Randomized algorithms for tensor train approximation

Author:
Stefano Ferioli

Supervisor:
Laura Grigori

Collaborator:
Mariana Graciela Martinez Aguilar

EPFL

Spring 2024

1 Introduction

Introduced by Oseledets in 2011 [7], the tensor train format (TT-format) allows to represent a tensor thorugh the contraction of several three-dimensional tensors called TT-cores. Thanks to a wide range of advantages, the TT-format has found applications over a wide range of fields, including machine learning, scientific computing, and quantum chemistry.

The purpose of this project is to explore how randomization can be used to achieve performance improvements in TT-factorization and TT-rounding algorithms. In the first part of the project, we examine both deterministic and randomized algorithms and we present a comprehensive numerical analysis, considering both the numerical error and the runtime. Thanks to this analysis we managed to estimate, for the considered algorithms, the asymptotic behavior of the error and the runtime when varying different parameters.

In the second part, we focus on the Streaming Tensor Train Approximation (STTA) algorithm by Kressner et al. [6]. We perform more numerical experiments using different sketching operators: Gaussian, Subsampled Randomized Hadamard Transform (SRHT), and Hashed Randomized Hadamard Transform (HRHT). From our experiments, we observe promising results using HRHT. Despite being structured, HRHT displays a very similar behaviour to Gaussian sketching operators. For this reason, in the last part we present our result in updating the proof of the error bound for the HRHT case. We are not aware of anyone else using hashed sketching operators on TTs.

This report is structured as follows. In Section 2, we set the notation utilized in this project. In Section 3, we introduce the notion of tensor train and we present some classic algorithms for TT-factorization, TT-sum and TT-rounding. In Section 4, we give an overview of three randomized algorithms, both for TT-factorization and TT-rounding, taken from the work of Al Daas et al. [1]. Then, we proceed by presenting our numerical experiments on the algorithms using a variety of testing tensors. In Section 5, we introduce the TT-factorization STTA algorithm by Kressner et al. [6]. After presenting the sketching operators, in Section 5.2, we present the results of our numerical experiments on STTA. Finally, in Section 6, we outline the error analysis for STTA with Gaussian sketching matrices and HRHT.

The Julia code for the implementation and numerical experiments is available at `https://github.com/ferioliste/TensorTrainApproximation`.

2 Notation

The following notation and terminology will be used in this report. The notation is largely taken form [2], [6] and [7]. We use Matlab notation to denote entries, fibers and slices of tensors and matrices. We start by defining some mathematical objects.

- $[n]$ is the set $\{i \in \mathbb{N} : 1 \leq i \leq n\}$.
- $\mathbf{t}$ is a finite subset of $\mathbb{N}$

- $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \cdots n_d}$ is a tensor. d is the order of the tensor.
- $\mathbf{A} \in \mathbb{R}^{n \times m}$ is a matrix. $\mathbf{I}_k$ is the identity matrix of size $k \times k$.
- $\|\cdot\|_F$ denotes the Frobenius norm, defined as $\|\mathcal{T}\|_F = \sqrt{\sum_{i_1,\ldots,i_d} \mathcal{T}[i_1,\ldots,i_d]^2}$
- $\mathbb{1}_A(\cdot)$ is the indicator function
- $\operatorname{vec}\colon \mathbb{R}^{n\times m} \to \mathbb{R}^{nm}$ takes a matrix in $\mathbb{R}^{n\times m}$ and stacks its columns into a tall nm-vector

Let $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \cdots n_d}$, and $k \in [d]$. We define mode-k unfolding of $\mathcal{T}$ as the matrix $\mathcal{T}^{(k)}$ having size $n_k \times \left(\prod_{i\neq k} n_i\right)$:

$$\mathcal{T}^{(k)}[i_k; j] = \mathcal{T}^{(k)}[i_k; i_1, \ldots, i_{k-1}, i_{k+1}, i_d] = \mathcal{T}[i_1, \ldots, i_d],$$

where the index i_k enumerates the rows and the other indices enumerate the columns. Similarly, given any $\mathrm{s} \subset [d]$, we define the mode-s unfolding $\mathcal{T}^{(\mathrm{s})}$ using i_k for all $k \in \mathrm{s}$ to enumerate the rows, and the remaining indices to enumerate the columns. The matrix $\mathcal{T}^{(\mathrm{s})}$ has size $\left(\prod_{i\in\mathrm{s}} n_i\right) \times \left(\prod_{i\in[d]\setminus\mathrm{s}} n_i\right)$. In particular, we denote the mode-$(1\!:\!k)$ unfolding $\mathcal{T}^{[k]}$ as $\mathcal{T}^{\leq k}$. The matrix $\mathcal{T}^{\leq k}$ has size $\left(\prod_{i=1}^{k} n_i\right) \times \left(\prod_{i=k+1}^{d} n_i\right)$.

Let $\mathcal{T}_1 \in \mathbb{R}^{n_1 \times n_2 \cdots n_d}$, $\mathcal{T}_2 \in \mathbb{R}^{m_1 \times m_2 \cdots m_b}$, $k \in [d]$, and $j \in [b]$ such that $n_k = m_j$. The contraction product between $\mathcal{T}_1$ and $\mathcal{T}_2$ is the tensor $\mathcal{Y} = \mathcal{T}_1 \times_{kj} \mathcal{T}_2$ of size

$$n_1 \times \cdots \times n_{k-1} \times n_{k+1} \times \cdots \times n_d \times m_1 \times \cdots \times m_{j-1} \times m_{j+1} \times \cdots \times m_b,$$

defined as

$$\mathcal{Y}^{\leq d-1} = (\mathcal{T}_1^{(k)})^\top \mathcal{T}_2^{(j)}.$$

This definition can be extended to allow contractions on any number of dimensions at the same time, in this case we use the symbol $\times_{(k_1\ldots k_s)(j_1\ldots j_s)}$. Note that when contracting on multiple dimensions the order of $(k_1 \ldots k_s)$ and $(j_1 \ldots j_s)$ does matter. Finally, we use negative dimension indices to refer to dimensions starting from the last one, that is, if $k < 0$, use $d + k + 1$ instead.

3 Tensor Train Format

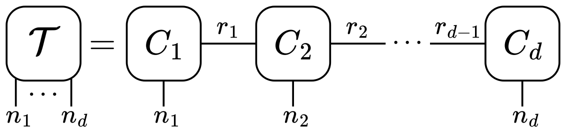
The idea behind the Tensor Train decomposition is to factorize a high-dimensional tensor into a sequence of contractions of three-dimensional tensors. Given a tensor $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \cdots n_d}$, we want to express it as

$$\mathcal{T} = C_1 \times_{(-1)(1)} C_2 \times_{(-1)(1)} \cdots C_{d-1} \times_{(-1)(1)} C_d, \tag{1}$$

where the third order tensors $C_\mu \in \mathbb{R}^{r_{\mu-1} \times n_\mu \times r_\mu}$ are called TT-cores. Compactly, we write $\mathcal{T} = C_1 \cdots C_d$. Note that $r_0 = r_d = 1$. The dimensions of the cores $r_1, r_2, ..., r_{d-1}$ are called TT-ranks and determine the complexity of working with a TT. The entrywise definition is the following:

$$\begin{aligned}\mathcal{T}[i_1, \ldots, i_d] &= C_1[1, i_1, :]C_2[:, i_2, :] \cdots C_d[:, i_d, 1] &\quad (2)\\ &= \sum_{\ell_1=1}^{r_1} \cdots \sum_{\ell_{d-1}=1}^{r_{d-1}} C_1[1, i_1, \ell_1] C_2[\ell_1, i_2, \ell_2] \cdots C_d[\ell_{d-1}, i_d, 1]. &\quad (3)\end{aligned}$$

The sequence of contractions in 1 can be represented using the following tensor diagram.



(4)

The TT representation of a tensor $\mathcal{T}$ is not unique. One could, for example, multiply one core by q and another one by $1/q$ and still get a valid representation on $\mathcal{T}$.

3.1 Factorization

Given any tensor $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \cdots n_d}$ we want to approximate it using a tensor train $\widetilde{\mathcal{T}}$. As a metric for the error we use the relative Frobenius error, $E = \|\mathcal{T} - \widetilde{\mathcal{T}}\|_F / \|\mathcal{T}\|_F$. To address this task, there are two possible approaches. We could set a tolerance for the error and find the best possible TT-ranks accordingly, or we could set the TT-ranks and find the best possible TT that achieves those ranks. In this project we focus on the second approach.

We start by presenting a revisited version of Oseledets' Algorithm 1 [7]. In order to adapt it to our chosen approach, in Algorithm 1, we replaced the rank-adaptive SVDs with fixed-rank SVDs. At every iteration of the for loop a new core is exctracted from the input tensor $\mathcal{T}$, until a tridimensional tensor remains.

Algorithm 1 Tensor Train Factorization using SVD

Input: Tensor $\mathcal{T}$ of size $n_1 \times \cdots \times n_d$, desired TT-ranks $r_1, r_2, \ldots, r_{d-1}$.
Output: Tensor train $\widetilde{\mathcal{T}} = C_1 \cdots C_d$ approximating $\mathcal{T}$, with TT-ranks $r_1, r_2, \ldots, r_{d-1}$.

```
1: $C \leftarrow \mathcal{T}$
2: for $\mu = 1$ to $d-1$ do
3:     $C \leftarrow \text{RESHAPE}(C, [r_{\mu-1} n_\mu, :])$
4:     $[\mathbf{U}, \mathbf{S}, \mathbf{V}] \leftarrow \text{SVD}(C)$
5:     $C_\mu \leftarrow \text{RESHAPE}(\mathbf{U}[:, 1\!:\!r_\mu], [r_{\mu-1}, n_\mu, r_\mu])$
6:     $C \leftarrow \mathbf{S}[1\!:\!r_\mu, 1\!:\!r_\mu]\mathbf{V}[:, 1\!:\!r_\mu]^\top$
7: end for
8: $C_d \leftarrow \text{RESHAPE}(C, [r_{d-1}, n_d, 1])$
```

Now, we want to prove that there is a value of r_μ above which there is no advantage in going and that this is $\text{RANK}\left(\mathcal{T}^{\leq \mu}\right)$ for all $1 \leq \mu \leq d-1$. The idea is to find r_μ such that we get an exact representation of the tensor, that is when do not truncate the SVDs.

We start with $\mu = 1$. We have that $C := \mathcal{T}^{\leq 1}$. Trivially, if we don't want to truncate its SVD, we have to pick $r_1 = \text{RANK}\left(\mathcal{T}^{\leq 1}\right)$.

For $\mu = 2$, we have $\tilde{C} := \text{RESHAPE}(C, [r_1 n_2, :])$. If we don't want to truncate its SVD, we have to pick

$$r_2 = \text{RANK}\left(\tilde{C}\right) = \text{RANK}\left((C_1^{\leq 2} \otimes \mathbf{I}_{n_2})\tilde{C}\right) = \text{RANK}\left(\mathcal{T}^{\leq 2}\right)$$

where the second equality holds because $(C_1^{\leq 1} \otimes \mathbf{I}_{n_2})$ is column-orthogonal, and the third holds by definition of C_1.

Similarly, for $\mu = 3$, we have $\tilde{\tilde{C}} := \text{RESHAPE}(C, [r_2 n_3, :])$ and

$$r_3 = \text{RANK}\left(\tilde{\tilde{C}}\right) = \text{RANK}\left((C_1^{\leq 2} \otimes \mathbf{I}_{n_2 n_3})(C_2^{\leq 2} \otimes \mathbf{I}_{n_3})\tilde{\tilde{C}}\right) = \text{RANK}\left(\mathcal{T}^{\leq 3}\right).$$

By induction, we use this argument for all $1 \leq \mu \leq d-1$.

We proved that if we choose $r_\mu = \text{RANK}\left(\mathcal{T}^{\leq \mu}\right)$ for all $1 \leq \mu \leq d-1$, we obtain an exact representation of the the tensor $\mathcal{T}$. This is an important property to keep in mind when working with TTs, as it demonstrates that the tensors that can be efficiently approximated are only those whose unfoldings have numerical low rank. In general, this happens with sparse tensors. However there exist dense examples, such as the Hilbert tensor and the square-root-sum tensor, which we define and use later in our numerical experiments.

Since given a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ we have $\text{RANK}(\mathbf{A}) \leq \min\{n, m\}$, in the TT case we have

$$r_\mu \leq \text{RANK}\left(\mathcal{T}^{\leq \mu}\right) \leq \min\left\{\prod_{i=1}^{\mu} n_i, \prod_{i=\mu+1}^{d} n_i\right\},$$

This relation allows us to define the function FIX_RANKS:

$$\tilde{r} = \text{FIX\_RANKS}\,(r; n)\,, \quad \text{with} \quad \tilde{r}_\mu = \min\left\{ r_\mu, \prod_{i=1}^{\mu} n_i, \prod_{i=\mu+1}^{d} n_i \right\},$$

that we will use in the following to cap target ranks that are too high.

3.2 Addition and rounding

Consider two tensor trains, $\mathcal{T}_1 = A_1 \cdots A_d$ and $\mathcal{T}_2 = B_1 \cdots B_d$, with the same dimensions. Our objective is to compute $\mathcal{Y} = \mathcal{T}_1 + \mathcal{T}_2$ without expanding any tensor into full-tensor representation. The slices of the TT-cores of $\mathcal{Y}$ are the following matrices.

$$\begin{aligned}
C_1[1, i_1, :] &= \begin{pmatrix} A_1[1, i_1, :] & B_1[1, i_1, :] \end{pmatrix} && \text{for} \quad 1 \le i_1 \le n_1 \\
C_\mu[:, i_\mu, :] &= \begin{pmatrix} A_\mu[:, i_\mu, :] & 0 \\ 0 & B_\mu[:, i_\mu, :] \end{pmatrix} && \text{for} \quad 2 \le \mu \le d-1, \quad 1 \le i_\mu \le n_\mu \\
C_d[:, i_d, 1] &= \begin{pmatrix} A_d[:, i_d, 1] \\ B_d[:, i_d, 1] \end{pmatrix} && \text{for} \quad 1 \le i_d \le n_d.
\end{aligned}$$

Indeed, using Definition 2, we have

$$\begin{aligned}
\mathcal{Y}[i_1, \ldots, i_d] &= C_1[1, i_1, :] C_2[:, i_2, :] \cdots C_d[:, i_d, 1] \\
&= \begin{pmatrix} A_1[1, i_1, :] & B_1[1, i_1, :] \end{pmatrix} \begin{pmatrix} A_2[:, i_2, :] & 0 \\ 0 & B_2[:, i_2, :] \end{pmatrix} \cdots \begin{pmatrix} A_d[:, i_d, 1] \\ B_d[:, i_d, 1] \end{pmatrix} \\
&= \begin{pmatrix} A_1[1, i_1, :] A_2[:, i_2, :] & B_1[1, i_1, :] B_2[:, i_2, :] \end{pmatrix} \begin{pmatrix} A_3[:, i_3, :] & 0 \\ 0 & B_3[:, i_3, :] \end{pmatrix} \cdots \begin{pmatrix} A_d[:, i_d, 1] \\ B_d[:, i_d, 1] \end{pmatrix} \\
&= \cdots = A_1[1, i_1, :] A_2[:, i_2, :] \cdots A_d[:, i_d, 1] + B_1[1, i_1, :] B_2[:, i_2, :] \cdots B_d[:, i_d, 1]
\end{aligned}$$

Although the algorithm is straightforward, it has a significant disadvantage: when summing two TTs, their ranks are added together. This leads to an increase of the TT-ranks. For this reason, we need a rounding algorithm, a procedure to reduce the ranks of a tensor train. We present Algorithm 2, which is a revisited version of Oseledets' Algorithm 2 [7]. In the first step, the tensor train is left-orthogonalized, that is the "horizontal" unfoldings of the cores are made row-orthogonal. In the second step, we have the actual compression.

4 Introduction to randomized algorithms

In this section, we start exploring how randomization can be used to improve the runtime performance of TT-factorization and TT-rounding. We present three randomized algorithms, one for TT-factorization and two for TT-rounding.

The first two algorithms are very similar to Algorithms 1 and 2, respectively. The only difference is that we replace the truncated SVDs with a basic version of the randomized range finder (rSVD) presented by Halko et al. in [5]. We recall the procedure for the randomized range finder in Algorithm 3. The corresponding algorithms, 4 and 5, are presented below.

The third randomized algorithm we present is taken from [1, Algorithm 3.1] and utilizes a completely different approach. We start by defining the notion of random Gaussian TT [1, Definition 3.1], before moving to the actual algorithm.

Definition 4.1 (random Gaussian TT). Given a set of dimensions $n_1, \ldots, n_d$ and a set of ranks $r_1, \ldots, r_{d-1}$, we define a random Gaussian TT, $\mathcal{G} = B_1 \cdots B_d$, where the entries of B_μ are iid Gaussian with mean 0 and variance $1/(r_{\mu-1} n_\mu r_\mu)$ for $1 \le \mu \le d$.

Algorithm 2 Tensor Train Rounding using SVD

Input: Tensor train $\mathcal{T} = C_1 \cdots C_d$ with TT-ranks $r_1, r_2, \ldots, r_{d-1}$, desired TT-ranks $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_{d-1}$.
Output: Tensor train $\widetilde{\mathcal{T}} = \widetilde{C}_1 \cdots \widetilde{C}_d$ approximating $\mathcal{T}$ with TT-ranks $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_{d-1}$.

```
 1: $\widetilde{C}_d \leftarrow C_d$
 2: for $\mu = d$ to $2$ do
 3:     $[\mathbf{Q}, \mathbf{R}] \leftarrow \textsc{qr}\left(\textsc{reshape}\left(\widetilde{C}_\mu, [r_{\mu-1}, n_\mu r_\mu]\right)^\top\right)$
 4:     $\widetilde{C}_\mu \leftarrow \textsc{reshape}\left(\mathbf{Q}^\top, [r_{\mu-1}, n_\mu, r_\mu]\right)$
 5:     $\widetilde{C}_{\mu-1} \leftarrow C_{\mu-1} \times_{31} \mathbf{R}^\top$
 6: end for
 7: for $\mu = 1$ to $d-1$ do
 8:     $[\mathbf{U}, \mathbf{S}, \mathbf{V}] \leftarrow \textsc{svd}\left(\textsc{reshape}\left(\widetilde{C}_\mu, [\tilde{r}_{\mu-1} n_\mu, r_\mu]\right)\right)$
 9:     $\widetilde{C}_\mu \leftarrow \textsc{reshape}\left(\mathbf{U}[:, 1{:}\tilde{r}_\mu], [\tilde{r}_{\mu-1}, n_\mu, \tilde{r}_\mu]\right)$
10:     $\widetilde{C}_{\mu+1} \leftarrow \widetilde{C}_{\mu+1} \times_{11} (\mathbf{S}[1{:}\tilde{r}_\mu, 1{:}\tilde{r}_\mu]\mathbf{V}[:, 1{:}\tilde{r}_\mu]^\top)$
11: end for
```

Algorithm 3 Randomized range finder (rSVD)

Input: Matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, target rank r
Output: Matrices $\mathbf{X} \in \mathbb{R}^{m \times r}$ and $\mathbf{Y} \in \mathbb{R}^{r \times n}$ such that $\mathbf{XY} \approx \mathbf{A}$

```
1: Generate Gaussian matrix $\Phi$ of shape $n \times r$
2: $[\mathbf{X}, \sim] \leftarrow \textsc{qr}\,(A\,\Phi)$
3: $\mathbf{Y} \leftarrow \mathbf{X}^\top \mathbf{A}$
```

Algorithm 4 Tensor Train Factorization using rSVD

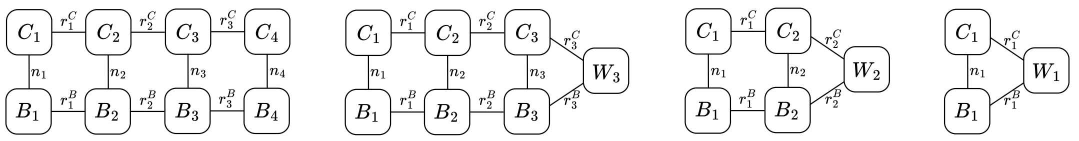
Input: Tensor $\mathcal{T}$ of size $n_1 \times \cdots \times n_d$, desired tt-ranks $r_1, r_2, \ldots, r_{d-1}$.
Output: Cores C_μ of shape $r_{\mu-1} \times n_\mu \times r_\mu$ for $1 \leq \mu \leq d$ of a tensor train approximation of $\mathcal{T}$. By convention, $r_0 = r_d = 1$.

```
1: Generate independent Gaussian matrices $\Phi_\mu$ of shape $(n_{\mu+1} \cdots n_d) \times r_\mu$ for $1 \leq \mu \leq d-1$
2: $C \leftarrow \mathcal{T}$
3: for $\mu = 1$ to $d-1$ do
4:     $C \leftarrow \textsc{reshape}\,(C, [r_{\mu-1} n_\mu, :])$
5:     $[\mathbf{X}, \sim] \leftarrow \textsc{qr}\,(C\,\Phi_\mu)$
6:     $C_\mu \leftarrow \textsc{reshape}\,(\mathbf{X}, [r_{\mu-1}, n_\mu, r_\mu])$
7:     $C \leftarrow \mathbf{X}^\top C$
8: end for
9: $C_d \leftarrow C$
```

Algorithm 5 Tensor Train Rounding using rSVD

```
Input: Tensor train $\mathcal{T} = C_1 \cdots C_d$ with TT-ranks $r_1, r_2, \dots, r_{d-1}$, desired TT-ranks $\tilde{r}_1, \tilde{r}_2, \dots, \tilde{r}_{d-1}$.
Output: Tensor train $\widetilde{\mathcal{T}} = \widetilde{C}_1 \cdots \widetilde{C}_d$ approximating $\mathcal{T}$ with TT-ranks $\tilde{r}_1, \tilde{r}_2, \dots, \tilde{r}_{d-1}$.
 1: $\widetilde{C}_d \leftarrow C_d$
 2: for $\mu = d$ to $2$ do
 3:     $[\mathbf{Q}, \mathbf{R}] \leftarrow \mathrm{QR}\left(\mathrm{RESHAPE}\left(\widetilde{C}_\mu, [r_{\mu-1}, n_\mu r_\mu]\right)^\top\right)$
 4:     $\widetilde{C}_\mu \leftarrow \mathrm{RESHAPE}\left(\mathbf{Q}^\top, [r_{\mu-1}, n_\mu, r_\mu]\right)$
 5:     $\widetilde{C}_{\mu-1} \leftarrow C_{\mu-1} \times_{31} \mathbf{R}^\top$
 6: end for
 7: Generate independent Gaussian matrices $\Phi_\mu$ of shape $r_\mu \times \tilde{r}_\mu$ for $1 \leq \mu \leq d-1$
 8: $C \leftarrow \widetilde{C}_1$
 9: for $\mu = 1$ to $d-1$ do
10:     $C \leftarrow \mathrm{RESHAPE}\left(C, [\tilde{r}_{\mu-1} n_\mu, r_\mu]\right)$
11:     $[\mathbf{X}, \sim] \leftarrow \mathrm{QR}\left(C\, \Phi_\mu\right)$
12:     $\widetilde{C}_\mu \leftarrow \mathrm{RESHAPE}\left(\mathbf{X}, [\tilde{r}_{\mu-1}, n_\mu, \tilde{r}_\mu]\right)$
13:     $C \leftarrow (\mathbf{X}^\top C)\ \mathrm{RESHAPE}\left(\widetilde{C}_{\mu+1}, [r_\mu n_{\mu+1}, r_{\mu+1}]\right)$
14: end for
15: $\widetilde{C}_d \leftarrow \mathrm{RESHAPE}\left(C, [\tilde{r}_{d-1}, n_d, 1]\right)$
```

In a first phase of the algorithm, we generate a random Gaussian TT $\mathcal{G} = B_1 \cdots B_d$ with ranks $\tilde{r}_1, \dots, \tilde{r}_{d-1}$. Then, we build the so-called partial contractions of $\mathcal{G}$ with $\mathcal{T} = C_1 \cdots C_d$ using the following scheme (here for $d = 4$).



(5)

that is defining $\mathbf{W}_{d-1} = C_d \times_{(2,3)(2,3)} B_d$ and then, recursively, $\mathbf{W}_{\mu-1} = (C_\mu \times_{31} \mathbf{W}_\mu) \times_{(2,3)(2,3)} B_\mu$.

After this first randomization step, starting with $\mu = 1$, we compute the QR factorization of the sketched matrix

$$C\, C_{>\mu} B_{>\mu}^\top = C\, \mathbf{W}_\mu,$$

where C is initialized as C_1 and then updated as in Algorithm 5. The corresponding Algorithm 6 is presented below. It's important to note that the tensor train produced by Algorithm 6 is left-orthogonal. This property is particularly useful if further rounding is necessary. In such cases, one could use, for example, Algorithm 2 and skip the orthogonalization phase (first loop).

4.1 Numerical Experiments

In this section, we make a numerical comparison of the algorithms presented. For the factorization algorithms, we test the performance on two dense tensors of size $n_1 \times \cdots \times n_d$ with $n_1 = \cdots = n_d = n$. As in [6], we use the Hilbert tensor

$$\mathcal{T}_{\mathsf{Hilbert}}[i_1, \dots, i_d] = (i_1 + \cdots + i_d - d + 1)^{-1}$$

and the square-root-sum tensor

$$\mathcal{T}_{\mathsf{sqrt}}[i_1, \dots, i_d] = \sqrt{\left|\sum_{j=1}^{d} \left(\frac{n_j - i_j}{n_j - 1} a + \frac{i_j - 1}{n_j - 1} b\right)\right|},$$

with $a = -0.2$ and $b = 2$. As proved in [4], the singular values of the unfoldings $\mathcal{T}^{\leq \mu}$ of both tensors have exponential decay. Thus , they can be efficiently represented as tensor trains, as shown in Section 3.1.

Algorithm 6 TT-rounding: randomize-then-orthogonalize

Input: Tensor train $\mathcal{T} = C_1 \cdots C_d$ with TT-ranks $r_1, r_2, \ldots, r_{d-1}$, desired TT-ranks $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_{d-1}$.
Output: Tensor train $\widetilde{\mathcal{T}} = \widetilde{C}_1 \cdots \widetilde{C}_d$ approximating $\mathcal{T}$ with TT-ranks $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_{d-1}$.

```
1: Generate Gaussian tensor train $\mathcal{G} = B_1 \cdots B_d$ with ranks $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_{d-1}$
2: $\mathbf{W}_{d-1} \leftarrow C_d \times_{(2,3)(2,3)} B_d$
3: for $\mu = d-1$ to $2$ do
4:     $\mathbf{W}_{\mu-1} \leftarrow (C_\mu \times_{31} \mathbf{W}_\mu) \times_{(2,3)(2,3)} B_\mu$
5: end for
6: $C \leftarrow C_1$
7: for $\mu = 1$ to $d-1$ do
8:     $C \leftarrow \mathrm{RESHAPE}\,(C, [\tilde{r}_{\mu-1} n_\mu,\, r_\mu])$
9:     $[\mathbf{X}, \sim] \leftarrow \mathrm{QR}\,(C\, \mathbf{W}_\mu)$
10:    $\widetilde{C}_\mu \leftarrow \mathrm{RESHAPE}\,(\mathbf{X}, [\tilde{r}_{\mu-1},\, n_\mu,\, \tilde{r}_\mu])$
11:    $C \leftarrow (\mathbf{X}^\top C)\, \mathrm{RESHAPE}\,(C_{\mu+1}, [r_\mu,\, n_{\mu+1} r_{\mu+1}])$
12: end for
13: $\widetilde{C}_d \leftarrow \mathrm{RESHAPE}\,(C, [\tilde{r}_{d-1},\, n_\mu,\, 1])$
```

For rounding algorithms, we consider two dense tensor trains. The first one has size $n_1 \times \cdots \times n_d$ with $n_1 = \cdots = n_d = n$ and it is defined as a factorized version of the Hilbert tensor. We obtain it by using Algorithm 1 with TT-ranks equal to $\mathrm{FIX\_RANKS}\,(20, \ldots, 20;\, n_1, \ldots, n_d)$. Similarly to [1], the second tensor train has size $2n_1 \times \cdots \times 2n_d$ with $n_1 = \cdots = n_d = n$ and is obtained perturbating a random TT $\mathcal{T}_1$ using another TT $\mathcal{T}_2$ as $\mathcal{T}_{\text{sum}} = \mathcal{T}_1 + \varepsilon \mathcal{T}_2$. Both random TTs have size $n_1 \times \cdots \times n_d$, TT-ranks $\mathrm{FIX\_RANKS}\,(20, \ldots, 20;\, n_1, \ldots, n_d)$ and their cores' entries are iid standard Gaussian.

4.1.1 TT-factorization

We present the results for TT-factorization focussing on benchmarking the runtime and the error performance. Initially, we considered evaluating memory usage as well, but this proved infeasible due to the complexity of tracking memory usage across the multiple external packages that we use in our Julia implementation.

The tests were conducted varying three parameters: the tensor size (n), the number of dimensions (d) and the target TT-rank (r). Given r the target TT-ranks $r_1, \ldots, r_{d-1}$ are obtained as $\mathrm{FIX\_RANKS}\,(r, \ldots, r;\, n, \ldots, n)$. We set $n = 5$, $d = 5$ and $r = 10$ as the default case, then we vary each parameter while keeping the other two fixed.

Regarding the measurement of the runtime, not only we timed the total runtime of the factorisation algorithms, but also the time taken by different phases of the algorithms. Concerning the measurement of error, we computed $E = \|\mathcal{T} - \widetilde{\mathcal{T}}\|_F / \|\mathcal{T}\|_F$ by contracting the tensor train $\widetilde{\mathcal{T}}$ back to full-tensor format and applying the formula directly.

All the tests were run on a IdeaPad L340 Gaming with the specifics in Table 1. We performed 10 trials per parameters set and we executed the tests in random order to reduce the correlation bias.

Model Name	IdeaPad L340 Gaming
CPU	Intel i7-9750HF
Number of Cores	4
Memory	16 GB
System type	64-bit

Table 1: System Specifics

In Figure 1, we present the results regarding the error performance. Overall, the randomized algorithm exhibits a behaviour very similar to the deterministic one. The only notable exception is observed with $\mathcal{T}_{\text{Hilbert}}$ when varying the number of dimension d, where the TT-rSVD algorithm outperforms TT-SVD in terms of

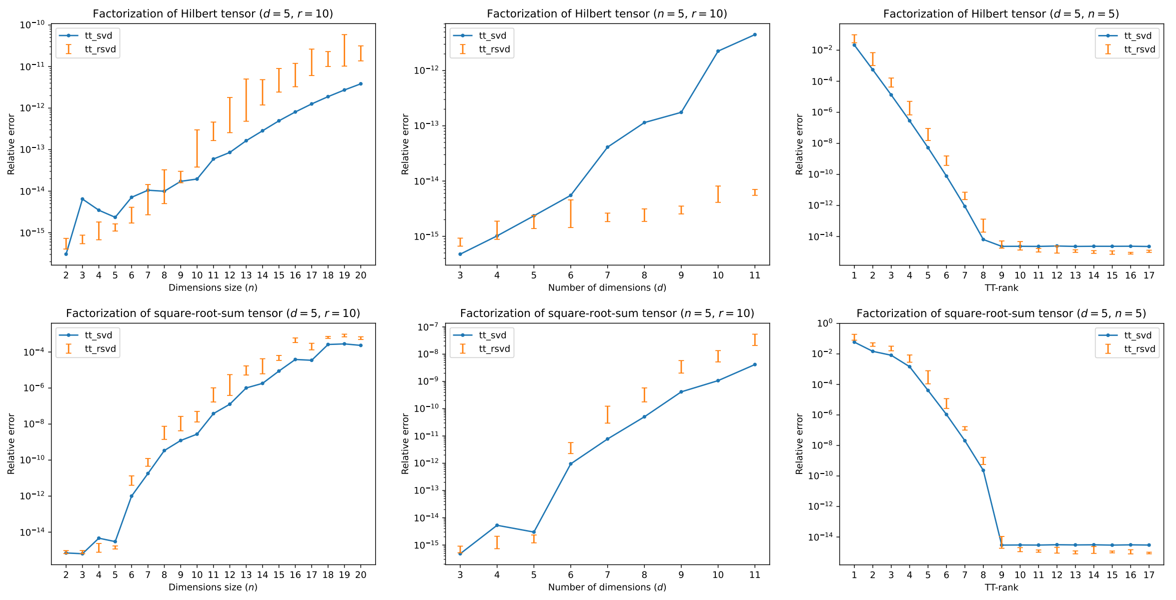


Figure 1: Error performance of TT-SVD 1 and TT-rSVD 4 on $\mathcal{T}_{\text{Hilbert}}$ and $\mathcal{T}_{\text{sqrt}}$ (1-st and 2-nd row, respectively), when varying the tensor size n, the number of dimensions d and the target TT-rank r (1-st, 2-nd, and 3-rd column, respectively)

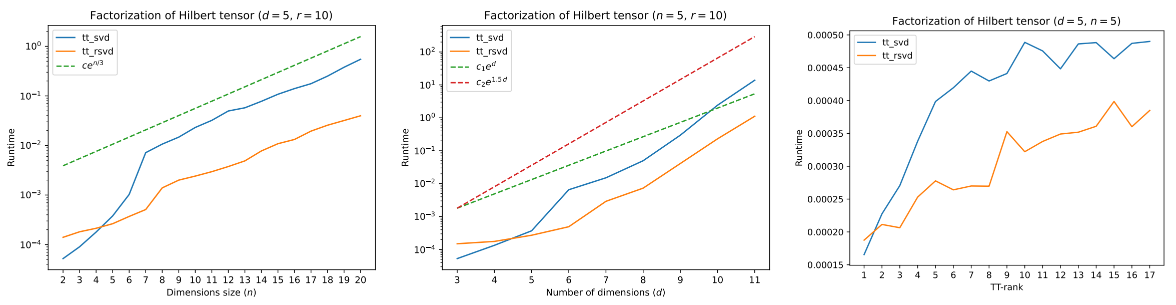


Figure 2: Runtime performance of TT-SVD 1 and TT-rSVD 4 on $\mathcal{T}_{\text{Hilbert}}$, when varying the tensor size n, the number of dimensions d and the target TT-rank r (1-st, 2-nd, and 3-rd plot, respectively)

error. Additionally, when examining the error performance varying the target rank r, it becomes evident why these tensors can be efficiently represented as tensor trains: the relative error decreases exponentially when r increases. Finally, we analyze the performance when varying the tensor size n and the number of dimensions d. In both cases, the error seems to grow exponentially. This trend can be explained by noticing that an effective value for r does increase if we increase n or d. Here, since r is held constant, we have a growth in the error.

In Figure 2, we present the runtime performance results, focusing solely on $\mathcal{T}_{\text{Hilbert}}$ as the runtime is independent of the testing tensor type. We observe that the two algorithms have the same asymptotic behavior, with the randomized algorithm generally performing faster.

Empirically, the runtime increases approximately as $e^{n/3}$ when varying r and as $e^{\frac{3}{2}d}$ when varying d. The runtime does not exhibit a strong dependence on the target rank r.

Finally, in Figure 3, we analyze how the runtime is divided among the different phases of the algorithms. For TT-SVD, we observe that computing the SVDs take up the majority of the runtime, while the matrix multiplication at **Line 6** in Algorithm 1 uses an overall negligible amount of time. In the case of TT-rSVD, nearly half of the runtime is spent on the sketching operation $C\Phi_\mu$ at **Line 5** in Algorithm 4. The rest of the runtime is uniformly divided among QR factorization (**Line 5**, only QR) and the matrix multiplication (**Line 7**). However, as n or d increase, the time required for the QR factorization becomes negligible.

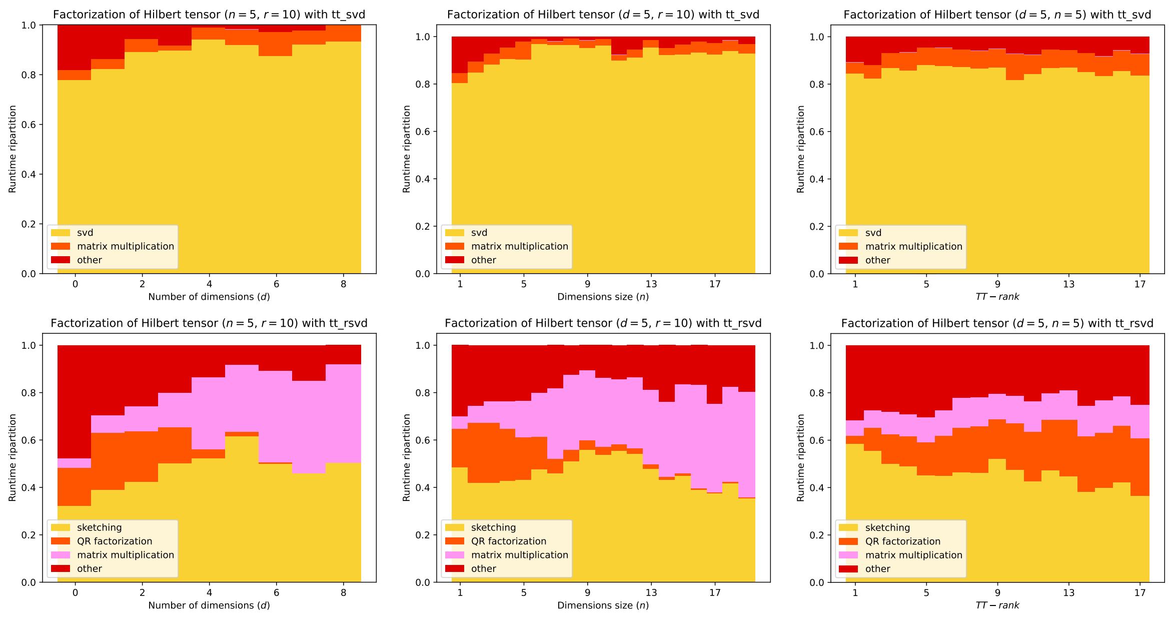


Figure 3: Runtime partitioning of TT-SVD 1 and TT-rSVD 4 (1-st and 2-nd row, respectively) on $\mathcal{T}_{\text{Hilbert}}$, when varying the tensor size n, the number of dimensions d and the target TT-rank r (1-st, 2-nd, and 3-rd column, respectively)

4.1.2 TT-rounding

Similarly to TT-factorization, for TT-rounding we focus on the runtime and the error performance. For $\mathcal{T}_{\text{Hilbert}}$, the tests were conducted varying the parameters n, d and r. On the other hand, for $\mathcal{T}_{\text{sum}}$, the tests were executed changing n, d and ε, fixing $r = 20$.

Again, in addition to the total runtime, we separately timed the different parts of the algorithms. Regarding the measurement of error, we computed $E = \|\mathcal{T} - \widetilde{\mathcal{T}}\|_F / \|\mathcal{T}\|_F$ by contracting both tensor trains $\mathcal{T}$ and $\widetilde{\mathcal{T}}$ back to full-tensor format and applying the formula directly. This is particularly important for $\mathcal{T}_{\text{Hilbert}}$, since we don't want the error introduced by the first factorization to be taken into account.

We performed 10 trials per parameters set and we executed the tests in random order to reduce the correlation bias.

In Figure 4, we present the error performance results of the three rounding algorithms. Overall, the randomized algorithm exhibits a similar behaviour to the deterministic one. The error for the randomized algorithms is generally of the same order of magnitute as for TT-SVD-rounding. As for factorization, increasing n causes an increas in the error for $\mathcal{T}_{\text{Hilbert}}$. On the other hand, the error does not seem to have a strong dependence on d for both tensors. The error also depends weakly on n and ε when considering $\mathcal{T}_{\text{sum}}$. Finally, similarly to factorization, for $\mathcal{T}_{\text{Hilbert}}$, the error decreases exponentially when the target TT-rank r increases.

In Figure 5, we present the results regarding the runtime performance. The runtime appears to increase linearly as n or d increase. The randomize-then-Orthogonalize algorithm exhibits the smallest slope and the shortest runtime overall. On the other hand, there doesn't seem to be a signification advantage in using rSVD instead of normal SVD. Additionally, the runtime doesn't show a strong dependence on the target rank r or the parameter ε.

Finally, in Figure 6, we observe how the runtime is divided among the different phases of the algorithms. As for the total runtime, TT-SVD-rounding and TT-rSVD-rounding show a similar behaviour. As for Randomize-then-orthogonalize the runtime is almost evenly divided between the generation of the partial contraction and the final orthogonalization.

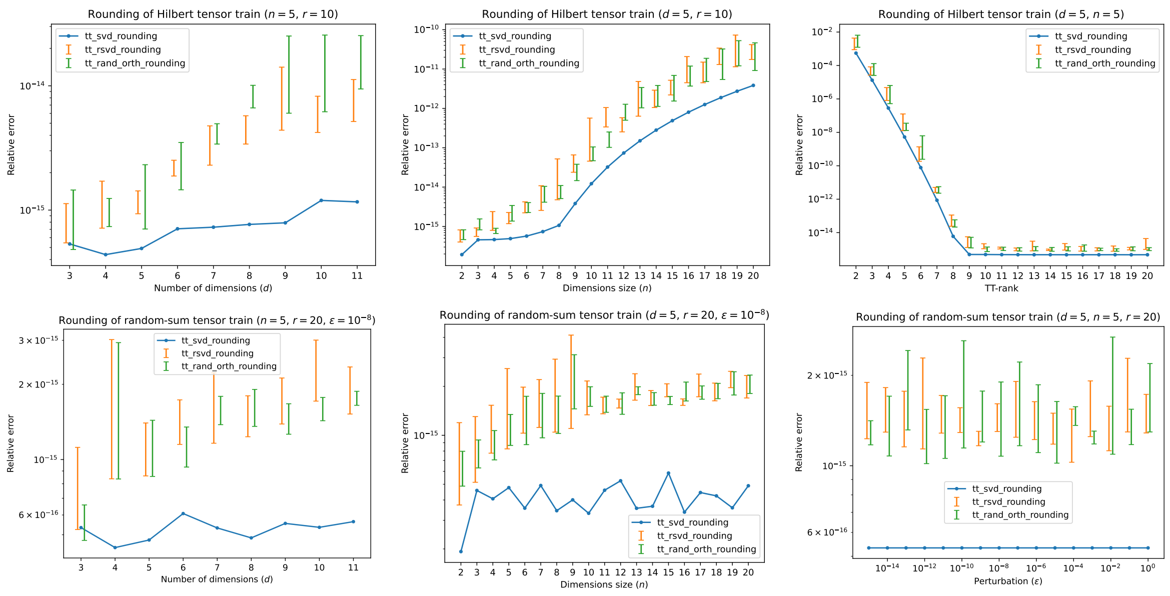


Figure 4: Error performance of TT-SVD-rounding 2 and TT-rSVD-rounding 5 on $\mathcal{T}_{\text{Hilbert}}$ and $\mathcal{T}_{\text{sum}}$ (1-st and 2-nd row, respectively), when varying the tensor size n, the number of dimensions d and the target TT-rank r or the perturbation factor ε (1-st, 2-nd, and 3-rd column, respectively)

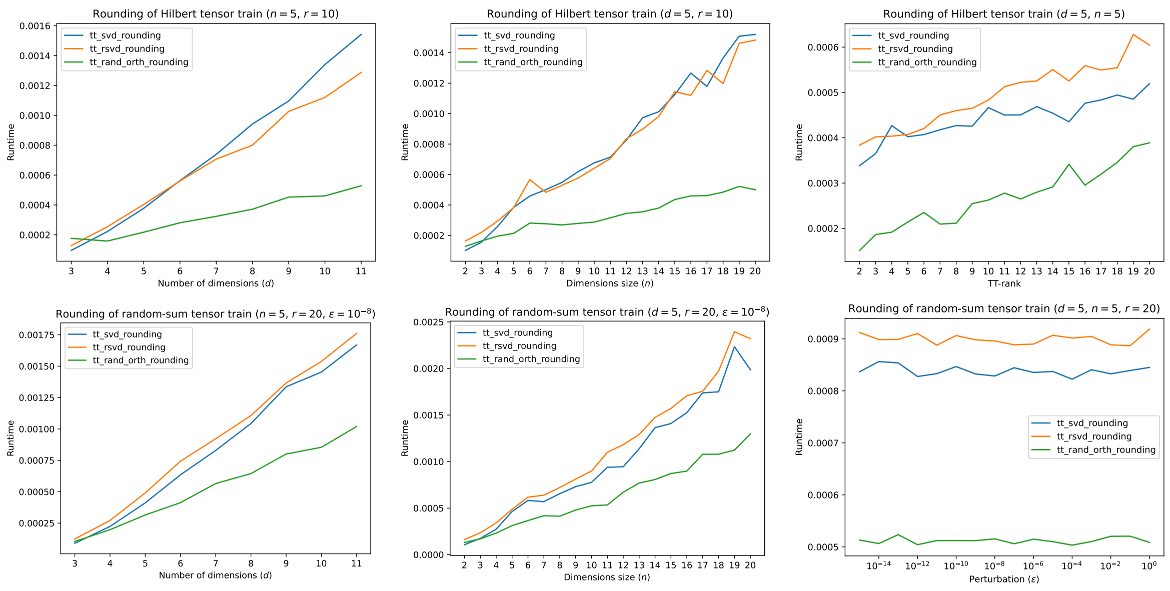


Figure 5: Runtime performance of TT-SVD-rounding 2 and TT-rSVD-rounding 5 on $\mathcal{T}_{\text{Hilbert}}$ and $\mathcal{T}_{\text{sum}}$ (1-st and 2-nd row, respectively), when varying the tensor size n, the number of dimensions d and the target TT-rank r or the perturbation factor ε (1-st, 2-nd, and 3-rd column, respectively)

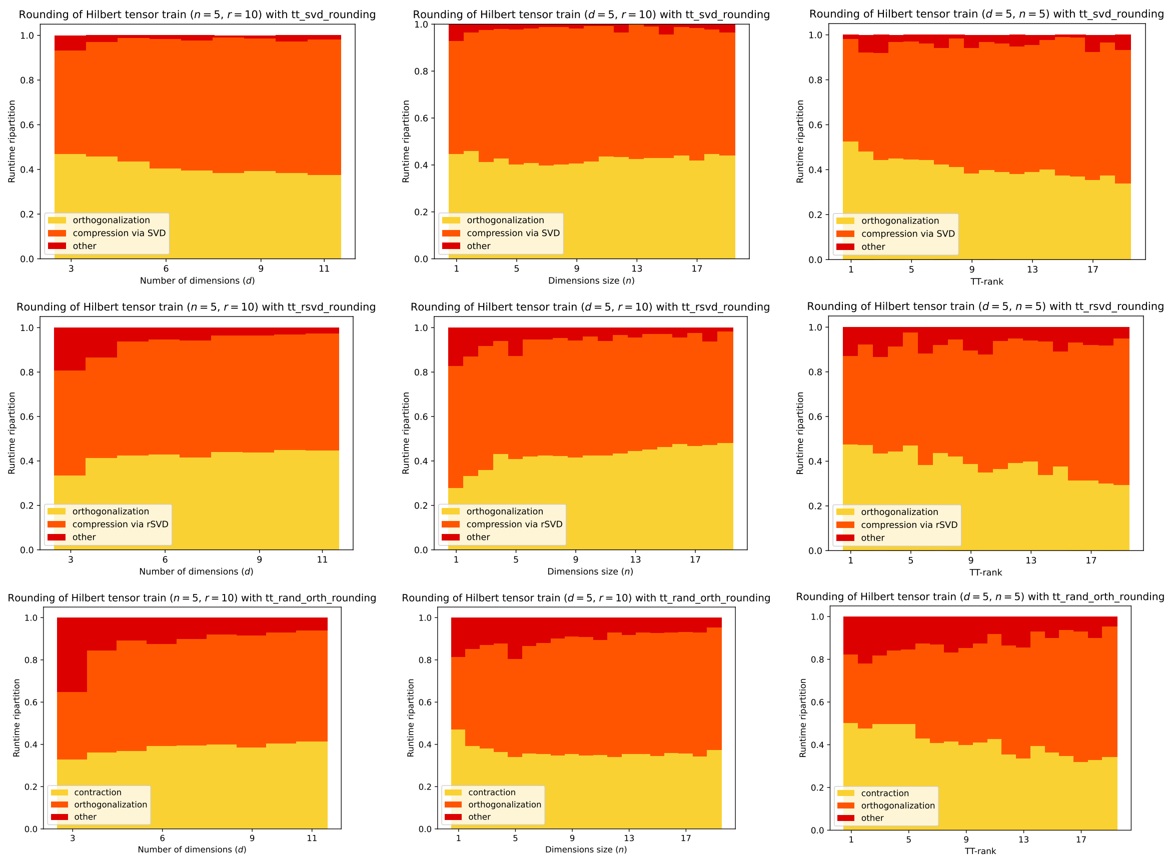


Figure 6: Runtime partitioning of TT-SVD-rounding 2, TT-rSVD-rounding 5, and Randomize-then-orthogonalize (1-st, 2-nd, and 3-rd row, respectively) on $\mathcal{T}_{\text{Hilbert}}$, when varying the tensor size n, the number of dimensions d and the target TT-rank r (1-st, 2-nd, and 3-rd column, respectively)

5 The STTA algorithm

In this section we present the Streaming Tensor Train Approximation (STTA) algorithm presented by Kressner et al. in [6]. We focus on the factorization algorithm, although it is possible to generalize it to rounding.

The idea of STTA is to build sketches of the unfoldings $\mathcal{T}^{\leq\mu}$ of the tensor $\mathcal{T}$. To do this, we use several sketching matrices

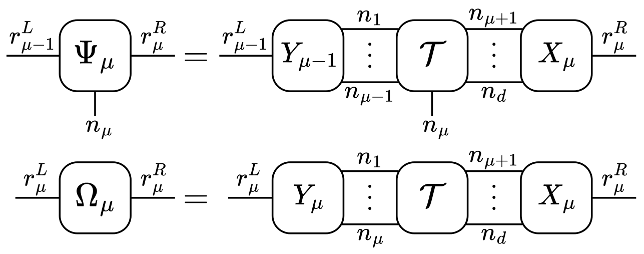
- $\mathbf{X}_\mu$ of shape $(n_{\mu+1}\cdots n_d) \times r_\mu^R$ for $1 \leq \mu \leq d-1$
- $\mathbf{Y}_\mu$ of shape $(n_1 \cdots n_\mu) \times r_\mu^L$ for $1 \leq \mu \leq d-1$

where we require either $r_\mu^R < r_\mu^L - 1$ for all $1 \leq \mu \leq d-1$ or $r_\mu^L < r_\mu^R - 1$ for all $1 \leq \mu \leq d-1$. The TT obtained using STTA will have TT-ranks $r_\mu = \min\{r_\mu^L, r_\mu^R\}$ for $1 \leq \mu \leq d-1$.

The sketches are constructed as follows.

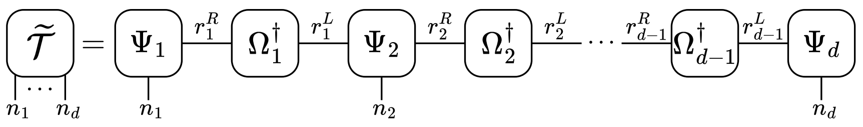
$$\Psi_\mu^{\leq 2} = (\mathbf{Y}_{\mu-1}^\top \otimes \mathbf{I}_{n_\mu})\mathcal{T}^{\leq\mu}\mathbf{X}_\mu \text{ with } \Psi_\mu \text{ having size } r_{\mu-1}^L \times n_\mu \times r_\mu^R \text{ for } 1 \leq \mu \leq d.$$
$$\Omega_\mu = \mathbf{Y}_\mu^\top C \mathbf{X}_\mu \text{ having size } r_\mu^L \times r_\mu^R \text{ for } 1 \leq \mu \leq d-1.$$

Here we use the convention $\mathbf{Y}_0 = \mathbf{X}_d = 1$. In practice, we use Algorithm 7 to compute all the sketches. Using tensor diagrams, this sketching operation can be represented as follows.



(6)

Concatenating these sketches, we can build an approximation of $\mathcal{T}$ as follows.

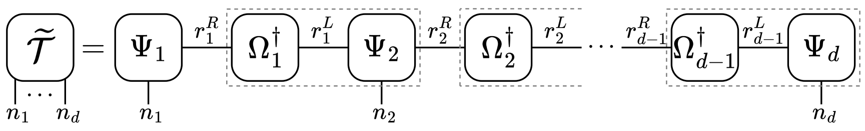


(7)

From here, to obtain a tensor train, we just need to contract each Ω_μ with an adjacent tensor $\Psi_{\hat{\mu}}$. Clearly, if $r^R_\mu < r^L_\mu$ for all $1 \leq \mu \leq d-1$ it will be preferable to contract each Ω_μ with the tensor on it right. As shown also in Algorithm 9, we can then define the cores of $\widetilde{\mathcal{T}}$ as

$$C_1 = \Psi_1,$$
$$C_\mu = \Omega^\dagger_{\mu-1} \times_{21} \Psi_\mu \quad \text{for } 2 \leq \mu \leq d.$$

Using tensor diagrams, this can be represented as follows.

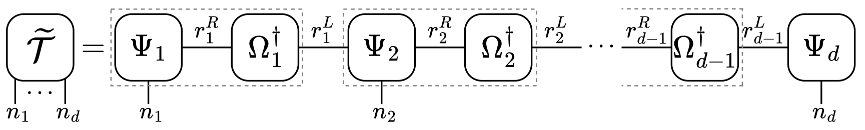


(8)

On the other hand, if $r^L_\mu < r^R_\mu$ for all $1 \leq \mu \leq d-1$, we define the cores of $\widetilde{\mathcal{T}}$ as

$$C_\mu = \Psi_\mu \times_{31} \Omega^\dagger_\mu \quad \text{for } 1 \leq \mu \leq d-1,$$
$$C_d = \Psi_d.$$

Using tensor diagrams, this can be represented as follows.



(9)

In both cases, special care must be taken when computing the pseudoinverse of Ω_μ, especially if this is badly conditioned. Following the same approach as in [6], we use Algorithm 8 which calculates the pseudoinverse of Ω_μ by computing its SVD and discarding all the singular values smaller than $\varepsilon_{\text{mach}}\|\Omega_\mu\|_2$, where $\varepsilon_{\text{mach}}$ is the machine precision.

Regarding the sketching matrices, we noticed that it is not always true that $\mathbf{X}_\mu$ and $\mathbf{Y}_\mu$ are tall and skinny matrices. Indeed, there can be short and fat sketching matrices, although we expect the majority of them to be tall and skinny. This translates into the fact that sometimes we "sketch on a higher dimension", which could cause some issues with specific sketching matrices, as we see in the following.

In addition, we highlight the fact that the approximation $\widetilde{\mathcal{T}}$ obtained via STTA algorithms does not depend on the normalization constants of the sketching matrices. This follows trivially from the generalized Niström method

$$\widehat{\mathbf{A}} = \mathbf{A}\mathbf{X}(\mathbf{Y}^\top\mathbf{A}\mathbf{X})^\dagger\mathbf{Y}^\top\mathbf{A}$$

that we use in Section 6 to prove the error bound for STTA.

Algorithm 7 STTA: sketch

Input: Tensor $\mathcal{T}$ of size $n_1 \times \cdots \times n_d$, left TT-ranks $r_1^L, \ldots, r_{d-1}^L$, right TT-ranks $r_1^R, \ldots, r_{d-1}^R$.
Output: Sketches Ψ_μ of size $r_{\mu-1}^L \times n_\mu \times r_\mu^R$ for $1 \le \mu \le d$, sketches Ω_μ of size $r_\mu^L \times r_\mu^R$ for $1 \le \mu \le d-1$

```
 1: Generate independent Gaussian* matrices $\mathbf{X}_\mu$ of shape $(n_{\mu+1} \cdots n_d) \times r_\mu^R$ for $1 \le \mu \le d-1$
 2: Generate independent Gaussian* matrices $\mathbf{Y}_\mu$ of shape $(n_1 \cdots n_\mu) \times r_\mu^L$ for $1 \le \mu \le d-1$
 3: for $\mu = 1$ to $d-1$ do
 4:     $C \leftarrow \mathcal{T}^{\le \mu}$
 5:     if $\mu = 1$ then
 6:         $\mathbf{Z} \leftarrow C\mathbf{X}_1$
 7:         $\Psi_\mu \leftarrow \text{RESHAPE}\left(\mathbf{Z}, \left[1, n_1, r_1^R\right]\right)$
 8:     else
 9:         $\mathbf{Z} \leftarrow C\mathbf{X}_\mu$
10:         $\mathbf{Z} \leftarrow \mathbf{Y}_{\mu-1}^\top \text{RESHAPE}\left(\mathbf{Z}, \left[\prod_{i=1}^{\mu-1} n_i, :\right]\right)$
11:         $\Psi_\mu \leftarrow \text{RESHAPE}\left(\mathbf{Z}, \left[r_{\mu-1}^L, n_\mu, r_\mu^R\right]\right)$
12:     end if
13:     $\Omega_\mu \leftarrow \mathbf{Y}_\mu^\top C \mathbf{X}_\mu$
14: end for
15: $\mathbf{Z} = \mathbf{Y}_{d-1}^\top C$
16: $\Psi_d \leftarrow \text{RESHAPE}\left(\mathbf{Z}, \left[r_{d-1}^L, n_d, 1\right]\right)$
```

Algorithm 8 Stable pseudoinverse

```
 1: function PINV($\Omega$)
 2:     $[\mathbf{U}, \mathbf{S}, \mathbf{V}] \leftarrow \text{SVD}(\Omega)$
 3:     $r \leftarrow \text{LENGTH}(\text{DIAG}(\mathbf{S}))$
 4:     norm2 = MAX (DIAG ($\mathbf{S}$))
 5:     $s^\dagger \leftarrow \text{ZEROS}([r, 1])$
 6:     for $i = 1$ to r do
 7:         if $\mathbf{S}_{ii} > \varepsilon_{\text{mach}} \cdot$ norm2 then      ▷ where $\varepsilon_{\text{mach}}$ is the machine precision
 8:             $s_i^\dagger = 1/\mathbf{S}_{ii}$
 9:         end if
10:     end for
11:     return $\left(\mathbf{V}\, \text{DIAG}\left(s^\dagger\right) \mathbf{U}^\top\right)$
12: end function
```

Algorithm 9 STTA: assemble (right)

Input: Sketches Ψ_μ of size $r_{\mu-1}^L \times n_\mu \times r_\mu^R$ for $1 \le \mu \le d$, sketches Ω_μ of size $r_\mu^L \times r_\mu^R$ for $1 \le \mu \le d-1$
Output: Tensor train $\widetilde{\mathcal{T}} = C_1 \cdots C_d$ approximating $\mathcal{T}$, with TT-ranks $r_1^R, r_2^R, \ldots, r_{d-1}^R$.

```
1: $C_1 \leftarrow \Psi_1$
2: for $\mu = 2$ to $d$ do
3:     $C_\mu \leftarrow \text{RESHAPE}\left(\text{PINV}(\Omega_{\mu-1})\, \Psi^{\le 1}, \left[r_{\mu-1}^R, n_\mu, r_\mu^R\right]\right)$
4: end for
```

5.1 Using non-Gaussian sketching matrices

In this section, we present three sketching matrices that we want to use in the STTA algorithm.

The first operator we use is the Gaussian sketching matrix [5]. Given n and ℓ, we define Gaussian matrix $\Phi \in \mathbb{R}^{\ell \times n}$ whose entries are iid standard Gaussian random variables. The cost of applying such operator to a data matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ is $\mathcal{O}(mn\ell)$.

The second sketching operator we present is the subsampled randomized Hadamard transform (SRHT) [8]. Given n power of two and ℓ, we define a SRHT matrix as

$$\Phi = \mathbf{RHD}$$

where

- $\mathbf{R} \in \mathbb{R}^{\ell \times n}$ is formed by a subset of ℓ rows of the identity $\mathbf{I}_n$, chosen uniformly at random with or without replacement,
- $\mathbf{H} = \frac{1}{\sqrt{n}}\mathbf{H}_n \in \mathbb{R}^{n \times n}$, where $\mathbf{H}_n$ is the Hadamard matrix of order n, recursively defined as

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \text{ and } \mathbf{H}_{2n} = \begin{bmatrix} \mathbf{H}_n & \mathbf{H}_n \\ \mathbf{H}_n & -\mathbf{H}_n \end{bmatrix},$$

- $\mathbf{D} \in \mathbb{R}^{n \times n}$ in a diagonal matrix with independent Rademacher entries.

The cost of applying a such operator to a data matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ is $\mathcal{O}(mn \log n + m\ell)$.

The third sketching operator we use is the hashed randomized Hadamard transform (HRHT) [3]. Given n power of two, ℓ and s, we define a SRHT matrix as

$$\Phi = \mathbf{S}_h\mathbf{HD}$$

where

- $\mathbf{S}_h \in \mathbb{R}^{\ell \times n}$ is a random s-hashing variant matrix, defined as $\mathbf{S}_h = \frac{1}{\sqrt{s}}\left(S^{(1)} + \cdots + S^{(s)}\right)$, where the s independent 1-hashing matrices $S^{(i)} \in \mathbb{R}^{\ell \times n}$ are zero matrices with one Rademacher entry per column in a random position,
- $\mathbf{H}$ is as above,
- $\mathbf{D}$ is as above.

The cost of applying a such operator to a data matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ is $\mathcal{O}(mn \log n + m\ell s)$. We call s the order of the sketching matrix.

For both SRHT and HRHT, if n is not a power of two, we pad the input data with zeros to make its dimension a power of two. Note that $n \leq 2^{\lceil \log_2 n \rceil} < 2n$, so zero padding doesn't change the asymptotic computational cost.

While the Gaussian matrices are somewhat the golden standard of randomized linear algebra, structured sketching matrices (see [5, Section 4.6] for a definition) bring many advantages and very little disadvantages. First, structured matrices significantly reduce the computational complexity thanks to the fast Hadamard transform algorithm. This, essentially, allows to exchange ℓ with $\log n$ in the asymptotic complexity. Second, structured sketching operators are much more memory efficient. Unlike Gaussian matrices, which require the storage of each individual entry, structured matrices can be described algorithmically generating on-the-fly the random components. Finally, the only small disadvantage of structured sketching matrices is that they usually need slightly higher ℓ to become an oblivious subspace embedding with the same parameters.

In the following, we will call respectively STTA-Gaussian, STTA-SRHT and STTA-HRHT the algorithms obtained by using these different sketching matrices.

5.2 Numerical experiments with STTA

In this section, we make a numerical comparison of the TT-factorization algorithms presented so far. As in Section 4.1, we test the performance on $\mathcal{T}_{\mathsf{Hilbert}}$ and $\mathcal{T}_{\mathsf{sqrt}}$. We restrict our analysis to the approximation error for different TT-ranks.

For $\mathcal{T}_{\mathsf{Hilbert}}$ we take $d = 7$ and $n = 5$, instead, with $\mathcal{T}_{\mathsf{sqrt}}$ we take $d = 5$ and $n = 10$. Given a target TT-rank r, we set the left TT-ranks to $r^L = \text{FIX\_RANKS}\,(r, \ldots, r;\, n, \ldots, n)$ and the right TT-ranks to $r^R = 2r^L$, so that the obtained TT will have TT-ranks r^L.

In Figure 7, we compare the algorithms TT-SVD, TT-rSVD, STTA-Gaussian, STTA-SRHT, and STTA-HRHT with order $s = 10$. Interestingly, HRHT obtains very similar results to Gaussian. However, when considering SRHT, the error is substantially worse than with the other sketching matrices. Finally, when comparing STTA with Gaussian sketching matrices with TT-SVD and TT-rSVD, we see how STTA obtains errors that are of the same magnitude of the ones from the other algorithms.

Now the question is: why does HRHT perform better than SRHT? As we mentioned in Section 5, in STTA, sometimes there can be sketching in higher dimension. However, if we sketch a matrix in higher dimension with SRHT, we have a few problems. First, necessarily, we need to sample the matrix $\mathbf{R}$ with replacement. Second, we sample at least one row twice because of the pigeonhole principle. Third, as a result, the sketched matrix will have at least two identical rows, property that we would prefer not to have in a matrix. We observe that HRHT does not have these problems. However, we point out that, even with Gaussian or HRHT, if we sketch on higher dimension, it is impossible to have linearly independent rows. Thus, the problem with SRHT cannot be that the resulting matrix is row-singular.

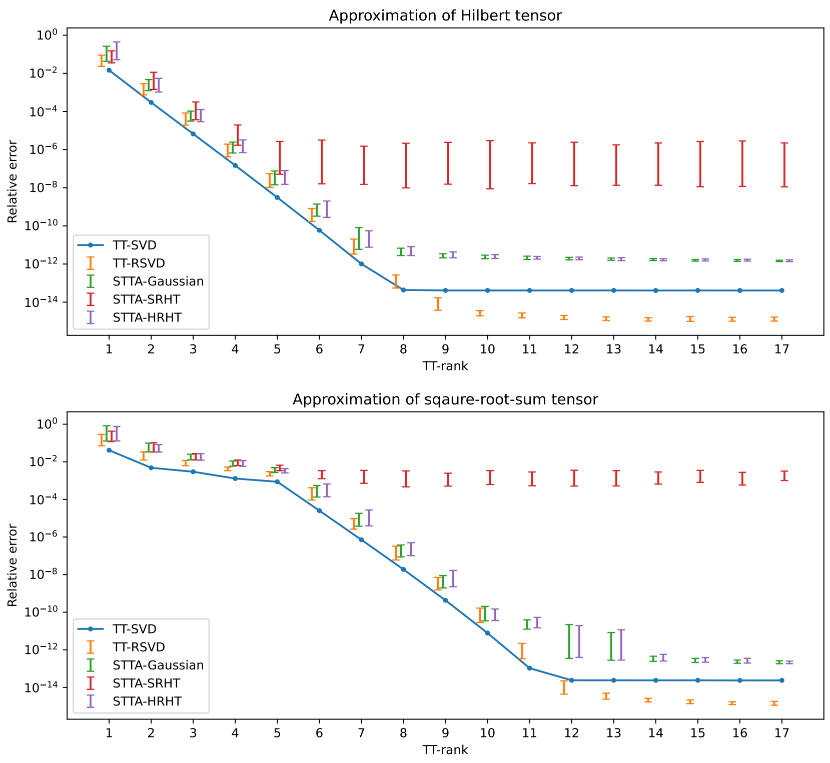


Figure 7: Error performance of TT-SVD, TT-rSVD, STTA-Gaussian, STTA-SRHT, and STTA-HRHT on $\mathcal{T}_{\text{Hilbert}}$ and $\mathcal{T}_{\text{sqrt}}$ (1-st and 2-nd plot, respectively), when varying the target TT-rank r.

6 Error analysis for STTA-Gaussian and STTA-HRHT

In this final section, we outline the error analysis for STTA-Gaussian presented in [6]. Then, we present our partial results in updating the proof for the HRHT case.

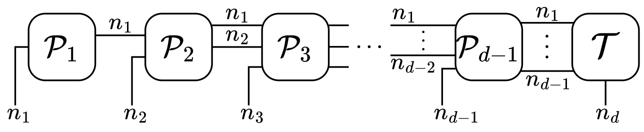
The idea behind the proof is to express the approximation 7 in terms of oblique projectors. We define the oblique projectors

$$\mathcal{P}_\mu = \mathcal{T}^{\le \mu}\mathbf{X}_\mu(\mathbf{Y}_\mu^\top \mathcal{T}^{\le \mu}\mathbf{X}_\mu)^\dagger \mathbf{Y}_\mu^\top.$$

We write the approximation $\widetilde{\mathcal{T}}$ as

$$\widetilde{\mathcal{T}}^{\le d-1} = (\mathcal{P}_1 \otimes \mathbf{I}) \cdots (\mathcal{P}_{d-2} \otimes \mathbf{I})(\mathcal{P}_{d-1} \otimes \mathbf{I})\mathcal{T}^{\le d-1},$$

where the sizes of the identity matrices $\mathbf{I}$ is such that all the products are well defined. In term of tensor diagram this becomes the following.



(10)

We present some preliminary results.

Theorem 1. *[9, Theorem 4.3] Consider two random Gaussian matrices $\mathbf{X}$ and $\mathbf{Y}$ of size $n \times r$ and $n \times (r+k)$, respectively, with $k > 0$. Then the expected approximation error of*

$$\widehat{\mathbf{A}} = \mathbf{A}\mathbf{X}(\mathbf{Y}^\top \mathbf{A}\mathbf{X})^\dagger \mathbf{Y}^\top \mathbf{A}$$

is bounded by

$$\mathbb{E}\|\widehat{\mathbf{A}} - \mathbf{A}\|_F^2 \le \left(1 + \frac{r}{k-1}\right)\left(1 + \frac{\widehat{r}}{r - \widehat{r} - 1}\right)\|\mathbf{A}_{\widehat{r}} - \mathbf{A}\|_F^2,$$

where $\mathbf{A}_{\widehat{r}}$ is any best rank $\widehat{r}$ approximation of $\mathbf{A}$ with $\widehat{r} < r - 1$.

Theorem 2. *[5, Proposition 10.1] Fixed two matrices $\mathbf{S}$ and $\mathbf{T}$, let $\mathbf{G}$ be a Gaussian matrix. Then*

$$\mathbb{E}\left[\|\mathbf{S}\mathbf{G}\mathbf{T}\|_F^2\right] = \|\mathbf{S}\|_F^2 \|\mathbf{T}\|_F^2$$

Theorem 3. *[5, Proposition 10.2] Let $\mathbf{G} \in \mathbb{R}^{n \times m}$ be a Gaussian matrix. Then*

$$\mathbb{E}\left[\|\mathbf{G}^\dagger\|_F^2\right] = \frac{n}{m-n-1}$$

Theorem 4. *[6, Proposition 3.1] The approximation $\widetilde{\mathcal{T}}$ returned by STTA is such that*

$$\|\widetilde{\mathcal{T}} - \mathcal{T}\|_F \le \sum_{\mu=1}^{d-1} \left\| \prod_{\alpha<\mu} (\mathcal{P}_\alpha \otimes \mathbf{I})(\mathbf{I} - \mathcal{P}_\mu)\mathcal{T}^{\le \mu} \right\|_F .$$

We are now ready to sketch the proof of the following theorem about the error bound for STTA-Gaussian, the complete proof can be found in [6].

Theorem 5. *[6, Proposition 3.2] Take the sketching matrices $\mathbf{X}_\mu$, $\mathbf{Y}_\mu$ to be independent standard Gaussian matrices with $r_\mu^R < r_\mu^L - 1$. Assume further that $\text{RANK}\left(T^{\le\mu}\right) \ge r_\mu^R$ for $1 \le \mu \le d-1$. Then for any $(\widehat{r}_1, \ldots, \widehat{r}_{d-1})$ such that $\widehat{r}_\mu < r_\mu^R - 1$ for $1 \le \mu \le d-1$, the STTA approximation $\widetilde{\mathcal{T}}$ of $\mathcal{T}$ obtained with Algorithm 9 satisfies*

$$\mathbb{E}\|\widetilde{\mathcal{T}} - \mathcal{T}\|_F \le \sum_{\mu=1}^{d-1} \left[\prod_{\alpha=1}^{\mu-1} c_\alpha\right] c'_\mu \sqrt{\sum_{k > \widehat{r}_\mu} \sigma_k(\mathcal{T}^{\le\mu})^2} \le \left(\sum_{\mu=1}^{d-1} \left[\prod_{\alpha=1}^{\mu-1} c_\alpha\right] c'_\mu\right) \|\mathcal{T}_{\widehat{r}} - \mathcal{T}\|_F^2,$$

where $\mathcal{T}_{\widehat{r}}$ is any best TT approximation of $\mathcal{T}$ with TT-ranks $(\widehat{r}_1, \ldots, \widehat{r}_{d-1})$, and c_μ and c'_μ are given by

$$c_\mu = 1 + \sqrt{\frac{r_\mu^R}{r_\mu^L - r_\mu^R - 1}} \qquad c'_\mu = \sqrt{1 + \frac{r_\mu^R}{r_\mu^L - r_\mu^R - 1}} \cdot \sqrt{1 + \frac{\widehat{r}_\mu}{r_\mu^R - \widehat{r}_\mu - 1}}$$

Proof. (sketch). We write the QR decomposition of $\mathcal{T}^{\le\mu}\mathbf{X}_\mu = \mathbf{Q}\mathbf{R}$. Since $\mathcal{T}^{\le\mu}\mathbf{X}_\mu$ have full column rank $\underline{\text{almost surely}}$, the factor $\mathbf{R} \in \mathbb{R}^{r_\mu^R \times r_\mu^R}$ is invertible. This allows us to rewrite the definition of $\mathcal{P}_\mu$ as

$$\mathcal{P}_\mu = \mathcal{T}^{\le\mu}\mathbf{X}_\mu(\mathbf{Y}_\mu^\top \mathcal{T}^{\le\mu}\mathbf{X}_\mu)^\dagger \mathbf{Y}_\mu^\top = \mathbf{Q}\mathbf{R}(\mathbf{Y}_\mu^\top \mathbf{Q}\mathbf{R})^\dagger \mathbf{Y}_\mu^\top = \mathbf{Q}(\mathbf{Y}_\mu^\top \mathbf{Q})^\dagger \mathbf{Y}_\mu^\top.$$

Completing the r_μ^R columns of $\mathbf{Q}$ to a square orthogonal matrix $[\mathbf{Q} \;\; \mathbf{Q}_\perp]$ and fixing $\mathbf{A}$ of size $(n_1 \cdots n_\mu) \times (n_{\mu+1} \cdots n_d)$, we prove that

$$\|\mathcal{P}_\mu \mathbf{A}\|_F \le \|(\mathbf{Y}_\mu^\top \mathbf{Q})^\dagger \mathbf{Y}_\mu^\top \mathbf{Q}\mathbf{Q}^\top \mathbf{A}\|_F + \|(\mathbf{Y}_\mu^\top \mathbf{Q})^\dagger \mathbf{Y}_\mu^\top \mathbf{Q}_\perp \mathbf{Q}_\perp^\top \mathbf{A}\|_F \le \|\mathbf{A}\|_F + \|(\mathbf{Y}_\mu^\top \mathbf{Q})^\dagger \mathbf{Y}_\mu^\top \mathbf{Q}_\perp \mathbf{Q}_\perp^\top \mathbf{A}\|_F.$$

By the orthogonal invariance of the Gaussian matrices $\mathbf{Z}_1 := \mathbf{Y}_\mu^\top \mathbf{Q}$ and $\mathbf{Z}_2 := \mathbf{Y}_\mu^\top \mathbf{Q}_\perp$ are also independent Gaussian random matrices. Applying Theorems 2 and 3, we obtain

$$\begin{aligned}\mathbb{E}_{\mathbf{Z}_1,\mathbf{Z}_2} \|\mathcal{P}_\mu \mathbf{A}\|_F \le \mathbb{E}_{\mathbf{Z}_1,\mathbf{Z}_2} \|\mathbf{Z}_1^\dagger \mathbf{Z}_2 \mathbf{Q}_\perp^\top \mathbf{A}\|_F &\le \sqrt{\mathbb{E}_{\mathbf{Z}_1,\mathbf{Z}_2} \|\mathbf{Z}_1^\dagger \mathbf{Z}_2 \mathbf{Q}_\perp^\top \mathbf{A}\|_F^2} \\ &= \sqrt{\mathbb{E}_{\mathbf{Z}_1} \|\mathbf{Z}_1^\dagger\|_F^2} \|\mathbf{Q}_\perp^\top \mathbf{A}\|_F = \sqrt{\frac{r_\mu^R}{r_\mu^L - r_\mu^R - 1}} \|\mathbf{Q}_\perp^\top \mathbf{A}\|_F \le c_\mu \|\mathbf{A}\|_F.\end{aligned}$$

Using the law of total expectation, we can prove that

$$\mathbb{E}_{\mathbf{X}_1 \mathbf{Y}_1, \ldots, \mathbf{X}_\mu, \mathbf{Y}_\mu} \left\| \prod_{\alpha=1}^{\mu-1} (\mathcal{P}_\alpha \otimes \mathbf{I})(\mathbf{I} - \mathcal{P}_\mu) \mathcal{T}^{\le\mu} \right\|_F \le c_1 \cdots c_{\mu-1} \mathbb{E}_{\mathbf{X}_\mu, \mathbf{Y}_\mu} \|(\mathbf{I} - \mathcal{P}_\mu) \mathcal{T}^{\le\mu}\|_F.$$

In addition, by Theorem 1, we have

$$\mathbb{E}_{\mathbf{X}_\mu, \mathbf{Y}_\mu} \|(\mathbf{I} - \mathcal{P}_\mu) \mathcal{T}^{\le\mu}\|_F^2 \le (c_\mu')^2 \sum_{k > \widehat{r}_\mu} \sigma_k(\mathcal{T}^{\le\mu})^2$$

for any $\widehat{r}_\mu < r_\mu^R - 1$, where $\sigma_k(\cdot)$ denotes the k-th largest singular value of a matrix. Finally, we apply Theorem 4

$$\begin{aligned}\mathbb{E}\|\widetilde{\mathcal{T}} - \mathcal{T}\|_F &\le \mathbb{E}\left[\sum_{\mu=1}^{d-1} \left\| \prod_{\alpha<\mu} (\mathcal{P}_\alpha \otimes \mathbf{I})(\mathbf{I} - \mathcal{P}_\mu) \mathcal{T}^{\le\mu} \right\|_F \right] = \sum_{\mu=1}^{d-1} \mathbb{E} \left\| \prod_{\alpha<\mu} (\mathcal{P}_\alpha \otimes \mathbf{I})(\mathbf{I} - \mathcal{P}_\mu) \mathcal{T}^{\le\mu} \right\|_F \\ &\le \sum_{\mu=1}^{d-1} c_1 \cdots c_{\mu-1} \mathbb{E}_{\mathbf{X}_\mu, \mathbf{Y}_\mu} \|(\mathbf{I} - \mathcal{P}_\mu) \mathcal{T}^{\le\mu}\|_F \le \sum_{\mu=1}^{d-1} \left[\prod_{\alpha=1}^{\mu-1} c_\alpha\right] \sqrt{\mathbb{E}_{\mathbf{X}_\mu, \mathbf{Y}_\mu} \|(\mathbf{I} - \mathcal{P}_\mu) \mathcal{T}^{\le\mu}\|_F^2} \\ &\le \sum_{\mu=1}^{d-1} \left[\prod_{\alpha=1}^{\mu-1} c_\alpha\right] c_\mu' \sqrt{\sum_{k > \widehat{r}_\mu} \sigma_k(\mathcal{T}^{\le\mu})^2}.\end{aligned}$$

The second inequality follows from the bound

$$\sum_{k > \widehat{r}_\mu} \sigma_k(\mathcal{T}^{\le\mu})^2 \le \|\mathcal{T} - \mathcal{T}_{\widehat{r}}\|_F^2.$$

□

We now try to update the proof for the HRHT case. In order to do so, we need to adjust several steps of the proof of Theorem 5.

First, we need a result similar Theorem 1. However, proving Theorem 1 for the HRHT case would require to characterize the distribution of $\Phi\mathbf{Q}$ for $\mathbf{Q}$ column orthogonal. The study of s-hashing sketching matrices is recent, so there is still a lack of theoretical results.

Second, we managed to weaken the hypotheses of Theorem 2 and to prove that it also applies to the HRHT case. The respective theorems are stated and proved below.

Theorem 6. *Fix two matrices* $\mathbf{S}$, $\mathbf{T}$ *and draw a random matrix* $\mathbf{G}$. *If* $\mathbb{E}[\mathbf{G}] = 0$ *and* $\mathbb{V}\mathrm{ar}(\operatorname{vec} \mathbf{G}) = c\mathbf{I}$ *for some constant* c, *then*

$$\mathbb{E}\|\mathbf{S}\mathbf{G}\mathbf{T}\|_F^2 = c\|\mathbf{S}\|_F^2 \|\mathbf{T}\|_F^2$$

Proof.

$$\|\mathbf{SGT}\|_F^2 = \sum_{i,j} (\mathbf{SGT})_{ij}^2 = \sum_{i,j} \left(\sum_{k,h} \mathbf{S}_{ik} \mathbf{G}_{kh} \mathbf{T}_{hj} \right)^2$$
$$= \sum_{i,j} \left(\sum_{k,h} \mathbf{S}_{ik}^2 \mathbf{G}_{kh}^2 \mathbf{T}_{hj}^2 + \sum_{(k,h)\neq(r,q)} \mathbf{S}_{ik} \mathbf{G}_{kh} \mathbf{T}_{hj} \mathbf{S}_{ir} \mathbf{G}_{rq} \mathbf{T}_{qj} \right)$$

follows that

$$\begin{aligned}
\mathbb{E}\|\mathbf{SGT}\|_F^2 &= \sum_{i,j} \left(\sum_{k,h} \mathbf{S}_{ik}^2 \mathbb{E}[\mathbf{G}_{kh}^2] \mathbf{T}_{hj}^2 + \sum_{(k,h)\neq(r,q)} \mathbf{S}_{ik} \mathbf{T}_{hj} \mathbf{S}_{ir} \mathbf{T}_{qj} \mathbb{E}[\mathbf{G}_{kh} \mathbf{G}_{rq}] \right) \\
&= \sum_{i,j} \left(\sum_{k,h} \mathbf{S}_{ik}^2 \underbrace{\mathbb{V}\mathrm{ar}(\mathbf{G}_{kh})}_{=c} \mathbf{T}_{hj}^2 + \sum_{(k,h)\neq(r,q)} \mathbf{S}_{ik} \mathbf{T}_{hj} \mathbf{S}_{ir} \mathbf{T}_{qj} \underbrace{\mathbb{C}\mathrm{ov}(\mathbf{G}_{kh},\, \mathbf{G}_{rq})}_{=0} \right) \\
&= c \sum_{i,j} \sum_{k,h} \mathbf{S}_{ik}^2 \mathbf{T}_{hj}^2 = c \left(\sum_{i,k} \mathbf{S}_{ik}^2 \right) \left(\sum_{h,j} \mathbf{T}_{hj}^2 \right) = c \|\mathbf{S}\|_F^2 \|\mathbf{T}\|_F^2
\end{aligned}$$

□

Corollary: If we pick $\mathbf{G}$ Gaussian matrix we obtain Theorem 2.

Theorem 7. *Fix* $\mathbf{Q} \in \mathbb{R}^{n\times m}$*, s.t.* $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$*. Then,* $\mathbb{R}^{\ell \times p} \ni \mathbf{Z} = \mathbf{\Phi Q} = \mathbf{S}_h \mathbf{HDQ}$ *is such that* $\mathbb{E}[\mathbf{Z}] = 0$ *and* $\mathbb{V}\mathrm{ar}(\mathrm{vec}\,\mathbf{Z}) = \frac{n}{\ell}\mathbf{I}$

Proof. To make the notation lighter we denote $\mathbf{S}_h$ as $\mathbf{S}$ and $\mathbf{D}_{ii}$ as δ_i for all i.

1. Proof that $\mathbb{E}[\mathbf{Z}] = 0$:

$$\mathbb{E}[\mathbf{Z}] = \mathbb{E}[\mathbf{SHDQ}] \overset{\perp\!\!\!\perp}{=} \mathbb{E}[\mathbf{S}]\mathbf{H}\mathbb{E}[\mathbf{D}]\mathbf{Q} = 0 \quad \text{since } \mathbb{E}[\mathbf{S}]_{ij} = 0 \;\; \forall i,j$$

2. Proof that $\mathbb{E}[\mathbf{Z}_{ij}^2] = \frac{n}{\ell}\ \forall i,j$

$$\begin{aligned}
\mathbb{E}[\mathbf{Z}_{ij}^2] &= \mathbb{E}\left[\left(\sum_{k,h} \mathbf{S}_{ik} \mathbf{H}_{kh} \delta_h \mathbf{Q}_{hj} \right)^2 \right] = \mathbb{E}\left[\sum_{k,h} \mathbf{S}_{ik}^2 \mathbf{H}_{kh}^2 \delta_h^2 \mathbf{Q}_{hj}^2 \right] \\
&\quad + \mathbb{E}\left[\sum_{\substack{k\neq r\\ h}} \mathbf{S}_{ik} \mathbf{H}_{kh} \delta_h \mathbf{Q}_{hj} \mathbf{S}_{ir} \mathbf{H}_{rh} \delta_h \mathbf{Q}_{hj} \right] + \mathbb{E}\left[\sum_{\substack{k,r\\ h\neq q}} \mathbf{S}_{ik} \mathbf{H}_{kh} \delta_h \mathbf{Q}_{hj} \mathbf{S}_{ir} \mathbf{H}_{rq} \delta_q \mathbf{Q}_{qj} \right] \\
&= \sum_{k,h} \underbrace{\mathbb{E}[\mathbf{S}_{ik}^2]}_{=1/\ell} \mathbf{H}_{kh}^2 \mathbf{Q}_{hj}^2 + \sum_{\substack{k\neq r\\ h}} \mathbb{E}[\mathbf{S}_{ik}\mathbf{S}_{ir}] \mathbf{H}_{kh} \mathbf{H}_{rh} \mathbf{Q}_{hj}^2 + \sum_{\substack{k,r\\ h\neq q}} \mathbb{E}[\mathbf{S}_{ik}\mathbf{S}_{ir}] \mathbf{H}_{kh} \mathbf{H}_{rq} \mathbb{E}[\delta_h \delta_q] \mathbf{Q}_{hj} \mathbf{Q}_{qj} \\
&= \frac{n}{\ell} + 0 + 0 = \frac{n}{\ell}
\end{aligned}$$

since $\mathbb{E}[\mathbf{S}_{ik}\mathbf{S}_{ir}] \overset{\perp\!\!\!\perp}{=} \mathbb{E}[\mathbf{S}_{ik}]\mathbb{E}[\mathbf{S}_{ir}] = 0$ for $k \neq r$, and $\mathbb{E}[\delta_h \delta_q] \overset{\perp\!\!\!\perp}{=} \mathbb{E}[\delta_h]\mathbb{E}[\delta_q] = 0$ for $h \neq q$.

3. Proof that $\mathbb{E}[\mathbf{Z}_{ij}\mathbf{Z}_{xy}] = 0\ \forall (i,j) \neq (x,y)$

- if $i = x$ and $j \neq y$

$$\mathbb{E}[\mathbf{Z}_{ij}\mathbf{Z}_{xy}] = \mathbb{E}\left[\left(\sum_{k,h}\mathbf{S}_{ik}\mathbf{H}_{kh}\delta_h\mathbf{Q}_{hj}\right)\left(\sum_{r,q}\mathbf{S}_{ir}\mathbf{H}_{rq}\delta_q\mathbf{Q}_{qy}\right)\right] = \mathbb{E}\left[\sum_{k,h}\mathbf{S}_{ik}^2\mathbf{H}_{kh}^2\delta_h^2\mathbf{Q}_{hj}\mathbf{Q}_{hy}\right]$$
$$+\,\mathbb{E}\left[\sum_{\substack{k\neq r\\ h}}\mathbf{S}_{ik}\mathbf{H}_{kh}\delta_h\mathbf{Q}_{hj}\mathbf{S}_{ir}\mathbf{H}_{rh}\delta_h\mathbf{Q}_{hy}\right] + \mathbb{E}\left[\sum_{\substack{k,r\\ h\neq q}}\mathbf{S}_{ik}\mathbf{H}_{kh}\delta_h\mathbf{Q}_{hj}\mathbf{S}_{ir}\mathbf{H}_{rq}\delta_q\mathbf{Q}_{qy}\right]$$
$$= \mathbb{E}\left[\sum_k \mathbf{S}_{ik}^2\right]\sum_h \mathbf{Q}_{hj}\mathbf{Q}_{hy} + \sum_{\substack{k\neq r\\ h}}\mathbb{E}[\mathbf{S}_{ik}\mathbf{S}_{ir}]\mathbf{H}_{kh}\mathbf{H}_{rh}\mathbf{Q}_{hj}\mathbf{Q}_{hy}$$
$$+\sum_{\substack{k,r\\ h\neq q}}\mathbb{E}[\mathbf{S}_{ik}\mathbf{S}_{ir}]\mathbf{H}_{kh}\mathbf{H}_{rq}\mathbb{E}[\delta_h\delta_q]\mathbf{Q}_{hj}\mathbf{Q}_{qy} = 0+0+0 = 0$$

since $\sum_h \mathbf{Q}_{hj}\mathbf{Q}_{hy} = \mathbf{Q}_{\cdot j}^\top\mathbf{Q}_{\cdot y} = 0$ for $j \neq y$, $\mathbb{E}[\mathbf{S}_{ik}\mathbf{S}_{ir}] \overset{\perp\!\!\!\perp}{=} \mathbb{E}[\mathbf{S}_{ik}]\mathbb{E}[\mathbf{S}_{ir}] = 0$ for $k \neq r$, and $\mathbb{E}[\delta_h\delta_q] \overset{\perp\!\!\!\perp}{=} \mathbb{E}[\delta_h]\mathbb{E}[\delta_q] = 0$ for $h \neq q$.

- if $i \neq x$

$$\mathbb{E}[\mathbf{Z}_{ij}\mathbf{Z}_{xy}] = \mathbb{E}\left[\left(\sum_{k,h}\mathbf{S}_{ik}\mathbf{H}_{kh}\delta_h\mathbf{Q}_{hj}\right)\left(\sum_{r,q}\mathbf{S}_{xr}\mathbf{H}_{rq}\delta_q\mathbf{Q}_{qy}\right)\right]$$
$$= \mathbb{E}\left[\sum_{k,h,r,q}\mathbf{S}_{ik}\mathbf{H}_{kh}\delta_h\mathbf{Q}_{hj}\mathbf{S}_{xr}\mathbf{H}_{rq}\delta_q\mathbf{Q}_{qy}\right] = \sum_{k,h,r,q}\mathbb{E}[\mathbf{S}_{ik}\mathbf{S}_{xr}]\mathbf{H}_{kh}\mathbf{H}_{rq}\mathbb{E}[\delta_h\delta_q]\mathbf{Q}_{hj}\mathbf{Q}_{qy} = 0$$

since $\mathbb{E}[\mathbf{S}_{ik}\mathbf{S}_{xr}] = \mathbb{E}[\mathbf{S}_{ik}\mathbb{E}[\mathbf{S}_{xr}|\mathbf{S}_{ik}]] = \mathbb{E}[\mathbf{S}_{ik}\cdot 0] = 0$ for $i \neq x$.

□

Third, we replace "almost surely" at the beginning of the proof with "with high probability".

Lastly, since HRHT are not orthogonally invariant, we need to find a formula directly for $\mathbb{E}\|\mathbf{Z}_1^\dagger\|_F^2 = \mathbb{E}\|(\Phi\mathbf{Q})^\dagger\|_F^2 = \mathbb{E}\|(\mathbf{S}_h\mathbf{HDQ})^\dagger\|_F^2$. However, since the proof of Theorem 3 heavily relies on the properties of the Wishart distribution, we didn't manage to find a rigorous result, especially with the additional term $\mathbf{Q}$. However, we managed to numerically estimate a candidate formula.

Theorem 8 (Candidate). *Let $m, n, p \in \mathbb{N}$, with $m < n-1$. Fix $\mathbf{Q} \in \mathbb{R}^{p\times m}$, s.t. $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}_m$. Let $\mathbb{R}^{n\times p} \ni \mathbf{\Phi} = \mathbf{S}_h\mathbf{HD}$ a HRHT sketching matrix with order s. Then*

$$\mathbb{E}\|(\mathbf{S}_h\mathbf{HDQ})^\dagger\|_F^2 \approx \frac{mn^2}{(n-m-1)s^2 2^{\lceil \log_2 p\rceil}}$$

To test the goodness of this candidate formula, we set $n = 128$, $p = 256$, $m = 64$, and $s = 10$ as the default case, then we vary each parameter while keeping the other three fixed. In Figure 8, we compare our candidate formula with a numerical estimation of $\mathbb{E}\|(\mathbf{S}_h\mathbf{HDQ})^\dagger\|_F^2$. To compute this, we generate a sample of 30 matrices $\mathbf{S}_h\mathbf{HDQ}$, where $\mathbf{Q}$ is obtained as the Q factor of the QR factorization of a Gaussian matrix. Then, we compute the pseudoinverse using Algorithm 8 and the Frobenius norm. Finally, we compute the average. As we can see, our candidate formula follows almost exactly the behaviour of the numerical estimations.

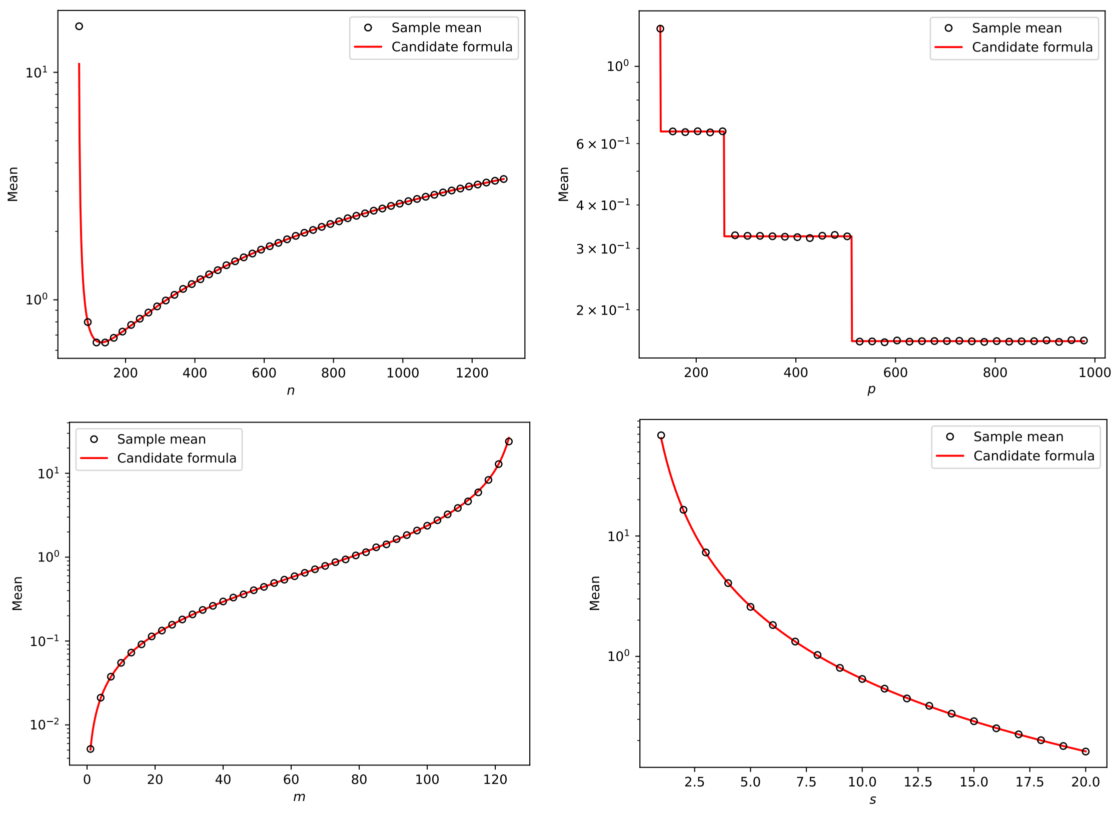


Figure 8: Numerical tests for Theorem 8. We set $n = 128$, $p = 256$, $m = 64$, and $s = 10$ and we vary each parameter while keeping the other three fixed.

7 Conclusion and future developments

In this project, we explored the use of randomization in TT-factorization and TT-rounding algorithms. We analyzed the performance of both deterministic and randomized algorithms. Our comparative analysis between deterministic and randomized approaches revealed that, although both exhibited errors of similar magnitudes, the randomized algorithms achieved significant speed-ups, although their asymptotic behavior remained the same.

The introduction of structured sketching operators, particularly HRHT, into the STTA algorithm has emerged as a promising development. Our experiments showed that HRHT has very similar performance to Gaussian sketching operators in terms of error when varying the target TT-rank. This highlights a potential of structured sketching operators to effectively improve the performance of randomized algorithms on TTs.

Looking ahead, there are promising directions for future research. Expanding the theoretical framework surrounding hashed structured sketching operators is critical, in particular to conclude our proof about the error bound for STTA-HRHT. In addition, the use of structured sketching operators could also be extended to TT-rounding.

References

[1] H. Al Daas, G. Ballard, P. Cazeaux, E. Hallman, A. Międlar, M. Pasha, T. W. Reid, and A. K. Saibaba, *Randomized algorithms for rounding in the tensor-train format*, SIAM Journal on Scientific Computing, 45 (2023), pp. A74–A95.

[2] S. Badreddine, *Leveraging Symmetries and Low-Rank Structure of Matrices and Tensors in High-Dimensional Quantum Chemistry Problems*, PhD dissertation, Sorbonne Université, INRIA, Laboratoire Jacques-Louis Lions, Paris, France, 2024.

[3] C. Cartis, J. Fiala, and Z. Shao, *Hashing embeddings of optimal dimension, with applications to linear least squares*, 2021.

[4] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, vol. 56 of Springer Series in Computational Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, second ed., 2019.

[5] N. Halko, P.-G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, 2010.

[6] D. Kressner, B. Vandereycken, and R. Voorhaar, *Streaming tensor train approximation*, 2022.

[7] I. V. Oseledets, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317.

[8] J. A. Tropp, *Improved analysis of the subsampled randomized hadamard transform*, 2011.

[9] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Practical sketching algorithms for low-rank matrix approximation*, SIAM Journal on Matrix Analysis and Applications, 38 (2017), p. 1454–1485.