SIAM J. SCI. COMPUT.
Vol. 45, No. 1, pp. A74-A95

RANDOMIZED ALGORITHMS FOR ROUNDING IN THE TENSOR-TRAIN FORMAT*

HUSSAM AL DAAS†, GREY BALLARD‡, PAUL CAZEAUX§, ERIC HALLMAN¶, AGNIESZKA MIĘDLAR§, MIRJETA PASHA‖, TIM W. REID¶, AND ARVIND K. SAIBABA¶

Abstract. The tensor-train (TT) format is a highly compact low-rank representation for high-dimensional tensors. TT is particularly useful when representing approximations to the solutions of certain types of parametrized partial differential equations. For many of these problems, computing the solution explicitly would require an infeasible amount of memory and computational time. While the TT format makes these problems tractable, iterative techniques for solving the PDEs must be adapted to perform arithmetic while maintaining the implicit structure. The fundamental operation used to maintain feasible memory and computational time is called *rounding*, which truncates the internal ranks of a tensor already in TT format. We propose several randomized algorithms for this task that are generalizations of randomized low-rank matrix approximation algorithms and provide significant reduction in computation compared to deterministic TT-rounding algorithms. Randomization is particularly effective in the case of rounding a sum of TT-tensors (where we observe 20× speedup), which is the bottleneck computation in the adaptation of GMRES to vectors in TT format. We present the randomized algorithms and compare their empirical accuracy and computational time with deterministic alternatives.

Key words. high-dimensional problems, randomized algorithms, tensor decompositions, tensor-train format

MSC codes. 15A69, 65F55, 65F99, 65Y20, 68W20

DOI. 10.1137/21M1451191

1. Introduction.

An increasing number of applications in science and technology involve the manipulation of multidimensional data, or tensors that are higher order equivalents of vectors (first-order) and matrices (second-order). The number of elements of a tensor as well as the storage consumption grow exponentially with the number of dimensions, a phenomenon known as the *curse of dimensionality*. When problems of high dimensions are concerned, beating the curse of dimensionality and finding a solution efficiently remains a challenge. Nevertheless, different tensor formats and methods based on tensor products [25, 26, 32, 36, 45] have shown potential

* Submitted to the journal's Methods and Algorithms for Scientific Computing section October 11, 2021; accepted for publication (in revised form) September 6, 2022; published electronically January 27, 2023.

https://doi.org/10.1137/21M1451191

Funding: This work was supported by the NSF: CCF-1942892 (second author), DMS-1819220 (third author), DMS-1745654 (fourth and seventh authors), CCF-1812927 (fifth author), DMS-1502640 (sixth author), and DMS-1821149 (eighth author).

† Computational Mathematics Group, Rutherford Appleton Laboratory, Didcot OX11 0QX, UK (hussam.al-daas@stfc.ac.uk).

‡ Department of Computer Science, Wake Forest University, Winston-Salem, NC 27106 USA (ballard@wfu.edu).

§ Department of Mathematics, Virginia Tech, Blacksburg, VA 24061-1026 USA (cazeaux@vt.edu, amiedlar@vt.edu).

¶ Department of Mathematics, North Carolina State University, Raleigh, NC 27607 USA (erhallma@ncsu.edu, twreid@alumni.ncsu.edu, asaibab@ncsu.edu).

‖ Department of Mathematics, Tufts University, Medford, MA 02155 USA (mirjeta.pasha@tufts.edu).

for mitigating the curse of dimensionality and tackling high-dimensional problems that could not be addressed with conventional methods. Initially, the concept of tensor decompositions was introduced in 1927 by expressing a tensor as the sum of a finite number of rank-one tensors [29]—also known as the *canonical format*. The canonical format's memory requirements are not high, though it can suffer from numerical stability issues [15, 30]. Tensors in Tucker form [7] are well known in quantum chemistry [15, 30] since they yield robust algorithms due to the ability to form an embedded manifold [35], but one of the disadvantages of the Tucker format is its storage consumption that still depends exponentially on the number of dimensions.

One of the most promising tensor formats is the tensor-train (TT) format, a tensor product format that was initially proposed in quantum physics, also known as *matrix product states* [21], and was reinvented in numerical linear algebra [44, 46]. It combines both the advantages of the canonical and Tucker formats, i.e., (1) the storage consumption of a tensor depends linearly on the number of dimensions and (2) there exist robust algorithms for the computation of best approximations. Applications of the TT format arise from various applications such as high-dimensional PDEs like the Fokker–Planck equations [18, 49], quantum physics [40, 51], high-dimensional data analysis [33, 34], machine learning [8, 12, 20, 42], and uncertainty quantification [39, 53], to mention just a few. Typically, those applications require an approximate solution of linear systems of equations, eigenvalue problems, or completion problems [4, 23, 48]. The TT format is a low-rank representation that, for TT-tensors with small rank, offers a tremendous reduction in the computational complexity and often exposes the structure of the problem. The use of low-rank structures such as the TT format [44] to represent high-dimensional objects allows the solution of linear high-dimensional problems by generalizing standard numerical linear algebra techniques to multi-index arrays of coefficients (tensors) and the multivariate functions they approximate.

In this paper, we focus on the problem of rounding a tensor in TT format; that is, assuming that we are given a TT-tensor, we want to find a compressed representation that is nearly as accurate as the original representation. There are several techniques for computing the initial TT-tensors which do not require forming the entire tensor explicitly [16, 37, 43, 50]. One such technique that is popularly used is called the TT-cross approximation. The standard TT approach to rounding, proposed by Oseledets [44], has two phases [44, Algorithm 2]: orthogonalization followed by compression (typically using the SVD). Here, by orthogonalization, we mean a sweep of orthogonalization steps across every tensor core. Analysis shows that the orthogonalization step dominates the computational cost of this approach. Motivated by this observation, the goal of this work is to develop randomized algorithms for rounding TT-tensors that avoid expensive orthogonalization. In the following, we present the main contributions of this paper.

Overview of the paper and main contributions. This paper develops several new randomized algorithms for rounding tensors in the TT format and is organized as follows. In section 2, we set some notation as well as review some basic material on randomized matrix algorithms and standard TT operations along with a detailed analysis of their computational costs. In section 3, we propose various new randomized algorithms for TT-rounding with the focus on randomize-then-orthogonalize, two-sided-randomization, and rounding of a sum of TT-tensors:

1. In Algorithm 3.1, randomize-then-orthogonalize, we propose to form randomized sketches of each core by nested contractions with a TT-tensor with random cores in a first step, before performing the orthogonalization sweep

on these much smaller matrices. Our analysis and experiments show that this approach allowed for the best speedup compared to the deterministic algorithm while retaining excellent accuracy.

2. In Algorithm 3.2, two-sided-randomization, we completely eliminate the need for separate orthogonalization and compression sweeps. Instead, we work with a two-sided randomized approach which computes products with two random tensors followed by a compression step (which involves orthogonalization of much smaller matrices). Although this approach is slightly more expensive in terms of flops count and is less accurate than techniques mentioned before, it eliminates the need of extensive orthogonalization and allows for the truncation phase to be more independent and highly parallelizable.
3. We extend the randomize-then-orthogonalize approach for compressing a TT-tensor that is presented as a sum of TT-tensors (Algorithm 3.3). This special case is of importance in many applications such as solving parametric linear systems in the TT format. The use of randomization enables significant performance improvements by exploiting the structure of the sum tensor in a way that a deterministic algorithm cannot.
4. Additionally, in Algorithm SM2.1, orthogonalize-then-randomize, we replace the SVD step in the standard TT-rounding algorithm with a randomized SVD assuming that the truncated ranks are known a priori. This method, while not competitive to the other proposed approaches, serves as a point of comparison.

We provide an analysis of the computational cost of the proposed algorithms in subsection 3.4 and show that they are computationally more efficient than existing algorithms. We justify our analysis through numerical experiments in section 4 on both synthetic data and tensors generated while solving parametric partial differential equations (PDEs). Some conclusions and future outlook are presented in section 5. The MATLAB code for the implementation and numerical experiments is publicly available at https://github.com/SAMSI-RandTensors/randomizedTT.

Related work. There have been several recent developments in obtaining low-rank compression of tensors. We limit our literature review to the publications dealing with TT-tensors described in [44], which is closest to our work, and refer the reader to review papers for other developments in tensor decompositions [1, 10, 11, 25]. Oseledets [44] proposes a method for rounding TT-tensors. A parallel version of this method is introduced and developed in [13]. Our newly proposed approaches are more computationally efficient compared to existing deterministic algorithms. Other works [2, 9, 31] discuss randomized algorithms for compressing tensors in the TT format. These approaches differ from ours in that they require access to the entries of the tensor, i.e., they do not assume that the tensor is already in TT format. A recent paper [3] also uses randomization to produce a TT approximation of a full tensor but relies on tensor actions (i.e., applications of the tensor on $N-1$ vectors, where N is the order of the tensor). Other methods for constructing a low-rank compression in the TT format involve alternating least squares [19]. The use of tensor random projections in which the random tensors are taken to be in TT format have also been considered in [6, 22, 47]. While these papers use randomization in the context of TT-tensors, none of them directly address the problem of rounding, which is the central focus of our paper.

2. Background. Here, we review the notation and necessary operations involving tensors in a modest amount of detail. For a more comprehensive exposition we refer the reader to [13, 36, 44].

2.1. Notation. We denote tensors by boldface script letters (e.g., $\boldsymbol{\mathcal{X}}$) and matrices by boldface Roman letters (e.g., $\mathbf{A}$). We follow MATLAB-like convention and denote the entries of a three-way tensor $\boldsymbol{\mathcal{X}}$ as $\boldsymbol{\mathcal{X}}(i,j,k)$. A colon denotes the entire range of indices in that dimension. We denote the column fibers as $\boldsymbol{\mathcal{X}}(:,j,k)$, row fibers as $\boldsymbol{\mathcal{X}}(j,:,k)$, and tube fibers as $\boldsymbol{\mathcal{X}}(j,k,:)$. The *mode-n unfolding* (or matricization) of the tensor $\boldsymbol{\mathcal{X}}$ is denoted as $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times (I/I_n)}$, where $I = I_1 I_2 \cdots I_n$. The columns of the mode-n unfolding are composed of the appropriate mode-n fibers, e.g., the columns of mode-1 unfolding are column fibers and the columns of mode-3 unfolding are tube fibers. Given a matrix $\mathbf{A} \in \mathbb{R}^{M \times I_n}$, the mode-$n$ product $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_n \mathbf{A}$ is defined by its mode-n unfolding $\mathbf{Y}_{(n)} = \mathbf{A}\mathbf{X}_{(n)}$. The norm of a tensor is equivalent to the Frobenius norm of any of its unfoldings: $\|\boldsymbol{\mathcal{X}}\| = \|\mathbf{X}_{(n)}\|_F$.

An *order-N tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is in the TT format if there exist positive integers $R_0, \ldots, R_N$ with $R_0 = R_N = 1$ and order-3 tensors $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},1}, \ldots, \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},N}$, called *TT-cores*, with $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ for $1 \le n \le N$, such that

$$\boldsymbol{\mathcal{X}}(i_1, \ldots, i_N) = \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},1}(i_1,:) \cdot \cdots \cdot \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n}(:,i_n,:) \cdot \cdots \cdot \boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},N}(:,i_N),$$

where $1 \le i_n \le I_n$. Note that because $R_0 = R_N = 1$, the first and last TT-cores are (order-2) matrices so $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},1}(i_1,:) \in \mathbb{R}^{R_1}$ and $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},N}(:,i_N) \in \mathbb{R}^{R_{N-1}}$. The $R_{n-1} \times R_n$ matrix $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n}(:,i_n,:)$ is referred to as the i_nth slice of the nth TT-core of $\boldsymbol{\mathcal{X}}$. It is worth mentioning that the TT decomposition is not unique due to the multiplicative nature of the format.

In order to express the arithmetic operations on TT-cores using linear algebra, we will often use two specific matrix unfoldings of the order-3 tensors. The *horizontal unfolding* of a TT-core $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n}$ corresponds to the concatenation of the slices $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n}(:,i_n,:)$ for $i_n = 1, \ldots, I_n$ horizontally. We denote the corresponding operator by $\mathcal{H}$, so that $\mathcal{H}(\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n})$ is an $R_{n-1} \times I_n R_n$ matrix. The *vertical unfolding* of a TT-core $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n}$ corresponds to the concatenation of the slices $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n}(:,i_n,:)$ for $i_n = 1, \ldots, I_n$ vertically. We denote the corresponding operator by $\mathcal{V}$, so that $\mathcal{V}(\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n})$ is an $R_{n-1} I_n \times R_n$ matrix; see Figure 2.1. Moreover, we will often make use of a *tensor network diagram* (see Figure 2.2) to graphically illustrate TT-tensor operations. Here nodes represent tensors and edges represent modes so that connected nodes can be contracted.

Let $\mathbf{X}_{(1:n)} \in \mathbb{R}^{(I_1 I_2 \cdots I_n) \times (I_{n+1} \cdots I_N)}$ denote an unfolding of the first n modes of a TT-tensor $\boldsymbol{\mathcal{X}}$. It has the rank R_n representation

$$\mathbf{X}_{(1:n)} = \mathcal{V}(\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},1:n}) \mathcal{H}(\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n+1:N}),$$

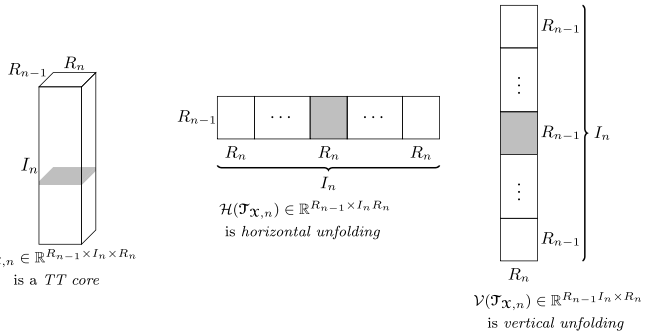

FIG 2.1. *Horizontal and vertical unfoldings of a TT-core* $\boldsymbol{\mathcal{T}}_{\boldsymbol{\mathcal{X}},n}$.

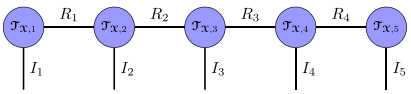

FIG 2.2. *Tensor network diagram for an order-5 TT-tensor.*

where in an extension of their earlier definitions, $\mathcal{V}(\mathcal{T}_{\mathcal{X},1:n}) \in \mathbb{R}^{(I_1 I_2 \cdots I_n) \times R_n}$ represents the mode-$(n+1)$ unfolding of the product of the first n TT-cores and $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1:N}) \in \mathbb{R}^{R_n \times (I_{n+1} \cdots I_N)}$ represents the mode-1 unfolding of the product of the final $N-n$ TT-cores. Likewise, we can write the same unfolding as a product of four matrices (see [13, eq. (2.3)]), i.e.,

$$\mathbf{X}_{(1:n)} = (\mathbf{I}_{I_n} \otimes \mathcal{V}(\mathcal{T}_{\mathcal{X},1:n-1}))\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})(\mathcal{H}(\mathcal{T}_{\mathcal{X},n+2:N}) \otimes \mathbf{I}_{I_{n+1}}). \tag{2.1}$$

Suppose we have two tensors $\mathcal{Y}$ and $\mathcal{Z}$ of the same dimension, and consider their sum $\mathcal{X}$. The cores of the tensor $\mathcal{X}$ can be expressed as

$$\mathcal{T}_{\mathcal{X},n}(:, i_n, :) = \begin{bmatrix} \mathcal{T}_{\mathcal{Y},n}(:, i_n, :) & \\ & \mathcal{T}_{\mathcal{Z},n}(:, i_n, :) \end{bmatrix}, \qquad 2 \le n \le N-1,$$

and for the first and the last core, we have

$$\mathcal{T}_{\mathcal{X},1}(i_1, :) = \begin{bmatrix} \mathcal{T}_{\mathcal{Y},1}(i_1, :) & \mathcal{T}_{\mathcal{Z},1}(i_1, :) \end{bmatrix} \quad \text{and} \quad \mathcal{T}_{\mathcal{X},N}(:, i_N) = \begin{bmatrix} \mathcal{T}_{\mathcal{Y},N}(:, i_N) \\ \mathcal{T}_{\mathcal{Z},N}(:, i_N) \end{bmatrix}.$$

Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ with $m \ge n$. We denote the *thin QR* factorization of $\mathbf{X}$ as $\mathbf{X} = \mathbf{QR}$, where $\mathbf{Q} \in \mathbb{R}^{m \times n}$ has orthonormal columns and $\mathbf{R} \in \mathbb{R}^{n \times n}$ is upper triangular; we also write $[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathbf{X})$ for use in algorithms. The SVD of $\mathbf{X}$ is denoted by $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where the matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$ has orthonormal columns containing the left singular vectors, $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with the singular values on the diagonal, and $\mathbf{V} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix, whose columns contain the right singular vectors. Assuming that the Householder QR algorithm is used and $\mathbf{Q}$ is formed explicitly, the computational cost of the QR factorization is $4mn^2 - \frac{4n^3}{3} + \mathcal{O}(n^2)$ flops. Given a threshold $\varepsilon > 0$, we truncate the singular values of $\mathbf{X}$ to obtain a rank-k approximation $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top$ of matrix $\mathbf{X}$, which satisfies $\|\mathbf{X} - \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top\|_F \le \varepsilon_{\mathrm{SVD}}\|\mathbf{X}\|_F$. This is denoted as $[\mathbf{U}_k, \mathbf{\Sigma}_k, \mathbf{V}_k] = \mathrm{SVD}(\mathbf{X}, \varepsilon_{\mathrm{SVD}})$. The computational cost of computing the SVD is $\mathcal{O}(mn^2)$ flops.

2.2. Randomized matrix algorithms. An important component of our approach is the use of randomized matrix methods for low-rank matrix approximation. In this subsection, we briefly review a few well-established randomized algorithms.

The first algorithm is the basic version of the randomized SVD proposed in [27]. Suppose we want to compute a low-rank approximation of a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$; let the ℓ denote the number of samples which is a sum of the target rank and a small oversampling parameter, such that $\ell \le \min\{m, n\}$. We generate a random matrix $\mathbf{\Omega} \in \mathbb{R}^{n \times \ell}$; in practice, we take the entries of this matrix to be independent and identically distributed standard Gaussian random variables. Then, we compute the product $\mathbf{Y} = \mathbf{X}\mathbf{\Omega}$ and obtain its thin QR factorization $\mathbf{Y} = \mathbf{QR}$. The main insight exploited by randomized SVD is that if the rank of $\mathbf{X}$ is close to r, or the singular values of $\mathbf{X}$ decay rapidly beyond r, then the range of $\mathbf{Q}$ approximates well the range

of $\mathbf{X}$ in the sense that $\mathbf{X} \approx \mathbf{Q}\mathbf{Q}^\top\mathbf{X}$; we then use $\mathbf{Q}\mathbf{Q}^\top\mathbf{X}$ as a low-rank approximation to $\mathbf{X}$. The computational cost of this approach is

$$C_{\text{randSVD}} = 2\ell mn + \mathcal{O}(\ell^2(m+n)) \quad \text{flops.}$$

Additional postprocessing can be performed to convert the low-rank approximation in the SVD format, or to truncate the low-rank approximation to the desired target rank; see [27] for additional details.

There is one variant of this algorithm that is of particular importance to our newly proposed methods: the generalized Nyström method [41]. The generalized Nyström method avoids the orthogonalization step when computing a low-rank approximation by using a two-sided randomized approach. Let us define two Gaussian random matrices $\mathbf{\Omega} \in \mathbb{R}^{n\times\rho}$ and $\mathbf{\Psi} \in \mathbb{R}^{t\times m}$, where $r \le \rho \le \min\{m,n\}$ (note that t also satisfies a similar inequality). A low-rank approximation to $\mathbf{X}$ is computed as

$$\mathbf{X} \approx \mathbf{Y}(\mathbf{\Psi}\mathbf{X}\mathbf{\Omega})^\dagger\mathbf{Z}, \tag{2.2}$$

where $\mathbf{Y} = \mathbf{X}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{\Psi}\mathbf{X}$. To implement the pseudoinverse, [41] suggests computing the QR factorization $\mathbf{\Psi}\mathbf{X}\mathbf{\Omega} = \mathbf{Q}\mathbf{R}$ and then obtaining the low-rank approximation $(\mathbf{Y}\mathbf{R}^{-1})(\mathbf{Q}^\top\mathbf{Z})$. If the low-rank approximation is desired in the SVD format, this can be done by additional postprocessing. In [41], the author recommends setting the sketch parameters as $\rho = r$ and $t = \lceil 1.5r \rceil$. The associated computational cost is

$$C_{\text{genNys}} = 2mn(\rho + t) + \mathcal{O}(t^2(m+n) + t\rho^2) \quad \text{flops.}$$

2.3. Standard TT arithmetic. In this subsection, we review the standard approach to TT-rounding, first proposed in [44], using the notation of [13]. We also review the concepts of tensor contractions.

To explain the rounding procedure for TT-tensors, we consider the following analogy from matrices. Let $\mathbf{Y} = \mathbf{A}\mathbf{B}$ be an outer product matrix where $\mathbf{A}$ is $m \times r$ and $\mathbf{B}$ is $r \times n$ and $r \le \min\{m,n\}$. To obtain an approximation of $\mathbf{Y}$ with rank $\ell < r$, we employ an orthogonalization step followed by a compression step. In the orthogonalization step, we want to make $\mathbf{Y}$ right orthogonal. That is, we compute the thin QR factorization $\mathbf{B}^\top = \mathbf{Q}\mathbf{R}$, and then compute $\mathbf{Z} = \mathbf{A}\mathbf{R}^\top$. This gives $\mathbf{Y} = \mathbf{A}\mathbf{B} = \mathbf{Z}\mathbf{Q}^\top$, where $\mathbf{Q}^\top$ has orthonormal rows. In the second step, we compress $\mathbf{Z}$ by computing the rank-ℓ truncated SVD $\mathbf{Z} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}_Z^\top$. To obtain an overall low-rank approximation to $\mathbf{Y}$, we compute $\mathbf{V} = \mathbf{Q}\mathbf{V}_Z$, so that $\mathbf{Y} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$.

Following [13], we say a tensor is *right orthogonal* if its horizontal unfoldings $\mathcal{H}(\mathcal{T}_{\mathcal{X},n})$ have orthonormal rows for $n = 2,\dots,N$ (all except the first core). Similarly, we say that a tensor is *left orthogonal* if its vertical unfoldings $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ have orthonormal columns for $n = 1,\dots,N-1$ (all except the last core).

Right-to-left orthogonalization. Suppose we are given a TT-tensor $\mathcal{Y}$. To obtain a right-orthogonal TT-tensor $\mathcal{X}$ equivalent to $\mathcal{Y}$, we first compute the thin QR factorization $\mathbf{Q}\mathbf{R} = \mathcal{H}(\mathcal{T}_{\mathcal{Y},N})^\top$ and set the core tensors $\mathcal{T}_{\mathcal{X},N-1}$ and $\mathcal{T}_{\mathcal{X},N}$ as

$$\mathcal{V}(\mathcal{T}_{\mathcal{Y},N-1})\mathcal{H}(\mathcal{T}_{\mathcal{Y},N}) = \mathcal{V}(\mathcal{T}_{\mathcal{Y},N-1})(\mathbf{Q}\mathbf{R})^\top = \underbrace{(\mathcal{V}(\mathcal{T}_{\mathcal{Y},N-1})\mathbf{R}^\top)}_{\mathcal{V}(\mathcal{T}_{\mathcal{X},N-1})}\ \underbrace{(\mathbf{Q}^\top)}_{\mathcal{H}(\mathcal{T}_{\mathcal{X},N})}.$$

Algorithm 2.1 Right-to-left orthogonalization

Require: A tensor $\mathcal{Y}$ in TT format
Ensure: $\mathcal{X}$ is a right-orthogonal tensor equivalent to $\mathcal{Y}$

function $\mathcal{X} = \text{ORTHOGONALIZERL}(\mathcal{Y})$
  $\mathcal{T}_{\mathcal{X},N} = \mathcal{T}_{\mathcal{Y},N}$
  for $n = N$ down to 2 **do**
    $[\mathcal{H}(\mathcal{T}_{\mathcal{X},n})^\top, \mathbf{R}] = \text{QR}(\mathcal{H}(\mathcal{T}_{\mathcal{X},n})^\top)$ ▷ thin QR factorization
    $\mathcal{V}(\mathcal{T}_{\mathcal{X},n-1}) = \mathcal{V}(\mathcal{T}_{\mathcal{Y},n-1}) \cdot \mathbf{R}^\top$ ▷ $\mathcal{T}_{\mathcal{X},n-1} = \mathcal{T}_{\mathcal{Y},n-1} \times_3 \mathbf{R}^\top$
  end for
end function

Algorithm 2.2 TT-rounding

Require: A tensor $\mathcal{Y}$ in TT format, user-defined threshold $\varepsilon_0 > 0$
Ensure: A tensor $\mathcal{X}$ in TT format with reduced ranks such that $\|\mathcal{X} - \mathcal{Y}\| \le \varepsilon_0 \|\mathcal{Y}\|$

function $\mathcal{X} = \text{TT-ROUNDING}(\mathcal{Y}, \varepsilon_0)$
  $\mathcal{X} = \text{ORTHOGONALIZERL}(\mathcal{Y})$
  Compute $\|\mathcal{Y}\|_F$ and the truncation threshold $\varepsilon_{\text{TT}} = \frac{\|\mathcal{Y}\|}{\sqrt{N-1}}\varepsilon_0$
  Set $\mathcal{T}_{\mathcal{X},1} = \mathcal{T}_{\mathcal{Y},1}$.
  for $n = 1$ to $N - 1$ **do**
    $[\mathcal{V}(\mathcal{T}_{\mathcal{X},n}), \mathbf{R}] = \text{QR}(\mathcal{V}(\mathcal{T}_{\mathcal{X},n}))$ ▷ thin QR factorization
    $[\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}}, \widehat{\mathbf{V}}] = \text{SVD}(\mathbf{R}, \varepsilon_{\text{TT}})$ ▷ ε_{TT}-truncated SVD factorization
    $\mathcal{V}(\mathcal{T}_{\mathcal{X},n}) = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})\widehat{\mathbf{U}}$ ▷ $\mathcal{T}_{\mathcal{X},n} = \mathcal{T}_{\mathcal{X},n} \times_3 \widehat{\mathbf{U}}$
    $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1}) = \widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top \mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})$ ▷ $\mathcal{T}_{\mathcal{X},n+1} = \mathcal{T}_{\mathcal{X},n+1} \times_1 (\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top)$
  end for
end function

This procedure is continued through cores $N-1, \ldots, 2$ but we do not orthogonalize the first core. The details of right-to-left orthogonalization are given in Algorithm 2.1, which will form the foundation for many of the subsequent algorithms. We can similarly obtain a left orthogonal tensor by processing the modes starting from mode-1, but we omit the details here.

TT-rounding. Suppose, now, that we want to round the tensor $\mathcal{Y}$ in the TT format, i.e., compress the TT format of a tensor by decreasing the TT-ranks $\{R_n\}$. In the first step of the TT-rounding approach, we first obtain a tensor $\mathcal{X}$ that is right orthogonal by applying Algorithm 2.1. Starting with mode-1, for each mode, we compute a low-rank approximation of the vertical unfolding $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$; rather than computing an SVD directly, we first compute the thin QR factorization of $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$, followed by an SVD of the upper triangular factor $\mathbf{R}$. We then obtain a low-rank approximation $\mathcal{V}(\mathcal{T}_{\mathcal{X},n}) \approx \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$. The number of singular values and vectors retained in the low-rank approximation depends on the threshold $\varepsilon_{\text{TT}} = \frac{\|\mathcal{Y}\|}{\sqrt{N-1}}\varepsilon_0$, where ε_0 is a user-defined threshold that controls the overall accuracy. We then rewrite $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ by combining it with the low-rank factor as $\mathcal{V}(\mathcal{T}_{\mathcal{X},n}) = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})\widehat{\mathbf{U}}$. The other two factors $\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$ are combined with the horizontal unfolding $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})$ for processing at the next step. This process is terminated after $N-1$ steps and the resulting tensor $\mathcal{X}$ satisfies $\|\mathcal{X} - \mathcal{Y}\| \le \varepsilon_0 \|\mathcal{Y}\|$. The details are given in Algorithm 2.2.

Algorithm 2.3 Right-to-left contraction of tensors $\mathcal{X}$ and $\mathcal{Y}$.

Require: Tensors $\mathcal{X}, \mathcal{Y}$ with consistent dimensions in TT format and ranks $\{R_n^{\mathcal{X}}\}$ and $\{R_n^{\mathcal{Y}}\}$, respectively.

Ensure: Matrices $\{\mathbf{W}_n\}$ satisfy $\mathbf{W}_n = \mathcal{H}(\mathcal{T}_{\mathcal{X},n+1:N})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})^\top$ for $1 \le n < N$

function $[\{\mathbf{W}_n\}] =$ PartialContractionsRL$(\mathcal{X}, \mathcal{Y})$
  $\mathbf{W}_{N-1} = \mathcal{H}(\mathcal{T}_{\mathcal{X},N})\mathcal{H}(\mathcal{T}_{\mathcal{Y},N})^\top$
  for $n = N-1$ down to 2 **do**
    $\mathcal{V}(\mathcal{T}_{\mathcal{Z},n}) = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathbf{W}_n$ ▷ $\mathcal{T}_{\mathcal{Z},n} = \mathcal{T}_{\mathcal{X},n} \times_3 \mathbf{W}_n$, for temporary $\mathcal{T}_{\mathcal{Z},n}$
    $\mathbf{W}_{n-1} = \mathcal{H}(\mathcal{T}_{\mathcal{Z},n})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n})^\top$ ▷ matrix multiplication, $\mathbf{W}_{n-1}$ is $R_{n-1}^{\mathcal{X}} \times R_{n-1}^{\mathcal{Y}}$
  end for
end function

Right-to-left partial contraction. We consider two TT-tensors $\mathcal{X}$ and $\mathcal{Y}$ with ranks $\{R_j^{\mathcal{X}}\}$ and $\{R_j^{\mathcal{Y}}\}$, respectively. For $n = 2, \dots, N$ we define the partial contraction matrices

$$\mathbf{W}_{n-1} = \mathcal{H}(\mathcal{T}_{\mathcal{X},n:N})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n:N})^\top \in \mathbb{R}^{R_{n-1}^{\mathcal{X}} \times R_{n-1}^{\mathcal{Y}}}. \tag{2.3}$$

These partial contractions can be computed sequentially as

$$\begin{aligned} \mathcal{V}(\mathcal{T}_{\mathcal{Z},n}) &= \mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathbf{W}_n, \\ \mathbf{W}_{n-1} &= \mathcal{H}(\mathcal{T}_{\mathcal{Z},n})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n})^\top \end{aligned} \tag{2.4}$$

for $n = 2, \dots, N-1$, with $\mathbf{W}_{N-1} = \mathcal{H}(\mathcal{T}_{\mathcal{X},N})\mathcal{H}(\mathcal{T}_{\mathcal{Y},N})^\top$. Here $\mathcal{Z}$ is a temporary TT-tensor with compatible dimensions and ranks.

The process of computing the matrices $\{\mathbf{W}_{n-1}\}_{n=2}^N$ according to (2.4) is called a *right-to-left partial contraction* of tensors $\mathcal{X}$ and $\mathcal{Y}$ and is illustrated in Figure 2.3. The corresponding algorithm is presented in Algorithm 2.3.

Detailed analysis of the overall computational costs of *right-to-left orthogonalization* (Algorithm 2.1), *TT-rounding* (Algorithm 2.2), and r*ight-to-left contraction* (Algorithm 2.3) is presented in section SM1.

3. Randomized algorithms for TT-rounding. In this section, we propose three new randomized algorithms to perform rounding of a tensor in the TT format, i.e., given an original TT-tensor $\mathcal{Y}$ with TT-ranks $\{R_n\}$ we seek a compressed TT-tensor representation $\mathcal{X}$ with a priori known target ranks $\{\ell_n\}$. In randomized SVD (see subsection 2.2), it is common to include an oversampling term; that is, if we seek a rank-r decomposition of a matrix $\mathbf{X}$, we use the number of samples (alternatively, columns of $\mathbf{\Omega}$) as $\ell = r + p$, where r is the target rank and p is the oversampling

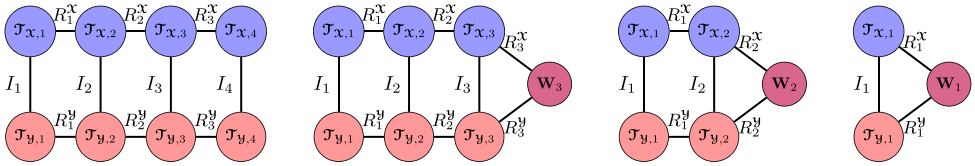

FIG 2.3. *Right-to-left partial contraction steps for $N = 4$.*

parameter. The resulting low-rank approximation $\mathbf{Q}\mathbf{Q}^\top\mathbf{X}$ is of rank ℓ. However, in the TT case, to save on notation, when we say target TT-ranks $\{\ell_n\}$, we assume that this rank automatically includes the necessary oversampling parameter.

Before we present the main algorithms, we describe a naive application of randomized SVD to rounding. The first algorithm we propose, *orthogonalize-then-randomize*, is very similar to the standard TT-rounding algorithm; the main difference is that we replace the truncated SVD step in Algorithm 2.2 with the basic version of the randomized SVD reviewed in subsection 2.2. The nomenclature of this algorithm is clear from the fact that there are two phases in this approach: an (already discussed) orthogonalization phase followed by a compression phase which utilizes randomized SVD. As we show in the analysis of the computational costs, subsection 3.4, and the numerical experiments, this algorithm is expensive and the costs are dominated by the first, i.e., orthogonalization, phase of the algorithm. Because of this, the details of this approach are relegated to the supplementary materials, section SM2.

3.1. Randomize-then-orthogonalize. First, we consider a new randomize-then-orthogonalize algorithm that uses randomization to reduce the overall computation cost of the TT-rounding procedure. It works by avoiding an expensive orthogonalization of the original TT-tensor $\mathcal{Y}$ with TT-ranks $\{R_n\}$ and instead uses randomization to reduce the computational cost. In contrast to the next approach in subsection 3.2 (two-sided-randomization), here we use randomization only on one side.

We first offer a way to construct random Gaussian TT-tensors whose cores are composed of independent random Gaussian entries.

DEFINITION 3.1 (random Gaussian TT-tensor). *Given a set of target TT-ranks $\{\ell_n\}$, we generate a random Gaussian TT-tensor $\mathcal{R} \in \mathbb{R}^{I_1\times\cdots I_N}$ such that each core tensor $\mathcal{T}_{\mathcal{R},n} \in \mathbb{R}^{\ell_{n-1}\times I_n\times\ell_n}$ is filled with random, independent, normally distributed entries with mean 0 and variance $1/(\ell_{n-1}I_n\ell_n)$ for $1 \le n \le N$.*

By this definition, while the cores of $\mathcal{R}$ have independent entries, the entries of the full tensor themselves are not independent. This normalization is chosen such that $\mathbb{E}\|\mathcal{T}_{\mathcal{R},n}\|_F^2 = 1$ and is sometimes necessary to ensure that no overflow occurs during the rounding computations. Note that constructing this random tensor requires only generating and storing $\sum_{n=1}^{N}\ell_{n-1}\ell_n I_n$ random entries. A related but distinct definition for a Gaussian TT-tensor is given in [47], but the approach taken here differs considerably in how we use the randomized tensor.

In Algorithm 3.1, we first generate a random Gaussian tensor $\mathcal{R}$ with given target TT-ranks $\{\ell_n\}$ following Definition 3.1. Next, we use the efficient multiplication of tensor $\mathcal{R}$ with a given tensor $\mathcal{Y}$ (see Algorithm 2.3) to obtain the sketches (sometimes also referred to as partial random projections) $\{\mathbf{W}_n\}$ of $\mathcal{Y}$ (*randomization phase*). A visualization of this process is provided in Figure 3.1. Finally, we construct a left-orthogonal compressed TT-tensor $\mathcal{X}$. Starting with $n = 1$ and $\mathcal{T}_{\mathcal{X},1} = \mathcal{T}_{\mathcal{Y},1}$, we compute the QR factorization of the sketched matrix

$$[\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})]\,\mathcal{H}(\mathcal{T}_{\mathcal{R},n+1:N})^\top = \mathbf{Q}_n\mathbf{R}_n. \tag{3.1}$$

The equation above is analogous to sketching a factored matrix $\mathbf{Y} = \mathbf{A}\mathbf{B}$ by computing the QR factorization of $\mathbf{A}\mathbf{B}\mathbf{\Omega}$. We emphasize that $\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})$ and $\mathcal{H}(\mathcal{T}_{\mathcal{R},n+1:N})$ should not be formed explicitly; instead, we compute the contraction

$$\mathbf{W}_n = \mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})\mathcal{H}(\mathcal{T}_{\mathcal{R},n+1:N})^\top$$

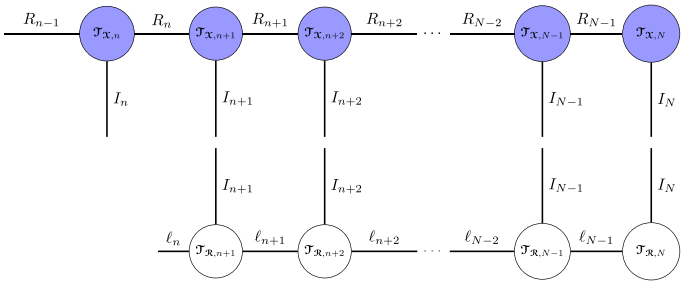

FIG 3.1. *Random projection for randomize-then-orthogonalize, Algorithm* 3.1.

efficiently using the process outlined in (2.4), then find the QR factorization of the small matrix

$$\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathbf{W}_n = \mathbf{Q}_n\mathbf{R}_n.$$

It can be seen via (2.4) that to compute $\mathbf{W}_n$ we must first find $\mathbf{W}_{n+1}, \ldots, \mathbf{W}_{N-1}$, so in order to avoid redundant computation we use Algorithm 2.3 to obtain the partial contractions $\{\mathbf{W}_n\}_{n=1}^{N-1}$ once, then store and reuse them.

Since at the nth step the first $n-1$ cores of $\mathcal{X}$ are already orthogonalized, they do not need to be considered explicitly in the factorization (3.1). By projecting $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})$ onto the column space of $\mathbf{Q}_n$, we approximate the product of the final $N-n+1$ cores as

$$\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N}) \approx \mathbf{Q}_n\mathbf{Q}_n^\top\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N}) = \mathbf{Q}_n\mathbf{M}_n\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N}).$$

Then, the cores are updated, i.e., $\mathcal{V}(\mathcal{T}_{\mathcal{X},n}) = \mathbf{Q}_n$ and $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1}) = \mathbf{M}_n\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1})$. It is important to mention that the randomize-then-orthogonalize approach produces a left-orthogonal tensor $\mathcal{X}$. We can use this observation to compress the tensor further. Therefore, if the ranks are not known a priori, then we choose the ranks to be sufficiently large and truncate them further by using Algorithm 2.2. In particular, since the output tensor of Algorithm 3.1 is left orthogonal, we can skip the orthogonalization phase (line 2 of Algorithm 2.2) and execute lines 3 to 10. This is what we do in our numerical experiments when the rank is not known a *priori*.

3.2. Two-sided-randomization. Analogous to the one-sided Algorithm 3.1, we start with generating two TT random Gaussian tensors $\mathcal{L}$ and $\mathcal{R}$ with given target TT-ranks $\{\ell_n\}$ and $\{\rho_n\}$ (with $\rho_n > \ell_n$) and computing the sketches $\{\mathbf{W}_n^L\}$, $\{\mathbf{W}_n^R\}$ of $\mathcal{Y}$ from the left and right, respectively (*randomization phase*; see Figure SM2). Next, for each $n = 1, \ldots, N-1$ we compute the SVD of a product of partial contractions $\mathbf{W}_n^L\mathbf{W}_n^R$, i.e., $\mathbf{W}_n^L\mathbf{W}_n^R = \mathbf{U}_n\mathbf{\Sigma}_n\mathbf{V}_n^\top$, and form left and right factor matrices

$$\mathbf{L}_n = \mathbf{W}_n^R\mathbf{V}_n(\mathbf{\Sigma}_n^\dagger)^{1/2} \quad \text{and} \quad \mathbf{R}_n = (\mathbf{\Sigma}_n^\dagger)^{1/2}\mathbf{U}_n^\top\mathbf{W}_n^L. \tag{3.2}$$

In order to highlight the significance of matrices $\mathbf{L}_n$ and $\mathbf{R}_n$, we consider the following unfolding of the TT-tensor $\mathcal{Y}$:

$$\mathbf{Y}_{(1:n)} = \mathcal{V}(\mathcal{T}_{\mathcal{Y},1:n})\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})$$

Algorithm 3.1 TT-rounding: Randomize-then-orthogonalize

Require: A tensor $\mathcal{Y}$ in TT format with ranks $\{R_n\}$, target TT-ranks $\{\ell_n\}$
Ensure: A tensor $\mathcal{X}$ in TT format with ranks $\{\ell_n\}$

function $\mathcal{X} =$ TT-RoundingRandOrth($\mathcal{Y}, \{\ell_n\}$)
  Select a random Gaussian TT-tensor $\mathcal{R}$ with target TT-ranks $\{\ell_n\}$
  $\{\mathbf{W}_n\} =$ PartialContractionsRL($\mathcal{Y}, \mathcal{R}$) ▷ compute partial random contractions
  $\mathcal{T}_{\mathcal{X},1} = \mathcal{T}_{\mathcal{Y},1}$
  for $n = 1$ to $N-1$ **do**
    $\mathbf{Z}_n = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ ▷ $\mathcal{T}_{\mathcal{X},n}$ is $\ell_{n-1} \times I_n \times R_n$
    $\mathbf{Y}_n = \mathbf{Z}_n \mathbf{W}_n$ ▷ form the sketched matrix
    $[\mathcal{V}(\mathcal{T}_{\mathcal{X},n}), \sim] = \mathrm{QR}(\mathbf{Y}_n)$ ▷ thin QR to compute an orthonormal basis
    $\mathbf{M}_n = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})^\top \mathbf{Z}_n$ ▷ form $\ell_n \times R_n$ matrix
    $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1}) = \mathbf{M}_n \mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1})$ ▷ $\mathcal{T}_{\mathcal{X},n+1} = \mathcal{T}_{\mathcal{Y},n+1} \times_1 \mathbf{M}_n$
  end for
end function

Algorithm 3.2 TT-rounding: Two-sided-randomization (generalized Nyström)

Require: A tensor $\mathcal{Y}$ in TT format with ranks $\{R_n\}$, target TT-ranks $\{\ell_n\}$ and $\{\rho_n\}$
Ensure: A tensor $\mathcal{X}$ in TT format with ranks $\{\ell_n\}$

function $\mathcal{X} =$ TT-RoundingRandOrth($\mathcal{Y}, \{\ell_n\}$)
  Generate random Gaussian TT-tensor $\mathcal{L}$ with ranks $\{\ell_n\}$
  Generate random Gaussian TT-tensor $\mathcal{R}$ with ranks $\{\rho_n\}$ ▷ choose $\rho_n > \ell_n$
  $\{\mathbf{W}_n^L\} =$ PartialContractionsLR($\mathcal{Y}, \mathcal{L}$)
    ▷ Precompute sketches from the left
  $\{\mathbf{W}_n^R\} =$ PartialContractionsRL($\mathcal{Y}, \mathcal{R}$)
    ▷ Precompute sketches from the right
  for $n = 1$ to $N-1$ **do**
    $[\mathbf{U}_n, \mathbf{\Sigma}_n, \mathbf{V}_n] = \mathrm{SVD}(\mathbf{W}_n^L \mathbf{W}_n^R)$ ▷ Compute SVD of $\mathbf{W}_n^L \mathbf{W}_n^R$
      ▷ $\mathbf{V}_n$ is $\rho_n \times \ell_n$
    $\mathbf{L}_n = \mathbf{W}_n^R \mathbf{V}_n (\mathbf{\Sigma}_n^\dagger)^{1/2}$ ▷ Determine internal $R_n \times \ell_n$ left factor $\mathbf{L}_n$
    $\mathbf{R}_n = (\mathbf{\Sigma}_n^\dagger)^{1/2} \mathbf{U}_n^\top \mathbf{W}_n^L$ ▷ Determine internal $\ell_n \times R_n$ right factor $\mathbf{R}_n$
  end for
  $\mathcal{V}(\mathcal{T}_{\mathcal{X},1}) = \mathcal{V}(\mathcal{T}_{\mathcal{Y},1}) \mathbf{L}_1$
  for $n = 2$ to $N-1$ **do**
    $\mathcal{H}(\mathcal{T}_{\mathcal{X},n}) = \mathbf{R}_{n-1} \mathcal{H}(\mathcal{V}(\mathcal{T}_{\mathcal{Y},n}) \mathbf{L}_n)$ ▷ $\mathcal{T}_{\mathcal{X},n} = \mathcal{T}_{\mathcal{Y},n} \times_1 \mathbf{R}_{n-1} \times_3 \mathbf{L}_n$
      ▷ hence $\mathcal{T}_{\mathcal{X},n}$ is $\ell_{n-1} \times I_n \times \ell_n$
  end for
  $\mathcal{H}(\mathcal{T}_{\mathcal{X},N}) = \mathbf{R}_{N-1} \mathcal{H}(\mathcal{T}_{\mathcal{Y},N})$
end function

with factors $\mathcal{V}(\mathcal{T}_{\mathcal{Y},1:n}) \in \mathbb{R}^{(I_1 \cdots I_n) \times R_n}$ and $\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N}) \in \mathbb{R}^{R_n \times (I_{n+1} \cdots I_N)}$. Similarly, we define matrices

$$\mathbf{\Psi}_n := \mathcal{V}(\mathcal{T}_{\mathcal{L},1:n})^\top \in \mathbb{R}^{\ell_n \times (I_1 \cdots I_n)},$$
$$\mathbf{\Omega}_n := \mathcal{H}(\mathcal{T}_{\mathcal{R},n+1:N})^\top \in \mathbb{R}^{(I_{n+1} \cdots I_N) \times \rho_n}$$

as the partial unfoldings of random Gaussian TT-tensors $\mathcal{L}$ and $\mathcal{R}$, respectively. Then multiplying matrix $\mathbf{Y}_{(1:n)}$ on the left by $\mathbf{\Psi}_n$ and on the right by $\mathbf{\Omega}_n$ yields

$$\mathbf{\Psi}_n \mathbf{Y}_{(1:n)} \mathbf{\Omega}_n = (\mathbf{\Psi}_n \mathcal{V}(\mathcal{T}_{\mathcal{Y},1:n}))(\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})\mathbf{\Omega}_n) = \mathbf{W}_n^L \mathbf{W}_n^R.$$

Following identity (2.2) illustrating the main idea of the generalized Nyström method for matrices discussed in subsection 2.2, we have

$$\begin{aligned}\mathbf{Y}_{(1:n)} &\approx \left(\mathbf{Y}_{(1:n)}\mathbf{\Omega}_n\right)\left(\mathbf{\Psi}_n \mathbf{Y}_{(1:n)}\mathbf{\Omega}_n\right)^\dagger \left(\mathbf{\Psi}_n \mathbf{Y}_{(1:n)}\right) \\ &= \mathcal{V}(\mathcal{T}_{\mathcal{Y},1:n})\mathbf{W}_n^R \left(\mathbf{W}_n^L \mathbf{W}_n^R\right)^\dagger \mathbf{W}_n^L \mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N}) \\ &= \mathcal{V}(\mathcal{T}_{\mathcal{Y},1:n})\mathbf{L}_n \mathbf{R}_n \mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N});\end{aligned}$$

see Figure SM3. Having all left and right factors at hand, for each core of the tensor $\mathcal{Y}$ (treating the first and last cores separately), we distribute them according to the formula

$$\mathcal{H}(\mathcal{T}_{\mathcal{X},n}) = \mathbf{R}_{n-1}\mathcal{H}(\mathcal{V}(\mathcal{T}_{\mathcal{Y},n})\mathbf{L}_n)$$

forming the cores of the resulting tensor $\mathcal{X}$; see Figure SM4.

In contrast to Algorithm 3.1, the two-sided-randomization approach does not produce an orthogonal tensor. However, with a little restructuring, it can be adapted to produce an orthogonal tensor (we do not discuss that here). This variation may be useful for the case when the target TT-ranks are not known in advance, and producing an orthogonal tensor can be used in conjunction with Algorithm 2.2 to further compress the tensor.

3.3. Rounding of TT-sums. One of the most common arithmetic operations that depends on TT-rounding is TT-summation, i.e., we want to compress a tensor $\mathcal{Y}$ that is available as the sum of s TT-tensors: $\mathcal{Y} = \mathcal{Y}^{(1)} + \cdots + \mathcal{Y}^{(s)}$. To reuse existing algorithms, there are two options available to us. For example, we can form the TT-tensor $\mathcal{Y}$ explicitly and then apply one of the compression algorithms proposed previously. As we will argue in subsection 3.4, the computational cost of this approach has cubic scaling with respect to the number of summands s using the TT-rounding approach, and a quadratic scaling with respect to s using the randomized approaches (randomize-then-orthogonalize and two-sided-randomization). This is computationally infeasible as s becomes large. Alternatively, we can form the partial sum $\mathcal{Y}^{(1)} + \mathcal{Y}^{(2)}$, compress this partial sum, add the resulting truncated term to the summand $\mathcal{Y}^{(3)}$, and proceed in the same way with the remaining terms; see, e.g., [5]. Variations of this approach can be performed using ideas from the summation methods described in [28, Chapter 4.1]. These approaches scale linearly with s, but assuming that one prescribes a certain tolerance at each pairwise summation step, this could lead to large intermediate TT-ranks even if the rank of the overall sum is relatively small.

In this subsection, we show how to combine the addition and randomized rounding operations to reduce further the computational costs, which is particularly effective when the number of summands s is large. The basic idea is to exploit the nonzero structure of the TT-cores of the sum of TT-tensors to avoid computing with zeros. Applying the orthogonalization phase, as required to perform the deterministic truncation phase, requires assembling the TT representation of the sum. Furthermore, the orthogonalization and multiplication by the triangular factor destroy the structure in the middle cores. By using randomization, we can avoid this explicit TT assembly of

Algorithm 3.3 TT-rounding of a sum: Randomize-then-orthogonalize

Require: Tensors $\{\mathcal{Y}^{(j)}\}_{1\le j\le s}$ in TT format with ranks $\{R_n^{(j)}\}_{1\le j\le s}$, target TT-ranks $\{\ell_n\}$

Ensure: A tensor $\mathcal{X}\approx\sum_{j=1}^{s}\mathcal{Y}^{(j)}$ in TT format with ranks $\{\ell_n\}$

function $\mathcal{X}=$ TT-ROUNDING-SUM-RANDORTH($\{\mathcal{Y}^{(j)}\}_{1\le j\le s},\{\ell_n\}$)
  Select random Gaussian TT-tensor $\mathcal{R}$ with ranks $\{\ell_n\}$
  for $j=1$ to s **do**
    $\{\mathbf{W}_n^{(j)}\}=$ PARTIALCONTRACTIONSRL($\mathcal{Y}^{(j)},\mathcal{R}$) ▷ precompute sketches from the right
  end for
  $\mathcal{T}_{\mathcal{X},1}=\begin{bmatrix}\mathcal{T}_{\mathcal{Y}^{(1)},1} & \dots & \mathcal{T}_{\mathcal{Y}^{(s)},1}\end{bmatrix}$
  for $n=1$ to $N-1$ **do**
    $\mathbf{Z}_n=\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ ▷ $\mathcal{T}_{\mathcal{X},n}$ is $\ell_{n-1}\times I_n\times\sum_{j=1}^{s}R_n^{(j)}$
    $\mathbf{Y}_n=\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\begin{bmatrix}\mathbf{W}_n^{(1)}\\ \vdots\\ \mathbf{W}_n^{(s)}\end{bmatrix}$ ▷ complete random sketch
    $[\mathcal{V}(\mathcal{T}_{\mathcal{X},n}),\sim]=\mathrm{QR}(\mathbf{Y}_n)$ ▷ thin QR factorization
    $\begin{bmatrix}\mathbf{M}_n^{(1)} & \dots & \mathbf{M}_n^{(s)}\end{bmatrix}=\mathcal{V}(\mathcal{T}_{\mathcal{X},n})^{\top}\mathbf{Z}_n$
    if $n<N-1$ **then** ▷ exploit structure in next internal core
      $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})=\begin{bmatrix}\mathbf{M}_n^{(1)}\mathcal{H}(\mathcal{T}_{\mathcal{Y}^{(1)},n+1}) & \cdots & \mathbf{M}_n^{(s)}\mathcal{H}(\mathcal{T}_{\mathcal{Y}^{(s)},n+1})\end{bmatrix}$
    else
      $\mathcal{T}_{\mathcal{X},N}=\sum_{j=1}^{s}\mathbf{M}_{n-1}^{(j)}\mathcal{T}_{\mathcal{Y}^{(j)},N}$
    end if
  end for
end function

the sum and avoid unnecessary computations on zeros. Algorithm 3.3 provides the pseudocode for rounding the sum of s input TT-tensors. To simplify the notation, we derive the efficient computations considering the case $s=2$, as the generalization will be clear. Let $\mathcal{Y}$ and $\mathcal{Z}$ be two TT-tensors and consider the TT-tensor $\mathcal{X}=\mathcal{Y}+\mathcal{Z}$. Let $\mathcal{R}$ be a given random Gaussian TT-tensor and let $\{\mathbf{W}_n^{\mathcal{Y}}\}$ and $\{\mathbf{W}_n^{\mathcal{Z}}\}$ for $n=1,\dots,N-1$ be the right-to-left partial contractions of $\mathcal{Y}$ and $\mathcal{Z}$ with $\mathcal{R}$. We have

$$\mathbf{X}_{(1:n)}=\mathcal{V}(\mathcal{T}_{\mathcal{X},1:n})\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1:N})=\begin{bmatrix}\mathcal{V}(\mathcal{T}_{\mathcal{Y},1:n}) & \mathcal{V}(\mathcal{T}_{\mathcal{Z},1:n})\end{bmatrix}\begin{bmatrix}\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})\\ \mathcal{H}(\mathcal{T}_{\mathcal{Z},n+1:N})\end{bmatrix},$$

and by using (2.1),

$$\mathcal{V}(\mathcal{T}_{\mathcal{X},1:n})=\begin{bmatrix}(\mathbf{I}_{I_n}\otimes\mathcal{V}(\mathcal{T}_{\mathcal{Y},1:n-1})) & (\mathbf{I}_{I_n}\otimes\mathcal{V}(\mathcal{T}_{\mathcal{Z},1:n-1}))\end{bmatrix}\begin{bmatrix}\mathcal{V}(\mathcal{T}_{\mathcal{Y},n}) & \\ & \mathcal{V}(\mathcal{T}_{\mathcal{Z},n})\end{bmatrix},$$

$$\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1:N})=\begin{bmatrix}\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1}) & \\ & \mathcal{H}(\mathcal{T}_{\mathcal{Z},n+1})\end{bmatrix}\begin{bmatrix}\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+2:N})\otimes\mathbf{I}_{I_{n+1}}\\ \mathcal{H}(\mathcal{T}_{\mathcal{Z},n+2:N})\otimes\mathbf{I}_{I_{n+1}}\end{bmatrix}.$$

The matrix $\mathbf{W}_n^{\mathcal{X}}$ can be expressed as

$$\mathbf{W}_n^{\mathcal{X}} = \mathcal{H}(\mathcal{T}_{\mathcal{X},n+1:N})\mathcal{H}(\mathcal{T}_{\mathcal{R},n+1:N})^\top = \begin{bmatrix}\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1:N})\\ \mathcal{H}(\mathcal{T}_{\mathcal{Z},n+1:N})\end{bmatrix}\mathcal{H}(\mathcal{T}_{\mathcal{R},n+1:N})^\top = \begin{bmatrix}\mathbf{W}_n^{\mathcal{Y}}\\ \mathbf{W}_n^{\mathcal{Z}}\end{bmatrix}.$$

This justifies the procedure in Algorithm 3.3 of computing the partial contractions separately for each summand (line 4) and concatenating them (line 9).

After the QR factorization of the projected matrix produces the truncated core, we compute the contraction between the new and old cores (line 11) and store the result in a matrix $\mathbf{M}_n = \begin{bmatrix}\mathbf{M}_n^{\mathcal{Y}} & \mathbf{M}_n^{\mathcal{Z}}\end{bmatrix}$ of size $\ell_n \times (R_n^{\mathcal{Y}} + R_n^{\mathcal{Z}})$. This matrix is now multiplied from the right by $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1:N})$ to compute the updated right factor of $\mathbf{X}_{(1:n)}$. This multiplication can be absorbed by the $(n+1)$th core as follows:

$$\begin{aligned}\mathbf{M}_n\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1:N}) &= \begin{bmatrix}\mathbf{M}_n^{\mathcal{Y}} & \mathbf{M}_n^{\mathcal{Z}}\end{bmatrix}\begin{bmatrix}\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1}) & \\ & \mathcal{H}(\mathcal{T}_{\mathcal{Z},n+1})\end{bmatrix}\begin{bmatrix}\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+2:N})\otimes\mathbf{I}_{I_{n+1}}\\ \mathcal{H}(\mathcal{T}_{\mathcal{Z},n+2:N})\otimes\mathbf{I}_{I_{n+1}}\end{bmatrix},\\ &= \begin{bmatrix}\mathbf{M}_n^{\mathcal{Y}}\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1}) & \mathbf{M}_n^{\mathcal{Z}}\mathcal{H}(\mathcal{T}_{\mathcal{Z},n+1})\end{bmatrix}\begin{bmatrix}\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+2:N})\otimes\mathbf{I}_{I_{n+1}}\\ \mathcal{H}(\mathcal{T}_{\mathcal{Z},n+2:N})\otimes\mathbf{I}_{I_{n+1}}\end{bmatrix}.\end{aligned}$$

Hence we update $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})$ as in line 13. For $n = N-1$ we have

$$\mathcal{T}_{\mathcal{X},N} = \begin{bmatrix}\mathbf{M}_{N-1} & \mathbf{M}_{N-1}\end{bmatrix}\begin{bmatrix}\mathcal{T}_{\mathcal{Y},N}\\ \mathcal{T}_{\mathcal{Z},N}\end{bmatrix} = \mathbf{M}_{N-1}\mathcal{T}_{\mathcal{Y},N} + \mathbf{M}_{N-1}\mathcal{T}_{\mathcal{Z},N}.$$

Generalizing this expression to s terms yields line 15.

3.4. Computational costs. To analyze the computational cost, we make the following assumptions that will simplify the analysis. Let $\mathcal{Y} \in \mathbb{R}^{I\times\cdots\times I}$ be a tensor of order N with ranks $(1, R, \ldots, R, 1)$ in TT format. We want to compress $\mathcal{Y}$ to obtain a TT-tensor $\mathcal{X}$ with ranks $(1, \ell, \ldots, \ell, 1)$. Here and in section SM1, we assume that $\ell = \Theta(R)$.

3.4.1. Randomized compression algorithms. We now analyze the computational cost of Algorithms 3.1 and 3.2. The computational cost of Algorithm SM2.1 is given in section SM2.

Randomize-then-orthogonalize (Algorithm 3.1*).* Denoting the total cost of Algorithm 3.1 with C_{RtO}, we analyze the main components that contribute to the total computational cost. Line 3 invokes Algorithm 2.3 with the corresponding cost denoted by C_{Contr}; see section SM1. Lines 7 and 9 contribute by a factor of $2IR\ell^2$ that is the cost of performing the multiplication $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathbf{W}_n$ of sizes $\ell I \times R$ with $R \times \ell$ and $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})^T\mathbf{Z}_n$ of sizes $\ell \times I\ell$ and $I\ell \times R$ that prepare the matrices $\mathbf{Y}_n$ and $\mathbf{M}_n$, respectively, for the next steps. The term $4I\ell^3 + \mathcal{O}(\ell^3)$ represents the cost of the thin QR factorization, in line 8, of the matrix $\mathbf{Y}_n$ of size $I\ell \times \ell$. Line 10 involves multiplication of matrices of size $\ell \times R$ and $R \times IR$ which costs $2IR^2\ell$ flops. The total cost of Algorithm 3.1 is

$$\begin{aligned}C_{\text{RtO}} =\ & C_{\text{Contr}} + (N-2)(2IR^2\ell + 4IR\ell^2 + 4I\ell^3) + \mathcal{O}(IR^2 + NR^3)\\ =\ & I(N-2)\cdot(4R^2\ell + 6R\ell^2 + 4\ell^3) + \mathcal{O}(IR^2 + NR^3) \quad \text{flops.}\end{aligned}$$

Two-sided-randomization (Algorithm 3.2*).* Here we analyze the total cost of Algorithm 3.2 and denote it by C_{2SR}. The cost of lines 4 and 5 is $2C_{\text{Contr}}$. Here, we recall that the cost of the partial contractions from the left and from the right is the

same since we assume that ranks are the same, i.e., $\rho_n = \ell_n = \ell$. However, in practice, we choose $\rho_n > \ell_n$ to be different for numerical stability. There are two other contributions to the cost that come from two different matrix multiplication: first, of sizes $\ell \times R$ and $R \times IR$, and second, of sizes $\ell \times R$ and $R \times I\ell$. The cost of the for loop starting with line 12 is $\mathcal{O}(NR^2)$. Hence, the total computational cost of Algorithm 3.2 is

$$\begin{aligned} C_{\text{2SR}} &= 2 \cdot C_{\text{Contr}} + (N-2)(2IR^2\ell + 2IR\ell^2) + \mathcal{O}(NR^2) \\ &= I(N-2) \cdot (6R^2\ell + 6R\ell^2) + \mathcal{O}(IR\ell + NR^2) \quad \text{flops.} \end{aligned}$$

Nakatsukasa [41] suggests oversampling from the left, i.e., taking $\rho_n = \lceil 1.5\ell_n \rceil$. We follow this suggestion in our numerical experiments. With this assumption, the cost is slightly higher, i.e., $I(N-2) \cdot (7R^2\ell + 8.5R\ell^2) + \mathcal{O}(IR\ell + NR^2)$ flops, due to the increased cost of the contraction (Algorithm 2.3) with a larger random tensor.

Comparison of different algorithms. To enable the comparison of the different algorithms, we set a target rank as $\ell = \beta R$, where $\beta \in (0, 1]$ is the ratio between the target rank ℓ and the current rank R. This allows us to compare the different algorithms more clearly. A summary of the dominant costs of the algorithms is provided in Table 3.1. For simplicity, we also provide a simplified representation of the computational costs with $\beta = \ell/R$. In Figure 3.2, we plot the speedup of the randomized algorithms compared to the TT-rounding algorithm; we used the simplified representation of the costs while generating the figure. It can be easily seen that all the proposed methods are faster than the TT-rounding algorithm. However, the speedup using the orthogonalize-then-randomize algorithm is very incremental. In contrast, the other two algorithms, randomize-then-orthogonalize and two-sided-randomization, have very similar costs (randomize-then-orthogonalize is slightly more efficient for smaller β) and have much higher speedups especially if $\beta \ll 1$. If $\beta \approx 1$, both algorithms are very close to the TT-rounding algorithm. Therefore, the proposed methods are most efficient if $\beta \ll 1$, i.e., the target rank ℓ is much smaller compared to the original rank R.

3.4.2. Rounding of TT-sums. To explain the benefits of the algorithm for rounding TT-sums in subsection 3.3, consider the summation of s tensors of order N (size I in each dimension) each with TT-ranks $(1, R, \ldots, R, 1)$. Suppose we form the TT-tensor $\mathcal{Y}$, which represents the summation $\mathcal{Y} = \sum_{j=1}^{s} \mathcal{Y}^{(j)}$, explicitly.

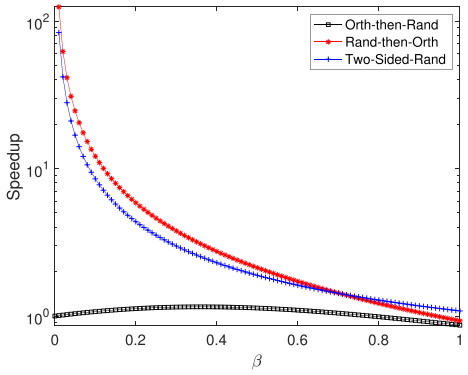

FIG 3.2. *Illustration of the speedups obtained by the randomized algorithms compared with the TT-rounding. Here $\beta = \ell/R$ is the ratio between the target rank and the original rank of the tensor. The speedup computations are based on the simplified cost analysis in Table* 3.1.

TABLE 3.1
Summary of the computational costs (discarding lower order terms) of the randomized algorithms proposed in this paper. For completeness, we also include the computational costs of the deterministic algorithms in subsection 2.3. *Orth and Contr refer to Algorithm* 2.1 *and Algorithm* 2.3*, respectively.*

Algorithms	Computational cost (flops)	Simplified cost (flops)
Orth	$(N-2)I(5R^3)$	–
Contr	$(N-2)I(2R^2\ell+2R\ell^2)$	–
TT-rounding	$(N-2)I(5R^3+6R^2\ell+2R\ell^2)$	$(N-2)IR^3(5+6\beta+2\beta^2)$
Orth-then-rand	$(N-2)I(5R^3+2R^2\ell+4R\ell^2+4\ell^3)$	$(N-2)IR^3(5+2\beta+4\beta^2+4\beta^3)$
Rand-then-orth	$(N-2)I(2R^2\ell+4R\ell^2+4\ell^3)$	$(N-2)IR^3(4\beta+6\beta^2+4\beta^3)$
Two-sided-rand	$(N-2)I(6R^2\ell+6R\ell^2)$	$(N-2)IR^3(6\beta+6\beta^2)$

The intermediate cores have size $sR \times I \times sR$; the first core is of size $I \times sR$ and the last core is of size $sR \times I$. Suppose the target compression rank in each mode is ℓ. To leading order, the cost of executing TT-rounding and orthogonalize-then-randomize is $\mathcal{O}(Ns^3R^3I)$. In contrast, the costs of using randomize-then-orthogonalize and the two-sided-randomization approach are both $\mathcal{O}(N\ell s^2R^2I)$. This can be beneficial if the number of summands s is large, or the rank R is large. This simple cost analysis does not take into account any structure present in the summation.

Carefully exploiting the structure, as in subsection 3.3, can reduce this cost. In particular, by using TT-rounding of a sum with s tensors of order N with randomize-then-orthogonalize summarized in Algorithm 3.3 the leading order computational cost is $\mathcal{O}(N\ell sR^2I)$ flops. Notice that by exploiting the structure of the tensor and using randomization, the leading order of the cost is decreased to be linear in s in contrast to cubic in s when no structure was taken into account and the TT-sum tensor was formed explicitly. This decrease in the computational cost is obviously more pronounced when the number of summands s is large. In what follows, we present the analysis of the computational cost of computing the sum of s TT-tensors by randomization and by exploiting the underlying tensor structure.

TT-rounding of a sum: Randomize-then-orthogonalize (Algorithm 3.3*).* We analyze the computational cost of Algorithm 3.3, which we denote by C_{RtOsum}. The leading order term is sourced from two main contributions: (1) line 4 that executes Algorithm 2.3 s times resulting in a total computational cost of $sI(N-2)(2R\ell^2+2R^2\ell)$ flops and (2) line 13 that represents s multiplications between matrices of size $IR \times R$ and $R \times \ell$ resulting in a total cost of $2sIR^2\ell$ flops. Next we analyze the source of the second leading order term present in the total cost. Line 9 contributes by a factor of two to the second leading term with a cost of $2sIR\ell^2$ flops resulting from multiplying matrices of size $I\ell \times sR$ and $sR \times \ell$. Another factor of two comes from the multiplication of two matrices of size $\ell \times I\ell$ and $I\ell \times sR$ in line 11, resulting in a total cost of $2sIR\ell^2$ flops. The last factor of two comes from the second leading order term of Algorithm 2.3. Computing the thin QR factorization of the matrix of size $I\ell \times \ell$ in line 10 costs $4I\ell^3$ flops. Hence, the total cost of Algorithm 3.3 is

$$\begin{aligned} C_{\text{RtOsum}} &= s \cdot C_{\text{Contr}} + (N-2)(2sIR^2\ell + 4sIR\ell^2 + 4I\ell^3) + \mathcal{O}(NIR^2) \\ &= I(N-2)\cdot(4sR^2\ell + 6sR\ell^2 + 4\ell^2) + \mathcal{O}(NIR^2) \quad \text{flops.} \end{aligned}$$

4. Numerical results. In this section, we illustrate numerically the performance of the newly developed algorithms using tensor data in TT format. We consider both synthetic as well as more realistic test examples. Additional numerical experiments are available in section SM4. All the numerical experiments were

performed on MATLAB R2021a running on a laptop computer with an Intel Core i9-9980H CPU and 64GB of RAM, using multithreading with four computational threads.

4.1. TT-tensor with a fixed target rank. In our first numerical experiment, we illustrate the performance of our rounding algorithms by rounding a random TT-tensor with a known low-rank representation. Throughout, we choose the ranks of the right-side randomization in the two-sided-randomization approach (Algorithm 3.2) to be $\rho = \lceil 1.5\ell \rceil$ as discussed in subsection 3.4.1.

The random TT-tensor $\boldsymbol{\mathcal{X}}$ is constructed by perturbing a random TT-tensor $\boldsymbol{\mathcal{X}}_1$ with the random TT-tensor $\epsilon\boldsymbol{\mathcal{X}}_2$ as follows: $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}_1 + \epsilon\boldsymbol{\mathcal{X}}_2$. TT-tensors $\boldsymbol{\mathcal{X}}_1,\ \boldsymbol{\mathcal{X}}_2 \in \mathbb{R}^{100\times\cdots\times 100}$ are order $N = 10$ normalized random TT-tensors of ranks $(1, 50, \ldots, 50, 1)$ (normalized according to their dimension as described in Definition 3.1), and ϵ is a perturbation scalar taking the values $\epsilon \in \{10^{-2}, 10^{-6}, 10^{-10}\}$. The ranks of the perturbed tensor $\boldsymbol{\mathcal{X}}$ are $(1, 50+50, \ldots, 50+50, 1)$, and the perturbation parameter ϵ determines how well tensor $\boldsymbol{\mathcal{X}}$ is approximated by the lower rank tensor $\boldsymbol{\mathcal{X}}_1$, i.e., if ϵ is small, then $\boldsymbol{\mathcal{X}}_1$ is a good rank-$(1, 50, \ldots, 50, 1)$ approximation of $\boldsymbol{\mathcal{X}}$.

We round the random TT-tensor $\boldsymbol{\mathcal{X}}$ using Algorithms 2.2, 3.1, 3.2, and SM2.1 to have ranks $(1, \ell, \ldots, \ell, 1)$, where we vary the parameter ℓ from 35 to 80 by an increment of 5. We present these results in Figure 4.1. The approximation error is the relative norm error between tensor $\boldsymbol{\mathcal{X}}$ and the approximate rounded tensor $\hat{\boldsymbol{\mathcal{X}}}$, i.e., $\frac{\|\boldsymbol{\mathcal{X}}-\hat{\boldsymbol{\mathcal{X}}}\|}{\|\boldsymbol{\mathcal{X}}\|}$. We also present the time speedup (computed with the average of five independent runs) of the randomized algorithms compared to the deterministic algorithm.

For all values of perturbation ϵ, the error resulting from the deterministic algorithm (Algorithm 2.2) decreases slightly until the ranks of the rounded tensor are $\ell = 50$. When $\ell > 50$, the error appears to be very close to ϵ. The errors resulting from the randomized algorithms (Algorithms 3.1, 3.2, and SM2.1) are greater than the error resulting from the deterministic algorithm. Additionally, the randomized algorithms produce a more accurate approximation when the target rank is larger and are more accurate when ϵ is small. The orthogonalize-then-randomize and the randomize-then-orthogonalize method (Algorithms 3.1 and SM2.1) produce a rounded tensor with similar levels of accuracy, while the two-sided-randomization approach is the least accurate. For smaller values of ϵ, there is less difference in the accuracy between the different algorithms. The randomize-then-orthogonalize algorithm is the fastest compared to the deterministic algorithm, followed by the two-sided-randomization and orthogonalize-then-randomize algorithms.

4.2. Solving a parametric PDE in the TT format. As a realistic test example, we consider the parameter dependent PDE referred to in the literature as the *cookie problem* [38, 52]; see section SM5 for details. Since it is known that the set of solutions of problem (SM5.1) admits a low-rank representation [14, 24], we consider a global linear system encapsulating all these linear systems, i.e.,

$$\left(\sum_{i=1}^{N} \mathbf{A}_{i,1} \otimes \ldots \otimes \mathbf{A}_{i,N}\right) \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{F}},$$

where $\mathbf{A}_{1,1}$ is the discretization of the operator over the spatial domain with constant parameter values, for each $2 \le i \le N$, $\mathbf{A}_{i,1}$ is the discretization of the operator over the domain multiplied by the characteristic function corresponding to the corresponding subdomain, and $\mathbf{A}_{i,i}$ is a diagonal matrix containing the parameter values for the corresponding parameter, and for each $2 \le j \ne i \le N$, $\mathbf{A}_{i,j}$ is the identity matrix.

We use the TT-GMRES algorithm [17] to solve this global linear system of equations. The preconditioned TT-GMRES algorithm builds the basis vectors in TT format $\mathcal{V}_1, \mathcal{V}_2, \ldots$ using the inexact Arnoldi procedure; since at each step the corresponding TT-tensors are rounded, this results in an inexact Krylov subspace method. The main bottleneck is the computation of two linear combinations in each iteration. First, the following sum of N tensors with the same ranks as $\mathcal{V}_k$ is formed when applying the operator to the kth basis vector computed at the previous iteration, i.e.,

$$\mathcal{W} = \sum_{i=1}^{N} (\mathbf{A}_{i,1} \otimes \cdots \otimes \mathbf{A}_{i,N})\, \mathcal{Y},$$

after application of the preconditioner, $\mathcal{Y} = ((\sum_{i=1}^{N} \mathbf{A}_{i,1})^{-1} \otimes I \otimes \cdots \otimes I)\mathcal{V}_k$. Second, a linear combination of $k+1$ tensors appears when using the Gram–Schmidt algorithm to orthogonalize $\mathcal{W}$ with respect to the previous basis vectors,

$$\mathcal{Z} = \left(\mathcal{W} - \sum_{j=1}^{k} h_{jk}\mathcal{V}_k\right), \qquad \mathcal{V}_{k+1} = \frac{1}{h_{k+1,k}}\mathcal{Z}, \qquad \begin{cases} h_{jk} = \langle \mathcal{V}_k, \mathcal{W} \rangle, & j = 1, \ldots, k, \\ h_{k+1,k} = \|\mathcal{Z}\|_F. \end{cases}$$

In both cases, the addition of TT-tensors is followed by a TT-rounding operation in order to reduce the ranks and keep the computations tractable. Hence, these steps are amenable to acceleration by using the randomized Algorithm 3.3 as a rounding procedure in the aforementioned computations. Because the second linear combination

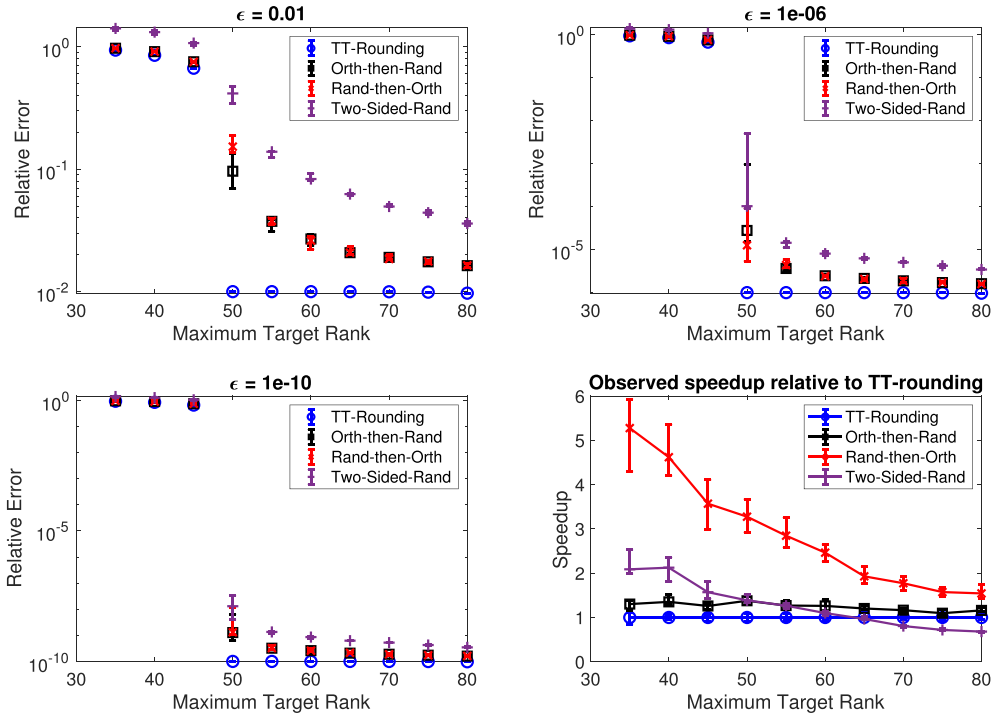

FIG 4.1. *Comparison of error between a low-rank tensor and full-rank perturbed tensor, and timings using the deterministic and randomized TT-rounding algorithms for ϵ perturbed tensor for different values of perturbation ϵ. Statistics were based on five independent runs.*

involves $k+1$ summands, the randomized implementation reduces greatly the cost of later TT-GMRES iterations in particular, as the leading order of its computational cost is decreased from cubic to linear in k, as detailed in subsection 3.4.

We perform numerical experiments with $N = 5$ using a piecewise linear finite element discretization with the mesh presented in Figure SM7, for various choices of the number of parameter samples, $I = I_2 = \cdots = I_N$, with values of ρ_i distributed linearly between 1 and 10. The relative tolerance of the TT-GMRES solver is set to 10^{-8}. We compare the naive, deterministic implementation of the preconditioned TT-GMRES algorithm with one using randomized summation and rounding steps. Results of the comparison are reported in Figure 4.2. On the right, we display all internal ranks of the TT representation of the Krylov vector computed at that iteration. We observe that the ranks of the basis vectors and number of iterations are the same using both implementations, and they depend only weakly on the dimension I of the parameter modes of the tensors. The speedup achieved by the randomized approach increases consistently with the number of parameter samples.

Taking a closer look at the timing statistics as presented in Figure 4.3, we note that the computation and rounding of the linear combinations identified above indeed dominate the computational cost in both cases and are a clear computational bottleneck for the deterministic implementation in particular, as the ranks of the sum tensor increase to as much as 2491 in these experiments. This explains the remarkable speedup obtained with the randomized approach.

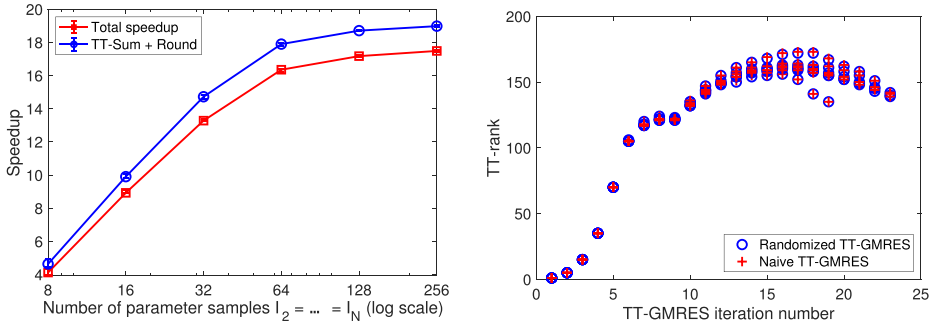

FIG 4.2. *Illustration of the speedups (left) and TT-ranks of the Krylov basis vectors (right) obtained by the deterministic and randomized summation and rounding algorithms within the TT-GMRES algorithm to solve problem* (SM5.1).

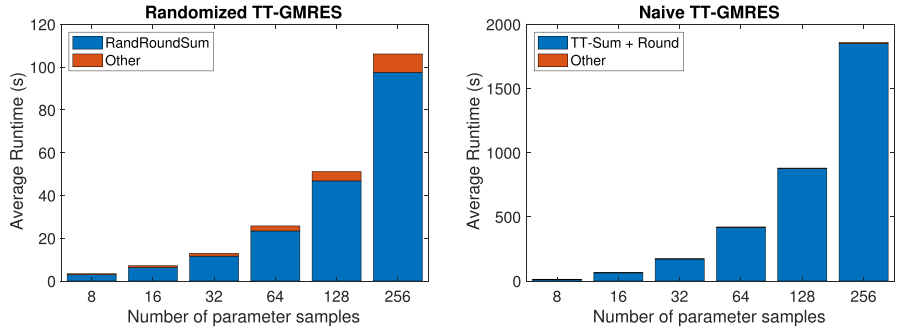

FIG 4.3. *Illustration of the timings using the deterministic and randomized summation and rounding algorithms within the TT-GMRES algorithm to solve problem* (SM5.1).

5. Conclusions and outlook. In this paper, we present randomized algorithms for rounding a tensor, assuming that we have an initial representation in the TT format. This initial representation may not be optimal in terms of storage, and the randomized compression techniques can be used to obtain a more efficient representation. We derive three different algorithms: orthogonalize-then-randomize, randomize-then-orthogonalize, and two-sided-randomization. We study the computational cost of these algorithms in some detail and show that it can be much smaller than the standard TT-rounding algorithm. Additionally, we consider the special case of rounding a TT-tensor that is represented as the sum of many TT-tensors. While applying each of the randomized algorithms proposed here can reduce the computational cost over standard TT-rounding, we further exploit the structure of the problem to reduce the computational cost to be linear in the number of summands. We perform extensive numerical experiments and achieve over 20× speedups on test problems compared to standard algorithms. There are many avenues for future investigations. First, it would be interesting to derive probabilistic bounds for the accuracy of the rounding approach. Second, we could consider extending our algorithms to the case where the TT-tensor is obtained as the Hadamard (or elementwise) product of two tensors. Finally, another extension worth considering is developing randomized rounding algorithms in the $\mathcal{H}$-Tucker format.

6. Acknowledgments. This work was initiated as a part of the Statistical and Applied Mathematical Sciences Institute (SAMSI) Program on Numerical Analysis in Data Science in fall 2020. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF). The authors thank Dr. Jocelyn Chi and Prof. Eric de Sturler for the constructive discussions and suggestions.

REFERENCES

[1] S. Ahmadi-Asl, S. Abukhovich, M. G. Asante-Mensah, A. Cichocki, A. H. Phan, T. Tanaka, and I. Oseledets, *Randomized algorithms for computation of Tucker decomposition and higher order SVD (HOSVD)*, IEEE Access, 9 (2021), pp. 28684–28706.

[2] S. Ahmadi-Asl, A. Cichocki, A. H. Phan, M. G. Asante-Mensah, M. M. Ghazani, T. Tanaka, and I. Oseledets, *Randomized algorithms for fast computation of low rank tensor ring model*, Mach. Learn. Sci. Technol., 2 (2020), 011001.

[3] N. Alger, P. Chen, and O. Ghattas, *Tensor train construction from tensor actions, with application to compression of large high order derivative tensors*, SIAM J. Sci. Comput., 42 (2020), pp. A3516–A3539.

[4] J. Ballani and L. Grasedyck, *A projection method to solve linear systems in tensor format*, Numer. Linear Algebra Appl., 20 (2013), pp. 27–43.

[5] R. Ballester-Ripoll and R. Pajarola, *Tensor decompositions for integral histogram compression and look-up*, IEEE Trans. Vis. Comput. Graph., 25 (2018), pp. 1435–1446.

[6] K. Batselier, W. Yu, L. Daniel, and N. Wong, *Computing low-rank approximations of large-scale matrices with the tensor network randomized SVD*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1221–1244.

[7] M. H. Beck, A. Jäckle, G. A. Worth, and H.-D. Meyer, *The multiconfiguration time-dependent Hartree (MCTDH) method: A highly efficient algorithm for propagating wavepackets*, Phys. Rep., 324 (2000), pp. 1–105.

[8] G. Beylkin, J. Garcke, and M. J. Mohlenkamp, *Multivariate regression and machine learning with sums of separable functions*, SIAM J. Sci. Comput., 31 (2009), pp. 1840–1857.

[9] M. Che and Y. Wei, *Randomized algorithms for the approximations of Tucker and the tensor train decompositions*, Adv. Comput. Math., 45 (2019), pp. 395–428.

[10] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, *Tensor networks for dimensionality reduction and large-scale optimization: Part* 1, *low-rank tensor decompositions*, Found. Trends Mach. Learn., 9 (2016), pp. 249–429.

[11] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. V. Oseledets, M. Sugiyama, and D. Mandic, *Tensor networks for dimensionality reduction and large-scale optimizations. Part 2: Applications and future perspectives*, Found. Trends, 9 (2017), pp. 431–673, https://doi.org/10.1561/2200000067.

[12] N. Cohen, O. Sharir, and A. Shashua, *On the expressive power of deep learning: A tensor analysis*, in Proceedings of the Conference on Learning Theory, PMLR, 2016, pp. 698–728.

[13] H. A. Daas, G. Ballard, and P. Benner, *Parallel algorithms for tensor train arithmetic*, SIAM J. Sci. Comput., 44 (2022), https://doi.org/10.1137/20M1387158.

[14] W. Dahmen, R. DeVore, L. Grasedyck, and E. Süli, *Tensor-sparsity of solutions to high-dimensional elliptic partial differential equations*, Found. Comput. Math., 16 (2016), pp. 813–874.

[15] 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.

[16] S. Dolgov and D. Savostyanov, *Parallel cross interpolation for high-precision calculation of high-dimensional integrals*, Comput. Phys. Commun., 246 (2020), 106869.

[17] S. V. Dolgov, *TT-GMRES: Solution to a linear system in the structured tensor format*, Russian J. Numer. Anal. Math. Modelling, 28 (2013), pp. 149–172.

[18] S. V. Dolgov, B. N. Khoromskij, and I. V. Oseledets, *Fast solution of parabolic problems in the tensor train/quantized tensor train format with initial application to the Fokker-Planck equation*, SIAM J. Sci. Comput., 34 (2012), pp. A3016–A3038.

[19] S. V. Dolgov and D. V. Savostyanov, *Alternating minimal energy methods for linear systems in higher dimensions*, SIAM J. Sci. Comput., 36 (2014), pp. A2248–A2271.

[20] D. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. Adams, *Advances in neural information processing systems*, in NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds., Vol. 28, 2015, pp. 2224–2232.

[21] M. Fannes, B. Nachtergaele, and R. F. Werner, *Finitely correlated states on quantum spin chains*, Comm. Math. Phys., 144 (1992), pp. 443–490.

[22] Y. Feng, K. Tang, L. He, P. Zhou, and Q. Liao, *Tensor Train Random Projection*, preprint, arXiv:2010.10797, 2020.

[23] P. Gelss, S. Klus, S. Matera, and C. Schütte, *Nearest-neighbor interaction systems in the tensor-train format*, J. Comput. Phys., 341 (2017), pp. 140–162.

[24] L. Grasedyck, *Existence and computation of low Kronecker-rank approximations for large linear systems of tensor product structure*, Computing, 72 (2004), pp. 247–265.

[25] L. Grasedyck, D. Kressner, and C. Tobler, *A literature survey of low-rank tensor approximation techniques*, GAMM-Mitt., 36 (2013), pp. 53–78.

[26] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, Springer Ser. Comput. Math. 42, Springer, New York, 2012.

[27] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[28] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.

[29] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Phys., 6 (1927), pp. 164–189.

[30] S. Holtz, T. Rohwedder, and R. Schneider, *The alternating linear scheme for tensor optimization in the tensor train format*, SIAM J. Sci. Comput., 34 (2012), pp. A683–A713.

[31] B. Huber, R. Schneider, and S. Wolf, *A randomized tensor train singular value decomposition*, in Compressed Sensing and Its Applications, Springer, New York, 2017, pp. 261–290.

[32] B. N. Khoromskij, *Tensors-structured numerical methods in scientific computing: Survey on recent advances*, Chemom. Intell. Lab. Syst., 110 (2012), pp. 1–19.

[33] S. Klus, P. Gelss, S. Peitz, and C. Schütte, *Tensor-based dynamic mode decomposition*, Nonlinearity, 31 (2018), 3359.

[34] S. Klus, P. Koltai, and C. Schütte, *On the numerical approximation of the Perron-Frobenius and Koopman operator*, J. Comput. Dyn., 3 (2016), pp. 51–79, https://doi.org/10.3934/jcd.2016003.

[35] O. Koch and C. Lubich, *Dynamical tensor approximation*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2360–2375.

[36] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.
[37] D. Kressner, R. Kumar, F. Nobile, and C. Tobler, *Low-rank tensor approximation for high-order correlation functions of Gaussian random fields*, SIAM-ASA J. Uncertain. Quantif., 3 (2015), pp. 393–416.
[38] D. Kressner and C. Tobler, *Low-rank tensor Krylov subspace methods for parametrized linear systems*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1288–1316.
[39] D. Loukrezis, U. Römer, T. Casper, S. Schöps, and H. De Gersem, *High-dimensional uncertainty quantification for an electrothermal field problem using stochastic collocation on sparse grids and tensor train decompositions*, Int. J. Numer. Model. Electron. Netw. Devices Fields, 31 (2018), e2222.
[40] H.-D. Meyer, F. Gatti, and G. A. Worth, *Basic MCTDH theory*, in Multidimensional Quantum Dynamics: MCTDH Theory and Applications, Wiley, New York, 2009, pp. 17–30.
[41] Y. Nakatsukasa, *Fast and Stable Randomized Low-Rank Matrix Approximation*, preprint, arXiv:2009.11392, 2020.
[42] A. Obukhov, M. Rakhuba, A. Liniger, Z. Huang, S. Georgoulis, D. Dai, and L. Van Gool, *Spectral tensor train parameterization of deep learning layers*, in Proceedings of the International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 3547–3555.
[43] I. Oseledets and E. Tyrtyshnikov, *TT-cross approximation for multidimensional arrays*, Linear Algebra Appl., 432 (2010), pp. 70–88.
[44] I. V. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317.
[45] I. V. Oseledets and S. V. Dolgov, *Solution of linear systems and matrix inversion in the TT-format*, SIAM J. Sci. Comput., 34 (2012), pp. A2718–A2739.
[46] 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.
[47] B. Rakhshan and G. Rabusseau, *Tensorized random projections*, in Proceedings of the International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 3306–3316.
[48] H. Rauhut, R. Schneider, and Ž. Stojanac, *Tensor completion in hierarchical tensor representations*, in Compressed Sensing and Its Applications, Springer, New York, 2015, pp. 419–450.
[49] L. Richter, L. Sallandt, and N. Nüsken, *Solving high-dimensional parabolic PDEs using the tensor train format*, in Proceedings of the 38th International Conference on Machine Learning, PMLR, 2021, pp. 8998–9009. https://proceedings.mlr.press/v139/richter21a.html.
[50] D. Savostyanov and I. Oseledets, *Fast adaptive interpolation of multi-dimensional arrays in tensor train format*, in Proceedings of the 2011 International Workshop on Multidimensional (nD) Systems, 2011, pp. 1–8.
[51] U. Schollwöck, *The density-matrix renormalization group*, Rev. Mod. Phys., 77 (2005), pp. 259–315.
[52] C. Tobler, *Low-Rank Tensor Methods for Linear Systems and Eigenvalue Problems*, Ph.D. thesis, ETH Zurich, 2012.
[53] Z. Zhang, X. Yang, I. V. Oseledets, G. E. Karniadakis, and L. Daniel, *Enabling high-dimensional hierarchical uncertainty quantification by ANOVA and tensor-train decomposition*, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., 34 (2014), pp. 63–76.