EPFL

MASTER THESIS

Randomized low-rank approximation of matrices and tensors

Author:
Davide PRADOVERA

Supervisor:
Prof. Daniel KRESSNER

Examiner:
Dr. Michael PETERS

A thesis submitted in fulfillment of the requirements
for the Master degree in Computational Science and Engineering

within the

Numerical Analysis and High-Performance Computing Group
Section of Mathematics
School of Basic Sciences

Submitted on June 23, 2017 (v. July 11, 2017)

EPFL

Abstract

School of Basic Sciences
Section of Mathematics
Numerical Analysis and High-Performance Computing Group

Master degree

Randomized low-rank approximation of matrices and tensors

by Davide PRADOVERA

Low-rank matrix and tensor approximation is a central topic in computational linear algebra, with countless applications in scientific computing and data analysis. In this thesis, we explore the possibility to compute low-rank approximations of matrices and tensors by using randomized techniques. In particular, we provide an overview of the existing literature on the topic, and we derive several extensions to tackle two variations of the compression problem: the computation of low-rank approximations with fixed rank and with fixed precision.

This thesis shows that randomized approaches provide a very appealing alternative to deterministic techniques. In particular, in the case of matrix compression, we describe methods which have (approximately) the same complexity as the most efficient deterministic algorithms, but are more appealing because of their robustness.

Regarding tensors, this thesis focuses on the Tensor Train (TT) format, which represents a generalization of the low-rank decomposition for matrices. In particular, we derive a randomized approach which performs better than the *state-of-the-art* deterministic algorithm for the compression of tensors in the TT format, by exploiting the efficiency with which some operations can be carried out in this format.

Moreover, this thesis extends such procedures to the compression of Hadamard products of matrices and of tensors, which represent fundamental operations in several linear algebra algorithms. In both cases, we devise randomized techniques which outperform their deterministic counterparts. Also, we show that randomness leads to a diminished memory exploitation.

To corroborate all these claims, we provide extensive numerical experiments.

Keywords: Low-rank approximation, principal component analysis, tensor decomposition, higher-order singular value decomposition (HOSVD), Tensor Train format, Hadamard product, randomized algorithms.

Acknowledgements

This thesis would not have been possible without the support and the contributions of many amazing people.

First of all, I would like to thank my thesis supervisor Prof. Daniel Kressner for allowing me to work on this incredibly interesting topic and for the many suggestions, which were always on target. I have learned a lot during the development of this project, and not only in computational linear algebra.

I am also very grateful towards Dr. Michael Peters for his helpful suggestions and support, and for agreeing to serve as my examiner.

Moreover, I would like to express my gratitude to all my fellow MA students at EPFL, who endured these last years with me. In particular, I want to thank Ondine and Stefano for always being there when I needed to take a break, as well as Alberto, Aleksa, Antonio, Carlos, Marco, Niccolò, Nicolò, Przemysław, Riccardo, Santo and Stefano for the many lunches together. I have really enjoyed the time spent with you, and hope we will stay in touch.

Farther away, but not less important, I am extremely grateful for the support I have received from my family: my mother, my grandmother, my aunt and my sister. Your encouragement has always been a great source of motivation for me.

Lastly, thank you, Chiara, for having been able to put up with me throughout these years, for helping me in finding and following my way, and for giving me a place to go back to when I am stranded.

Contents

Dedicated to my father

Chapter 1

Introduction

Matrices with low numerical rank are ubiquitous in a plethora of applications in mathematics, engineering and computer science, among which we cite just a few names: recommender systems, principal component analysis (PCA), compressive sampling, numerical solution of PDEs.

For such matrices, it is often appealing to derive a low-rank approximation of the form

$$\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{Z}^{\top} \tag{1.1}$$

with the two factors $\boldsymbol{W}$ and $\boldsymbol{Z}$ being "tall" matrices with just a few columns.

This representation is desirable for several reasons. Foremost, the compressed matrix can be stored very efficiently, and we can compute matrix-vector products quite cheaply. Moreover, the two factors provide an insight on the range and on the row space of the matrix: indeed, if the approximation (1.1) is sufficiently precise, the result obtained by applying $\boldsymbol{A}$ (resp. $\boldsymbol{A}^{\top}$) onto a general vector lies mostly within the range of $\boldsymbol{W}$ (resp. $\boldsymbol{Z}$).

Several decades of research have lead to many classical methods which can be applied to obtain representation (1.1): these include the singular value decomposition (SVD), the interpolative decomposition (ID) and the rank-revealing QR (RRQR) decomposition. Each method comes with its own advantages and disadvantages: in particular, it has proven complicated to design algorithms which are both efficient and robust.

More recently, on the wake of a rising interest in randomized approaches to numerical problems, Halko et al. [HMT11] have presented a stochastic algorithm to tackle the low-rank approximation problem, which achieves both efficiency and robustness, at the cost of a (slightly) increased approximation error. In this thesis we perform a brief review of this randomized procedure, and generalize it to deal with different versions of the compression problem.

In particular, we derive a randomized approach to compute low-rank approximations of *tensors*. Tensors are the natural generalizations of matrices to dimensions higher than 2. They provide a very compact representation of high-dimensional data, which is useful in many applications in a wide variety of fields, for instance in the numerical approximation of stochastic PDEs (see e.g. [BNT07]) and in molecular quantum dynamics (see e.g. [Lub08] and [Ven+07]).

For tensors, a low-rank representation generalizing (1.1) is even more appealing, due to the countless issues which arise when increasing the number of dimensions (the order) of the tensor. Among these, the most critical is probably the *curse of*

dimensionality, which states that most operations involving tensors have a computational cost increasing exponentially in the order. At a glance, this may not seem too troublesome, but many applications involve tensors of order 100 or 1000: in these cases it is impossible even to store such tensors on most computers!

For this reason, in this thesis we mostly consider tensors in the Tensor Train (TT) format, which can be manipulated (using the majority of the basic operations) without being affected by the *curse of dimensionality*. For tensors in this format, we devise algorithms which outperform the *state-of-the-art* deterministic approach.

Moreover, we devise randomized techniques to compress Hadamard products of tensors, which represent one of the most basic operations in tensor-based computations. In this case we derive an algorithm whose memory requirements and complexity are considerably lower than the ones of its deterministic counterpart, as shown by several numerical experiments.

Chapter 2

Preliminaries

2.1 Notation and basic operations with matrices

We start by summarizing the notation employed for matrices and vectors. Throughout this thesis, we represent vectors and matrices with bold letters, respectively lower- and upper-case, e.g. $\boldsymbol{v} \in \mathbb{R}^m$ and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ for some positive integers m and n.

In particular, we indicate with $\boldsymbol{I}$ the identity matrix, whose size we often clarify by using a subscript, e.g. $\boldsymbol{I}_m \in \mathbb{R}^{m \times m}$.

Elements of vectors and matrices are referred to in a MATLAB®-like format, e.g. $\boldsymbol{v}(i)$ and $\boldsymbol{A}(i,j)$ for suitable indices i and j. Moreover, we use a MATLAB® format to access sub-vectors and sub-matrices as well. For instance, if $\boldsymbol{A} \in \mathbb{R}^{20 \times 20}$, we have the following block representations:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}(1:5, 1:10) & \boldsymbol{A}(1:5, 11:20) \\ \boldsymbol{A}(6:20, 1:10) & \boldsymbol{A}(6:20, 11:20) \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}(1:5, :) \\ \boldsymbol{A}(6:20, :) \end{bmatrix}$$

In the later chapters we also make use of the `diag` operation from MATLAB®, which is defined as follows. Given a vector $\boldsymbol{v} \in \mathbb{R}^m$, we define the matrix $\texttt{diag}(\boldsymbol{v}) \in \mathbb{R}^{m \times m}$ as

$$(\texttt{diag}(\boldsymbol{v}))\,(i,j) = \boldsymbol{v}(i)\boldsymbol{I}_m(i,j) \quad \text{for } (i,j) \in \{1,\dots,m\}^2$$

Moreover, given a square matrix $\boldsymbol{A} \in \mathbb{R}^{m \times m}$, we define the vector $\texttt{diag}(\boldsymbol{A}) \in \mathbb{R}^m$ as

$$(\texttt{diag}(\boldsymbol{A}))\,(i) = \boldsymbol{A}(i,i) \quad \text{for } i \in \{1,\dots,m\}$$

Given a vector $\boldsymbol{v} \in \mathbb{R}^m$, we represent as $\|\boldsymbol{v}\|$ its 2-norm

$$\|\boldsymbol{v}\| = \left(\sum_{i=1}^{m} \boldsymbol{v}(i)^2 \right)^{1/2} \tag{2.1}$$

whereas for a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ we employ the notations $\|\boldsymbol{A}\|$ and $\|\boldsymbol{A}\|_F$ respectively for the spectral norm

$$\|\boldsymbol{A}\| = \max_{\boldsymbol{x} \in \mathbb{R}^n \setminus \{\boldsymbol{0}\}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \tag{2.2}$$

and for the Frobenius norm

$$\|\boldsymbol{A}\|_F = \left(\sum_{j=1}^{n} \|\boldsymbol{A}(:,j)\|^2 \right)^{1/2} \tag{2.3}$$

It is trivial to verify that the 2-norm and both matrix norms are unitarily invariant: given a matrix $\boldsymbol{U}$ with orthonormal columns (i.e. such that $\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$), it holds that

$$\|\boldsymbol{v}\| = \|\boldsymbol{U}\boldsymbol{v}\| \quad \text{and} \quad |||\boldsymbol{A}||| = |||\boldsymbol{U}\boldsymbol{A}||| \quad \text{for any } \boldsymbol{v} \text{ and } \boldsymbol{A} \text{ of suitable size}$$

with $|||\cdot|||$ being either the Frobenius or the spectral norm. Moreover, we will make use of the following property.

Proposition 2.1 (Submultiplicativity)**.** *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times k}$. Then*

$$|||\boldsymbol{A}\boldsymbol{B}||| \leqslant |||\boldsymbol{A}|||\,|||\boldsymbol{B}||| \tag{2.4}$$

with $|||\cdot|||$ being either the Frobenius or the spectral norm.

Proof. Let us consider the spectral norm first.

If $\boldsymbol{B}$ has rank 0, $\|\boldsymbol{A}\boldsymbol{B}\| = \|\boldsymbol{B}\| = 0$ and the claim holds trivially. Otherwise, we may restrict the maximum in definition (2.2) to vectors which do not belong to the kernel of $\boldsymbol{B}$, to obtain

$$\begin{aligned}
\|\boldsymbol{A}\boldsymbol{B}\| &= \max_{\substack{\boldsymbol{x}\in\mathbb{R}^k\setminus\{\boldsymbol{0}\}\\ \boldsymbol{B}\boldsymbol{x}\neq\boldsymbol{0}}} \frac{\|\boldsymbol{A}\boldsymbol{B}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \\
&\leqslant \max_{\substack{\boldsymbol{x}\in\mathbb{R}^k\setminus\{\boldsymbol{0}\}\\ \boldsymbol{B}\boldsymbol{x}\neq\boldsymbol{0}}} \frac{\|\boldsymbol{A}\boldsymbol{B}\boldsymbol{x}\|}{\|\boldsymbol{B}\boldsymbol{x}\|} \max_{\substack{\boldsymbol{x}\in\mathbb{R}^k\setminus\{\boldsymbol{0}\}\\ \boldsymbol{B}\boldsymbol{x}\neq\boldsymbol{0}}} \frac{\|\boldsymbol{B}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \\
&\leqslant \max_{\boldsymbol{y}\in\mathbb{R}^n\setminus\{\boldsymbol{0}\}} \frac{\|\boldsymbol{A}\boldsymbol{y}\|}{\|\boldsymbol{y}\|} \max_{\boldsymbol{x}\in\mathbb{R}^k\setminus\{\boldsymbol{0}\}} \frac{\|\boldsymbol{B}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \|\boldsymbol{A}\|\,\|\boldsymbol{B}\|
\end{aligned}$$

If $|||\cdot|||$ is the Frobenius norm, we can apply the submultiplicativity of the spectral norm (or equivalently its definition) to obtain

$$\|\boldsymbol{A}\boldsymbol{B}\|_F^2 = \sum_{j=1}^{k} \|\boldsymbol{A}\boldsymbol{B}(:,j)\|^2 \leqslant \sum_{j=1}^{k} \|\boldsymbol{A}\|^2 \|\boldsymbol{B}(:,j)\|^2 = \|\boldsymbol{A}\|^2 \|\boldsymbol{B}\|_F^2$$

□

We also introduce several kinds of matrix-matrix multiplication. Consider $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{m'\times n'}$. We define:

- the Kronecker product $\boldsymbol{A} \otimes \boldsymbol{B}$ as the $(mm') \times (nn')$ matrix with entries

$$(\boldsymbol{A} \otimes \boldsymbol{B})\,(i' + m'(i-1), j' + n'(j-1)) = \boldsymbol{A}(i,j)\boldsymbol{B}(i',j') \tag{2.5}$$

- if $m = m'$ and $n = n'$, the Hadamard product $\boldsymbol{A} * \boldsymbol{B}$ as the $m \times n$ matrix with entries

$$(\boldsymbol{A} * \boldsymbol{B})\,(i,j) = \boldsymbol{A}(i,j)\boldsymbol{B}(i,j) \tag{2.6}$$

- if $n = n'$, the Khatri-Rao product $\boldsymbol{A} \odot \boldsymbol{B}$ as the $(mm') \times n$ matrix with entries

$$(\boldsymbol{A} \odot \boldsymbol{B})\,(i' + m'(i-1), j) = \boldsymbol{A}(i,j)\boldsymbol{B}(i',j) \tag{2.7}$$

- if $m = m'$, the transpose Khatri-Rao product $\boldsymbol{A} \odot^\top \boldsymbol{B}$ as the $m \times (nn')$ matrix given by

$$\boldsymbol{A} \odot^\top \boldsymbol{B} = \left(\boldsymbol{A}^\top \odot \boldsymbol{B}^\top\right)^\top \tag{2.8}$$

In all the definitions above, the indices i, j, i' and j' range over all the possible values for which $\boldsymbol{A}(i, j)$ and $\boldsymbol{B}(i', j')$ make sense.

In the following chapters, we will make extensive use of the bilinearity of all the products defined above, as well as of the following properties of the Kronecker product, which can be proven quite easily by comparing the entries of the two sides of the equations.

Proposition 2.2. *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{B} \in \mathbb{R}^{m'\times n'}$, $\boldsymbol{C} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{D} \in \mathbb{R}^{n'\times k'}$. Then*

$$(\boldsymbol{A} \otimes \boldsymbol{B})^\top = \boldsymbol{A}^\top \otimes \boldsymbol{B}^\top \tag{2.9}$$

and

$$(\boldsymbol{A} \otimes \boldsymbol{B})\,(\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{AC}) \otimes (\boldsymbol{BD}) \tag{2.10}$$

Also, it holds that

$$\boldsymbol{I}_m \otimes \boldsymbol{I}_n = \boldsymbol{I}_{mn} \tag{2.11}$$

2.2 Notation and basic operations with tensors

In this section we review the basic notation for tensors. For any integer $d > 0$ and any tuple of positive integers $(n_1, \ldots, n_d)$ of size d, we represent with a calligraphic bold upper-case letter the tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1\times\ldots\times n_d}$. We indicate single entries of the tensor by generalizing the notation used for matrices: for a multi-index $(i_1, \ldots, i_d) \in \{1, \ldots, n_1\} \times \ldots \times \{1, \ldots, n_d\}$, the corresponding entry of the tensor is $\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d)$.

The extraction of rows and columns can be generalized to tensors, again by using a MATLAB®-like notation: given a d-dimensional tensor $\boldsymbol{\mathcal{A}}$, we may fix any number of indices to obtain a tensor with reduced order, by collapsing some of the modes. For instance, given $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{3\times2\times4\times6}$, we may consider the matrix $\boldsymbol{\mathcal{A}}(:, 1, :, 4) \in \mathbb{R}^{3\times4}$, given by

$$\boldsymbol{\mathcal{A}}(:, 1, :, 4) = \begin{bmatrix} \boldsymbol{\mathcal{A}}(1,1,1,4) & \boldsymbol{\mathcal{A}}(1,1,2,4) & \boldsymbol{\mathcal{A}}(1,1,3,4) & \boldsymbol{\mathcal{A}}(1,1,4,4) \\ \boldsymbol{\mathcal{A}}(2,1,1,4) & \boldsymbol{\mathcal{A}}(2,1,2,4) & \boldsymbol{\mathcal{A}}(2,1,3,4) & \boldsymbol{\mathcal{A}}(2,1,4,4) \\ \boldsymbol{\mathcal{A}}(3,1,1,4) & \boldsymbol{\mathcal{A}}(3,1,2,4) & \boldsymbol{\mathcal{A}}(3,1,3,4) & \boldsymbol{\mathcal{A}}(3,1,4,4) \end{bmatrix}$$

If we fix all the indices except one (resp. two), we talk about *fibers* (resp. *slices*) of the tensor.

Moreover, tensors can be reshaped into matrices using the following operation.

Definition 2.1 (Tensor unfolding)**.** *Given a d-dimensional tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1\times\ldots\times n_d}$ and $j \in \{1, \ldots, d-1\}$, the j^{th} unfolding of $\boldsymbol{\mathcal{A}}$ is the matrix $\boldsymbol{\mathcal{A}}^{<j>} \in \mathbb{R}^{(n_1\cdots n_j)\times(n_{j+1}\cdots n_d)}$ whose entries are*

$$\boldsymbol{\mathcal{A}}^{<j>}(i_{row}(i_1, \ldots, i_j), i_{col}(i_{j+1}, \ldots, i_d)) = \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) \tag{2.12}$$

for $(i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d$. The maps

$$i_{row} : I_1 \times \ldots \times I_j \to \{1, \ldots, n_1 \cdots n_j\} \text{ and } i_{col} : I_{j+1} \times \ldots \times I_d \to \{1, \ldots, n_{j+1} \cdots n_d\}$$

define a colexicographical ordering on $I_1 \times \ldots \times I_j$ *and* $I_{j+1} \times \ldots \times I_d$ *respectively, i.e.*

$$i_{row}(i_1, \ldots, i_j) = 1 + \sum_{\mu=1}^{j} (i_\mu - 1) \prod_{\nu=1}^{\mu-1} n_\nu$$

$$i_{col}(i_{j+1}, \ldots, i_d) = 1 + \sum_{\mu=j+1}^{d} (i_\mu - 1) \prod_{\nu=j+1}^{\mu-1} n_\nu$$

We additionally define the degenerate 0^{th} *unfolding* $\boldsymbol{\mathcal{A}}^{<0>}$ *as the* 1^{st} *unfolding of the tensor* $\boldsymbol{\mathcal{A}}' \in \mathbb{R}^{1 \times n_1 \times \ldots \times n_d}$*, defined entry-wise as*

$$\boldsymbol{\mathcal{A}}'(1, i_1, \ldots, i_d) = \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) \quad \textit{for } (i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d$$

and the d^{th} *unfolding* $\boldsymbol{\mathcal{A}}^{<d>}$ *as the* d^{th} *unfolding of the tensor* $\boldsymbol{\mathcal{A}}'' \in \mathbb{R}^{n_1 \times \ldots \times n_d \times 1}$*, defined entry-wise as*

$$\boldsymbol{\mathcal{A}}''(i_1, \ldots, i_d, 1) = \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) \quad \textit{for } (i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d$$

In particular, $\boldsymbol{\mathcal{A}}^{<0>} \in \mathbb{R}^{1 \times (n_1 \cdots n_d)}$ *and* $\boldsymbol{\mathcal{A}}^{<d>} \in \mathbb{R}^{(n_1 \cdots n_d) \times 1}$.

It is trivial to see that, if we fix the size of a d-dimensional tensor $\boldsymbol{\mathcal{A}}$ and an index $j \in \{0, \ldots, d\}$, the j^{th} unfolding $\boldsymbol{\mathcal{A}}^{<j>}$ uniquely identifies $\boldsymbol{\mathcal{A}}$. Equivalently, for any $j \in \{0, \ldots, d\}$, the "j^{th} unfolding" operation defines a bijection from $\mathbb{R}^{n_1 \times \ldots \times n_d}$ to $\mathbb{R}^{(n_1 \cdots n_j) \times (n_{j+1} \cdots n_d)}$.

In MATLAB® (and in any environment which relies on column-major array storage), the computation of any unfolding of a full tensor can be carried out without moving entries in memory, since data is stored in colexicographical order.

In particular, the conversion from tensor to matrix format (and vice versa) can be easily carried out using the `reshape` command. For instance, the j^{th} unfolding (for $j \in \{0, \ldots, d\}$) of a dimensional tensor $\boldsymbol{\mathcal{A}}$ of size $\boldsymbol{n} = (n_1, \ldots, n_d)$ can be obtained as

```
reshape(A, [prod(n(1:j)), prod(n(j+1:d))])
```

In addition, for any tensor $\boldsymbol{\mathcal{A}}$ we define the Frobenius norm

$$\|\boldsymbol{\mathcal{A}}\|_F = \left(\sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d)^2 \right)^{1/2} \tag{2.13}$$

and we generalize the matrix-vector product operation as follows.

Definition 2.2 (n-mode product)**.** *Given a* d*-dimensional tensor* $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$*, an integer* $j \in \{1, \ldots, d\}$ *and a matrix* $\boldsymbol{M} \in \mathbb{R}^{m \times n_j}$*, the* j^{th} *mode product between* $\boldsymbol{\mathcal{A}}$ *and* $\boldsymbol{M}$ *is the* d*-dimensional tensor* $(\boldsymbol{\mathcal{A}} \times_j \boldsymbol{M}) \in \mathbb{R}^{n_1 \times \ldots \times n_{j-1} \times m \times n_{j+1} \times \ldots \times n_d}$ *such that*

$$(\boldsymbol{\mathcal{A}} \times_j \boldsymbol{M})(i_1, \ldots, i_{j-1}, \alpha, i_{j+1}, \ldots, i_d) = \sum_{i_j=1}^{n_j} \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) \boldsymbol{M}(\alpha, i_j) \tag{2.14}$$

To conclude, we observe that the Kronecker and Hadamard products can be naturally generalized to tensors. Given $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{n'_1 \times \ldots \times n'_d}$, their

Kronecker product is the tensor $\boldsymbol{\mathcal{A}} \otimes \boldsymbol{\mathcal{B}} \in \mathbb{R}^{(n_1 n_1') \times \ldots \times (n_d n_d')}$ with entries

$$(\boldsymbol{\mathcal{A}} \otimes \boldsymbol{\mathcal{B}})\,(i_1' + n_1'(i_1 - 1), \ldots, i_d' + n_d'(i_d - 1)) = \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d)\boldsymbol{\mathcal{B}}(i_1', \ldots, i_d') \quad (2.15)$$

for all suitable multi-indices $(i_1, \ldots, i_d)$ and $(i_1', \ldots, i_d')$.

Moreover, if $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ have the same size, then we can define their Hadamard product $\boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ entry-wise as

$$(\boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}})\,(i_1, \ldots, i_d) = \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d)\boldsymbol{\mathcal{B}}(i_1, \ldots, i_d) \quad (2.16)$$

Chapter 3

Matrix compression

The general low-rank approximation problem for matrices may be formulated as follows: given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ (we assume $m \geqslant n$ without loss of generality) and an integer $r \leqslant n$, we look for two matrices $\boldsymbol{W} \in \mathbb{R}^{m \times r}$ and $\boldsymbol{Z} \in \mathbb{R}^{n \times r}$, such that the error

$$\epsilon(\boldsymbol{W}, \boldsymbol{Z}) = |||\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top||| \tag{3.1}$$

is minimized, with $|||\cdot|||$ being either the spectral or the Frobenius norm.

For unitarily invariant norms (e.g. the two norms considered above), the Eckart-Young and Mirsky theorems (see e.g. [EY36] and [Mir60]) guarantee that the problem of finding the two factors $\boldsymbol{W}$ and $\boldsymbol{Z}$ always admits an optimal solution. In particular, one such solution may be found by truncating (at rank r) the singular value decomposition of $\boldsymbol{A}$: indeed, if

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top \tag{3.2}$$

is the SVD of $\boldsymbol{A}$, then the minimum of ϵ over $\mathbb{R}^{m \times r} \times \mathbb{R}^{n \times r}$ is equal to

$$\epsilon\left(\boldsymbol{W}_{(r)}, \boldsymbol{Z}_{(r)}\right) = |||\boldsymbol{\Sigma}((r+1):n, (r+1):n)|||$$

with the optimal factors being

$$\boldsymbol{W}_{(r)} = \boldsymbol{U}(:, 1:r) \quad \text{and} \quad \boldsymbol{Z}_{(r)} = \boldsymbol{V}(:, 1:r)\boldsymbol{\Sigma}(1:r, 1:r) \tag{3.3}$$

It is trivial to see that this solution is not unique (except when $\boldsymbol{A}$ has rank 0). Indeed, for any non-singular matrix $\boldsymbol{M} \in \mathbb{R}^{r \times r}$, the factors

$$\boldsymbol{W}^*_{(r)} = \boldsymbol{W}_{(r)}\boldsymbol{M} \quad \text{and} \quad \boldsymbol{Z}^*_{(r)} = \boldsymbol{Z}_{(r)}\boldsymbol{M}^{-\top}$$

are still optimal, since $\boldsymbol{W}^*_{(r)}\ \boldsymbol{Z}^*_{(r)}{}^\top = \boldsymbol{W}_{(r)}\boldsymbol{Z}^\top_{(r)}$.

Given this result, a naive algorithm to find a low-rank approximation for a general unstructured matrix of size $m \times n$ may be based on the explicit computation (and successive truncation) of the full SVD of the matrix. The resulting computational cost is $\mathcal{O}\left(mn^2\right)$ flops, independently of r.

More advanced techniques rely on a RRQR factorization (see e.g. [Cha87]) to identify the dominant range of the matrix, and then manipulate the results to obtain the desired low-rank approximation. For most matrices these approaches are quite efficient, having a complexity of $\mathcal{O}(mnr)$ flops, but in some extreme cases (see e.g. [GE96]) their cost is the same as the computation of the full SVD, i.e. $\mathcal{O}\left(mn^2\right)$ flops.

In comparison, the randomized algorithm presented in the upcoming section has a computational cost approximately equal to $\mathcal{O}(mnr)$, and has the benefit of being robust. The disadvantage of this stochastic approach is, of course, the presence of an intrinsically positive probability that the algorithm will fail.

3.1 Randomized matrix compression with fixed rank

Due to the orthonormality of the columns of $\boldsymbol{U}$, it is trivial to see that the optimal low-rank approximation (3.3) of the matrix $\boldsymbol{A}$ may be obtained by projecting the matrix onto the range of $\boldsymbol{U}(:, 1:r)$, i.e. the span of the r dominant left-singular vectors of $\boldsymbol{A}$:

$$\boldsymbol{W}_{(r)}\boldsymbol{Z}_{(r)}^{\top} = \boldsymbol{U}(:, 1:r)\boldsymbol{\Sigma}(1:r, 1:r)\boldsymbol{V}(:, 1:r)^{\top} = \boldsymbol{U}(:, 1:r)\boldsymbol{U}(:, 1:r)^{\top}\boldsymbol{A}$$

Hence, we can find a low-rank approximation in two steps: first, we compute a left factor $\boldsymbol{W}$ (with full rank) whose range is an approximation of the range of $\boldsymbol{U}(:, 1:r)$. Then we find the corresponding right factor $\boldsymbol{Z}$ as

$$\boldsymbol{Z} = \boldsymbol{A}^{\top}\boldsymbol{W}\left(\boldsymbol{W}^{\top}\boldsymbol{W}\right)^{-1}$$

so that

$$\boldsymbol{W}\boldsymbol{Z}^{\top} = \boldsymbol{W}(\boldsymbol{W}^{\top}\boldsymbol{W})^{-1}\boldsymbol{W}^{\top}\boldsymbol{A} \approx \boldsymbol{U}(:, 1:r)\boldsymbol{U}(:, 1:r)^{\top}\boldsymbol{A}$$

since $\boldsymbol{W}(\boldsymbol{W}^{\top}\boldsymbol{W})^{-1}\boldsymbol{W}^{\top}$ represents the projection matrix onto the range of $\boldsymbol{W}$.

In particular, we remark that the term $(\boldsymbol{W}^{\top}\boldsymbol{W})^{-1}$ may be omitted if $\boldsymbol{W}$ has orthonormal columns. To avoid the computation of this term, in the following we will always require that the left factor $\boldsymbol{W}$ is properly orthogonalized.

Thus, the problem is now reduced to finding an approximation of the dominant subspace (of dimension r) of the range of $\boldsymbol{A}$. To this aim, we follow the intuition provided in [HMT11]: let

$$\boldsymbol{E} = \boldsymbol{A} - \boldsymbol{W}_{(r)}\boldsymbol{Z}_{(r)}^{\top}$$

represent the perturbation from the desired low-rank approximation. In particular, due to the properties of the truncated SVD, the range of $\boldsymbol{E}$ and the range of $\boldsymbol{W}_{(r)}$ are orthogonal, i.e. $\boldsymbol{W}_{(r)}^{\top}\boldsymbol{E} = \boldsymbol{0}$.

Given a family of independent Gaussian random vectors $\{\boldsymbol{\omega}^{(l)}\}_{l=1}^{n} \subset \mathbb{R}^n$, we may randomly sample the range of $\boldsymbol{A}$ as follows:

$$\mathbf{y}^{(l)} = \boldsymbol{A}\boldsymbol{\omega}^{(l)} = \boldsymbol{W}_{(r)}\boldsymbol{Z}_{(r)}^{\top}\boldsymbol{\omega}^{(l)} + \boldsymbol{E}\boldsymbol{\omega}^{(l)} \quad \text{for } l = 1, \dots, n$$

Almost surely, the action of the perturbation $\boldsymbol{E}$ onto the random vectors makes each sample fall outside the range of $\boldsymbol{W}_{(r)}$. Accordingly, depending on the relative magnitude of the singular values of $\boldsymbol{A}$, the span of just r samples $\{\mathbf{y}^{(l)}\}_{l=1}^{r}$ may provide quite a poor approximation for the range of $\boldsymbol{W}_{(r)}$. To reduce the error, we can set an oversampling parameter $p > 0$ and use an enriched set of samples $\{\mathbf{y}^{(l)}\}_{l=1}^{r+p}$, which yields a much better approximation (see Theorem 3.2).

Now we compute (e.g. through a QR decomposition) the matrix $\widetilde{\boldsymbol{W}} \in \mathbb{R}^{m\times(r+p)}$, whose columns form an orthonormal basis of the span of $\{\mathbf{y}^{(l)}\}_{j=1}^{r+p}$. This leads to a

rank-$(r+p)$ approximation of the form

$$\boldsymbol{A} \approx \widetilde{\boldsymbol{A}} = \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top \boldsymbol{A} = \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{Z}}^\top$$

In order to shrink this approximation to rank r, we carry out a *recompression* step by computing the best rank-r approximation of $\widetilde{\boldsymbol{A}}$. In particular, we observe that, since $\widetilde{\boldsymbol{W}}$ has orthonormal columns, the SVD of $\widetilde{\boldsymbol{A}}$ is of the form

$$\widetilde{\boldsymbol{A}} = \left(\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{U}}\right)\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}^\top$$

for some orthogonal matrix $\widetilde{\boldsymbol{U}} \in \mathbb{R}^{(r+p)\times(r+p)}$. Hence, it suffices to find the best rank-r approximation of $\widetilde{\boldsymbol{Z}}^\top$, and then multiply it by $\widetilde{\boldsymbol{W}}$ from the left.

This time we can explicitly compute the SVD of the "tall" matrix $\widetilde{\boldsymbol{Z}} \in \mathbb{R}^{n\times(r+p)}$, and truncate it at rank r. The resulting factors for the approximation of $\boldsymbol{A}$ are

$$\boldsymbol{W} = \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{U}}(:, 1:r) \quad \text{and} \quad \boldsymbol{Z} = \widetilde{\boldsymbol{V}}(:, 1:r)\widetilde{\boldsymbol{\Sigma}}(1:r, 1:r)$$

with $\widetilde{\boldsymbol{V}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{U}}^\top$ being the SVD of $\widetilde{\boldsymbol{Z}}$.

The resulting procedure is summarized in Algorithm 1.

Algorithm 1: Randomized compression with fixed rank

Data: matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, positive integers r and p (with $r < n$ and $p + r \leqslant n$)
Result: $\boldsymbol{W} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{Z} \in \mathbb{R}^{n\times r}$, with $\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{I}$ and $\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{Z}^\top$

1 Generate Gaussian matrix $\boldsymbol{\Omega} \in \mathbb{R}^{n\times(r+p)}$
2 $\boldsymbol{Y} := \boldsymbol{A}\boldsymbol{\Omega}$
3 Compute QR decomposition: $\boldsymbol{Y} =: \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{R}}$
4 $\widetilde{\boldsymbol{Z}} := \boldsymbol{A}^\top\widetilde{\boldsymbol{W}}$
5 Compute SVD: $\widetilde{\boldsymbol{Z}} =: \widetilde{\boldsymbol{V}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{U}}^\top$
6 $\boldsymbol{W} := \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{U}}(:, 1:r)$
7 $\boldsymbol{Z} := \widetilde{\boldsymbol{V}}(:, 1:r)\widetilde{\boldsymbol{\Sigma}}(1:r, 1:r)$

Assuming that $\boldsymbol{A}$ is treated as a general dense matrix, the total computational cost is

$$\mathcal{O}(n(r+p)T_{rng} + mn(r+p))$$

flops, with T_{rng} being the cost of drawing a single random sample. If we assume (as we will do from here onward) that the cost of generating random samples is negligible, the overall complexity is $\mathcal{O}(mn(r+p))$. In particular, when $r + p \ll n$ (which is usually the case in practice), the complexity of the algorithm is dominated by the cost of computing the two products $\boldsymbol{A}\boldsymbol{\Omega}$ and $\boldsymbol{A}^\top\widetilde{\boldsymbol{W}}$.

More generally, if the costs of matrix-vector multiplications by $\boldsymbol{A}$ and $\boldsymbol{A}^\top$ are respectively T_{mult} and T'_{mult}, the complexity of the algorithm is

$$\mathcal{O}\big((T_{mult} + T'_{mult})(r+p) + m(r+p)^2\big)$$

3.1.1 Theoretical bounds for the randomized algorithm

The compressed matrix resulting from Algorithm 1 satisfies the following *a priori* bounds.

Theorem 3.1. *Let $\boldsymbol{W}$ and $\boldsymbol{Z}$ be the two factors computed by Algorithm 1, and assume $p \geqslant 2$. If $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_n \geqslant 0$ are the singular values of $\boldsymbol{A}$, then*

$$\mathbb{E}\|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\|_F \leqslant \left(1 + \sqrt{1 + \frac{r}{p-1}}\right)\left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2} \tag{3.4}$$

and

$$\mathbb{E}\|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\| \leqslant \left(2 + \sqrt{\frac{r}{p-1}}\right)\sigma_{r+1} + \frac{e\sqrt{r+p}}{p}\left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2} \tag{3.5}$$

Proof. Let $\widetilde{\boldsymbol{W}}$ and $\tilde{\boldsymbol{Z}}$ be defined as in Algorithm 1.

Using the triangular inequality we can write

$$\begin{aligned}|||\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top||| &\leqslant |||\boldsymbol{A} - \widetilde{\boldsymbol{W}}\tilde{\boldsymbol{Z}}^\top||| + |||\widetilde{\boldsymbol{W}}\tilde{\boldsymbol{Z}}^\top - \boldsymbol{W}\boldsymbol{Z}^\top||| \\ &= |||\boldsymbol{A} - \widetilde{\boldsymbol{W}}\tilde{\boldsymbol{Z}}^\top||| + |||\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top|||\end{aligned}$$

with $|||\cdot|||$ being either the Frobenius or spectral norm.

We can bound the expected value of the first term by using Theorems 10.5 and 10.6 in [HMT11], which state that

$$\mathbb{E}\|\boldsymbol{A} - \widetilde{\boldsymbol{W}}\tilde{\boldsymbol{Z}}^\top\|_F \leqslant \left(1 + \frac{r}{p-1}\right)^{1/2}\left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2}$$

and

$$\mathbb{E}\|\boldsymbol{A} - \widetilde{\boldsymbol{W}}\tilde{\boldsymbol{Z}}^\top\| \leqslant \left(1 + \sqrt{\frac{r}{p-1}}\right)\sigma_{r+1} + \frac{e\sqrt{r+p}}{p}\left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2}$$

To bound the second term, we observe that $\boldsymbol{W}\boldsymbol{Z}^\top$ is, by construction, the best rank-r approximation of $\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\boldsymbol{A}$ (in both the Frobenius and the spectral norm, due to the Mirsky theorem). Hence

$$|||\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top||| = \min_{\substack{\boldsymbol{B}\in\mathbb{R}^{m\times n} \\ \operatorname{rank}\boldsymbol{B}\leqslant r}} |||\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\boldsymbol{A} - \boldsymbol{B}||| \leqslant |||\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\boldsymbol{A} - \boldsymbol{C}|||$$

for any $\boldsymbol{C} \in \mathbb{R}^{m\times n}$ with rank at most r.

In particular, we may pick $\boldsymbol{C} = \widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\boldsymbol{A}_{(r)}$, with $\boldsymbol{A}_{(r)}$ being the best rank-r approximation of $\boldsymbol{A}$. This yields

$$|||\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top||| \leqslant |||\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\left(\boldsymbol{A} - \boldsymbol{A}_{(r)}\right)|||$$

By orthogonality, it holds that

$$\|\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top\| = \|\boldsymbol{I}_{r+p}\| = 1$$

and we can apply the submultiplicativity of $|||\cdot|||$ (see Proposition 2.1) to obtain

$$|||\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top \left(\boldsymbol{A} - \boldsymbol{A}_{(r)}\right)||| \leqslant |||\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top|||\,|||\boldsymbol{A} - \boldsymbol{A}_{(r)}||| = |||\boldsymbol{A} - \boldsymbol{A}_{(r)}|||$$

or equivalently

$$\|\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top \left(\boldsymbol{A} - \boldsymbol{A}_{(r)}\right)\|_F \leqslant \left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2} \quad \text{and} \quad \|\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{W}}^\top \left(\boldsymbol{A} - \boldsymbol{A}_{(r)}\right)\| \leqslant \sigma_{r+1}$$

Hence, the expected value of the residual can be bounded as

$$\mathbb{E}\|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\|_F \leqslant \left(1 + \frac{r}{p-1}\right)^{1/2} \left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2} + \left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2}$$

and

$$\mathbb{E}\|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\| \leqslant \left(1 + \sqrt{\frac{r}{p-1}}\right) \sigma_{r+1} + \frac{e\sqrt{r+p}}{p} \left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2} + \sigma_{r+1}$$

□

For fixed values of r and p, Theorem 3.1 proves that Algorithm 1 is (in average) quasi-optimal in the Frobenius norm, but not in the spectral norm. For this reason, the application of the Frobenius norm appears more natural in the framework of randomized low-rank approximation.

We also have the following estimates on the failure probability of the algorithm.

Theorem 3.2. *Consider the hypotheses of Theorem 3.1, with $p \geqslant 4$. Then it holds*

$$\|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\|_F \leqslant \left(2 + 10\sqrt{\frac{r}{p}}\right) \left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2} + 11\sqrt{1 + \frac{r}{p}}\,\sigma_{r+1} \tag{3.6}$$

except with probability $7e^{-p}$ at most, and

$$\|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\| \leqslant 8\frac{\sqrt{r+p}}{p+1} \left(\sum_{j=r+1}^{n} \sigma_j^2\right)^{1/2} + \left(2 + 20\sqrt{1 + \frac{r}{p}}\right) \sigma_{r+1} \tag{3.7}$$

except with probability $6e^{-p}$ at most.

Proof. The proof is analogous to the one of Theorem 3.1, with the only difference that we use Theorems 10.7 and 10.8 (with the values $t = e$ and $u = \sqrt{2p}$) from [HMT11] instead of 10.5 and 10.6. □

3.2 The inverse problem: low-rank approximation with fixed precision

In most applications, the target rank r is not known in advance. Instead, we may wish to compress a matrix with fixed precision: given $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ (with $m \geqslant n$) and

$\varepsilon > 0$, we seek $\boldsymbol{W} \in \mathbb{R}^{m \times \bar{r}}$ and $\boldsymbol{Z} \in \mathbb{R}^{n \times \bar{r}}$ such that

$$\frac{\|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\|}{\|\boldsymbol{A}\|} \leqslant \varepsilon \tag{3.8}$$

with $\bar{r}$ being the smallest integer such that (3.8) is feasible.

A simple, deterministic way to solve this problem starts with the explicit computation of the SVD of $\boldsymbol{A}$. Then the value of $\bar{r}$ can be deduced from its singular values ($\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_n \geqslant 0$), as the smallest integer which satisfies

$$\sum_{j=\bar{r}+1}^{n} \sigma_j^2 \leqslant \varepsilon^2 \sum_{j=1}^{n} \sigma_j^2$$

for the Frobenius norm, or

$$\sigma_{\bar{r}+1} \leqslant \varepsilon\, \sigma_1$$

for the spectral norm, and the two factors can be computed by truncating the SVD at rank $\bar{r}$.

Similarly to the fixed-rank case, this process can be sped up by applying a RRQR decomposition. This typically reduces the complexity from $\mathcal{O}\left(mn^2\right)$ to $\mathcal{O}(mn\bar{r})$, at the cost of a decrease in the robustness of the algorithm.

A randomized approach, obtained as a suitable generalization of Algorithm 1, can achieve both efficiency and robustness. To derive such an algorithm, the main issue is finding an inexpensive way to compute or estimate the relative error (3.8). Indeed, once such an estimate is available, it suffices to apply the following adaptive approach: we proceed by using the framework described in Section 3.1, and we keep adding new samples from the range of $\boldsymbol{A}$ until the approximation $\boldsymbol{W}\boldsymbol{Z}^\top$ achieves the desired precision.

Accordingly, we devote the upcoming sections to the description of several approaches to tackle this issue.

3.2.1 Bounds for the Frobenius norm

Assume that the relative precision is specified in terms of the Frobenius norm.

Let $\boldsymbol{W}\boldsymbol{Z}^\top$ be a rank-r approximation of $\boldsymbol{A}$, with $\boldsymbol{W}$ having orthonormal columns, and let $\boldsymbol{Z} = \boldsymbol{A}^\top\boldsymbol{W}$. We wish to verify if (3.8) is satisfied.

To this aim, we can observe that the following orthogonal decomposition holds:

$$\boldsymbol{A} = \boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top + \boldsymbol{W}\boldsymbol{Z}^\top = \left(\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top\right)\boldsymbol{A} + \boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{A}$$

Indeed, $\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top$ and $\boldsymbol{W}\boldsymbol{W}^\top$ are projection matrices onto orthogonal subspaces of $\mathbb{R}^m$.

Thus, the Pythagorean theorem yields

$$\|\boldsymbol{A}\|_F^2 = \|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\|_F^2 + \|\boldsymbol{W}\boldsymbol{Z}^\top\|_F^2 \tag{3.9}$$

and (3.8) is equivalent to

$$\|\boldsymbol{A}\|_F^2 - \|\boldsymbol{W}\boldsymbol{Z}^\top\|_F^2 \leqslant \varepsilon^2\, \|\boldsymbol{A}\|_F^2 \tag{3.10}$$

Let us consider the term $\|\boldsymbol{W}\boldsymbol{Z}^\top\|_F$. Since $\boldsymbol{W}$ has orthonormal columns, it is sufficient to compute the norm of the "tall" right factor $\boldsymbol{Z}$. If we also allow the explicit computation of $\|\boldsymbol{A}\|_F$ (which needs to be carried out only once), then we obtain the procedure summarized in Algorithm 2.

Algorithm 2: Randomized compression with fixed (Frobenius) precision with explicit computation of $\|\boldsymbol{A}\|_F$

Data: matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\varepsilon > 0$
Result: $\boldsymbol{W} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{Z} \in \mathbb{R}^{n\times r}$, with $\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{I}_m$ and such that (3.8) holds for the Frobenius norm

```
Compute ‖A‖_F
W := [ ], Z := [ ]
while ‖A‖_F^2 − ‖Z‖_F^2 > ε^2 ‖A‖_F^2 do
    Generate Gaussian vector ω ∈ R^n
    w := (I_m − W W^⊤) A ω
    w := w/‖w‖
    W := [W | w]
    z := A^⊤ w
    Z := [Z | z]
end
```

Since each iteration adds a single column to $\boldsymbol{Z}$, the computation of $\|\boldsymbol{Z}\|_F$ can be carried out recursively in merely $\mathcal{O}(nr)$ total operations. If $\boldsymbol{A}$ is stored as a full matrix, the computation of its Frobenius norm takes $\mathcal{O}(mn)$ flops, not affecting the overall complexity $\mathcal{O}(mnr)$, with the dominant operations being, as in the fixed-rank case, the multiplications $\boldsymbol{A}\boldsymbol{\omega}$ and $\boldsymbol{A}^\top\boldsymbol{w}$.

However, due to the randomness of the approach, with high probability the final rank r will be larger than the optimal one $\bar{r}$. In some cases (see e.g. Section 3.3.3), the algorithm may overshoot the optimal rank $\bar{r}$ by a considerable amount. As such, it may be advisable to add a recompression step at the end of the algorithm:

1. Compute the SVD of $\boldsymbol{Z}$: $\boldsymbol{Z} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\top$.
2. Find the smallest r' which satisfies

$$\|\boldsymbol{A}\|_F^2 - \sum_{j=1}^{r'} \sigma_j^2 \leqslant \varepsilon^2 \|\boldsymbol{A}\|_F^2 \tag{3.11}$$

   with $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_r \geqslant 0$ being the diagonal entries of $\boldsymbol{\Sigma}$.
3. Update $\boldsymbol{W}^* = \boldsymbol{W}\boldsymbol{U}(:, 1:r')$ and $\boldsymbol{Z}^* = \boldsymbol{V}(:, 1:r')\boldsymbol{\Sigma}(1:r', 1:r')$.

Error estimation without the norm of the full matrix

In many cases of interest, computing $\|\boldsymbol{A}\|_F$ is much more expensive than $\mathcal{O}(mn)$ flops. For instance, assume that $\boldsymbol{A}$ is not stored in memory, and that we can access it only through matrix-vector multiplications. In this case, computing the Frobenius norm, or even estimating it with reasonable accuracy, would be much more expensive than the rest of the algorithm.

To tackle these situations, we proceed to reformulate (3.10) in a cheaper way. We start by providing a lower-bound for the denominator: using (3.9), we obtain

$$\|\boldsymbol{A}\|_F^2 = \|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\|_F^2 + \|\boldsymbol{W}\boldsymbol{Z}^\top\|_F^2 \geqslant \|\boldsymbol{W}\boldsymbol{Z}^\top\|_F^2 = \|\boldsymbol{Z}\|_F^2$$

In particular, as the algorithm goes on, the ranks of $\boldsymbol{W}$ and $\boldsymbol{Z}$ increase, and the discrepancy between $\|\boldsymbol{Z}\|_F$ and $\|\boldsymbol{A}\|_F$ becomes progressively smaller.

This fact proves extremely useful in the estimation of the numerator. Assume that $\widetilde{\boldsymbol{W}}$ is the matrix (with orthonormal columns) obtained by carrying out $q > 0$ more steps of the algorithm (i.e. it has q more columns than $\boldsymbol{W}$, and $\widetilde{\boldsymbol{W}} = [\boldsymbol{W}|\ldots]$), and let $\widetilde{\boldsymbol{Z}} = \boldsymbol{A}^\top\widetilde{\boldsymbol{W}}$. Since $\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{Z}}^\top$ is a "richer" approximation of $\boldsymbol{A}$, we use its norm to approximate the value of $\|\boldsymbol{A}\|_F$.

In particular, a suitable value of q should be chosen using the prior knowledge on the spectrum of $\boldsymbol{A}$. If we expect the singular values of $\boldsymbol{A}$ to decay exponentially (see e.g. Section 3.3.2), it is enough to choose a small q, whereas a larger value should be used for slower decays (see e.g. Section 3.3.3).

In summary, as an approximation of (3.10), we enforce the condition

$$\|\widetilde{\boldsymbol{Z}}\|_F^2 - \|\boldsymbol{Z}\|_F^2 \leqslant \varepsilon^2\,\|\widetilde{\boldsymbol{Z}}\|_F^2 \tag{3.12}$$

This leads to Algorithm 3, whose complexity is again $\mathcal{O}(mnr)$.

Algorithm 3: Randomized compression with fixed (Frobenius) precision without computing $\|\boldsymbol{A}\|_F$

Data: matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\varepsilon > 0$, integer $q > 0$
Result: $\boldsymbol{W} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{Z} \in \mathbb{R}^{n\times r}$, with $\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{I}_m$ and such that (3.8) holds with high probability for the Frobenius norm

Generate Gaussian matrix $\boldsymbol{\Omega} \in \mathbb{R}^{n\times q}$
$\boldsymbol{Y} := \boldsymbol{A}\boldsymbol{\Omega}$
Compute QR decomposition: $\boldsymbol{Y} =: \boldsymbol{W}\boldsymbol{R}$
$\boldsymbol{Z} := \boldsymbol{A}^\top\boldsymbol{W}$
$r := q$
while $\|\boldsymbol{Z}\|_F^2 - \|\boldsymbol{Z}(:, 1:(r-q)\|_F^2 > \varepsilon^2\|\boldsymbol{Z}\|_F^2$ **do**
  $r := r + 1$
  Generate Gaussian vector $\boldsymbol{\omega} \in \mathbb{R}^n$
  $\boldsymbol{w} := \left(\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top\right)\boldsymbol{A}\boldsymbol{\omega}$
  $\boldsymbol{w} := \boldsymbol{w}/\|\boldsymbol{w}\|$
  $\boldsymbol{W} := [\boldsymbol{W}\,|\,\boldsymbol{w}]$
  $\boldsymbol{z} := \boldsymbol{A}^\top\boldsymbol{w}$
  $\boldsymbol{Z} := [\boldsymbol{Z}\,|\,\boldsymbol{z}]$
end

In particular, we remark that $\widetilde{\boldsymbol{W}}$ and $\widetilde{\boldsymbol{Z}}$ are used instead of $\boldsymbol{W}$ and $\boldsymbol{Z}$ as the final low-rank factors, since $\widetilde{\boldsymbol{W}}\widetilde{\boldsymbol{Z}}^\top$ always represents a better approximation of $\boldsymbol{A}$.

As above, we may wish to add a recompression step at the end of the procedure. However, since this time the Frobenius norm of $\boldsymbol{A}$ is not known, we need to replace $\|\boldsymbol{A}\|_F$ in (3.11) with $\|\widetilde{\boldsymbol{Z}}\|_F$.

3.2.2 Bounds for the spectral norm

If the spectral norm is used to prescribe the desired accuracy, we need to proceed differently. In this case, it is always possible to approximate very efficiently the spectral norm of the full matrix $\boldsymbol{A}$ using few iterations, say k, of the power method (see e.g. [GL89]), using $\mathcal{O}(mnk)$ flops in total (if $\boldsymbol{A}$ is stored as a full matrix).

However, it is difficult to estimate the absolute error

$$\|\boldsymbol{A} - \boldsymbol{W}\boldsymbol{Z}^\top\| = \|\left(\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top\right)\boldsymbol{A}\|$$

without increasing the complexity of the algorithm. Indeed, such an approximation needs to be computed approximately r times (once for every new sample used to update $\boldsymbol{W}$), and using the power method each time would lead to the increased complexity $\mathcal{O}(mnrk)$.

We may tackle this issue by adding more than one random sample at a time, as described in Algorithm 4: given the sampling step $q > 0$, we only need to apply the power method r/q times, leading to an overall complexity of $\mathcal{O}(mnr\,(1 + k/q))$.

Algorithm 4: Randomized compression with fixed (spectral) precision with the power method

Data: matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\varepsilon > 0$, integer $q > 0$
Result: $\boldsymbol{W} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{Z} \in \mathbb{R}^{n\times r}$, with $\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{I}_m$ and such that (3.8) holds for the spectral norm

```
Compute ‖A‖
W := [ ], Z := [ ]
while ‖(I_m − WW^⊤)A‖ > ε‖A‖ do
    for l = 1, ..., q do
        Generate Gaussian vector ω ∈ R^n
        w := (I_m − WW^⊤) Aω
        w := w/‖w‖
        W := [W | w]
        z := A^⊤ w
        Z := [Z | z]
    end
    Compute ‖(I_m − WW^⊤)A‖
end
```

Error estimation with a randomized approach

A different approach, which does not rely on a staggered sampling of the error, can be derived as described in [HMT11]: instead of computing the actual value of $\|\left(\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top\right)\boldsymbol{A}\|$, we approximate it using a randomized estimator, which is much cheaper to evaluate.

In particular, we can estimate the spectral norm of a matrix by using samples from its range, as shown by the following result, which is a generalization of Lemma 4.1 in [HMT11] and Theorem 1 in [Dix83].

Theorem 3.3. *Let $\boldsymbol{B} \in \mathbb{R}^{m\times n}$, and consider $q > 0$ i.i.d. random vectors in $\{\boldsymbol{\omega}^{(l)}\}_{l=1}^{q} \subset \mathbb{R}^n$, which follow a unitarily invariant distribution $\mathcal{D}$. Then, for any $\theta > 0$ it holds that*

$$\|\boldsymbol{B}\| \leqslant \theta \max_{l=1,\dots,q} \|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\| \tag{3.13}$$

except with probability $\Pr\left(|\omega| < \frac{1}{\theta}\right)^q$ at most, with ω being a random variable distributed as one of the components of $\boldsymbol{\omega}^{(1)}$.

Proof. We follow the idea of the proof of Theorem 1 in [Dix83].

If $\|\boldsymbol{B}\| = 0$, the claim is trivial. Assume $\|\boldsymbol{B}\| > 0$ and let $l \in \{1, \dots, q\}$. We consider the (full) SVD of $\boldsymbol{B}$:

$$\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

and compute the components of $\boldsymbol{\omega}^{(l)}$ along the right singular vectors of $\boldsymbol{B}$:

$$\boldsymbol{\xi} = \boldsymbol{V}^\top\boldsymbol{\omega}^{(l)} \qquad \text{i.e.} \qquad \boldsymbol{\omega}^{(l)} = \boldsymbol{V}\boldsymbol{\xi}$$

Since $\mathcal{D}$ is unitarily invariant, $\boldsymbol{\xi}$ still follows the same distribution as $\boldsymbol{\omega}^{(l)}$.

We can express the norm of $\boldsymbol{B}\boldsymbol{\omega}^{(l)}$ as

$$\|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\| = \|\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top\boldsymbol{\omega}^{(l)}\| = \|\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\xi}\| = \|\boldsymbol{\Sigma}\boldsymbol{\xi}\| = \left(\sum_{j=1}^{n} \boldsymbol{\xi}(j)^2\boldsymbol{\Sigma}(j,j)^2\right)^{1/2}$$

which implies that

$$\|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\| \geqslant |\boldsymbol{\xi}(1)|\,\boldsymbol{\Sigma}(1,1) = |\boldsymbol{\xi}(1)|\|\boldsymbol{B}\|$$

Hence, the event $\{\|\boldsymbol{B}\| > \theta\|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\|\}$ implies $\{\|\boldsymbol{B}\| > \theta|\boldsymbol{\xi}(1)|\|\boldsymbol{B}\|\}$, and we can derive a bound on the failure probability

$$\begin{aligned}\Pr\left(\|\boldsymbol{B}\| > \theta\|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\|\right) &\leqslant \Pr\left(\|\boldsymbol{B}\| > \theta|\boldsymbol{\xi}(1)|\|\boldsymbol{B}\|\right)\\ &= \Pr\left(1 > \theta|\boldsymbol{\xi}(1)|\right)\\ &= \Pr\left(|\omega| < \frac{1}{\theta}\right)\end{aligned}$$

with ω being a random variable distributed as $\boldsymbol{\xi}(1)$.

Using the independence of the samples $\{\boldsymbol{\omega}^{(l)}\}_{l=1}^{q}$, we derive

$$\begin{aligned}\Pr\left(\|\boldsymbol{B}\| > \theta \max_{l=1,\dots,q} \|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\|\right) &= \Pr\left(\bigcap_{l=1}^{q}\{\|\boldsymbol{B}\| > \theta\|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\|\}\right) =\\ &= \prod_{l=1}^{q}\Pr\left(\|\boldsymbol{B}\| > \theta\|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\|\right) \leqslant \Pr\left(|\omega| < \frac{1}{\theta}\right)^q\end{aligned}$$

which, after taking the complement, implies the claim. □

In particular, since we have chosen to apply Gaussian vectors to solve the compression problem, we can restrict Theorem 3.3 to the Gaussian case (which is unitarily invariant) and obtain the following corollary.

Corollary 3.3.1. *Let $\boldsymbol{B} \in \mathbb{R}^{m\times n}$, and consider $q > 0$ independent Gaussian random vectors $\{\boldsymbol{\omega}^{(l)}\}_{l=1}^{q} \subset \mathbb{R}^n$. Then, for any $\alpha > 0$ it holds that*

$$\|\boldsymbol{B}\| \leqslant \sqrt{\frac{2}{\pi}}\,\alpha \max_{l=1,\dots,q} \|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\| \tag{3.14}$$

except with probability α^{-q} at most.

Proof. If $\alpha \leqslant 1$, the claim holds trivially, since the bound on the failure probability is at least 1.

Let $\alpha > 1$. We define $\theta > 0$ as the unique value which satisfies

$$\alpha^{-1} = \Pr\left(|\omega| < \frac{1}{\theta}\right)$$

with ω being a 1-dimensional Gaussian random variable. By applying Theorem 3.3 with this choice of θ, we obtain

$$\Pr\left(\|\boldsymbol{B}\| \leqslant \theta \max_{l=1,\dots,q} \|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\|\right) \geqslant 1 - \alpha^{-q}$$

Now, we observe that

$$\alpha^{-1} = \int_{-1/\theta}^{1/\theta} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \mathrm{d}x \leqslant \int_{-1/\theta}^{1/\theta} \frac{1}{\sqrt{2\pi}} \mathrm{d}x = \sqrt{\frac{2}{\pi}}\,\theta^{-1}$$

which implies that $\theta \leqslant \sqrt{\frac{2}{\pi}}\alpha$.

Thus, it holds

$$\Pr\left(\|\boldsymbol{B}\| \leqslant \sqrt{\frac{2}{\pi}}\,\alpha \max_{l=1,\dots,q} \|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\|\right) \geqslant \Pr\left(\|\boldsymbol{B}\| \leqslant \theta \max_{l=1,\dots,q} \|\boldsymbol{B}\boldsymbol{\omega}^{(l)}\|\right) \geqslant 1 - \alpha^{-q}$$

□

Using this result, we can devise a very efficient procedure to compress a matrix with fixed precision (in the spectral norm), on the wake of Algorithm 4.2 in [HMT11]. Assume that $\boldsymbol{W} \in \mathbb{R}^{m\times k}$ is the current approximation (with orthonormal columns) of the left factor, so that $\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{W}^\top\boldsymbol{A}$. To estimate the current error, we consider $q > 0$ independent Gaussian vectors $\{\boldsymbol{\omega}^{(l)}\}_{l=1}^{q} \subset \mathbb{R}^n$ and we compute

$$10\sqrt{\frac{2}{\pi}} \max_{l=1,\dots,q} \|\left(\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top\right)\boldsymbol{A}\boldsymbol{\omega}^{(l)}\|$$

If the result is smaller than $\varepsilon\|\boldsymbol{A}\|$, we stop the algorithm, since the desired precision ε is achieved (with probability at least $1 - 10^{-q}$). Otherwise, we add a new column to $\boldsymbol{W}$ and repeat the check.

We summarize this procedure in Algorithm 5.

To evaluate the complexity of the algorithm, we have to take into account the fact that the set of samples $\{\left(\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top\right)\boldsymbol{A}\boldsymbol{\omega}^{(l)}\}_{l=1}^{q}$ is updated recursively. Indeed, each step requires the computation of a single new sample from the range of $\boldsymbol{A}$, and its orthogonalization with respect to the range of $\boldsymbol{W}$ (see line 14), as well as the update of the right factor (see line 12). Both operations take $\mathcal{O}(mn)$ flops, and are repeated r

Algorithm 5: Randomized compression with fixed (spectral) precision with a randomized norm estimate

Data: matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\varepsilon > 0$, integer $q > 0$
Result: $\boldsymbol{W} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{Z} \in \mathbb{R}^{n\times r}$, with $\boldsymbol{W}^\top \boldsymbol{W} = \boldsymbol{I}_m$ and such that (3.8) holds with high probability for the spectral norm

Compute $\|\boldsymbol{A}\|$
for $l = 1, \dots, q$ **do**
  Generate Gaussian vector $\boldsymbol{\omega} \in \mathbb{R}^n$
  $\boldsymbol{y}^{(l)} := \boldsymbol{A}\boldsymbol{\omega}$
end
$\boldsymbol{W} := [\,],\ \boldsymbol{Z} := [\,]$
while $10\sqrt{2/\pi}\max_{l=1,\dots,q} \|\boldsymbol{y}^{(l)}\| \geqslant \varepsilon\|\boldsymbol{A}\|$ **do**
  $k :=$ random element of $\{1, \dots, q\}$
  $\boldsymbol{w} := \boldsymbol{y}^{(k)}/\|\boldsymbol{y}^{(k)}\|$
  $\boldsymbol{W} := [\boldsymbol{W} \,|\, \boldsymbol{w}]$
  $\boldsymbol{z} := \boldsymbol{A}^\top \boldsymbol{w}$
  $\boldsymbol{Z} := [\boldsymbol{Z} \,|\, \boldsymbol{z}]$
  Generate Gaussian vector $\boldsymbol{\omega} \in \mathbb{R}^n$
  $\boldsymbol{y}^{(k)} := \left(\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top\right)\boldsymbol{A}\boldsymbol{\omega}$
  for $l = 1, \dots, k-1, k+1, \dots, q$ **do**
    $\boldsymbol{y}^{(l)} := \left(\boldsymbol{I}_m - \boldsymbol{w}\boldsymbol{w}^\top\right)\boldsymbol{y}^{(l)}$
  end
end

times. If we include the cost of the initialization of the set of samples (which requires $\mathcal{O}(mn)$ flops per sample), the overall complexity of the algorithm is $\mathcal{O}(mn(r+q))$.

Also, as in the case of the Frobenius norm, it may be desirable to add a recompression step (which costs $\mathcal{O}\big(m(r+q)^2 + nr(r+q)\big)$ flops) at the end of the procedure.

3.3 Numerical examples

3.3.1 Computational environment

All numerical experiments described in this and in the upcoming chapters refer to implementations in MATLAB® 2016a, executed on the Malc1 cluster at EPFL (`malc1.epfl.ch`). In particular, we use 8 cores belonging to a single node with an Intel® Xeon® X5675E processor with frequency 3.07 GHz, and 191 GB of RAM.

Moreover, the MATLAB® implementations of the algorithms described in this thesis are available at `https://github.com/pradover/TT_HOSVD`.

3.3.2 Discretization of a rational function

Given the integer $n > 0$, we consider a uniform discretization of the interval $\left[\frac{1}{10}, \frac{n}{10}\right]$: $x_i = \frac{i}{10}$ for $i \in \{1, \dots, n\}$. For any fixed $\gamma \in \mathbb{R}^+$, we define the matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ entry-wise as

$$\boldsymbol{A}(i,j) = f_{2,\gamma}(x_i, x_j) = (x_i + x_j)^{-\gamma} \quad \text{for } (i,j) \in \{1, \dots, n\}^2$$

Due to the smoothness of $f_{2,\gamma}$ in $\left(0, \frac{n+1}{10}\right)^2$, the singular values of $\boldsymbol{A}$ decay with (at least) exponential rate (see e.g. [GL17]). For example, if $\gamma = 1$ and $n = 10^5$, the numerical rank is only 54 (i.e. there are exactly 54 singular values larger than the machine epsilon).

We consider the choices of γ and n specified above, and we apply Algorithm 1 to compress $\boldsymbol{A}$ down to rank $r \in \{10, 20, 30\}$. In particular, we wish to investigate how the solution depends on the oversampling parameter $p \in \{0, 1, \dots, 8\}$.

In Figure 3.1 we report the relative error (in the Frobenius norm) of each low-rank approximation. In particular, the plots show the average, minimum and maximum values of the error after $N = 20$ simulation with each configuration (r, p). It appears that small values of p (e.g. $p = 4$) are enough to achieve the optimal accuracy, in agreement with the numerical observations in [HMT11].

Moreover, we can verify that the theoretical bounds of Section 3.1.1 are satisfied. Indeed, the average error satisfies the bound (3.4) (see green dashed line), and (3.6) holds for all samples (see black dash-dotted line).

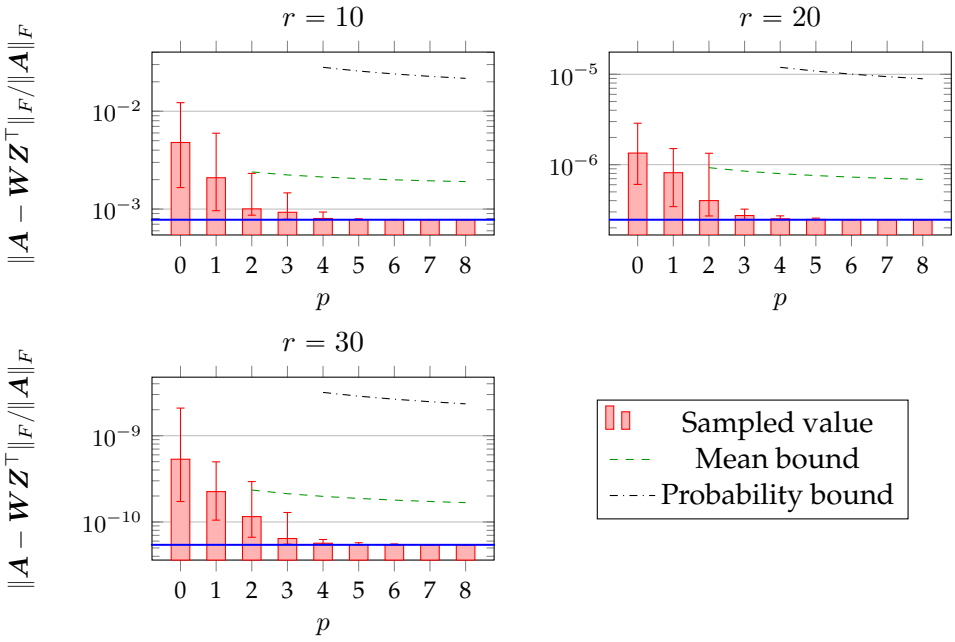

FIGURE 3.1: Average relative error versus oversampling for Algorithm 1. The error bars indicate the range of values obtained over $N = 20$ simulations. In blue the optimal error. In green and black the bounds from Theorems 3.1 and 3.2 respectively.

3.3.3 Compression of a Matérn kernel

Given the two positive parameters ν and ρ, we define the Matérn covariance function:

$$C_\nu^\rho(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{\rho} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu} r}{\rho} \right)$$

for $r \geqslant 0$, with K_ν being the modified Bessel function of the second kind. Such functions are employed in several machine learning techniques, among which we cite kernel regression (see e.g. [Bis09]) and Gaussian processes (see e.g. [RW05]).

Given the integer $n > 0$, we consider a uniform discretization of the interval $[\frac{1}{2n}, 1 - \frac{1}{2n}]$ ($x_i = \frac{2i-1}{2n}$, for $i \in \{1, \dots, n\}$), and we define the symmetric matrix

$\boldsymbol{A} \in \mathbb{R}^{n\times n}$ as

$$\boldsymbol{A}(i,j) = C_\nu^\rho(|x_i - x_j|) \quad \text{for } (i,j) \in \{1,\dots,n\}^2$$

We fix $\rho = 1$, $\nu = \frac{3}{2}$ and $n = 10^5$, so that the singular values of $\boldsymbol{A}$ have a slow polynomial decay ($\sigma_j \sim j^{-4.3}$ approximately). Our objective is to compress the matrix with fixed precision $\varepsilon \in \{10^{-2}, 10^{-6}\}$ in the Frobenius norm. To this aim we apply Algorithm 3 (including the recompression step) with $q \in \{1,2,3,5,7,10,15,20\}$, and observe the results of $N = 20$ trials with each configuration (ε, q).

The relative error of the approximation is shown in Figure 3.2. For $\varepsilon = 10^{-2}$ we can see that really small values of q (even $q = 1$) are sufficient to achieve the desired precision. However, for $\varepsilon = 10^{-6}$ the average error is smaller than ε only for $q \geqslant 15$.

This is due to the error introduced by the approximation (3.12). Indeed, since the singular values of $\boldsymbol{A}$ decay quite slowly, the norm of $\tilde{\boldsymbol{Z}}$ does not approximate the norm of the full matrix with enough accuracy for small values of q. Thus, Algorithm 3 is stopped before a sufficiently rich approximation of the range of $\boldsymbol{A}$ is identified.

In Figure 3.3 we report the rank of the approximation obtained, both before and after the recompression step. For both choices of ε, we can observe that the discrepancy between the ranks before and after recompression is approximately equal to q. This is due to the fact that (3.12) is consistent with the condition used in the recompression step, since both rely on $\tilde{\boldsymbol{Z}}$ to estimate the norm of the full matrix.

For $\varepsilon = 10^{-2}$, the final ranks always equal the optimal one, attesting that the approximate condition (3.12) is reliable in stopping the algorithm. This is not true for $\varepsilon = 10^{-6}$, where the final ranks are below the optimal one for small values of q.

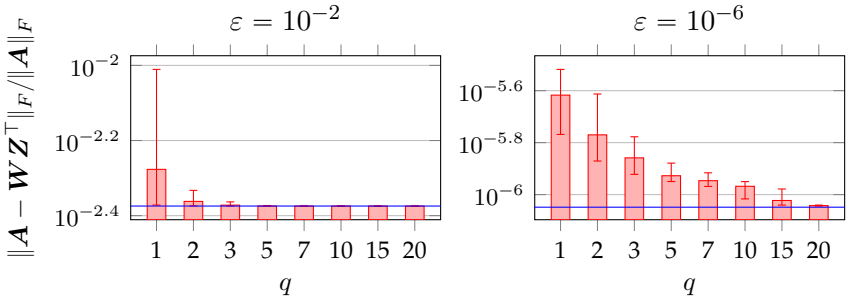

FIGURE 3.2: Average relative error versus sampling parameter for Algorithm 3. The error bars indicate the range of values obtained over $N = 20$ simulations. In blue the optimal error.

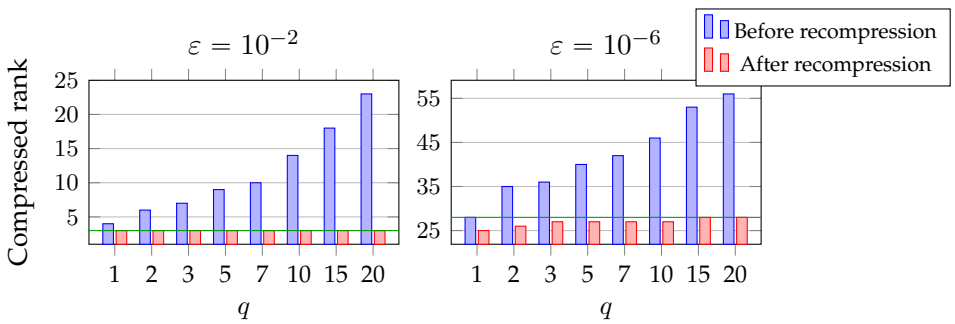

FIGURE 3.3: Modal values of the compressed rank (before and after recompression) versus sampling parameter for Algorithm 3. In green the optimal rank.

3.3.4 Compression with non-Gaussian random vectors

We consider the matrix $\boldsymbol{A}$ defined in Section 3.3.3. Our aim is to compress it to ranks $r \in \{5, 15, 30\}$ using modified versions of Algorithm 1, where the random matrix $\boldsymbol{\Omega}$ is sampled according to different distributions. In particular, we consider sample vectors with independent entries drawn from the uniform distribution over $(0, 1)$, and "rank-1" Gaussian sample vectors, i.e. vectors of the form $\boldsymbol{\omega} = \boldsymbol{\omega}_1 \otimes \boldsymbol{\omega}_2$, with $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ being $\sqrt{n}$-dimensional (independent) Gaussian vectors.

For these distributions there are no theoretical results similar to the ones presented in Section 3.1.1. However, they have advantages from the computational point of view: both are (on most calculators) cheaper to sample than the Gaussian distribution. Moreover, in the "rank-1" case, the structured format of the vector can be exploited to gain efficiency (see e.g. [ZJD15], [KP16], and Sections 4.3 and 5.2 of this thesis).

In Figure 3.4 we compare the relative errors with the ones achieved in the Gaussian case. For all approaches, we choose $p \in \{1, 2, 3, 5, 7, 10, 15, 20\}$ as the sampling parameter.

We can observe that the results obtained with the three distributions are similar: all of them appear to converge at the same rate, and the ranges of the values are of similar size. In particular, all appear to satisfy (3.4) and (3.6).

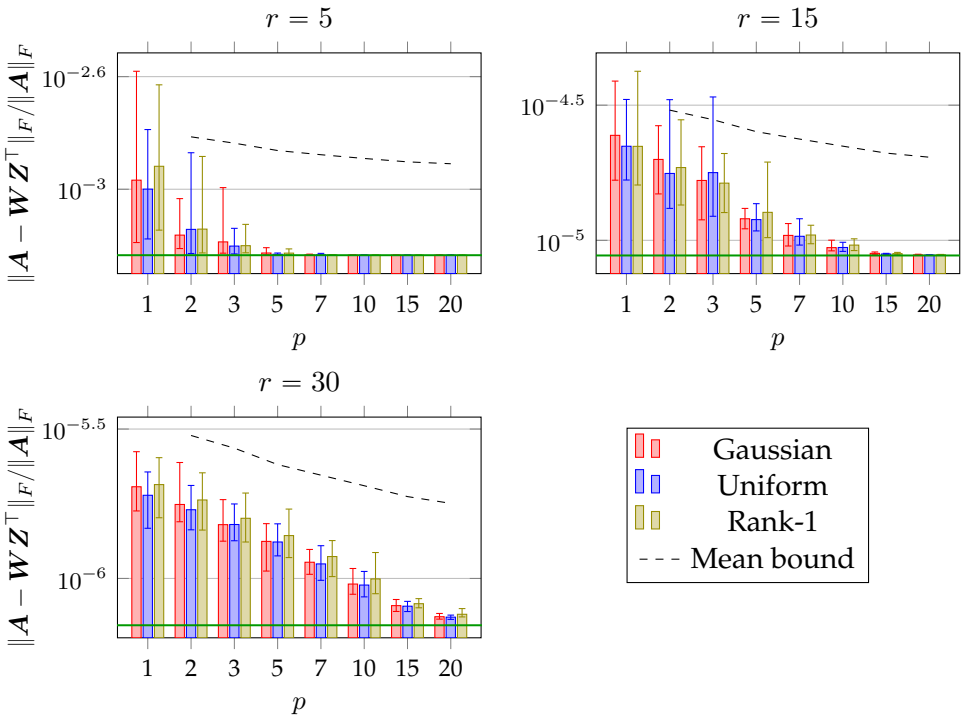

FIGURE 3.4: Average relative error versus oversampling for Algorithm 1. The bars connect the minimum and the maximum value of the relative error obtained over $N = 20$ simulations. In green the optimal error. In black the bound from Theorem 3.1. The bound from Theorem 3.2 is not shown for ease of visualization, since it is several orders of magnitude larger than the sampled values.

Chapter 4

Compression of tensors in the TT format

The Tensor Train (TT) format was introduced by Oseledets and Tyrtyshnikov (see [OT09]) as a way to break the curse of dimensionality. Indeed, most operations in the TT format have polynomial complexity in the order of the tensor. The format had already been used for a few decades in several physics applications (see e.g. [Aff+87] and [FNW92]) under the name Matrix Product States (MPS).

4.1 Overview of the Tensor Train format

We say that the d-dimensional tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ admits a TT representation with TT-ranks $(r_0 = 1, r_1, \ldots, r_{d-1}, r_d = 1)$ if there exist d tensors of order 3

$$\boldsymbol{\mathcal{A}}_j \in \mathbb{R}^{r_{j-1} \times n_j \times r_j} \quad \text{for } j \in \{1, \ldots, d\}$$

(which are commonly called *cores* of the representation), such that

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) = \sum_{\alpha_1, \ldots, \alpha_{d-1}} \boldsymbol{\mathcal{A}}_1(1, i_1, \alpha_1) \boldsymbol{\mathcal{A}}_2(\alpha_1, i_2, \alpha_2) \ldots \boldsymbol{\mathcal{A}}_d(\alpha_{d-1}, i_d, 1) \tag{4.1}$$

for $(i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d$. In the expression above, the sum over α_j is carried out from 1 to r_j (for $j \in \{1, \ldots, d-1\}$), and we define

$$I_j = \{1, \ldots, n_j\} \quad \text{for } j \in \{1, \ldots, d\}$$

We will use the following notation to represent the mode-2 slices of each core

$$\boldsymbol{A}_j(i_j) = \boldsymbol{\mathcal{A}}_j(:, i_j, :) \in \mathbb{R}^{r_{j-1} \times r_j} \quad \text{for } i_j \in I_j, \text{ for } j \in \{1, \ldots, d\},$$

so that we can interpret (4.1) as a chain of matrix products:

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) = \boldsymbol{A}_1(i_1) \boldsymbol{A}_2(i_2) \ldots \boldsymbol{A}_d(i_d) \quad \text{for } (i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d \tag{4.2}$$

Another useful (graphic) representation of tensors in the TT format is given by *tensor network diagrams* (see e.g. [Oru14] and [Ste16]). An example of such a representation is given in Figure 4.1: each node corresponds to a core tensor, and each edge to an index over which a sum is computed.

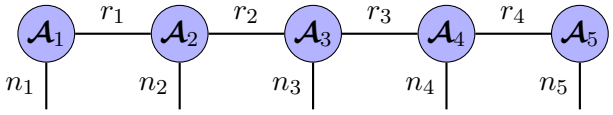

FIGURE 4.1: Tensor network diagram for a 5-dimensional tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \ldots \times n_5}$ in the TT format (4.2) with TT-ranks $(1, r_1, r_2, r_3, r_4, 1)$.

For fixed TT-ranks, the TT representation of a tensor is not unique, similarly to the non-uniqueness of the dyadic representation with fixed rank (1.1) of a matrix. However, the following result guarantees the existence of a TT decomposition and an upper-bound on the minimal TT-ranks. Moreover, it provides an algorithm to convert any tensor into the TT format.

Theorem 4.1. *Let $\mathcal{A} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$. There exists a decomposition of the form (4.2) with $r_0 = r_d = 1$ and*

$$r_j = \operatorname{rank} \mathcal{A}^{<j>} \quad \text{for } j \in \{1, \ldots, d-1\} \tag{4.3}$$

Proof. We follow the idea of the proof of Theorem 2.1 in [Ose11].

Define $(r_1, \ldots, r_{d-1})$ as in (4.3). Consider a rank-r_1 dyadic decomposition of $\mathcal{A}^{<1>}$:

$$\mathcal{A}^{<1>} = \boldsymbol{W}_1 \boldsymbol{Z}_1^\top \tag{4.4}$$

with

$$\boldsymbol{W}_1 \in \mathbb{R}^{n_1 \times r_1} \quad \text{and} \quad \boldsymbol{Z}_1 \in \mathbb{R}^{(n_2 \cdots n_d) \times r_1}$$

We define the first core $\mathcal{A}_1 \in \mathbb{R}^{1 \times n_1 \times r_1}$ so that $(\mathcal{A}_1)^{<2>} = \boldsymbol{W}_1$.

Let $\mathcal{B}_1$ be the d-dimensional tensor of size $r_1 \times n_2 \times n_3 \times \ldots \times n_d$ such that

$$(\mathcal{B}_1)^{<1>} = \boldsymbol{Z}_1^\top \tag{4.5}$$

and assume $\widetilde{r}_2 = \operatorname{rank} (\mathcal{B}_1)^{<2>}$.

Consider a rank-$\widetilde{r}_2$ dyadic decomposition of $(\mathcal{B}_1)^{<2>}$:

$$(\mathcal{B}_1)^{<2>} = \boldsymbol{W}_2 \boldsymbol{Z}_2^\top \tag{4.6}$$

with

$$\boldsymbol{W}_2 \in \mathbb{R}^{(r_1 n_2) \times \widetilde{r}_2} \quad \text{and} \quad \boldsymbol{Z}_2 \in \mathbb{R}^{(n_3 \cdots n_d) \times \widetilde{r}_2}$$

We define the second core $\mathcal{A}_2 \in \mathbb{R}^{r_1 \times n_2 \times \widetilde{r}_1}$ so that $(\mathcal{A}_2)^{<2>} = \boldsymbol{W}_2$. Then we proceed recursively to compute all the cores of the TT representation.

At the end of this left-to-right sweep, the TT-ranks of the decomposition are $(1, r_1, \widetilde{r}_2, \ldots, \widetilde{r}_{d-1}, 1)$, where $\widetilde{r}_j = \operatorname{rank} (\mathcal{B}_{j-1})^{<2>}$ for $j \in \{2, \ldots, d-1\}$. Hence, it suffices to prove that

$$\operatorname{rank} (\mathcal{B}_{j-1})^{<2>} = \widetilde{r}_j \leqslant r_j = \operatorname{rank} \mathcal{A}^{<j>} \quad \text{for } j \in \{2, \ldots, d-1\} \tag{4.7}$$

To this aim, we prove that

$$\operatorname{rank} (\mathcal{B}_1)^{<l>} \leqslant \operatorname{rank} \mathcal{A}^{<l>} \quad \text{for } l \in \{2, \ldots, d-1\} \tag{4.8}$$

Proof of claim (4.8): From (4.4) we deduce that

$$\boldsymbol{Z}_1^\top = \left(\boldsymbol{W}_1^\top \boldsymbol{W}_1\right)^{-1} \boldsymbol{W}_1^\top \boldsymbol{A}^{<1>} =: \widetilde{\boldsymbol{W}}_1^\top \boldsymbol{\mathcal{A}}^{<1>}$$

or equivalently

$$\boldsymbol{Z}_1^\top = \left(\boldsymbol{\mathcal{A}} \times_1 \widetilde{\boldsymbol{W}}_1^\top\right)^{<1>}$$

Then (4.5) implies that $\boldsymbol{\mathcal{B}}_1 = \boldsymbol{\mathcal{A}} \times_1 \widetilde{\boldsymbol{W}}_1^\top$.

Now, let $l \in \{2, \ldots, d-1\}$ be fixed and consider the rank-r_l dyadic decomposition of $\boldsymbol{\mathcal{A}}^{<l>}$:

$$\boldsymbol{\mathcal{A}}^{<l>} = \boldsymbol{W}_l' \, \boldsymbol{Z}_l'^\top$$

It is easy to verify that the following rank-r_l decomposition holds for $(\boldsymbol{\mathcal{B}}_1)^{<l>}$:

$$(\boldsymbol{\mathcal{B}}_1)^{<l>} = \left(\boldsymbol{I}_{n_l} \otimes \ldots \otimes \boldsymbol{I}_{n_2} \otimes \widetilde{\boldsymbol{W}}_1^\top\right) \boldsymbol{W}_l' \, \boldsymbol{Z}_l'^\top = \boldsymbol{W}_l'' \, \boldsymbol{Z}_l'^\top$$

This implies that rank $(\boldsymbol{\mathcal{B}}_1)^{<l>} \leqslant r_l$ and proves the claim. □

Using a similar argument (by replacing $\boldsymbol{\mathcal{A}}$ with $\boldsymbol{\mathcal{B}}_{j-1}$ and $\boldsymbol{\mathcal{B}}_1$ with $\boldsymbol{\mathcal{B}}_j$), we can verify that

$$\text{rank } (\boldsymbol{\mathcal{B}}_j)^{<l>} \leqslant \text{rank } (\boldsymbol{\mathcal{B}}_{j-1})^{<l+1>} \quad \text{for } l \in \{2, \ldots, d-j\}$$

for $j \in \{2, \ldots, d-2\}$. The claim (4.7) follows immediately, since

$$\text{rank } (\boldsymbol{\mathcal{B}}_{j-1})^{<2>} \leqslant \text{rank } (\boldsymbol{\mathcal{B}}_{j-2})^{<3>} \leqslant \ldots \leqslant \text{rank } (\boldsymbol{\mathcal{B}}_1)^{<j>} \leqslant \text{rank } \boldsymbol{\mathcal{A}}^{<j>}$$

for $j \in \{2, \ldots, d-1\}$.

At this point, we have obtained a TT decomposition whose j^{th} TT-rank is at most r_j (for $j \in \{1, \ldots, d-1\}$). This proves the theorem, since, in order to obtain a TT decomposition with TT-ranks $(1, r_1, \ldots, r_{d-1}, 1)$, it suffices to add a zero-padding to all the cores with deficient size. □

The proof of the previous Theorem is constructive. As such, it provides an algorithm to convert any tensor into the TT format. The complexity of the algorithm is dominated by the cost of the computation of the d dyadic decompositions of the form (4.4) and (4.6), which can be carried out using a RRQR decomposition: if we assume for simplicity that

$$n_1 = \ldots = n_d = n \quad \text{and} \quad r_1 = \ldots = r_{d-1} = r,$$

the cost for computing $\boldsymbol{\mathcal{A}}^{<1>}$ is $\mathcal{O}\left(n^d r\right)$ flops, while for $(\boldsymbol{\mathcal{B}}_j)^{<2>}$ (with $j \in \{1, \ldots, d-2\}$) it is $\mathcal{O}\left(n^{d-j} r^2\right)$ flops, leading to an overall complexity of $\mathcal{O}\left(n^d r^2\right)$.

The complexity of the conversion to the TT format with respect to d can be reduced to polynomial time if the initial tensor $\boldsymbol{\mathcal{A}}$ is stored in some structured format. For instance, the cost is linear in d if $\boldsymbol{\mathcal{A}}$ is stored using the canonical decomposition (see

e.g. [Ose11]):

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) = \sum_{\alpha=1}^{\bar{r}} \boldsymbol{U}_1(i_1, \alpha) \ldots \boldsymbol{U}_d(i_d, \alpha) \quad \text{for } (i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d \tag{4.9}$$

with $\bar{r}$ being the canonical rank of $\boldsymbol{\mathcal{A}}$, and $\boldsymbol{U}_j \in \mathbb{R}^{n_j \times \bar{r}}$ (for $j \in \{1, \ldots, d\}$) being the canonical factors.

It is easy to compute a TT-decomposition of $\boldsymbol{\mathcal{A}}$ with TT-ranks $(1, \bar{r}, \ldots, \bar{r}, 1)$ by introducing $d-2$ additional identity matrices of size $\bar{r} \times \bar{r}$ into (4.9): for any $(i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d$, it holds

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) = \sum_{\alpha_1, \ldots, \alpha_{d-1}} \boldsymbol{U}_1(i_1, \alpha_1) \boldsymbol{I}_{\bar{r}}(\alpha_1, \alpha_2) \ldots \boldsymbol{I}_{\bar{r}}(\alpha_{d-2}, \alpha_{d-1}) \boldsymbol{U}_{d-1}(i_{d-1}, \alpha_{d-1}) \boldsymbol{U}_d(i_d, \alpha_{d-1})$$

where the sum ranges from 1 to $\bar{r}$ for each α_j (for $j \in \{1, \ldots, d-1\}$).

Now we can define the inner cores of the TT representation:

$$\boldsymbol{\mathcal{A}}_j(\alpha_{j-1}, i_j, \alpha_j) = \boldsymbol{I}_{\bar{r}}(\alpha_{j-1}, \alpha_j) \boldsymbol{U}_j(i_j, \alpha_j) \quad \text{for } (i_j, \alpha_{j-1}, \alpha_j) \in I_j \times \{1, \ldots, \bar{r}\}^2$$

for $j \in \{2, \ldots, d-1\}$, and the outer cores accordingly:

$$\boldsymbol{\mathcal{A}}_1(1, i_1, \alpha_1) = \boldsymbol{U}_1(i_1, \alpha_1) \quad \text{for } (i_1, \alpha_1) \in I_1 \times \{1, \ldots, \bar{r}\}$$

$$\boldsymbol{\mathcal{A}}_d(\alpha_{d-1}, i_d, 1) = \boldsymbol{U}_d(i_d, \alpha_{d-1}) \quad \text{for } (i_d, \alpha_{d-1}) \in I_d \times \{1, \ldots, \bar{r}\}$$

4.2 Compression of tensors in the TT format using deterministic methods

Now we focus on the following problem: let $\boldsymbol{\mathcal{A}}$ be a tensor in the TT format (4.2) with TT-ranks $(1, r_1, \ldots, r_{d-1}, 1)$. We want to find a new (possibly approximate) TT decomposition of $\boldsymbol{\mathcal{A}}$ with cores $\boldsymbol{\mathcal{A}}_j^*$ whose TT-ranks $(1, r_1^*, \ldots, r_{d-1}^*, 1)$ are smaller.

Two interesting formulations of this problem can be stated in same spirit as in Chapter 3:

- compress the tensor down to some arbitrary TT-ranks;
- compress the tensor so that the relative Frobenius error between the original tensor and the new one $\boldsymbol{\mathcal{A}}^*$, i.e.

$$\frac{\|\boldsymbol{\mathcal{A}}^* - \boldsymbol{\mathcal{A}}\|_F}{\|\boldsymbol{\mathcal{A}}\|_F}, \tag{4.10}$$

  is smaller than some prescribed accuracy.

Using the terminology from [Ose11], we will refer to the two cases as *truncation* and *rounding* problems respectively.

To devise an efficient algorithm to approach these problems, it is useful to define the concept of left- and right-orthogonality for a tensor in the TT format.

4.2.1 Orthogonalization of tensors in the TT format

Being the natural generalization of the QR decomposition for tensors in the TT format, orthogonalization inherits a crucial role in the manipulation of tensors.

For ease of notation, we first introduce the concept of interface matrices for tensors in the TT format.

Definition 4.1 (Interface matrices). *Let $\boldsymbol{\mathcal{A}}$ be a d-dimensional tensor in the TT format (4.2) with TT-ranks $(1, r_1, \ldots, r_{d-1}, 1)$ and let $j \in \{1, \ldots, d-1\}$. We consider the $(j+1)$-dimensional tensor $\boldsymbol{\mathcal{B}}$ of size $n_1 \times \ldots \times n_j \times r_j$ with entries*

$$\boldsymbol{\mathcal{B}}(i_1, \ldots, i_j, :) = \boldsymbol{A}_1(i_1) \ldots \boldsymbol{A}_j(i_j) \quad \text{for } (i_1, \ldots, i_j) \in I_1 \times \ldots \times I_j$$

and we define the left interface matrix as

$$\boldsymbol{\mathcal{A}}_{\leqslant j} = \boldsymbol{\mathcal{B}}^{<j>} \in \mathbb{R}^{(n_1 \cdots n_j) \times r_j} \tag{4.11}$$

Similarly, we consider the $(d-j+1)$-dimensional tensor $\boldsymbol{\mathcal{C}}$ of size $r_j \times n_{j+1} \times \ldots \times n_d$ with entries

$$\boldsymbol{\mathcal{C}}(:, i_{j+1}, \ldots, i_d) = \boldsymbol{A}_{j+1}(i_{j+1}) \ldots \boldsymbol{A}_d(i_d) \quad \text{for } (i_{j+1}, \ldots, i_d) \in I_{j+1} \times \ldots \times I_d$$

and we define the right interface matrix as

$$\boldsymbol{\mathcal{A}}_{\geqslant j+1} = \left(\boldsymbol{\mathcal{C}}^{<1>}\right)^\top \in \mathbb{R}^{(n_{j+1} \cdots n_d) \times r_j} \tag{4.12}$$

Since the TT factorization of a tensor is not unique, it is important to observe that interface matrices depend on the particular TT decomposition and not just on the tensor itself.

The importance of interface matrices is due to the fact that they appear in the dyadic decompositions of tensor unfoldings: given $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format (4.2), for $j \in \{1, \ldots, d-1\}$, it holds

$$\boldsymbol{\mathcal{A}}^{<j>} = \boldsymbol{\mathcal{A}}_{\leqslant j} \boldsymbol{\mathcal{A}}_{\geqslant j+1}^\top$$

Given this result, we define the orthogonality of a tensor in the TT format in terms of the orthogonality of its interface matrices.

Definition 4.2 (Left- and right-orthogonality). *Let $\boldsymbol{\mathcal{A}}$ be a d-dimensional tensor in the TT format (4.2) with TT-ranks $(1, r_1, \ldots, r_{d-1}, 1)$. We say that $\boldsymbol{\mathcal{A}}$ is left-orthogonal if*

$$\boldsymbol{\mathcal{A}}_{\leqslant j}^\top \boldsymbol{\mathcal{A}}_{\leqslant j} = \boldsymbol{I}_{r_j} \tag{4.13}$$

for $j \in \{1, \ldots, d-1\}$, and that it is right-orthogonal if

$$\boldsymbol{\mathcal{A}}_{\geqslant j+1}^\top \boldsymbol{\mathcal{A}}_{\geqslant j+1} = \boldsymbol{I}_{r_j} \tag{4.14}$$

for $j \in \{1, \ldots, d-1\}$.

There is an alternative way to characterize the orthogonality of a tensor, as stated by the following Lemma.

Lemma 4.2. *Let $\boldsymbol{\mathcal{A}}$ be as in Definition 4.2. $\boldsymbol{\mathcal{A}}$ is left-orthogonal if and only if*

$$\left(\boldsymbol{\mathcal{A}}_j\right)^{<2>\top}\left(\boldsymbol{\mathcal{A}}_j\right)^{<2>} = \boldsymbol{I}_{r_j} \tag{4.15}$$

for $j \in \{1, \ldots, d-1\}$, and right-orthogonal if and only if

$$\left(\boldsymbol{\mathcal{A}}_{j+1}\right)^{<1>}\left(\boldsymbol{\mathcal{A}}_{j+1}\right)^{<1>\top} = \boldsymbol{I}_{r_j} \tag{4.16}$$

for $j \in \{1, \ldots, d-1\}$.

Proof. We only provide the proof for the left-orthogonality. The other case can be treated in an analogous fashion.

For the first core we have

$$\boldsymbol{\mathcal{A}}_{\leqslant 1} = \left(\boldsymbol{\mathcal{A}}_1\right)^{<2>}$$

Thus, (4.13) and (4.15) are equivalent for $j = 1$.

We proceed by induction. Let $l \in \{1, \ldots, d-2\}$, and assume that (4.13) and (4.15) are equivalent for $j \in \{1, \ldots, l\}$. We have that

$$\boldsymbol{\mathcal{A}}_{\leqslant l+1} = \left(\boldsymbol{I}_{n_{l+1}} \otimes \boldsymbol{\mathcal{A}}_{\leqslant l}\right)\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>}$$

and we can apply Proposition 2.2 to obtain

$$\begin{aligned}
\boldsymbol{\mathcal{A}}_{\leqslant l+1}^{\top}\boldsymbol{\mathcal{A}}_{\leqslant l+1} &= \left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>\top}\left(\boldsymbol{I}_{n_{l+1}} \otimes \boldsymbol{\mathcal{A}}_{\leqslant l}\right)^{\top}\left(\boldsymbol{I}_{n_{l+1}} \otimes \boldsymbol{\mathcal{A}}_{\leqslant l}\right)\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>} \\
&= \left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>\top}\left(\boldsymbol{I}_{n_{l+1}} \otimes \left(\boldsymbol{\mathcal{A}}_{\leqslant l}^{\top}\boldsymbol{\mathcal{A}}_{\leqslant l}\right)\right)\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>}
\end{aligned}$$

Assume that either (4.13) or (4.15) is satisfied for all values of $j \in \{1, \ldots, d-1\}$. Then, in particular, we have that $\boldsymbol{\mathcal{A}}_{\leqslant l}^{\top}\boldsymbol{\mathcal{A}}_{\leqslant l} = \boldsymbol{I}_{r_l}$ due to the induction hypothesis. As such, it holds that

$$\begin{aligned}
\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>\top}\left(\boldsymbol{I}_{n_{l+1}} \otimes \left(\boldsymbol{\mathcal{A}}_{\leqslant l}^{\top}\boldsymbol{\mathcal{A}}_{\leqslant l}\right)\right)\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>} &= \left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>\top}\left(\boldsymbol{I}_{n_{l+1}} \otimes \boldsymbol{I}_{r_l}\right)\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>} \\
&= \left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>\top}\boldsymbol{I}_{n_{l+1}r_l}\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>} \\
&= \left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>\top}\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>}
\end{aligned}$$

which implies

$$\boldsymbol{\mathcal{A}}_{\leqslant l+1}^{\top}\boldsymbol{\mathcal{A}}_{\leqslant l+1} = \left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>\top}\left(\boldsymbol{\mathcal{A}}_{l+1}\right)^{<2>}$$

□

Using this easier definition, we can efficiently orthogonalize a tensor $\boldsymbol{\mathcal{A}}$ in the TT format (either to the left or to the right) by following the intuition used in the proof of Theorem 4.1 (see e.g. [Ose11]).

We proceed as follows for the left-orthogonalization. We consider the QR decomposition of the second unfolding of the first core

$$\left(\boldsymbol{\mathcal{A}}_1\right)^{<2>} = \boldsymbol{Q}_1\boldsymbol{R}_1$$

We overwrite the first core so that $\left(\boldsymbol{\mathcal{A}}_1^*\right)^{<2>} = \boldsymbol{Q}_1$, and we use $\boldsymbol{R}_1$ to update the second core:

$$\boldsymbol{\mathcal{A}}_2^* = \boldsymbol{\mathcal{A}}_2 \times_1 \boldsymbol{R}_1$$

In this way, (4.15) is satisfied for $j = 1$, and the overall tensor has not been affected, since

$$\boldsymbol{A}_1(i_1)\boldsymbol{A}_2(i_2) = \boldsymbol{Q}_1(i_1,:)\boldsymbol{R}_1\boldsymbol{A}_2(i_2) = \boldsymbol{A}_1^*(i_1)\boldsymbol{A}_2^*(i_2) \quad \text{for } (i_1,i_2) \in I_1 \times I_2$$

Now we consider $(\boldsymbol{\mathcal{A}}_2^*)^{<2>}$ and proceed similarly, by computing its QR decomposition $(\boldsymbol{\mathcal{A}}_2^*)^{<2>} = \boldsymbol{Q}_2\boldsymbol{R}_2$, overwriting the second core with $\boldsymbol{Q}_2$ (so that (4.15) is satisfied for $j = 2$), and updating the third core using $\boldsymbol{R}_2$. Then we iterate over all the cores.

The resulting procedure is summarized in Algorithm 6. Assuming that

$$n_1 = \ldots = n_d = n \quad \text{and} \quad r_1 = \ldots = r_{d-1} = r,$$

the complexity of the algorithm is $\mathcal{O}\left(dnr^3\right)$ flops, since each QR decomposition, as well as each 1^{st} mode (or 3^{rd} mode) product, costs $\mathcal{O}\left(nr^3\right)$ flops.

We also provide a graphical depiction of the algorithm in Figure 4.2. In particular, we show the left-orthogonalization of a $n_1 \times \ldots \times n_4$ tensor with TT-ranks $(1, r_1, r_2, r_3, 1)$. Each pink rectangle contains the two factors appearing in the QR decomposition of the gray block in the line above, after a suitable reshaping. It is interesting to observe that each of the matrices $\boldsymbol{R}_j$ (with $j \in \{1, \ldots, d-1\}$) represents an interface between a left-orthogonal and a non-orthogonal part, comprised respectively of green and blue nodes.

4.2.2 Truncation of tensors in the TT format in two sweeps

To solve the truncation problem, we can exploit the orthogonalization algorithm as follows. We right-orthogonalize the original tensor $\boldsymbol{\mathcal{A}}$, leading to the TT decomposition with cores $\boldsymbol{\mathcal{A}}'_1, \ldots, \boldsymbol{\mathcal{A}}'_d$.

We can express the first unfolding of $\boldsymbol{\mathcal{A}}$ using the newly obtained cores

$$\boldsymbol{\mathcal{A}}^{<1>} = \boldsymbol{\mathcal{A}}'_{\leqslant 1}\, {\boldsymbol{\mathcal{A}}'_{\geqslant 2}}^{\top} = \left(\boldsymbol{\mathcal{A}}'_1\right)^{<2>} {\boldsymbol{\mathcal{A}}'_{\geqslant 2}}^{\top}$$

and compute the SVD of the left interface matrix

$$\left(\boldsymbol{\mathcal{A}}'_1\right)^{<2>} = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1^{\top}$$

Since $\boldsymbol{\mathcal{A}}$ has been right-orthogonalized, the right interface matrix $\boldsymbol{\mathcal{A}}'_{\geqslant 2}$ has orthonormal columns. Hence the SVD of the first unfolding is of the form

$$\boldsymbol{\mathcal{A}}^{<1>} = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\left(\boldsymbol{\mathcal{A}}'_{\geqslant 2}\boldsymbol{V}_1\right)^{\top}$$

and it is possible to obtain the best rank-r_1^* approximation of $\boldsymbol{\mathcal{A}}^{<1>}$ by truncating each of the factors of the SVD (see Chapter 3)

$$\boldsymbol{W}_1 = \boldsymbol{U}_1(:, 1:r_1^*) \quad \text{and} \quad \boldsymbol{Z}_1 = \boldsymbol{V}_1(:, 1:r_1^*)\boldsymbol{\Sigma}_1(1:r_1^*, 1:r_1^*),$$

so that

$$(\boldsymbol{\mathcal{A}}_1)^{<2>} \approx \boldsymbol{W}_1\boldsymbol{Z}_1^{\top} \quad \text{and} \quad \boldsymbol{\mathcal{A}}^{<1>} \approx \boldsymbol{W}_1\left(\boldsymbol{\mathcal{A}}'_{\geqslant 2}\boldsymbol{Z}_1\right)^{\top}$$

Algorithm 6: Left- or right-orthogonalization of a tensor in the TT format

Data: tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_d$, direction
Result: $\boldsymbol{\mathcal{A}}^* \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}^*_1, \ldots, \boldsymbol{\mathcal{A}}^*_d$, orthogonal in the specified direction

if *direction=='left'* **then**
  $\boldsymbol{\mathcal{A}}^*_1 := \boldsymbol{\mathcal{A}}_1$
  for $j = 1, 2, \ldots, d-1$ **do**
    Compute QR decomposition: $\left(\boldsymbol{\mathcal{A}}^*_j\right)^{<2>} =: \boldsymbol{Q}_j \boldsymbol{R}_j$
    $\boldsymbol{\mathcal{A}}^*_j := \texttt{reshape}(\boldsymbol{Q}_j, [r_{j-1}, n_j, r_j])$
    $\boldsymbol{\mathcal{A}}^*_{j+1} := \boldsymbol{\mathcal{A}}_{j+1} \times_1 \boldsymbol{R}_j$
  end
else
  $\boldsymbol{\mathcal{A}}^*_d := \boldsymbol{\mathcal{A}}_d$
  for $j = d, d-1, \ldots, 2$ **do**
    Compute QR decomposition: $\left(\boldsymbol{\mathcal{A}}^*_j\right)^{<1>^\top} =: \boldsymbol{Q}_j \boldsymbol{R}_j$
    $\boldsymbol{\mathcal{A}}^*_j := \texttt{reshape}(\boldsymbol{Q}^\top_j, [r_{j-1}, n_j, r_j])$
    $\boldsymbol{\mathcal{A}}^*_{j-1} := \boldsymbol{\mathcal{A}}_{j-1} \times_3 \boldsymbol{R}_j$
  end
end

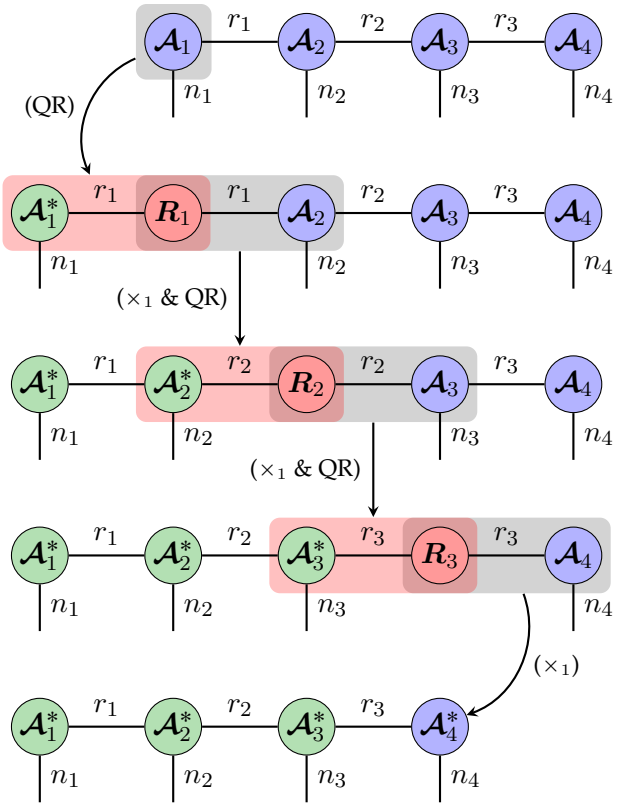

FIGURE 4.2: Tensor network diagram for the left-orthogonalization of $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_4}$, according to Algorithm 6. Green nodes have second unfoldings with orthonormal columns.

Just replacing the first core with its compressed version would not reduce the TT-ranks. Instead, we define the first core of $\boldsymbol{\mathcal{A}}^*$ so that $(\boldsymbol{\mathcal{A}}_1^*)^{<2>} = \boldsymbol{W}_1$, and we compute a partial update of the second core:

$$\boldsymbol{\mathcal{A}}_2^{**} = \boldsymbol{\mathcal{A}}_2' \times_1 \boldsymbol{Z}_1^\top$$

After these operations we have

$$\boldsymbol{A}_1'(i_1)\boldsymbol{A}_2'(i_2) \approx \boldsymbol{W}_1(i_1,:)\boldsymbol{Z}_1^\top \boldsymbol{A}_2'(i_2) = \boldsymbol{\mathcal{A}}_1^*(i_1)\boldsymbol{\mathcal{A}}_2^{**}(i_2) \quad \text{for } (i_1,i_2) \in I_1 \times I_2$$

with $\boldsymbol{\mathcal{A}}_1^* \in \mathbb{R}^{1\times n_1 \times r_1^*}$ and $\boldsymbol{\mathcal{A}}_2^{**} \in \mathbb{R}^{r_1^* \times n_2 \times r_2}$.

Now we consider the second unfolding of $\boldsymbol{\mathcal{A}}$

$$\begin{aligned}
\boldsymbol{\mathcal{A}}^{<2>} = \boldsymbol{\mathcal{A}}_{\leqslant 2}' \, \boldsymbol{\mathcal{A}}_{\geqslant 3}'^{\top} &= \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{\mathcal{A}}_{\leqslant 1}'\right) \left(\boldsymbol{\mathcal{A}}_2'\right)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}'^{\top} \\
&\approx \left(\boldsymbol{I}_{n_2} \otimes \left(\boldsymbol{W}_1 \boldsymbol{Z}_1^\top\right)\right) \left(\boldsymbol{\mathcal{A}}_2'\right)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}'^{\top} \\
&= \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1\right) \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{Z}_1^\top\right) \left(\boldsymbol{\mathcal{A}}_2'\right)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}'^{\top} \\
&= \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1\right) \left(\boldsymbol{\mathcal{A}}_2' \times_1 \boldsymbol{Z}_1^\top\right)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}'^{\top} \\
&= \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1\right) \left(\boldsymbol{\mathcal{A}}_2^{**}\right)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}'^{\top}
\end{aligned}$$

and we compute the SVD of $(\boldsymbol{\mathcal{A}}_2^{**})^{<2>}$: $(\boldsymbol{\mathcal{A}}_2^{**})^{<2>} = \boldsymbol{U}_2 \boldsymbol{\Sigma}_2 \boldsymbol{V}_2^\top$.

We observe that

$$\left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1\right)^\top \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1\right) = \boldsymbol{I}_{n_2} \otimes \left(\boldsymbol{W}_1^\top \boldsymbol{W}_1\right) = \boldsymbol{I}_{n_2} \otimes \boldsymbol{I}_{r_1^* n_2} = \boldsymbol{I}_{r_1^* n_2},$$

and that the right interface matrix $\boldsymbol{\mathcal{A}}_{\geqslant 3}'$ has orthonormal columns. Hence, the SVD of the second unfolding is of the form

$$\boldsymbol{\mathcal{A}}^{<2>} = \left(\left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1\right) \boldsymbol{U}_2\right) \boldsymbol{\Sigma}_2 \left(\boldsymbol{\mathcal{A}}_{\geqslant 3}' \boldsymbol{V}_2\right)^\top$$

Then, if we define

$$\boldsymbol{W}_2 = \boldsymbol{U}_2(:, 1:r_2^*) \quad \text{and} \quad \boldsymbol{Z}_2 = \boldsymbol{V}_2(:, 1:r_2^*)\boldsymbol{\Sigma}_2(1:r_2^*, 1:r_2^*),$$

we can compute the second core of $\boldsymbol{\mathcal{A}}^*$ so that $(\boldsymbol{\mathcal{A}}_2^*)^{<2>} = \boldsymbol{W}_2$, and update the third core:

$$\boldsymbol{\mathcal{A}}_3^{**} = \boldsymbol{\mathcal{A}}_3' \times_1 \boldsymbol{Z}_2^\top$$

Now we iterate over all the cores, leading to the procedure summarized in Algorithm 7. Assuming for simplicity that

$$n_1 = \ldots = n_d = n, \quad r_1 = \ldots = r_{d-1} = r \quad \text{and} \quad r_1^* = \ldots = r_{d-1}^* = r^*,$$

the cost of each SVD is $\mathcal{O}\left(nr^3\right)$, while the cost of each 1^{st} (or 3^{rd}) mode product is $\mathcal{O}\left(nr^2r^*\right)$ flops. The algorithm could be sped up by replacing the SVD with a RRQR decomposition, which typically decreases the complexity to $\mathcal{O}\left(nr^2r^*\right)$. Still, the total cost cannot be smaller than the cost of the initial orthogonalization, i.e. $\mathcal{O}\left(dnr^3\right)$ flops.

Algorithm 7: Truncation of a tensor in the TT format

Data: tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\mathcal{A}_1, \ldots, \mathcal{A}_d$, desired TT-ranks $(r_0^* = 1, r_1^*, \ldots, r_{d-1}^*, r_d^* = 1)$

Result: left-orthogonal $\mathcal{A}^* \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\mathcal{A}_1^*, \ldots, \mathcal{A}_d^*$ with the specified TT-ranks

$\mathcal{A}' :=$ Algorithm 6$(\mathcal{A}$, 'right'$)$
$\mathcal{A}_1^{**} := \mathcal{A}_1'$
for $j = 1, 2, \ldots, d-1$ **do**
  Compute SVD: $\left(\mathcal{A}_j^{**}\right)^{<2>} =: \boldsymbol{U}_j \boldsymbol{\Sigma}_j \boldsymbol{V}_j^\top$
  $\boldsymbol{W}_j := \boldsymbol{U}_j(:, 1 : r_j^*)$
  $\boldsymbol{Z}_j := \boldsymbol{V}_j(:, 1 : r_j^*)\boldsymbol{\Sigma}_j(1 : r_j^*, 1 : r_j^*)$
  $\mathcal{A}_j^* := \texttt{reshape}(\boldsymbol{W}_j, [r_{j-1}^*, n_j, r_j^*])$
  $\mathcal{A}_{j+1}^{**} := \mathcal{A}_{j+1}' \times_1 \boldsymbol{Z}_j^\top$
end
$\mathcal{A}_d^* := \mathcal{A}_d^{**}$

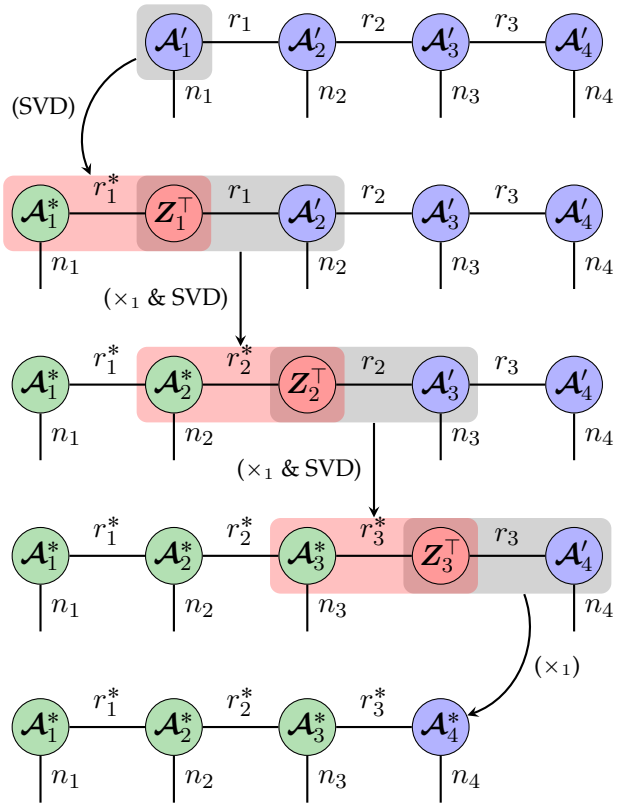

FIGURE 4.3: Tensor network diagram for the compression of $\mathcal{A} \in \mathbb{R}^{n_1 \times \ldots \times n_4}$ using Algorithm 7. Green nodes have second unfoldings with orthonormal columns, whereas blue nodes have first unfoldings with orthonormal rows.

We show in Figure 4.3 how the procedure is carried out on a $n_1 \times \ldots \times n_4$ right-orthogonal tensor $\boldsymbol{\mathcal{A}}'$ with TT-ranks $(1, r_1, r_2, r_3, 1)$. In particular, we can observe that each of the "local" right factors $\boldsymbol{Z}_j$ (for $j \in \{1, \ldots, d-1\}$) acts as an interface between a left-orthogonal and a right-orthogonal sub-tensor, represented respectively in green and blue.

4.2.3 Rounding of tensors in the TT format in two sweeps

Algorithm 7 can be easily modified to tackle the rounding problem: in this case the final TT-ranks are unknown, but we want to guarantee that the relative error (4.10) is smaller than some tolerance $\varepsilon > 0$. To this aim, we compute each new rank r_j^* (for $j \in \{1, \ldots, d-1\}$) in such a way that the approximation has the adjusted relative accuracy δ (with $\delta \leqslant \varepsilon$): if we proceed from left to right, this can be expressed as

$$\frac{\|\boldsymbol{U}_j(:, 1:r_j^*)\boldsymbol{\Sigma}_j(1:r_j^*, 1:r_j^*)\boldsymbol{V}_j(:, 1:r_j^*)^\top - \left(\boldsymbol{\mathcal{A}}_j^{**}\right)^{<2>}\|_F}{\|\boldsymbol{\mathcal{A}}\|_F} \leqslant \delta \tag{4.17}$$

Theorem 4.3 below shows that δ is within a factor (which depends on the order d) of the required accuracy ε. Accordingly, to solve the rounding problem with precision ε, it is enough to run Algorithm 7 with condition (4.17) and $\delta = \varepsilon/\sqrt{d-1}$ (see Algorithm 8).

Algorithm 8: Rounding of a tensor in the TT format

Data: tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_d$, $\varepsilon > 0$
Result: left-orthogonal $\boldsymbol{\mathcal{A}}^* \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1^*, \ldots, \boldsymbol{\mathcal{A}}_d^*$ such that the relative error (4.10) is at most ε

$\delta := \varepsilon/\sqrt{d-1}$
$\boldsymbol{\mathcal{A}}' :=$ Algorithm 6($\boldsymbol{\mathcal{A}}$, 'right')
$\boldsymbol{\mathcal{A}}_1^{**} := \boldsymbol{\mathcal{A}}_1'$
for $j = 1, 2, \ldots, d-1$ **do**
  Compute SVD: $\left(\boldsymbol{\mathcal{A}}_j^{**}\right)^{<2>} =: \boldsymbol{U}_j\boldsymbol{\Sigma}_j\boldsymbol{V}_j^\top$
  Find the smallest r_j^* such that (4.17) holds
  $\boldsymbol{W}_j := \boldsymbol{U}_j(:, 1:r_j^*)$
  $\boldsymbol{Z}_j := \boldsymbol{V}_j(:, 1:r_j^*)\boldsymbol{\Sigma}_j(1:r_j^*, 1:r_j^*)$
  $\boldsymbol{\mathcal{A}}_j^* :=$ `reshape`$(\boldsymbol{W}_j, [r_{j-1}^*, n_j, r_j^*])$
  $\boldsymbol{\mathcal{A}}_{j+1}^{**} := \boldsymbol{\mathcal{A}}_{j+1}' \times_1 \boldsymbol{Z}_j^\top$
end
$\boldsymbol{\mathcal{A}}_d^* := \boldsymbol{\mathcal{A}}_d^{**}$

Theorem 4.3. *The tensor $\boldsymbol{\mathcal{A}}^*$ resulting from Algorithm 8 is such that*

$$\|\boldsymbol{\mathcal{A}}^* - \boldsymbol{\mathcal{A}}\|_F \leqslant \varepsilon\|\boldsymbol{\mathcal{A}}\|_F \tag{4.18}$$

Proof. We follow the idea behind the proof of Theorem 2.2 in [Ose11], and we employ the same notation used in Algorithm 8.

For each $j \in \{0, \dots, d-1\}$, we define the partial approximation $\boldsymbol{\mathcal{A}}^{(j)}$ as the $n_1 \times \dots \times n_d$ tensor which admits the TT decomposition with cores

$$\boldsymbol{\mathcal{A}}^*_1, \ \dots, \ \boldsymbol{\mathcal{A}}^*_j, \ \boldsymbol{\mathcal{A}}^{**}_{j+1}, \ \boldsymbol{\mathcal{A}}'_{j+2}, \ \dots, \ \boldsymbol{\mathcal{A}}'_d$$

where

$$\boldsymbol{\mathcal{A}}^{**}_{j+1} = \boldsymbol{\mathcal{A}}'_{j+1} \times_1 \boldsymbol{Z}_j^\top \quad \text{for } j = 1, \dots, d-1$$

and $\boldsymbol{\mathcal{A}}^{**}_1 = \boldsymbol{\mathcal{A}}'_1$.

It is important to observe that, due to the properties of 1st and 3rd mode products, $\boldsymbol{\mathcal{A}}^{(j)}$ also admits the alternative TT decomposition with cores

$$\boldsymbol{\mathcal{A}}^*_1, \ \dots, \ \boldsymbol{\mathcal{A}}^*_{j-1}, \ \boldsymbol{\mathcal{A}}^*_j \times_3 \boldsymbol{Z}_j, \ \boldsymbol{\mathcal{A}}'_{j+1}, \ \dots, \ \boldsymbol{\mathcal{A}}'_d$$

for any $j \in \{1, \dots, d-1\}$.

With this definition, each $\boldsymbol{\mathcal{A}}^{(j)}$ corresponds to the approximation of $\boldsymbol{\mathcal{A}}$ immediately after the j^{th} iteration of Algorithm 8. In particular, $\boldsymbol{\mathcal{A}}^{(0)} = \boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{A}}^{(d-1)} = \boldsymbol{\mathcal{A}}^*$.

Since a tensor and all of its unfoldings have the same Frobenius norm, we can express the error as

$$\|\boldsymbol{\mathcal{A}}^* - \boldsymbol{\mathcal{A}}\|_F = \|\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(0)}\|_F = \| \left(\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(1)}\right)^{<1>} + \left(\boldsymbol{\mathcal{A}}^{(1)} - \boldsymbol{\mathcal{A}}^{(0)}\right)^{<1>} \|_F$$

The two terms can be manipulated as

$$\left(\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(1)}\right)^{<1>} = (\boldsymbol{\mathcal{A}}^*_1)^{<2>} \left(\boldsymbol{\mathcal{A}}^{(d-1)}_{\geqslant 2} - \boldsymbol{\mathcal{A}}^{(1)}_{\geqslant 2}\right)^\top = \boldsymbol{W}_1 \left(\boldsymbol{\mathcal{A}}^{(d-1)}_{\geqslant 2} - \boldsymbol{\mathcal{A}}^{(1)}_{\geqslant 2}\right)^\top$$

and

$$\begin{aligned}
\left(\boldsymbol{\mathcal{A}}^{(1)} - \boldsymbol{\mathcal{A}}^{(0)}\right)^{<1>} &= (\boldsymbol{\mathcal{A}}^*_1 \times_3 \boldsymbol{Z}_1 - \boldsymbol{\mathcal{A}}^{**}_1)^{<2>} \ {\boldsymbol{\mathcal{A}}'_{\geqslant 2}}^\top \\
&= \left(\boldsymbol{W}_1 \boldsymbol{Z}_1^\top - (\boldsymbol{\mathcal{A}}^{**}_1)^{<2>}\right) {\boldsymbol{\mathcal{A}}'_{\geqslant 2}}^\top \\
&= \left(\boldsymbol{W}_1 \boldsymbol{W}_1^\top (\boldsymbol{\mathcal{A}}^{**}_1)^{<2>} - (\boldsymbol{\mathcal{A}}^{**}_1)^{<2>}\right) {\boldsymbol{\mathcal{A}}'_{\geqslant 2}}^\top \\
&= \left(\boldsymbol{W}_1 \boldsymbol{W}_1^\top - \boldsymbol{I}_{n_1}\right) (\boldsymbol{\mathcal{A}}^{**}_1)^{<2>} \ {\boldsymbol{\mathcal{A}}'_{\geqslant 2}}^\top
\end{aligned}$$

Since $\boldsymbol{W}_1$ has orthonormal columns, it holds that

$$\boldsymbol{W}_1^\top \left(\boldsymbol{W}_1 \boldsymbol{W}_1^\top - \boldsymbol{I}_{n_1}\right) = \boldsymbol{0}$$

Hence, the Pythagorean theorem can be applied:

$$\begin{aligned}
\|\boldsymbol{\mathcal{A}}^* - \boldsymbol{\mathcal{A}}\|_F^2 &= \| \left(\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(1)}\right)^{<1>} \|_F^2 + \| \left(\boldsymbol{\mathcal{A}}^{(1)} - \boldsymbol{\mathcal{A}}^{(0)}\right)^{<1>} \|_F^2 \\
&= \|\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(1)}\|_F^2 + \|\boldsymbol{W}_1 \boldsymbol{Z}_1^\top - (\boldsymbol{\mathcal{A}}^{**}_1)^{<2>}\|_F^2
\end{aligned}$$

Now we consider the second unfolding of $\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(1)}$ and proceed similarly, obtaining

$$\left(\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(2)}\right)^{<2>} = (\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1) \boldsymbol{W}_2 \left(\boldsymbol{\mathcal{A}}^{(d-1)}_{\geqslant 3} - \boldsymbol{\mathcal{A}}^{(2)}_{\geqslant 3}\right)^\top$$

and

$$\left(\boldsymbol{\mathcal{A}}^{(2)} - \boldsymbol{\mathcal{A}}^{(1)}\right)^{<2>} = (\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1)\left(\boldsymbol{W}_2\boldsymbol{W}_2^\top - \boldsymbol{I}_{n_2 r_1^*}\right)(\boldsymbol{\mathcal{A}}_2^{**})^{<2>}\ \boldsymbol{\mathcal{A}}'_{\geqslant 3}{}^\top$$

After removing the common term $\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1$, which has orthonormal columns and does not affect the Frobenius norm, the remaining leading terms are orthogonal, and the Pythagorean theorem yields

$$\|\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(1)}\|_F^2 = \|\boldsymbol{\mathcal{A}}^{(d-1)} - \boldsymbol{\mathcal{A}}^{(2)}\|_F^2 + \|\boldsymbol{W}_2\boldsymbol{Z}_2^\top - (\boldsymbol{\mathcal{A}}_2^{**})^{<2>}\|_F^2$$

A recursive argument leads to the expression

$$\|\boldsymbol{\mathcal{A}}^* - \boldsymbol{\mathcal{A}}\|_F^2 = \sum_{j=1}^{d-1} \|\boldsymbol{W}_j\boldsymbol{Z}_j^\top - \left(\boldsymbol{\mathcal{A}}_j^{**}\right)^{<2>}\|_F^2$$

from which, by using the uniform bound (4.17), we derive

$$\|\boldsymbol{\mathcal{A}}^* - \boldsymbol{\mathcal{A}}\|_F^2 \leqslant (d-1)\delta^2 \|\boldsymbol{\mathcal{A}}\|_F^2 = \varepsilon^2 \|\boldsymbol{\mathcal{A}}\|_F^2$$

□

4.3 Compression of tensors in the TT format using a randomized approach

In the last section we have described the *state-of-the-art* deterministic algorithms to compress tensors in the TT format, and we have shown that the accuracy and efficiency of the procedure rely on the first "sweep", which is used to orthogonalize the tensor. In the same spirit as the one applied to matrices in Chapter 3, we may hope to reduce the complexity of the algorithm from $\mathcal{O}\left(dnr^3\right)$ to $\mathcal{O}\left(dnr^2r^*\right)$. However, the intrinsic cost ($\mathcal{O}\left(dnr^3\right)$ flops) due to the orthogonalization step cannot be avoided in a deterministic approach.

In this section we describe a randomized algorithm which achieves the desired complexity $\mathcal{O}\left(dnr^2r^*\right)$, by exploiting the stunning efficiency with which some operations can be executed in the TT format.

4.3.1 Multidimensional tensor contraction

First, we introduce the operation which the algorithm relies on.

Definition 4.3. *Let $\boldsymbol{\mathcal{A}}$ be a tensor of size $n_1 \times \ldots \times n_d$, and let $j \in \{0, \ldots, d-1\}$. Consider $d-j$ arbitrary vectors $\{\boldsymbol{u}_l\}_{l=j+1}^d$, with $\boldsymbol{u}_l \in \mathbb{R}^{n_l}$ for $l \in \{j+1, \ldots, d\}$.*

We define the mode-$(j+1, \ldots, d)$ contraction of $\boldsymbol{\mathcal{A}}$ onto the rank-1 tensor $\otimes_{l=j+1}^d \boldsymbol{u}_l$ as the $n_1 \times \ldots \times n_j$ tensor (or a scalar if $j = 0$) with entries defined by

$$\left\langle \boldsymbol{\mathcal{A}}, \bigotimes_{l=j+1}^{d} \boldsymbol{u}_l \right\rangle_{(j+1,\ldots,d)} (i_1, \ldots, i_j) = \sum_{i_{j+1}, \ldots, i_d} \boldsymbol{\mathcal{A}}(i_1, \ldots, i_d)\, \boldsymbol{u}_{j+1}(i_{j+1}) \ldots \boldsymbol{u}_d(i_d) \tag{4.19}$$

for $(i_1, \ldots, i_j) \in I_1 \times \ldots \times I_j$*, with the sum over* i_l *being carried out between 1 and* n_l*, for* $l \in \{j+1, \ldots, d\}$.

It is trivial to see that the contraction $\langle \boldsymbol{\mathcal{A}}, \bigotimes_{l=j+1}^{d} \boldsymbol{u}_l \rangle_{(j+1,\ldots,d)}$ can be obtained by removing all the singleton dimensions from

$$\left(\boldsymbol{\mathcal{A}} \times_{j+1} \boldsymbol{u}_{j+1}^\top \times_{j+2} \boldsymbol{u}_{j+2}^\top \ldots \times_d \boldsymbol{u}_d^\top\right) \in \mathbb{R}^{n_1 \times \ldots \times n_j \times 1 \times \ldots \times 1}$$

In particular, if the tensor $\boldsymbol{\mathcal{A}}$ is in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_d$, the terms appearing in the sum (4.19) are separable:

$$\begin{aligned}\left\langle \boldsymbol{\mathcal{A}}, \bigotimes_{l=j+1}^{d} \boldsymbol{u}_l \right\rangle_{(j+1,\ldots,d)} (i_1, \ldots, i_j) = \\ = \boldsymbol{\mathcal{A}}_1(i_1) \ldots \boldsymbol{\mathcal{A}}_j(i_j) \left(\sum_{i_{j+1}} \boldsymbol{\mathcal{A}}_{j+1}(i_{j+1}) \boldsymbol{u}_{j+1}(i_{j+1}) \right) \ldots \left(\sum_{i_d} \boldsymbol{\mathcal{A}}_d(i_d) \boldsymbol{u}_d(i_d) \right) \end{aligned} \quad (4.20)$$

From this expression, we can devise a very efficient algorithm to compute contractions in the TT format, whose original version can be found in [Ose11], where only full contractions are considered.

We proceed in this way: let $\boldsymbol{\mathcal{A}}$ be a tensor of size $n_1 \times \ldots \times n_d$, whose TT decomposition has cores $\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_d$ and TT-ranks $(1, r_1, \ldots, r_{d-1}, 1)$. Also, let $\{\boldsymbol{u}_l\}_{l=j+1}^{d}$ be such that $\langle \boldsymbol{\mathcal{A}}, \bigotimes_{l=j+1}^{d} \boldsymbol{u}_l \rangle_{(j+1,\ldots,d)}$ is defined.

Following (4.20), we compute the vector

$$\boldsymbol{v}_d = \sum_{i_d} \boldsymbol{\mathcal{A}}_d(i_d)\, \boldsymbol{u}_d(i_d) \in \mathbb{R}^{r_{d-1}}$$

and we proceed recursively from right to left

$$\boldsymbol{v}_l = \sum_{i_l} (\boldsymbol{\mathcal{A}}_l(i_l)\, \boldsymbol{v}_{l+1})\, \boldsymbol{u}_l(i_l) \in \mathbb{R}^{r_{l-1}} \quad \text{for } l = d-1, \ldots, j+1$$

Now it suffices to update the j^{th} core

$$\boldsymbol{\mathcal{A}}'_j = \boldsymbol{\mathcal{A}}_j \times_3 \boldsymbol{v}_{j+1}^\top \in \mathbb{R}^{r_{j-1} \times n_j \times 1}$$

so that the TT decomposition of the contraction is

$$\left\langle \boldsymbol{\mathcal{A}}, \bigotimes_{l=j+1}^{d} \boldsymbol{u}_l \right\rangle_{(j+1,\ldots,d)} (i_1, \ldots, i_j) = \boldsymbol{\mathcal{A}}_1(i_1) \ldots \boldsymbol{\mathcal{A}}_{j-1}(i_{j-1}) \boldsymbol{\mathcal{A}}'_j(i_j)$$

This procedure is summarized in Algorithm 9, and costs $\mathcal{O}\big((d-j)nr^2\big)$ flops.

This algorithm also admits an intriguing graphical representation using tensor network diagrams. In Figure 4.4 we consider a tensor with size $n_1 \times \ldots \times n_5$ and TT-ranks $(1, r_1, \ldots, r_4, 1)$, which is contracted along the modes $(3, 4, 5)$ onto the rank-1 tensor $\bigotimes_{l=3}^{5} \boldsymbol{u}_l$: in the first step, we collapse $\boldsymbol{\mathcal{A}}_5$ onto $\boldsymbol{u}_5$ to obtain $\boldsymbol{v}_5$; then $\boldsymbol{\mathcal{A}}_4$, $\boldsymbol{u}_4$ and $\boldsymbol{v}_5$ are used to compute $\boldsymbol{v}_4$ and so on. At the end of the procedure, we collapse the second core onto $\boldsymbol{v}_3$ to obtain the desired TT representation.

Algorithm 9: Mode-$(j+1,\dots,d)$ contraction of a tensor in the TT format

Data: tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \dots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \dots, \boldsymbol{\mathcal{A}}_d$ and TT-ranks $(1, r_1, \dots, r_{d-1}, 1)$, index $j \in \{0, \dots, d-1\}$, vectors $\{\boldsymbol{u}_l\}_{l=j+1}^{d}$ of suitable size

Result: $\boldsymbol{\mathcal{A}}' = \langle \boldsymbol{\mathcal{A}}, \bigotimes_{l=j+1}^{d} \boldsymbol{u}_l \rangle_{(j+1,\dots,d)} \in \mathbb{R}^{n_1 \times \dots \times n_j}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \dots, \boldsymbol{\mathcal{A}}_{j-1}, \boldsymbol{\mathcal{A}}'_j$

$\boldsymbol{v}_{d+1} := 1$
for $l = d, d-1, \dots, j+1$ **do**
  $\boldsymbol{v}_l := \texttt{zeros}(r_{l-1}, 1)$
  for $i_l = 1, 2, \dots, n_l$ **do**
    $\boldsymbol{v}_l := \boldsymbol{v}_l + (\boldsymbol{\mathcal{A}}_l(i_l)\boldsymbol{v}_{l+1})\boldsymbol{u}_l(i_l)$
  end
end
$\boldsymbol{\mathcal{A}}'_j := \boldsymbol{\mathcal{A}}_j \times_3 \boldsymbol{v}_{j+1}^{\top}$

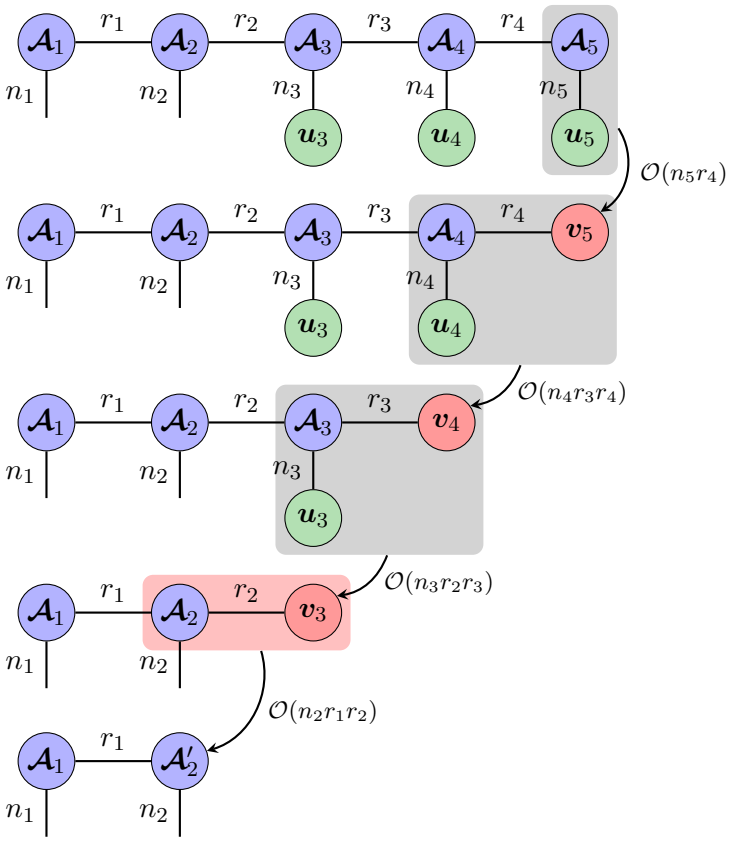

FIGURE 4.4: Tensor network diagram for the contraction of $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \dots \times n_5}$ along the modes $(3, 4, 5)$ onto the rank-1 tensor $\bigotimes_{l=3}^{5} \boldsymbol{u}_l$. At each step, one or two edges are collapsed according to Algorithm 9.

In particular, we may easily evaluate the complexity of the algorithm from the diagram, as follows. For each value of $l \in \{d, d-1, \dots, j+1\}$ we identify the set N_l of nodes (either cores of the tensor, vectors $\boldsymbol{u}_l$, or partial results of the procedure) which are used in the corresponding step of Algorithm 9: such sets are represented as gray rectangles in Figure 4.4. With this definition, the complexity of the step can be simply obtained as the product of the labels of all the edges having (at least) an element of N_l as an endpoint.

Among the many applications of multidimensional contractions, we consider one which is very useful in the context of tensor compression (see Section 4.3.2): given a tensor $\boldsymbol{\mathcal{A}}$, contractions can be applied to compute the matrix-vector product between any unfolding $\boldsymbol{\mathcal{A}}^{<j>}$ and structured "rank-1" vectors, as stated by the following result.

Lemma 4.4. *Let $\boldsymbol{\mathcal{A}}$, j and $\{\boldsymbol{u}_l\}_{l=j+1}^{d}$ be as in Definition 4.3. Let the vector $\boldsymbol{u}^* \in \mathbb{R}^{n_{j+1} \cdots n_d}$ be the Kronecker product*

$$\boldsymbol{u}^* = \boldsymbol{u}_d \otimes \ldots \otimes \boldsymbol{u}_{j+1} \tag{4.21}$$

Then

$$\boldsymbol{\mathcal{A}}^{<j>} \boldsymbol{u}^* = \left(\left\langle \boldsymbol{\mathcal{A}}, \bigotimes_{l=j+1}^{d} \boldsymbol{u}_l \right\rangle_{(j+1,\dots,d)} \right)^{<j>} \tag{4.22}$$

Proof. Let i_{row} and i_{col} be defined as in Definition 2.1. Since i_{col} and the Kronecker product (4.21) induce, respectively, a colexicographical and a lexicographical ordering of the indices $(i_{j+1}, \dots, i_d)$, we have that

$$\boldsymbol{u}^*(i_{col}(i_{j+1}, \dots, i_d)) = \boldsymbol{u}_{j+1}(i_{j+1}) \dots \boldsymbol{u}_d(i_d) \quad \text{for } (i_{j+1}, \dots, i_d) \in I_{j+1} \times \ldots \times I_d$$

Hence

$$\begin{aligned}
\left(\boldsymbol{\mathcal{A}}^{<j>} \boldsymbol{u}^*\right)(i_{row}(i_1, \dots, i_j)) &= \sum_{\alpha=1}^{n_{j+1} \cdots n_d} \boldsymbol{\mathcal{A}}^{<j>}(i_{row}(i_1, \dots, i_j), \alpha) \boldsymbol{u}^*(\alpha) \\
&= \sum_{i_{j+1}, \dots, i_d} \boldsymbol{\mathcal{A}}(i_1, \dots, i_d)\, \boldsymbol{u}_{j+1}(i_{j+1}) \dots \boldsymbol{u}_d(i_d) \\
&= \langle \boldsymbol{\mathcal{A}}, \otimes_{l=j+1}^{d} \boldsymbol{u}_l \rangle_{(j+1,\dots,d)}(i_1, \dots, i_j) \\
&= \left(\langle \boldsymbol{\mathcal{A}}, \otimes_{l=j+1}^{d} \boldsymbol{u}_l \rangle_{(j+1,\dots,d)} \right)^{<j>} (i_{row}(i_1, \dots, i_j))
\end{aligned}$$

for $(i_1, \dots, i_j) \in I_1 \times \ldots \times I_j$. □

Moreover, it is also important to fully understand the meaning of the partial results $\boldsymbol{v}_k$ for $k \in \{j+2, \dots, d\}$ in Algorithm 9. It is easy to check that they appear in the expansion of contractions along fewer modes: indeed, it is trivial to verify that, for any $k \in \{j, \dots, d-1\}$, it holds

$$\left\langle \boldsymbol{\mathcal{A}}, \bigotimes_{l=k+1}^{d} \boldsymbol{u}_l \right\rangle_{(k+1,\dots,d)} (i_1, \dots, i_k) = \boldsymbol{\mathcal{A}}_1(i_1) \dots \boldsymbol{\mathcal{A}}_{k-1}(i_{k-1}) \boldsymbol{\mathcal{A}}_k(i_k) \boldsymbol{v}_{k+1} \tag{4.23}$$

for $(i_1, \dots, i_k) \in I_1 \times \ldots \times I_k$.

4.3.2 Truncation of tensors in the TT format in a single sweep

Now that we have an efficient way to compute the action of any unfolding onto a structured vector, we can try to apply the ideas introduced in Section 3.1 to compress any unfolding of the target tensor.

Given the usual tensor $\boldsymbol{\mathcal{A}}$ in the TT format (4.2), we consider the first unfolding $\boldsymbol{\mathcal{A}}^{<1>} = \boldsymbol{\mathcal{A}}_{\leqslant 1}\boldsymbol{\mathcal{A}}_{\geqslant 2}^{\top}$. We seek a rank-$r_1^*$ dyadic approximation of the form

$$\boldsymbol{\mathcal{A}}^{<1>} \approx \boldsymbol{W}_1 \boldsymbol{Z}_1^{\top} \quad \text{with } \boldsymbol{W}_1^{\top}\boldsymbol{W}_1 = \boldsymbol{I}_{r_1^*}$$

This time, we cannot identify the left factor $\boldsymbol{W}_1$ just from the left interface $\boldsymbol{\mathcal{A}}_{\leqslant 1}$, since in general the TT decomposition of $\boldsymbol{\mathcal{A}}$ is not right-orthogonal: instead, we need to use the whole unfolding (in this issue resides the reason why the orthogonalization step is required to guarantee efficiency in the deterministic approach).

We do this in a randomized fashion, as follows. We multiply the matrix by $r_1^* + p$ random vectors (with $p > 0$ being an oversampling parameter):

$$\mathbf{y}^{(l)} = \boldsymbol{\mathcal{A}}^{<1>}\boldsymbol{\omega}^{(l)} \quad \text{for } l \in \{1, \dots, r_1^* + p\}$$

From the resulting samples we derive an approximation for the left factor $\boldsymbol{W}_1$, e.g. by computing the orthogonal factor of the QR decomposition of the matrix $\left[\mathbf{y}^{(1)} | \dots | \mathbf{y}^{(r_1^*+p)}\right]$.

If each random vector $\boldsymbol{\omega}^{(l)} \in \mathbb{R}^{n_2 \cdots n_d}$ were drawn from a Gaussian distribution, computing the corresponding sample would require $\mathcal{O}\left(rn^{d-1}\right)$ flops, making the algorithm extremely inefficient. Instead, in the wake of Lemma 4.4, we choose random vectors with a "rank-1" structure: to generate each sample $\boldsymbol{\omega}^{(l)}$, for $l \in \{1, \dots, r_1^* + p\}$, we draw $d-1$ Gaussian random vectors $\{\boldsymbol{\omega}_j^{(l)}\}_{j=2}^{d}$ (with $\boldsymbol{\omega}_j^{(l)} \in \mathbb{R}^{n_j}$ for $j \in \{2, \dots, d\}$), and we take

$$\boldsymbol{\omega}^{(l)} = \boldsymbol{\omega}_d^{(l)} \otimes \dots \otimes \boldsymbol{\omega}_2^{(l)}$$

With this definition, the computation of each of the samples $\{\mathbf{y}^{(l)}\}_{l=1}^{r_1^*+p}$ can be carried out very efficiently as a multidimensional contraction, thanks to Lemma 4.4.

Once the samples have been computed, they are orthogonalized and arranged in a $n_1 \times (r_1^* + p)$ matrix $\boldsymbol{W}_1'$ with orthonormal columns, so that we have the approximate rank-$(r_1^* + p)$ decomposition

$$\boldsymbol{\mathcal{A}}^{<1>} \approx \boldsymbol{W}_1' \left(\boldsymbol{W}_1'^{\top} \boldsymbol{\mathcal{A}}^{<1>} \right) = \boldsymbol{W}_1' \left(\boldsymbol{W}_1'^{\top} \left(\boldsymbol{\mathcal{A}}_1\right)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 2}^{\top} \right)$$

Similarly to Algorithms 6 and 7, $\boldsymbol{W}_1'$ is used to replace the first core of the tensor, so that $\left(\boldsymbol{\mathcal{A}}_1'\right)^{<2>} = \boldsymbol{W}_1'$, and we use the right factor of the rank-$(r_1^* + p)$ approximation of $\left(\boldsymbol{\mathcal{A}}_1\right)^{<2>}$, i.e.

$$\boldsymbol{Z}_1' = \left(\boldsymbol{\mathcal{A}}_1\right)^{<2>\top} \boldsymbol{W}_1',$$

to update the second core:

$$\boldsymbol{\mathcal{A}}_2'' = \boldsymbol{\mathcal{A}}_2 \times_1 \boldsymbol{Z}_1'^{\top}$$

Now we consider the second unfolding

$$\boldsymbol{\mathcal{A}}^{<2>} = \boldsymbol{\mathcal{A}}_{\leqslant 2}\boldsymbol{\mathcal{A}}_{\geqslant 3}^{\top} = \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{\mathcal{A}}_{\leqslant 1}\right) \left(\boldsymbol{\mathcal{A}}_2\right)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}^{\top}$$

$$\begin{aligned}
&\approx \left(\boldsymbol{I}_{n_2} \otimes \left(\boldsymbol{W}_1' \, \boldsymbol{Z}_1'^{\top}\right)\right) (\boldsymbol{\mathcal{A}}_2)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}^{\top} \\
&= \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1'\right) \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{Z}_1'^{\top}\right) (\boldsymbol{\mathcal{A}}_2)^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}^{\top} \\
&= \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1'\right) (\boldsymbol{\mathcal{A}}_2'')^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}^{\top}
\end{aligned}$$

Since $\boldsymbol{W}_1'$ has orthonormal columns, so does $\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1'$, and it suffices to estimate the range of $(\boldsymbol{\mathcal{A}}_2'')^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}^{\top}$. Once again, this can be done using multidimensional contractions: indeed, it is sufficient to start the computation of the new samples

$$\mathbf{y}^{(l)} = \boldsymbol{\mathcal{A}}^{<2>} \left(\boldsymbol{\omega}_d^{(l)} \otimes \ldots \otimes \boldsymbol{\omega}_3^{(l)}\right) \quad \text{for } l \in \{1, \ldots, r_2^* + p\}$$

with $\{\boldsymbol{\omega}_j^{(l)}\}_{j=3}^d$ (for $l \in \{1, \ldots, r_2^* + p\}$) being a suitable subset of the random vectors used in the previous step.

In particular, we are not interested in the contributions given by the original first and second cores: thus, we just compute the value of $\boldsymbol{v}_3$ in Algorithm 9, which represents one of the partial samples

$$\bar{\mathbf{y}}^{(l)} = \boldsymbol{\mathcal{A}}_{\geqslant 3}^{\top} \left(\boldsymbol{\omega}_d^{(l)} \otimes \ldots \otimes \boldsymbol{\omega}_3^{(l)}\right) \quad \text{for } l \in \{1, \ldots, r_2^* + p\}$$

Then we multiply each of these vectors (from the left) by $(\boldsymbol{\mathcal{A}}_2'')^{<2>}$ to get the desired results. The final samples are orthonormalized and arranged in a $(n_2(r_1^*+p)) \times (r_2^*+p)$ matrix $\boldsymbol{W}_2'$ (with orthonormal columns), so that we have the approximate rank-(r_2^*+p) decomposition

$$\boldsymbol{\mathcal{A}}^{<2>} \approx \Big(\left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1'\right) \boldsymbol{W}_2' \Big) \Big(\boldsymbol{W}_2'^{\top} (\boldsymbol{\mathcal{A}}_2'')^{<2>} \boldsymbol{\mathcal{A}}_{\geqslant 3}^{\top} \Big)$$

Now the second core is overwritten in such a way that $(\boldsymbol{\mathcal{A}}_2')^{<2>} = \boldsymbol{W}_2'$, and the third core is updated using

$$\boldsymbol{Z}_2' = (\boldsymbol{\mathcal{A}}_2'')^{<2>\top} \boldsymbol{W}_2'$$

so that

$$\boldsymbol{\mathcal{A}}_3'' = \boldsymbol{\mathcal{A}}_3 \times_1 \boldsymbol{Z}_2'^{\top}$$

Then this process is repeated on all the cores.

Since this results in a tensor with TT-ranks $(1, r_1^* + p, \ldots, r_{d-1}^* + p, 1)$, we carry out a (deterministic) truncation of the tensor down to the desired TT-ranks. In particular, since the tensor $\boldsymbol{\mathcal{A}}'$ is left-orthogonal by construction, it suffices to apply Algorithm 7 in reverse, without the need for a preliminary orthogonalization step.

The resulting procedure is summarized in Algorithm 11, where the subroutine described in Algorithm 10 is used to compute the random samples through multidimensional contractions. Since Algorithm 10 computes the samples recursively, its cost is $\mathcal{O}\big(dnr^2k\big)$ flops, which results in $\mathcal{O}\big(dnr^2(r^* + p)\big)$ flops when applied within Algorithm 11.

This operation, and the update in line 10 (which costs $\mathcal{O}\big(nr^2(r^* + p)\big)$ flops per iteration), are responsible for the overall complexity of the algorithm, which equals $\mathcal{O}\big(dnr^2(r^* + p)\big)$.

Algorithm 10: Generation of samples for the range of tensor unfoldings

Data: tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_d$ and TT-ranks $(r_0 = 1, r_1, \ldots, r_{d-1}, r_d = 1)$, number of samples k

Result: $\{\boldsymbol{Y}_j\}_{j=2}^{d}$, with $\boldsymbol{Y}_j = \boldsymbol{\mathcal{A}}_{\geqslant j}^{\top} \boldsymbol{\Omega}_j \in \mathbb{R}^{r_{j-1} \times k}$ for a structured random matrix $\boldsymbol{\Omega}_j \in \mathbb{R}^{(n_j \cdots n_d) \times k}$ (for $j = 2, \ldots, d$)

$\boldsymbol{Y}_{d+1} := [1| \ldots |1]$ % of size $1 \times k$
for $j = d, d-1, \ldots, 2$ **do**
  Generate Gaussian matrix $\boldsymbol{\Omega}_j = \left[\boldsymbol{\omega}_j^{(1)} \middle| \ldots \middle| \boldsymbol{\omega}_j^{(k)}\right] \in \mathbb{R}^{n_j \times k}$
  $\boldsymbol{Y}_j := \texttt{zeros}(r_{j-1}, k)$
  for $l = 1, \ldots, k$ **do**
    for $i_j = 1, \ldots, n_j$ **do**
      $\boldsymbol{Y}_j(:, l) := \boldsymbol{Y}_j(:, l) + \left(\boldsymbol{\mathcal{A}}_j(i_j) \boldsymbol{Y}_{j+1}(:, l)\right) \boldsymbol{\omega}_j^{(l)}(i_j)$
    end
  end
end

Algorithm 11: Randomized truncation of a tensor in the TT format

Data: tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_d$, desired TT-ranks $(r_0^* = 1, r_1^*, \ldots, r_{d-1}^*, r_d^* = 1)$, integer $p \geqslant 0$

Result: $\boldsymbol{\mathcal{A}}^* \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1^*, \ldots, \boldsymbol{\mathcal{A}}_d^*$ with the specified TT-ranks, such that $\boldsymbol{\mathcal{A}} \approx \boldsymbol{\mathcal{A}}^*$

$k := \max\{r_1^*, \ldots, r_{d-1}^*\} + p$
$\{\boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_d\} :=$ Algorithm 10$(\boldsymbol{\mathcal{A}}, k)$
$\boldsymbol{\mathcal{A}}_1'' := \boldsymbol{\mathcal{A}}_1$
$r_{old}^* := 1$
for $j = 1, 2, \ldots, d-1$ **do**
  $\boldsymbol{Y}_j' := \left(\boldsymbol{\mathcal{A}}_j''\right)^{<2>} \boldsymbol{Y}_{j+1}(:, r_j^* + p)$
  Compute QR decomposition: $\boldsymbol{Y}_j' =: \boldsymbol{W}_j' \boldsymbol{R}_j$
  $\boldsymbol{\mathcal{A}}_j' := \texttt{reshape}(\boldsymbol{W}_j', [r_{old}^*, n_j, r_j^* + p])$
  $\boldsymbol{Z}_j' := \left(\boldsymbol{\mathcal{A}}_j''\right)^{<2>\top} \boldsymbol{W}_j'$
  $\boldsymbol{\mathcal{A}}_{j+1}'' := \boldsymbol{\mathcal{A}}_{j+1} \times_1 \boldsymbol{Z}_j'^{\top}$
  $r_{old}^* := r_j^* + p$
end
$\boldsymbol{\mathcal{A}}_d' := \boldsymbol{\mathcal{A}}_d''$
$\boldsymbol{\mathcal{A}}^* :=$ Algorithm 7$(\boldsymbol{\mathcal{A}}', (1, r_1^*, \ldots, r_{d-1}^*, 1))$ % from right to left

4.3.3 Randomized rounding of tensors in the TT format

By applying Algorithm 11, it is possible to truncate a tensor down to some prescribed TT-ranks. In this section we describe a possible way to extend this procedure to tackle the rounding problem with precision $\varepsilon > 0$.

Given the usual tensor $\boldsymbol{\mathcal{A}}$ in the TT-format (4.2), we wish to find a tensor $\boldsymbol{\mathcal{A}}^*$ with smaller TT-ranks, such that the relative Frobenius error (4.10) is smaller than ε. Due to the trivial generalization of Theorem 4.3 to non-orthogonal tensors, it is sufficient to require that

$$\frac{\|\boldsymbol{W}'_j \, \boldsymbol{W}'^{\top}_j \left(\boldsymbol{\mathcal{A}}''_j\right)^{<2>} \boldsymbol{\mathcal{A}}^{\top}_{\geqslant j+1} - \left(\boldsymbol{\mathcal{A}}''_j\right)^{<2>} \boldsymbol{\mathcal{A}}^{\top}_{\geqslant j+1}\|_F}{\|\boldsymbol{\mathcal{A}}\|_F} \leqslant \frac{\varepsilon}{\sqrt{d-1}} \tag{4.24}$$

for $j \in \{1, \dots, d-1\}$, using the same notation as in Algorithm 11.

Unfortunately, in this case it is quite expensive to compute the left hand side, or even to estimate it with reasonable accuracy, due to the presence of the right interface matrix $\boldsymbol{\mathcal{A}}_{\geqslant j+1}$. In this section we describe an alternative adaptive approach, which achieves a cost very similar to the truncation case.

Let $\delta = \varepsilon/\sqrt{d-1}$, and assume that $\widehat{r}_j$ is the δ-rank of $\boldsymbol{\mathcal{A}}^{<j>}$ (for $j \in \{1, \dots, d-1\}$), i.e. it is the smallest integer such that there exists a $(n_1 \cdots n_j) \times \widehat{r}_j$ matrix $\boldsymbol{U}_j$ with orthonormal columns such that

$$\frac{\|\left(\boldsymbol{U}_j \boldsymbol{U}_j^{\top} - \boldsymbol{I}_{n_1 \cdots n_j}\right) \boldsymbol{\mathcal{A}}^{<j>}\|_F}{\|\boldsymbol{\mathcal{A}}\|_F} \leqslant \delta$$

Theorems 4.1 and 4.3 ensure that there exists a tensor with TT-ranks $(1, \widehat{r}_1, \dots, \widehat{r}_{d-1}, 1)$ which solves the rounding problem.

Consider a vector $(1, r_1^{(0)}, \dots, r_{d-1}^{(0)}, 1)$ containing guesses for the TT-ranks of the compressed tensor. We start by approximating $\boldsymbol{\mathcal{A}}$ with a tensor $\boldsymbol{\mathcal{A}}'$ having the specified TT-ranks. To this aim, we run Algorithm 11 with $p = 0$ (in particular we skip the recompression step, i.e. line 14).

Now we wish to know whether $\boldsymbol{\mathcal{A}}'$ achieves (4.24) for $j \in \{1, \dots, d-1\}$. We may start by checking if the condition

$$r_j^{(0)} \geqslant \widehat{r}_j \quad \text{for } j \in \{1, \dots, d-1\}$$

holds. However, the ranks $\{\widehat{r}_j\}_{j=1}^{d-1}$ are unknown (and it would be expensive to compute them). Hence, we follow a different route: using Algorithm 8, we recompress $\boldsymbol{\mathcal{A}}'$ down to precision ε, and check if all the TT-ranks (except r_0 and r_d) decrease by at least $q > 0$.

Here we are assuming that, in order to be a good (enough) approximation of $\boldsymbol{\mathcal{A}}$, the tensor $\boldsymbol{\mathcal{A}}'$ should be able to identify accurately the first $\widehat{r}_j + q$ singular values of $\boldsymbol{\mathcal{A}}^{<j>}$, for $j \in \{1, \dots, d-1\}$. In particular, the parameter q is used to counter two effects due to the randomness of the algorithm: foremost, the uncertainty in the approximation of the first $\widehat{r}_j$ singular values (and vectors) of the unfolding $\boldsymbol{\mathcal{A}}^{<j>}$, since they represent the true objective of the compression; in second measure, the error in the approximation of the remaining $r_j^{(0)} - \widehat{r}_j$ singular values, which may

cause the recompression to stop earlier than required, due to a poor estimation of $\|\mathcal{A}' - \mathcal{A}\|_F$ and $\|\mathcal{A}\|_F$.

As such, the value of q should be prescribed based on the available information on the spectra of the unfoldings. For exponentially decaying singular values, a small value of q is usually enough (see Section 4.4.3), whereas a larger value should be used for slower decays.

If the check is successful, we accept $\mathcal{A}'$ (or its recompressed version) as the solution of the rounding problem. Otherwise, we increase all the dubious ranks (i.e. those which do not decrease enough through the recompression step) by some fixed amount $m > 0$. This results in a vector containing the new guesses $(1, r_1^{(1)}, \ldots, r_{d-1}^{(1)}, 1)$.

Now we restart the procedure using the updated guesses for the TT-ranks, and we repeat this truncation–rounding–update cycle until the condition on the TT-ranks is satisfied (see Algorithm 12).

Algorithm 12: Randomized rounding of a tensor in the TT format

Data: tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\mathcal{A}_1, \ldots, \mathcal{A}_d$, $\varepsilon > 0$, guess for the TT-ranks $(1, r_1^{(0)}, \ldots, r_{d-1}^{(0)}, 1)$, positive integers q and m

Result: $\mathcal{A}^* \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\mathcal{A}_1^*, \ldots, \mathcal{A}_d^*$ such that the relative error (4.10) is at most ε with high probability

```
l := 0
do
    A' := Algorithm 11(A, (1, r_1^(l), ..., r_{d-1}^(l), 1), 0)
    A* := Algorithm 8(A', ε)                          % from right to left
    (1, r_1*, ..., r_{d-1}*, 1) = TT-ranks(A*)
    changed := false
    for j = 1, ..., d − 1 do
        if r_j^(l) ⩾ r_j* + q then
            r_j^(l+1) := r_j^(l)
        else
            r_j^(l+1) := r_j^(l) + m
            changed := true
        end
    end
    l := l + 1
while changed
```

To analyze the complexity of the algorithm, we assume for ease of notation that

$$r_1^{(0)} = \ldots = r_{d-1}^{(0)} = r^{(0)} \quad \text{and} \quad \widehat{r}_1 = \ldots = \widehat{r}_{d-1} = \widehat{r} = r^{(0)} + km - q$$

for some integer k. The algorithm carries out approximately k iterations, with the l^{th} one (for $l \in \{1, \ldots, k\}$) costing $\mathcal{O}\left(dnr^2(r^{(0)} + lm)\right)$ flops because of Algorithm 11.

This results in an overall complexity equal to

$$\sum_{l=1}^{k} \mathcal{O}\left(dnr^2(r^{(0)} + lm)\right) = \mathcal{O}\left(dnr^2\left(r^{(0)} + km\right)k\right) = \mathcal{O}\left(dnr^2\left(\widehat{r} + q\right)k\right)$$

We may expect to be able to optimize the algorithm in a recursive fashion by exploiting the fact that the tensors $\boldsymbol{\mathcal{A}}'$ have ranks which increase as the algorithm proceeds. Indeed, we can reduce the total cost of the sampling step (see lines 2 and 6 in Algorithm 11) by recycling the previously used random vectors. However, it is impossible to speed up the computation of the compressed cores (line 10), since the orthogonal matrices $\boldsymbol{W}_j'$ (for $j \in \{2, \ldots, d-1\}$) cannot be updated in an incremental way.

4.4 Numerical examples

4.4.1 Truncation of a Scholes-like tensor

We consider $d > 0$, a domain $\Omega \subset \mathbb{R}^d$, and a symmetric matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ with entries $\boldsymbol{\Sigma}(\alpha, \beta) = \sigma_{\alpha\beta}$ for $(\alpha, \beta) \in \{1, \ldots, d\}^2$. We define a second order differential operator over $C^2(\Omega)$ as (see [PP07])

$$\mathcal{L} = \sum_{\alpha=1}^{d} \sum_{\beta=\alpha+1}^{d} \sigma_{\alpha\beta} \frac{\partial^2}{\partial x_\alpha \partial x_\beta}$$

which can be discretized, e.g. using finite differences, with the linear operator

$$A = \sum_{\alpha=1}^{d} \sum_{\beta=\alpha+1}^{d} \sigma_{\alpha\beta} W_\alpha W_\beta$$

In particular, each W_α (for $\alpha \in \{1, \ldots, d\}$) can be interpreted as the discrete gradient operator with respect to the α^{th} component.

If Ω is discretized using a d-dimensional uniform Cartesian grid with m points in each dimension, the operators W_α and A can be interpreted as $m^d \times m^d$ matrices:

$$\boldsymbol{W}_\alpha = \boldsymbol{I}_m \otimes \ldots \otimes \boldsymbol{I}_m \otimes \underbrace{\boldsymbol{B}}_{\alpha^{\text{th}} \text{ position}} \otimes \boldsymbol{I}_m \otimes \ldots \otimes \boldsymbol{I}_m$$

and

$$\boldsymbol{A} = \sum_{\alpha=1}^{d-1} \sum_{\beta=\alpha+1}^{d} \sigma_{\alpha\beta} \left(\boldsymbol{I}_m \otimes \ldots \otimes \boldsymbol{I}_m \otimes \underbrace{\boldsymbol{B}}_{\alpha^{\text{th}} \text{ position}} \otimes \boldsymbol{I}_m \otimes \ldots \right.$$
$$\left. \ldots \otimes \boldsymbol{I}_m \otimes \underbrace{\boldsymbol{B}}_{\beta^{\text{th}} \text{ position}} \otimes \boldsymbol{I}_m \otimes \ldots \otimes \boldsymbol{I}_m \right) \quad (4.25)$$

with $\boldsymbol{B}$ being a $m \times m$ discretization of the 1-dimensional gradient operator.

If each $m \times m$ matrix appearing in the Kronecker product is reshaped into a vector of size m^2, using the change of indices

$$i^*(i, j) = i + m(j - 1) \quad \text{for } (i, j) \in \{1, \ldots, m\}^2,$$

then we can interpret $\boldsymbol{A}$ as a $m^2 \times \ldots \times m^2$ tensor $\boldsymbol{\mathcal{A}}$ of order d, whose entries are

$$\boldsymbol{\mathcal{A}}(i^*(i_1, j_1), \ldots, i^*(i_d, j_d)) = \boldsymbol{A}(m^{d-1}i_1 + \ldots + mi_{d-1} + i_d, m^{d-1}j_1 + \ldots + mj_{d-1} + j_d)$$

for $(i_l, j_l) \in \{1, \ldots, m\}^2$, for $l \in \{1, \ldots, d\}$.

In this representation, (4.25) can be viewed as the canonical decomposition of $\boldsymbol{\mathcal{A}}$ with canonical rank $d(d-1)/2$: indeed,

$$\boldsymbol{\mathcal{A}}(i^*(i_1, j_1), \ldots, i^*(i_d, j_d)) = \sum_{\alpha=1}^{d-1} \sum_{\beta=\alpha+1}^{d} \boldsymbol{U}_1((i_1, j_1), (\alpha, \beta)) \ldots \boldsymbol{U}_d((i_d, j_d), (\alpha, \beta))$$

where

$$\boldsymbol{U}_l((i_l, j_l), (\alpha, \beta)) = \begin{cases} \boldsymbol{B}(i_l, j_l) & \text{if } l \in \{\alpha, \beta\} \\ \boldsymbol{I}_m(i_l, j_l) & \text{if } l \notin \{\alpha, \beta\} \end{cases} \quad \text{for } l \in \{1, \ldots, d\}$$

From here, we can derive a representation of $\boldsymbol{\mathcal{A}}$ in the TT format with TT-ranks $(1, d(d-1)/2, \ldots, d(d-1)/2, 1)$ by applying the strategy described in Section 4.1.

However, it can be proven (see [Ose11]) that there exists a TT representation of $\boldsymbol{\mathcal{A}}$ whose TT-ranks $(1, r_1^*, \ldots, r_{d-1}^*, 1)$ are given by

$$r_j^* = 2 + \min\{j, d-j\} \quad \text{for } j \in \{1, \ldots, d-1\}$$

For this reason, we wish to compress the tensor to TT-ranks $(1, r_1^*, \ldots, r_{d-1}^*, 1)$ by applying Algorithms 7 and 11 (respectively deterministic and randomized truncation) for $d \in \{5, 10, \ldots, 40\}$. We choose $m = 10$ and $\boldsymbol{B}$ as the first-order finite differences approximation of the 1-dimensional gradient operator on a uniform grid (we ignore the leading constant due to the grid size, since it does not affect our analysis). The elements of $\boldsymbol{\Sigma}$ are randomly drawn from the uniform distribution.

The comparison is carried out in the computational environment described in Section 3.3.1, yielding the results shown in Figure 4.5. In particular, in this and the following examples, we employ the TTeMPS toolbox (`http://anchp.epfl.ch/TTeMPS`) for the storage and for most of the basic operations (e.g. sums, norms, orthogonalizations) in the TT format.

For all simulations, both algorithms yield a relative error smaller than 10^{-13}, in agreement with the theoretical bound on the TT-ranks. As d increases, the error tends to increase: for both algorithms this is mainly due to the round-off errors introduced by computing QR decompositions (and SVDs in the deterministic approach).

In particular, the error appears higher (in the majority of the simulations) for Algorithm 3: most likely, this is due to the fact that the deterministic algorithm has to compute QR decompositions of bigger matrices. Indeed, the first sweep of Algorithm 3 deals with matrices of size $(nd(d-1)/2) \times (d(d-1)/2)$, whereas the largest matrix whose QR decomposition is computed in Algorithm 7 has approximately size $(n(d/2 + p + 2)) \times (d/2 + p + 2)$.

In terms of computing time, the deterministic algorithm runs faster for small tensors ($d \in \{5, 10\}$), while the opposite holds true for bigger tensors. For instance, for $d = 40$, Algorithm 11 is more than four times faster than Algorithm 7, with a running time of 70.0 seconds versus 304 seconds.

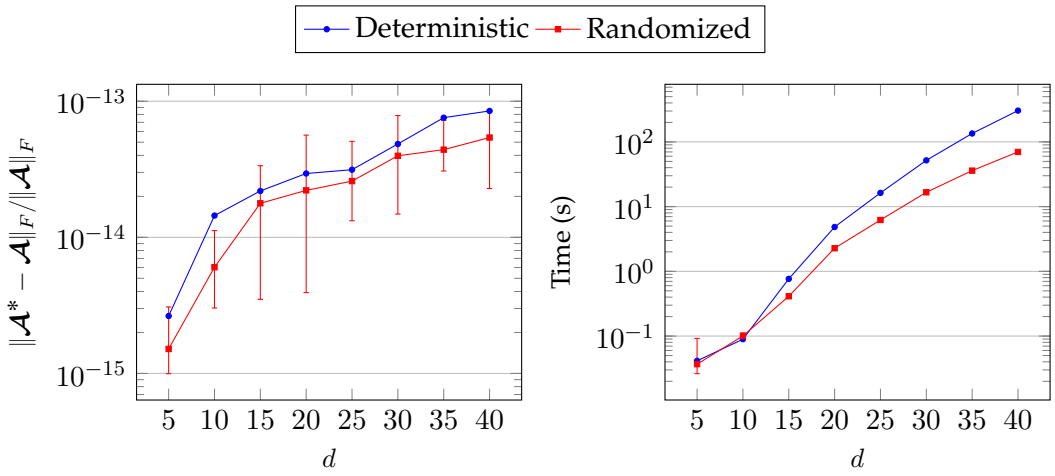

FIGURE 4.5: Relative error (left) and computing time (right) for Algorithms 7 and 11. For the randomized algorithm, we use an oversampling $p = 2$. The error bars show the range of values obtained over $N = 20$ simulations.

4.4.2 Truncation of the discretization of a rational function

We consider the positive integers d and n, as well as $\gamma \in \mathbb{R}^+$, and we discretize uniformly the interval $[\frac{1}{10}, \frac{n}{10}]$ with the points $x_i = \frac{i}{10}$, for $i \in \{1, \dots, n\}$. We define the d-dimensional tensor $\mathcal{A}$ of size $n \times \dots \times n$ as the natural generalization of the matrix A defined in Section 3.3.2:

$$\mathcal{A}(i_1, \dots, i_d) = f_{d,\gamma}(x_{i_1}, \dots, x_{i_d}) = (x_{i_1} + \dots + x_{i_d})^{-\gamma} \quad \text{for } (i_1, \dots, i_d) \in \{1, \dots, n\}^d$$

Due to the smoothness of $f_{d,\gamma}$ in $\left(0, \frac{n+1}{10}\right)^d$, each unfolding of $\mathcal{A}$ has singular values which decay (at least) exponentially (see e.g. [GL17]). Indeed, if the tensor is compressed down to its numerical rank (i.e. it is rounded to machine precision), the ranks are quite small: for $n = 50$, $\gamma = 1$ and $d \in \{3, 5, 10\}$ the TT-ranks are uniformly bounded by 10.

Given the values of n, γ and d specified above, the tensor $\mathcal{A}$ is generated by applying an exponential sum approximation (see e.g. [BH05] and [McL16]) with 25 terms. With this choice, we obtain a tensor with uniform TT-ranks $(1, 25, \dots, 25, 1)$, which approximates the function $f_{d,\gamma}$ with (at least) a uniform precision $2.3 \cdot 10^{-10}$ over the n^d sample points $\{\frac{1}{10}, \dots, \frac{n}{10}\}^d$.

Now we compress the tensor to uniform TT-ranks $(1, 4, \dots, 4, 1)$ using Algorithm 11, while keeping the result of Algorithm 7 as a benchmark. In particular, we are interested in observing how the results depend on the oversampling parameter p.

The behavior of the relative error is shown in Figure 4.6. As in the matrix case (see Section 3.3.2), small values of p (e.g. $p = 3$) are sufficient to achieve the optimal accuracy. This agrees with the fast decay rate of the singular values of all the unfoldings of $\mathcal{A}$.

In particular, the error appears to converge to the optimal value with (at least) exponential rate in p, even though the theoretical analysis of the randomized approach (see Section 3.1.1) cannot be easily generalized to "rank-1" random vectors.

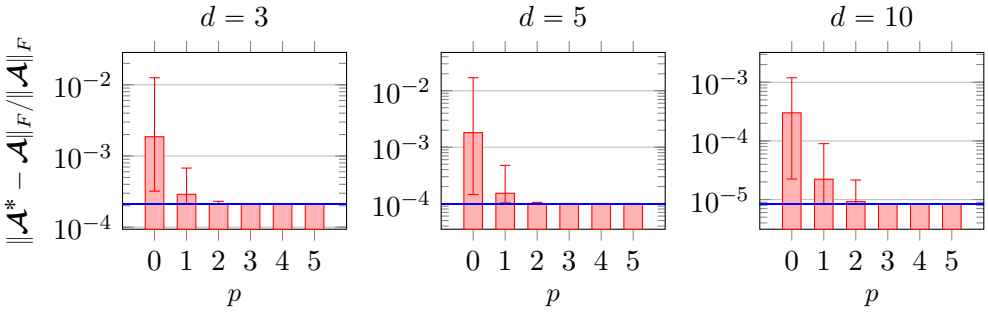

FIGURE 4.6: Relative error versus oversampling for Algorithm 11. The bars connect the minimum and the maximum value of the relative error obtained over $N = 20$ simulations. In blue the error obtained with the deterministic algorithm.

4.4.3 Rounding of a random tensor

Given the positive integers n, r and d (with $n \geqslant r$), we can build a d-dimensional tensor $\boldsymbol{\mathcal{A}}$ of size $n \times \ldots \times n$ with TT-ranks $(1, r, \ldots, r, 1)$ in the following way. We consider d randomly generated $n \times r$ matrices with orthogonal columns $\{\boldsymbol{U}_j\}_{j=1}^d$ and we use them to prescribe the canonical decomposition of $\boldsymbol{\mathcal{A}}$:

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) = \sum_{\alpha=1}^{r} \boldsymbol{U}_1(i_1, \alpha) \ldots \boldsymbol{U}_d(i_d, \alpha)$$

It is trivial to verify that the TT decomposition of $\boldsymbol{\mathcal{A}}$ obtained by using the method described in Section 4.1 is both left- and right-orthogonal.

With a small modification, we can prescribe an arbitrary decay for the singular values of the unfoldings. Let $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_r > 0$ and consider

$$\boldsymbol{\mathcal{B}}(i_1, \ldots, i_d) = \sum_{\alpha=1}^{r} \sigma_\alpha \boldsymbol{U}_1(i_1, \alpha) \ldots \boldsymbol{U}_d(i_d, \alpha) \tag{4.26}$$

Proceeding again as in Section 4.1, we may compute a right-orthogonal TT decomposition of $\boldsymbol{\mathcal{B}}$ with

$$\boldsymbol{\mathcal{B}}_1(1, i_1, \alpha_1) = \sigma_{\alpha_1} \boldsymbol{U}_1(i_1, \alpha_1) \quad \text{for } (i_1, \alpha_1) \in I_1 \times \{1, \ldots, r\}$$

For this decomposition, it is easy to verify that all unfoldings have (non-zero) singular values $(\sigma_1, \ldots, \sigma_r)$.

Now we fix $n = 50$, $r = 50$ and $d = 20$, and we prescribe an exponential decay $\sigma_j = e^{1-j}$ for $j \in \{1, \ldots, r\}$. Using Algorithms 8 and 12, we wish to compress $\boldsymbol{\mathcal{B}}$ down to precision $\varepsilon \in \{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}\}$. In particular, we choose $m = 3$ and $q = 2$ in the randomized approach.

By generalizing the Eckart-Young theorem (see e.g. [EY36]), we can identify the solution of the rounding problem whose TT-ranks are uniformly the smallest among all solutions: if we want the relative error (4.10) to be smaller than ε, it is sufficient to truncate the sum appearing in (4.26) at the term r^*, with r^* being the smallest positive integer such that

$$\sum_{j=r^*+1}^{r} \sigma_j^2 \leqslant \varepsilon^2 \sum_{j=1}^{r} \sigma_j^2$$

For the choices of $(\sigma_j)_{j=1}^r$ and ε specified above, the corresponding values of r^* are $\{6, 11, 15, 20\}$ respectively.

For the randomized algorithm we consider two cases: first, we assume that no initial guess is given, or equivalently that $r_1^{(0)} = \ldots = r_{d-1}^{(0)} = 0$; secondly, we assume that we are given an initial guess with $r_1^{(0)} = \ldots = r_{d-1}^{(0)} = \lceil r^*/2 \rceil$.

In Figure 4.7 we show the results of the numerical experiment, which is carried out in the environment described in Section 3.3.1. From the left plot, we can observe that the required precision is almost always satisfied, and that the relative error is within a small factor of ε even in case of failure. From the right plot we can see that Algorithm 12 is always faster than Algorithm 8. We can also observe that the gap between the speed of the two approaches is particularly pronounced for larger values of ε: this is due to the fact that the cost of the randomized approach depends only weakly on the prescribed precision.

In Figure 4.8 we report the largest TT-ranks of the compressed tensors at the end of each algorithm. For the randomized approach we also plot the values before the last recompression, i.e. the final TT-ranks of $\mathcal{C}'$ in Algorithm 12. In all cases, we can verify that only a modest oversampling (measured as the variation between the TT-ranks before and after recompression) is needed for the completion of Algorithm 12.

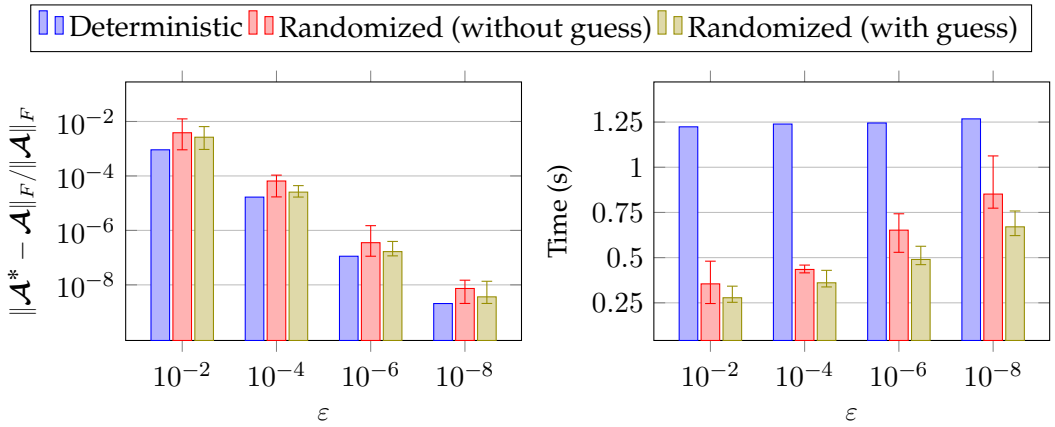

FIGURE 4.7: Relative error (left) and computing time (right) for Algorithms 8 and 12. The bars connect the minimum and the maximum values obtained over $N = 20$ simulations.

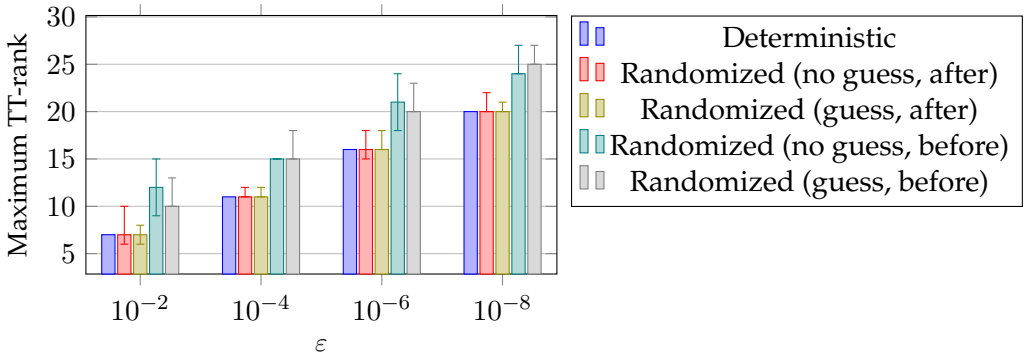

FIGURE 4.8: Maximum TT-rank of the compressed tensor for Algorithms 8 and 12. For the randomized algorithm the plot shows the modal values of the maximal TT-ranks over $N = 20$ simulations, both before and after recompression. Also, the bars show the range of values obtained.

Chapter 5

Compression of Hadamard products

Hadamard products are a fundamental operation in matrix and tensor computations, whose importance is mostly due to the fact that they represent the discrete version of point-wise products of functions. They can be carried out in a very trivial way on matrices and tensors which are stored as full arrays.

However, many issues arise if the two factors are in a structured format, e.g. the low-rank dyadic decomposition (1.1) for matrices and the TT format (4.2) for tensors. In these cases, Hadamard products increase the sizes of the factors/cores: as we show in the upcoming sections, all the ranks grow in a quadratic fashion. This leads to a substantial increase in the computational cost of most manipulations of the matrix/tensor, and also in the memory required for its storage.

For this reason, it is important to recompress properly the result of the Hadamard product. Still, methods which rely, as an intermediate step, on the computation of the full product are quite expensive, since they are affected by the issues described above. In this chapter, we devise randomized algorithms which can compress Hadamard products without computing the full matrix/tensor, exploiting the particular structure that Hadamard products induce on the dyadic decomposition and on the TT format.

5.1 Compression of Hadamard products of matrices

Consider two $m \times n$ matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, whose low-rank approximations, with ranks r_A and r_B respectively, are known:

$$\boldsymbol{A} = \boldsymbol{W}_A \boldsymbol{Z}_A^\top \quad \text{and} \quad \boldsymbol{B} = \boldsymbol{W}_B \boldsymbol{Z}_B^\top$$

In particular, we assume that $r_A \geqslant r_B$ without loss of generality.

A rank-$(r_A r_B)$ dyadic decomposition of $\boldsymbol{C} = \boldsymbol{A} * \boldsymbol{B}$ can be found quite easily:

$$\boldsymbol{C} = \left(\boldsymbol{W}_A \odot^\top \boldsymbol{W}_B\right) \left(\boldsymbol{Z}_A \odot^\top \boldsymbol{Z}_B\right)^\top = \left(\boldsymbol{W}_A \odot^\top \boldsymbol{W}_B\right) \left(\boldsymbol{Z}_A^\top \odot \boldsymbol{Z}_B^\top\right) \tag{5.1}$$

Indeed, for any $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$, it holds

$$\boldsymbol{C}(i,j) = \boldsymbol{A}(i,j)\boldsymbol{B}(i,j) = \sum_{l=1}^{r_A} \sum_{k=1}^{r_B} \boldsymbol{W}_A(i,l)\boldsymbol{Z}_A(j,l)\boldsymbol{W}_B(i,k)\boldsymbol{Z}_B(j,k)$$

$$= \sum_{l=1}^{r_A} \sum_{k=1}^{r_B} \left(\boldsymbol{W}_A \odot^\top \boldsymbol{W}_B\right)(i, k + (l-1)r_B) \left(\boldsymbol{Z}_A \odot^\top \boldsymbol{Z}_B\right)(j, k + (l-1)r_B)$$
$$= \left(\boldsymbol{W}_A \odot^\top \boldsymbol{W}_B\right)(i, :) \left(\boldsymbol{Z}_A^\top \odot \boldsymbol{Z}_B^\top\right)(:, j)$$

In order to recompress $\boldsymbol{C}$ to a lower rank, we may follow a deterministic approach, as follows:

- Compute the QR decomposition of the right factor: $\boldsymbol{Z}_A \odot^\top \boldsymbol{Z}_B = \boldsymbol{QR}$.
- Update the left factor: $\boldsymbol{W}' = \left(\boldsymbol{W}_A \odot^\top \boldsymbol{W}_B\right) \boldsymbol{R}^\top$.
- Obtain a low-rank approximation of $\boldsymbol{W}'$ with the desired rank or precision (e.g. through a RRQR decomposition):

$$\boldsymbol{W}' \approx \widetilde{\boldsymbol{W}} \widetilde{\boldsymbol{Z}}^\top$$

- Define the new low-rank factors of $\boldsymbol{C}$ as $\boldsymbol{W}^* = \widetilde{\boldsymbol{W}}$ and $\boldsymbol{Z}^* = \boldsymbol{Q}\widetilde{\boldsymbol{Z}}$

Instead, a randomized approach can obtain a low-rank approximation of $\boldsymbol{C}$ without computing the full factors $\boldsymbol{W}_A \odot^\top \boldsymbol{W}_B$ and $\boldsymbol{Z}_A \odot^\top \boldsymbol{Z}_B$. To this aim, we modify the algorithms described in Section 3.1 to exploit the particular structure of $\boldsymbol{C}$: indeed, we can compute matrix-vector products involving $\boldsymbol{C}$ in $\mathcal{O}(mr_Ar_B)$ flops, using only $\mathcal{O}(mr_B + nr_A)$ extra memory, as follows.

For $i \in \{1, \dots, m\}$, it holds that

$$(\boldsymbol{C}\mathbf{x})(i) = \sum_{j=1}^{n} \boldsymbol{A}(i,j)\boldsymbol{B}(i,j)\mathbf{x}(j) =$$
$$= \sum_{j,k=1}^{n} \boldsymbol{A}(i,j)\boldsymbol{B}(i,k)\mathbf{x}(j)\boldsymbol{I}_n(j,k) = \left(\boldsymbol{A}\,\mathtt{diag}(\mathbf{x})\boldsymbol{B}^\top\right)(i,i)$$

which implies that

$$\boldsymbol{C}\mathbf{x} = \mathtt{diag}\left(\boldsymbol{A}\mathtt{diag}(\mathbf{x})\boldsymbol{B}^\top\right) = \mathtt{diag}\left(\boldsymbol{W}_A\boldsymbol{Z}_A^\top\mathtt{diag}(\mathbf{x})\boldsymbol{Z}_B\boldsymbol{W}_B^\top\right) \tag{5.2}$$

Thus, we can proceed with the computation from inside out:

- $\boldsymbol{Y}_1 = \mathtt{diag}(\mathbf{x})\boldsymbol{Z}_B$ can be computed in just $\mathcal{O}(nr_B)$ flops due to the diagonal structure of the first factor. Indeed, it suffices to evaluate each row as

$$\boldsymbol{Y}_1(i,:) = \mathbf{x}(i)\boldsymbol{Z}_B(i,:) \quad \text{for } i \in \{1, \dots, n\}$$

- $\boldsymbol{Y}_2 = \boldsymbol{W}_A\left(\boldsymbol{Z}_A^\top\boldsymbol{Y}_1\right)$ can be computed in $\mathcal{O}(mr_Ar_B)$ flops.
- Since we just need the diagonal terms of $\boldsymbol{Y}_2\boldsymbol{W}_B^\top$, we do not compute the full matrix. Instead, we can just obtain each component as

$$(\boldsymbol{C}\mathbf{x})(i) = \sum_{j=1}^{r_B} \boldsymbol{Y}_2(i,j)\boldsymbol{W}_B(i,j) \quad \text{for } i \in \{1, \dots, m\}$$

using $\mathcal{O}(mr_B)$ flops overall.

Similarly, the computation of matrix-vector products involving $\boldsymbol{C}^\top$ can be carried out as:

$$\boldsymbol{C}^\top \mathbf{y} = \mathtt{diag}\left(\boldsymbol{A}^\top \mathtt{diag}(\mathbf{y}) \boldsymbol{B}\right) = \mathtt{diag}\left(\boldsymbol{Z}_A \boldsymbol{W}_A^\top \mathtt{diag}(\mathbf{y}) \boldsymbol{W}_B \boldsymbol{Z}_B^\top\right) \tag{5.3}$$

Again, this requires only $\mathcal{O}(mr_Ar_B)$ flops, and $\mathcal{O}(mr_B + nr_A)$ extra memory.

Using (5.2) and (5.3), we can specialize Algorithms 1 and 3 to tackle the case of Hadamard products. We report the extension of Algorithm 3 in Algorithm 13. The modified version of Algorithm 1 can be obtained in the same spirit.

Algorithm 13: Randomized compression with fixed (Frobenius) precision

Data: matrices $\boldsymbol{W}_A \in \mathbb{R}^{m\times r_A}$, $\boldsymbol{Z}_A \in \mathbb{R}^{n\times r_A}$, $\boldsymbol{W}_B \in \mathbb{R}^{m\times r_B}$, and $\boldsymbol{Z}_B \in \mathbb{R}^{n\times r_B}$, $\varepsilon > 0$, integer $q > 0$

Result: $\boldsymbol{W} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{Z} \in \mathbb{R}^{n\times r}$, with $\boldsymbol{W}^\top\boldsymbol{W} = \boldsymbol{I}_m$ and such that $\|\left(\boldsymbol{W}_A\boldsymbol{Z}_A^\top\right) * \left(\boldsymbol{W}_B\boldsymbol{Z}_B^\top\right) - \boldsymbol{W}\boldsymbol{Z}^\top\|_F \leqslant \varepsilon \|\left(\boldsymbol{W}_A\boldsymbol{Z}_A^\top\right) * \left(\boldsymbol{W}_B\boldsymbol{Z}_B^\top\right)\|_F$ holds with high probability

Generate Gaussian matrix $\boldsymbol{\Omega} \in \mathbb{R}^{n\times q}$

$\boldsymbol{Y} := \left(\left(\boldsymbol{W}_A\boldsymbol{Z}_A^\top\right) * \left(\boldsymbol{W}_B\boldsymbol{Z}_B^\top\right)\right)\boldsymbol{\Omega}$ % column-wise using (5.2)

Compute QR decomposition: $\boldsymbol{Y} =: \boldsymbol{W}\boldsymbol{R}$

$\boldsymbol{Z} := \left(\left(\boldsymbol{Z}_A\boldsymbol{W}_A^\top\right) * \left(\boldsymbol{Z}_B\boldsymbol{W}_B^\top\right)\right)\boldsymbol{W}$ % column-wise using (5.3)

$r := 0$

while $\|\boldsymbol{Z}\|_F^2 - \|\boldsymbol{Z}(:, 1:r)\|_F^2 > \varepsilon^2\|\boldsymbol{Z}\|_F^2$ **do**

  $r := r + 1$

  Generate Gaussian vector $\boldsymbol{\omega} \in \mathbb{R}^n$

  $\boldsymbol{y} := \left(\left(\boldsymbol{W}_A\boldsymbol{Z}_A^\top\right) * \left(\boldsymbol{W}_B\boldsymbol{Z}_B^\top\right)\right)\boldsymbol{\omega}$ % using (5.2)

  $\boldsymbol{w} := \left(\boldsymbol{I}_m - \boldsymbol{W}\boldsymbol{W}^\top\right)\boldsymbol{y}$

  $\boldsymbol{w} := \boldsymbol{w}/\|\boldsymbol{w}\|$

  $\boldsymbol{W} := [\boldsymbol{W} \,|\, \boldsymbol{w}]$

  $\boldsymbol{z} := \left(\left(\boldsymbol{Z}_A\boldsymbol{W}_A^\top\right) * \left(\boldsymbol{Z}_B\boldsymbol{W}_B^\top\right)\right)\boldsymbol{w}$ % using (5.3)

  $\boldsymbol{Z} := [\boldsymbol{Z} \,|\, \boldsymbol{z}]$

end

The complexity of Algorithm 13 is $\mathcal{O}(mr_Ar_B(r+q))$, with the dominant steps being the matrix-vector products carried out by exploiting (5.2) and (5.3).

Moreover, the total memory required throughout the execution of the algorithm is just $\mathcal{O}(mr_B + nr_A + m(r+q))$, due to (5.2) and (5.3), and to the storage of the final factors. Under the reasonable (for ε small enough) assumption that $r \geqslant r_A$, it can be simplified as $\mathcal{O}(m(r+q))$.

Similarly, the extension of the fixed-rank case requires $\mathcal{O}(mr_Ar_B(r+p))$ flops, with p being the oversampling parameter, and $\mathcal{O}(m(r+p))$ memory.

In comparison, the deterministic approach described above costs approximately $\mathcal{O}\left(mr_A^2r_B^2\right)$ flops, and needs $\mathcal{O}(mr_Ar_B)$ memory.

5.2 Compression of Hadamard products of tensors in the TT format

Consider two tensors of size $n_1 \times \ldots \times n_d$ in the TT format:

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) = \boldsymbol{\mathcal{A}}_1(i_1) \ldots \boldsymbol{\mathcal{A}}_d(i_d) \quad \text{and} \quad \boldsymbol{\mathcal{B}}(i_1, \ldots, i_d) = \boldsymbol{\mathcal{B}}_1(i_1) \ldots \boldsymbol{\mathcal{B}}_d(i_d)$$

for $(i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d$, with TT-ranks $(1, r_1^A, \ldots, r_{d-1}^A, 1)$ and $(1, r_1^B, \ldots, r_{d-1}^B, 1)$ respectively.

Using the same reasoning applied in the previous section, we can easily verify (see e.g. [Ose11]) that the tensor $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$ admits the following TT representation with TT-ranks $(1, r_1^A r_1^B, \ldots, r_{d-1}^A r_{d-1}^B, 1)$:

$$\boldsymbol{\mathcal{C}}(i_1, \ldots, i_d) = \underbrace{\left(\boldsymbol{\mathcal{A}}_1(i_1) \otimes \boldsymbol{\mathcal{B}}_1(i_1)\right)}_{\boldsymbol{\mathcal{C}}_1(i_1)} \ldots \underbrace{\left(\boldsymbol{\mathcal{A}}_d(i_d) \otimes \boldsymbol{\mathcal{B}}_d(i_d)\right)}_{\boldsymbol{\mathcal{C}}_d(i_d)} \tag{5.4}$$

for $(i_1, \ldots, i_d) \in I_1 \times \ldots \times I_d$.

As before, we wish to recompress $\boldsymbol{\mathcal{C}}$ down to some prescribed rank or precision. One way to do this is to explicitly compute each of the $(r_{j-1}^A r_{j-1}^B) \times n_j \times (r_j^A r_j^B)$ cores appearing in (5.4) (for $j = 1, \ldots, d$), and then apply Algorithm 7 or 8, depending on the kind of compression we wish to achieve.

In particular, it is important to observe the following property: even if the TT decompositions of both $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ are (either left- or right-) orthogonal, in general the TT decomposition (5.4) is not.

Indeed, assume that both $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ have left-orthogonal TT decompositions. For any $j \in \{1, \ldots, d\}$ it holds

$$\begin{aligned}
\left(\boldsymbol{\mathcal{C}}_j^{<2>}\right)^\top \boldsymbol{\mathcal{C}}_j^{<2>} &= \sum_{i_j=1}^{n_j} \boldsymbol{\mathcal{C}}_j(i_j)^\top \boldsymbol{\mathcal{C}}_j(i_j) \\
&= \sum_{i_j=1}^{n_j} \left(\boldsymbol{\mathcal{A}}_j(i_j) \otimes \boldsymbol{\mathcal{B}}_j(i_j)\right)^\top \left(\boldsymbol{\mathcal{A}}_j(i_j) \otimes \boldsymbol{\mathcal{B}}_j(i_j)\right) \\
&= \sum_{i_j=1}^{n_j} \left(\boldsymbol{\mathcal{A}}_j(i_j)^\top \boldsymbol{\mathcal{A}}_j(i_j)\right) \otimes \left(\boldsymbol{\mathcal{B}}_j(i_j)^\top \boldsymbol{\mathcal{B}}_j(i_j)\right)
\end{aligned}$$

which, in general, is different from

$$\begin{aligned}
&\sum_{i_j=1}^{n_j} \sum_{i'_j=1}^{n_j} \left(\boldsymbol{\mathcal{A}}_j(i_j)^\top \boldsymbol{\mathcal{A}}_j(i_j)\right) \otimes \left(\boldsymbol{\mathcal{B}}_j(i'_j)^\top \boldsymbol{\mathcal{B}}_j(i'_j)\right) = \\
&\qquad = \left(\sum_{i_j=1}^{n_j} \boldsymbol{\mathcal{A}}_j(i_j)^\top \boldsymbol{\mathcal{A}}_j(i_j)\right) \otimes \left(\sum_{i'_j=1}^{n_j} \boldsymbol{\mathcal{B}}_j(i'_j)^\top \boldsymbol{\mathcal{B}}_j(i'_j)\right) = \\
&\qquad\quad = \left(\left(\boldsymbol{\mathcal{A}}_j^{<2>}\right)^\top \boldsymbol{\mathcal{A}}_j^{<2>}\right) \otimes \left(\left(\boldsymbol{\mathcal{B}}_j^{<2>}\right)^\top \boldsymbol{\mathcal{B}}_j^{<2>}\right) = \boldsymbol{I}_{r_j^A} \otimes \boldsymbol{I}_{r_j^B} = \boldsymbol{I}_{r_j^A r_j^B}
\end{aligned}$$

As such, it is impossible to avoid the preliminary orthogonalization step in Algorithms 7 and 8. This leads us to conclude that the deterministic approach is quite

expensive: assuming that

$$r_1^A = \ldots = r_{d-1}^A = r_1^B = \ldots = r_{d-1}^B = r \quad \text{and} \quad n_1 = \ldots = n_d = n \tag{5.5}$$

the procedure costs $\mathcal{O}\left(dnr^6\right)$ flops (due to the orthogonalization step), and needs $\mathcal{O}\left(dnr^4\right)$ memory (to store $\boldsymbol{\mathcal{C}}$ before compressing it), independently of the final TT-ranks.

As in the matrix case, it is possible to improve both complexity and memory requirements using a randomized approach.

5.2.1 Sampling the range of the unfoldings of Hadamard products

First, we consider the problem of finding a low-rank approximation for the j^{th} unfolding of $\boldsymbol{\mathcal{C}}$, with $j \in \{1, \ldots, d-1\}$. As in Section 4.3, we solve this task by sampling the range of the unfolding with some random vectors:

$$\mathbf{y}^{(\alpha)} = \boldsymbol{\mathcal{C}}^{<j>}\boldsymbol{\omega}^{(\alpha)} \quad \text{with } \boldsymbol{\omega}^{(\alpha)} \in \mathbb{R}^{(n_{j+1}\cdots n_d)}, \text{ for } \alpha \in \{1, 2, \ldots\}$$

In particular, we have seen that, if we choose "rank-1" random vectors of the form

$$\boldsymbol{\omega}^{(\alpha)} = \boldsymbol{\omega}_d^{(\alpha)} \otimes \ldots \otimes \boldsymbol{\omega}_{j+1}^{(\alpha)}, \quad \text{with } \boldsymbol{\omega}_l^{(\alpha)} \in \mathbb{R}^{n_l} \text{ for } l \in \{j+1, \ldots, d\},$$

then we can use multidimensional contractions to compute efficiently the samples. In addition, in this case we can exploit the structure of $\boldsymbol{\mathcal{C}}$ to speed up the process even further, and avoid storing the uncompressed cores $\boldsymbol{\mathcal{C}}_l$, for $l \in \{j+1, \ldots, d\}$.

Indeed, let $l \in \{j+1, \ldots, d\}$ and consider lines 6–8 in Algorithm 10: in the case at hand, we want to compute

$$\boldsymbol{y}_l^{(\alpha)} = \boldsymbol{\mathcal{C}}_{\geqslant l}^{\top}\left(\boldsymbol{\omega}_d^{(\alpha)} \otimes \ldots \otimes \boldsymbol{\omega}_l^{(\alpha)}\right) = \sum_{i_l=1}^{n_l}\left(\boldsymbol{\mathcal{C}}_l(i_l)\boldsymbol{y}_{l+1}^{(\alpha)}\right)\boldsymbol{\omega}_l^{(\alpha)}(i_l)$$

with $\boldsymbol{y}_{l+1}^{(\alpha)} \in \mathbb{R}^{(r_l^A r_l^B)}$ being either a previously computed vector, or 1 if $l = d$. Due to (5.4), this can be expressed as

$$\boldsymbol{y}_l^{(\alpha)} = \sum_{i_l=1}^{n_l}\left(\left(\boldsymbol{\mathcal{A}}_l(i_l) \otimes \boldsymbol{\mathcal{B}}_l(i_l)\right)\boldsymbol{y}_{l+1}^{(\alpha)}\right)\boldsymbol{\omega}_l^{(\alpha)}(i_l) \tag{5.6}$$

Now, consider the matrices $\boldsymbol{Y}_l^{(\alpha)} \in \mathbb{R}^{r_{l-1}^B \times r_{l-1}^A}$ and $\boldsymbol{Y}_{l+1}^{(\alpha)} \in \mathbb{R}^{r_l^B \times r_l^A}$, obtained by reshaping the vectors $\boldsymbol{y}_l^{(\alpha)}$ and $\boldsymbol{y}_{l+1}^{(\alpha)}$ respectively, so that

$$\left(\boldsymbol{Y}_l^{(\alpha)}\right)^{<2>} = \boldsymbol{y}_l^{(\alpha)} \quad \text{and} \quad \left(\boldsymbol{Y}_{l+1}^{(\alpha)}\right)^{<2>} = \boldsymbol{y}_{l+1}^{(\alpha)}$$

We can exploit the fact that

$$\left(\boldsymbol{A} \otimes \boldsymbol{B}\right)\left(\boldsymbol{C}\right)^{<2>} = \left(\boldsymbol{B}\boldsymbol{C}\boldsymbol{A}^{\top}\right)^{<2>} \tag{5.7}$$

which holds for any matrices $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$ of suitable size (see e.g. [Sch04]), and rewrite (5.6) as

$$\boldsymbol{Y}_l^{(\alpha)} = \sum_{i_l=1}^{n_l} \left(\boldsymbol{\mathcal{B}}_l(i_l) \boldsymbol{Y}_{l+1}^{(\alpha)} \boldsymbol{\mathcal{A}}_l(i_l)^\top \right) \boldsymbol{\omega}_l^{(\alpha)}(i_l) \tag{5.8}$$

In this way we can compute the new sample in $\mathcal{O}\left(n_l r_{l-1}^A r_l^B (r_l^A + r_{l-1}^B)\right)$ flops at most.

Hence, Algorithm 10 can be efficiently applied to the case of Hadamard products. If we employ assumption (5.5), the total cost of generating a single set of samples (i.e. running the algorithm with $k = 1$) is $\mathcal{O}\left(dnr^3\right)$ flops, and the memory required (excluding the storage of the tensors $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$) is just $\mathcal{O}\left(dr^2\right)$.

5.2.2 Low-rank approximation of tensors in the TT format

Let us consider the case of the truncation of the Hadamard product of two tensors to some prescribed TT-ranks $(1, r_1^*, \ldots, r_{d-1}^*, 1)$ (the rounding case will follow as a generalization of Algorithm 12).

First, we run Algorithm 10 as discussed in the previous section. In particular, we set $k = \max_{j=1,\ldots,d-1} r_j^* + p$, with $p > 0$ being the oversampling parameter. In this way, we obtain k samples from the row space of each right interface matrix $\boldsymbol{\mathcal{C}}_{\geqslant j}$, for $j \in \{2, \ldots, d\}$. As before, we reshape the samples as matrices:

$$\boldsymbol{Y}_j^{(l)} \in \mathbb{R}^{r_{j-1}^B \times r_{j-1}^A}, \quad \text{with } \left(\boldsymbol{Y}_j^{(l)}\right)^{<2>} = \boldsymbol{\mathcal{C}}_{\geqslant j}^\top \left(\boldsymbol{\omega}_d^{(l)} \otimes \ldots \otimes \boldsymbol{\omega}_j^{(l)}\right)$$

for $l \in \{1, \ldots, k\}$ and $j \in \{2, \ldots, d\}$.

Now we can apply a procedure similar to Algorithm 11, and iteratively compute each core of the compressed tensor, as follows. The first unfolding of $\boldsymbol{\mathcal{C}}$ is of the form

$$\boldsymbol{\mathcal{C}}^{<1>} = \boldsymbol{\mathcal{C}}_{\leqslant 1} \boldsymbol{\mathcal{C}}_{\geqslant 2}^\top = (\boldsymbol{\mathcal{C}}_1)^{<2>} \boldsymbol{\mathcal{C}}_{\geqslant 2}^\top$$

and its range can be sampled as

$$\boldsymbol{\mathcal{C}}^{<1>} \boldsymbol{\omega}^{(l)} = \boldsymbol{\mathcal{C}}^{<1>} \left(\boldsymbol{\omega}_d^{(l)} \otimes \ldots \otimes \boldsymbol{\omega}_2^{(l)}\right) = (\boldsymbol{\mathcal{C}}_1)^{<2>} \left(\boldsymbol{Y}_2^{(l)}\right)^{<2>}$$

for $l \in \{1, \ldots, r_1^* + p\}$.

Due to (5.4), we have the following representation of each sample:

$$\boldsymbol{\mathcal{C}}^{<1>} \boldsymbol{\omega}^{(l)} = \begin{bmatrix} \boldsymbol{\mathcal{A}}_1(1) \otimes \boldsymbol{\mathcal{B}}_1(1) \\ \vdots \\ \boldsymbol{\mathcal{A}}_1(n_1) \otimes \boldsymbol{\mathcal{B}}_1(n_1) \end{bmatrix} \left(\boldsymbol{Y}_2^{(l)}\right)^{<2>} = \begin{bmatrix} \left(\boldsymbol{\mathcal{A}}_1(1) \otimes \boldsymbol{\mathcal{B}}_1(1)\right) \left(\boldsymbol{Y}_2^{(l)}\right)^{<2>} \\ \vdots \\ \left(\boldsymbol{\mathcal{A}}_1(n_1) \otimes \boldsymbol{\mathcal{B}}_1(n_1)\right) \left(\boldsymbol{Y}_2^{(l)}\right)^{<2>} \end{bmatrix}$$

Using (5.7), this can be expressed as

$$\boldsymbol{\mathcal{C}}^{<1>} \boldsymbol{\omega}^{(l)} = \begin{bmatrix} \left(\boldsymbol{\mathcal{B}}_1(1) \boldsymbol{Y}_2^{(l)} \boldsymbol{\mathcal{A}}_1(1)^\top\right)^{<2>} \\ \vdots \\ \left(\boldsymbol{\mathcal{B}}_1(n_1) \boldsymbol{Y}_2^{(l)} \boldsymbol{\mathcal{A}}_1(n_1)^\top\right)^{<2>} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{B}}_1(1) \boldsymbol{Y}_2^{(l)} \boldsymbol{\mathcal{A}}_1(1)^\top \\ \vdots \\ \boldsymbol{\mathcal{B}}_1(n_1) \boldsymbol{Y}_2^{(l)} \boldsymbol{\mathcal{A}}_1(n_1)^\top \end{bmatrix} \tag{5.9}$$

where the "second unfolding" operator has been omitted since it acts on a scalar.

We can merge all the samples into the $n_1 \times (r_1^* + p)$ matrix

$$\boldsymbol{\mathcal{C}}^{<1>} \left[\boldsymbol{\omega}^{(1)} \,\middle|\, \dots \,\middle|\, \boldsymbol{\omega}^{(r_1^*+p)} \right]$$

and compute its QR decomposition. If we indicate with $\boldsymbol{W}_1$ the orthogonal factor of such a decomposition, we have the following rank-$(r_1^* + p)$ approximation for the first unfolding of $\boldsymbol{\mathcal{C}}$:

$$\boldsymbol{\mathcal{C}}^{<1>} \approx \boldsymbol{W}_1 \left(\boldsymbol{W}_1^\top \left(\boldsymbol{\mathcal{C}}_1\right)^{<2>} \boldsymbol{\mathcal{C}}_{\geqslant 2}^\top \right)$$

Now we can reshape $\boldsymbol{W}_1$ to obtain the first core of the compressed tensor:

$$\boldsymbol{\mathcal{C}}'_1 \in \mathbb{R}^{1 \times n_1 \times (r_1^*+p)} \quad \text{with} \quad \left(\boldsymbol{\mathcal{C}}'_1\right)^{<2>} = \boldsymbol{W}_1$$

Moreover, it is useful to define the $(r_1^A r_1^B) \times (r_1^* + p)$ "local interface matrix"

$$\boldsymbol{Z}_1 = \left(\boldsymbol{\mathcal{C}}_1\right)^{<2>\,\top} \boldsymbol{W}_1$$

which can be expressed as

$$\begin{aligned} \boldsymbol{Z}_1 &= \left[\left(\boldsymbol{\mathcal{A}}_1(1) \otimes \boldsymbol{\mathcal{B}}_1(1)\right)^\top \,\middle|\, \dots \,\middle|\, \left(\boldsymbol{\mathcal{A}}_1(n_1) \otimes \boldsymbol{\mathcal{B}}_1(n_1)\right)^\top \right] \boldsymbol{W}_1 \\ &= \sum_{i_1=1}^{n_1} \left(\boldsymbol{\mathcal{A}}_1(i_1)^\top \otimes \boldsymbol{\mathcal{B}}_1(i_1)^\top\right) \underbrace{\boldsymbol{W}_1(i_1, :)}_{\text{as a } 1 \times (r_1^*+p) \text{ matrix}} \end{aligned}$$

With these definitions, the second unfolding can be approximated as

$$\begin{aligned} \boldsymbol{\mathcal{C}}^{<2>} = \boldsymbol{\mathcal{C}}_{\leqslant 2} \boldsymbol{\mathcal{C}}_{\geqslant 3}^\top &= \left(\boldsymbol{I}_{n_2} \otimes \left(\boldsymbol{\mathcal{C}}_1\right)^{<2>} \right) \left(\boldsymbol{\mathcal{C}}_2\right)^{<2>} \boldsymbol{\mathcal{C}}_{\geqslant 3}^\top \\ &\approx \left(\boldsymbol{I}_{n_2} \otimes \left(\boldsymbol{W}_1 \boldsymbol{Z}_1^\top \right) \right) \left(\boldsymbol{\mathcal{C}}_2\right)^{<2>} \boldsymbol{\mathcal{C}}_{\geqslant 3}^\top \\ &= \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1 \right) \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{Z}_1^\top \right) \left(\boldsymbol{\mathcal{C}}_2\right)^{<2>} \boldsymbol{\mathcal{C}}_{\geqslant 3}^\top \end{aligned}$$

Since $\boldsymbol{W}_1$ has orthogonal columns, $\boldsymbol{I}_{n_2} \otimes \boldsymbol{W}_1$ does as well, and it suffices to approximate the range of

$$\left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{Z}_1^\top \right) \left(\boldsymbol{\mathcal{C}}_2\right)^{<2>} \boldsymbol{\mathcal{C}}_{\geqslant 3}^\top$$

which can be sampled as

$$\left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{Z}_1^\top \right) \left(\boldsymbol{\mathcal{C}}_2\right)^{<2>} \boldsymbol{\mathcal{C}}_{\geqslant 3}^\top \boldsymbol{\omega}^{(l)} = \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{Z}_1^\top \right) \left(\boldsymbol{\mathcal{C}}_2\right)^{<2>} \left(\boldsymbol{Y}_3^{(l)} \right)^{<2>}$$

for $l \in \{1, \dots, r_2^* + p\}$.

Expression (5.9) can be extended to this case as well, yielding the block representation (of size $n_2(r_1^* + p)$)

$$\left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{Z}_1^\top \right) \left(\boldsymbol{\mathcal{C}}_2\right)^{<2>} \boldsymbol{\mathcal{C}}_{\geqslant 3}^\top \boldsymbol{\omega}^{(l)} = \begin{bmatrix} \boldsymbol{Z}_1^\top \left(\boldsymbol{\mathcal{B}}_2(1) \boldsymbol{Y}_3^{(l)} \boldsymbol{\mathcal{A}}_2(1)^\top \right)^{<2>} \\ \vdots \\ \boldsymbol{Z}_1^\top \left(\boldsymbol{\mathcal{B}}_2(n_2) \boldsymbol{Y}_3^{(l)} \boldsymbol{\mathcal{A}}_2(n_2)^\top \right)^{<2>} \end{bmatrix} \tag{5.10}$$

for $l \in \{1, \dots, r_2^* + p\}$.

After repeating this operation for all the samples, we build a $(n_2(r_1^*+p)) \times (r_2^*+p)$ matrix, whose QR decomposition yields the orthogonal factor $\boldsymbol{W}_2$. By reshaping this factor we obtain the second core of the compressed tensor

$$\boldsymbol{\mathcal{C}}'_2 \in \mathbb{R}^{(r_1^*+p)\times n_2 \times (r_2^*+p)} \quad \text{with} \quad \left(\boldsymbol{\mathcal{C}}'_2\right)^{<2>} = \boldsymbol{W}_2$$

and we also compute the second "local interface matrix"

$$\boldsymbol{Z}_2 = \left(\boldsymbol{\mathcal{C}}_2\right)^{<2>\top} \left(\boldsymbol{I}_{n_2} \otimes \boldsymbol{Z}_1\right) \boldsymbol{W}_2 \in \mathbb{R}^{(r_2^A r_2^B)\times (r_2^*+p)}$$

which, by generalizing (5.10), can be expressed as

$$\begin{aligned} \boldsymbol{Z}_2 &= \left[\left(\boldsymbol{\mathcal{A}}_2(1) \otimes \boldsymbol{\mathcal{B}}_1(1)\right)^\top \boldsymbol{Z}_1 \,\middle|\, \dots \,\middle|\, \left(\boldsymbol{\mathcal{A}}_2(n_2) \otimes \boldsymbol{\mathcal{B}}_2(n_2)\right)^\top \boldsymbol{Z}_1 \right] \boldsymbol{W}_2 \\ &= \sum_{i_2=1}^{n_2} \left(\boldsymbol{\mathcal{A}}_2(i_2)^\top \otimes \boldsymbol{\mathcal{B}}_2(i_2)^\top\right) \boldsymbol{Z}_1 \boldsymbol{W}_2^{(i_2)} \end{aligned} \tag{5.11}$$

with $\boldsymbol{W}_2^{(i_2)} = \boldsymbol{W}_2\left((i_2-1)(r_1^*+p) + (1:(r_1^*+p)), :\right)$ being a $(r_1^*+p) \times (r_2^*+p)$ block of $\boldsymbol{W}_2$:

$$\boldsymbol{W}_2 = \left[\begin{array}{c} \boldsymbol{W}_2^{(1)} \\ \hline \dots \\ \hline \boldsymbol{W}_2^{(n_2)} \end{array}\right]$$

Then we proceed iteratively until we compute the $(d-1)^{\text{th}}$ core $\boldsymbol{\mathcal{C}}'_{d-1}$ and the last "local interface matrix"

$$\boldsymbol{Z}_{d-1} = \left(\boldsymbol{\mathcal{C}}_{d-1}\right)^{<2>\top} \left(\boldsymbol{I}_{n_{d-1}} \otimes \boldsymbol{Z}_{d-2}\right) \boldsymbol{W}_{d-1} \in \mathbb{R}^{(r_{d-1}^A r_{d-1}^B)\times (r_{d-1}^*+p)}$$

which we use to compute the last core of the compressed tensor:

$$\boldsymbol{\mathcal{C}}'_d = \boldsymbol{\mathcal{C}}_d \times_1 \boldsymbol{Z}_{d-1}^\top \in \mathbb{R}^{(r_{d-1}^*+p)\times n_d \times 1}$$

Since the tensor $\boldsymbol{\mathcal{C}}'$ has TT-ranks $(1, r_1^*+p, \dots, r_{d-1}^*+p, 1)$, which are larger than the desired ones, we carry out a deterministic compression step by applying Algorithm 7. The complete procedure is summarized in Algorithm 14.

To analyze the complexity of the algorithm, it is necessary to identify the cost of lines 6 and 10. In particular, to simplify the notation, we assume that $r_1^* = \dots = r_{d-1}^* = r^*$.

We start from the former step, by considering the block decomposition introduced in (5.10). Under assumption (5.5), we can easily see that the computation of each of the n blocks requires $\mathcal{O}\left(r^3 + r^2(r^*+p)\right)$ flops.

Similarly, the cost of line 10 can be found by referring to (5.11), and considering each term of the sum. First, we look at the product $(\boldsymbol{\mathcal{A}}_j(i_j)^\top \otimes \boldsymbol{\mathcal{B}}_j(i_j)^\top)\boldsymbol{Z}_{j-1}$. If we compute the product column-wise, we can apply (5.7) to carry out the whole multiplication in $\mathcal{O}\left(r^3(r^*+p)\right)$ flops. Then we need to compute the product between the result and one of the blocks of $\boldsymbol{W}_j$, which can be done in $\mathcal{O}\left(r^2(r^*+p)^2\right)$ flops.

All the other operations appearing in Algorithm 14 are cheaper, or cost a similar number of flops. In particular, line 2 requires $\mathcal{O}\left(dnr^3(r^*+p)\right)$ flops (see Section 5.2.1),

Algorithm 14: Randomized truncation of the Hadamard product of two tensors in the TT format

Data: tensors $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_d$ and $\boldsymbol{\mathcal{B}}_1, \ldots, \boldsymbol{\mathcal{B}}_d$ respectively, desired TT-ranks $(r_0^* = 1, r_1^*, \ldots, r_{d-1}^*, r_d^* = 1)$, integer $p \geqslant 0$

Result: $\boldsymbol{\mathcal{C}}^* \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{C}}_1^*, \ldots, \boldsymbol{\mathcal{C}}_d^*$ with the specified TT-ranks, with $\boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}} \approx \boldsymbol{\mathcal{C}}^*$

$k := \max\{r_1^*, \ldots, r_{d-1}^*\} + p$

$\{\boldsymbol{Y}_2^{(1)}, \ldots, \boldsymbol{Y}_2^{(k)}, \boldsymbol{Y}_3^{(1)}, \ldots, \boldsymbol{Y}_d^{(k)}\} :=$ Algorithm 10$(\boldsymbol{\mathcal{C}}, k)$ % using (5.8)

$\boldsymbol{Z}_0 := 1$

for $j = 1, 2, \ldots, d-1$ **do**

  for $l = 1, 2, \ldots, r_j^* + p$ **do**

    $\boldsymbol{y}_j^{(l)} := \left(\boldsymbol{I}_{n_j} \otimes \boldsymbol{Z}_{j-1}^\top\right) (\boldsymbol{\mathcal{C}}_j)^{<2>} \left(\boldsymbol{Y}_{j-1}^{(l)}\right)^{<2>}$ % using (5.10)

  end

  Compute QR decomposition: $\left[\boldsymbol{y}_j^{(1)} \,\middle|\, \ldots \,\middle|\, \boldsymbol{y}_j^{(r_j^*+p)}\right] =: \boldsymbol{W}_j \boldsymbol{R}_j$

  $\boldsymbol{\mathcal{C}}'_j := \texttt{reshape}(\boldsymbol{W}_j, [r_{j-1}^* + p, n_j, r_j^* + p])$

  $\boldsymbol{Z}_j := (\boldsymbol{\mathcal{C}}_j)^{<2>\top} \left(\boldsymbol{I}_{n_j} \otimes \boldsymbol{Z}_{j-1}\right) \boldsymbol{W}_j$ % using (5.11)

end

$\boldsymbol{\mathcal{C}}'_d := \boldsymbol{\mathcal{C}}_d \times_1 \boldsymbol{Z}_{d-1}^\top$

$\boldsymbol{\mathcal{C}}^* :=$ Algorithm 7$(\boldsymbol{\mathcal{C}}', (1, r_1^*, \ldots, r_{d-1}^*, 1))$ % from right to left

and both the QR decompositions and the recompression step take $\mathcal{O}\left(dn(r^* + p)^3\right)$ flops overall.

In summary, the total complexity of the algorithm is

$$\mathcal{O}\left(dnr^3(r^* + p) + dnr^2(r^* + p)^2 + dn(r^* + p)^3\right)$$

Since $r^* + p \leqslant r^2$, we can simplify this expression to

$$\mathcal{O}\left(dnr^3(r^* + p) + dnr^2(r^* + p)^2\right)$$

Moreover, we may introduce the additional assumption that $r \leqslant r^* + p$ (i.e. that we do not wish to compress $\boldsymbol{\mathcal{C}}$ to TT-ranks which are lower than the ones of the two factors $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$) to conclude that the final cost is $\mathcal{O}\left(dnr^2(r^* + p)^2\right)$ flops.

Besides being cheaper than the deterministic approach (which has been analyzed in Section 5.2), Algorithm 14 also has lower memory requirements: indeed, just

$$\mathcal{O}\left(dr^2(r^* + p) + dn(r^* + p)^2\right)$$

memory is needed.

Given this procedure, we can easily tackle the rounding problem by modifying Algorithm 12: it suffices to replace the call to Algorithm 11 with one to Algorithm 14. This leads to Algorithm 15, where the integers m and q represent respectively the sampling step and the recompression gap (see Section 4.3.3).

A complexity analysis can be carried out similarly to the one of the original algorithm. For ease of notation we assume that both the initial and the final TT-ranks

Algorithm 15: Randomized rounding of the Hadamard product of two tensors in the TT format

Data: tensors $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{A}}_1, \ldots, \boldsymbol{\mathcal{A}}_d$ and $\boldsymbol{\mathcal{B}}_1, \ldots, \boldsymbol{\mathcal{B}}_d$ respectively, $\varepsilon > 0$, guess for the TT-ranks $(1, r_1^{(0)}, \ldots, r_{d-1}^{(0)}, 1)$, positive integers q and m

Result: $\boldsymbol{\mathcal{C}}^* \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ in the TT format with cores $\boldsymbol{\mathcal{C}}_1^*, \ldots, \boldsymbol{\mathcal{C}}_d^*$, such that $\|\boldsymbol{\mathcal{C}}^* - \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}\|_F \leqslant \varepsilon \|\boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}\|_F$ holds with high probability

```
l := 0
do
    C' := Algorithm 14(A, B, (1, r_1^(l), ..., r_{d-1}^(l), 1), 0)
    C* := Algorithm 8(C', ε)                          % from right to left
    (1, r_1^*, ..., r_{d-1}^*, 1) = TT-ranks(C*)
    changed := false
    for j = 1, ..., d − 1 do
        if r_j^(l) ⩾ r_j^* + q then
            r_j^(l+1) := r_j^(l)
        else
            r_j^(l+1) := r_j^(l) + m
            changed := true
        end
    end
    l := l + 1
while changed
```

are uniform:

$$r_1^{(0)} = \ldots = r_{d-1}^{(0)} = r^{(0)} \quad \text{and} \quad \widehat{r}_1 = \ldots = \widehat{r}_{d-1} = \widehat{r}$$

Also, we assume that $\widehat{r} = r^{(0)} + km - q$ for some integer k.

Thus, the l^{th} iteration (out of k) of Algorithm 15 takes $\mathcal{O}\big(dnr^2(r^{(0)} + lm)^2\big)$ flops because of Algorithm 14. This leads to an overall complexity of

$$\sum_{l=1}^{k} \mathcal{O}\Big(dnr^2(r^{(0)} + lm)^2\Big) = \mathcal{O}\Big(dnr^2(r^{(0)} + km)^2 k\Big) = \mathcal{O}\big(dnr^2(\widehat{r} + q)^2 k\big)$$

5.3 Numerical examples

5.3.1 Square of a 2-dimensional rational function through a Hadamard power

We consider the matrix $\boldsymbol{A}$ defined in Section 3.3.2, with $n = 10^5$ and $\gamma = 1$, and we compute its low-rank approximation with precision $\varepsilon = 10^{-8}$ in the Frobenius norm:

$$\boldsymbol{A} \approx \widetilde{\boldsymbol{A}} = \boldsymbol{W}_A \boldsymbol{Z}_A^\top$$

In particular the rank of the decomposition is $r_A = 30$.

Now we wish to compute a low-rank approximation of $\widetilde{\boldsymbol{A}} * \widetilde{\boldsymbol{A}}$ with fixed precision $\varepsilon \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. To this aim, we apply the two approaches (deterministic and randomized) described in Section 5.1. In particular, we choose $q \in \{1, \dots, 6\}$ for the randomized algorithm. Due to the smoothness of $f_{2,2}(x, y) = (x + y)^{-2}$, the singular value decay of $\boldsymbol{A} * \boldsymbol{A}$ is still (at least) exponential. Hence, we expect small values of q to be sufficient to achieve the required precision.

We run the algorithms in the environment described in Section 3.3.1, and we plot the results in Figures 5.1 and 5.2.

In Figure 5.1 we show the relative errors of the final low-rank approximation. Similarly to the results of Section 3.3.3, we can observe that q plays a role analogous to the oversampling parameter in the fixed-rank case. In particular, we can verify that small values of q (e.g. $q = 3$) are enough to achieve the prescribed precision. Also, we report that the final rank of the dyadic approximation (after the recompression step in the randomized case) is always the same in both approaches, reaching the minimal size for which the low-rank approximation problem is feasible.

In Figure 5.2 we compare the computing time of the deterministic and randomized approaches. We can see that Algorithm 13 is always much faster. In particular, since the deterministic algorithm depends only weakly on ε, the relative efficiency of the randomized approach is more pronounced for larger values of ε.

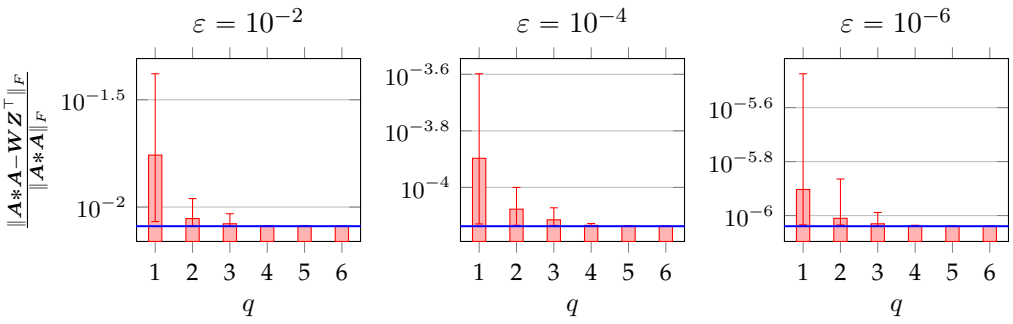

FIGURE 5.1: Relative error versus sampling parameter for Algorithm 13. The error bars indicate the range of values obtained in $N = 20$ simulations. In blue the result of the deterministic approach.

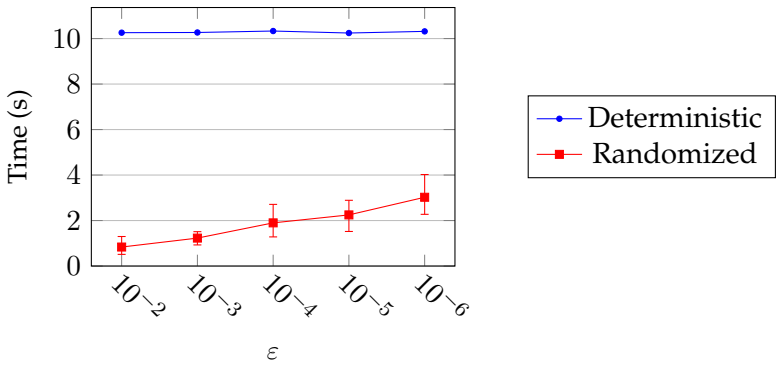

FIGURE 5.2: Computing time for Algorithm 13 and the deterministic counterpart. For the randomized algorithm, we choose a sampling parameter $q = 3$. The error bars indicate the range of values obtained in $N = 20$ simulations.

5.3.2 Hadamard product of high-dimensional rational functions

Let $\mathcal{A}$ be the tensor defined in Section 4.4.2, with $d = 15$, $n = 30$, and $\gamma = 1$. Moreover, we consider $\mathcal{B}$, the tensor obtained in the same way but with $\gamma = 1/2$. Such tensors are generated by applying an exponential sum approximation with 20 terms. With this choice, we obtain tensors with uniform TT-ranks $(1, 20, \dots, 20, 1)$, which provide approximate discretizations of

$$f_{d,1}(x_1, \dots, x_d) = (x_1 + \dots + x_d)^{-1} \quad \text{and} \quad f_{d,1/2}(i_1, \dots, i_d) = (x_1 + \dots + x_d)^{-1/2}$$

We wish to recompress the Hadamard product $\mathcal{C} = \mathcal{A} * \mathcal{B}$ down to precision $\varepsilon \in \{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}\}$, using two approaches: the first is the explicit computation of $\mathcal{C}$, followed by the execution of Algorithm 8; the second is Algorithm 15 with $m = 2$ and $q = 1$, without any initial guess for the compressed TT-ranks (i.e. we set $r_1^{(0)} = \dots = r_{d-1}^{(0)} = 0$). Such small choices for m and q are motivated by the fact that the spectra of all unfoldings of $\mathcal{C}$ are (at least) exponentially decaying, due to the regularity properties of $f_{d,3/2}$ in $\left(0, \frac{n+1}{10}\right)^d$.

Figures 5.3 and 5.4 show the results of the simulations, obtained in the environment described in Section 3.3.1. From Figure 5.3 (left) we can observe that the two approaches yield similar errors, achieving the desired accuracy in almost all simulations. However, even in case of failure, the value of the relative error is within a small factor of ε.

The computing time is shown in Figure 5.3 (right), where we can see that the randomized approach is much faster than the deterministic one. Also, it is interesting to note that, for $\varepsilon \geqslant 10^{-6}$, it is faster to run Algorithm 15 than to carry out just the preliminary computation of $\mathcal{C}$, which is required for the deterministic approach.

In Figure 5.4 we show the maximum TT-ranks at the end of the two algorithms. In particular, for the randomized approach we also show the TT-ranks before the last recompression step, i.e. the final TT-ranks of $\mathcal{C}'$ in Algorithm 15. We can observe that a small amount of oversampling (measured as the variation between the TT-ranks before and after recompression) is enough to guarantee the correctness of the randomized procedure, which (except for a single sample with $\varepsilon = 10^{-4}$) achieves the same TT-ranks as the deterministic approach.

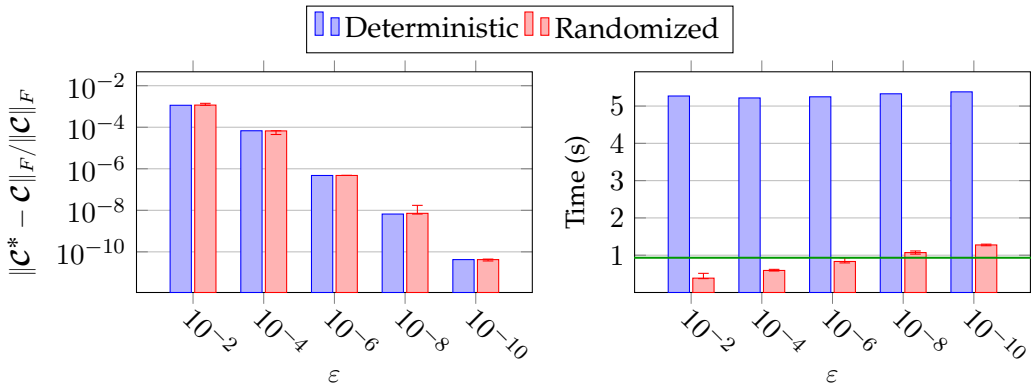

FIGURE 5.3: Relative error (left) and computing time (right) for the deterministic approach and for Algorithm 15. The bars connect the minimum and the maximum values obtained over $N = 20$ simulations. The green line in the right plot represents the time needed for the computation of $\mathcal{C}$ in the deterministic approach.

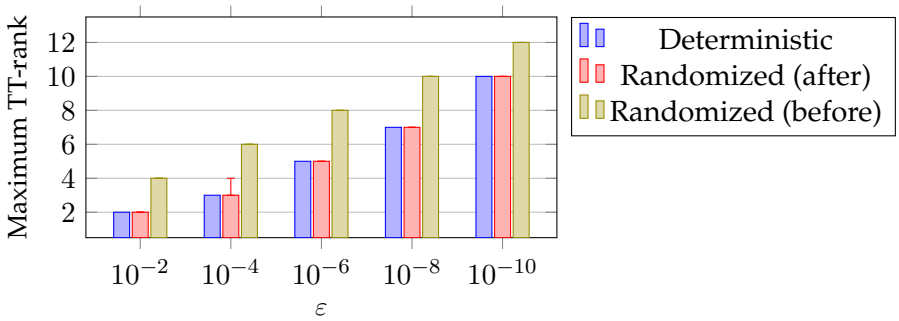

FIGURE 5.4: Maximum TT-rank of the compressed tensor for the deterministic approach and for Algorithm 15. For the randomized algorithm the plot shows the modal values of the maximal TT-ranks over $N = 20$ simulations, both before and after recompression. Also, the bars show the range of values obtained.

5.3.3 Hadamard product of random tensors

We build the random tensors $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ by following the procedure described in Section 4.4.3. In particular, we choose $n = 50$, $d \in \{10, 20, 40\}$, and $r \in \{4, 6, \dots, 40\}$. For each value of r, we choose an exponentially decaying spectrum $\sigma_j = a^{1-j}$ (for $j \in \{1, \dots, r\}$), with $a > 1$ such that σ_r is approximately equal to the machine precision, i.e. we set $a \simeq \varepsilon_m^{1/(1-r)}$.

We want to recompress $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}} * \boldsymbol{\mathcal{B}}$ down to uniform TT-ranks $(1, r, \dots, r, 1)$. To this aim, we follow several strategies:

- we compute $\boldsymbol{\mathcal{C}}$ and apply Algorithm 7,
- we compute $\boldsymbol{\mathcal{C}}$ and apply Algorithm 11 (with $p = 5$),
- we apply Algorithm 14 (with $p = 5$).

In particular, we want to compare the computing times of the three approaches. The results are shown in Figure 5.5 for the computational environment described in Section 3.3.1.

We can observe that, for all choices of d, the three approaches have similar speeds for small values of r: actually, Algorithm 14 is outperformed by the other two procedures for $r \leqslant 8$. The fact that the deterministic approach is more efficient for small tensors appears reasonable, since it relies more heavily on highly optimized built-in MATLAB® functions.

In contrast, for larger values of r, both approaches which are based on the explicit computation of $\boldsymbol{\mathcal{C}}$ perform quite poorly when compared to Algorithm 14. For instance, if we consider $d = 40$ and $r = 30$, Algorithm 14 has an average running time of 6.28 seconds, versus 35.99 seconds for Algorithm 11 (speed-up ratio: 5.73) and 205.3 seconds for Algorithm 7 (speed-up ratio: 32.7).

In particular, as in the previous example, we can identify several configurations ($r \geqslant 22$) where Algorithm 14 is faster than the computation of $\boldsymbol{\mathcal{C}}$, which represents a preliminary step in the other approaches.

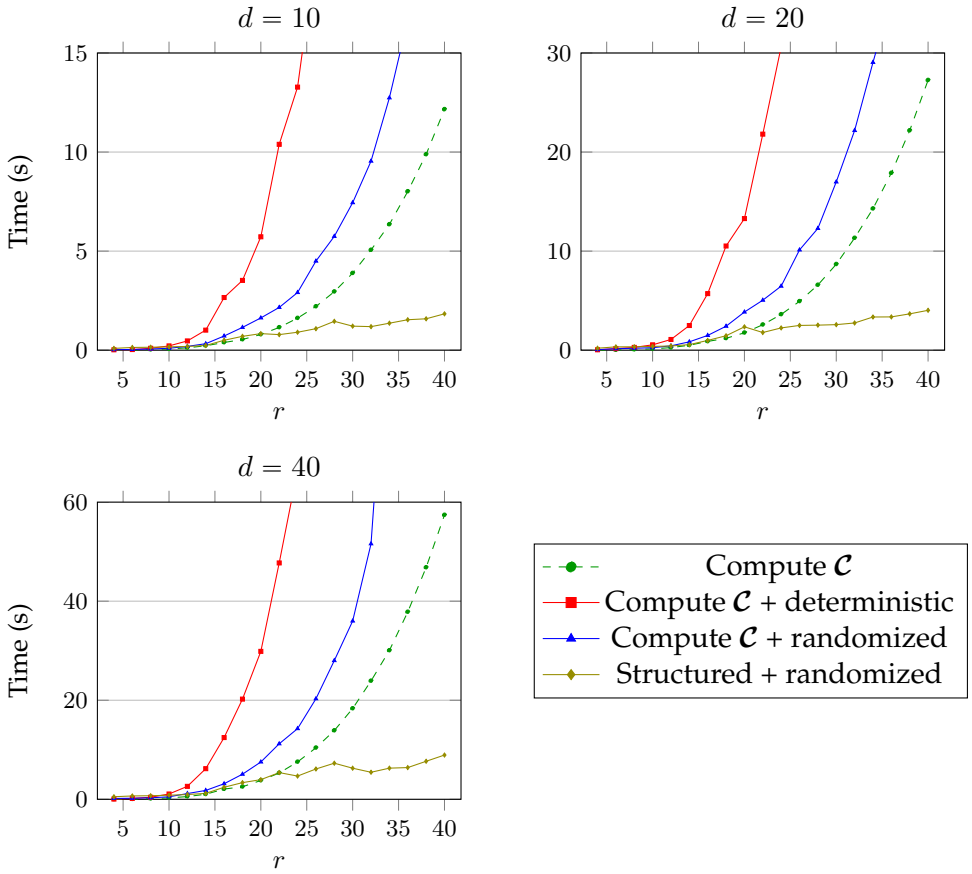

FIGURE 5.5: Computing time versus initial TT-ranks for Algorithms 7, 11 and 14. For the two randomized approaches we show the average value over $N = 20$ simulations. In green the time needed just for the computation of $\mathcal{C}$.

Chapter 6

Conclusion

We have shown that randomized approaches to low-rank matrix and tensor approximation are quite satisfactory. Indeed, at the cost of a small oversampling, they can achieve a similar accuracy as the corresponding deterministic techniques.

Moreover, they are often more efficient than any deterministic algorithm. This is mainly due to the high degree of flexibility allowed by randomness, which leads to a better exploitation of the structure of the data which needs to be compressed.

For instance, in the compression of tensors in the TT format, we have described how randomness allows to skip the orthogonalization step, which is required in any (efficient) deterministic approach. A similar effect can be observed in the case of Hadamard products, where we have derived randomized approaches which avoid the computation of the full products.

Among the several topics which have been considered in this thesis, the most striking results are the ones obtained on tensors in the TT format. Indeed, Algorithms 11 and 12 (in the standard case) and Algorithms 14 and 15 (in the Hadamard product case) represent very interesting alternatives to their deterministic counterpart, both in terms of efficiency and memory requirements.

Moreover, it is particularly impressive to observe the accuracy achieved by all algorithms for the compression of tensors in the TT format, considering that they rely on "rank-1" random vectors. For such vectors, a thorough theoretical analysis, generalizing the available results from the Gaussian case, is still missing, even though a relevant effort is being made in current research. Still, we have shown with several numerical examples that "rank-1" random vectors provide an efficient way to obtain samples from the ranges of large matrices, for which other techniques would be too expensive.

Bibliography

[Aff+87] I. Affleck et al. "Rigorous results on valence-bond ground states in antiferromagnets". In: *Phys. Rev. Lett.* 59 (7 1987), pp. 799–802. DOI: 10.1103/PhysRevLett.59.799.

[BH05] D. Braess and W. Hackbusch. "Approximation of 1/x by exponential sums in $[1, \infty)$". In: *IMA Journal of Numerical Analysis* 25.4 (2005). DOI: 10.1093/imanum/dri015.

[Bis09] C.M. Bishop. *Pattern recognition and machine learning*. Information Science and Statistics. Springer, 2009. ISBN: 978-0387-31073-2.

[BNT07] I. Babuška, F. Nobile, and R. Tempone. "A Stochastic Collocation Method for Elliptic Partial Differential Equations with Random Input Data". In: *SIAM Journal on Numerical Analysis* 45.3 (2007), pp. 1005–1034. DOI: 10.1137/050645142.

[CD05] Z. Chen and J.J. Dongarra. "Condition Numbers of Gaussian Random Matrices". In: *SIAM Journal on Matrix Analysis and Applications* 27.3 (2005), pp. 603–620. DOI: 10.1137/040616413.

[Cha87] T.F. Chan. "Rank revealing QR factorizations". In: *Linear Algebra and its Applications* 88 (1987), pp. 67–82. DOI: 10.1016/0024-3795(87)90103-0.

[Dix83] J.D. Dixon. "Estimating Extremal Eigenvalues and Condition Numbers of Matrices". In: *SIAM Journal on Numerical Analysis* 20.4 (1983), pp. 812–814. DOI: 10.1137/0720053.

[EY36] C. Eckart and G. Young. "The approximation of one matrix by another of lower rank". In: *Psychometrika* 1.3 (1936), pp. 211–218. ISSN: 1860-0980. DOI: 10.1007/BF02288367.

[FNW92] M. Fannes, B. Nachtergaele, and R.F. Werner. "Finitely correlated states on quantum spin chains". In: *Communications in Mathematical Physics* 144.3 (1992), pp. 443–490. DOI: 10.1007/BF02099178.

[GE96] M. Gu and S.C. Eisenstat. "Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization". In: *SIAM Journal on Scientific Computing* 17.4 (1996), pp. 848–869. DOI: 10.1137/0917055.

[GL17] M. Griebel and G. Li. "On the decay rate of the singular values of bivariate functions". In: 2017. URL: http://wissrech.ins.uni-bonn.de/research/pub/li/INSPreprint1702.pdf.

[GL89] G.H. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins Series in the Mathematical Sciences 3. The Johns Hopkins University Press, 1989.

[HMT11] N. Halko, P.G. Martinsson, and A.J. Tropp. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Review* 53.2 (2011), pp. 217–288. DOI: 10.1137/090771806.

[HP92] Y.P. Hong and C.-T. Pan. "Rank-Revealing QR Factorizations and the Singular Value Decomposition". In: *Mathematics of Computation* 58.197 (1992), pp. 213–232. ISSN: 00255718, 10886842.

[KB09] T.G. Kolda and B.W. Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51.3 (2009), pp. 455–500. DOI: 10.1137/07070111X.

[KP16] D. Kressner and L. Periša. *Recompression of Hadamard Products of Tensors in Tucker Format*. Tech. rep. 2016. URL: http://sma.epfl.ch/~anchpcommon/publications/ttensors_pp.pdf.

[Lib+07] E. Liberty et al. "Randomized algorithms for the low-rank approximation of matrices". In: *Proceedings of the National Academy of Sciences* 104.51 (2007), pp. 20167–20172. DOI: 10.1073/pnas.0709640104.

[Lub08] C. Lubich. *From Quantum to Classical Molecular Dynamics: Reduced Models and Numerical Analysis*. Zurich Lectures in Advanced Mathematics. European Mathematical Society Publishing House, 2008. DOI: 10.4171/067.

[McL16] W. McLean. "Exponential sum approximations for $t^{-\beta}$". In: *ArXiv e-prints* (2016). arXiv: 1606.00123.

[Mir60] L. Mirsky. "Symmetric gauge functions and unitarily invariant norms". In: *The Quarterly Journal of Mathematics* 11.1 (1960), pp. 50–59. DOI: 10.1093/qmath/11.1.50.

[Oru14] R. Orus. "A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States". In: *Annals Phys.* 349 (2014), pp. 117–158. DOI: 10.1016/j.aop.2014.06.013.

[Ose11] I.V. Oseledets. "Tensor-Train Decomposition". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317. DOI: 10.1137/090752286.

[OT09] I.V. Oseledets and E.E. Tyrtyshnikov. "Breaking the Curse of Dimensionality, Or How to Use SVD in Many Dimensions". In: *SIAM Journal on Scientific Computing* 31.5 (2009), pp. 3744–3759. DOI: 10.1137/090748330.

[Par98] B. Parlett. *The Symmetric Eigenvalue Problem*. Society for Industrial and Applied Mathematics, 1998. DOI: 10.1137/1.9781611971163.

[PP07] J. Persson and L. von Persson. "Pricing European multi-asset options using a space-time adaptive FD-method". In: *Computing and Visualization in Science* 10.4 (2007), pp. 173–183. DOI: 10.1007/s00791-007-0072-y.

[RW05] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN: 026218253X.

[Sch04] K. Schäcke. "On the Kronecker Product". MA thesis. University of Waterloo, 2004.

[Ste16] M. Steinlechner. "Riemannian Optimization for Solving High-Dimensional Problems with Low-Rank Tensor Structure". PhD thesis. EPFL, 2016. DOI: doi:10.5075/epfl-thesis-6958.

[Ven+07] O. Vendrell et al. "Full-dimensional (15-dimensional) quantum-dynamical simulation of the protonated water dimer. I. Hamiltonian setup and analysis of the ground vibrational state". In: *The Journal of Chemical Physics* 127.18 (2007), p. 184302. DOI: 10.1063/1.2787588.

[ZJD15] K. Zhong, P. Jain, and I.S. Dhillon. "Efficient Matrix Sensing Using Rank-1 Gaussian Measurements". In: *Algorithmic Learning Theory: 26th International Conference, ALT 2015, Banff, AB, Canada, October 4-6, 2015, Proceedings*. Springer International Publishing, 2015, pp. 3–18. DOI: 10.1007/978-3-319-24486-0_1.