Tensor Train Decomposition for solving high-dimensional Mutual Hazard Networks

Dissertation zur Erlangung des Doktorgrades der Naturwissenschaften (Dr. rer. nat.) der Fakultät für Physik

der Universität Regensburg

vorgelegt von

Peter Georg

aus Trostberg

im Jahr 2022

Promotionsgesuch eingereicht am: 17. Januar 2022
Die Arbeit wurde angeleitet von: Prof. Dr. Tilo Wettig

Prüfungsausschuss:
Prof. Dr. Jascha Repp
Prof. Dr. Tilo Wettig
Prof. Dr. Rainer Spang
Prof. Dr. Jaroslav Fabian

Datum Promotionskolloquium: 5. Oktober 2022

Abstract

We describe the process of enabling the Mutual Hazard Network model for large data sets, i.e., for high dimensions, by using the Tensor Train decomposition. We first briefly review the Mutual Hazard Network model and explain its limitations when using classical methods. We then introduce the Tensor Train format and explain how to perform required operations in it with a particular emphasis on solving systems of linear equations. Next, we explain how to apply the Tensor Train format to the Mutual Hazard Network. Furthermore, we describe some technical aspects of the software implementation. Finally, we present numerical results of different methods used to solve linear systems which occur in the Mutual Hazard Network model. These methods allow the complexity in the number of events d to be reduced from $\mathcal{O}(2^d)$ to $\mathcal{O}(d^3)$, thereby enabling the Mutual Hazard Network model to be applied to larger data sets.

Acknowledgements

I would like to express my gratitude and appreciation to all the people who made this work possible. First and foremost I would like to thank Tilo Wettig who made it possible for me to work on this thesis in the first place. I am grateful for his support whenever needed, but above all for his trust in me and for giving me the freedom to work on this topic as I see fit. I would also like to thank Lars Grasedyck and Rainer Spang, and especially Maren Klever and Rudolf Schill, who are all working on this joint project.

Regardless of this work, I would like to take this opportunity to thank Andreas Schäfer and Tilo Wettig for their support during my studies and my further academic career to this day. I also wish to thank my two office colleagues, Nils Meyer and Stefan Solbrig, for their support from day one since we shared an office. Almost ten years have passed since then and I am very grateful to have had the pleasure of sharing an office with such nice colleagues during this time. Thanks for all the interesting and funny conversations on all kinds of topics. Further, I would like to mention Heidi Decock and Monika Maschek and thank them for their support with all sorts of administrative tasks and beyond.

I would also like to express my gratitude to Fritz Wünsch who entrusted me with the responsibility of the C++ & Qt course at the beginning of my doctoral studies. Teaching this course has greatly improved my own C++ skills and thus contributed to this work.

My special thanks go to all colleagues who accompanied me in all these years for their mutual support and making each work day, no matter how stressful or annoying, much more enjoyable. Here I would like to mention in particular Marius Löffler, Rudolf Rödl, Daniel Richtmann, Maximilian Schlemmer, Jakob Simeth, and Simon Weishäupl. Thank you for your friendship!

I would also like to thank my teammates and friends at Rugby Club Regensburg 2000 for their support over the past few years. Without this balance to my daily work, I would certainly not be who I am today.

Lastly, I want to express my gratitude to my parents, my two brothers, and my sister for their constant support. And last but not least, I am eternally grateful to my beloved fiancée, Manon Portal, for all her support in all imaginable situations.

Preface

The work presented in this dissertation is part of an interdisciplinary project, combining advanced numerical methods and high-performance computing with cancer genomics. In order to make progress with difficult problems, the combination of individual disciplines can be helpful as each discipline can bring in its own expertise. In our case, theoretical physics, we contribute our expertise in algorithmic development and implementation of highly efficient software. Here, we use known numerical methods applied in theoretical physics, adapt them for these new applications and develop them further. Some of these newly developed methods can then be used again in physics or in other fields. The same applies to the software implementation.

Among others, researchers from the fields of applied mathematics, Lars Grasedyck[1] and Maren Klever[1], bioinformatics, Rainer Spang[2] and Rudolf Schill[2], and theoretical physics, Tilo Wettig and myself, are working on this project. Working in such an interdisciplinary environment has been a very interesting and pleasant experience.

[1] Institute for Geometry and Applied Mathematics, RWTH Aachen University
[2] Department of Statistical Bioinformatics, University of Regensburg

Contents

List of Algorithms

Chapter 1

Introduction

Cancer has a major impact on society in Germany and across the world. According to the World Health Organisation (WHO) about one in six deaths in 2018 is due to cancer, making it the second most cause of death globally. Due to the higher life expectancy the share is even higher in developed countries, e.g., for the year 2019 about 25 % of all deaths in Germany are attributed to cancer as reported by the Statistisches Bundesamt. Consequently, cancer is one of the most pressing and important research topics with large resources being invested worldwide. Increased funding in recent years, e.g., by the internationally known Cancer Moonshot Initiative (USA) launched by Barack Obama in 2016, led to encouraging progress. With these advances new research challenges emerge, of which many involve complex computations. Hence it has been recognized that expertise in applied mathematics, physics, and computation can contribute to solving these.

Cancer is a complex dynamic disease driven by various events [1]. A better understanding of this process is needed to improve tumor therapy. In tumor progression, the rates of events that have not yet occured are determined by the combination of pre-existing events, i.e., it can be modeled as a Markov process on the state space of all possible combinations of events. The state space grows exponentially with the number of events d. The recent increase of d in available datasets, mainly due to progress in genome sequencing, results in a state space explosion. Hence, tumor progression models need to mitigate the state space explosion to be applicable to current and future datasets. In this dissertation we restrict ourselves to one such progression model in particular: The Mutual Hazard Network (MHN) model introduced in [2]. See [3, 4] for a review of other current progression models.

Chapter 2

Problem statement

The main objective of this work is to enable the MHN tensor progression model for large datasets. To understand the current limitations we briefly summarize this model with an emphasis on implementation aspects. A more mathematical approach to the MHN model is presented in [5], for biological aspects see [2].

In the MHN model tumor progression is modeled as a continuous-time Markov chain on a state space $S = \{0,1\}^d$ of all combinations of predefined events d. Its transition rate matrix[1] $Q_\Theta \in \mathbb{R}^{S \times S}$ is given by

$$Q_\Theta = \sum_{i=1}^{d} Q_i \tag{2.1}$$

with

$$Q_i = \left[\bigotimes_{j=1}^{i-1} \begin{pmatrix} 1 & 0 \\ 0 & \Theta_{ij} \end{pmatrix}\right] \otimes \begin{pmatrix} -\Theta_{ii} & 0 \\ \Theta_{ii} & 0 \end{pmatrix} \otimes \left[\bigotimes_{j=i+1}^{d} \begin{pmatrix} 1 & 0 \\ 0 & \Theta_{ij} \end{pmatrix}\right] \tag{2.2}$$

and d^2 parameters $\left(\Theta_{ij}\right) =: \Theta \in \mathbb{R}^{+\,d \times d}$.

Given a dataset $\mathcal{D}$ of tumors, the optimal parameters Θ_{ij} are found by maximizing the marginal log-likelihood score

$$\mathcal{S}_{\mathcal{D}}(\Theta) = \langle \mathbf{p}_{\mathcal{D}}, \log \mathbf{p}_\Theta \rangle \tag{2.3}$$

where the logarithm of $\mathbf{p}_\Theta$ is taken element-wise. $\mathbf{p}_{\mathcal{D}} \in \mathbb{R}^{+\,2^d}$ represents an empirical probability distribution defined by $\mathcal{D}$ on S where an entry $\mathbf{p}_{\mathcal{D}}(\mathbf{x})$ is the relative frequency of observed tumors with state $\mathbf{x}$ in $\mathcal{D}$. The marginal distribution $\mathbf{p}_\Theta$ is given by

$$\mathbf{p}_\Theta = \left[\mathrm{Id} - Q_\Theta\right]^{-1} \mathbf{p}_\emptyset \tag{2.4}$$

with initial probability distribution $\mathbf{p}_\emptyset = (1, 0, \dots, 0)^T \in \mathbb{R}^{2^d}$.

[1]A transition rate matrix Q satisfies $Q(i,i) \leq 0 \ \forall i$, $Q(i,j) \geq 0 \ \forall i \neq j$, and $\sum_i Q(i,j) = 0 \ \forall j$.

In practice we do not optimize the log-likelihood score $\mathcal{S}_{\mathcal{D}}(\Theta)$ directly but add a penalty term to avoid overfitting:

$$\mathcal{S}_{\mathcal{D}}(\Theta) - \lambda \sum_{i \neq j} \left|\log \Theta_{ij}\right| \tag{2.5}$$

where λ is a tuning parameter. The penalty term promotes sparsity of the networks, i.e. it is chosen such that off-diagonal elements of Θ are pushed towards one. This means that many events do not interact, which is a desired effect.

Using classical algorithms for the optimization of $\mathcal{S}_{\mathcal{D}}(\Theta)$ it is also required to calculate the partial derivatives of $\mathcal{S}_{\mathcal{D}}(\Theta)$ with respect to each parameter Θ_{ij}. These are given by

$$\frac{\partial \mathcal{S}_{\mathcal{D}}(\Theta)}{\partial \Theta_{ij}} = \left\langle \mathbf{q}, \frac{\partial Q_\Theta}{\partial \Theta_{ij}} \mathbf{p}_\Theta \right\rangle \tag{2.6}$$

with

$$\mathbf{q} = \left[\mathrm{Id} - Q_\Theta\right]^{-T} \left(\mathbf{p}_{\mathcal{D}} \oslash \mathbf{p}_\Theta\right) \tag{2.7}$$

where $\oslash$ denotes element-wise division.

The advantage of the MHN model is that it allows us to fully describe the transition rate matrix $Q_\Theta \in \mathbb{R}^{2^d \times 2^d}$ using only d^2 parameters Θ_{ij}. Hence the model is named after the matrix Θ, which is called a Mutual Hazard Network (MHN) in [2].

However, finding the optimal parameters Θ_{ij} requires solving equations involving Q_Θ and other objects of exponential size, e.g. $\mathbf{p}_\Theta$. Hence when using classical methods the computational complexity and data requirements grow exponentially with d. In practice this limits the computation of the MHN model to $d \lessapprox 25$ on modern workstations.

To achieve our main objective – enable the MHN tensor progression model for large datasets – it is critical to avoid any exponential growth with d in both computational complexity and data requirements.

Chapter 3

Tensors

In this chapter, we will briefly introduce tensors as multidimensional generalizations of vectors and matrices and clarify the associated notations and operations where we restrict ourselves to aspects which are relevant in the following chapters.

But first we note that throughout the rest of this work, when using $\sum$, $\prod$, or any other similar notation with bounds, we omit the lower bound if it is obvious, e.g. equal to one, for brevity. We omit the index entirely if the lower bound is obvious and the index is not needed at all, i.e., if the elements do not dependent on the index. For example, we write $\prod_k^d n_k$ instead of $\prod_{k=1}^d n_k$ and $\times_k^d n_k$ instead of $\times_{k=1}^d n_k$ and shorten $\times_k^d 2$ by $\times^d 2$.

3.1 Definition

An object $x \in \mathbb{C}^{\times_k^d n_k}$ is called a *tensor* of *dimension* d. Each direction $i \in \{1, \dots, d\}$ is called a *mode* of x, and each n_i is called the *i-mode size*. A single element of a tensor x is obtained by $x(i_1, \dots, i_d)$ with *indices* $i_k \in \{1, \dots, n_k\}$.

3.2 Linear operators

A d-dimensional linear operator $X\colon \mathbb{C}^{\times_k^d n_k} \to \mathbb{C}^{\times_k^d m_k}$ is represented as a tensor in $\mathbb{C}^{(\times_k^d m_k)\times(\times_k^d n_k)}$.[1] A single element of X is obtained by $X(i_1, \dots, i_d; j_1, \dots, j_d)$ with indices $i_k \in \{1, \dots, m_k\}$ and $j_k \in \{1, \dots, n_k\}$.[1]

[1] We use parentheses and semicolons to group the mode sizes and indices into those that belong to the origin tensor space and those that belong to the target tensor space. For brevity we often refer to a multidimensional linear operator as tensor operator or simply operator.

Note that the definition of a multidimensional linear operator is not unique Other definitions are also possible by reordering the modes $m_1, \dots, m_d$ and $n_1, \dots, n_d$. One advantage of this convention is that the definition of the transpose of a multidimensional linear operator is then simply given by

$$(X^T)(j_1, \dots, j_d; i_1, \dots, i_d) = X(i_1, \dots, i_d; j_1, \dots, j_d) \tag{3.1}$$

with the transposed linear operator $X^T \in \mathbb{C}^{(\times_k^d n_k) \times (\times_k^d m_k)}$.

Having the similar shape of a tensor and a tensor operator and the necessity to add superfluous symbols to clarify the differences in mind, it is obvious that a tensor x and a tensor operator X hardly differ. In particular we may consider an operator of dimension d as a tensor of twice the dimension in many cases. This allows us to often only refer to tensors while the same applies to operators as well. In cases where there is a difference between tensors and tensor operators we will state this explicitly.

3.3 Indexing

We have already seen how to use indexing to obtain a single entry of a tensor or operator. However, this simple form is often not sufficient to allow for an efficient notation. Hence we introduce the following additional ways of indexing. In this section x is always a tensor in $\mathbb{C}^{\times_k^d n_k}$ and X an operator in $\mathbb{C}^{(\times_k^d m_k) \times (\times_k^d n_k)}$.

Free indices

It is often useful to only fix certain indices and leave others free. To do so we use colons to indicate the free modes, e.g., we may obtain $x(:, i_2, \dots, i_{d-1}, :) \in \mathbb{C}^{n_1 \times n_d}$, $x(i_1, :, \dots, :, i_d) \in \mathbb{C}^{n_2 \times \dots \times n_{d-1}}$, and $X(:, \dots, :; j_1, \dots, j_d) \in \mathbb{C}^{\times_k^d m_k}$.

Multiindex

A *multiindex* $\overline{i_p, \dots, i_q}$ is an index which takes all possible combinations of the underlying indices $i_p, \dots, i_q$. Using colexicographical ordering[2] the index is given by

$$\overline{i_p, \dots, i_q} = \sum_{k=p}^{q} \left(\prod_{j=p}^{k-1} n_j \right) i_k. \tag{3.2}$$

This allows us to consider tensors as vectors by using a single multiindex of all indices. This is referred to as *linearization.* We will often use linearization and therefore introduce a shortened notation

$$x[:] = x(\overline{:, \dots, :}) = x(:, \dots, :) \tag{3.3}$$

with $x[:] \in \mathbb{C}^{\prod_k^d n_k}$.

[2]Colexicographical ordering means that the last index i_q is the slowest changing index when storing multidimensional objects in linear storage.

Similarly, we also define a linearization for tensor operators which allows us to consider these as matrices given by

$$X[\,:\,,\,:\,] = X(\overline{\,:\,,\dots,\,:\,};\overline{\,:\,,\dots,\,:\,}) = X(\,:\,,\dots,\,:\,;\,:\,,\dots,\,:\,) \tag{3.4}$$

with $X[\,:\,,\,:\,] \in \mathbb{C}^{m_1\cdots m_d \times n_1\cdots n_d}$.[3]

3.4 Basic operations

Many basic operations which are defined for vectors or matrices can easily be applied to the multidimensional case of tensors. This is especially true for element-wise operations, e.g., addition, subtraction or the element-wise multiplication. We will not define these obvious operations. However, for some operations this is not obvious and sometimes there is no unique definition. This applies in particular to the different types of products.

Tensor product

Given two tensors $a \in \mathbb{C}^{\times_k^d m_k}$ and $b \in \mathbb{C}^{\times_k^f n_k}$ the tensor product[4] $c = a \otimes b$ is given by

$$c\big(i_1,\dots,i_d,j_1,\dots,j_f\big) = a(i_1,\dots,i_d)\; b\big(j_1,\dots,j_f\big) \tag{3.5}$$

with $c \in \mathbb{C}^{m_1 \times \cdots \times m_d \times n_1 \times \cdots \times n_f}$.

Kronecker product

The Kronecker product is usually defined for matrices only. Indeed, we only give the definition for matrices to distinguish it from the tensor product: Given two matrices $A \in \mathbb{C}^{m_1 \times n_1}$ and $B \in \mathbb{C}^{m_2 \times n_2}$ the Kronecker product $C = A \boxtimes B$ is given by

$$C\big(\overline{i_2, i_1}, \overline{j_2, j_1}\big) = A(i_1, j_1)\; B(i_2, j_2) \tag{3.6}$$

with $C \in \mathbb{C}^{m_1 m_2 \times n_1 n_2}$. However, given its definition for matrices the generalization to higher dimensions is obvious. Note the symbol $\boxtimes$ – which is also used in [6] – to denote the Kronecker product instead of the more usual $\otimes$. Often the same symbol $\otimes$ is used for both operations. We use different symbols to avoid any confusion.

Operator-by-Tensor product

Given a tensor operator $A \in \mathbb{C}^{(\times_k^d m_k) \times (\times_k^d n_k)}$ and a tensor $b \in \mathbb{C}^{\times_k^d n_k}$ the operator-by-tensor product $c = A\,b$ with $c \in \mathbb{C}^{\times_k^d m_k}$ is given by

$$c[i] = \sum_{s}^{\prod_k^d n_k} A[i,s]\; b[s]\,.[5] \tag{3.7}$$

[3]In other works, linearization is often referred to as vectorization for tensors and as matricization for operators. However, there is no single naming nor notation, especially not for the matricization.

[4]The outer product, which is very similar to the tensor product and is usually defined for vectors, may also be generalized for tensors. For two tensors a and b the outer product is given by $c = a \otimes_{\mathbb{C}} b = a \otimes b^*$.

Operator-by-Operator product

Given two tensor operators $A \in \mathbb{C}^{(\times_k^d m_k)\times(\times_k^f l_k)}$ and $B \in \mathbb{C}^{(\times_k^f l_k)\times(\times_k^d n_k)}$ the operator-by-operator product $C = A\,B$ with $C \in \mathbb{C}^{(\times_k^d m_k)\times(\times_k^d n_k)}$ is given by

$$C[i,j] = \sum_s^{\prod_k^d l_k} A[i,s]\; B[s,j]\,. \tag{3.8}$$

Inner product

Given two tensors $a \in \mathbb{C}^{\times_k^d n_k}$ and $b \in \mathbb{C}^{\times_k^d n_k}$ the inner product $\langle a, b\rangle \in \mathbb{C}$ is given by

$$\langle a, b\rangle = a^\dagger\, b = \sum_i^{\prod_k^d n_k} (a^*)[i]\; b[i]\,. \tag{3.9}$$

Given two tensor operators $A \in \mathbb{C}^{(\times_k^d m_k)\times(\times_k^d n_k)}$ and $B \in \mathbb{C}^{(\times_k^d m_k)\times(\times_k^d n_k)}$ the (Frobenius) inner product $\langle A, B\rangle \in \mathbb{C}$ is given by

$$\langle A, B\rangle = \mathrm{tr}(A^\dagger\, B) = \sum_i^{\prod_k^d m_k} \sum_j^{\prod_k^d n_k} (A^*)[i,j]\; B[i,j]\,. \tag{3.10}$$

Norm

There are many different norms for vectors and matrices of which many can be generalized to higher dimensions. In higher dimensions we then distinguish between tensor and operator norms instead of vector and matrix norms. We limit ourselves to the p-norm for both tensors and operators. The p-norm is given by

$$\|x\|_p = \left(\sum_i^{\prod_k^d n_k} |x[i]|^p\right)^{\frac{1}{p}} \qquad \text{and} \qquad \|X\|_p = \left(\sum_i^{\prod_k^d m_k} \sum_j^{\prod_k^d n_k} |X[i,j]|^p\right)^{\frac{1}{p}} \tag{3.11}$$

for a tensor x and tensor operator X, respectively. In most cases we use the 2-norm which is also known as Euclidian norm in case of vectors, Frobenius norm in case of matrices, and canonical norm as it is induced by the inner product as following for tensors and operators:[6]

$$\|x\| = \|x\|_2 = \sqrt{\langle x, x\rangle} \qquad\qquad \|X\| = \|X\|_2 = \sqrt{\langle X, X\rangle} \tag{3.12}$$

[5]We do not define the tensor-by-operator product as it is already defined by the operator-by-tensor product due to the property $c = b\,A = \left(A^T\, b^T\right)^T = A^T\, b$. Note that we do not distinguish between a tensor and its transposed for brevity, i.e., we do not distinguish between column and row vectors, but simply always assume that tensors are of the correct type in each operation.

[6]For the canonical norm we usually omit the explicit two in $\|.\|_2$.

Chapter 4

Tensor Train Decomposition

There exit many different decompositions for vectors, matrices, and multidimensional tensors. Some of these decompositions are defined in terms of specific properties of their individual parts. Others reveal interesting properties of the original object. In this chapter we look at decompositions that factorize multidimensional tensors into sums and products of lower dimensional tensors, where the total storage cost of all factors is lower than the storage cost of the multidimensional tensor itself. This is crucial for multidimensional tensors since the storage cost of storing them explicitly increases exponentially with their dimension d. For this use case various different tensor decompositions have been proposed. We restrict ourselves to the Tensor Train (TT) decomposition [7, 8]. Other possible decompositions are, e.g., *Tucker* [9] and *Hierarchical Tucker* [10, 11, 12]. Indeed the TT format can be derived as a special case from the Hierarchical Tucker format. Note that one of the main issues of interest for all these decompositions is how to find an optimal exact representation or approximation of a high dimensional tensor. We will not look into this particular issue as we will later show that all tensors in the MHN are either given in a format convertible to the TT format or a TT representation can easily be found.

In this chapter we first introduce two basic tensor decompositions folowed by the TT format. For the TT format we will then introduce important properties and describe how to perform basic operations. Next we introduce the very important rounding operation, which can be used to approximate tensors in the TT format with tensors, again in the TT format, with lower storage requirements. We then show how linear systems of equations can be solved in the TT format and how to utilize one of these techniques to accelerate the approximate evaluation of linear algebra operations.

At this point it should already be pointed out that no uniform notation and naming for TT exists. Depending on the field of research the format itself and algorithms developed for it are known under different names. Rather than referencing the various names throughout the chapter, we use one naming convention and notation, and give a brief overview of the corresponding names used in other disciplines at the end of this chapter.

4.1 Rank-One Tensors

A tensor $x \in \mathbb{C}^{\times_k^d n_k}$ is *rank one* if there exist $x^{(k)} \in \mathbb{C}^{n_k}$ $\forall k = 1, \dots, d$ such that

$$x = \bigotimes_k^d x^{(k)}. \tag{4.1}$$

This means that all of its elements are defined by

$$x(i_1, \dots, i_d) = \prod_k^d x^{(k)}(i_k) \, . \tag{4.2}$$

4.2 CP Decomposition

The idea of the CP decomposition is to express a tensor as the sum of a finite number of rank-one tensors. While this has been first proposed in [13] this decomposition has been named CANDECOMP/PARAFAC (CP) [14] after its introduction in [15] and [16]. For an overview of different names for the CP decomposition see [17, Table 3.1].

The CP decomposition of a tensor $x \in \mathbb{C}^{\times_k^d n_k}$ is given by

$$x = \sum_\alpha^r x_\alpha = \sum_\alpha^r \bigotimes_k^d x_\alpha^{(k)} \tag{4.3}$$

with *factors* $x_\alpha^{(k)} \in \mathbb{C}^{n_k}$ and *rank*[1] $r \in \mathbb{N}^+$.

Using element-wise notation (4.3) may be written as

$$x(i_1, \dots, i_d) = \sum_\alpha^r \prod_k^d x_\alpha^{(k)}(i_k) \, . \tag{4.4}$$

A core advantage of the CP decomposition is its low storage cost of $\mathcal{O}\Big(r \sum_k^d n_k\Big)$. However, the format suffers from two crucial drawbacks. First, determining the optimal rank of a given tensor x is an NP-hard problem [18]. This is negligible in case all tensors are given in CP format. Second, approximation of a tensor in CP format with one of a lower rank can be an ill-posed problem [19]. Although methods for computing the best low-rank approximation in the CP format exist, cf. [20], these are not guaranteed to work and might even fail if a good approximation is known to exist. This is an issue as many operations on tensors in CP format yield results in CP format with increased rank. This leads to steadily increasing ranks and hence to increased storage cost and computational complexity in many cases. This is an issue especially for iterative algorithms.

[1]Note that in [13] the *rank* of a tensor x is defined as the smallest number of summands r required to express a tensor x as a sum of rank-one tensors. For simplicity we call the number of summands r of a particular CP representation *rank* regardless of whether it is the *optimal rank*.

The TT decomposition, which we describe next, suffers from similar drawbacks. Its main advantage is that the approximation of a TT tensor with one of lower ranks is not an ill-posed problem, i.e., a method to calculate the best low-rank approximation exists and is guaranteed to work. This leads to a significantly increased usability. For many applications, this is even what makes the TT format usable in the first place.

4.3 Tensor Train Format

A tensor $x \in \mathbb{C}^{\times_k^d n_k}$ is said to be in the TT format if all of its elements are defined by

$$x(i_1, \dots, i_d) = \prod_k^d x^{(k)}(:, i_k, :) \tag{4.5}$$

with *cores* $x^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$ and *ranks* $r_k \in \mathbb{N}^+$. Boundary conditions $r_0 = r_d = 1$ are imposed to make the right-hand side a scalar.[2] We refer to tensors in the TT format as TT tensors for brevity. For simplicity in estimating storage cost and computational complexity we also define the *rank* $r := \max_k r_k$. Similarly we define the maximum of all mode sizes $n := \max_k n_k$. Then the storage cost of the TT decomposition is of $\mathcal{O}(dnr^2)$. The computational complexity to evaluate a single entry of x is of $\mathcal{O}(dr^2)$.[3]

Respectively, a linear operator $X \in \mathbb{C}^{(\times_k^d m_k) \times (\times_k^d n_k)}$ is said to be in the TT format if all of its elements are defined by

$$X(i_1, \dots, i_d; j_1, \dots, j_d) = \prod_k^d X^{(k)}(:, i_k, j_k, :) \tag{4.6}$$

with cores $X^{(k)} \in \mathbb{C}^{r_{k-1} \times m_k \times n_k \times r_k}$. As with tensors, we refer to linear operators in the TT format as TT operators.

Throughout this chapter we often refer to TT tensors only although the same is applicable to TT operators. This can be easily seen as a TT operator $X \in \mathbb{C}^{(\times_k^d m_k) \times (\times_k^d n_k)}$ may be viewed as a TT tensor $x \in \mathbb{C}^{\times_k^d m_k n_k}$ at any time by using multiindices $\overline{i_k, j_k}$, i.e., its cores are defined by

$$x^{(k)}\left(:, \overline{i_k, j_k}, :\right) = X^{(k)}(:, i_k, j_k, :)\,. \tag{4.7}$$

In cases where there is a difference, we specifically mention TT operators.

[2]Strictly speaking the right-hand side is an element of $\mathbb{C}^{1 \times 1} \cong \mathbb{C}$. It would therefore be necessary to take the trace of the right-hand side to obtain a scalar. For brevity we omit the trace.

[3]At first glance it seems that $d-1$ matrix-by-matrix products have to be evaluated. Exploiting the fact that the first and last core are indeed vectors – not matrices – for fixed indices i_k these can be replaced by matrix-by-vector products.

4.4 Conversion from CP to Tensor Train

We first rewrite (4.5) to emphasize the differences from the CP decomposition.

$$x = \sum_{\alpha_0}^{r_0} \cdots \sum_{\alpha_d}^{r_d} \bigotimes_{k}^{d} x^{(k)}(\alpha_{k-1}, :, \alpha_k) \tag{4.8}$$

$$x(i_1, \dots, i_d) = \sum_{\alpha_0}^{r_0} \cdots \sum_{\alpha_d}^{r_d} \prod_{k}^{d} x^{(k)}(\alpha_{k-1}, i_k, \alpha_k) \tag{4.9}$$

With these it is easy to define the conversion from CP to the TT format by comparing (4.8) to (4.3) or (4.9) to (4.4) and paying attention to the boundary condition $r_0 = r_d = 1$.

Let $\tilde{x} \in \mathbb{C}^{\times_k^d n_k}$ be a tensor in CP format with rank $\tilde{r}$. Then a tensor $x = \tilde{x}$ in TT format is given by

$$x^{(k)}(\alpha_{k-1}, i_k, \alpha_k) = \begin{cases} \tilde{x}^{(k)}_{\alpha_k}(i_k) & \text{for} \quad k = 1 \\ \delta(\alpha_{k-1}, \alpha_k)\ \tilde{x}^{(k)}_{\alpha_k}(i_k) & \forall \quad k = 2, \dots, d-1 \\ \tilde{x}^{(k)}_{\alpha_{k-1}}(i_k) & \text{for} \quad k = d \end{cases} \tag{4.10}$$

with cores $x^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$ and ranks $r_k = \tilde{r}\ \forall 1 \leq k < d$. The conversion does not require any arithmetic operations, but merely consists of reshaping objects. Hence the computational complexity is given by the required copies which are of $\mathcal{O}(dnr^2)$. In the special case of a rank-one tensor, i.e., $\tilde{r} = 1$, (4.10) may be simplified to $x^{(k)}[:] = \tilde{x}^{(k)}(:)$.

An alternative approach is to first convert all $\tilde{r}$ rank-one tensors $\tilde{x}_\alpha$ to the TT format and then evaluate the sum in the TT format. Both approaches yield the same result.

4.5 Notation

In this chapter we introduce various notations which we will use throughout the rest of the work to ease working with the TT format. Throughout this section $x \in \mathbb{C}^{\times_k^d n_k}$ is a TT tensor with cores $x^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$, $X \in \mathbb{C}^{(\times_k^d m_k) \times (\times_k^d n_k)}$ is a TT operator with cores $X^{(k)} \in \mathbb{C}^{r_{k-1} \times m_k \times n_k \times r_k}$, and k, p, q are indices with suitable conditions, e.g., $p < q$.

Core notation for TT tensor and TT operator:

$$x^{(k)}\langle i_k \rangle = x^{(k)}(:, i_k, :)$$
$$X^{(k)}\langle i_k, j_k \rangle = X^{(k)}(:, i_k, j_k, :)$$

Subtrain of TT tensor and TT operator:

$$x^{(p:q)}\langle\overline{i_p,\dots,i_q}\rangle = \prod_{k=p}^{q} x^{(k)}\langle i_k\rangle\,, \qquad x^{(p:q)} \in \mathbb{C}^{r_{p-1}\times n_p\cdots n_q\times r_q}$$

$$X^{(p:q)}\langle\overline{i_p,\dots,i_q},\overline{j_p,\dots,j_q}\rangle = \prod_{k=p}^{q} X^{(k)}\langle i_k, j_k\rangle\,, \qquad X^{(p:q)} \in \mathbb{C}^{r_{p-1}\times m_p\cdots m_q\times n_p\cdots n_q\times r_q}$$

Linearized core and subtrain of TT tensor:

$$x^{[k]}\big(\overline{\alpha_{k-1}, i_k, \alpha_k}\big) = x^{(k)}(\alpha_{k-1}, i_k, \alpha_k)\,, \qquad x^{[k]} \in \mathbb{C}^{r_{k-1}n_k r_k}$$

$$x^{[p:q]}\big(\overline{\alpha_{p-1}, i_p, \dots, i_q, \alpha_q}\big) = x^{(p:q)}\big(\alpha_{p-1}, i_p, \dots, i_q, \alpha_q\big)\,, \qquad x^{[p:q]} \in \mathbb{C}^{r_{p-1}n_p\cdots n_q r_q}$$

Left-folded TT tensor core:

$$x^{|k\rangle}\big(\overline{\alpha_{k-1}, i_k}, \alpha_k\big) = x^{(k)}(\alpha_{k-1}, i_k, \alpha_k)\,, \qquad x^{|k\rangle} \in \mathbb{C}^{r_{k-1}n_k\times r_k}$$

Right-folded TT tensor core:

$$x^{\langle k|}\big(\alpha_{k-1}, \overline{i_k, \alpha_k}\big) = x^{(k)}(\alpha_{k-1}, i_k, \alpha_k)\,, \qquad x^{\langle k|} \in \mathbb{C}^{r_{k-1}\times n_k r_k}$$

Folded TT operator core:

$$X^{|k|}\big(\overline{\alpha_{k-1}, i_k}, \overline{j_k, \alpha_k}\big) = X^{(k)}(\alpha_{k-1}, i_k, j_k, \alpha_k)\,, \qquad X^{|k|} \in \mathbb{C}^{r_{k-1}m_k\times n_k r_k}$$

Left-folded subtrain of TT tensor:

$$x^{|p:q\rangle}\big(\overline{\alpha_{p-1}, i_p, \dots, i_q}, \alpha_q\big) = x^{(p:q)}\big(\alpha_{p-1}, i_p, \dots, i_q, \alpha_q\big)\,, \qquad x^{|p:q\rangle} \in \mathbb{C}^{r_{p-1}n_p\cdots n_q\times r_q}$$

Right-folded subtrain of TT tensor:

$$x^{\langle p:q|}\big(\alpha_{p-1}, \overline{i_p, \dots, i_q, \alpha_q}\big) = x^{(p:q)}\big(\alpha_{p-1}, i_p, \dots, i_q, \alpha_q\big)\,, \qquad x^{\langle p:q|} \in \mathbb{C}^{r_{p-1}\times n_p\cdots n_q r_q}$$

Folded subtrain of TT tensor:

$$x^{|k:k+1|}\big(\overline{\alpha_{k-1}, i_k}, \overline{i_{k+1}, \alpha_{k+1}}\big) = x^{(k:k+1)}\big(\alpha_{k-1}, \overline{i_k, i_{k+1}}, \alpha_{k+1}\big)\,, \quad x^{|k:k+1|} \in \mathbb{C}^{r_{k-1}n_k\times n_{k+1}r_{k+1}}$$

Frame matrix of a core and subtrain of TT tensor:

$$x^{!(k)} = x^{|1:k-1\rangle} \boxtimes \mathrm{Id}_{n_k} \boxtimes x^{\langle k+1:d|^{T}}, \qquad x^{!(k)} \in \mathbb{C}^{n_1\cdots n_d\times r_{k-1}n_k r_k}$$

$$x^{!(p:q)} = x^{|1:p-1\rangle} \boxtimes \mathrm{Id}_{n_p\cdots n_q} \boxtimes x^{\langle q+1:d|^{T}}, \qquad x^{!(p:q)} \in \mathbb{C}^{n_1\cdots n_d\times r_{p-1}n_p\cdots n_q r_q}$$

4.6 Linearity

The TT format is linear in its cores. This follows directly from the definition of the TT format for tensors in (4.5) and operators in (4.6). The linearity allows us to write a given TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ as a product of its frame matrix of the k-th core or p to q-th subtrain and its linearized k-th core or p to q-th subtrain, respectively,

$$x = x^{!(k)}\, x^{[k]} = x^{!(p:q)}\, x^{[p:q]}. \tag{4.11}$$

4.7 Uniqueness

The TT representation of a tensor $x \in \mathbb{C}^{\times_k^d n_k}$ is not unique. Given a TT representation of a tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with cores $x^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$ an equivalent representation $\tilde{x} = x$ with cores $\tilde{x}^{(k)} \in \mathbb{C}^{\tilde{r}_{k-1} \times n_k \times \tilde{r}_k}$ is evidently given by

$$\tilde{x}^{(k)}\langle i_k\rangle := P_{k-1}^{-1}\, x^{(k)}\langle i_k\rangle\ P_k \qquad\qquad \forall k = 1, \dots, d \tag{4.12}$$

with $P_k \in \mathbb{C}^{r_k \times \tilde{r}_k}$ and its right inverse $P_k^{-1} \in \mathbb{C}^{\tilde{r}_k \times r_k}$. From (4.12) we can derive two special cases which will be useful later:

i) Given two TT representations $\tilde{x}$ and x of a tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\tilde{x}^{(k)} = x^{(k)}$ for all but the p-th and $(p+1)$-th core where $1 \le p < d$, the representations are equivalent if

$$\tilde{x}^{|p\rangle} = c^{|p\rangle} \qquad \text{and} \qquad \tilde{x}^{\langle p+1|} = R\, x^{\langle p+1|} \qquad \text{where} \qquad x^{|p\rangle} = c^{|p\rangle}\, R \tag{4.13}$$

with $c^{(p)} \in \mathbb{C}^{r_{p-1} \times n_p \times \tilde{r}_p}$ and $R \in C^{\tilde{r}_p \times r_p}$ and it holds $\tilde{r}_k = r_k\ \forall k \neq p$.[4]

ii) Given two TT representations $\tilde{x}$ and x of a tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\tilde{x}^{(k)} = x^{(k)}$ for all but the p-th and $(p-1)$-th core where $1 < p \le d$, the representations are equivalent if

$$\tilde{x}^{\langle p|} = c^{\langle p|} \qquad \text{and} \qquad \tilde{x}^{|p-1\rangle} = x^{|p-1\rangle}\, L \qquad \text{where} \qquad x^{\langle p|} = L\, c^{\langle p|} \tag{4.14}$$

with $c^{(p)} \in \mathbb{C}^{\tilde{r}_{p-1} \times n_p \times r_p}$ and $L \in C^{r_{p-1} \times \tilde{r}_{p-1}}$ and it holds $\tilde{r}_k = r_k\ \forall k \neq p-1$.

4.8 Basic operations

In this section we will introduce several basic operations which are supported in the TT format. For most we will simply describe how to execute the operation, but give no derivation. An explicit derivation is not required in most cases as it simply consists of using (4.5) or (4.6) followed by basic mathematical transformations. For each operation we will also give an estimate of its computational complexity. Note that not all operations defined on tensors or linear operators can be evaluated in the TT format.

[4]We make use of the core notation here to indicate that $c^{(p)}$ has the structure of a core.

4.8.1 Scalar multiplication

Given a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with cores $x^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$ and a scalar $\lambda \in \mathbb{C}$, the multiplication $y = \lambda\, x$ is a TT tensor with cores $y^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$. The cores $y^{(k)}$ are not uniquely defined. A possible definition is to multiply one of the d cores by λ, e.g., multiply the first core with λ in which case the cores are defined by

$$c^{(1)} = \lambda\, a^{(1)} \qquad \text{and} \qquad c^{(k)} = a^{(k)} \qquad \forall k = 2, \dots, d. \tag{4.15}$$

An equivalent approach is to scale all cores by the same scalar, i.e., define the cores by

$$c^{(1)} = \operatorname{sgn}(\lambda)\sqrt[d]{|\lambda|}\, a^{(1)} \qquad \text{and} \qquad c^{(k)} = \sqrt[d]{|\lambda|}\, a^{(k)} \qquad \forall k = 2, \dots, d. \tag{4.16}$$

Although the first approach requires less arithmetic operations it still requires copying all other cores, i.e., the computational complexity of both is $\mathcal{O}(dnr^2)$. The advantage of the second approach is a potentially increased numerical stability by ensuring that the norms of all cores are in a similar range if that was the case prior to scaling a by λ.

4.8.2 Addition

For TT tensors $a, b \in \mathbb{C}^{\times_k^d n_k}$ with cores $a^{(k)} \in \mathbb{C}^{r_{k-1}^{(a)} \times n_k \times r_k^{(a)}}$ and $b^{(k)} \in \mathbb{C}^{r_{k-1}^{(b)} \times n_k \times r_k^{(b)}}$ the sum $c = a + b$ is a TT tensor with its cores $c^{(k)} \in \mathbb{C}^{r_{k-1}^{(a)}+r_{k-1}^{(b)} \times n_k \times r_k^{(a)}+r_k^{(b)}}$ defined by

$$\begin{aligned}
c^{(1)}\langle i_1\rangle &= \begin{bmatrix} a^{(1)}\langle i_1\rangle & b^{(1)}\langle i_1\rangle \end{bmatrix}, \\
c^{(k)}\langle i_k\rangle &= \begin{bmatrix} a^{(k)}\langle i_k\rangle & 0 \\ 0 & b^{(k)}\langle i_k\rangle \end{bmatrix} \qquad \forall k = 2, \dots, d-1, \\
c^{(d)}\langle i_d\rangle &= \begin{bmatrix} a^{(d)}\langle i_d\rangle \\ b^{(d)}\langle i_d\rangle \end{bmatrix}.
\end{aligned} \tag{4.17}$$

The addition of two TT tensors requires no arithmetic operations. The computational complexity is given by the amount of data to copy, which is of $\mathcal{O}\Big(dn\left(r^{(a)} + r^{(b)}\right)^2\Big)$.

4.8.3 Element-wise product

For TT tensors $a, b \in \mathbb{C}^{\times_k^d n_k}$ with cores $a^{(k)} \in \mathbb{C}^{r_{k-1}^{(a)} \times n_k \times r_k^{(a)}}$ and $b^{(k)} \in \mathbb{C}^{r_{k-1}^{(b)} \times n_k \times r_k^{(b)}}$ the element-wise product $c = a \odot b$ is a TT tensor with its cores $c^{(k)} \in \mathbb{C}^{r_{k-1}^{(a)} r_{k-1}^{(b)} \times n_k \times r_k^{(a)} r_k^{(b)}}$ defined by

$$c^{(k)}\langle i_k\rangle = a^{(k)}\langle i_k\rangle \boxtimes b^{(k)}\langle i_k\rangle \qquad \forall k = 1, \dots, d. \tag{4.18}$$

The element-wise product requires $\mathcal{O}\Big(dn\left(r^{(a)} r^{(b)}\right)^2\Big)$ arithmetic operations.

4.8.4 Inner product

For TT tensors $a, b \in \mathbb{C}^{\times_k^d n_k}$ with cores $a^{(k)} \in \mathbb{C}^{r^{(a)}_{k-1} \times n_k \times r^{(a)}_k}$ and $b^{(k)} \in \mathbb{C}^{r^{(b)}_{k-1} \times n_k \times r^{(b)}_k}$ the inner product is given by

$$\langle a, b \rangle = \prod_k^d \left(\sum_{i_k}^{n_k} a^{(k)^*}\langle i_k \rangle \boxtimes b^{(k)} \langle i_k \rangle \right) . \tag{4.19}$$

The straightforward implementation, calculating the element-wise product followed by summation over all mode sizes and ranks, requires $\mathcal{O}\Big(dn \left(r^{(a)} r^{(b)}\right)^2\Big)$ arithmetic operations. By reordering the summation, not explicitly evaluating the Kronecker product, using auxiliary objects, and exploiting the boundary conditions $r_0 = r_d = 1$ the cost can be reduced to $\mathcal{O}\big(dnr^{(a)} r^{(b)} \left(r^{(a)} + r^{(b)}\right)\big)$, see Alg. 4.1.[5] Note that this recursive calculation is prone to numerical instabilities, in particular due to numerical underflows. Hence we normalize the auxiliary object s in every step and scale it accordingly afterwards.[6]

Algorithm 4.1: Inner product in the TT format

```
Input: TT tensors $a, b \in \mathbb{C}^{\times_k^d n_k}$
Output: Inner Product $s = \langle a, b \rangle$
1 Initialize $s := \mathrm{Id}_1$ and $\lambda := 0$
2 for $k = 1$ to $d$ do
3     $s := \sum_{\alpha^{(a)}_{k-1}}^{r^{(a)}_{k-1}} \left[ s\left(\alpha^{(a)}_{k-1}, :\right) \otimes a^{(k)^*}\left(\alpha^{(a)}_{k-1}, :, :\right)\right]$    // $s \in \mathbb{C}^{r^{(b)}_{k-1} \times n_k \times r^{(a)}_k}$
4     $s := \sum_{i_k}^{n_k} \sum_{\alpha^{(b)}_{k-1}}^{r^{(b)}_{k-1}} \left[ s\left(\alpha^{(b)}_{k-1}, i_k, :\right) \otimes b^{(k)}\left(\alpha^{(b)}_{k-1}, i_k, :\right)\right]$    // $s \in \mathbb{C}^{r^{(a)}_k \times r^{(b)}_k}$
5     $\lambda := \lambda + \log(\|s\|)$
6     $s := \|s\|^{-1} s$
7 return $s := \exp(\lambda)\ s$    // $s \in \mathbb{C}^{r^{(a)}_d \times r^{(b)}_d} = \mathbb{C}^{1\times 1} \cong \mathbb{C}$
```

[5] A partial derivation of this approach is given in Sec. B.1.

[6] Calculating a product of numbers $\prod_i \lambda_i$ where $|\lambda_i| < 1$ can result in numerical underflow. In Alg. 4.1 $\lambda_i \geq 0\ \forall i$, hence we can use the equality $\prod_i \lambda_i = \exp\big(\sum_i \log(\lambda_i)\big)$ to avoid this issue.

4.8.5 Operator-by-Tensor product

For TT operator $A \in \mathbb{C}^{(\times_k^d m_k)\times(\times_k^d n_k)}$ and TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ with cores $A^{(k)} \in \mathbb{C}^{r_{k-1}^{(A)} \times m_k \times n_k \times r_k^{(A)}}$ and $b^{(k)} \in \mathbb{C}^{r_{k-1}^{(b)} \times n_k \times r_k^{(b)}}$ the operator-by-tensor product $c = A\, b$ is a TT tensor with its cores $c^{(k)} \in \mathbb{C}^{r_{k-1}^{(A)} r_{k-1}^{(b)} \times m_k \times r_k^{(A)} r_k^{(b)}}$ defined by

$$c^{(k)}\left\langle i_k\right\rangle = \sum_{j_k}^{n_k} \left[A^{(k)}\left\langle i_k, j_k\right\rangle \boxtimes b^{(k)}\left\langle j_k\right\rangle\right] \qquad \forall k = 1, \dots, d. \tag{4.20}$$

The operator-by-tensor product requires $\mathcal{O}\Big(dmn\left(r^{(A)}r^{(b)}\right)^2\Big)$ arithmetic operations.

4.8.6 Operator-by-Operator product

For TT operators $A \in \mathbb{C}^{(\times_k^d m_k)\times(\times_k^d l_k)}$ and $B \in \mathbb{C}^{(\times_k^d l_k)\times(\times_k^d n_k)}$ with cores $A^{(k)} \in \mathbb{C}^{r_{k-1}^{(A)} \times m_k \times l_k \times r_k^{(A)}}$ and $B^{(k)} \in \mathbb{C}^{r_{k-1}^{(A)} \times l_k \times n_k \times r_k^{(A)}}$ the operator-by-operator product $C = A\, B$ is a TT operator with its cores $C^{(k)} \in \mathbb{C}^{r_{k-1}^{(A)} r_{k-1}^{(B)} \times m_k \times n_k \times r_k^{(A)} r_k^{(B)}}$ defined by

$$C^{(k)}\left\langle i_k, j_k\right\rangle = \sum_{h_k}^{l_k} \left[A^{(k)}\left\langle i_k, h_k\right\rangle \boxtimes B^{(k)}\left\langle h_k, j_k\right\rangle\right] \qquad \forall k = 1, \dots, d. \tag{4.21}$$

The operator-by-operator product requires $\mathcal{O}\Big(dlmn\left(r^{(A)}r^{(B)}\right)^2\Big)$ arithmetic operations.

4.9 Orthogonality

In this section we will introduce the important property of TT tensors and operators called orthogonality.[7] We will often use this property throughout the rest of this chapter.

4.9.1 Definition

A TT core $x^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$ is called *left-orthogonal* if

$$x^{|k\rangle^\dagger}\, x^{|k\rangle} = \sum_{i_k}^{n_k} x^{(k)}\langle i_k\rangle^\dagger\; x^{(k)}\langle i_k\rangle = \mathrm{Id}_{r_k}. \tag{4.22}$$

Given a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with cores $x^{(k)}$ left-orthogonal $\forall k = 1, \dots, p$ its subtrains $x^{(1:q)}$ with $1 \le q \le p$ are called left-orthogonal and it can be shown that

$$x^{|1:q\rangle^\dagger}\, x^{|1:q\rangle} = \mathrm{Id}_{r_q} \qquad \forall q = 1, \dots, p. \tag{4.23}$$

[7] Usually the term *orthogonal* is used for real matrices and *unitary* is used for complex matrices. To be consistent with literature we use *orthogonal* independent of whether the TT tensor is real or complex.

Respectively, a TT core $x^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$ is called *right-orthogonal* if

$$x^{\langle k|}\, x^{\langle k|^\dagger} = \sum_{i_k}^{n_k} x^{(k)}\langle i_k\rangle\; x^{(k)}\langle i_k\rangle^\dagger = \mathrm{Id}_{r_{k-1}}. \tag{4.24}$$

Given a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with cores $x^{(k)}$ right-orthogonal $\forall k = p, \dots, d$ its subtrains $x^{(q:d)}$ with $p \le q \le d$ are called right-orthogonal and it can be shown that

$$x^{\langle q:d|}\, x^{\langle q:d|^\dagger} = \mathrm{Id}_{r_{q-1}} \qquad\qquad \forall q = p, \dots, d. \tag{4.25}$$

Furthermore, the frame matrix $x^{!(k)}$ of a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ is called *orthogonal* if its subtrains $x^{(1:k-1)}$ and $x^{(k+1:d)}$ are left- or right-orthogonal, respectively.

The proof for (4.23) can be found in Sec. B.2. The proof for (4.25) is analogous.

4.9.2 Orthogonalization

Given a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with cores $x^{(k)} \in \mathbb{C}^{r_{k-1} \times n_k \times r_k}$ an equivalent representation $\tilde{x} = x$ with cores $\tilde{x}^{(k)} \in \mathbb{C}^{\tilde{r}_{k-1} \times n_k \times \tilde{r}_k}$ whose p-th ($1 \le p < d$) core $\tilde{x}^{(p)}$ is left-orthogonal may be obtained by following (4.13) and therefore setting

$$\tilde{x}^{|p\rangle} := Q^{|p\rangle} \qquad \text{and} \qquad \tilde{x}^{\langle p+1|} := R\, x^{\langle p+1|}, \tag{4.26}$$

and $\tilde{x}^{(k)} := x^{(k)}$ otherwise where $Q^{|p\rangle}$ and R are chosen such that

$$x^{|p\rangle} = Q^{|p\rangle}\, R \qquad \text{and} \qquad Q^{|p\rangle^\dagger}\, Q^{|p\rangle} = \mathrm{Id}_{r_p}. \tag{4.27}$$

The QR decomposition of $x^{|p\rangle}$ is an obvious choice which satisfies both conditions. Note that while the ranks $\tilde{r}_k$ are equal to r_k $\forall k \ne p$, $\tilde{r}_p$ is not necessarily equal to r_p as it depends on the choice of $Q^{|p\rangle}$ and R. In particular, when using the *thin* [21] or *reduced* [22] QR decomposition, $\tilde{r}_p = \min\{r_{p-1}n_p, r_p\}$, i.e., the ranks of $\tilde{x}$ might be lower than the ranks of x although $x = \tilde{x}$.

Given a TT tensor x an equivalent representation $\tilde{x} = x$ with all but the last core left-orthogonal may be obtained by repeating this method in a structured way as shown on the left in Alg. 4.2. In practice we use a slightly modified Alg. 4.3 to improve numerical stability by normalizing R in each step before multiplying it with the neighboring core and scale $x^{(d)}$ accordingly afterwards.[8] The computational complexity of both algorithms is of $\mathcal{O}(dnr^3)$ and the rank $\tilde{r}$ has an upper bound of $\prod_k^d n_k$.

The respective method and algorithm for right-orthogonalization are very similar and have the same computational complexity and other properties. Since both methods are very similar, both are shown side by side in Alg. 4.2 and 4.3.

[8]This is essentially the same idea we already employed for the inner product in Alg. 4.1.

Algorithm 4.2: Orthogonalization of TT tensor

Input: TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$

Output: TT tensor $\tilde{x} = x$

Output: $\tilde{x}^{(1:d-1)}$ **left**-orthogonal

```
1 Initialize $p^{(1)} := x^{(1)}$
2 for $k = 1$ to $d - 1$ do
3     $[\tilde{x}^{|k\rangle}, R] := \mathrm{QR}(p^{|k\rangle})$
4         $p^{\langle k+1|} := R\, x^{\langle k+1|}$
5 $\tilde{x}^{(d)} := p^{(d)}$
```

Output: $\tilde{x}^{(2:d)}$ **right**-orthogonal

```
1 Initialize $p^{(d)} := x^{(d)}$
2 for $k = d$ to $2$ by $-1$ do
3     $[L, \tilde{x}^{\langle k|}] := \mathrm{LQ}(p^{\langle k|})$
4         $p^{|k-1\rangle} := x^{|k-1\rangle} L$
5 $\tilde{x}^{(1)} := p^{(1)}$
```

Algorithm 4.3: Orthogonalization of TT tensor with normalization

Input: TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$

Output: TT tensor $\tilde{x} = x$

Output: $\tilde{x}^{(1:d-1)}$ **left**-orthogonal

```
1 Initialize $p^{(1)} := x^{(1)}$ and $\lambda := 0$
2 for $k = 1$ to $d - 1$ do
3     $[\tilde{x}^{|k\rangle}, R] := \mathrm{QR}(p^{|k\rangle})$
4         $p^{\langle k+1|} := \|R\|^{-1}\, R\, x^{\langle k+1|}$
5             $\lambda := \lambda + \log(\|R\|)$
6 $\tilde{x}^{(d)} := \exp(\lambda)\; p^{(d)}$
```

Output: $\tilde{x}^{(2:d)}$ **right**-orthogonal

```
1 Initialize $p^{(d)} := x^{(d)}$ and $\lambda := 0$
2 for $k = d$ to $2$ by $-1$ do
3     $[L, \tilde{x}^{\langle k|}] := \mathrm{LQ}(p^{\langle k|})$
4         $p^{|k-1\rangle} := \|L\|^{-1}\, x^{|k-1\rangle} L$
5             $\lambda := \lambda + \log(\|L\|)$
6 $\tilde{x}^{(1)} := \exp(\lambda)\; p^{(1)}$
```

4.9.3 Norm

The orthogonality of a TT tensor is not directly related to its norm, but merely gives us another approach to calculate its norm. Given a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with orthogonal frame matrix $x^{!(p)}$ where $1 \le p \le d$, its norm is given by

$$\|x\| = \left\|x^{(p)}\right\|. \tag{4.28}$$

The proof of this assertion can be found in Sec. B.3. This allows us to compute the norm of x either by orthogonalization or inner product $\langle x, x\rangle$. Both approaches require $\mathcal{O}(dnr^3)$ arithmetic operations. However, the latter suffers from worse numerical stability, especially if we use one of the modified variants with improved numerical stability for the former.

4.10 Rounding

We have seen in Sec. 4.8 that almost all of the basic operations may increase the ranks of a TT tensor leading to an increased storage and computational cost for subsequent operations. Especially for iterative algorithms this can quickly lead to very large ranks nullifying the benefits of using the TT format. Furthermore, the resulting TT tensor of many operations suffers from suboptimal ranks, i.e., an equivalent representation of the same tensor with lower ranks exists. For example when adding up the same TT tensor all ranks are doubled. However, we can also compute the same quantity by scaling the TT tensor by two which does not increase the ranks. Apparently the addition leads to suboptimal ranks.

Even though the left- and right-orthogonalization may be used to obtain an equivalent representation with possible lower ranks this is not a sufficient solution as its behavior depends on the mode sizes and can not be controlled. Furthermore, we do not always require the full accuracy but want to approximate a tensor maintaining a given accuracy. In other words, we want to find the best possible representation minimizing all ranks approximating the original tensor with a given accuracy.

Before we describe the method to obtain such a representation, let's first remember that a similar problem with a well-known solution exists for matrices: Given a matrix $A \in \mathbb{C}^{m\times n}$ the singular value decomposition (SVD) is given by[9]

$$A(i,j) = \sum_{\alpha}^{r} U(i,\alpha)\ S(\alpha,\alpha)\ V(\alpha,j) \tag{4.29}$$

with $U \in \mathbb{C}^{m\times r}$ left-orthogonal[10], $S \in \mathbb{C}^{r\times r}$ diagonal matrix of singular values in descending order, $V \in \mathbb{C}^{r\times n}$ right-orthogonal[10], and $r = \min\{m,n\}$[11]. An approximation $\tilde{A} \approx A$ is given by considering only the $\tilde{r} < r$ largest singular values of A

$$\tilde{A}(i,j) = \sum_{\alpha}^{\tilde{r}} U(i,\alpha)\ S(\alpha,\alpha)\ V(\alpha,j) = \sum_{\alpha}^{\tilde{r}} \tilde{U}(i,\alpha)\ \tilde{S}(\alpha,\alpha)\ \tilde{V}(\alpha,j) \tag{4.30}$$

with absolute error

$$\left\|A - \tilde{A}\right\| = \sqrt{\sum_{\alpha=\tilde{r}+1}^{r} S(\alpha,\alpha)^2}\,. \tag{4.31}$$

$\tilde{U} \in \mathbb{C}^{m\times\tilde{r}}$, $\tilde{S} \in \mathbb{C}^{\tilde{r}\times\tilde{r}}$, $\tilde{V} \in \mathbb{C}^{\tilde{r}\times n}$ are submatrices of U, S, V discarding all columns and rows corresponding to the smallest $r - \tilde{r}$ singular values. We denote this operation as $\mathrm{SVD}_{\tilde{r}}$ and refer to it as $\tilde{r}$-*truncated* SVD. It has been proven, e.g., in [23], that $\tilde{A}$ is

[9]Contrary to the common definition of the SVD we do not use the conjugate transpose of V.

[10] U and V are *semi-unitary* matrices. However, for a semi-unitary (rectangular) matrix A it is not obvious which of the conditions $A^\dagger A = \mathrm{Id}$ or $AA^\dagger = \mathrm{Id}$ applies. The former applies if the number of rows exceeds the number of columns, the latter otherwise. Hence we use *left-orthogonal* ($A^\dagger A = \mathrm{Id}$) and *right-orthogonal* ($AA^\dagger = \mathrm{Id}$) for clarification.

[11]We use the *thin* or *economy-sized* SVD not storing unnecessary columns of U and rows of V.

the best possible approximation of A for a given $\tilde{r}$. However, in most cases we do not want to find the best possible approximation for a given $\tilde{r}$, but instead find the lowest possible $\tilde{r}$ for which a given accuracy δ is maintained. Hence we define the *δ-truncated* SVD of a given matrix $A \in \mathbb{C}^{m \times n}$ denoted by $\mathrm{SVD}_\delta(A)$

$$\mathrm{SVD}_\delta(A) = \tilde{U}\,\tilde{S}\,\tilde{V} = \tilde{A} \approx A = U\,S\,V = \mathrm{SVD}(A) \tag{4.32}$$

with $\tilde{U} \in \mathbb{C}^{m \times \tilde{r}}$ left-orthogonal, $\tilde{S} \in \mathbb{C}^{\tilde{r} \times \tilde{r}}$ diagonal matrix of the $\tilde{r}$ largest singular values in descending order, $\tilde{V} \in \mathbb{C}^{\tilde{r} \times n}$ right-orthogonal, and $\tilde{r} \in \mathbb{N}^+$ the smallest possible number of the largest singular values to be considered to satisfy $\left\|A - \tilde{A}\right\| \leq \delta\, \|A\|$. Note that we use the same notation for both types of truncation, i.e., whether its subscript is a number of singular values or a desired accuracy is crucial.

Next we describe the *rounding* method which allows us to approximate a TT tensor x by $\tilde{x} = \mathcal{R}_\varepsilon(x) \approx x$ with $\|x - \tilde{x}\| \leq \varepsilon\, \|x\|$ following [24, 8]. The basic idea is to determine each optimal rank $\tilde{r}_k \leq r_k$ independently. The rank r_k is determined by the k-th *unfolding* matrix

$$x\big(\overline{i_1, \dots, i_k}, \overline{i_{k+1}, \dots, i_d}\big) = \sum_{\alpha_k=1}^{r_k} x^{|k\rangle}\big(\overline{i_1, \dots, i_k}\big)\; x^{\langle k+1|}\big(\overline{i_{k+1}, \dots, i_d}\big)\,. \tag{4.33}$$

Hence we may obtain the optimal rank $\tilde{r}_k$ maintaining a given accuracy δ_k by calculating the δ_k-truncated SVD of the k-th unfolding matrix. Obviously we do not want to calculate the SVD of this possibly very large matrix directly. Instead we assume that $x^{!(k)}$ is orthogonal. Then the SVD of the k-th unfolding matrix may be obtained by calculating the SVD of the smaller matrix $x^{|k\rangle}$. Retaining (4.13) we can now define a method to optimize a single rank of x:

Given a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with orthogonal frame matrix $x^{!(p)}$ where $1 \leq p < d$, an approximation $\tilde{x} \approx x$ with optimal rank $\tilde{r}_p \leq r_p$ and $\|x - \tilde{x}\| \leq \delta_p\, \|x\|$ is given by setting

$$\tilde{x}^{|p\rangle} = \tilde{U}^{|p\rangle} \quad \text{and} \quad \tilde{x}^{\langle p+1|} = \tilde{S}\,\tilde{V}\,x^{\langle p+1|} \quad \text{where} \quad \mathrm{SVD}_{\delta_p}\big(\tilde{x}^{|p\rangle}\big) = \tilde{U}^{|p\rangle}\,\tilde{S}\,\tilde{V} \tag{4.34}$$

and $\tilde{x}^{(k)} = x^{(k)}$ otherwise.

Notice that the $(p+1)$-th frame matrix of the resulting tensor $\tilde{x}$ is orthogonal. Because of that it is now clear how to use this method to optimize all $d-1$ ranks in a structured way. It only remains to be shown how the errors of each step accumulate to the total error. Using the sub-additivity property of the norm it is easy to see that $\|x - \tilde{x}\| \leq \sum_k^{d-1} \delta_k\, \|x\|$. In [8, Theorem 2.2] a lower bound

$$\|x - \tilde{x}\| \leq \sqrt{\sum_k^{d-1} \delta_k^2}\; \|x\| \leq \sum_k^{d-1} \delta_k\, \|x\| \tag{4.35}$$

for the total error has been proven. While we are free to choose different accuracies δ_k for each step we usually choose a common value $\delta = \delta_k$. A possible choice for δ satisfying $\|x - \tilde{x}\| \leq \varepsilon\, \|x\|$ is

$$\delta = \frac{1}{\sqrt{d-1}}\,\varepsilon. \tag{4.36}$$

The resulting algorithm which we refer to as *truncation* is shown on the left in Alg. 4.4.

Algorithm 4.4: Truncation of TT tensor

Input: TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ and accuracy ε
Output: TT tensor $\tilde{x} \approx x$ with optimal ranks, $\|x - \tilde{x}\| \le \varepsilon \|x\|$

```
Input: $x^{(2:d)}$ right-orthogonal
Output: $\tilde{x}^{(1:d-1)}$ left-orthogonal

1 Initialize $p^{(1)} := x^{(1)}$ and $\delta := \varepsilon \sqrt[-2]{d-1}$
2 for $k = 1$ to $d-1$ do
3     $[\tilde{x}^{|k\rangle}, S, V] := \mathrm{SVD}_\delta(p^{|k\rangle})$
4     $p^{\langle k+1|} := S\, V\, x^{\langle k+1|}$
5 $\tilde{x}^{(d)} := p^{(d)}$
```

```
Input: $x^{(1:d-1)}$ left-orthogonal
Output: $\tilde{x}^{(2:d)}$ right-orthogonal

1 Initialize $p^{(d)} := x^{(d)}$ and $\delta := \varepsilon \sqrt[-2]{d-1}$
2 for $k = d$ to $2$ by $-1$ do
3     $[U, S, \tilde{x}^{\langle k|}] := \mathrm{SVD}_\delta(p^{\langle k|})$
4     $p^{|k-1\rangle} := x^{|k-1\rangle}\, U\, S$
5 $\tilde{x}^{(1)} := p^{(1)}$
```

Note that the chosen assumption to efficiently calculate the SVD of the k-th unfolding matrix is ambiguous. The other equally good choice is to assume that $x^{!(k+1)}$ is orthogonal. The SVD of the k-th unfolding matrix may then be obtained by calculating the SVD of $x^{\langle k+1|}$.[12] The corresponding algorithm is very similar and basically just runs in the other direction, hence both are shown side by side in Alg. 4.4.

We have seen that both the algorithms for truncation and orthogonalization exist in pairs, one starting with the first core and one starting with the last core. To better distinguish between these algorithms for truncation and orthogonalization, we refer to them as *left-to-right* or *right-to-left* truncation and orthogonalization depending on wheter we start with the first or last core, respectively.

The rounding method is given by combining one of the truncation algorithms in Alg. 4.4 with a matching algorithm from Alg. 4.2, or a variant thereof in Alg. 4.3, to ensure the initial orthogonalization. In practice we use a variant of the orthogonalization algorithms with improved numeric stability and apply similar modifications to the truncation algorithm. Again, we normalize all matrices in each step before multiplying with the neighboring core. In addition we normalize the core we start with and all orthogonal cores after truncation.[13] Afterwards we scale each core with the same value to restore the correct norm. Hence the resulting tensor has no guaranteed orthogonality properties anymore but instead the norms of all cores are equal. These modifications have been applied to the truncation algorithms in Alg. 4.5. The computational complexity of all variants of the truncation algorithms is of $\mathcal{O}(dnr^3)$. This is the same computational complexity as for the orthogonalization and thus also for the full rounding algorithm.

[12]Note that $x^{|k\rangle}$ and $x^{\langle k+1|}$ are the only two cores depending on r_k, i.e., we need to calculate the SVD of one or the other to optimize r_k. Hence there are no further reasonable choices.

[13]Note that the canonical norm of a left- or right-orthogonal matrix is given by its mode sizes.

Algorithm 4.5: Truncation of TT tensor with normalization

Input: TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ and accuracy ε

Output: TT tensor $\tilde{x} \approx x$ with optimal ranks, $\|x - \tilde{x}\| \leq \varepsilon \, \|x\|$

```
Input: $x^{(2:d)}$ right-orthogonal
1 Initialize $p^{(1)} := \|x^{(1)}\|^{-1} \, x^{(1)}$,
    $\delta := \varepsilon \sqrt[-2]{d-1}$, and $\lambda := \log(\|x^{(1)}\|)$
2 for $k = 1$ to $d-1$ do
3     $[U,\, S,\, V] := \mathrm{SVD}_\delta(p^{|k\rangle})$
4     $p^{\langle k+1|} := \|S\|^{-1} \, S\, V\, x^{\langle k+1|}$
5     $\lambda := \lambda + \log(\|S\|) + \log(\|U\|)$
6     $\tilde{x}^{|k\rangle} := \|U\|^{-1} \, U$
7 $\tilde{x}^{(d)} := p^{(d)}$

Input: $x^{(1:d-1)}$ left-orthogonal
1 Initialize $p^{(d)} := \|x^{(d)}\|^{-1} \, x^{(d)}$,
    $\delta := \varepsilon \sqrt[-2]{d-1}$, and $\lambda := \log(\|x^{(d)}\|)$
2 for $k = d$ to $2$ by $-1$ do
3     $[U,\, S,\, V] := \mathrm{SVD}_\delta(p^{\langle k|})$
4     $p^{|k-1\rangle} := \|S\|^{-1} \, x^{|k-1\rangle}\, U\, S$
5     $\lambda := \lambda + \log(\|S\|) + \log(\|V\|)$
6     $\tilde{x}^{\langle k|} := \|V\|^{-1} \, V$
7 $\tilde{x}^{(1)} := p^{(1)}$

8 for $k = 1$ to $d$ do
9     $\tilde{x}^{(k)} := \exp(\lambda d^{-1}) \; \tilde{x}^{(k)}$
```

4.11 Linear Systems

In many problems, including MHN, it is required to solve a system of linear equations

$$A\,x = b. \tag{4.37}$$

These can, e.g., be solved using direct or iterative methods, where the former is often not a viable choice, especially for large systems. An alternative approach is to solve the linear system by considering an optimization problem. Given a hermitian positive definite operator A the solution of the linear system is the minimizer of the function

$$J_{A,b}(x) = \frac{1}{2}\, \langle x, A\,x \rangle - \mathrm{Re}(\langle x, b \rangle)\,. \tag{4.38}$$

In this section we first describe one particular well-known iterative method, GMRES, and show how this method, exemplary for many iterative methods, can be adapted for the case of A, x, and b in the TT format. After that we describe algorithms solving the linear system in the TT format by optimizing the function $J_{A,b}(x)$ given above.

4.11.1 Generalized Minimal Residual

The Generalized Minimal Residual (GMRES) [25] method is a well-established Krylov subspace method to solve indefinite non-symmetric linear systems $A\,x = b$ with a given accuracy ε. Usually the restarted GMRES method denoted by GMRES(m) and shown in Alg. 4.6 is used. Compared to [25, Algorithm 4] the implementation of GMRES(m) described in Alg. 4.6 includes the proposed changes for a practical implementation described in [25, Sec. 3.2]. Note that we can either use the classical Gram-Schmidt (CGS) or modified Gram-Schmidt (MGS) method for the orthogonalization in lines $10-12$. The CGS method is known to have inferior numerical stability than MGS. See [26] for an analysis of the numerical behavior of both orthogonalization methods. Where the difference matters, we explicitly refer to GMRES(m) using the classical or modified Gram-Schmidt orthogonalization as CGS-GMRES(m) or MGS-GMRES(m), respectively.

Note that, like for many other algorithms that are widely used, many variants of the GMRES method exist, e.g., flexible GMRES [27], GMRES with deflated restarts [28], and loose GMRES [29]. In addition it is very often combined with a preconditioning method. See also [30] for an overview of different variants of GMRES. We restrict ourselves to the restarted GMRES(m) with no preconditioning.

Next we describe how to adapt the GMRES(m) method for the TT format, i.e., all initial operators and tensors are given in the TT format and all auxiliary tensors and the solution shall be in the TT format as well. First of all, we point out that all operations to be performed on objects in the TT format are feasible and have been described in Sec. 4.8[14]. However, two of these operations, the operator-by-tensor product and the addition, lead to increased ranks in each step of the iterative method. Hence for a practical implementation we have to reduce the ranks in each step, e.g., by using the rounding method introduced in Sec. 4.10. Apparently this has to be done in a controlled manner maintaining the convergence of the iterative method.

The calculation of the Krylov tensors v_j involves operator-by-tensor products. It has been shown in [31, 32, 33, 34] that an inexact solution of these products is sufficient. Furthermore, the required accuracy may be relaxed proportional to the inverse of the norm of the residual. Various strategies for such a relaxed accuracy have been proposed in [31, 32, 33, 34]. One possible strategy is to use

$$\delta_j = \|b - A\,x\| \; |g(j)|^{-1} \, \varepsilon \tag{4.39}$$

as a relaxed accuracy for the operator-by-tensor product. This means we can replace $v_{j+1} := A\,v_j$ in line 9 of Alg. 4.6 by $v_{j+1} := \mathcal{R}_{\delta_j}\big(A\,v_j\big)$. The initial residual v_1 may also be approximated. In this case we can not further relax the accuracy but have to use ε. However, instead of only approximating the operator-by-tensor product we also apply the rounding method after the subtraction, i.e., we replace $v_1 := b - A\,x$ in line 4 by $v_1 := \mathcal{R}_\varepsilon(b - \mathcal{R}_\varepsilon(A\,x))$.

[14]Strictly speaking the subtraction of two TT tensors is not directly possible, but can be implemented using a scalar multiplication followed by an addition.

Algorithm 4.6: GMRES(m)

Input: Operator $A \in \mathbb{C}^{n \times n}$, right-hand side $b \in \mathbb{C}^n$, initial guess $x_0 \in \mathbb{C}^n$, accuracy ε, and number of inner iterations m

Output: Solution $x \in \mathbb{C}^n$ with $\|b - A\,x\| \leq \varepsilon\,\|b\|$

```
1  Initialize: $x := x_0$, $g \in \mathbb{C}^m$, and $R \in \mathbb{C}^{m+1 \times m}$
2  do
3      Initialize: $g := \mathbf{0}$, $R := \mathbf{0}$, and $j := 1$
4        $v_j := b - A\,x$
5      $g(j) := \|v_j\|$
6      while $j \leq m$ and $|g(j)| > \varepsilon\,\|b\|$ do
7            $v_j := \|v_j\|^{-1}\, v_j$
9          $v_{j+1} := A\,v_j$

           /* CGS-GMRES($m$) */                              /* MGS-GMRES($m$) */
10         for $i = 1$ to $j$ do                             for $i = 1$ to $j$ do
11             $R(i,j) := \langle v_{j+1}, v_i \rangle$              $R(i,j) := \langle v_{j+1}, v_i \rangle$
12         $v_{j+1} := v_{j+1} - \sum_{i=1}^{j} R(i,j)\; v_i$          $v_{j+1} := v_{j+1} - R(i,j)\; v_i$

14         for $i = 1$ to $j-1$ do  $R(i\,{:}\,i+1, j) := G_i\, R(i\,{:}\,i+1, j)$
15         $R(j+1, j) := \|v_{j+1}\|$
16         Find Givens rotation $G_j \in \mathbb{C}^{2 \times 2}$ such that $G_j\, R(j\,{:}\,j+1, j) = r\,\mathrm{e}_1$
17         $R(j\,{:}\,j+1, j) := G_j\, R(j\,{:}\,j+1, j)$
18           $g(j\,{:}\,j+1) := G_j\, g(j\,{:}\,j+1)$
19         $j := j + 1$
20     Compute solution $y \in \mathbb{C}^j$ of reduced system $R(1\,{:}\,j, 1\,{:}\,j)\; y = g(1\,{:}\,j)$
21     $x := x + \sum_{i=1}^{j} y(i)\; v_i$
22 while $|g(j)| > \varepsilon\,\|b\|$
```

The orthogonalization of the Krylov tensors v_j requires calculating multiple additions or subtractions. Formally we may not approximate any of the interim results but have to evaluate those exactly to maintain the orthogonality. Only after the orthogonalization is done the Krylov tensor v_j may be approximated using the relaxed accuracy δ_j. This applies to both the CGS and MGS case. However, in case of using CGS it is reasonable to approximate the sum $\sum_i^j R(i,j)\ v_i$ before subtracting it from v_{j+1} and then rounding again afterwards. But we have to calculate the sum exactly, i.e., we must not apply the rounding operation after each addition. This is not only true for this summation, but for any sum in general it is better to sum exactly and approximate only the final result. This is especially true if the summands differ significantly in magnitude. Depending on the number of summands and their ranks this calculation may not be practically feasible. Hence for a practical implementation we require a method to calculate the approximate solution of a sum of TT tensors. In case of using MGS it is not really reasonable to approximate each of the subtractions. Nevertheless, for a practical implementation we have to apply the rounding operation after each subtraction. Therefore, assuming a method to approximately calculate the sum exists, using CGS instead of MGS seems to be the better choice in terms of stability.

In [35] it has been shown that the convergence of an iterative method is maintained if the rounding error is sufficiently small. Following this statement we apply the rounding operation to the correction of the solution, i.e., we may replace $x := x + \sum_i^j y(i)\ v_i$ in line 21 of Alg. 4.6 by $x := \mathcal{R}_\varepsilon\Big(x + \mathcal{R}_\varepsilon\Big(\sum_i^j y(i)\ v_i\Big)\Big)$. The aforementioned arguments concerning the calculation of the sum apply here as well.

So far we have always used an accuracy for the approximation of any TT tensor which depends on the accuracy of the GMRES(m) method. We introduce two new accuracies, δ and γ, to allow for better control of the approximations. These accuracies have to be chosen sufficiently small to ensure that the desired accuracy ε of the solution can be met. The final algorithms adapting GMRES for the TT format are shown in Alg. 4.7 and can be compared to the adaptations in [36] and [37].

To estimate the computational complexity of TT-GMRES(m) – assuming that we are using the rounding operation – it is sufficient to add up the computational complexity of all rounding operations since these are dominant. The correction of the solution is performed only once after m inner iterations, hence we give the computational complexity for one iteration and add the computational effort for the correction proportionally. Using MGS for the orthogonalization this is given by

$$\mathcal{O}\Big(dn\ \Big(\big(r^{(A)}r^{(x)}\big)^3 + \big(r^{(A)}r^{(v)}\big)^3 + \big(r^{(b)} + r^{(v)}\big)^3 + \big(r^{(x)} + r^{(v)}\big)^3 + m^2 {r^{(v)}}^3\Big)\Big) \tag{4.40}$$

where $r^{(x)}$ is the maximum rank of the solution x and $r^{(v)}$ is the maximum rank of all Krylov tensors. Using CGS an additional term of $\mathcal{O}\Big(dn\ \big(mr^{(v)}\big)^3\Big)$ is added.

Algorithm 4.7: TT-GMRES(m)

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$, $\delta$, $\gamma$,
       and number of inner iterations $m$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \leq \varepsilon\,\|b\|$
1  Initialize: $x := x_0$, $g \in \mathbb{C}^m$, and $R \in \mathbb{C}^{m+1\times m}$
2  do
3      Initialize: $g := \mathbf{0}$, $R := \mathbf{0}$, and $j := 1$
4      $v_j := \mathcal{R}_\delta(b - \mathcal{R}_\delta(A\,x))$
5      $g(j) := \|v_j\|$
6      while $j \leq m$ and $|g(j)| > \varepsilon\,\|b\|$ do
7          $v_j := \|v_j\|^{-1}\, v_j$
8          $\delta_j := \|b - A\,x\|\,|g(j)|^{-1}\,\delta$
9          $v_{j+1} := \mathcal{R}_{\delta_j}(A\,v_j)$
           /* TT-CGS-GMRES($m$) */
10         for $i = 1$ to $j$ do
11             $R(i,j) := \langle v_{j+1}, v_i\rangle$
12         $v_{j+1} := \mathcal{R}_{\delta_j}\big(v_{j+1} - \mathcal{R}_\delta\big(\sum_{i=1}^{j} R(i,j)\ v_i\big)\big)$
           /* TT-MGS-GMRES($m$) */
10         for $i = 1$ to $j$ do
11             $R(i,j) := \langle v_{j+1}, v_i\rangle$
12             $v_{j+1} := \mathcal{R}_\delta(v_{j+1} - R(i,j)\ v_i)$
13         $v_{j+1} := \mathcal{R}_{\delta_j}(v_{j+1})$
14         for $i = 1$ to $j-1$ do $R(i:i+1,j) := G_i\,R(i:i+1,j)$
15         $R(j+1,j) := \|v_{j+1}\|$
16         Find Givens rotation $G_j \in \mathbb{C}^{2\times 2}$ such that $G_j\,R(j:j+1,j) = r\,\mathrm{e}_1$
17         $R(j:j+1,j) := G_j\,R(j:j+1,j)$
18         $g(j:j+1) := G_j\,g(j:j+1)$
19         $j := j+1$
20     Compute solution $y \in \mathbb{C}^j$ of reduced system $R(1:j,1:j)\ y = g(1:j)$
21     $x := \mathcal{R}_\gamma\big(x + \mathcal{R}_\gamma\big(\sum_{i=1}^{j} y(i)\ v_i\big)\big)$
22 while $|g(j)| > \varepsilon\,\|b\|$
```

4.11.2 Alternating Linearized Scheme

In this section, we describe the alternating linearized scheme (ALS) introduced in [38] to solve linear systems in the TT format by optimization. The main idea is to take advantage of the linearity of the TT format and optimize $J_{A,b}(x)$ over one core $x^{(k)}$ at a time while the other cores are fixed. Usually all cores are updated one after the other by looping through all cores in one direction and then in the reverse direction. This is repeated until $\|b - A\,x\| \le \varepsilon\ \|b\|$ for a given accuracy ε.

The *local* optimization problem is given by

$$\tilde{x}^{(k)} = \arg\min_{x^{(k)}} J_{A,b}(x)\,. \tag{4.41}$$

Using the linearity (4.11) we can transform this to

$$\tilde{x}^{(k)} = \arg\min_{x^{(k)}} J_{A_k,b_k}\big(x^{[k]}\big) \tag{4.42}$$

with

$$A_k = x^{!(k)^\dagger}\, A[:,:]\; x^{!(k)} \qquad\text{and}\qquad b_k = x^{!(k)^\dagger}\, b[:] \tag{4.43}$$

where $A_k \in \mathbb{C}^{r^{(x)}_{k-1} n_k r^{(x)}_k \times r^{(x)}_{k-1} n_k r^{(x)}_k}$ and $b_k \in \mathbb{C}^{r^{(x)}_{k-1} n_k r^{(x)}_k}$. The minimum $\tilde{x}^{(k)}$ of the local optimization problem is then given by the solution of the local linear system

$$A_k\, \tilde{x}^{[k]} = b_k. \tag{4.44}$$

The local linear system is small enough to be solved by a standard method, e.g., using GMRES(m) described in Alg. 4.6 or even a direct solver. While iterating over each core, it is actually sufficient to replace $x^{(k)}$ by $\tilde{x}^{(k)}$ for each local system and then move on to the next core. However, the properties of the local problem may be improved by ensuring that $x^{!(k)}$ is orthogonal in every step, see [38, Theorem 4.1]. In particular the condition number of A_k is then bound by the condition number of A and A_k is hermitian positive definite, i.e., the local system (4.44) has an unique solution. After an initial orthogonalization, single steps of left- or right-orthogonalization are sufficient to ensure the desired orthogonality. Note that we multiply the $(k-1)$-th or $(k+1)$-th core by the non-orthogonal part of $x^{|k\rangle}$ or $x^{\langle k|}$, respectively, as part of the orthogonalization. This is actually not necessary as we optimize that core next anyway. But we update the next core anyway to use it as the initial guess for solving the next local system.

The orthogonality of $x^{!(k)}$ also allows to determine the necessary accuracy γ of the local solution $\tilde{x}^{[k]}$ to maintain the given overall accuracy ε. Indeed the residual of the local system $\big\|b_k - A_k\,\tilde{x}^{[k]}\big\|$ is a lower bound of the residual $\|b - A\,x\|$:

$$\begin{aligned}
\big\|b_k - A_k\,\tilde{x}^{[k]}\big\| &= \Big\|x^{!(k)^\dagger}\, b[:] - x^{!(k)^\dagger}\, A[:,:]\; x^{!(k)}\,\tilde{x}^{[k]}\Big\| \\
&= \Big\|x^{!(k)^\dagger}\, b[:] - x^{!(k)^\dagger}\, A[:,:]\; \tilde{x}[:]\Big\| \\
&\le \Big\|x^{!(k)^\dagger}\Big\|\ \|b - A\,x\| = \|b - A\,x\|\,.
\end{aligned} \tag{4.45}$$

This means that the local linear system has to be solved with at least the same accuracy as desired for the solution of the full linear system, i.e., $\gamma \leq \varepsilon$. Solving the local linear system with a higher accuracy might improve the convergence rate of the ALS algorithm at the expense of increased time to solve the local systems. Since we are solving a local linear system for each core it is recommended to use an accuracy γ for the local solver which scales with the number of cores d, e.g., we recommend to set $\gamma := \sqrt[2]{d}\,\varepsilon$ or higher.

While the local system is of small size the naive calculation of A_k and b_k scale exponentially with d. Fortunately, A_k and b_k can be calculated using only the TT cores of A, b, and x. By starting from (4.43), re-written to express A_k for every single entry explicitly using index notation, using the definition of $x^{!(k)}$ and exploiting the TT format of A we notice that we can calculate A_k using auxiliary objects $\Psi_{A,k}$ and $\Phi_{A,k}$. Furthermore, we recognize that these auxiliary objects are defined recursively. We save ourselves the lengthy but straightforward derivation and instead only give the result for A_k:

$$\begin{aligned} A_k\Big(\overline{\alpha^{(x)}_{k-1}, i_k, \alpha^{(x)}_k}, \overline{\alpha'^{(x)}_{k-1}, i'_k, \alpha'^{(x)}_k}\Big) &= \sum_{\alpha^{(A)}_{k-1}}^{r^{(A)}_{k-1}} \sum_{\alpha^{(A)}_k}^{r^{(A)}_k} \Psi_{A,k-1}\Big(\alpha^{(x)}_{k-1}, \alpha^{(A)}_{k-1}, \alpha'^{(x)}_{k-1}\Big) \cdot \\ &\cdot A^{(k)}\Big(\alpha^{(A)}_{k-1}, i_k, i'_k, \alpha^{(A)}_k\Big)\ \Phi_{A,k}\Big(\alpha^{(x)}_k, \alpha^{(A)}_k, \alpha'^{(x)}_k\Big) \end{aligned} \tag{4.46}$$

with

$$\begin{aligned} \Psi_{A,k}\Big(\alpha^{(x)}_k, \alpha^{(A)}_k, \alpha'^{(x)}_k\Big) &= \sum_{\alpha^{(x)}_{k-1}}^{r^{(x)}_{k-1}} \sum_{\alpha'^{(x)}_{k-1}}^{r^{(x)}_{k-1}} \sum_{\alpha^{(A)}_{k-1}}^{r^{(A)}_{k-1}} \sum_{i_k}^{n_k} \sum_{i'_k}^{n_k} \Psi_{A,k-1}\Big(\alpha^{(x)}_{k-1}, \alpha^{(A)}_{k-1}, \alpha'^{(x)}_{k-1}\Big) \cdot \\ &\cdot x^{(k)^*}\Big(\alpha^{(x)}_{k-1}, i_k, \alpha^{(x)}_k\Big)\ A^{(k)}\Big(\alpha^{(A)}_{k-1}, i_k, i'_k, \alpha^{(A)}_k\Big)\ x^{(k)}\Big(\alpha'^{(x)}_{k-1}, i'_k, \alpha'^{(x)}_k\Big) \end{aligned} \tag{4.47}$$

and

$$\begin{aligned} \Phi_{A,k}\Big(\alpha^{(x)}_k, \alpha^{(A)}_k, \alpha'^{(x)}_k\Big) &= \sum_{\alpha^{(x)}_{k+1}}^{r^{(x)}_{k+1}} \sum_{\alpha'^{(x)}_{k+1}}^{r^{(x)}_{k+1}} \sum_{\alpha^{(A)}_{k+1}}^{r^{(A)}_{k+1}} \sum_{i_{k+1}}^{n_{k+1}} \sum_{i'_{k+1}}^{n_{k+1}} x^{(k+1)^*}\Big(\alpha^{(x)}_k, i_{k+1}, \alpha^{(x)}_{k+1}\Big) \cdot \\ &\cdot A^{(k+1)}\Big(\alpha^{(A)}_k, i_{k+1}, i'_{k+1}, \alpha^{(A)}_{k+1}\Big)\ x^{(k+1)}\Big(\alpha'^{(x)}_k, i'_{k+1}, \alpha'^{(x)}_{k+1}\Big) \cdot \\ &\cdot \Phi_{A,k+1}\Big(\alpha^{(x)}_{k+1}, \alpha^{(A)}_{k+1}, \alpha'^{(x)}_{k+1}\Big) \end{aligned} \tag{4.48}$$

where $\Psi_{A,k}, \Phi_{A,k} \in \mathbb{C}^{r^{(x)}_k \times r^{(A)}_k \times r^{(x)}_k}$. By convention $\Psi_{A,0} = \Phi_{A,d} = 1$. The calculation of $\Psi_{A,k}$ and $\Phi_{A,k-1}$ require $\mathcal{O}\Big(n^2 {r^{(A)}}^2 {r^{(x)}}^4\Big)$ arithmetic operations each. The same computational effort is required to compute A_k. Although this is already a significant improvement compared to the exponential growth, these can be further reduced to $\mathcal{O}\Big(n r^{(A)} {r^{(x)}}^2 \left(n r^{(A)} + r^{(x)}\right)\Big)$ for $\Psi_{A,k}$ and $\Phi_{A,k-1}$, and $\mathcal{O}\Big(n^2 r^{(A)} {r^{(x)}}^2 \left(r^{(A)} + {r^{(x)}}^2\right)\Big)$ for A_k. This is achieved by not evaluating all contractions in one step, but gradually using temporary objects. The computation of A_k is then the dominant contribution. However, A_k is often only required to compute matrix-by-vector products $A_k\, v$ which require $\mathcal{O}\left(n^2 {r^{(x)}}^4\right)$ operations. These matrix-by-vector products can also be calculated using $\Psi_{A,k}$, $\Phi_{A,k}$ and $A^{(k)}$ without explicitly evaluating A_k first. This allows us to reduce the computational complexity to $\mathcal{O}\left(n r^{(A)} {r^{(x)}}^2 \left(n r^{(A)} + r^{(x)}\right)\right)$, i.e., the same as required to compute the auxiliary objects $\Psi_{A,k}$ and $\Phi_{A,k}$ themselves.

The right-hand side b_k of the local linear system is calculated analogously:

$$\begin{aligned}
b_k\left(\overline{\alpha_{k-1}^{(x)}, i_k, \alpha_k^{(x)}}\right) = \sum_{\alpha_{k-1}^{(b)}}^{r_{k-1}^{(b)}} \sum_{\alpha_k^{(b)}}^{r_k^{(b)}} \Psi_{b,k-1}\left(\alpha_{k-1}^{(x)}, \alpha_{k-1}^{(b)}\right) \cdot \\
\cdot\, b^{(k)}\left(\alpha_{k-1}^{(b)}, i_k, \alpha_k^{(b)}\right)\, \Phi_{b,k}\left(\alpha_k^{(x)}, \alpha_k^{(b)}\right)
\end{aligned} \tag{4.49}$$

with

$$\begin{aligned}
\Psi_{b,k}\left(\alpha_k^{(x)}, \alpha_k^{(b)}\right) = \sum_{\alpha_{k-1}^{(x)}}^{r_{k-1}^{(x)}} \sum_{\alpha_{k-1}^{(b)}}^{r_{k-1}^{(b)}} \sum_{i_k}^{n_k} \Psi_{b,k-1}\left(\alpha_{k-1}^{(x)}, \alpha_{k-1}^{(b)}\right) \cdot \\
\cdot\, {x^{(k)}}^*\left(\alpha_{k-1}^{(x)}, i_k, \alpha_k^{(x)}\right)\, b^{(k)}\left(\alpha_{k-1}^{(b)}, i_k, \alpha_k^{(b)}\right)
\end{aligned} \tag{4.50}$$

and

$$\begin{aligned}
\Phi_{b,k}\left(\alpha_k^{(x)}, \alpha_k^{(b)}\right) = \sum_{\alpha_{k+1}^{(x)}}^{r_{k+1}^{(x)}} \sum_{\alpha_{k+1}^{(b)}}^{r_{k+1}^{(b)}} \sum_{i_{k+1}}^{n_{k+1}} {x^{(k+1)}}^*\left(\alpha_k^{(x)}, i_{k+1}, \alpha_{k+1}^{(x)}\right) \cdot \\
\cdot\, b^{(k+1)}\left(\alpha_k^{(b)}, i_{k+1}, \alpha_{k+1}^{(b)}\right)\, \Phi_{b,k+1}\left(\alpha_{k+1}^{(x)}, \alpha_{k+1}^{(b)}\right)
\end{aligned} \tag{4.51}$$

where $\Psi_{b,k}$, $\Phi_{b,k} \in \mathbb{C}^{r_k^{(x)} \times r_k^{(b)}}$. Again we set $\Psi_{b,0} = \Phi_{b,d} = 1$. The calculation of $\Psi_{b,k}$, $\Phi_{b,k-1}$, and b_k each require $\mathcal{O}\left(n r^{(b)} r^{(x)} \left(r^{(b)} + r^{(x)}\right)\right)$ arithmetic operations.

The auxiliary objects Ψ and Φ are computed recursively. In particular, if a core $x^{(p)}$ is updated only $\Psi_{A,k}$ and $\Psi_{b,k}$ or $\Phi_{A,k}$ and Φ_{b_k} need to be re-calculated $\forall k \geq p$ or $k < p$, respectively. Optimizing all cores in a structured way as initially proposed, e.g., first from left to right and then from right to left, we can update the auxiliary objects as required while looping through the cores. We only need to initially calculate all Ψ or Φ once depending on the initial direction.

A possible implementation of the ALS algorithm is described in Alg. 4.8. Note that both inner loops only loop over $d-1$ cores. This prevents a core from being unnecessarily optimized twice in a row. Additionally, it helps to avoid conditionals within the loop. The initial right-orthogonalization and calculation of $\Phi_{A,k}$ and $\Phi_{b,k}$ are done in one fused loop to improve performance. Furthermore, it is easy to see that we can use the same storage for $\Psi_{A,k}$ and $\Phi_{A,k}$ and for $\Psi_{b,k}$ and $\Phi_{b,k}$ as we never require both auxiliary objects with equal k at the same time.

Algorithm 4.8: TT-ALS

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$ and $\gamma \le \varepsilon$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \le \varepsilon\,\|b\|$
1  Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1$
2  for $k = d$ to 2 by $-1$ do
3      $[L,\, x^{\langle k|}] := \mathrm{LQ}(x^{\langle k|})$
4          $x^{|k-1\rangle} := x^{|k-1\rangle}\,L$
5      Form $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
6  do
7      for $k = 1$ to $d-1$ do
8          Form $A_k$ and $b_k$ folowing (4.46) and (4.49)
9          Solve $A_k\,\tilde{x}^{[k]} = b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
10         $[x^{|k\rangle},\, R] := \mathrm{QR}(\tilde{x}^{|k\rangle})$                    // Orthogonalize $x^{!(k+1)}$
11             $x^{\langle k+1|} := R\,x^{\langle k+1|}$
12         Update $\Psi_{A,k}$ and $\Psi_{b,k}$ following (4.47) and (4.50)
13     for $k = d$ to 2 by $-1$ do
14         Form $A_k$ and $b_k$ folowing (4.46) and (4.49)
15         Solve $A_k\,\tilde{x}^{[k]} = b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
16         $[L,\, x^{\langle k|}] := \mathrm{LQ}(\tilde{x}^{\langle k|})$                    // Orthogonalize $x^{!(k-1)}$
17             $x^{|k-1\rangle} := x^{|k-1\rangle}\,L$
18         Update $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
19 while $\|b - A\,x\| > \varepsilon\,\|b\|$
```

The recursive calculation of the auxiliary objects Ψ and Φ suffers from similar numerical stability issues as the calculation of the inner product. Hence we apply a similar strategy to avoid these issues as previously employed in Alg. 4.1 for the inner product, i.e., we normalize each Ψ and Φ. In addition we normalize the non-orthogonal part of $x^{|k\rangle}$ or $x^{\langle k|}$, respectively, to improve the numerical stability of the orthogonalization. We then scale the right-hand side of each local linear system and the final solution accordingly. The resulting algorithm is described in Alg. 4.9.

Algorithm 4.9: TT-ALS with normalization

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k) \times (\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$ and $\gamma \leq \varepsilon$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \leq \varepsilon\,\|b\|$
1  Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1, \lambda_x := 0$
2  for $k = d$ to 2 by $-1$ do
3      $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$,   $x^{|k-1\rangle} := x^{|k-1\rangle}\,\Big(\|L\|^{-1}\,L\Big)$,   $\lambda_x := \lambda_x + \log(\|L\|)$
4      Form $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
5      $\lambda_{A,k-1} := \big\|\Phi_{A,k-1}\big\|$,  $\Phi_{A,k-1} := \big\|\Phi_{A,k-1}\big\|^{-1}\,\Phi_{A,k-1}$
6      $\lambda_{b,k-1} := \big\|\Phi_{b,k-1}\big\|$,   $\Phi_{b,k-1} := \big\|\Phi_{b,k-1}\big\|^{-1}\,\Phi_{b,k-1}$
7  $\lambda_x := \lambda_x + \log\big(\|x^{(1)}\|\big)$,   $x^{(1)} := \big\|x^{(1)}\big\|^{-1}\,x^{(1)}$
8  do
9      for $k = 1$ to $d-1$ do
10         Form $A_k$ and $b_k$ folowing (4.46) and (4.49)
11         $\lambda := \exp\Big(\sum_{i=1}^{d-1}\big(\log(\lambda_{b,i}) - \log(\lambda_{A,i})\big) - \lambda_x\Big)$
12         Solve $A_k\,\tilde{x}^{[k]} = \lambda\,b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
13         $\big[x^{|k\rangle},\, R\big] := \mathrm{QR}\big(\tilde{x}^{|k\rangle}\big)$                    // Orthogonalize $x^{!(k+1)}$
14             $x^{\langle k+1|} := \Big(\|R\|^{-1}\,R\Big)\,x^{\langle k+1|}$,   $\lambda_x := \lambda_x + \log(\|R\|)$         // $\big\|x^{\langle k+1|}\big\| = 1$
15         Update $\Psi_{A,k}$ and $\Psi_{b,k}$ following (4.47) and (4.50)
16         $\lambda_{A,k} := \big\|\Psi_{A,k}\big\|$,  $\Psi_{A,k} := \lambda_{A,k}^{-1}\,\Psi_{A,k}$
17         $\lambda_{b,k} := \big\|\Psi_{b,k}\big\|$,   $\Psi_{b,k} := \lambda_{b,k}^{-1}\,\Psi_{b,k}$
18     for $k = d$ to 2 by $-1$ do
19         Form $A_k$ and $b_k$ folowing (4.46) and (4.49)
20         $\lambda := \exp\Big(\sum_{i=1}^{d-1}\big(\log(\lambda_{b,i}) - \log(\lambda_{A,i})\big) - \lambda_x\Big)$
21         Solve $A_k\,\tilde{x}^{[k]} = \lambda\,b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
22         $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(\tilde{x}^{\langle k|}\big)$                    // Orthogonalize $x^{!(k-1)}$
23             $x^{|k-1\rangle} := x^{|k-1\rangle}\,\Big(\|L\|^{-1}\,L\Big)$,   $\lambda_x := \lambda_x + \log(\|L\|)$         // $\big\|x^{|k-1\rangle}\big\| = 1$
24         Update $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
25         $\lambda_{A,k-1} := \big\|\Phi_{A,k-1}\big\|$,  $\Phi_{A,k-1} := \lambda_{A,k-1}^{-1}\,\Phi_{A,k-1}$
26         $\lambda_{b,k-1} := \big\|\Phi_{b,k-1}\big\|$,   $\Phi_{b,k-1} := \lambda_{b,k-1}^{-1}\,\Phi_{b,k-1}$
27 while $\|b - \exp(\lambda_x)\;A\,x\| > \varepsilon\,\|b\|$
28 $x := \exp(\lambda_x)\;x$
```

Although it is not easy to determine the leading term of the computational complexity of a full left-to-right-to-left sweep, a good estimation – assuming that we calculate A_k explicitly – is given by $\mathcal{O}\Big(dn^2 r^{(A)} {r^{(x)}}^2 \left(r^{(A)} + {r^{(x)}}^2\right) + dnr^{(b)}r^{(x)}\left(r^{(b)} + r^{(x)}\right)\Big)$. However, we have to compute the residual $\|b - A\,x\|$ of the current solution x afterwards to determine convergence. This requires $\mathcal{O}\Big(dn\left(r^{(A)}r^{(x)} + r^{(b)}\right)^3 + dn^2\left(r^{(A)}r^{(x)}\right)^2\Big)$ arithmetic operations. While the algorithm does not suffer from exponential growth, the total cost might actually be dominated by the calculation of the convergence criteria. Unfortunately, there is no cheaper way to calculate the exact residual. An alternative approach is to not calculate the residual, but instead assume that the method converged once no further progress is being made. One possible metric for the progress of the method is the relative change of the updated core $\tilde{x}^{(k)}$ to the previous solution $x^{(k)}$ in every local optimization step. We stop once this relative change is lower than a given threshold ε_F for every local optimization step k of a full left-to-right-to-left sweep. This means we stop once

$$\left\|\tilde{x}^{(k)} - x^{(k)}\right\| \leq \varepsilon_F \left\|\tilde{x}^{(k)}\right\| \qquad \forall k. \tag{4.52}$$

Another measure for the progress of the method is the residual of the updated local system using the previous solution $x^{(k)}$. Again we stop once the local residual is lower than a given threshold ε_R for every local optimization step, i.e., once

$$\left\|b_k - A_k\,x^{(k)}\right\| \leq \varepsilon_R \left\|b_k\right\| \qquad \forall k. \tag{4.53}$$

While both metrics are meaningful measurements of the progress there is no correlation between ε_F or ε_R and the given accuracy ε.

The proposed ALS to calculate the solution x of the linear system $A\,x = b$ seems to be a very promising method. However, it suffers from two major drawbacks. The first is that we have to set the ranks of the solution x a priori, i.e., the ranks are not set adaptively as required. This can easily lead to ranks either being set too low or too high. If we underestimate the ranks we may not obtain the solution with the given accuracy. If we overestimate the ranks the required computational effort will be higher than actually necessary. While this is certainly a disadvantage, it can be managed. However, the second drawback is a more severe issue: The ALS has no guarantee of finding the global minimum, i.e., the method might be stuck in local minima. Furthermore, it is hard to detect whether the method is stuck in a local minimum. Fortunately, the method works for many problems in practice.

The issue of potentially trapping in local minima makes it all the more important to choose a sensible initial guess. If none exists for a particular linear system it is recommended to use a normalized random tensor instead of, e.g., the zero tensor.

4.11.3 Modified Alternating Linearized Scheme

The modified ALS (MALS) [38] has been proposed to address the disadvantage of fixed ranks in ALS. The general idea here is to not optimize over one core $x^{(k)}$ at a time, but over the contraction of two subsequent cores, i.e., over a subtrain $x^{(k:k+1)}$, at a time. However, we still sweep through all cores in one direction first and in the other direction afterwards like for ALS. This means when looping from left-to-right we optimize over $x^{(k:k+1)}$ and over $x^{(k-1:k)}$ otherwise. Thus when looping through all cores the subtrains overlap. Apart from that, we proceed analogously to ALS. We only describe the process optimizing over $x^{(k:k+1)}$ going from left-to-right since only a single modification is required for the other direction.

To form the local linear system we also have to replace $A^{(k)}$ and $b^{(k)}$ by the corresponding subtrains in (4.44), (4.46), and (4.49). The local linear system is then given by

$$A_{k:k+1}\, \tilde{x}^{[k:k+1]} = b_{k:k+1}. \tag{4.54}$$

with

$$\begin{aligned} A_{k:k+1}\Big(\overline{\alpha_{k-1}^{(x)}, \overline{i_k, i_{k+1}}, \alpha_k^{(x)}}, \overline{{\alpha'}_{k-1}^{(x)}, \overline{i'_k, i'_{k+1}}, {\alpha'}_k^{(x)}}\Big) &= \sum_{\alpha_{k-1}^{(A)}}^{r_{k-1}^{(A)}} \sum_{\alpha_{k+1}^{(A)}}^{r_{k+1}^{(A)}} \Psi_{A,k-1}\Big(\alpha_{k-1}^{(x)}, \alpha_{k-1}^{(A)}, {\alpha'}_{k-1}^{(x)}\Big) \cdot \\ &\cdot A^{(k:k+1)}\Big(\alpha_{k-1}^{(A)}, \overline{i_k, i_{k+1}}, \overline{i'_k, i'_{k+1}}, \alpha_{k+1}^{(A)}\Big)\ \Phi_{A,k+1}\Big(\alpha_{k+1}^{(x)}, \alpha_{k+1}^{(A)}, {\alpha'}_{k+1}^{(x)}\Big) \end{aligned} \tag{4.55}$$

and

$$\begin{aligned} b_{k:k+1}\Big(\overline{\alpha_{k-1}^{(x)}, \overline{i_k, i_{k+1}}, \alpha_{k+1}^{(x)}}\Big) &= \sum_{\alpha_{k-1}^{(b)}}^{r_{k-1}^{(b)}} \sum_{\alpha_{k+1}^{(b)}}^{r_{k+1}^{(b)}} \Psi_{b,k-1}\Big(\alpha_{k-1}^{(x)}, \alpha_{k-1}^{(b)}\Big) \cdot \\ &\cdot b^{(k:k+1)}\Big(\alpha_{k-1}^{(b)}, \overline{i_k, i_{k+1}}, \alpha_{k+1}^{(b)}\Big)\ \Phi_{b,k+1}\Big(\alpha_{k+1}^{(x)}, \alpha_{k+1}^{(b)}\Big) \end{aligned} \tag{4.56}$$

where $A_{k:k+1} \in \mathbb{C}^{r_{k-1}^{(x)} n_k n_{k+1} r_{k+1}^{(x)} \times r_{k-1}^{(x)} n_k n_{k+1} r_{k+1}^{(x)}}$ and $b_{k:k+1} \in \mathbb{C}^{r_{k-1}^{(x)} n_k n_{k+1} r_{k+1}^{(x)}}$. The calculation of the auxiliary objects $\Psi_{A,k}$, $\Psi_{b,k}$, $\Phi_{A,k}$, and Φ_{b_k} remains the same.

After solving the local linear system, its solution $\tilde{x}^{(k:k+1)}$ is separated back into updated cores $\tilde{x}^{(k)}$ and $\tilde{x}^{(k+1)}$. An apparent approach to separate $\tilde{x}^{(k:k+1)}$ into two factors of the necessary form is by calculating the SVD of $\tilde{x}^{|k:k+1|} \in \mathbb{C}^{r_{k-1}^{(x)} n_k \times n_{k+1} r_{k+1}^{(x)}}$. We then set

$$x^{|k\rangle} := U \qquad \text{and} \qquad x^{\langle k+1|} := S\,V \qquad \text{with} \qquad \text{SVD}\big(\tilde{x}^{|k:k+1|}\big) = U\,S\,V. \tag{4.57}$$

This way the orthogonality of $x^{!(k+1)}$ for the next step is ensured as well. The rank $r_k^{(x)}$ of the updated solution x is determined by the number of singular values of $\tilde{x}^{|k:k+1|}$.

It is now apparent how to adaptively adjust the ranks: We simply use the δ-truncated SVD. Due to the orthogonality of $x^{!(k:k+1)}$ the *local* error of $\tilde{x}^{|k:k+1|}$ introduced by the

δ-truncated SVD is the same as the *global* error of x, i.e., the error can be controlled.[15] While we are free to choose different values of δ for each local optimization step we choose a common value of δ for all steps. Considering the accumulation of the local errors looping over all d cores an upper bound for δ is given by $\sqrt[-2]{d}\,\varepsilon$.[16]

In [39] it has been suggested to use the residual instead of the relative error to determine how many singular values to keep in the δ-truncated SVD to avoid understimating the ranks. This means we search for the smallest possible number of the largest singular values to be considered to satisfy

$$\left\|b_{k:k+1} - A_{k:k+1}\,\hat{x}^{[k:k+1]}\right\| \leq \delta\,\left\|b_{k:k+1}\right\| \tag{4.58}$$

instead of

$$\left\|\tilde{x}^{[k:k+1]} - \hat{x}^{[k:k+1]}\right\| \leq \delta\,\left\|\tilde{x}^{[k:k+1]}\right\| \tag{4.59}$$

where $\hat{x}^{|k:k+1|} \approx \tilde{x}^{|k:k+1|}$ is the result of the δ-truncated SVD of $\tilde{x}^{|k:k+1|}$. However, determining the optimal rank satisfying the condition (4.58) is more expensive as it actually requires calculating $A_{k:k+1}\,\tilde{x}^{[k:k+1]}$ multiple times with different numbers of singular values considered.

Sweeping in the other direction we only need to replace (4.57) by

$$x^{\langle k-1|} := U\,S \qquad \text{and} \qquad x^{|k\rangle} := V \qquad \text{with} \qquad \mathrm{SVD}_\delta\left(\tilde{x}^{|k-1:k|}\right) = U\,S\,V. \tag{4.60}$$

Applying the described modifications to ALS, see Alg. 4.8, then gives the MALS algorithm as shown in Alg. 4.10. Note that we should also apply the modifications to improve the numerical stability of ALS by normalization, see Alg. 4.9, to MALS. These modifications have been omitted to highlight the differences between ALS and MALS, but are straight-forward to apply and are shown in Alg. A.1 for completeness. The accuracy δ used for the truncated SVD and the accuracy γ used to solve the local linear system do not depend on each other. Still, it is beneficial to choose $\gamma < \delta$, e.g., by setting $\gamma := c\,\delta$ with $c < 1$ and where $c = 0.5$ is recommended, to prevent the truncation from catching the noise of the too imprecise solution.

The MALS addresses the disadvantage of fixed ranks in ALS at the expense of increased computational complexity. The calculation of the local objects, A_k and b_k, requires $\mathcal{O}\left(n^4 r^{(A)} {r^{(x)}}^2 \left(r^{(A)} + {r^{(x)}}^2\right)\right) + \mathcal{O}\left(n^2 r^{(b)} r^{(x)} \left(r^{(b)} + r^{(x)}\right)\right)$ arithmetic operations. Another $\mathcal{O}\left(n^4 {r^{(A)}}^3\right)$, $\mathcal{O}\left(n^2 {r^{(b)}}^3\right)$, and $\mathcal{O}\left(n^2 {r^{(x)}}^3\right)$ for the evaluation of the subtrains of A, b, and x, respectively. Plus $\mathcal{O}\left(n^3 {r^{(x)}}^3\right)$ arithmetic operations are needed for the separation of $\tilde{x}^{(k:k+1)}$ into two cores. Hence the overall computational complexity of MALS is given by $\mathcal{O}\left(dn^4 r^{(A)} \left(r^{(A)} + {r^{(x)}}^2\right)^2\right) + \mathcal{O}\left(dn^2 r^{(b)} \left(r^{(b)} + r^{(x)}\right)^2\right)$.

[15] This is only true if using the Euclidian respectively the Frobenius norm to calculate the error.
[16] Compare to the accuracy used for each truncated SVD in the rounding method in Sec. 4.10.

Algorithm 4.10: TT-MALS

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$, $\delta \leq \sqrt[-2]{d}\,\varepsilon$, and $\gamma \leq \varepsilon$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \leq \varepsilon\,\|b\|$
1  Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1$
2  for $k = d$ to $2$ by $-1$ do
3      $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$
4          $x^{|k-1\rangle} := x^{|k-1\rangle}\,L$
5      Form $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
6  do
7      for $k = 1$ to $d-1$ do
8          Form $A_{k:k+1}$ and $b_{k:k+1}$ folowing (4.55) and (4.56)
9          Solve $A_{k:k+1}\,\tilde{x}^{[k:k+1]} = b_{k:k+1}$ with accuracy $\gamma$ and initial guess $x^{[k:k+1]}$
10         $\big[x^{|k\rangle},\, S,\, V\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k:k+1|}\big)$        // Orthogonalize $x^{!(k+1)}$
11             $x^{\langle k+1|} := S\,V$
12         Update $\Psi_{A,k}$ and $\Psi_{b,k}$ following (4.47) and (4.50)
13     for $k = d$ to $2$ by $-1$ do
14         Form $A_{k-1:k}$ and $b_{k-1:k}$ folowing (4.55) and (4.56)
15         Solve $A_{k-1:k}\,\tilde{x}^{[k-1:k]} = b_{k-1:k}$ with accuracy $\gamma$ and initial guess $x^{[k-1:k]}$
16         $\big[U,\, S,\, x^{\langle k|}\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k-1:k|}\big)$        // Orthogonalize $x^{!(k-1)}$
17             $x^{|k-1\rangle} := U\,S$
18         Update $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
19 while $\|b - A\,x\| > \varepsilon\,\|b\|$
```

4.11.4 (Modified) Alternating Linearized Scheme with enrichment

Both ALS and MALS suffer from the fundamental issue of possibly getting stuck in a local minima and thus never achieving the desired accuracy of the solution. This is due to the fact that both methods only optimize local problems. These local optimizations do not contain sufficient information about the direction towards the global solution. In this sections we introduce modifications to (M)ALS which incorporate these informations leading to (more) reliable methods.

This idea is motivated by the observation in [39] that even by expanding $x^{(k)}$ with random numbers after the local optimization the global convergence properties can be improved. This concept has been further developed in [40]: After updating the k-th core $x^{(k)}$ in each local step, this and the next core are expanded according to

$$x^{|k\rangle} := \begin{bmatrix} x^{|k\rangle} & \eta^{|k\rangle} \end{bmatrix} \qquad \text{and} \qquad x^{\langle k+1|} := \begin{bmatrix} x^{\langle k+1|} \\ 0 \end{bmatrix} \tag{4.61}$$

or

$$x^{\langle k|} := \begin{bmatrix} x^{\langle k|} \\ \eta^{\langle k|} \end{bmatrix} \qquad \text{and} \qquad x^{|k-1\rangle} := \begin{bmatrix} x^{|k-1\rangle} & 0 \end{bmatrix} \tag{4.62}$$

when looping from left to right or reverse, respectively, where $\eta^{(k)} \in \mathbb{C}^{r^{(x)}_{k-1} \times n_k \times \hat{r}_k}$ or $\eta^{(k)} \in \mathbb{C}^{\hat{r}_{k-1} \times n_k \times r^{(x)}_k}$, respectively, and the zero blocks are of appropriate shape. This expansion by $\eta^{(k)}$ is called *enrichment* in [41, 42, 43, 44]. The *enrichment rank* $\hat{r}_k$ can be freely chosen and determines how much additional information is being added. In doing so, $r^{(x)}_k$ or $r^{(x)}_{k-1}$ is increased by $\hat{r}_k$ or $\hat{r}_{k-1}$, respectively, after each local optimization. Note that the enrichment does not change the current solution x but merely adds new components to the k-th core which are all cancelled by the zero block added to the neighboring core. Still the enrichment may improve the convergence. This is possible as the next local linear system is formed using $x^{!(k+1)}$ or $x^{!(k-1)}$, see (4.43), i.e., the core with the added zero block to cancel the enrichment is omitted.

In (M)ALS we make sure that $x^{!(k+1)}$ or $x^{!(k-1)}$ is orthogonal after each local step k to improve the conditioning of the next local linear system. The enrichment by $\eta^{(k)}$ most certainly does not preserve the orthogonality, i.e., we have to re-orthogonalize afterwards. Accordingly, the orthogonalization before the enrichment is not necessary. However, we keep this orthogonalization as the re-orthogonalization is then cheap. In case of MALS it is part of the truncation step which we have to do before applying the enrichment.

The additive increase in ranks after each local optimization makes it necessary to decrease the ranks in a later step to avoid high ranks. In case of MALS no modifications are required as the ranks are chosen adaptively anyway. In case of ALS we replace the QR or LQ decomposition by the δ-truncated SVD to obtain orthogonalized factors and reduce one of the local ranks at the same time. At first this might seem like a drawback, but it actually solves the disadvantage of fixed ranks in ALS while only slightly increasing the computational complexity due to the additively increasing ranks. This is also the reason why enrichment if often only used for ALS and not combined with MALS. Accordingly we restrict ourselves to ALS with enrichment. However, the ideas of MALS and enrichment can be combined which is mostly straight-forward.

The resulting ALS with random enrichment, i.e., with $\eta^{(k)}$ filled with random numbers, is shown in Alg. 4.11.[17] Here we chose the same enrichment rank $\hat{r}$ for every enrichment step. Again we omitted the normalization steps in order to not distract from the interesting differences. The normalization with added enrichment is very similar as without.

[17]The MALS with random enrichment is shown in Alg. A.2 for completeness.

Algorithm 4.11: TT-ALS with random enrichment

Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side), TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies ε, $\delta \le \sqrt[-2]{d}\,\varepsilon$, and $\gamma \le \varepsilon$, enrichment rank $\hat{r}$

Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \le \varepsilon\,\|b\|$

```
Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1$
for $k = d$ to 2 by $-1$ do
    $[L,\, x^{\langle k|}] := \mathrm{LQ}(x^{\langle k|})$
        $x^{|k-1\rangle} := x^{|k-1\rangle}\, L$
    Form $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
do
    for $k = 1$ to $d-1$ do
        Form $A_k$ and $b_k$ folowing (4.46) and (4.49)
        Solve $A_k\,\tilde{x}^{[k]} = b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
        $[x^{|k\rangle},\, S,\, V] := \mathrm{SVD}_\delta(\tilde{x}^{|k\rangle})$            // Orthogonalize $x^{!(k+1)}$
            $x^{\langle k+1|} := S\,V\,x^{\langle k+1|}$
        Enrich $x^{|k\rangle}$ by random $\eta^{|k\rangle}$ and update $x^{\langle k+1|}$ according to (4.61)
        $[x^{|k\rangle},\, R] := \mathrm{QR}(x^{|k\rangle})$            // Re-orthogonalize $x^{!(k+1)}$
            $x^{\langle k+1|} := R\,x^{\langle k+1|}$
        Update $\Psi_{A,k}$ and $\Psi_{b,k}$ following (4.47) and (4.50)
    for $k = d$ to 2 by $-1$ do
        Form $A_k$ and $b_k$ folowing (4.46) and (4.49)
        Solve $A_k\,\tilde{x}^{[k]} = b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
        $[U,\, S,\, x^{\langle k|}] := \mathrm{SVD}_\delta(\tilde{x}^{\langle k|})$            // Orthogonalize $x^{!(k-1)}$
            $x^{|k-1\rangle} := x^{|k-1\rangle}\, U\, S$
        Enrich $x^{\langle k|}$ by random $\eta^{\langle k|}$ and update $x^{|k-1\rangle}$ according to (4.62)
        $[L,\, x^{\langle k|}] := \mathrm{LQ}(x^{\langle k|})$            // Re-orthogonalize $x^{!(k-1)}$
            $x^{|k-1\rangle} := x^{|k-1\rangle}\, L$
        Update $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
while $\|b - A\,x\| > \varepsilon\,\|b\|$
```

We only add an additional normalization step after the re-orthogonalization following the enrichment step. As already explained for MALS it is beneficial to choose $\gamma < \delta$ to prevent the truncation from catching the noise of the too imprecise solution. Furthermore, it can also be advantageous here to use the residual instead of the relative error to determine how many singular values to keep in the δ-truncated SVD.

The enrichment increases the ranks of the current solution by $\hat{r}$. To avoid this increase in ranks in the final solution it is recommended to run another full sweep skipping the enrichment step after the given accuracy of the solution x has been reached.

ALS with random enrichment improves the convergence rate and helps avoid getting stuck in local minima. Unfortunately, this is not sufficient for complicated problems for which it might still get stuck in local minima. Having introduced the concept of enrichment we next want to motivate a better choice for $\eta^{(k)}$ which is more sophisticated than simply adding random values.

In [42] the ALS$(t+z)$ algorithm has been introduced which combines ALS with steepest descent (SD). The basic idea of the SD algorithm is to use the residual to push the solution in the right direction by setting $\tilde{x} = x + \alpha z$ where $z = b - A\,x$ and $\alpha \in \mathbb{C}$. In ALS$(t+z)$ one step of SD is applied followed by one or multiple ALS sweeps. This is repeated until a given accuracy has been reached. The SD step may be interpreted as a *global* enrichment which enriches all cores of the solution x with the corresponding core of the residual z. While this algorithm shows good results it is merely the motivation and basis for the alternating minimal energy (AMEn) algorithm introduced in [43].[18] The AMEn algorithm mixes the SD and ALS steps. This means that in each step we first update $x^{(k)}$ by solving the local linear system, update the residual z and use $z^{(k)}$ as enrichment. In practice calculating the exact residual is too expensive and actually not necessary since even an approximation of $z^{(k)}$ is sufficient. There exist different methods to approximate the residual, see [43, Sec. 4]. We restrict ourselves to using ALS itself to approximate the residual $z = b - A\,x$. We will explain in more details how to use ALS for fast approximate solution of linear algebra problems in Sec. 4.12.

In a nutshell, the idea is to optimize $J_{\mathrm{Id},b-A\,x}(z)$. The local optimum is then given by

$$z^{[k]} = \hat{b}_k - \hat{A}_k\,x^{[k]} \tag{4.63}$$

with

$$\hat{A}_k = {z^{!(k)}}^{\dagger}\,A[:,:]\,x^{!(k)} \qquad \text{and} \qquad \hat{b}_k = {z^{!(k)}}^{\dagger}\,b[:] \tag{4.64}$$

where $\hat{A}_k \in \mathbb{C}^{r^{(z)}_{k-1} n_k r^{(z)}_k \times r^{(x)}_{k-1} n_k r^{(x)}_k}$ and $\hat{b}_k \in \mathbb{C}^{r^{(z)}_{k-1} n_k r^{(z)}_k}$.

[18]The two articles [42] and [43] are a two-part series with [41] being a combined version of both articles.

Again, $\hat{A}_k$ and $\hat{b}_k$ can be calculated efficiently using auxiliary objects:

$$\hat{A}_k\Big(\overline{\alpha_{k-1}^{(z)}, i_k, \alpha_k^{(z)}}, \overline{{\alpha'}_{k-1}^{(x)}, i'_k, {\alpha'}_k^{(x)}}\Big) = \sum_{\alpha_{k-1}^{(A)}}^{r_{k-1}^{(A)}} \sum_{\alpha_k^{(A)}}^{r_k^{(A)}} \hat{\Psi}_{A,k-1}\Big(\alpha_{k-1}^{(z)}, \alpha_{k-1}^{(A)}, {\alpha'}_{k-1}^{(x)}\Big) \cdot \\ \cdot A^{(k)}\Big(\alpha_{k-1}^{(A)}, i_k, i'_k, \alpha_k^{(A)}\Big)\ \hat{\Phi}_{A,k}\Big(\alpha_k^{(z)}, \alpha_k^{(A)}, {\alpha'}_k^{(x)}\Big) \tag{4.65}$$

with

$$\hat{\Psi}_{A,k}\Big(\alpha_k^{(z)}, \alpha_k^{(A)}, {\alpha'}_k^{(x)}\Big) = \sum_{\alpha_{k-1}^{(z)}}^{r_{k-1}^{(z)}} \sum_{{\alpha'}_{k-1}^{(x)}}^{r_{k-1}^{(x)}} \sum_{\alpha_{k-1}^{(A)}}^{r_{k-1}^{(A)}} \sum_{i_k}^{n_k} \sum_{i'_k}^{n_k} \hat{\Psi}_{A,k-1}\Big(\alpha_{k-1}^{(z)}, \alpha_{k-1}^{(A)}, {\alpha'}_{k-1}^{(x)}\Big) \cdot \\ \cdot {z^{(k)}}^*\Big(\alpha_{k-1}^{(z)}, i_k, \alpha_k^{(z)}\Big)\ A^{(k)}\Big(\alpha_{k-1}^{(A)}, i_k, i'_k, \alpha_k^{(A)}\Big)\ x^{(k)}\Big({\alpha'}_{k-1}^{(x)}, i'_k, {\alpha'}_k^{(x)}\Big) \tag{4.66}$$

and

$$\hat{\Phi}_{A,k}\Big(\alpha_k^{(z)}, \alpha_k^{(A)}, {\alpha'}_k^{(x)}\Big) = \sum_{\alpha_{k+1}^{(z)}}^{r_{k+1}^{(z)}} \sum_{{\alpha'}_{k+1}^{(x)}}^{r_{k+1}^{(x)}} \sum_{\alpha_{k+1}^{(A)}}^{r_{k+1}^{(A)}} \sum_{i_{k+1}}^{n_{k+1}} \sum_{i'_{k+1}}^{n_{k+1}} {z^{(k+1)}}^*\Big(\alpha_k^{(z)}, i_{k+1}, \alpha_{k+1}^{(z)}\Big) \cdot \\ \cdot A^{(k+1)}\Big(\alpha_k^{(A)}, i_{k+1}, i'_{k+1}, \alpha_{k+1}^{(A)}\Big)\ x^{(k+1)}\Big({\alpha'}_k^{(x)}, i'_{k+1}, {\alpha'}_{k+1}^{(x)}\Big) \cdot \\ \cdot \hat{\Phi}_{A,k+1}\Big(\alpha_{k+1}^{(z)}, \alpha_{k+1}^{(A)}, {\alpha'}_{k+1}^{(x)}\Big) \tag{4.67}$$

where $\hat{\Psi}_{A,k}$, $\hat{\Phi}_{A,k} \in \mathbb{C}^{r_k^{(z)} \times r_k^{(A)} \times r_k^{(x)}}$ and $\hat{\Psi}_{A,0} = \hat{\Phi}_{A,d} = 1$, and

$$\hat{b}_k\Big(\overline{\alpha_{k-1}^{(z)}, i_k, \alpha_k^{(z)}}\Big) = \sum_{\alpha_{k-1}^{(b)}}^{r_{k-1}^{(b)}} \sum_{\alpha_k^{(b)}}^{r_k^{(b)}} \hat{\Psi}_{b,k-1}\Big(\alpha_{k-1}^{(z)}, \alpha_{k-1}^{(b)}\Big) \cdot \\ \cdot b^{(k)}\Big(\alpha_{k-1}^{(b)}, i_k, \alpha_k^{(b)}\Big)\ \hat{\Phi}_{b,k}\Big(\alpha_k^{(z)}, \alpha_k^{(b)}\Big) \tag{4.68}$$

with

$$\hat{\Psi}_{b,k}\Big(\alpha_k^{(z)}, \alpha_k^{(b)}\Big) = \sum_{\alpha_{k-1}^{(z)}}^{r_{k-1}^{(z)}} \sum_{\alpha_{k-1}^{(b)}}^{r_{k-1}^{(b)}} \sum_{i_k}^{n_k} \hat{\Psi}_{b,k-1}\Big(\alpha_{k-1}^{(z)}, \alpha_{k-1}^{(b)}\Big) \cdot \\ \cdot {z^{(k)}}^*\Big(\alpha_{k-1}^{(z)}, i_k, \alpha_k^{(z)}\Big)\ b^{(k)}\Big(\alpha_{k-1}^{(b)}, i_k, \alpha_k^{(b)}\Big) \tag{4.69}$$

and

$$\hat{\Phi}_{b,k}\Big(\alpha_k^{(z)}, \alpha_k^{(b)}\Big) = \sum_{\alpha_{k+1}^{(z)}}^{r_{k+1}^{(z)}} \sum_{\alpha_{k+1}^{(b)}}^{r_{k+1}^{(b)}} \sum_{i_{k+1}}^{n_{k+1}} {z^{(k+1)}}^*\Big(\alpha_k^{(z)}, i_{k+1}, \alpha_{k+1}^{(z)}\Big) \cdot \\ \cdot b^{(k+1)}\Big(\alpha_k^{(b)}, i_{k+1}, \alpha_{k+1}^{(b)}\Big)\ \hat{\Phi}_{b,k+1}\Big(\alpha_{k+1}^{(z)}, \alpha_{k+1}^{(b)}\Big) \tag{4.70}$$

where $\hat{\Psi}_{b,k}$, $\hat{\Phi}_{b,k} \in \mathbb{C}^{r_k^{(z)} \times r_k^{(b)}}$ and $\hat{\Psi}_{b,0} = \hat{\Phi}_{b,d} = 1$.

Comparing these to (4.46) – (4.51) we see that these are almost identical except that x has been replaced by z twice. Thus the computational complexity of ALS approximating the residual is dominated by the same leading term as the ALS solving the linear system with some $r^{(x)}$ replaced by $r^{(z)}$.

Usually a rough estimate of the residual is already sufficient to improve the convergence. Hence we choose a small rank $\hat{r} = r^{(z)}$. As the initial value for z we use a randomized tensor. Additionally we only run a single local optimization step updating $z^{(k)}$ when $x^{(k)}$ is updated to approximate the corrected z. This allows us to run the ALS solving $A\,x = b$ and the ALS approximating $z = b - A\,x$ simultaneously as shown in Alg. 4.12. The computational complexity of this variant of AMEn is of the same leading order as the computational complexity of ALS with random enrichment.

Note that while we may also apply enrichment to the ALS approximating the residual z this seems to be unnecessary. Especially since we always only run a single local optimization before x is updated again. However, we may also apply the enrichment by the residual z to MALS where it then makes perfect sense to use MALS to update the residual z as well. We refer to this variant of the algorithm as modified AMEn (MAMEn), see Alg. A.3.

The updated core $z^{(k)}$ of the residual can not be directly used as the enrichment $\eta^{(k)}$. An additional mapping from $\mathbb{C}^{r_{k-1}^{(z)} \times n_k \times r_k^{(z)}}$ to $\mathbb{C}^{r_{k-1}^{(x)} \times n_k \times r_k^{(z)}}$ or $\mathbb{C}^{r_{k-1}^{(z)} \times n_k \times r_k^{(x)}}$, depending on the current loop direction, is required. The enrichment $\eta^{(k)}$ is then given by

$$\eta^{\langle k|} = \vec{P}_{k-1}\, z^{\langle k|} \qquad \text{where} \qquad \vec{P}_{k-1} = x^{|1:k-1\rangle^\dagger}\, z^{|1:k-1\rangle} \tag{4.71}$$

or

$$\eta^{|k\rangle} = z^{|k\rangle}\, \overleftarrow{P}_k \qquad \text{where} \qquad \overleftarrow{P}_k = z^{\langle k+1:d|}\, x^{\langle k+1:d|^\dagger} \tag{4.72}$$

for the left-to-right or right-to-left sweep, respectively. The explicit calculation of the mapping $\vec{P}_k$ and $\overleftarrow{P}_k$ is not feasible. Fortunately, we can apply these mappings directly to $\hat{b}_k$ and $\hat{A}_k$ in (4.63) which, by exploiting the orthogonality, leads to

$$\eta^{[k]} = \vec{b}_k - \vec{A}_k x^{[k]} \qquad \text{or} \qquad \eta^{[k]} = \overleftarrow{b}_k - \overleftarrow{A}_k x^{[k]} \tag{4.73}$$

Algorithm 4.12: TT-AMEn

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k) \times (\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$, $\delta \leq \sqrt[-2]{d}\,\varepsilon$, and $\gamma \leq \varepsilon$,
       enrichment rank $\hat{r}$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \leq \varepsilon\,\|b\|$
1  Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1$
2  Initialize: $\hat{\Psi}_{A,0} = \hat{\Phi}_{A,d} = 1$, $\hat{\Psi}_{b,0} = \hat{\Phi}_{b,d} = 1$, randomized tensor $z$
3  for $k = d$ to $2$ by $-1$ do
4      $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$,     $x^{|k-1\rangle} := x^{|k-1\rangle}\,L$
5      $\big[\_,\, z^{\langle k|}\big] := \mathrm{LQ}\big(z^{\langle k|}\big)$
6      Form $\Phi_{A,k-1}$, $\Phi_{b,k-1}$, $\hat{\Phi}_{A,k-1}$, $\hat{\Phi}_{b,k-1}$ following (4.48), (4.51), (4.67), (4.70)
7  do
8      for $k = 1$ to $d - 1$ do
9          Form $A_k$, $b_k$, $\hat{A}_k$, $\hat{b}_k$, $\vec{A}_k$, $\vec{b}_k$ by (4.46), (4.49), (4.65), (4.68), (4.74), (4.75)
10         Solve $A_k\,\tilde{x}^{[k]} = b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
11         $\big[x^{|k\rangle},\, S,\, V\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k\rangle}\big)$,     $x^{\langle k+1|} := S\,V\,x^{\langle k+1|}$
12                  $z^{[k]} := \hat{b}_k - \hat{A}_k\,\tilde{x}^{[k]}$,     $\eta^{[k]} := \vec{b}_k - \vec{A}_k\,\tilde{x}^{[k]}$
13         Enrich $x^{|k\rangle}$ by $\eta^{|k\rangle}$ and update $x^{\langle k+1|}$ according to (4.61)
14         $\big[x^{|k\rangle},\, R\big] := \mathrm{QR}\big(x^{|k\rangle}\big)$,     $x^{\langle k+1|} := R\,x^{\langle k+1|}$          // Orthogonalize $x^{!(k+1)}$
15         $\big[z^{|k\rangle},\, \_\big] := \mathrm{QR}\big(z^{|k\rangle}\big)$                                  // Orthogonalize $z^{!(k+1)}$
16         Update $\Psi_{A,k}$, $\Psi_{b,k}$, $\hat{\Psi}_{A,k}$, $\hat{\Psi}_{b,k}$ following (4.47), (4.50), (4.66), (4.69)
17     for $k = d$ to $2$ by $-1$ do
18         Form $A_k$, $b_k$, $\hat{A}_k$, $\hat{b}_k$, $\overleftarrow{A}_k$, $\overleftarrow{b}_k$ by (4.46), (4.49), (4.65), (4.68), (4.77), (4.78)
19         Solve $A_k\,\tilde{x}^{[k]} = b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
20         $\big[U,\, S,\, x^{\langle k|}\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{\langle k|}\big)$,     $x^{|k-1\rangle} := x^{|k-1\rangle}\,U\,S$
21                  $z^{[k]} := \hat{b}_k - \hat{A}_k\,\tilde{x}^{[k]}$,     $\eta^{[k]} := \overleftarrow{b}_k - \overleftarrow{A}_k\,\tilde{x}^{[k]}$
22         Enrich $x^{\langle k|}$ by $\eta^{\langle k|}$ and update $x^{|k-1\rangle}$ according to (4.62)
23         $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$,     $x^{|k-1\rangle} := x^{|k+1\rangle}\,L$          // Orthogonalize $x^{!(k-1)}$
24         $\big[\_,\, z^{\langle k|}\big] := \mathrm{LQ}\big(z^{\langle k|}\big)$                                  // Orthogonalize $z^{!(k-1)}$
25         Update $\Phi_{A,k-1}$, $\Phi_{b,k-1}$, $\hat{\Phi}_{A,k-1}$, $\hat{\Phi}_{b,k-1}$ following (4.48), (4.51), (4.67), (4.70)
26 while $\|b - A\,x\| > \varepsilon\,\|b\|$
```

with

$$\vec{A}_k\left(\overline{\alpha_{k-1}^{(x)}, i_k, \alpha_k^{(z)}}, \overline{\alpha'^{(x)}_{k-1}, i'_k, \alpha'^{(x)}_k}\right) = \sum_{\alpha_{k-1}^{(A)}}^{r_{k-1}^{(A)}} \sum_{\alpha_k^{(A)}}^{r_k^{(A)}} \Psi_{A,k-1}\left(\alpha_{k-1}^{(x)}, \alpha_{k-1}^{(A)}, \alpha'^{(x)}_{k-1}\right) \cdot$$

$$\cdot A^{(k)}\left(\alpha_{k-1}^{(A)}, i_k, i'_k, \alpha_k^{(A)}\right) \hat{\Phi}_{A,k}\left(\alpha_k^{(z)}, \alpha_k^{(A)}, \alpha'^{(x)}_k\right) \tag{4.74}$$

$$\vec{b}_k\left(\overline{\alpha_{k-1}^{(x)}, i_k, \alpha_k^{(z)}}\right) = \sum_{\alpha_{k-1}^{(b)}}^{r_{k-1}^{(b)}} \sum_{\alpha_k^{(b)}}^{r_k^{(b)}} \Psi_{b,k-1}\left(\alpha_{k-1}^{(x)}, \alpha_{k-1}^{(b)}\right) \cdot$$

$$\cdot b^{(k)}\left(\alpha_{k-1}^{(b)}, i_k, \alpha_k^{(b)}\right) \hat{\Phi}_{b,k}\left(\alpha_k^{(z)}, \alpha_k^{(b)}\right) \tag{4.75}$$

$$\tag{4.76}$$

where $\vec{A}_k \in \mathbb{C}^{r_{k-1}^{(x)} n_k r_k^{(z)} \times r_{k-1}^{(x)} n_k r_k^{(x)}}$ and $\vec{b}_k \in \mathbb{C}^{r_{k-1}^{(x)} n_k r_k^{(z)}}$, or

$$\overleftarrow{A}_k\left(\overline{\alpha_{k-1}^{(z)}, i_k, \alpha_k^{(x)}}, \overline{\alpha'^{(x)}_{k-1}, i'_k, \alpha'^{(x)}_k}\right) = \sum_{\alpha_{k-1}^{(A)}}^{r_{k-1}^{(A)}} \sum_{\alpha_k^{(A)}}^{r_k^{(A)}} \hat{\Psi}_{A,k-1}\left(\alpha_{k-1}^{(z)}, \alpha_{k-1}^{(A)}, \alpha'^{(x)}_{k-1}\right) \cdot$$

$$\cdot A^{(k)}\left(\alpha_{k-1}^{(A)}, i_k, i'_k, \alpha_k^{(A)}\right) \Phi_{A,k}\left(\alpha_k^{(x)}, \alpha_k^{(A)}, \alpha'^{(x)}_k\right) \tag{4.77}$$

$$\overleftarrow{b}_k\left(\overline{\alpha_{k-1}^{(z)}, i_k, \alpha_k^{(x)}}\right) = \sum_{\alpha_{k-1}^{(b)}}^{r_{k-1}^{(b)}} \sum_{\alpha_k^{(b)}}^{r_k^{(b)}} \hat{\Psi}_{b,k-1}\left(\alpha_{k-1}^{(z)}, \alpha_{k-1}^{(b)}\right) \cdot$$

$$\cdot b^{(k)}\left(\alpha_{k-1}^{(b)}, i_k, \alpha_k^{(b)}\right) \Phi_{b,k}\left(\alpha_k^{(x)}, \alpha_k^{(b)}\right). \tag{4.78}$$

where $\overleftarrow{A}_k \in \mathbb{C}^{r_{k-1}^{(z)} n_k r_k^{(x)} \times r_{k-1}^{(x)} n_k r_k^{(x)}}$ and $\overleftarrow{b}_k \in \mathbb{C}^{r_{k-1}^{(z)} n_k r_k^{(x)}}$.

Note that we can re-use the auxiliary objects required to calculate A_k, b_k, $\hat{A}_k$, and $\hat{b}_k$. While it may seem that the mapping is only required to ensure the enrichment tensor is of appropriate size it also makes sense logically. For the computation of A_k, b_k, $\hat{A}_k$, and $\hat{b}_k$ we use the cores of x which have already been updated and enriched by the residual z during this particular loop from left to right or right to left and the cores of the residual z otherwise. E.g., when looping from left to right the computation of $\vec{A}_k$ and $\vec{b}_k$ includes all $p < k$ cores of x and all $q > k$ cores of z.

All variants of ALS with enrichment should also include normalization steps to improve numerical stability like the variants without enrichment. The (M)AMEn algorithms with normalization are shown in Alg. A.4 – A.6

4.12 Fast Approximated Linear Algebra

We have already seen that the result of many operations working on operands in TT format is given in TT format but with increased ranks. This means we essentially have to always reduce the ranks afterwards using the rounding method described in Sec. 4.10. For a TT tensor the complexity of the rounding algorithm is of $\mathcal{O}(dnr^3)$. E.g., for TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$ and TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ with ranks $r_k^{(A)}$ and $r_k^{(b)}$ the operator-by-tensor product $c = A\,b$ with ranks $r_k^{(c)} = r_k^{(A)} r_k^{(b)}$ itself requires $\mathcal{O}\Big(dn^2 \left(r^{(A)} r^{(b)}\right)^2\Big)$ arithmetic operations. Compared to $\mathcal{O}\Big(dn \left(r^{(A)} r^{(b)}\right)^3\Big)$ arithmetic operations required to reduce the ranks of c afterwards, i.e., the total cost of the operation is dominated by the rounding algorithm if $n \ll r^{(A)} r^{(b)}$.

For the addition of two TT tensors it is not as bad as the ranks are only added not multiplied. However, there are cases in which we need to calculate the sum of multiple tensors. In this case it is crucial to calculate the sum exactly and only apply the rounding operation once afterwards, i.e., we must not apply the rounding operation after each addition. This is especially important if the summands differ significantly in magnitude. Otherwise the error of the approximate solution may be too large. Suppose we sum s tensors with same rank r, then the computational complexity of the rounding operation after the summation is of $\mathcal{O}\Big(dn \left(sr\right)^3\Big)$.

Instead of describing more efficient approaches for both the approximate solution of the operator-by-tensor product and the summation of tensors we consider the more generic operation

$$x = \sum_i^s \lambda_i\, M_i\, v_i \tag{4.79}$$

with $\lambda_i \in \mathbb{C}$, TT operators $M_i \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, and TT tensors $v_i \in \mathbb{C}^{\times_k^d n_k}$ with ranks $r_k^{(M_i)}$ and $r_k^{(v_i)}$.[19] The basic idea is to find an approximate solution x with given accuracy ε by minimization of the error $\left\|\sum_i^s \lambda_i\, M_i\, v_i - x\right\|$. This is the same approach as in Sec. 4.11 for solving linear systems and has, e.g., been used in [40] to approximate operator-by-tensor products. Indeed, we can reuse the algorithms for solving linear systems by optimization, i.e., ALS and modifications thereof, introduced in Sec. 4.11 by setting

$$A \coloneqq \mathrm{Id} \qquad\qquad \text{and} \qquad\qquad b \coloneqq \sum_i^s \lambda_i\, M_i\, v_i \tag{4.80}$$

with $r_k^{(A)} = 1$ and $r_k^{(b)} = \sum_i^s r_k^{(M_i)} r_k^{(v_i)}$ $\forall k = 1, \dots, d-1$. Thus the application of ALS to find an approximate solution of (4.79) with accuracy ε is straight-forward and we only describe possible simplifications and pitfalls due to $A = \mathrm{Id}$.

[19] For brevity we use $r^{(M)} = \max_i r^{(M_i)}$ and $r^{(v)} = \max_i r^{(v_i)}$ in computational complexity estimations. Additionally we may certainly assume that $r^{(x)} \ll r^{(M)} r^{(v)}$.

Starting with the basic ALS the local optimization problem (4.44) is simplified to

$$\tilde{x}^{[k]} = b_k = x^{!(k)} \sum_i^s \lambda_i \, (M_i \, v_i) \tag{4.81}$$

as A_k, which is defined in (4.43), is equal to the identity matrix due to $A = \mathrm{Id}$ and the ensured orthogonality of $x^{!(k)}$ in every local optimization problem. It follows that $\Psi_{A,k} = \Phi_{A,k} \in \mathbb{C}^{r^{(x)} \times 1 \times r^{(x)}} \cong \mathbb{C}^{r^{(x)} \times r^{(x)}}$ are given by $\mathrm{Id}_{r^{(x)}}$. From (4.81) follows that the updated local optimum $\tilde{x}^{(k)}$ is simply given by b_k. In particular the previous $x^{(k)}$ is not only not required but of no use at all. When solving linear systems the previous solution has been used as the initial guess. This means there is no need anymore to update the next core in truncation or orthogonalization steps as this core will be optimized next anyway. The only exception is at the end of a full left-to-right-to-left sweep: The first core will not be optimized next as it is the first core to be optimized in the next sweep. However, the algorithm might be terminated if the desired accuracy has been reached. Hence in this case the next core needs to be updated appropriately in truncation and orthogonalization steps.

For the calculation of each local system we have two possible options. We can either use the same calculation (4.49) – (4.51) using b_k, $\Psi_{b,k}$, and $\Phi_{b,k}$ which requires $\mathcal{O}\Big(n r^{(x)} {r^{(b)}}^2\Big)$ arithmetic operations with $r^{(b)} = s r^{(M)} r^{(v)}$. Or we calculate the local system (4.81) by first transforming it to

$$\tilde{x}^{[k]} = b_k = x^{!(k)} \sum_i^s \lambda_i \, (M_i \, v_i) = \sum_i^s \lambda_i \, x^{!(k)} \, (M_i \, v_i) = \sum_i^s \lambda_i \, b_{i,k} \tag{4.82}$$

with $b_{i,k}$ defined similar to (4.49) but for every $b = M_i \, v_i$ and using recursively calculated auxiliary objects $\Psi_{b,i,k}$ and $\Phi_{b,i,k}$ analogue to (4.50) and (4.51). This requires $\mathcal{O}\Big(n r^{(x)} {r^{(b)}}^2\Big)$ operations with $r^{(b)} = r^{(M)} r^{(v)}$ for each summand. Hence the computational complexity can be reduced from $\mathcal{O}\Big(n s^2 r^{(x)} {r^{(M)}}^2 {r^{(v)}}^2\Big)$ to $\mathcal{O}\Big(n s r^{(x)} {r^{(M)}}^2 {r^{(v)}}^2\Big)$.

Instead of using the exact error $\left\|\sum_i^s \lambda_i \, M_i \, v_i - x\right\|$ to determine convergence the previously introduced alternatives using the current progress of the method as an indicator of convergence can be used again. Note that the relative local change (4.52) and the local residual (4.53) of the current solution x are identical for the given problem (4.79). Using ALS to approximate (4.79) the penalty of calculating the exact error is even more noticeable. The naive approach, neglecting the operator-by-tensor products themselves, requires $\mathcal{O}\Big(dn \left(s r^{(M)} r^{(v)}\right)^3\Big)$ arithmetic operations. The computational complexity can be reduced using the equality

$$\left\|\sum_i^s \lambda_i \, M_i \, v_i - x\right\| = \|x\| - 2\,\mathrm{Re}\big(\langle x, \sum_i^s \lambda_i \, M_i \, v_i \rangle\big) + \left\|\sum_i^s \lambda_i \, M_i \, v_i\right\|. \tag{4.83}$$

However, the last term, although it is constant and therefore only needs to be computed once at the beginning, already requires $\mathcal{O}\Big(dn \left(s r^{(M)} r^{(v)}\right)^3\Big)$ arithmetic operations.

This means that the leading term of the computational complexity to calculate the exact error is about the same as the one of the direct computation of (4.79) followed by a single application of the rounding operation. Hence it is crucial to not use the exact error to determine the convergence of the method.

When moving from ALS to MALS or adding random enrichment there is nothing to be considered in particular compared to solving systems of linear equations. For (M)AMEn, however, one potential pitfall should be pointed out: $\hat{\Psi}_{A,k}$ and $\hat{\Phi}_{A,k} \in \mathbb{C}^{r_k^{(z)} \times 1 \times r_k^{(x)}}$, which are defined in (4.66) and (4.67), may be replaced by matrices in $C^{r_k^{(z)} \times r_k^{(x)}}$ similar to $\Psi_{A,k}$ and $\Phi_{A,k}$ but are not equal to Id.

4.13 A remark on notation and naming

In this chapter we used some notation and naming conventions. However this does not reflect a single convention which has been agreed upon. Instead we have chosen the notation and names we thought to be best and even introduced some new notations. This was necessary since different notations and names are used in the literature depending on the scientific background of the authors. The used notation and naming is similar to the one used by mathematicians. We will not list here all possible naming conventions, but still give some examples of different naming for the most crucial parts.

The TT decomposition itself is known in quantum physics since the late 1980s as Matrix Product States (MPS) [45, 46, 47, 48]. For MPS we distinguish between periodic and non-periodic MPS. The TT decomposition corresponds to the non-periodic case. The Tensor Ring decomposition has been proposed in [49] to adapt TT for the periodic case. In the periodic case the boundary conditions $r_0 = r_d = 1$ are loosened to $r_0 = r_d$. This means the result of (4.5) is not a scalar anymore. To solve this the trace of the product has to be taken. However, loosening the boundary condition has further consequences which can not be solved that easily. In [50] some ideas to solve these issues have been proposed with varying degrees of success.

ALS is known in quantum physics as Density Matrix Renormalization Group (DMRG) [51, 52] and used to compute the ground state of many-body systems. This done by optimization of the Rayleigh Quotient instead of the energy function $J_{A,b}(x)$ (4.38).[20] It has later been adapted to solve systems of linear equations, cf. [54]. The name DMRG is used for both ALS and MALS. Hence it is often explicitly stated whether one core or the contraction of two cores are optimized at a time. In quantum physics the cores are called *sites*, i.e., we then refer to these algorithms as *one-site* or *two-site* DMRG. A similar technique to enrichment has also been introduced to DMRG in [55]. This algorithm is referred to as *corrected* one-site DMRG and has been compared to AMEn in [56]. See [57, 58] for a reasonable up to date review of different DMRG variants.

[20]Optimizing the Rayleigh Quotient ALS can also be used to compute eigenvalues, see [53].

Chapter 5

Application of Tensor Train to Mutual Hazard Networks

In the previous chapter we have introduce the Tensor Train (TT) decomposition. In this chapter we will now describe how to apply this to the specific problem of the Mutual Hazard Network (MHN). To do so we use the tensor product for $\otimes$ in (2.2) instead of the Kronecker product, i.e., $Q_\Theta \in \mathbb{R}^{(\times^d 2)\times(\times^d 2)}$ and all vectors, e.g., $\mathbf{p}_\mathcal{D}$ and $\mathbf{p}_\emptyset$, are tensors in $\mathbb{R}^{+\times^d 2}$. For the TT format to be applicable to the MHN model we need to show that all required tensors and operators can be converted to TT tensors and operators, respectively, and that all required operations are supported in the TT format. We will show that in this chapter step by step. At the end of the chapter we briefly discuss some aspects of the optimizer regardless of whether the TT format is used or not.

5.1 Mutual Hazard Networks Operator

We call $[\mathrm{Id} - Q_\Theta]$ with transition rate matrix Q_Θ, see (2.1), the MHN operator. The transition rate matrix Q_Θ consists of d summands Q_i which are all rank one, i.e., these can be easily converted to TT operators with rank one using (4.10). The identity operator Id can also be represented as a TT operator $\mathrm{Id}_{\mathrm{TT}}$ with rank one and cores $\mathrm{Id}_{\mathrm{TT}}^{|k|} = \mathrm{Id}_2$. Evaluating the sum and subtraction the resulting TT operator $[\mathrm{Id} - Q_\Theta]$ is of rank $d+1$. However it is not always sensible to evaluate the MHN operator as we might take advantage of its form. For instance when applying $[\mathrm{Id} - Q_\Theta]$ to a tensor x it might be beneficial to not evaluate $[\mathrm{Id} - Q_\Theta]$ first but instead compute the (approximate) result via

$$y = [\mathrm{Id} - Q_\Theta]\, x = \sum_{i=0}^{d} \lambda_i\, Q_i\, x \tag{5.1}$$

with $Q_0 = \mathrm{Id}$, $\lambda_0 = 1$ and $\lambda_i = -1\ \forall i \neq 0$. Note that this equation is of the same form as (4.79), i.e., we can use the methods described in Sec. 4.12 for approximate calculation.

5.2 Probability distributions

The initial probability distribution $\mathbf{p}_\emptyset$ can be represented as a TT tensor with rank one and cores $\mathbf{p}_\emptyset^{[k]} = \mathrm{e}_1$. However, the empirical probability distribution $\mathbf{p}_\mathcal{D}$ which is defined by the data set can not be easily represented as a TT tensor with low rank. Fortunately, for all current data sets $\mathbf{p}_\mathcal{D}$ is sparse which we will later use to justify a workaround. This is probably also true for future data sets. An upper bound of non-zero entries, which is also an upper bound for the rank of $\mathbf{p}_\mathcal{D}$, is given by the amount of data points which will very likely always be much smaller than 2^d for reasonable large d.

However, there is a more serious issue concerning probability distributions related to TT. A probability distribution $\mathbf{p}$ is essentially defined by two basic properties:

i) All entries of $\mathbf{p}$ are non-negative

ii) $\|\mathbf{p}\|_1 = 1$

The second property requires us to be able to calculate the 1-norm as it is used for normalization or as a termination criterion in iterative methods. Both approaches used for the 2-norm obviously do not work for the 1-norm since they exploit the connection between the inner product and the 2-norm. Nevertheless, the inner product, or rather the scalar product, can also be used to calculate the 1-norm of a probability distribution:

$$\|\mathbf{p}\|_1 = \sum_i |\mathbf{p}[i]| = \sum_i \mathbf{p}[i] = \sum_i \mathbf{p}[i]\ \mathrm{u}[i] = \langle \mathbf{p}, \mathrm{u} \rangle \tag{5.2}$$

where $\mathrm{u} \in \mathbb{R}^{\times^d 2}$ is a tensor with $\mathrm{u}[i] = 1\ \forall i$. The all-ones tensor u can be represented as a TT tensor with rank one and cores $\mathrm{u}^{(k)} = \mathbf{1}$, i.e., its cores are also all-ones tensors. Here we used property i) and the fact that a probability distribution is real-valued, i.e., it is not necessary to take the absolute value of $\mathbf{p}[i]$ as $|\mathbf{p}[i]| = \mathbf{p}[i]$ anyway. Hence property ii) poses no problem if property i) is preserved. This is the case as long as we perform all calculations exactly and do not use any form of approximation. This is obviously not possible working in the TT format as approximations are required to keep the ranks low. Neither the rounding operation nor any of the iterative methods described in Sec. 4.11 and 4.12 preserve this property. The rounding operation is based on optimizing the 2-norm. A rounding operation based on the 1-norm would be necessary. But as far as we know, such an operation does not exist. Fortunately, a slight violation often only leads to marginally increased errors without any further consequences. However, this requires special consideration in some cases and is indeed a problem that can not be neglected.

5.3 Marginal log-likelihood score

The marginal log-likelihood score function $\mathcal{S}_{\mathcal{D}}(\Theta)$, (2.3), needs to be frequently evaluated when optimizing its parameters Θ_{ij}. For this it is necessary to compute $\mathbf{p}_\Theta$, (2.4), for a given Θ_{ij} first. In (2.4) we, of course, do not compute the inverse of $[\mathrm{Id} - Q_\Theta]$ but instead solve the linear system

$$[\mathrm{Id} - Q_\Theta]\, \mathbf{p}_\Theta = \mathbf{p}_\emptyset. \tag{5.3}$$

We have already seen that both $[\mathrm{Id} - Q_\Theta]$ and $\mathbf{p}_\emptyset$ can be represented in the TT format. Hence we can use any of the algorithms for TT introduced in Sec. 4.11 to compute $\mathbf{p}_\Theta$.[1]

By exploiting the fact that Q_Θ is a transition rate matrix, there is another way to compute $\mathbf{p}_\Theta$ using an iterative method based on the uniformization method [59], cf. [5]: Given an upper bound $\gamma \geq \max_{\mathbf{x}\in S} |Q_\Theta[\mathbf{x},\mathbf{x}]| > 0$ on the diagonal entries of Q_Θ the marginal distribution $\mathbf{p}_\Theta$ is given by

$$\mathbf{p}_\Theta = \frac{1}{1+\gamma} \sum_{k=0}^{\infty} \left(\frac{\gamma}{1+\gamma}\right)^k \left[\mathrm{Id} + \frac{1}{\gamma}\, Q_\Theta\right]^k \mathbf{p}_\emptyset \tag{5.4}$$

with a natural approximation

$$\mathbf{p}_\Theta \approx \tilde{\mathbf{p}}_\Theta = \frac{1}{1+\gamma} \sum_{k=0}^{K} \left(\frac{\gamma}{1+\gamma}\right)^k \left[\mathrm{Id} + \frac{1}{\gamma} Q_\Theta\right]^k \mathbf{p}_\emptyset. \tag{5.5}$$

However, the approximation $\tilde{\mathbf{p}}_\Theta$ does not satisfy the condition $\|\tilde{\mathbf{p}}_\Theta\|_1$, i.e., it is not a probability distribution. This can be corrected by scaling $\tilde{\mathbf{p}}_\Theta$ accordingly. The proper scaling factor λ^{-1} is given by

$$\begin{aligned}\lambda &= \|\tilde{\mathbf{p}}_\Theta\|_1 \\ &= \left\| \frac{1}{1+\gamma} \sum_{k=0}^{K} \left(\frac{\gamma}{1+\gamma}\right)^k \left[\mathrm{Id} + \frac{1}{\gamma}\, Q_\Theta\right]^k \mathbf{p}_\emptyset \right\|_1 \\ &= \left\| \frac{1}{1+\gamma} \sum_{k=0}^{K} \left(\frac{\gamma}{1+\gamma}\right)^k \right\|_1 \\ &= \frac{1}{1+\gamma} \sum_{k=0}^{K} \left(\frac{\gamma}{1+\gamma}\right)^k \end{aligned} \tag{5.6}$$

where we used the fact that $\left[\mathrm{Id} + \frac{1}{\gamma}\, Q\right]$ is a *left transition matrix*.[2] Applying a left transition matrix on a probability distribution from the left yields a probability distribution.

[1]Strictly speaking all methods in Sec. 4.11 based on optimization are limited to hermitian positive definite systems. Nevertheless, these can still be formally applied to any linear system but with no guaranteed convergence to the global minimum, i.e., the solution of the linear system. Another approach is to solve $A^\dagger A\, x = A^\dagger b$ instead of $A\, x = b$. This linear system is hermitian positive definite independent of A. However this squares the ranks of the operator and increases the ranks of the right-hand side which is very likely to slow down the computation. Therefore, we recommend to apply these algorithms directly to the original linear system despite the loss of theoretical support.

[2]A transition or *stochastic* matrix P satisfies $0 \leq P(i,j) \leq 1\ \forall i,j$ and $\sum_i P(i,j) = 1\ \forall j$.

Algorithm 5.1: Uniformization method

```
Input: Transition rate matrix $Q_\Theta \in \mathbb{R}^{2^d \times 2^d}$, probability distribution $\mathbf{p}_\emptyset \in \mathbb{R}^{+2^d}$,
       $\gamma \geq \max_{\mathbf{x}\in S} |Q_\Theta[\mathbf{x},\mathbf{x}]|$, and accuracy $\varepsilon$
Output: Probability distribution $\mathbf{p} \in \mathbb{R}^{+2^d}$ with $\|[\mathrm{Id} - Q_\Theta]\,\mathbf{p} - \mathbf{p}_\emptyset\| \leq \varepsilon\,\|\mathbf{p}_\emptyset\|$
1  Initialize $k = 0$, $s = 1$, $\lambda = 1$, $p_0 = \mathbf{p}_\emptyset$, and $q = \mathbf{p}_\emptyset$
2  while $\|c^{-1}\,[\mathrm{Id} - Q_\Theta]\,p_k - \mathbf{p}_\emptyset\| > \varepsilon\,\|\mathbf{p}_\emptyset\|$ do
3          $s := \frac{\gamma}{1+\gamma}\,s$
4          $\lambda := \lambda + s$
5          $q := [\mathrm{Id} + \gamma^{-1}\,Q_\Theta]\,q$                    // $\|q\|_1 = 1$
7      $p_{k+1} := p_k + s\,q$                                   // $\|p_{k+1}\|_1 = \lambda$
9          $k := k + 1$
10 return $\mathbf{p} := \lambda^{-1}\,p_k$
```

The individual terms of the sum and product can be calculated recursively. This enables an efficient implementation as shown in Alg. 5.1. Note that we used the relative error with respect to the 2-norm to determine when the approximation is sufficiently accurate. However, this requires applying the operator $[\mathrm{Id} - Q_\Theta]$ which is quite expensive. A cheaper, but still reasonable stopping criterion is the scaling factor λ which only depends on γ (and K), i.e., we stop once $\|\tilde{\mathbf{p}}_\Theta\|_1 = \lambda$ is sufficiently close to one. This also means that the convergence rate of the uniformization method highly depends on the quality of the upper bound γ, i.e., we prefer a γ which is equal to or greater than but very close to $\max_{\mathbf{x}\in S} |Q_\Theta[\mathbf{x},\mathbf{x}]|$. On the other hand γ should be reasonable cheap to calculate. Calculating γ exactly using all diagonal elements of Q_Θ requires $\mathcal{O}(d2^d)$ arithmetic operations. This exponential growth in computational complexity needs to be avoided. Instead, we estimae the upper bound γ by evaluating the maximum absolute diagonal entry $\gamma_i = \max_{\mathbf{x}\in S} |Q_i[\mathbf{x},\mathbf{x}]| > 0$ of each Q_i given by

$$\gamma_i = \Theta_{ii} \prod_{j=1, j\neq i}^{d} \max\left\{\Theta_{ij}, 1\right\}. \tag{5.7}$$

An upper bound on the maximum value of the diagonal entries of Q_Θ is then given by

$$\gamma = \sum_{i=1}^{d} \gamma_i \geq \max_{\mathbf{x}\in S} |Q_\Theta[\mathbf{x},\mathbf{x}]|. \tag{5.8}$$

This only requires $\mathcal{O}(d)$ arithmetic operations for each γ_i or $\mathcal{O}(d^2)$ in total. However, this is only a good estimation for an upper bound of the absolute diagonal entries of Q_Θ if most off-diagonal elements of Θ are less than or equal to one, or at least close to one.

Some adjustments are required to make the Alg. 5.1 usable for TT as well. In fact, we not only insert the rounding operation at appropriate places to maintain low ranks, but also apply simple transformations using (2.1) and (5.8) to (5.5):

$$\begin{aligned}\mathbf{p}_\Theta \approx \tilde{\mathbf{p}}_\Theta &= \frac{1}{1+\gamma}\sum_{k=0}^{K}\left(\frac{\gamma}{1+\gamma}\right)^k \left[\mathrm{Id} + \frac{1}{\gamma}Q_\Theta\right]^k \mathbf{p}_\emptyset \\ &= \frac{1}{1+\gamma}\sum_{k=0}^{K}\left(\frac{\gamma}{1+\gamma}\right)^k \left[\frac{1}{\gamma}\sum_{i=1}^{d}(\gamma_i\,\mathrm{Id} + Q_i)\right]^k \mathbf{p}_\emptyset \qquad (5.9)\end{aligned}$$

Algorithm 5.2: TT-Uniformization method

```
Input: Transition rate matrix $Q_\Theta \in \mathbb{R}^{(\times^d 2)\times(\times^d 2)}$, probability distribution
       $\mathbf{p}_\emptyset \in \mathbb{R}^{\times^d 2}$, $\gamma_i \geq \max_{\mathbf{x}\in S}|Q_i[\mathbf{x},\mathbf{x}]|\ \forall i = 1,\dots,d$, and accuracies $\varepsilon$ and $\delta$
Output: Probability distribution $\mathbf{p} \in \mathbb{R}^{\times^d 2}$ with $\|[\mathrm{Id} - Q_\Theta]\,\mathbf{p} - \mathbf{p}_\emptyset\| \leq \varepsilon\,\|\mathbf{p}_\emptyset\|$
1  Initialize $k = 0$, $s = 1$, $\lambda = 1$, $\gamma := \sum_{i=1}^d \gamma_i$, $p_0 = \mathbf{p}_\emptyset$, and $q = \mathbf{p}_\emptyset$
2  while $\|\lambda^{-1}\,[\mathrm{Id} - Q_\Theta]\,p_k - \mathbf{p}_\emptyset\| > \varepsilon\,\|\mathbf{p}_\emptyset\|$ do
3          $s := \frac{\gamma}{1+\gamma}\,s$
4          $\lambda := \lambda + s$
5          $q := \gamma^{-1}\,\mathcal{R}_\delta\left(\sum_{i=1}^{d}\mathcal{R}_\delta(\gamma_i\,q + Q_i\,q)\right)$
6          $q := \|q\|_1^{-1}\,q$                                          // $\|q\|_1 = 1$
7      $p_{k+1} := \mathcal{R}_\delta(p_k + s\,q)$
8      $p_{k+1} := \lambda\,\|p_{k+1}\|_1^{-1}\,p_{k+1}$                          // $\|p_{k+1}\|_1 = \lambda$
9          $k := k + 1$
10 return $\mathbf{p} := \lambda^{-1}\,p_k$
```

The resulting uniformization method adapted for TT is shown in Alg. 5.2. Note that we restore the correct 1-norm after each approximation. However, it is not clear if this actually is an improvement. After an approximation the 1-norm is not correct anymore, but the approximation is not necessarily improved by correcting this fact. Actually, depending on the used metric, the approximation might even get worse. In fact, a correct norm for the intermediate steps is not implicitly necessary, only the final result must have the correct norm. The alternative approach is to omit lines 6 and 8 and instead, only correct the 1-norm of the final result. For the calculation of q in line 5 we can either use the rounding operation described in Sec. 4.10 or one of the methods for fast approximated linear algebra described in Sec. 4.12. When using the former, the computational complexity for extending the series by one term is given by $\mathcal{O}\Big(dn\,\Big(\big(dr^{(q)}\big)^3 + \big(r^{(p)} + r^{(q)}\big)^3\Big)\Big)$ with the maximum ranks $r^{(q)}$ of q and $r^{(p)}$ of p.

Unfortunately, Alg. 5.2 has a weak spot in line 7. There we are effectively calculating a sum of K summands. We have already mentioned previously that when evaluating a sum it is recommended to approximate only the final result to keep the error low. This is especially true if the summands differ significantly in magnitude, which is the case here. However, it is difficult to only approximate the final result in this case. To do so, we would have to store all intermediate results of the recursively calculated product $\left(\frac{\gamma}{1+\gamma}\right)^k \left[\frac{1}{\gamma} \sum_{i=1}^d (\gamma_i \operatorname{Id} + Q_i)\right]^k \mathbf{p}_\emptyset$. This is not a feasible option, i.e., we have to stick with approximating after each addition.

Once $\mathbf{p}_\Theta$ has been computed we can evaluate the marginal log-likelihood score function

$$\mathcal{S}_\mathcal{D}(\Theta) = \langle \mathbf{p}_\mathcal{D}, \log \mathbf{p}_\Theta \rangle . \tag{2.3}$$

The element-wise logarithm of $\mathbf{p}_\Theta$ can not be calculated directly in the TT format. However, $\mathbf{p}_\mathcal{D}$ is not given in TT format anyway, i.e., we need a workaround to compute the scalar product of a sparse full tensor and a TT tensor. A possible workaround is naturally given by the sparsity of $\mathbf{p}_\mathcal{D}$:

$$\mathcal{S}_\mathcal{D}(\Theta) = \sum_{\mathbf{x} \in \mathcal{D}} \mathbf{p}_\mathcal{D}[\mathbf{x}] \, \log \mathbf{p}_\Theta[\mathbf{x}] \tag{5.10}$$

where $\mathbf{x} \in \mathcal{D}$ means all multiindices $\mathbf{x}$ for which $\mathbf{p}_\mathcal{D}[\mathbf{x}] \neq 0$. This is a feasible workaround as long as the number of non-zero entries in $\mathbf{p}_\mathcal{D}$ is reasonable low. Note that the computational complexity for each single summand $\mathbf{x} \in \mathcal{D}$ is dominated by the evaluation of $\mathbf{p}_\Theta[\mathbf{x}]$ which requires $\mathcal{O}(dr^2)$ arithmetic operations. Fortunately, the evaluation of $\mathbf{p}_\Theta[\mathbf{x}]$ for each $\mathbf{x} \in \mathcal{D}$ is independent of each other, i.e., we can trivially parallelize this task with a very high efficiency.

In Sec. 5.2 we discussed the issue of negative entries appearing in a probability distribution due to approximations in the TT format. This applies here to $\mathbf{p}_\Theta$ and has serious consequences as the logarithm of a non-positive number is undefined. We propose to use a cutoff to deal with this problem, i.e., we replace $\mathbf{p}_\Theta[\mathbf{x}]$ in (5.10) by $\max \{\mathbf{p}_\Theta[\mathbf{x}], \varepsilon\}$ where $0 < \varepsilon \in \mathbb{R}$. The justification for this workaround is that if $\mathbf{p}_\Theta[\mathbf{x}] \ll \mathbf{p}_\mathcal{D}[\mathbf{x}] > 0$ for one or multiple $\mathbf{x} \in \mathcal{D}$ then the current parameters Θ_{ij} are certainly not optimal. Thus the exact score does not matter as long as the affected $\mathbf{x}$-th summands add a sufficiently large negative penalty to it to allow the optimizer to progress further. For this to work ε must be significantly smaller than $\min_{\mathbf{x} \in \mathcal{D}} \mathbf{p}_\mathcal{D}[\mathbf{x}]$. This might be an issue for future data sets as with increasing d and number of data points, the number of non-zero entries in $\mathbf{p}_\mathcal{D}$ also increases. As a result, the non-zero entries of $\mathbf{p}_\mathcal{D}$ become smaller and smaller due to the condition $\|\mathbf{p}_\mathcal{D}\|_1 = 1$. Once the parameters Θ_{ij} are close to the optimum the issue is less likely to occur since then $\mathbf{p}_\Theta[\mathbf{x}] \approx \mathbf{p}_\mathcal{D}[\mathbf{x}] \;\; \forall \mathbf{x}$. Note that this indeed is also a problem when not using any approximations at all: For certain $\mathbf{x} \in \mathcal{D}$, $\mathbf{p}_\Theta[\mathbf{x}]$ may be equal to zero even though $\mathbf{p}_\mathcal{D}[\mathbf{x}] \neq 0$.

5.4 Gradient of the score function

Using classical algorithms for the optimization of $\mathcal{S}_{\mathcal{D}}(\Theta)$ it is also required to calculate its gradient, i.e., calculate its partial derivatives with respect to each parameter Θ_{ij} given by

$$\frac{\partial \mathcal{S}_{\mathcal{D}}(\Theta)}{\partial \Theta_{ij}} = \left\langle \left[\mathrm{Id} - Q_{\Theta}\right]^{-T} \left(\mathbf{p}_{\mathcal{D}} \oslash \mathbf{p}_{\Theta}\right), \frac{\partial Q_{\Theta}}{\partial \Theta_{ij}} \mathbf{p}_{\Theta} \right\rangle \tag{2.6 \& 2.7}$$

The element-wise division of two TT tensors can not be calculated directly in the TT format and $\mathbf{p}_{\mathcal{D}}$ is not given in the TT format. Again, we use a similar workaround given by the sparsity of $\mathbf{p}_{\mathcal{D}}$ as for the calculation of the score function itself in (5.10):

$$\frac{\partial \mathcal{S}_{\mathcal{D}}(\Theta)}{\partial \Theta_{ij}} = \sum_{\mathbf{x} \in \mathcal{D}} \frac{\mathbf{p}_{\mathcal{D}}[\mathbf{x}]}{\mathbf{p}_{\Theta}[\mathbf{x}]} \left[\mathrm{Id} - Q_{\Theta}\right]^{-T} \mathrm{e}_{\mathbf{x}} \frac{\partial Q_{\Theta}}{\partial \Theta_{ij}} \mathbf{p}_{\Theta} \tag{5.11}$$

where $\mathrm{e}_{\mathbf{x}} \in \mathbb{R}^{\times^d 2}$ is a single-entry TT tensor with $\mathrm{e}_{\mathbf{x}}[\mathbf{x}] = 1$ and all other entries equal to zero. The tensor $\mathrm{e}_{\mathbf{x}}$ can be represented as a TT tensor with rank one and cores $\mathrm{e}_{\mathbf{x}}^{[k]} = \mathrm{e}_{\mathbf{x}_k}$ where $\mathbf{x}_k$ is the k-th index of the multiindex $\mathbf{x}$. Unfortunately, this means that we have to solve a linear system

$$\left[\mathrm{Id} - Q_{\Theta}\right]^{T} \mathbf{q}_{\mathbf{x}} = \mathrm{e}_{\mathbf{x}} \tag{5.12}$$

for each $\mathbf{x} \in \mathcal{D}$. Note that we can not use the uniformization method described in Sec. 5.3 to solve this linear system as Q_{Θ}^{T} is not a transition rate matrix. Fortunately, we can solve these linear systems independent of each other, i.e., we can trivially parallelize this task with a very high efficiency. In addition, we do not have to calculate $\mathbf{p}_{\Theta}[\mathbf{x}]$ twice, once for the score function and once for the gradient, as the score function and the gradient usual have to be both evaluated for the same parameters Θ_{ij}, i.e., the loops over $\mathbf{x} \in \mathcal{D}$ in (5.10) and (5.11) can be fused.

The problem of negative or zero entries in $\mathbf{p}_{\Theta}$ can be solved for (5.11) by simply omitting affected summands.

5.5 Optimizer

While not part of the main topic, let's briefly consider the choice of a suitable optimizer for the log-likelihood score (2.3). This is important because this choice directly affects the total execution time as it determines how often we need to solve (2.3) and (2.6). However, not all optimizers are applicable to this problem.

One reason is the boundary condition of the parameters $\Theta_{ij} > 0$ which needs to be ensured at any time, i.e., only optimizers which support bound constraint are suitable. L-BFGS-B [60] is a common choice for such problems. However, L-BFGS-B, like many other optimizers which support bound constraints, needs to be combined with a complex line search method supporting bound constraints as well. Usually L-BFGS-B is combined with the line search routine of Moré and Thuente [61]. This line search is not only complex to implement, but also tends to require more evaluations of the score function and its gradient than line search methods for unconstrained optimization problems.

Fortunately, we can easily avoid the boundary condition by not optimizing Θ_{ij} but $\vartheta_{ij} = \log(\Theta_{ij}) \in \mathbb{R}$. When evaluating the score function or its gradient we simply convert from ϑ_{ij} to θ_{ij} and vice versa whenever needed. Note that we then need to calculate the partial derivative of $\mathcal{S}_{\mathcal{D}}(\vartheta)$ with respect to each parameter ϑ_{ij}. This parameter transformation allows us to, e.g., use BFGS [62], L-BFGS [63, 64], or a variant of gradient descent (GD), cf. [65], combined with efficient line search routines ensuring the *Wolfe* [66, (3.6)] or even *strong Wolfe* conditions [66, (3.7)].

Note that L-BFGS, a variant of BFGS with reduced memory requirements by lowering the accuracy of the approximated inverse Hessian matrix compared to BFGS, has no real advantages over BFGS in this case since the memory requirements of BFGS of $\mathcal{O}(d^2)$ are negligible for for current data sets, but may even be disadvantageous due to the less accurate approximation. Hence BFGS is preferred over L-BFGS in this case.

The other reason why not all optimizer are applicable to this problem is the added penalty term in (2.5). This term is not differentiable at any point, i.e., the objective function including this term is non-differentiable. This is, e.g., an issue for GD and BFGS as these optimizers only have guaranteed convergence for differentiable functions. A variant of the L-BFGS designed in particular for such penalty terms is the Orthant-wise limited-memory quasi-Newton (OWL-QN) [67, 68] method. However, this algorithm has not yet been implemented in our software. Fortunately, BFGS works very well despite the non-differentiability and has indeed been used in [2].

A common approach in machine learning to reduce the time to solution is to not calculate the actual gradient using the entire data set but only an estimate thereof using a randomly selected subset. The size of the subset can be dynamically increased to ensure that the exact optimum is reached once close to it while allowing fast computation of the estimated gradient in the early phase. In the early phase an estimation of the gradient is often sufficient to still drive the solution towards the global optimum. Note that while it seems that simply replacing non-zero entries of $\mathbf{p}_{\mathcal{D}}$ in (2.7) by zero with a certain probability, instead of randomly selecting a subset of the data set and creating $\mathbf{p}_{\mathcal{D}}$ a new, is sufficient it is indispensable to ensure that the modified $\mathbf{p}_{\mathcal{D}}$ is a probability distribution afterwards, i.e., $\|\mathbf{p}_{\mathcal{D}}\|_1 = 1$. The concept of only using an estimate of the gradient can be applied to various optimizers. These variants often carry the prefix *stochastic* or *online*, cf. [69]. Having in mind the issue of non-differentiability a stochastic variant of OWL-QN seems most promising, cf. [70].

Chapter 6

Software Implementation

When we started working on the topic, we first looked for and evaluated existing implementations of the TT format. The TT-Toolbox [71] is the de facto reference implementation. However, this implementation is not developed with a focus on performance, but to develop and test new algorithms. In addition, the TT-Toolbox is based on Matlab [72] which is a proprietary programming language. Our focus is to develop open source software that can be used independently of any proprietary solutions. Later an effort to rewrite the TT-Toolbox in Python [73] with its computing core implemented in Fortran [74] has been started. However, this attempt was not necessarily successful and the development stalled. Other libraries implementing TT are often tailored towards a specific field of use, e.g., [75, 76, 77] for Neural Networks and [78, 79, 80, 81, 82] for quantum physics. [83, 84, 85] are examples for implementations implementation which try to be generic enough to be used for different use cases. Unfortunately, development of some of these libraries appears to have stalled or even been discontinued. None of the listed libraries was considered a viable choice either due to the high specialization, current state of development, or some have simply not yet been publicly available at the time. Instead, we decided to implement our own library in C++17 heavily relying on template metaprogramming (TMP). We not only implemented the TT format ourselves, but also developed our own underlying tensor library. The tensor libraries of Boost [86] and Eigen [87] have been evaluated, but again neither was found to be a viable choice.

Implementing a library from scratch many design decisions have to be made. Some of these have a major impact on how and for what the library can be used. One major design decision of our library was to not allow dynamically changing the mode sizes of a tensor but fix mode sizes at compile time. While this limits the flexibility it dramatically lowers the amount of required dynamic memory allocations and deallocations. This decision was made specifically for the MHN model as the mode sizes of all involved tensors are two, i.e., the cores of a TT tensor or operator are small. In terms of C++ this means that the mode sizes define the tensor's class type, i.e., tensors are implemented as a class template with template parameters specifying the mode sizes.

In this chapter we will first explain required programming features to implement the tensor class with an arbitrary number of mode sizes set at compile time. Next we will discuss some limitations imposed by this design decision and especially the impact on the algorithms introduced in Chapter 4. Afterwards we describe one particular generic feature of the library. Finally we explain how to specialize function templates depending on type properties up to C++17 and show how this can be simplified in C++20.

As of writing this work, our library consisted of ∼24 000 lines of code using the C++17 standard revision depending on one feature [88] from C++20. The release of the software under an open source license is planned for after the completion of the dissertation.

6.1 Parameter Pack

For the implementation of the class template `Tensor` representing a (full) tensor in $\mathbb{K}^{\times_k^d n_k}$, where $\mathbb{K}$ can either be $\mathbb{R}$ or $\mathbb{C}$, we need to be able to specify an arbitrary number of mode sizes using non-type template parameters. For that we use a non-type template *parameter pack* introduced in C++11 [89, 90], i.e., the class declaration is given by **`template<class T, Size... Ns> class Tensor;`** where `...` specifies that `Ns` is a parameter pack representing an arbitrary number of non-type template arguments of type `Size`.[1] `Size` itself is a type alias referring to an unsigned integer type, usually `std::size_t`. The template parameter `T` is used to specify the data type of each element of the tensor, i.e., it is used to specify whether it is a real or complex tensor and to set the floating-point precision with the default being double precision. In C++11 parameter packs are essentially restricted to expansions using `...`. This means we have to use recursive function calls to be able to deal with one element of `Ns` at a time.

An example using pack expansion is shown in Code 6.1.[2] In this example we calculate the linearized size, i.e., the total amount of elements, of the tensor. The calculation requires two helper functions which are both called `helper` but differ in parameters. The version without any parameters is required to end the recursive function call in line 6.[3] This is quite a lot of code for such a simple task and hence motivated the addition of *fold expressions* [93, 94] in C++17. These allow us to reduce, i.e., fold, a parameter pack over a binary operator with an optional initial value at the beginning or at the end. The Code 6.1 rewritten with fold expressions is much easier to read, write, and understand, see Code 6.2.[4]

[1]A template with at least one parameter pack is called a *variadic template*.

[2]Note that in this example most functions are marked as *constexpr functions* using the `constexpr` specifier. These functions can be evaluated at compile time and thus be used where only *constant expressions*, i.e., expressions which can be evaluated at compile time, are allowed. However, constexpr functions are not required to be evaluated at compile time. In C++20 *immediate functions* [91] have been introduced. These are required to produce a compile time constant. If a function is evaluated at compile time it makes no difference whether it is a constexpr or immediate function.

[3]The two versions of the `helper` function can be combined into one version using compile time conditions [92], also known as *constexpr if*, introduced in C++17.

[4]Note that we have simplified both Code 6.1 and 6.2 to focus on the features discussed.

Code 6.1: Parameter pack expansion [C++11]

```cpp
constexpr Size helper() {
    return 1;
}
template<class T, class ...Ts>
constexpr Size helper(T const N, Ts const... Ns) {
    return N * helper(Ns...);
}
template<class T, Size... Ns>
struct Tensor
{
    static constexpr Size size()
    {
        return helper(Ns...);
    }
};
```

Code 6.2: Code 6.1 rewritten with fold expressions [C++17]

```cpp
template<class T, Size... Ns>
struct Tensor
{
    static constexpr Size size()
    {
        return Ns * ... * 1;
    }
};
```

Code 6.3: Simultaneous parameter pack expansion [C++11]

```cpp
template<class T, Size... Ns>
struct Tensor
{
    template<Size... Offsets, class ...Is>
    T& periodicShiftByOffset(Is const ...indices)
    {
        return operator()((Offsets + indices) % Ns...);
    }
};
```

In Code 6.3 we show a snippet of a possible extensions of the `Tensor` class template using a simultaneous parameter pack expansion in line 7. Simultaneous parameter pack expansions require all parameter packs to be of the same length.

Although we can implement all required functionality in C++17, some features still require quite a lot of boilerplate code or are cumbersome to implement. Here we only want to mention three cases:

1. Storing a parameter pack, e.g., as a class member, is not directly possible. The usual solution is to store a `std::tuple` constructed from it instead.
2. Accessing a particular element of a parameter pack by index requires to, e.g., first construct a `std::tuple` from it and then access the element using `std::get`.
3. In Code 6.3 line 4 we need to add checks that all `Is` are of appropriate type, e.g., `Size`, and the length of `Is` is the same as that of `Ns`.[5]

Fortunately, C++ is a continuously evolving programming language which attempts to fix such inconveniences in new revisions of the C++ standard. A proposal which covers all three use cases has already been developed [95] and proposed to the C++ Standards Committee for inclusion in a future C++ standard revision.

6.2 Fixed Size Limitations

The design decision of the library to not allow dynamically changing the mode sizes of a tensor but fix all mode sizes at compile time imposes some limitations on the implementation of the Tensor Train format. Put simply, this means that all mode sizes and ranks, and hence the number of dimensions, of a TT tensor have to be set at compile time. In the case of MHN, determining the mode sizes and dimensions at compile time is not a big issue. All mode sizes are two and the number of dimensions is fixed for a particular data set. However, the requirement to also fix the ranks at compile time means that these can not be adapted dynamically. In particular this affects the rounding algorithm presented in Sec. 4.10 and all rank adaptive variants of the ALS algorithm presented in Sec. 4.11 and their adapted versions for fast approximated linear algebra outlined in Sec. 4.12.

For the rounding algorithm we need to modify the truncation part, see Alg. 4.4. In addition to an accuracy ε we have to set the ranks r_k of the truncated TT tensor as input. Usually we only set the maximum rank r instead of all individual ranks as this is desired in almost all cases anyway. Having to specify each rank r_k only makes the applications unnecessarily complicated. All ranks r_k of the resulting TT tensor are then either set to r or lower due to the conditions imposed on the ranks by the left-to-right ($r_k \leq r_{k-1}n_k$) and right-to-left ($r_k \leq r_{k+1}n_{k+1}$) orthogonalization or truncation. We use the given or determined ranks r_k to set the number of singular values kept in the truncated SVD, i.e, we replace all δ-truncated SVDs by r_k-truncated SVDs. The given

[5]For parameter pack `Offsets` we also need to check the length but not the type.

accuracy ε, respectively δ, is used to emit a warning, including the actual error, if the desired accuracy is not met. However, it has no effect on the actual ranks. In contrast to the accuracy ε, which is a function argument, the rank r has to be specified as a template argument in C++. Accordingly, for all algorithms which use the rounding operation internally, a maximum rank must be specified in addition to the accuracy. This applies, for example, to Alg. 4.7 and 5.2.

All rank adaptive variants of the ALS algorithm use δ-truncated SVDs with a given accuracy δ internally. As with the rounding method, all δ-truncated SVDs have to be replaced by r_k-truncated SVDs. Again, we only set a maximum rank r and determine the ranks r_k according to the rank conditions imposed by the orthogonalization. This means that the ranks of the solution x have to be set a priori for all variants of ALS.

The requirement to set all ranks a priori can lead to ranks either being set too low or too high. If we underestimate the ranks we may not maintain the desired accuracy. If we overestimate the ranks the required computational effort will be higher than actually necessary. These disadvantages come with a potential benefit in performance. In addition, there is a proposal to add just-in-time (JIT) compilation to C++ [96] which could help to at least mitigate some of these drawbacks.

6.3 Lazy Evaluation

To motivate lazy evaluation, let's consider a simple example in tensor arithmetics: Given tensors a and b of same shape the sum $d = a + b$ is defined by element-wise addition, i.e., in C++ we would implement an overloaded `operator+` which returns a new object of type tensor. This function involves a loop over all indices of d. However, when we consider the more complex, but still simple, expression $d = a + b + c$ the overloaded `operator+` returning a tensor object has to be called twice: The implementation first creates a new temporary tensor $t = a + b$ which is then summed up with c to form d calling `operator+` a second time. This is obviously not efficient as it involves two loops over the same indices and an unnecessary temporary object. The obvious optimization is to fuse the two loops and directly form d. However, the question is how to implement this in a generic way without having to define a function for every possible expression.

Delayed, or lazy evaluation solves this optimization problem. In C++ this can be implemented by having the `operator+`, or any other arithmetic operator, return an object of a class which represents the unevaluated operation of two, or any number of, tensors. Objects of classes representing unevaluated operations on tensors can themselves be used in expressions. This effectively leads to the construction of expression trees in case of compound expressions. These expression trees are only evaluated once assigned to an actual tensor object or evaluation is explicitly requested. An exception are expressions containing temporary objects. These must be evaluated immediately since the temporary objects are no longer available after the expression. This points to a pitfall of this concept: The user must ensure that all objects referenced in an expression tree still exist at the time of evaluation. Lazy evaluation is particularly efficient for element-wise operations. Expression templates [97] are a reasonable implementation of this concept in C++. These are, for example, used in the two probably most well-known libraries for Lattice QCD, Chroma [98] and Grid [99].

In our library we go one step further and not only delay the evaluation, but also allow partial evaluation of expressions. This means that if only individual entries of a tensor are required, only these are actually evaluated. Which entries are to be computed can either be determined by using functions which change the view on the tensor, e.g., reducing the dimension of the tensor by fixing the index of a particular mode, or access individual entries manually. Hence, we refer to classes representing unevaluated expressions as *views*. A very similar approach is used in Eigen [87].

This concept has been applied to tensors in our library as described above and additionally to TT tensors and operators. In case of TT tensors or operators, lazy evaluation is based on the loop over its cores instead of entries. This works very well for many basic arithmetic operations in the TT format as they are computed core-wise. When calculating the individual cores, the lazy evaluation of tensors comes into play.

6.4 Function Template Specialization

Dealing with many different class templates we need a way to define template functions depending on the arguments' type properties. In C++ properties of types are queried using *type traits*.[6] Basic type traits, e.g., to query whether a basic type is signed or unsigned, are provided by the metaprogramming library part of the C++ standard library. Additional type traits, especially to query properties of user-defined types, can be defined as required. This section is called “function template specialization”, but actually we will technically not specialize any function template. In C++ only explicit (full) function template specializations are allowed. However, these are of low interest since we usually want to define functions for a group of types with certain properties and not a for a specific instantiation of a class template. Instead, we define function templates as needed and then conditionally disable these in function overloading based on type traits. This allows separate function overloads, in other words *specializations*, based on type properties.

Since C++11[7], this can be implemented by exploiting the “Substitution Failure Is Not An Error” (SFINAE) rule, see Code 6.4.[8] This rule applies during overload resolution of function templates and states that if substitution of the explicitly specified or deduced type for a template parameter fails, then the function template is ignored in overload resolution rather than resulting in an error.

[6]Other type traits can be used to modify types, e.g., to remove `const` specifier from a given type.

[7]Strictly speaking this was already possible in C++98, but was much more cumbersome to implement.

[8]The alias template `If` is only used to improve readability.

Code 6.4: Conditional function template overloading using SFINAE [C++11]

```
template<class B>
using If = typename std::enable_if<B{}, int>::type;

template<class T, If<std::is_signed<T>> = 0>
T abs(T const &);

template<class T, If<std::is_unsigned<T>> = 0>
T abs(T const &);
```

Code 6.5: Template function specialization using compile time conditions [C++17]

```
template<class> constexpr bool dependentFalse = false;

template<class T>
T abs(T const &) {
    if constexpr (std::is_signed_v<T>) {...}
    else if constexpr(std::is_unsigned_v<T>) {...}
    else static_assert(dependentFalse<T>, "Unsupported type.");
}
```

Another approach is to use compile time conditions [92], also known as *constexpr if*, introduced in C++17. However, this means that either all function definitions or at least references to other function templates, with different names, for all possible types need to be part of a single function template definition. This quickly leads to very messy code. Hence it is only recommended to use compile time conditions for this purpose if the number of case distinctions is very small.

Code 6.6: Conditioal function template overloading using constraints [C++20]

```
template<class T> requires std::is_signed_v<T>
T abs(T const &);

template<class T> requires std::unsigned_integral<T>
T abs(T const &);
```

Constraints [100] were introduced in C++20 as a proper alternative to SFINAE. We may associate function (or class) templates with a constraint which specifies requirements on template arguments. These requirements are then used to select the most appropriate function overload. This means that constraints are actually designed for conditional function template overloading and are not abused for an unintended purpose. Accordingly, the syntax is very clean and easy to read, see Code 6.6. A big advantage of using constraints instead of SFINAE is that violations are detected early in the template instantiation process which allows for easy to follow compiler diagnostics in case of an error. Very often the term concepts is used instead of constraints. However, a *concept* is actually a named set of requirements which can be used in constraints, i.e., concepts are the equivalent of type traits while constraints are the equivalent of SFINAE in this context. Indeed, both concepts and type traits can be used with constraints to specify requirements. In Code 6.6, `std::is_signed`[9] is a type trait and `std::unsigned_integral` is a concept. As of writing this work, concepts and constraints are not yet being used in our library, but it is planned to use both. However, this means a lot of code changes in the entire library and raising compiler requirements. Accordingly, a wide availability of compilers supporting concepts and constraints should be awaited.

[9]`std::is_signed_v`, which is identical to `std::is_signed<T>::value`, is a helper variable template to simplify the syntax. Such helpers with suffix `_v` are defined for all type traits in the standard library.

Chapter 7

Numerical results

In the previous chapters we have seen that we have to solve the linear systems

$$[\mathrm{Id} - Q_\Theta]\ \mathbf{p}_\Theta = \mathbf{p}_\emptyset \qquad \text{and} \qquad [\mathrm{Id} - Q_\Theta]^T\ \mathbf{q}_\mathbf{x} = \mathrm{e}_\mathbf{x} \qquad \forall \mathbf{x} \in \mathcal{D} \qquad (5.3\ \&\ 5.12)$$

in order to calculate the marginal log-likelihood score and its gradient, respectively, which are required for the optimization of Θ. Solving these linear systems of equations is indeed the greatest challenge we face in the MHN model. Once these are solved, all other calculations are relatively easy to deal with.

In this chapter we show numerical results[1] solving (5.3) using the algorithms presented in Sec. 4.11 and 5.3. We limit ourselves to showing results for (5.3) only as the difficulty to solve (5.12) varies a lot depending on $\mathrm{e}_\mathbf{x}$, but is often similar to that of (5.3). In addition, a worse accuracy is often sufficient for the gradient, in contrast to the marginal log-likelihood score. We first show results of the uniformization method since this method exploits the structure of the system and hence might be a good candidate. Next we move on to GMRES before we finally compare different variants of ALS.

Instead of choosing a particular data set we use random values for Θ. This allows us to set d to any desired number independent of available data sets. The random values are chosen such that their logarithmic values are normally distributed with mean $\mu = 0$ and standard deviation $\sigma = 0.125$ or $\sigma = 0.25$ for off-diagonal or diagonal elements of Θ, respectively. Note that while for real data sets it is assumed that many values of Θ are equal to one, we have not taken this into account. This assumption actually only applies for the global optimum, but the optimizer may choose arbitrary values for Θ in search of the optimum. Hence we only assumed that all values are around one. Note that a Θ chosen in this way represents one of the more difficult cases.

[1]All measurements were performed on an AMD Epyc 7543P using the GNU Compiler Collection's (GCC) C++ compiler version 11 with optimization level 3 and targeting the native architecture.

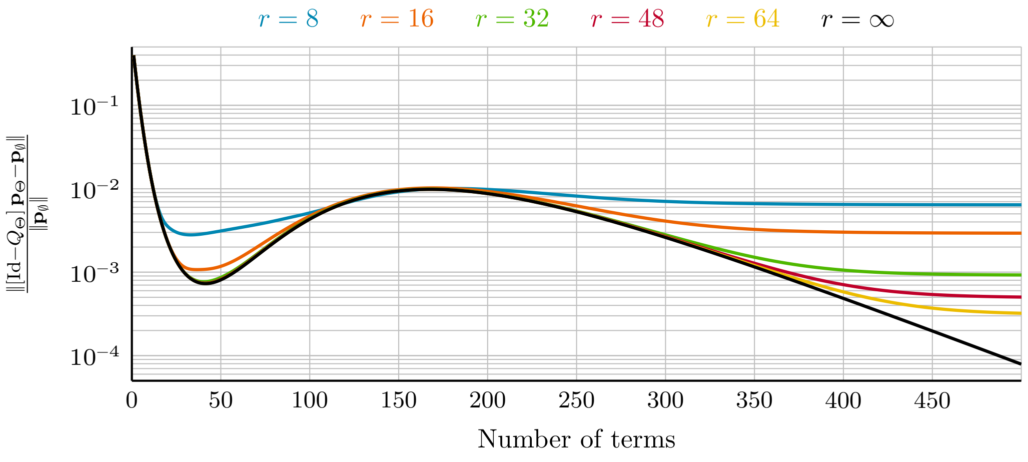

Figure 7.1: Progress of the uniformization method for $d = 20$ and different ranks r of the solution. The reference denoted by $r = \infty$ is not using the TT format.

7.1 Uniformization method

The uniformization method, see Alg. 5.2, exploits the specific structure of the MHN operator. Thus, one might assume that this is a competitive method to solve the linear system. However, the convergence rate of the method highly depends on the quality of the upper bound γ. In Fig. 7.1 the progress of the method for $d = 20$ and different ranks r of the solution is depicted. Note that the error is calculated without correcting the 1-norm of the solution or any other interim object. Remember that our software implementation requires fixed ranks, i.e., the rank r is used for approximating q and p_{k+1}, i.e., it determines the maximum rank of both. For reference the progress of the uniformization method not using the TT format is plotted as well. The shape of the reference curve, which appears strange at the beginning, can be explained by the usage of the 2-norm to calculate the relative error, although the uniformization method optimizes the 1-norm. The direct dependency between the maximum rank r and the maximum achievable accuracy of the solution is clearly visible.

The lowest relative error that can be achieved can be reduced further by increasing the maximum rank r at the expense of a longer running time. The required runtime of the uniformization method for a fixed number of terms plotted against the maximum rank r is shown in Fig. 7.2.[2] The time is normalized by a theoretically calculated execution time for $r = 1$. This is done by taking the time needed for $r = 32$ and scaling it according to the theoretical scaling in r. Since the method for fast calculation of approximated results of sums, described in Sec. 4.12, is not used for the calculation of q, the computational complexity for extending the series by one term is given by $\mathcal{O}(d^4 r^3)$, i.e., the theoretically expected runtime scaling in r is r^3. The time measurements confirm this.

[2]Runtime measurements should always be carried out several times and the corresponding deviations should be indicated. In this case, however, the error bars are so small that they are not visible anyway.

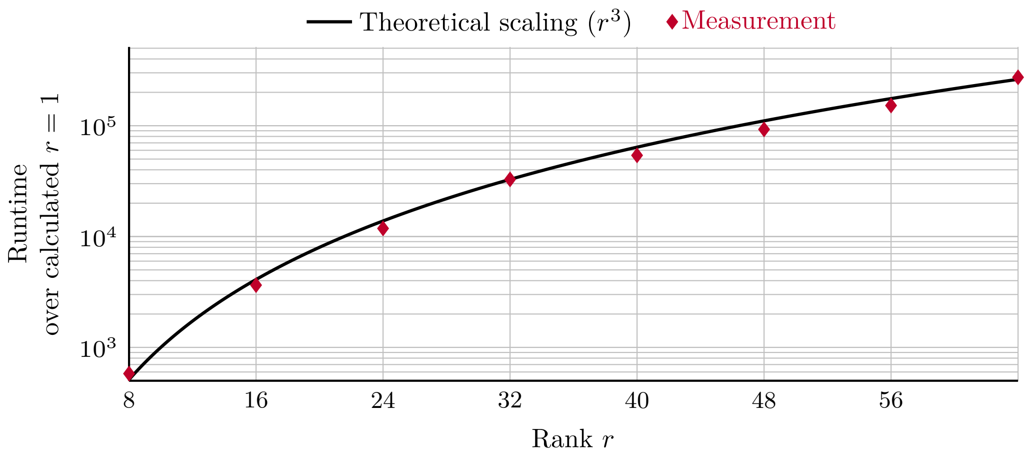

Figure 7.2: Runtime scaling of the uniformization method calculating solution $\mathbf{p}_\Theta$ with rank r and a fixed number of terms for $d = 20$.

Although the focus here is on the uniformization method adapted for the TT format, it is certainly necessary to point out the very slow progress of the method, regardless of the TT format. For example, to achieve an accuracy of 10^{-5}, the series may only be stopped after approximately 600 steps. An explanation for this very slow progress can be found in the upper bound γ. The rate of convergence of the uniformization method is given by the rate of convergence of the series

$$\frac{1}{1+\gamma}\sum_{k=0}^{\infty}\left(\frac{\gamma}{1+\gamma}\right)^k = 1. \tag{7.1}$$

For larger γ this series converges more slowly. For the given randomized Θ with $d = 20$, $\max_{\mathbf{x}\in S}|Q_\Theta[\mathbf{x},\mathbf{x}]| \approx 20$. But the exactly calculated value is usually not being used for γ as the required computational effort is of $\mathcal{O}(d2^d)$, i.e., too expensive for practical applications. Instead, we use an upper bound calculated according to (5.8), which has a lower computational complexity of $\mathcal{O}(d^2)$, but tends to overestimate the actual value. In this particular case the upper bound is given by $\gamma \approx 50$. Unfortunately, a better estimation does not exist.

7.2 Generalized Minimal Residual

The GMRES adaptation for the TT format, see Alg. 4.7, allows to set different accuracies for the approximation of the Krylov tensors and the solution. It even allows to further relax the accuracy for some operations on the Krylov tensors. Since we have to set fixed ranks for all TT tensors a priori we use the same maximum rank for all Krylov tensors, i.e., we do not use the relaxed accuracy δ_j, but always use δ. While this still allows us to set different maximum ranks for the Krylov tensors and the solution, we have used the same maximum rank r for both in the following numerical results.

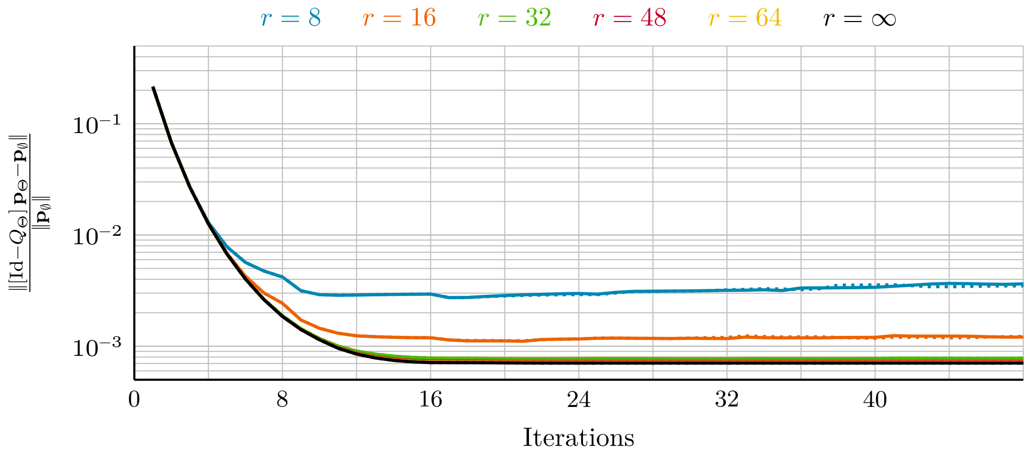

Figure 7.3: Progress of GMRES(8) for $d = 20$ and different ranks r of the solution and Krylov tensors. Both CGS (dotted lines) and MGS (solid lines) variants are shown. The reference denoted by $r = \infty$ is not using the TT format.

In Fig. 7.3 no further progress of GMRES(8) can be seen after a few iterations. This is independent of the chosen maximum rank and also affects the variant without using the TT format, i.e., further increasing the ranks is of no use as the method itself is the limit. Already starting from rank $r = 32$ there is no noticeable difference anymore to the reference. Also note that there is practically no difference between the variant using CGS and the variant using MGS for the orthogonalization. The short restart length of $m = 8$ is intentionally chosen to reduce the required computational effort required for each iteration, which when using MGS is given by $\mathcal{O}(d^4r^3 + dr^3m^2)$ when using MGS and increases to $\mathcal{O}(d^4r^3 + dr^3m^3)$ when using CGS. However, a longer restart length is needed to allow further progress of the method. The effects of the restart length m on the progress of the method for different dimensions d are shown in Fig. 7.4. The TT format was intentionally not used here to rule out possible side effects. Although no general statement can be derived from these results, a clear dependence of the minimum required restart length to ensure convergence on the dimension can be seen and setting $m \gtrapprox d$ seems to be a reasonable recommendation.

Fig. 7.5 repeats Fig. 7.3 with an increased restart length of $m = 32$. Note that for $r > 16$ we do not calculate the sum required in the correction explicitly and only approximate once afterwards, but instead also approximate interim results of the sum to speed up the calculation. In order to keep the resulting errors as small as possible we calculate the sum using a binary tree with the summands on the leaves. In fact, this has no influence on the behavior of GMRES which we would like to point out here. Looking at the reference curve we see that the relative error of the solution decreases right before and after a restart and otherwise remains almost constant. When using the TT format there is no drop before the restart, instead the relative error often even increases. The decrease before the restart can be explained by the basis of the Krylov subspace which

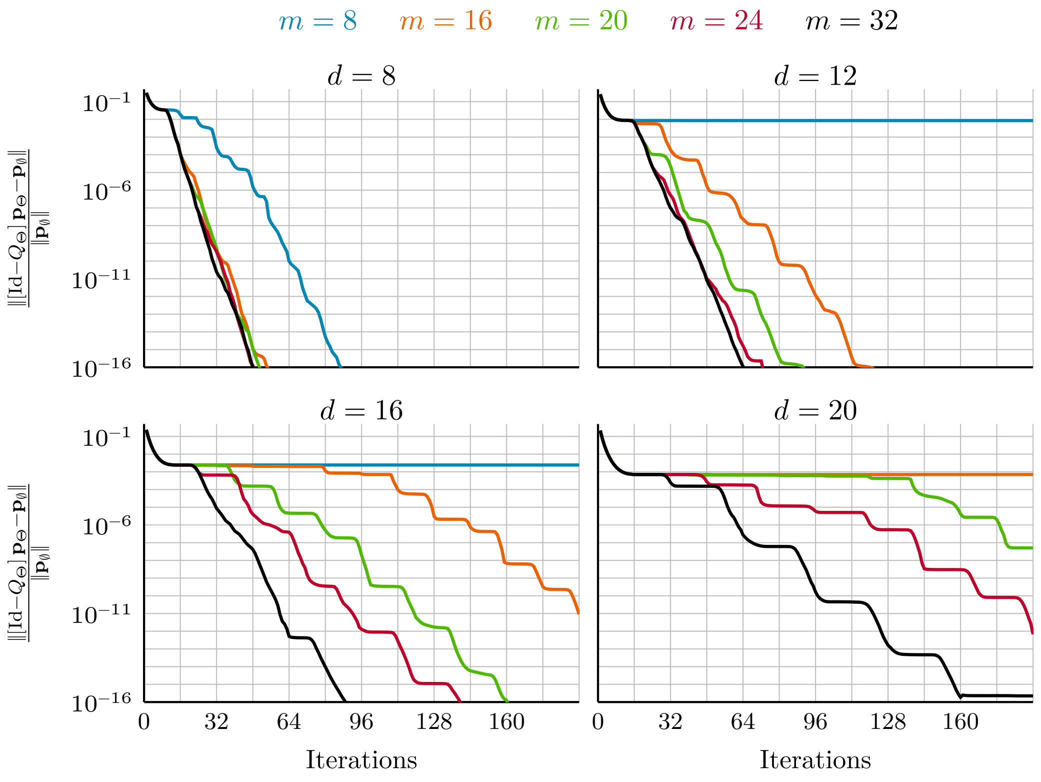

Figure 7.4: Progress of GMRES(m) with $m = \{8, 16, 20, 24, 32\}$ for $d = \{8, 12, 16, 20\}$.

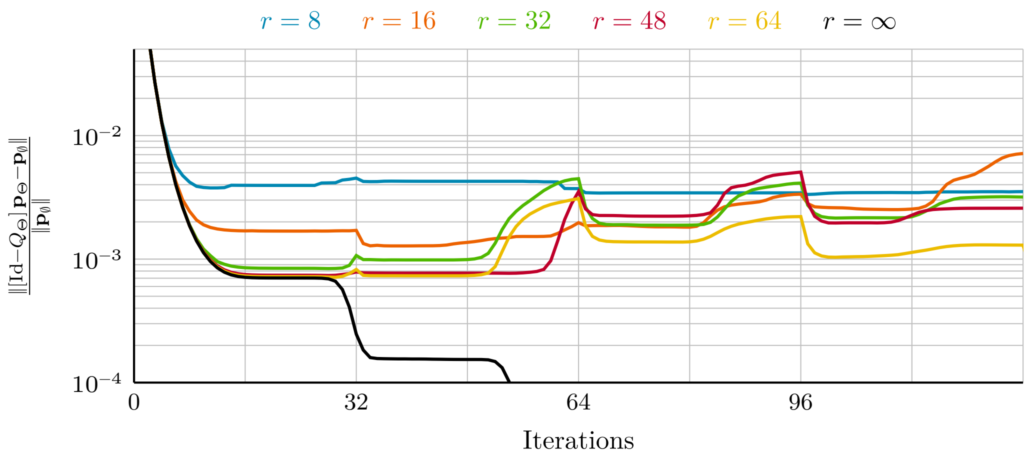

Figure 7.5: Progress of MGS-GMRES(32) for $d = 20$ and different ranks r of the solution and Krylov tensors. The reference denoted by $r = \infty$ is not using the TT format.

increases with each inner iteration and seems to be sufficiently large at this point. This clearly indicates that the Krylov tensors are calculated too imprecisely, i.e., the error introduced by the approximations is too large. This behavior when ranks are too low, i.e., when errors due to approximations are too high, is different from what has been seen before. So far, this has only restricted the best possible achievable relative error, but in this case the method simply no longer works. Hence, it is of utmost importance to choose the ranks of the Krylov tensors high enough to always ensure that their error is small enough. Note that while using CGS can reduce the error of the Krylov tensors, the method still suffers from the same problem, i.e., the only real solution to the problem is to increase the maximum rank r. In the case shown in Fig. 7.5 there is a noticeable difference between using CGS and MGS, but we only show the variant using MGS for the sake of clarity.

7.3 Alternating Linearized Scheme

The ALS method, unlike the other methods, is not a classical method adapted for the TT format, but is specifically designed for this format.[3] In Sec. 4.11 we have introduced two types of modifications that aim to improve the convergence rate and stability of the original variant. In Fig. 7.6 we compare six variants for a given dimension $d = 20$. In order to be able to compare the variants independently of the quality of the implementation, the relative error was deliberately plotted against the number of iterations and not against the runtime. The variants are ordered such that when going from left to right we add random enrichment in the first step and enrichment based on the residual in the second. Moving from the first to the second row these are then combined with using two consecutive cores to build the local systems instead of a single core. Consequently, it could be assumed that the lower right the variant is placed, the faster it converges. In any case, however, the required computational effort increases in this direction, in particular due to the composition of two cores and the additional effort required to calculate the residual. Random enrichment requires little extra computational effort.

In Fig. 7.6 improvements through enrichment and the combination of two cores are noticeable, but have limited effects. ALS without any modifications in most cases requires three left-to-right-to-left sweeps to converge or to be close to the best possible achievable accuracy for a given rank r. In comparison, some of the other variants only need two sweeps, i.e., one sweep can be saved. However, this certainly does not make up for the additional required computational effort. With MALS, compared to ALS, the estimated size of the local linear systems is by a factor of n larger. Hence the computational effort to construct and solve these is at least n^2 times higher, and thus also the estimated computational effort for one sweep according to theoretical considerations, see Sec. 4.11. In addition, there does not seem to be any problem with local minima either, i.e., one of the main reasons to introduce these modifications in the first place does not apply in case of the MHN operator. The modifications of the ALS method are therefore sim-

[3]ALS has only been derived for hermitian positive definite operators, but we apply it to (5.3) directly.

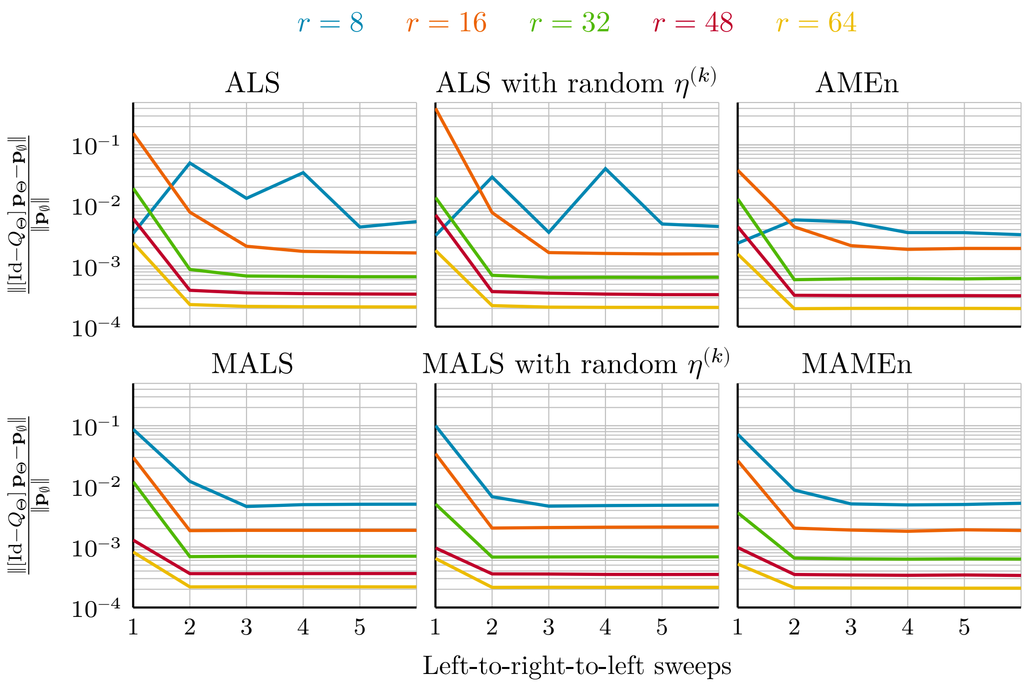

Figure 7.6: Progress of several variants of ALS for $d = 20$ and different ranks r of the solution and enrichment rank $\hat{r} = 4$, if applicable.

ply not necessary. Again, the maximum achievable accuracy improves with increasing ranks. Unfortunately, a further increase in ranks costs a lot of computing power. On the positive side, the number of required sweeps to reach the best possible accuracy for a given maximum rank does not increase with the maximum rank.

In Fig. 7.7 we see that ALS works pretty well regardless of the number of dimensions. In order to better recognize the dependency of the best achievable relative error on d and r, we also plotted the relative error against the dimension d for different ranks r of the solution in Fig. 7.8. Here we see, on the left, that from $d = 20$ to approximately 32 or 36, the best achievable relative error gets even better with higher dimensions. For dimensions $d > 36$ the best achievable error is almost constant for a given r in most cases, except for $r = 16$, in which case it deteriorates slightly in higher dimensions. Nevertheless, one can say that to a good approximation the maximum achievable accuracy depends only on the rank r and not on the dimension d. Unfortunately, sometimes instabilities occur after convergence has apparently already been achieved as can be seen in Fig. 7.7. Fortunately, these are mostly corrected again after a few sweeps. Hence it is very important to stop the method at the right time, i.e., one shall not simply stop after a pre-defined number of sweeps as the right-hand side of Fig. 7.8 also suggests. These degradations in the relative error are very difficult to detect without calculating the exact relative error after each sweep, which should be avoided since it can be very computationally expensive.

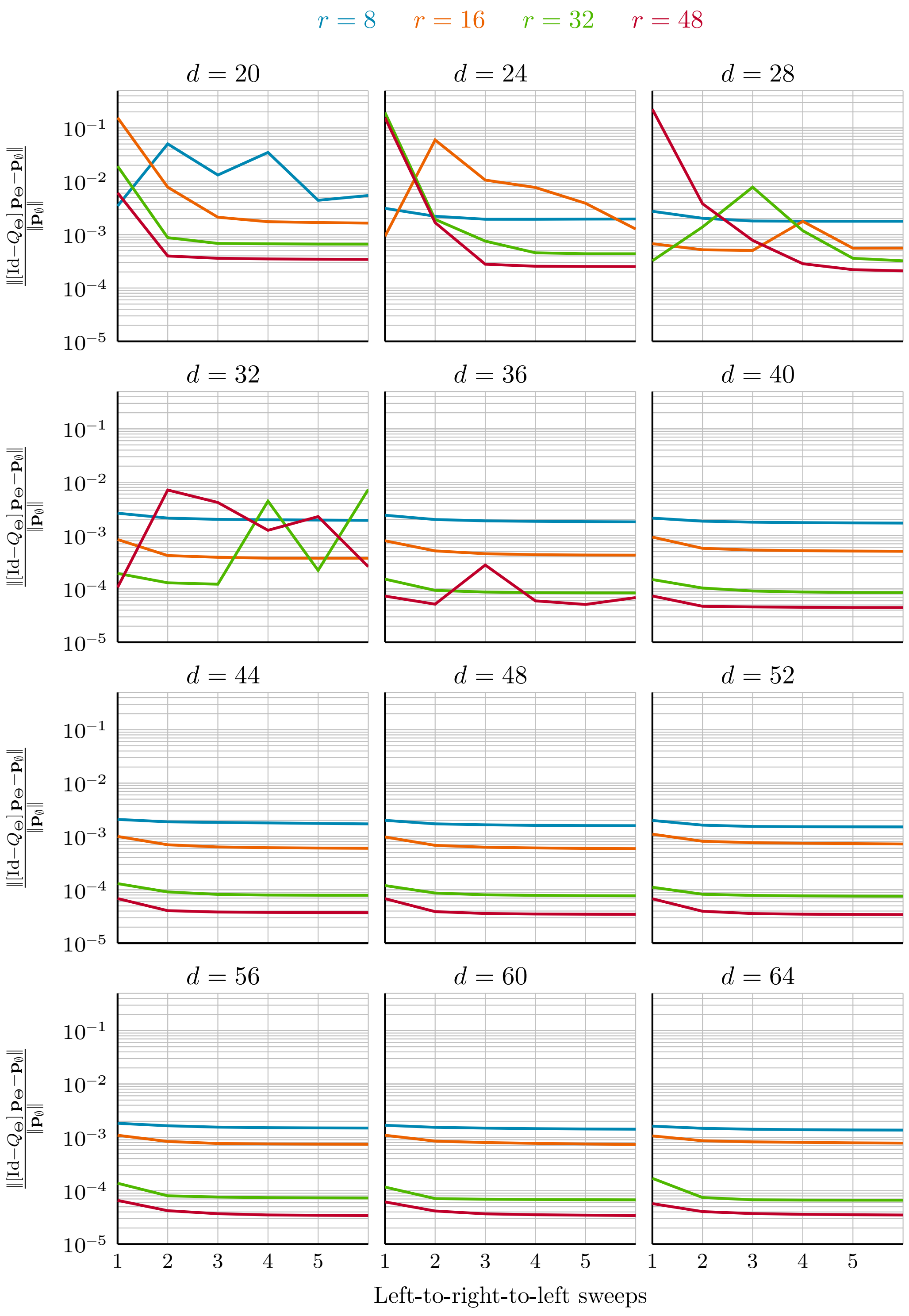

Figure 7.7: Progress of ALS for different dimensions d and ranks r of the solution.

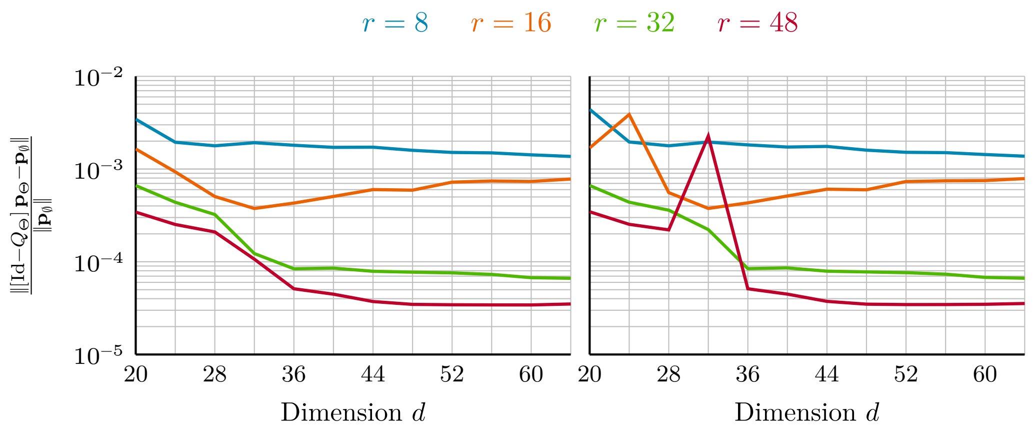

Figure 7.8: Best achieved relative error after six sweeps (left) or relative error after five sweeps (right) of ALS depending on dimension d for different ranks r of the solution.

The computational complexity of one ALS sweep is of $\mathcal{O}(d^3r^2 + d^2r^4 + cdr^4)$ where c is the average amount of matrix-by-vector products $A_k\,v$ required to solve a local linear system. In case of using GMRES as a local solver, c approximately corresponds to the number of iterations required. The computational complexity can be reduced to $\mathcal{O}(d^3r^2 + cd^2r^3)$ by not explicitly evaluating A_k, but explicitly using the auxiliary objects $\Psi_{A,k-1}$ and $\Phi_{A,k}$ and the core $A^{(k)}$ to compute $A_k\,v$. Together with the knowledge from Fig. 7.8 that, to a good approximation the maximum achievable accuracy depends only on the rank r and not on the dimension d, it follows that the complexity in d for the calculation of the marginal log-likelihood score is reduced from $\mathcal{O}(2^d)$ to $\mathcal{O}(d^3)$ by using the TT decomposition together with ALS. The best achievable accuracy of the solution is indirectly set as desired by the rank r. An increased rank r also increases the required computational effort, but is sufficiently independent of the dimension d. This is the essential step to enable the MHN model to be applied to larger data sets than ever before.

Chapter 8

Conclusion

In this thesis we briefly introduced the MHN model which mitigates the state space explosion by describing the transition rate matrix $Q_\Theta \in \mathbb{R}^{2^d \times 2^d}$ using only d^2 parameter Θ_{ij}. However, the computational effort required to obtain these parameters by optimization of the marginal log-likelihood score still scales exponentially in d when using classical methods, i.e., the model is limited to relatively small data sets. The aim of this work was to remove this limitation by using the TT decomposition which allows us to approximate high dimensional tensors with lower storage requirements. We described required operations to be performed in the TT format and presented different methods to solve systems of linear equations in this format. We then applied this to the MHN model and pointed out existing problems and possible solutions.

In the numerical results we focused on solving the linear system (5.3) required to compute the marginal log-likelihood score as it is the most crucial part in solving the MHN model. We presented results of all methods we have introduced in the previous chapters to solve this kind of linear systems, namely the uniformization method, GMRES, ALS, and variants thereof. As long as a method works, the lowest relative error that can be achieved for a given rank r is approximately the same for all methods.

The uniformization method exploits the specific form of the linear system and hence is expected to show good results. However, its convergence rate highly depends on finding an upper bound of the absolute diagonal entries of Q_Θ which is close to the actual value. It is not feasible to calculate the exact value as it requires $\mathcal{O}(d2^d)$ arithmetic operations. A possible approximation of an upper bound, which requires only $\mathcal{O}(d^2)$ arithmetic operations, is given by (5.8). Unfortunately this approximation easily tends to be an overestimation of the actual value leading to slow convergence. An advantage of the uniformization method is that it, with a small correction, takes into account that the result is a probability distribution. The other methods do not preserve this condition. However, this is only true if not using the TT format, or at least never using any approximation. Apparently this is not possible, approximations are necessary to keep the computational effort in check.

The GMRES algorithm on the other hand is often used as a black box solver as it works for many different linear systems. We have shown how to adapt this algorithm for the TT format. Both this algorithm and the uniformization method suffer from similar issues when being adapted for the TT format. A particular severe issue is error propagation. This actually applies to any method recursively calculating quantities and relying on approximating interim results to keep the ranks, and thus the computational complexity, in check. Errors introduced to a recursively calculated quantity by an approximation in one step can never be corrected in the following steps. In case of the uniformization method the solution itself and each new term are calculated recursively. In each step both are approximated introducing errors which propagate further in all of the following steps. For GMRES the same applies to the calculation of the Krylov tensors. In this case choosing a short restart length would help to mitigate this issue. Unfortunately, we have seen that when using GMRES to compute the marginal log-likelihood score a minimum restart length, which increases with the dimension, is required to ensure convergence. The results of GMRES can be improved by utilizing ALS to compute an approximate solution of required sums. However, it does not seem reasonable to use ALS to improve GMRES if ALS can also be used directly as a solver.

The ALS method has been specifically designed for the TT format, i.e., it exploits specific properties of the TT format. In particular, there is no noticeable dependency of the rank, which is used to set the maximum achievable accuracy, on the dimension of the problem. In addition it does not suffer from the error propagation issue. The suggested modifications of ALS to improve the convergence rate and reduce the risk to get stuck in a local minima show only little benefits in this case. The fact that the plain ALS method usually only requires three left-to-right-to-left sweeps simply leaves little room for improvement. Still, these variants can be advantageous in particularly difficult cases. For solving the marginal log-likelihood score, ALS, or a variant thereof, is clearly preferable to the other two methods.

In summary, the essential building blocks required to enable the MHN model to be applied to larger data sets than ever before by using the TT decomposition have been shown. The computational complexity – in particular that to calculate the marginal log-likelihood score – has been reduced from $\mathcal{O}(2^d)$ to $\mathcal{O}(d^3r^2 + cd^2r^3)$, i.e., the curse of dimensionalty has been overcome and thus the set goal of this thesis has been achieved.

The next step is to put these building blocks together in such a way that they can be applied to existing data sets. Depending on the data set, some preprocessing of the data, which requires specific expertise in bioinformatics, might be necessary. This expertise is brought in by project partners, who should be able to run the provided software without any specific expertise about its implementation. In addition, there is room for improvements and extensions, some of which are already being worked on within the working group. For example, Johannes Schuster is currently working, among other things, on an efficient implementation of the algorithms for fast approximate solution of linear algebra problems, depicted in Sec. 4.12, as part of his master's thesis. Extensions of the MHN model are also being considered. These may require mathematical operations which can not be computed directly in the TT format, e.g., possible implementations of the matrix exponential in the TT format have been developed in [101]. Apart from the TT format, the choice of the optimizer offers further potential for enhancements as outlined in Sec. 5.5. And last but not least, the library itself offers opportunities for performance optimizations. While it was designed with high efficiency in mind, there is definitely room for improvements here. Initial work in this direction was done in [102] with a performance analysis of the two probably most relevant reoccurring functions in terms of performance, QR and SVD.

Appendix A

Additional Algorithms

Algorithm A.1: TT-MALS with normalization

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$, $\delta \le \sqrt[-2]{d}\,\varepsilon$, and $\gamma \le \varepsilon$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \le \varepsilon\,\|b\|$
Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1, \lambda_x := 0$
for $k = d$ to $2$ by $-1$ do
    $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$,   $x^{|k-1\rangle} := x^{|k-1\rangle}\,\big(\|L\|^{-1}\,L\big)$,   $\lambda_x := \lambda_x + \log(\|L\|)$
    Form $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
    $\lambda_{A,k-1} := \|\Phi_{A,k-1}\|$,  $\Phi_{A,k-1} := \|\Phi_{A,k-1}\|^{-1}\,\Phi_{A,k-1}$
    $\lambda_{b,k-1} := \|\Phi_{b,k-1}\|$,  $\Phi_{b,k-1} := \|\Phi_{b,k-1}\|^{-1}\,\Phi_{b,k-1}$
$\lambda_x := \lambda_x + \log\big(\|x^{(1)}\|\big)$,   $x^{(1)} := \|x^{(1)}\|^{-1}\,x^{(1)}$
do
    for $k = 1$ to $d-1$ do
        Form $A_{k:k+1}$ and $b_{k:k+1}$ folowing (4.55) and (4.56)
        $\lambda := \exp\Big(\sum_{i=1,i\neq k}^{d-1}\big(\log(\lambda_{b,i}) - \log(\lambda_{A,i})\big) - \lambda_x\Big)$
        Solve $A_{k:k+1}\,\tilde{x}^{[k:k+1]} = \lambda\,b_{k:k+1}$ with accuracy $\gamma$ and initial guess $x^{[k:k+1]}$
        $\big[x^{|k\rangle},\, S,\, V\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k:k+1|}\big)$                // Orthogonalize $x^{!(k+1)}$
            $x^{\langle k+1|} := \big(\|S\|^{-1}\,S\big)\,V$,   $\lambda_x := \lambda_x + \log(\|S\|)$          // $\|x^{\langle k+1|}\| = 1$
        Update $\Psi_{A,k}$ and $\Psi_{b,k}$ following (4.47) and (4.50)
        $\lambda_{A,k} := \|\Psi_{A,k}\|$,  $\Psi_{A,k} := \lambda_{A,k}^{-1}\,\Psi_{A,k}$
        $\lambda_{b,k} := \|\Psi_{b,k}\|$,  $\Psi_{b,k} := \lambda_{b,k}^{-1}\,\Psi_{b,k}$
    for $k = d$ to $2$ by $-1$ do
        Form $A_{k-1:k}$ and $b_{k-1:k}$ folowing (4.55) and (4.56)
        $\lambda := \exp\Big(\sum_{i=1}^{d-1}\big(\log(\lambda_{b,i}) - \log(\lambda_{A,i})\big) - \lambda_x\Big)$
        Solve $A_{k-1:k}\,\tilde{x}^{[k-1:k]} = \lambda\,b_{k-1:k}$ with accuracy $\gamma$ and initial guess $x^{[k-1:k]}$
        $\big[U,\, S,\, x^{\langle k|}\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k-1:k|}\big)$                // Orthogonalize $x^{!(k-1)}$
            $x^{|k-1\rangle} := U\,\big(\|S\|^{-1}\,S\big)$,   $\lambda_x := \lambda_x + \log(\|S\|)$          // $\|x^{|k-1\rangle}\| = 1$
        Update $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
        $\lambda_{A,k-1} := \|\Phi_{A,k-1}\|$,  $\Phi_{A,k-1} := \lambda_{A,k-1}^{-1}\,\Phi_{A,k-1}$
        $\lambda_{b,k-1} := \|\Phi_{b,k-1}\|$,  $\Phi_{b,k-1} := \lambda_{b,k-1}^{-1}\,\Phi_{b,k-1}$
while $\|b - \exp(\lambda_x)\;A\,x\| > \varepsilon\,\|b\|$
$x := \exp(\lambda_x)\;x$
```

Algorithm A.2: TT-MALS with random enrichment

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$, $\delta \leq \sqrt[-2]{d}\,\varepsilon$, and $\gamma \leq \varepsilon$,
       enrichment rank $\hat{r}$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \leq \varepsilon\,\|b\|$
Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1$
for $k = d$ to 2 by $-1$ do
    $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$
        $x^{|k-1\rangle} := x^{|k-1\rangle}\,L$
    Form $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
do
    for $k = 1$ to $d-1$ do
        Form $A_{k:k+1}$ and $b_{k:k+1}$ folowing (4.55) and (4.56)
        Solve $A_{k:k+1}\,\tilde{x}^{[k:k+1]} = b_{k:k+1}$ with accuracy $\gamma$ and initial guess $x^{[k:k+1]}$
        $\big[x^{|k\rangle},\, S,\, V\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k:k+1|}\big)$                // Orthogonalize $x^{!(k+1)}$
                $x^{\langle k+1|} := S\,V$
        Enrich $x^{|k\rangle}$ by random $\eta^{|k\rangle}$ and update $x^{\langle k+1|}$ according to (4.61)
        $\big[x^{|k\rangle},\, R\big] := \mathrm{QR}\big(x^{|k\rangle}\big)$                // Re-orthogonalize $x^{!(k+1)}$
            $x^{\langle k+1|} := R\,x^{\langle k+1|}$
        Update $\Psi_{A,k}$ and $\Psi_{b,k}$ following (4.47) and (4.50)
    for $k = d$ to 2 by $-1$ do
        Form $A_{k-1:k}$ and $b_{k-1:k}$ folowing (4.55) and (4.56)
        Solve $A_{k-1:k}\,\tilde{x}^{[k-1:k]} = b_{k-1:k}$ with accuracy $\gamma$ and initial guess $x^{[k-1:k]}$
        $\big[U,\, S,\, x^{\langle k|}\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k-1:k|}\big)$                // Orthogonalize $x^{!(k-1)}$
                $x^{|k-1\rangle} := U\,S$
        Enrich $x^{\langle k|}$ by random $\eta^{\langle k|}$ and update $x^{|k-1\rangle}$ according to (4.62)
        $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$                // Re-orthogonalize $x^{!(k-1)}$
            $x^{|k-1\rangle} := x^{|k-1\rangle}\,L$
        Update $\Phi_{A,k-1}$ and $\Phi_{b,k-1}$ following (4.48) and (4.51)
while $\|b - A\,x\| > \varepsilon\,\|b\|$
```

Algorithm A.3: TT-MAMEn

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$, $\delta \le \sqrt[-2]{d}\,\varepsilon$, and $\gamma \le \varepsilon$,
       enrichment rank $\hat{r}$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \le \varepsilon\,\|b\|$
Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1$
Initialize: $\hat{\Psi}_{A,0} = \hat{\Phi}_{A,d} = 1$, $\hat{\Psi}_{b,0} = \hat{\Phi}_{b,d} = 1$, randomized tensor $z$
for $k = d$ to 2 by $-1$ do
    $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$,    $x^{|k-1\rangle} := x^{|k-1\rangle}\,L$
    $\big[\_,\, z^{\langle k|}\big] := \mathrm{LQ}\big(z^{\langle k|}\big)$
    Form $\Phi_{A,k-1}$, $\Phi_{b,k-1}$, $\hat{\Phi}_{A,k-1}$, $\hat{\Phi}_{b,k-1}$ following (4.48), (4.51), (4.67), (4.70)
do
    for $k = 1$ to $d-1$ do
        Form $A_{k:k+1}$, $b_{k:k+1}$, $\hat{A}_{k:k+1}$, $\hat{b}_{k:k+1}$, $\vec{A}_{k:k+1}$, $\vec{b}_{k:k+1}$ by (4.55), (4.56), etc.
        Solve $A_{k:k+1}\,\tilde{x}^{[k:k+1]} = b_{k:k+1}$ with accuracy $\gamma$ and initial guess $x^{[k:k+1]}$
        $\big[x^{|k\rangle},\, S,\, V\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k:k+1|}\big)$,    $x^{\langle k+1|} := S\,V$
            $\tilde{z}^{[k:k+1]} := \hat{b}_{k:k+1} - \hat{A}_{k:k+1}\,\tilde{x}^{[k:k+1]}$,    $\big[z^{|k\rangle},\, \_,\, \_\big] := \mathrm{SVD}_{\hat{r}}\big(\tilde{z}^{|k:k+1|}\big)$
            $\tilde{\eta}^{[k:k+1]} := \vec{b}_{k:k+1} - \vec{A}_{k:k+1}\,\tilde{x}^{[k:k+1]}$,    $\big[\eta^{|k\rangle},\, \_,\, \_\big] := \mathrm{SVD}_{\hat{r}}\big(\tilde{\eta}^{|k:k+1|}\big)$
        Enrich $x^{|k\rangle}$ by $\eta^{|k\rangle}$ and update $x^{\langle k+1|}$ according to (4.61)
            $\big[x^{|k\rangle},\, R\big] := \mathrm{QR}\big(x^{|k\rangle}\big)$,    $x^{\langle k+1|} := R\,x^{\langle k+1|}$
        Update $\Psi_{A,k}$, $\Psi_{b,k}$, $\hat{\Psi}_{A,k}$, $\hat{\Psi}_{b,k}$ following (4.47), (4.50), (4.66), (4.69)
    for $k = d$ to 2 by $-1$ do
        Form $A_{k-1:k}$, $b_{k-1:k}$, $\hat{A}_{k-1:k}$, $\hat{b}_{k-1:k}$, $\overleftarrow{A}_{k-1:k}$, $\overleftarrow{b}_{k-1:k}$ by (4.55), (4.56), etc.
        Solve $A_{k-1:k}\,\tilde{x}^{[k-1:k]} = b_{k-1:k}$ with accuracy $\gamma$ and initial guess $x^{[k-1:k]}$
        $\big[U,\, S,\, x^{\langle k|}\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k-1:k|}\big)$,    $x^{|k-1\rangle} := U\,S$
            $\tilde{z}^{[k-1:k]} := \hat{b}_{k-1:k} - \hat{A}_{k-1:k}\,\tilde{x}^{[k-1:k]}$,    $\big[\_,\, \_,\, z^{\langle k|}\big] := \mathrm{SVD}_{\hat{r}}\big(\tilde{z}^{|k-1:k|}\big)$
            $\tilde{\eta}^{[k-1:k]} := \overleftarrow{b}_{k-1:k} - \overleftarrow{A}_{k-1:k}\,\tilde{x}^{[k-1:k]}$,    $\big[\_,\, \_,\, \eta^{\langle k|}\big] := \mathrm{SVD}_{\hat{r}}\big(\tilde{\eta}^{|k-1:k|}\big)$
        Enrich $x^{\langle k|}$ by $\eta^{\langle k|}$ and update $x^{|k-1\rangle}$ according to (4.62)
            $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$,    $x^{|k-1\rangle} := x^{|k-1\rangle}\,L$
        Update $\Phi_{A,k-1}$, $\Phi_{b,k-1}$, $\hat{\Phi}_{A,k-1}$, $\hat{\Phi}_{b,k-1}$ following (4.48), (4.51), (4.67), (4.70)
while $\|b - A\,x\| > \varepsilon\,\|b\|$
```

Algorithm A.4: TT-(M)AMEn with normalization (Part 1 of 2)

```
Input: TT operator $A \in \mathbb{C}^{(\times_k^d n_k)\times(\times_k^d n_k)}$, TT tensor $b \in \mathbb{C}^{\times_k^d n_k}$ (right-hand side),
       TT tensor $x_0 \in \mathbb{C}^{\times_k^d n_k}$ (initial guess), accuracies $\varepsilon$, $\delta \leq \sqrt[-2]{d}\,\varepsilon$, and $\gamma \leq \varepsilon$,
       enrichment rank $\hat{r}$
Output: Solution TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with $\|b - A\,x\| \leq \varepsilon\,\|b\|$
1 Initialize: $x := x_0$, $\Psi_{A,0} = \Phi_{A,d} = 1$, $\Psi_{b,0} = \Phi_{b,d} = 1$, $\lambda_x := 0$
2 Initialize: $\hat{\Psi}_{A,0} = \hat{\Phi}_{A,d} = 1$, $\hat{\Psi}_{b,0} = \hat{\Phi}_{b,d} = 1$, randomized tensor $z$
3 for $k = d$ to 2 by $-1$ do
4     $[L,\, x^{\langle k|}] := \mathrm{LQ}(x^{\langle k|})$,   $x^{|k-1\rangle} := x^{|k-1\rangle}\left(\|L\|^{-1}\, L\right)$,   $\lambda_x := \lambda_x + \log(\|L\|)$
5     $[\_,\, z^{\langle k|}] := \mathrm{LQ}(z^{\langle k|})$
6     Form $\Phi_{A,k-1}$, $\Phi_{b,k-1}$, $\hat{\Phi}_{A,k-1}$, $\hat{\Phi}_{b,k-1}$ following (4.48), (4.51), (4.67), (4.70)
7     $\lambda_{A,k-1} := \|\Phi_{A,k-1}\|$,  $\Phi_{A,k-1} := \lambda_{A,k-1}^{-1}\,\Phi_{A,k-1}$,  $\hat{\Phi}_{A,k-1} := \lambda_{A,k-1}^{-1}\,\Phi_{A,k-1}$
8     $\lambda_{b,k-1} := \|\Phi_{b,k-1}\|$,  $\Phi_{b,k-1} := \lambda_{b,k-1}^{-1}\,\Phi_{b,k-1}$,  $\hat{\Phi}_{b,k-1} := \lambda_{b,k-1}^{-1}\,\Phi_{b,k-1}$
9 $\lambda_x := \lambda_x + \log(\|x^{(1)}\|)$,   $x^{(1)} := \|x^{(1)}\|^{-1}\, x^{(1)}$
```

Algorithm A.5: TT-AMEn with normalization (Part 2 of 2)

```
10 do
11     for $k = 1$ to $d-1$ do
12         Form $A_k$, $b_k$, $\hat{A}_k$, $\hat{b}_k$, $\vec{A}_k$, $\vec{b}_k$ by (4.46), (4.49), (4.65), (4.68), (4.74), (4.75)
13         $\lambda := \exp\Big(\sum_{i=1}^{d-1} \big(\log(\lambda_{b,i}) - \log(\lambda_{A,i})\big) - \lambda_x\Big)$
14         Solve $A_k\, \tilde{x}^{[k]} = \lambda\, b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
15         $\big[x^{|k\rangle},\, S,\, V\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k\rangle}\big)$, $x^{\langle k+1|} := \Big(\|S\|^{-1}\ S\Big)\ V\, x^{\langle k+1|}$, $\lambda_x := \lambda_x + \log(\|S\|)$
16         $z^{[k]} := \lambda\, \hat{b}_k - \hat{A}_k\, \tilde{x}^{[k]}$,   $\eta^{[k]} := \lambda\, \vec{b}_k - \vec{A}_k\, \tilde{x}^{[k]}$
17         Enrich $x^{|k\rangle}$ by $\eta^{|k\rangle}$ and update $x^{\langle k+1|}$ according to (4.61)
18         $\big[x^{|k\rangle},\, R\big] := \mathrm{QR}\big(x^{|k\rangle}\big)$,   $x^{\langle k+1|} := \Big(\|R\|^{-1}\ R\Big)\ x^{\langle k+1|}$,   $\lambda_x := \lambda_x + \log(\|R\|)$
19         $\big[z^{|k\rangle},\, R\big] := \mathrm{QR}\big(z^{|k\rangle}\big)$
20         Update $\Psi_{A,k}$, $\Psi_{b,k}$, $\hat{\Psi}_{A,k}$, $\hat{\Psi}_{b,k}$ following (4.47), (4.50), (4.66), (4.69)
21         $\lambda_{A,k} := \big\|\Psi_{A,k}\big\|$,  $\Psi_{A,k} := \lambda_{A,k}^{-1}\, \Psi_{A,k}$,   $\hat{\Psi}_{A,k} := \lambda_{A,k}^{-1}\, \Psi_{A,k}$
22         $\lambda_{b,k} := \big\|\Psi_{b,k}\big\|$,   $\Psi_{b,k} := \lambda_{b,k}^{-1}\, \Psi_{b,k}$,   $\hat{\Psi}_{b,k} := \lambda_{b,k}^{-1}\, \Psi_{b,k}$
23     for $k = d$ to $2$ by $-1$ do
24         Form $A_k$, $b_k$, $\hat{A}_k$, $\hat{b}_k$, $\overleftarrow{A}_k$, $\overleftarrow{b}_k$ by (4.46), (4.49), (4.65), (4.68), (4.77), (4.78)
25         $\lambda := \exp\Big(\sum_{i=1}^{d-1} \big(\log(\lambda_{b,i}) - \log(\lambda_{A,i})\big) - \lambda_x\Big)$
26         Solve $A_k\, \tilde{x}^{[k]} = \lambda\, b_k$ with accuracy $\gamma$ and initial guess $x^{[k]}$
27         $\big[U,\, S,\, x^{\langle k|}\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{\langle k|}\big)$, $x^{|k-1\rangle} := x^{|k-1\rangle}\, U\ \Big(\|S\|^{-1}\ S\Big)$, $\lambda_x := \lambda_x + \log(\|S\|)$
28         $z^{[k]} := \lambda\, \hat{b}_k - \hat{A}_k\, \tilde{x}^{[k]}$,   $\eta^{[k]} := \lambda\, \overleftarrow{b}_k - \overleftarrow{A}_k\, \tilde{x}^{[k]}$
29         Enrich $x^{\langle k|}$ by $\eta^{\langle k|}$ and update $x^{|k-1\rangle}$ according to (4.62)
30         $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$,   $x^{|k-1\rangle} := x^{|k-1\rangle}\ \Big(\|L\|^{-1}\ L\Big)$,   $\lambda_x := \lambda_x + \log(\|L\|)$
31         $\big[L,\, z^{\langle k|}\big] := \mathrm{LQ}\big(z^{\langle k|}\big)$
32         Update $\Phi_{A,k-1}$, $\Phi_{b,k-1}$, $\hat{\Phi}_{A,k-1}$, $\hat{\Phi}_{b,k-1}$ following (4.48), (4.51), (4.67), (4.70)
33         $\lambda_{A,k-1} := \big\|\Phi_{A,k-1}\big\|$,  $\Phi_{A,k-1} := \lambda_{A,k-1}^{-1}\, \Phi_{A,k-1}$,  $\hat{\Phi}_{A,k-1} := \lambda_{A,k-1}^{-1}\, \Phi_{A,k-1}$
34         $\lambda_{b,k-1} := \big\|\Phi_{b,k-1}\big\|$,   $\Phi_{b,k-1} := \lambda_{b,k-1}^{-1}\, \Phi_{b,k-1}$,   $\hat{\Phi}_{b,k-1} := \lambda_{b,k-1}^{-1}\, \Phi_{b,k-1}$
35 while $\|b - \exp(\lambda_x)\ A\, x\| > \varepsilon\, \|b\|$
36 $x := \exp(\lambda_x)\ x$
```

Algorithm A.6: TT-MAMEn with normalization (Part 2 of 2)

```
do
    for $k = 1$ to $d-1$ do
        Form $A_{k:k+1}$, $b_{k:k+1}$, $\hat{A}_{k:k+1}$, $\hat{b}_{k:k+1}$, $\vec{A}_{k:k+1}$, $\vec{b}_{k:k+1}$ by (4.55), (4.56), etc.
        $\lambda := \exp\Big(\sum_{i=1, i\neq k}^{d-1} \big(\log(\lambda_{b,i}) - \log(\lambda_{A,i})\big) - \lambda_x\Big)$
        Solve $A_{k:k+1}\,\tilde{x}^{[k:k+1]} = \lambda\, b_{k:k+1}$ with accuracy $\gamma$ and initial guess $x^{[k:k+1]}$
        $\big[x^{|k\rangle},\, S,\, V\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k:k+1|}\big)$,   $x^{\langle k+1|} := \Big(\|S\|^{-1}\; S\Big)\; V$,   $\lambda_x := \lambda_x + \log(\|S\|)$
            $\tilde{z}^{[k:k+1]} := \lambda\,\hat{b}_{k:k+1} - \hat{A}_{k:k+1}\,\tilde{x}^{[k:k+1]}$,   $\big[z^{|k\rangle},\, \_,\, \_\big] := \mathrm{SVD}_{\hat{r}}\big(\tilde{z}^{|k:k+1|}\big)$
            $\tilde{\eta}^{[k:k+1]} := \lambda\,\vec{b}_{k:k+1} - \vec{A}_{k:k+1}\,\tilde{x}^{[k:k+1]}$,   $\big[\eta^{|k\rangle},\, \_,\, \_\big] := \mathrm{SVD}_{\hat{r}}\big(\tilde{\eta}^{|k:k+1|}\big)$
        Enrich $x^{|k\rangle}$ by $\eta^{|k\rangle}$ and update $x^{\langle k+1|}$ according to (4.61)
        $\big[x^{|k\rangle},\, R\big] := \mathrm{QR}\big(x^{|k\rangle}\big)$,   $x^{\langle k+1|} := \Big(\|R\|^{-1}\; R\Big)\; x^{\langle k+1|}$,   $\lambda_x := \lambda_x + \log(\|R\|)$
        Update $\Psi_{A,k}$, $\Psi_{b,k}$, $\hat{\Psi}_{A,k}$, $\hat{\Psi}_{b,k}$ following (4.47), (4.50), (4.66), (4.69)
        $\lambda_{A,k} := \big\|\Psi_{A,k}\big\|$,   $\Psi_{A,k} := \lambda_{A,k}^{-1}\,\Psi_{A,k}$,   $\hat{\Psi}_{A,k} := \lambda_{A,k}^{-1}\,\Psi_{A,k}$
        $\lambda_{b,k} := \big\|\Psi_{b,k}\big\|$,   $\Psi_{b,k} := \lambda_{b,k}^{-1}\,\Psi_{b,k}$,   $\hat{\Psi}_{b,k} := \lambda_{b,k}^{-1}\,\Psi_{b,k}$
    for $k = d$ to $2$ by $-1$ do
        Form $A_{k-1:k}$, $b_{k-1:k}$, $\hat{A}_{k-1:k}$, $\hat{b}_{k-1:k}$, $\overleftarrow{A}_{k-1:k}$, $\overleftarrow{b}_{k-1:k}$ by (4.55), (4.56), etc.
        $\lambda := \exp\Big(\sum_{i=1, i\neq k}^{d-1} \big(\log(\lambda_{b,i}) - \log(\lambda_{A,i})\big) - \lambda_x\Big)$
        Solve $A_{k-1:k}\,\tilde{x}^{[k-1:k]} = \lambda\, b_{k-1:k}$ with accuracy $\gamma$ and initial guess $x^{[k-1:k]}$
        $\big[U,\, S,\, x^{\langle k|}\big] := \mathrm{SVD}_\delta\big(\tilde{x}^{|k-1:k|}\big)$,   $x^{|k-1\rangle} := U\,\Big(\|S\|^{-1}\; S\Big)$,   $\lambda_x := \lambda_x + \log(\|S\|)$
            $\tilde{z}^{[k-1:k]} := \lambda\,\hat{b}_{k-1:k} - \hat{A}_{k-1:k}\,\tilde{x}^{[k-1:k]}$,   $\big[\_,\, \_,\, z^{\langle k|}\big] := \mathrm{SVD}_{\hat{r}}\big(\tilde{z}^{|k-1:k|}\big)$
            $\tilde{\eta}^{[k-1:k]} := \lambda\,\overleftarrow{b}_{k-1:k} - \overleftarrow{A}_{k-1:k}\,\tilde{x}^{[k-1:k]}$,   $\big[\_,\, \_,\, \eta^{\langle k|}\big] := \mathrm{SVD}_{\hat{r}}\big(\tilde{\eta}^{|k-1:k|}\big)$
        Enrich $x^{\langle k|}$ by $\eta^{\langle k|}$ and update $x^{|k-1\rangle}$ according to (4.62)
        $\big[L,\, x^{\langle k|}\big] := \mathrm{LQ}\big(x^{\langle k|}\big)$,   $x^{|k-1\rangle} := x^{|k-1\rangle}\,\Big(\|L\|^{-1}\; L\Big)$,   $\lambda_x := \lambda_x + \log(\|L\|)$
        Update $\Phi_{A,k-1}$, $\Phi_{b,k-1}$, $\hat{\Phi}_{A,k-1}$, $\hat{\Phi}_{b,k-1}$ following (4.48), (4.51), (4.67), (4.70)
        $\lambda_{A,k-1} := \big\|\Phi_{A,k-1}\big\|$,   $\Phi_{A,k-1} := \lambda_{A,k-1}^{-1}\,\Phi_{A,k-1}$,   $\hat{\Phi}_{A,k-1} := \lambda_{A,k-1}^{-1}\,\Phi_{A,k-1}$
        $\lambda_{b,k-1} := \big\|\Phi_{b,k-1}\big\|$,   $\Phi_{b,k-1} := \lambda_{b,k-1}^{-1}\,\Phi_{b,k-1}$,   $\hat{\Phi}_{b,k-1} := \lambda_{b,k-1}^{-1}\,\Phi_{b,k-1}$
while $\|b - \exp(\lambda_x)\;A\,x\| > \varepsilon\,\|b\|$
$x := \exp(\lambda_x)\;x$
```

Appendix B

Derivations

B.1 Tensor Train: Motivation for optimized calculation of inner product

The algorithm shown in Alg. 4.1 to compute the inner product of two TT tensors with reduced computational complexity can be motivated by the following mathematical transformations:

$$
\begin{aligned}
\langle a, b \rangle &= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \left[(a^* \odot b)(i_1, \ldots, i_d) \right] \\
&= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \left[\left(\prod_{k=1}^{d} a^{(k)^*} \langle i_k \rangle \right) \left(\prod_{k=1}^{d} b^{(k)} \langle i_k \rangle \right) \right] \\
&= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \left[\left(\prod_{k=1}^{d} a^{(d-k+1)} \langle i_{d-k+1} \rangle^\dagger \right)^T \cdot \left(\prod_{k=1}^{d} b^{(k)} \langle i_k \rangle \right) \right] \\
&= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \left[\left(\prod_{k=1}^{d} a^{(d-k+1)} \langle i_{d-k+1} \rangle^\dagger \right) \cdot \left(\prod_{k=1}^{d} b^{(k)} \langle i_k \rangle \right) \right] \\
&= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \left[a^{(1:d)} \langle \overline{i_1, \ldots, i_d} \rangle^\dagger \cdot b^{(1:d)} \langle \overline{i_1, \ldots, i_d} \rangle \right] \\
&= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \left[a^{(2:d)} \langle \overline{i_2, \ldots, i_d} \rangle^\dagger \cdot a^{(1)} \langle i_1 \rangle^\dagger \cdot b^{(1)} \langle i_1 \rangle \cdot b^{(1:d)} \langle \overline{i_1, \ldots, i_d} \rangle \right] \\
&= \sum_{i_2=1}^{n_2} \cdots \sum_{i_d=1}^{n_d} \left[a^{(2:d)} \langle \overline{i_2, \ldots, i_d} \rangle^\dagger \cdot \underbrace{\left\{ \sum_{i_1=1}^{n_1} a^{(1)} \langle i_1 \rangle^\dagger \cdot b^{(1)} \langle i_1 \rangle \right\}}_{=: s_1} \cdot b^{(2:d)} \langle \overline{i_2, \ldots, i_d} \rangle \right]
\end{aligned}
$$

B.2 Tensor Train: Orthogonality of subtrain

Given a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with cores $x^{(k)}$ left-orthogonal $\forall k = 1, \dots, p$ its subtrains $x^{(1:q)}$ with $1 \leq q \leq p$ are called left-orthogonal and it can be shown that

$$x^{|1:q\rangle^\dagger} x^{|1:q\rangle} = \mathrm{Id}_{r_q} \qquad \forall q = 1, \dots, p. \tag{4.23}$$

Proof:

$$\begin{aligned}
x^{|1:q\rangle^\dagger} x^{|1:q\rangle} &= \sum_{i_1=1}^{n_1} \cdots \sum_{i_q=1}^{n_q} \left[x^{(1:q)^\dagger} \left\langle \overline{i_1, \dots, i_q} \right\rangle \cdot x^{(1:q)} \left\langle \overline{i_1, \dots, i_q} \right\rangle \right] \\
&= \sum_{i_1=1}^{n_1} \cdots \sum_{i_q=1}^{n_q} \left[x^{(2:q)^\dagger} \left\langle \overline{i_2, \dots, i_q} \right\rangle \cdot x^{(1)^\dagger} \left\langle i_1 \right\rangle \cdot x^{(1)} \left\langle i_1 \right\rangle \cdot x^{(2:q)} \left\langle \overline{i_2, \dots, i_q} \right\rangle \right] \\
&= \sum_{i_2=1}^{n_2} \cdots \sum_{i_q=1}^{n_q} \left[x^{(2:q)^\dagger} \left\langle \overline{i_2, \dots, i_q} \right\rangle \cdot \underbrace{\left\{ \sum_{i_1=1}^{n_1} x^{(1)^\dagger} \left\langle i_1 \right\rangle \cdot x^{(1)} \left\langle i_1 \right\rangle \right\}}_{\mathrm{Id}_{r_1} \text{ by Eq. 4.22}} \cdot x^{(2:q)} \left\langle \overline{i_2, \dots, i_q} \right\rangle \right] \\
&= \sum_{i_2=1}^{n_2} \cdots \sum_{i_q=1}^{n_q} \left[x^{(2:q)^\dagger} \left\langle \overline{i_2, \dots, i_q} \right\rangle \cdot x^{(2:q)} \left\langle \overline{i_2, \dots, i_q} \right\rangle \right] \\
&\vdots \\
&= \sum_{i_q=1}^{n_q} \left[x^{(q:q)^\dagger} \left\langle i_q \right\rangle \cdot x^{(q:q)} \left\langle i_q \right\rangle \right] = \mathrm{Id}_{r_q}
\end{aligned}$$

□

B.3 Tensor Train: Norm exploiting orthogonality

Given a TT tensor $x \in \mathbb{C}^{\times_k^d n_k}$ with orthogonal frame matrix $x^{!(p)}$, its norm is given by

$$\|x\| = \left\|x^{(p)}\right\|. \tag{4.28}$$

Proof:

$$\|x\|^2 = \langle x, x\rangle = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \left[{x^{(1:d)}}^\dagger \left\langle \overline{i_1, \dots, i_d} \right\rangle \cdot x^{(1:d)} \left\langle \overline{i_1, \dots, i_d} \right\rangle \right]$$

$$= \sum_{i_p=1}^{n_p} \cdots \sum_{i_d=1}^{n_d} \left[{x^{(p:d)}}^\dagger \left\langle \overline{i_p, \dots, i_d} \right\rangle \cdot \right.$$

$$\left. \cdot \underbrace{\left\{ \sum_{i_1=1}^{n_1} \cdots \sum_{i_{p-1}=1}^{n_{p-1}} {x^{(1:p-1)}}^\dagger \left\langle \overline{i_1, \dots, i_{p-1}} \right\rangle \cdot x^{(1:p-1)} \left\langle \overline{i_1, \dots, i_{p-1}} \right\rangle \right\}}_{\mathrm{Id}_{r_{p-1}} \text{by Eq. 4.23}} \cdot x^{(p:d)} \left\langle \overline{i_p, \dots, i_d} \right\rangle \right]$$

$$= \sum_{i_p=1}^{n_p} \cdots \sum_{i_d=1}^{n_d} \left[{x^{(p:d)}}^\dagger \left\langle \overline{i_p, \dots, i_d} \right\rangle \cdot x^{(p:d)} \left\langle \overline{i_p, \dots, i_d} \right\rangle \right]$$

$$= \sum_{i_p=1}^{n_p} \cdots \sum_{i_d=1}^{n_d} \operatorname{tr} \left[{x^{(p:d)}}^\dagger \left\langle \overline{i_p, \dots, i_d} \right\rangle \cdot x^{(p:d)} \left\langle \overline{i_p, \dots, i_d} \right\rangle \right]$$

$$= \sum_{i_p=1}^{n_p} \cdots \sum_{i_d=1}^{n_d} \operatorname{tr} \left[{x^{(p+1:d)}}^\dagger \left\langle \overline{i_{p+1}, \dots, i_d} \right\rangle \cdot {x^{(p)}}^\dagger \left\langle i_p \right\rangle \cdot x^{(p)} \left\langle i_p \right\rangle \cdot x^{(p+1:d)} \left\langle \overline{i_{p+1}, \dots, i_d} \right\rangle \right]$$

$$= \sum_{i_p=1}^{n_p} \cdots \sum_{i_d=1}^{n_d} \operatorname{tr} \left[{x^{(p)}}^\dagger \left\langle i_p \right\rangle \cdot x^{(p)} \left\langle i_p \right\rangle \cdot x^{(p+1:d)} \left\langle \overline{i_{p+1}, \dots, i_d} \right\rangle \cdot {x^{(p+1:d)}}^\dagger \left\langle \overline{i_{p+1}, \dots, i_d} \right\rangle \right]$$

$$= \sum_{i_p=1}^{n_p} \operatorname{tr} \left[{x^{(p)}}^\dagger \left\langle i_p \right\rangle \cdot x^{(p)} \left\langle i_p \right\rangle \cdot \right.$$

$$\left. \cdot \underbrace{\left\{ \sum_{i_{p+1}=1}^{n_{p+1}} \cdots \sum_{i_d=1}^{n_d} x^{(p+1:d)} \left\langle \overline{i_{p+1}, \dots, i_d} \right\rangle \cdot {x^{(p+1:d)}}^\dagger \left\langle \overline{i_{p+1}, \dots, i_d} \right\rangle \right\}}_{\mathrm{Id}_{r_p} \text{by Eq. 4.25}} \right]$$

$$= \sum_{i_p=1}^{n_p} \operatorname{tr} \left[{x^{(p)}}^\dagger \left\langle i_p \right\rangle \cdot x^{(p)} \left\langle i_p \right\rangle \right] = \left\|x^{(p)}\right\|^2$$

□

References

[1] D. Hanahan and R. A. Weinberg. "Hallmarks of Cancer: The Next Generation." *Cell* 144.5 (2011). DOI: 10.1016/j.cell.2011.02.013.

[2] R. Schill et al. "Modelling cancer progression using Mutual Hazard Networks." *Bioinformatics* 36.1 (2019). DOI: 10.1093/bioinformatics/btz513.

[3] N. Beerenwinkel et al. "Cancer Evolution: Mathematical Models and Computational Inference." *Systematic Biology* 64.1 (2014). DOI: 10.1093/sysbio/syu081.

[4] K. Hainke, J. Rahnenführer, and R. Fried. "Cumulative disease progression models for cross-sectional data: A review and comparison." *Biometrical Journal* 54.5 (2012). DOI: 10.1002/bimj.201100186.

[5] P. Georg et al. "Low-rank tensor methods for Markov chains with applications to tumor progression models" (2020). ARXIV: 2006.08135.

[6] P. Comon. "Tensors : A brief introduction." *IEEE Signal Processing Magazine* 31.3 (2014). DOI: 10.1109/MSP.2014.2298533.

[7] I. V. Oseledets. "A new tensor decomposition." *Doklady Mathematics* 80.1 (2009). DOI: 10.1134/S1064562409040115.

[8] I. V. Oseledets. "Tensor-Train Decomposition." *SIAM Journal on Scientific Computing* 33.5 (2011). DOI: 10.1137/090752286.

[9] L. R. Tucker. "Some mathematical notes on three-mode factor analysis." *Psychometrika* 31.3 (1966). DOI: 10.1007/BF02289464.

[10] W. Hackbusch and S. Kühn. "A New Scheme for the Tensor Representation." *Journal of Fourier Analysis and Applications* 15.5 (2009). DOI: 10.1007/s00041-009-9094-9.

[11] S. Kühn. "Hierarchische Tensordarstellung" (2012). URN: urn:nbn:de:bsz:15-qucosa-98906.

[12] L. Grasedyck. "Hierarchical Singular Value Decomposition of Tensors." *SIAM Journal on Matrix Analysis and Applications* 31.4 (2010). DOI: 10.1137/090764189.

[13] F. L. Hitchcock.
"The Expression of a Tensor or a Polyadic as a Sum of Products."
Journal of Mathematics and Physics 6.1–4 (1927).
DOI: 10.1002/sapm192761164.

[14] H. A. L. Kiers.
"Towards a standardized notation and terminology in multiway analysis."
Journal of Chemometrics 14.3 (2000).
DOI: 10.1002/1099-128X(200005/06)14:3<105::AID-CEM582>3.0.CO;2-I.

[15] J. D. Carroll and J.-J. Chang.
"Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition." *Psychometrika* 35.3 (1970).
DOI: 10.1007/bf02310791.

[16] R. Harshman. "Foundations of the parafac procedure: models and conditions for an 'exploratory' multimodal factor analysis."
UCLA Working Papers in Phonetics. 1970.

[17] T. G. Kolda and B. W. Bader. "Tensor Decompositions and Applications."
SIAM Review 51.3 (2009). DOI: 10.1137/07070111X.

[18] J. Håstad. "Tensor rank is NP-complete." *Journal of Algorithms* 11.4 (1990).
DOI: 10.1016/0196-6774(90)90014-6.

[19] V. de Silva and L.-H. Lim. "Tensor Rank and the Ill-Posedness of the Best Low-Rank Approximation Problem."
SIAM Journal on Matrix Analysis and Applications 30.3 (2008).
DOI: 10.1137/06066518X.

[20] G. Beylkin and M. J. Mohlenkamp.
"Algorithms for Numerical Analysis in High Dimensions."
SIAM Journal on Scientific Computing 26.6 (2005). DOI: 10.1137/040604959.

[21] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*
Johns Hopkins University Press, 1996. ISBN: 0801854148.

[22] L. N. Trefethen and D. Bau III. *Numerical Linear Algebra.*
Society for Industrial and Applied Mathematics, 1997. ISBN: 978-0898713619.

[23] L. Mirsky. "Symmetric Gauge Functions and Unitarily Invariant Norms."
The Quarterly Journal of Mathematics 11.1 (1960).
DOI: 10.1093/qmath/11.1.50.

[24] I. Oseledets and E. Tyrtyshnikov.
"TT-cross approximation for multidimensional arrays."
Linear Algebra and its Applications 432.1 (2010).
DOI: 10.1016/j.laa.2009.07.024.

[25] Y. Saad and M. H. Schultz. "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems." *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986). DOI: 10.1137/0907058.

[26] Å. Björck. "Solving linear least squares problems by Gram-Schmidt orthogonalization." *BIT Numerical Mathematics* 7.1 (1967). DOI: 10.1007/BF01934122.

[27] Y. Saad. "A Flexible Inner-Outer Preconditioned GMRES Algorithm." *SIAM Journal on Scientific Computing* 14.2 (1993). DOI: 10.1137/0914028.

[28] R. B. Morgan. "GMRES with Deflated Restarting." *SIAM Journal on Scientific Computing* 24.1 (2002). DOI: 10.1137/S1064827599364659.

[29] A. H. Baker, E. R. Jessup, and T. Manteuffel. "A Technique for Accelerating the Convergence of Restarted GMRES." *SIAM Journal on Matrix Analysis and Applications* 26.4 (2005). DOI: 10.1137/S0895479803422014.

[30] Q. Zou. "GMRES algorithms over 35 years" (2021). ARXIV: 2110.04017.

[31] A. Bouras and V. Frayssé. "Inexact Matrix-Vector Products in Krylov Methods for Solving Linear Systems: A Relaxation Strategy." *SIAM Journal on Matrix Analysis and Applications* 26.3 (2005). DOI: 10.1137/S0895479801384743.

[32] L. Giraud, S. Gratton, and J. Langou. "A note on relaxed and flexible GMRES." *CERFACS* TR/PA/04/41 (2004).

[33] V. Simoncini and D. B. Szyld. "Theory of Inexact Krylov Subspace Methods and Applications to Scientific Computing." *SIAM Journal on Scientific Computing* 25.2 (2003). DOI: 10.1137/S1064827502406415.

[34] J. v. Eshof and G. L. G. Sleijpen. "Inexact Krylov Subspace Methods for Linear Systems." *SIAM Journal on Matrix Analysis and Applications* 26.1 (2005). DOI: 10.1137/S0895479802403459.

[35] W. Hackbusch, B. N. Khoromskij, and E. E. Tyrtyshnikov. "Approximate iterations for structured matrices." *Numerische Mathematik* 109.3 (2008). DOI: 10.1007/s00211-008-0143-0.

[36] S. V. Dolgov. "TT-GMRES: solution to a linear system in the structured tensor format." *Russian Journal of Numerical Analysis and Mathematical Modelling* 28.2 (2013). DOI: 10.1515/rnam-2013-0009.

[37] J. Ballani and L. Grasedyck.
"A projection method to solve linear systems in tensor format."
Numerical Linear Algebra with Applications 20.1 (2013).
DOI: 10.1002/nla.1818.

[38] S. Holtz, T. Rohwedder, and R. Schneider. "The Alternating Linear Scheme for Tensor Optimization in the Tensor Train Format."
SIAM Journal on Scientific Computing 34.2 (2012). DOI: 10.1137/100818893.

[39] I. V. Oseledets and S. V. Dolgov.
"Solution of Linear Systems and Matrix Inversion in the TT-Format."
SIAM Journal on Scientific Computing 34.5 (2012). DOI: 10.1137/110833142.

[40] I. Oseledets. "DMRG Approach to Fast Linear Algebra in the TT-Format."
Computational Methods in Applied Mathematics 11.3 (2011).
DOI: doi:10.2478/cmam-2011-0021.

[41] S. V. Dolgov and D. V. Savostyanov. "Alternating Minimal Energy Methods for Linear Systems in Higher Dimensions."
SIAM Journal on Scientific Computing 36.5 (2014). DOI: 10.1137/140953289.

[42] S. V. Dolgov and D. V. Savostyanov. "Alternating minimal energy methods for linear systems in higher dimensions. Part I: SPD systems" (2013).
ARXIV: 1301.6068.

[43] S. V. Dolgov and D. V. Savostyanov.
"Alternating minimal energy methods for linear systems in higher dimensions. Part II: Faster algorithm and application to nonsymmetric systems" (2013).
ARXIV: 1304.1222.

[44] S. V. Dolgov. "Tensor product methods in numerical simulation of high-dimensional dynamical problems." PhD thesis. Leipzig University, 2014.
URN: urn:nbn:de:bsz:15-qucosa-151129.

[45] I. Affleck et al.
"Rigorous results on valence-bond ground states in antiferromagnets."
Phys. Rev. Lett. 59 (7 1987). DOI: 10.1103/PhysRevLett.59.799.

[46] M. Fannes, B. Nachtergaele, and R. F. Werner.
"Finitely correlated states on quantum spin chains."
Communications in Mathematical Physics 144.3 (1992).
DOI: 10.1007/BF02099178.

[47] A. Klümper, A. Schadschneider, and J. Zittartz. "Matrix Product Ground States for One-Dimensional Spin-1 Quantum Antiferromagnets."
Europhysics Letters (EPL) 24.4 (1993). DOI: 10.1209/0295-5075/24/4/010.

[48] D. Perez-Garcia et al. "Matrix Product State Representations" (2006).
DOI: 10.48550/ARXIV.QUANT-PH/0608197.

[49] Q. Zhao et al. "Tensor Ring Decomposition" (2016). ARXIV: 1606.05535.

[50] O. Mickelin and S. Karaman.
"On Algorithms for and Computing with the Tensor Ring Decomposition" (2018). ARXIV: 1807.02513.

[51] S. R. White. "Density matrix formulation for quantum renormalization groups." *Phys. Rev. Lett.* 69 (19 1992). DOI: 10.1103/PhysRevLett.69.2863.

[52] S. R. White. "Density-matrix algorithms for quantum renormalization groups." *Phys. Rev. B* 48 (14 1993). DOI: 10.1103/PhysRevB.48.10345.

[53] S. Dolgov et al. "Computation of extreme eigenvalues in higher dimensions using block tensor train format." *Computer Physics Communications* 185.4 (2014). DOI: 10.1016/j.cpc.2013.12.017.

[54] E. Jeckelmann. "Dynamical density-matrix renormalization-group method." *Phys. Rev. B* 66 (4 2002). DOI: 10.1103/PhysRevB.66.045114.

[55] S. R. White.
"Density matrix renormalization group algorithms with a single center site." *Phys. Rev. B* 72 (18 2005). DOI: 10.1103/PhysRevB.72.180403.

[56] S. V. Dolgov and D. V. Savostyanov. "Corrected One-Site Density Matrix Renormalization Group and Alternating Minimal Energy Algorithm." *Lecture Notes in Computational Science and Engineering.* Springer International Publishing, 2014. DOI: 10.1007/978-3-319-10705-9_33.

[57] U. Schollwöck. "The density-matrix renormalization group." *Rev. Mod. Phys.* 77 (1 2005). DOI: 10.1103/RevModPhys.77.259.

[58] U. Schollwöck.
"The density-matrix renormalization group in the age of matrix product states." *Annals of Physics* 326.1 (2011). January 2011 Special Issue. DOI: 10.1016/j.aop.2010.09.012.

[59] W. Grassmann. "Transient solutions in markovian queueing systems." *Computers & Operations Research* 4.1 (1977). DOI: 10.1016/0305-0548(77)90007-7.

[60] R. H. Byrd et al.
"A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM Journal on Scientific Computing* 16.5 (1995). DOI: 10.1137/0916069.

[61] J. J. Moré and D. J. Thuente.
"Line Search Algorithms with Guaranteed Sufficient Decrease." *ACM Trans. Math. Softw.* 20.3 (1994). DOI: 10.1145/192115.192132.

[62] R. Fletcher. *Practical Methods of Optimization.* Second. John Wiley & Sons, New York, NY, USA, 1987.

[63] J. Nocedal. "Updating quasi-Newton matrices with limited storage." *Mathematics of computation* 35.151 (1980). DOI: 10.1090/S0025-5718-1980-0572855-7.

[64] D. C. Liu and J. Nocedal.
"On the limited memory BFGS method for large scale optimization."
Mathematical Programming 45.1 (1989). DOI: 10.1007/BF01589116.

[65] S. Ruder. "An overview of gradient descent optimization algorithms" (2016).
ARXIV: 1609.04747.

[66] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd ed.
Springer, New York, NY, USA, 2006. DOI: 10.1007/978-0-387-40065-5.

[67] G. Andrew and J. Gao.
"Scalable Training of L1-Regularized Log-Linear Models."
Proceedings of the 24th International Conference on Machine Learning.
Association for Computing Machinery, 2007. DOI: 10.1145/1273496.1273501.

[68] P. Gong and J. Ye. "A Modified Orthant-Wise Limited Memory Quasi-Newton Method with Convergence Analysis." *Proceedings of the 32nd International Conference on International Conference on Machine Learning* 37 (2015).
PMLR: v37/gonga15.

[69] N. N. Schraudolph, J. Yu, and S. Günter.
"A Stochastic Quasi-Newton Method for Online Convex Optimization."
Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics 2 (2007). PMLR: v2/schraudolph07a.

[70] J. Wangni. "Training L1-Regularized Models with Orthant-Wise Passive Descent Algorithms" (2017). ARXIV: 1704.07987.

[71] I. V. Oseledets et al. *TT-Toolbox*. GITHUB: oseledets/TT-Toolbox.

[72] *Matlab*. The MathWorks Inc., Natick, Massachusetts, USA.

[73] I. V. Oseledets et al. *ttpy: Python implementation of the TT-Toolbox*.
GITHUB: oseledets/ttpy.

[74] I. V. Oseledets et al. *tt-fort: Fortran computing core of the TT-Toolbox*.
GITHUB: oseledets/tt-fort.

[75] R. Ballester-Ripoll et al. *tntorch: Tensor Network Learning with PyTorch*.
GITHUB: rballester/tntorch.

[76] J. Kossaifi et al. "TensorLy: Tensor Learning in Python."
Journal of Machine Learning Research 20.26 (2019). JMLR: v20/18-277.

[77] A. Novikov et al. "Tensor Train Decomposition on TensorFlow (T3F)."
Journal of Machine Learning Research 21.30 (2020). JMLR: v21/18-008.

[78] D. Suess and M. Holzäpfel.
"mpnum: A matrix product representation library for Python."
Journal of Open Source Software 2.20 (2017). DOI: 10.21105/joss.00465.

[79] D. Jaschke, M. L. Wall, and L. D. Carr.
"Open source Matrix Product States: Opening ways to simulate entangled many-body quantum systems in one dimension."
Computer Physics Communications 225 (2018).
DOI: `10.1016/j.cpc.2017.12.015`.

[80] A. Milsted et al. *evoMPS: An implementation of the time dependent variational principle for matrix product states.* GITHUB: `amilsted/evoMPS`.

[81] M. Fishman, S. R. White, and E. M. Stoudenmire.
"The ITensor Software Library for Tensor Network Calculations" (2020).
ARXIV: `2007.14822`.

[82] S. A. Matveev et al.
"Oscillating stationary distributions of nanoclusters in an open system."
Mathematical and Computer Modelling of Dynamical Systems 26.6 (2020).
DOI: `10.1080/13873954.2020.1793786`.

[83] P. Gelß et al. *Scikit-TT: Tensor Train Toolbox.* GITHUB: `PGelss/scikit_tt`.

[84] B. Huber and S. Wolf. *Xerus - A General Purpose Tensor Library.*
URL: `libxerus.org`.

[85] C. Roberts et al.
"TensorNetwork: A Library for Physics and Machine Learning" (2019).
ARXIV: `1905.01330`.

[86] *Boost.* URL: `boost.org`.

[87] G. Guennebaud, B. Jacob, et al. *Eigen.* URL: `eigen.tuxfamily.org`.

[88] L. Dionne. *Familiar template syntax for generic lambdas.* 2017.
JTC1/SC22/WG21: `P0428`.

[89] D. Gregor et al. *Proposed Wording for Variadic Templates.* 2007.
JTC1/SC22/WG21: `N2242`.

[90] D. Gregor and E. Niebler. *Extending Variadic Template Template Parameters.* 2008. JTC1/SC22/WG21: `N2555`.

[91] R. Smith, A. Sutton, and D. Vandevoorde. *Immediate functions.* 2018.
JTC1/SC22/WG21: `P1073`.

[92] J. Maurer. *constexpr if: A slightly different syntax.* 2016.
JTC1/SC22/WG21: `P0292`.

[93] A. Sutton and R. Smith. *Folding expressions.* 2014. JTC1/SC22/WG21: `N4295`.

[94] T. L. Jehan. *Unary Folds and Empty Parameter Packs (revision 1).* 2015.
JTC1/SC22/WG21: `P0036`.

[95] B. Revzin. *Generalized pack declaration and usage.* [Working draft]. 2020.
JTC1/SC22/WG21: `P1858`.

[96] H. Finkel. *C++ Should Support Just-in-Time Compilation.* [Working draft]. 2020. JTC1/SC22/WG21: `P1609`.

[97] T. Veldhuizen. "Expression Templates." *C++ Report* 7 (1995).

[98] R. G. Edwards and B. Joó. "The Chroma Software System for Lattice QCD." *Nuclear Physics B - Proceedings Supplements* 140 (2005). LATTICE 2004. DOI: `10.1016/j.nuclphysbps.2004.11.254`.

[99] P. A. Boyle et al. "Grid: A next generation data parallel C++ QCD library." *PoS* LATTICE 2015 (2016). DOI: `10.22323/1.251.0023`.

[100] A. Sutton. *Wording Paper, C++ extensions for Concepts.* 2017. JTC1/SC22/WG21: `P0734`.

[101] M. Huber. "Implementation of the matrix exponential function in the Tensor-Train Format." Bachelor's Thesis. University of Regensburg, 2022.

[102] T. Sterr. "Performance Analysis of a C++ Tensor Library with Applications to Cancer Progression Models." Bachelor's Thesis. University of Regensburg, 2022.