned in [38] for the two-dimensional case. If all TT-ranks are equal to 1, then M is represented as a Kronecker product of d matrices,

$$M = M_1 \otimes M_2 \otimes \dots \otimes M_d,$$

and that is a standard generalization of a rank-1 tensor to the matrix (operator) case [3, 4, 37]. Suppose now that we have a matrix M in the TT-format (4.1) and a vector x in the TT-format with TT-cores X_k and entries $X(j_1, \dots, j_d)$. The matrix-by-vector product in this situation is the computation of the following sum:

$$Y(i_1, \dots, i_d) = \sum_{j_1, \dots, j_d} M(i_1, \dots, i_d, j_1, \dots, j_d) X(j_1, \dots, j_d).$$

The resulting tensor will be also in the TT-format. Indeed,

$$\begin{aligned} Y(i_1, \dots, i_d) &= \sum_{j_1, \dots, j_d} M_1(i_1, j_1) \dots M_d(i_d, j_d) X_1(j_1) \dots X_d(j_d) \\ &= \sum_{j_1, \dots, j_d} \Big(M_1(i_1, j_1) \otimes X_1(j_1) \Big) \dots \Big(M_d(i_d, j_d) \otimes X_d(j_d) \Big) \\ &= Y_1(i_1) \dots Y_d(i_d), \end{aligned}$$

where

$$Y_k(i_k) = \sum_{j_k} \Big(M_k(i_k, j_k) \otimes X_k(j_k) \Big).$$

A formal description is presented in Algorithm 5.

Algorithm 5. Matrix-by-vector product.

Require: Matrix M in the TT-format with cores $M_k(i_k, j_k)$, and vector x in the TT-format with cores $X_k(j_k)$.
Ensure: Vector $y = Mx$ in the TT-format with cores Y_k.
  for $k = 1$ to d **do**
    $Y_k(i_k) = \sum_{j_k} (M_k(i_k, j_k) \otimes X_k(j_k))$.
  end for

The TT-ranks for $\mathbf{Y}$ are the product of ranks for the matrix and for the vector. The computation of Y_k can be realized as a matrix-by-matrix product. The summation over j_k is equivalent to the product of a matrix of size $r^2 n \times n$ (obtained from $M_k(i_k, j_k)$

by reshaping and permuting the dimensions) by a matrix of size $n \times r^2$ (obtained from $X_k(j_k)$). The complexity of such a matrix-by-matrix product is $\mathcal{O}(n^2r^4)$, and for the total matrix-by-vector product in the TT-format, $\mathcal{O}(dn^2r^4)$. However, almost every time one has to approximate the result afterwards to avoid rank growth. Application of the TT-rounding algorithm requires $\mathcal{O}(dnr^6)$ operations. If n is large, the Tucker format can be used to approximate both the matrix and the vector. The Tucker format for a matrix means that the matrix is first approximated as

$$M \approx \sum_{\alpha_1,\ldots,\alpha_d} G(\alpha_1,\ldots,\alpha_d)U_1(\alpha_1) \otimes U_2(\alpha_2) \ldots \otimes U_d(\alpha_d),$$

where $U_k(\alpha_k)$ is an $n \times n$ matrix, and the TT-decomposition is applied to the core $\mathbf{G}$; see [31], where a detailed description for the three-dimensional case is given. It can be shown that in this case the product can be performed in $\mathcal{O}(dn^2r^2 + dnr^4 + dr^8)$ operations. This gives reduced complexity if $n \geq Cr^2$ for some constant C. However, in practice, the $\mathcal{O}(r^8)$ term can be time consuming (starting from $r \geq 20$ on modern workstations). The situation is the same with the Tucker format, and several techniques have been proposed to evaluate the matrix-by-vector product quickly [13, 34, 31]. The idea is to avoid formation of the product in the TT-format exactly (which leads to huge ranks) but to combine multiplication and rounding in one step. Such techniques can be generalized from the Tucker case to the TT-case, and that is a topic of ongoing research. The expected complexity of this algorithm (with the assumption that the approximate TT-ranks of the product are also $\mathcal{O}(r)$), is $\mathcal{O}(dn^2r^4)$ if the Tucker decomposition is not used, and $\mathcal{O}(dn^2r^2 + dr^6)$ if it is used.

5. Numerical example. Consider the following operator:

$$H = \Delta_d + c_v \sum_i \cos(x - x_i) + c_w \sum_{i<j} \cos(x_i - x_j), \tag{5.1}$$

the one considered in [3, 4], where the canonical format was used. We have chosen the simplest possible discretization (3-point discretization of the one-dimensional Laplacian with zero boundary conditions on $[0, 1]$). After the discretization, we are left with an $n^d \times n^d$ matrix H and are looking for the minimal eigenvalue of H:

$$Hx = \lambda x, \quad ||x||_2 = 1, \quad \lambda \to \min.$$

First, the matrix H is approximated in the matrix TT-format (4.1),

$$H(i_1,\ldots,i_d,j_1,\ldots,j_d) = H_1(i_1,j_1)\ldots H_d(i_d,j_d).$$

This representation is obtained from the canonical representation of H, which is easy to get. As discussed before, Δ_d can be represented as a canonical rank-d tensor, and moreover, its TT-ranks are equal to 2. The same is true for the "one-particle" interaction $c_v \sum_i \cos(x - x_i)$, which becomes a Laplace-like tensor after the discretization. The two-particle term $c_w \sum_{i<j} \cos(x_i - x_j)$ gives TT-ranks not higher than 6. Indeed,

$$\cos(x_i - x_j) = \cos(x_i)\cos(x_j) + \sin(x_i)\sin(x_j),$$

and each summand, due to results of [3, 4], can be approximated by a rank-3 tensor to arbitrary precision. In our numerical experiments, we represent this term in the canonical format with $\frac{d(d+1)}{2}$ terms, and then convert to the TT-format. After the

TABLE 5.1
Results for the computation of the minimal eigenvalue.

n	8	16	32	64
$\lambda_{\min}$	$2.41 \cdot 10^3$	$2.51 \cdot 10^3$	$2.56 \cdot 10^3$	$2.58 \cdot 10^3$
δ	$6 \cdot 10^{-2}$	$2.7 \cdot 10^{-2}$	$7 \cdot 10^{-3}$	—
Average time for one iteration (sec)	$3 \cdot 10^{-2}$	$3.6 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$6.2 \cdot 10^{-2}$

transformation from the canonical representation to the TT-format, we get H in the TT-format with TT-ranks not higher than $2 + 2 + 6 = 10$. Now we have to find the minimal eigenpair of the matrix H. A starting guess has been chosen to be the eigenvector corresponding to the smallest in magnitude eigenvalue of the d-dimensional Laplace operator Δ_d. It is well known that it has the form

$$X(i_1, \ldots, i_d) = \sin \frac{\pi i_1}{n+1} \ldots \sin \frac{\pi i_d}{n+1};$$

i.e., it corresponds to a rank-1 tensor.

After that we applied a simple power iteration to the shifted matrix $\widehat{H} = cI - H$, where the shift c was chosen to make the smallest eigenvalue of H the largest in magnitude eigenvalue of the shifted matrix. It is easy to see that the identity matrix has the canonical rank 1; thus its TT-ranks are equal to 1 also, and the TT-ranks of $\widehat{H}$ are no more than $1 + 10 = 11$, and basic linear algebra in the TT-format can be used.

We have taken $d = 19$ and the one-dimensional grid sizes $n = 8, 16, 32, 64$; therefore the maximal mode size for the matrix has been $64^2 = 4096$. The matrix was compressed by the canonical-to-TT compression algorithm, and then the power iteration was applied. After each matrix-by-vector multiplication the TT-ranks of the approximate solution increase, and the rounding is performed. The final algorithm looks like

$$v := T_\varepsilon(Hv), \quad v = \frac{v}{||v||},$$

where $T_\varepsilon(Hv)$ is the result of the application of the TT-rounding algorithm to the vector Hv with the truncation parameter ε. This is surely not the best method for the computation of the smallest eigenpair; it was used just to test the rounding procedure and the matrix-by-vector subroutine. The parameters c_v, c_w in (5.1) were set to $c_v = 100$, $c_w = 5$. The computed eigenvalues for different n are given in Table 5.1. By "time for one iteration" we mean the total time required for the matrix-by-vector product and for the rounding with the parameter $\varepsilon = 10^{-6}$. δ is the estimated error of the eigenvalue of the operator (i.e., the model error), where for the exact eigenvalue we take the eigenvalue computed for the largest n (here it is $n = 64$). We can see that the eigenvalue stabilizes. To detect convergence of the iterative process for the discrete problem, we used the scaled residual, $||Ax - \lambda x||/|\lambda|$, and stopped when it was smaller than 10^{-5}. The number of iterations for the power method was of order $1000 - 2000$; we do not present it here. The TT-ranks for the solution were not higher than 4 in all cases. Note that for these small values of n timings, for one iteration grow very mildly when increasing n; it is interesting to explain the nature of this behavior. Table 5.1 shows the "internal convergence" of the method with increasing grid size. We can check that the computed structured vector is indeed an approximate eigenvector by looking at the residue. The problem of checking that it indeed delivers the smallest

eigenvalue (not some other eigenvalue) is difficult and is under investigation, as well as comparison with full tensor solves for small dimensions.

6. Conclusion and future work. We have presented a new format that can be used to approximate tensors. In some sense the TT-format (1.3) is just another form of writing the tree format of [32] or the subspace approach of [18, 15]: the same three-dimensional tensors as defining parameters, the same complexity estimates. However, the compact form of the TT-format gives a big advantage over the approaches mentioned above. It gives a clear way for a stable and fast rounding procedure. It is based entirely on a sequence of QR and SVD decompositions of matrices and does not require any recursion. Its implementation required only about 150 lines of MATLAB code,[5] compared to several thousands of code lines in C and Fortran for the recursive TT-format. It also allows fast and intuitive implementation of the basic linear algebra operations: matrix-by-vector multiplication, addition, dot product, and norm. We showed how to apply these subroutines to compute the smallest eigenvalue of a high-dimensional operator. This is a simplified model example, but it confirms that the TT-format can be used for the solution of high-dimensional problems efficiently.

There is great room for improvement and further development. Ideas presented in this paper, have already been applied to the solution of different problems: stochastic partial differential equations [25], high-dimensional elliptic equations [24], and elliptic equations with variable coefficients [11]. The ongoing work is to apply the TT-format for the solution of Schroedinger equation in quantum molecular dynamics, with preliminary experiments showing it is possible to treat Henon–Heiles potentials [28] with $d = 256$ degrees of freedom.

Acknowledgments. This paper is dedicated to the memory of my Grandfather, Bejaev Ivan Osipovich (1918–2010). I miss you.

I would like to thank all reviewers of this paper for their hard work, which motivated me a lot. I would like to thank the anonymous referee that provided the proof of the conjecture on the TT-ranks of the Scholes-like tensor. I would like to thank Dr. Venera Khoromskaia for proofreading the paper and for providing helpful suggestions on improving the manuscript.

REFERENCES

[1] I. Babuška, F. Nobile, and R. Tempone, *A stochastic collocation method for elliptic partial differential equations with random input data*, SIAM J. Numer. Anal., 45 (2007), pp. 1005–1034.

[2] I. Babuška, R. Tempone, and G. E. Zouraris, *Galerkin finite element approximations of stochastic elliptic partial differential equations*, SIAM J. Numer. Anal., 42 (2004), pp. 800–825.

[3] G. Beylkin and M. J. Mohlenkamp, *Numerical operator calculus in higher dimensions*, Proc. Natl. Acad. Sci. USA, 99 (2002), pp. 10246–10251.

[4] G. Beylkin and M. J. Mohlenkamp, *Algorithms for numerical analysis in high dimensions*, SIAM J. Sci. Comput., 26 (2005), pp. 2133–2159.

[5] R. Bro, *PARAFAC: Tutorial and applications*, Chemometrics Intell. Lab. Syst., 38 (1997), pp. 149–171.

[6] J. D. Carroll and J. J. Chang, *Analysis of individual differences in multidimensional scaling via n-way generalization of Eckart-Young decomposition*, Psychometrika, 35 (1970), pp. 283–319.

[5] MATLAB codes are available at http://spring.inm.ras.ru/osel.

[7] P. Comon, *Tensor decomposition: State of the art and applications*, in Mathematics in Signal Processing V, J. G. McWhirter and I. K. Proudler, eds., Oxford University Press, Oxford, UK, 2002.
[8] L. de Lathauwer, B. de Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278.
[9] L. de Lathauwer, B. de Moor, and J. Vandewalle, *On best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of high-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342.
[10] V. de Silva and L.-H. Lim, *Tensor rank and the ill-posedness of the best low-rank approximation problem*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1084–1127.
[11] S. V. Dolgov, B. N. Khoromskij, I. V. Oseledets, and E. E. Tyrtyshnikov, *Tensor Structured Iterative Solution of Elliptic Problems with Jumping Coefficients*, Preprint 55, Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany, 2010.
[12] M. Espig, *Effiziente Bestapproximation mittels Summen von Elementartensoren in hohen Dimensionen*, Ph.D. thesis, Fakultat fur Mathematik und Informatik, University of Leipzig, Leipzig, Germany, 2007.
[13] S. A. Goreinov, I. V. Oseledets, and D. V. Savostyanov, *Wedderburn Rank Reduction and Krylov Subspace Method for Tensor Approximation. Part* 1: *Tucker Case*, ArXiv preprint arXiv:1004.1986, 2010.
[14] L. Grasedyck, *Existence and computation of low Kronecker-rank approximations for large systems in tensor product structure*, Computing, 72 (2004), pp. 247–265.
[15] L. Grasedyck, *Hierarchical singular value decomposition of tensors*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2029–2054.
[16] 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 (2006), pp. 177–202.
[17] W. Hackbusch and B. N. Khoromskij, *Low-rank Kronecker-product approximation to multi-dimensional nonlocal operators.* II. *HKT representation of certain operators*, Computing, 76 (2006), pp. 203–225.
[18] W. Hackbusch and S. Kühn, *A new scheme for the tensor representation*, J. Fourier Anal. Appl., 15 (2009), pp. 706–722.
[19] R. A. Harshman, *Foundations of the Parafac procedure: Models and conditions for an explanatory multimodal factor analysis*, UCLA Working Papers in Phonetics, 16 (1970), pp. 1–84.
[20] J. Håstad, *Tensor rank is NP-complete*, J. Algorithms, 11 (1990), pp. 644–654.
[21] R. Hübener, V. Nebendahl, and W. Dür, *Concatenated tensor network states*, New J. Phys., 12 (2010), 025004.
[22] B. N. Khoromskij and V. Khoromskaia, *Multigrid accelerated tensor approximation of function related multidimensional arrays*, SIAM J. Sci. Comput., 31 (2009), pp. 3002–3026.
[23] 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 (2011), pp. 45–65.
[24] B. N. Khoromskij and I. V. Oseledets, *Quantics-TT Approximation of Elliptic Solution Operators in Higher Dimensions*, Preprint 79, MIS MPI, 2009.
[25] B. N. Khoromskij and I. V. Oseledets, *QTT-approximation of elliptic solution operators in high dimensions*, Rus. J. Numer. Anal. Math. Model, 26 (2011), pp. 303–322.
[26] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.
[27] C. Lubich, *From Quantum to Classical Molecular Dynamics: Reduced Models and Numerical Analysis*, EMS, Zurich, 2008.
[28] M. Nest and H.-D. Meyer, *Benchmark calculations on high-dimensional Henon-Heiles potentials with the multi-configuration time dependent Hartree (MCTDH) method*, J. Chem. Phys., 117 (2002), 10499.
[29] I. Oseledets and E. Tyrtyshnikov, *TT-cross approximation for multidimensional arrays*, Linear Algebra Appl., 432 (2010), pp. 70–88.
[30] I. V. Oseledets, D. V. Savostianov, and E. E. Tyrtyshnikov, *Tucker dimensionality reduction of three-dimensional arrays in linear time*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 939–956.
[31] I. V. Oseledets, D. V. Savostyanov, and E. E. Tyrtyshnikov, *Linear algebra for tensor problems*, Computing, 85 (2009), pp. 169–188.
[32] 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 (2009), pp. 3744–3759.

[33] J. Persson and L. von Persson, *Pricing European multi-asset options using a space-time adaptive fd-method*, Comput. Vis. Sci., 10 (2007), pp. 173–183.

[34] D. V. Savostyanov and E. E. Tyrtyshnikov, *Approximate multiplication of tensor matrices based on the individual filtering of factors*, Comput. Math. Math. Phys., 49 (2009), pp. 1662–1677.

[35] I. Sloan and H. Wozniakowski, *When are quasi-Monte Carlo algorithms efficient for high dimensional integrals*, J. Complexity, 14 (1998), pp. 1–33.

[36] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.

[37] E. E. Tyrtyshnikov, *Tensor approximations of matrices generated by asymptotically smooth functions*, Sb. Math., 194 (2003), pp. 941–954.

[38] C. F. Van Loan and N. Pitsianis, *Approximation with Kronecker products*, in Linear Algebra for Large Scale and Real-Time Applications (Leuven, 1992), NATO Adv. Sci. Inst. Ser. E Appl. Sci. 232, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993, pp. 293–314.

[39] C. F. Van Loan, *Tensor network computations in quantum chemistry*, Technical report, available online at www.cs.cornell.edu/cv/OtherPdf/ZeuthenCVL.pdf, 2008.

[40] O. Vendrell, F. Gatti, D. Lauvergnat, and H.-D. Meyer, *Full-dimensional (15-dimensional) quantum-dynamical simulation of the protonated water dimer. I. Hamiltonian setup and analysis of the ground vibrational state.*, J. Chem. Phys., 127 (2007), pp. 184302–184318.

[41] X. Wang and I. H. Sloan, *Why are high-dimensional finance problems often of low effective dimension?*, SIAM J. Sci. Comput., 27 (2005), pp. 159–183.