TENSOR COMPUTATIONS WITH DIMENSIONALITY MANIPULATIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Tianyi Shi

May 2022

TENSOR COMPUTATIONS WITH DIMENSIONALITY MANIPULATIONS

Tianyi Shi, Ph.D.

Cornell University 2022

Methodologies that ensure the compressibility of tensors are introduced. Bounds on the storage costs with respect to various tensor formats are derived. A new algorithm combining data-sparse tensor formats and factored alternating direction implicit method is designed to solve Sylvester tensor equations, and incorporated in a fast spectral Poisson equation solver on cubes with optimal complexity.

New parallelizable algorithms for computing the tensor-train decomposition of tensors in original format, streaming data, Tucker format, and that satisfy algebraic relations, are proposed. Based on the input format, the algorithms involve deterministic or probabilistic aspects, and all have guarantees of accuracy. Scaling analysis and numerical experiments are provided to demonstrate computational and storage efficiency.

An ultraspherical spectral method is developed for fractional partial differential equations via the Caffarelli–Silvestre extension on disk and rectangular domains. A parallel domain decomposition solver is designed for multi-core performance of non-smooth functions. The discretized equation is solved via direct tensor equation solvers, and numerical performance is shown with a fractional PDE constrained optimization problem.

Linear systems in electron correlation calculation from computational chemistry are converted into tensor equations to reduce computing and storage costs. Several algorithms are developed to exploit the sparsity and data-sparsity of

chemical structures. Numerical results indicate that tensor equation solvers are competitive over traditional linear system solvers with both canonical and localized orbital bases formulations.

The quantized tensor-train format of tensors is introduced to approximate analytic functions via Chebyshev polynomial expansions. Analysis of different types of singularities is carried out, leading to theoretical guarantees of coefficient storage compressibility.

BIOGRAPHICAL SKETCH

Tianyi Shi was introduced to computational mathematics and optimization by Wotao Yin at the University of California, Los Angeles. He began his PhD in Applied Mathematics at Cornell University in August of 2017 under the supervision of Alex Townsend and became deeply attracted by numerical multilinear algebra and scientific computing. He will be a postdoctoral fellow at Lawrence Berkeley National Laboratory advised by Sherry Li, starting in July 2022.

Dedicated to Xinru.

ACKNOWLEDGEMENTS

My graduate life at Cornell started with an unsuccessful trip to Ithaca for campus visit. My flight only allowed half of the travelers on board because of a malfunction, so I flew to Syracuse, and got dropped off at midnight by two friendly locals on their way home to Binghamton. Fortunately, the campus visit was worth all these detours, as I became deeply attracted by Cornell, by CAM, and by the research ideas of Alex Townsend, whom I decided to become a student of after his talk to prospective students and a personal chat with him. Alex is a great applied mathematician and advisor. He has broadened my horizon in many fields of computational mathematics, and he has shaped my perspective in understanding problems as a numerical analyst. He sets high standards for his students, but he provides help and advice in mathematical reasoning, coding, and scientific writing along the way. He is enthusiastic and has a lot of interesting ideas, so personal meetings, group meetings, and reading groups are always productive and I can learn many established and innovative perspectives. He believes applied mathematics is a highly collaborative subject, and encourages his students to work with researchers in other fields and in other institutions.

I am fortunate enough to work with and receive advice from other distinguished researchers. I am grateful to Harbir Antil, who supported me as an intern at George Mason University in the summer of 2019, and encouraged my interests in fractional PDEs and PDE-constrained optimization. I am also grateful to Sherry Li, who supported me as a research intern at Lawrence Berkeley National Laboratory in the summer of 2021, introduced me to an interesting interdisciplinary research project, funded my last year as a graduate student, and helped me secure a postdoctoral fellow position at LBL. I have benefited

from the guidance of many others, including David Bindel, Zichao (Wendy) Di, Martin Head-Gordon, Drew Kouri, Yang Liu, and Madeleine Udell.

I thank my colleagues and friends, Abdulrahman Aldossary, Nicolas Boullé, Dan Fortunato, Erika Fowler-Decatur, Marc Aurèle Gilles, Diana Halikias, Andrew Horning, Shiqi Ma, Dongping Qi, Lily Reeves, Thomas Reeves, Max Ruth, Runxi Shen, Zhenling Wang, Heather Wilber, Annan Yu, Mengxia Zhang, Xinran Zhu, and Jennifer Zvonek, who have not only provided constant scientific inspirations, but also made my journey outside of academics incredibly fun with board games, barbeques, hotpots, and bouldering. I thank my parents for continuously support me through my ups and downs in a foreign country. Adopting my cat, Witch, has been one of my best choices over the last few years. She has grown with me, moved with me, and accompanied me during my happiness and sadness. Finally, I want to express my love and gratitude to my partner Xinru. A long-distance relationship between Cornell and Stanford is filled with text messages, video calls, and monthly flights, but each reunion is fun and rewarding. During the pandemic, she moved to Ithaca and worked remotely. We had so many memorable moments together cooking, exercising, visiting attractions, and exploring restaurants. We support each other on life-changing events and decisions, and I look forward to our future journey together.

CONTENTS

CHAPTER 1

INTRODUCTION

Methods that use dimensionality manipulations aim to convert a d-dimensional problem into another one with dimension $d' \neq d$ for efficiency and well-posedness. In particular, a method is called to utilize dimensionality reduction if $d' < d$, and dimensionality increase if $d' > d$. This thesis explores using dimensionality manipulations to obtain data-sparse tensor formats, solve high dimensional tensor equations, and apply them in developing methods to compute and explain applications in physical and chemical simulations. In particular, dimensionality reduction is used to exploit low rank structures to alleviate the curse of dimensionality, and dimensionality increase gives rise to better-defined problems and more efficient algorithms.

In Chapter 2, we present three methodologies that ensure the compressibility of tensors. These methodologies include the algebraic structures and smoothness properties of the generating functions, and displacement structure that the tensors satisfy. For each methodology, we derive bounds on storage costs in various low rank tensor formats that partially explain the abundance of compressible tensors in applied mathematics. Theories related to tensor displacement structure also allow us to develop an efficient Sylvester tensor equation solver, which is essential in designing spectral partial differential equation (PDE) solvers that have optimal complexity.

In Chapter 3, to incorporate high-performance computing into dimensionality reduction, we propose four parallelizable algorithms that compute the tensor-train (TT) format from various tensor inputs: (1) Parallel-TTSVD for traditional format, (2) PSTT and its variants for streaming data, (3) Tucker2TT

for Tucker format, and (4) TT-fADI for solutions of Sylvester tensor equations. These algorithms have theoretical guarantees of accuracy, and scaling analysis and numerical results indicate computational and storage efficiency.

In Chapter 4, we present a spectral method based on an ultraspherical polynomial discretization to solve fractional PDEs on rectangular and disk domains via the Caffarelli–Silvestre extension. We rewrite the original problems into different equations on hexahedrons and cylinders to deal with nonlocality, and solve the resulting discretized system using tensor equation solvers. We also design a parallelizable domain decomposition algorithm for highly non-smooth functions. We demonstrate the numerical performance of this spectral method by applying it to an optimization problem with fractional PDE constraints.

In Chapter 5, we compute the electron correlation energy in quantum chemistry via several innovative ways involving tensor equation solvers. Depending on chemical formulations, we convert the original matrix linear system to different tensor equations, so that the sparsity and data-sparsity can be used for efficiency. We derive the running time complexity for each algorithm, and numerical results show that these methods are faster than traditional linear system solvers for realistic test cases from chemical experiments.

In Chapter 6, we introduce the quantized tensor-train (QTT) tensor format to approximate analytic functions with Chebyshev polynomial expansions. For different types of singularity points, we provide theoretical guarantees of compressibility based on Chebyshev coefficient evaluations. The bounds lead to accurate coefficient tensor representations, which are cheaper in storage costs over the traditional routine that stores coefficients in a vector.

1.1 Why are tensors important?

A wide variety of applications lead to problems involving data or solutions that can be represented by tensors, which are higher-dimensional generalizations of matrices. One of the simplest ways of obtaining a tensor is to reorganize elements in a matrix without changing the column-major ordering. We show in this thesis that certain tensors formulated in this way can capture additional underlying structures. In addition, we emphasize tensors generated by discretizations of continuous problems. For example, we can sample a 3D function $f(x, y, z)$ on grid points $(x_1, \ldots, x_N)$, $(y_1, \ldots, y_N)$, and $(z_1, \ldots, z_N)$, to obtain a tensor $\mathcal{F} \in \mathbb{C}^{N \times N \times N}$ defined element-wise by

$$\mathcal{F}_{i,j,k} = f(x_i, y_j, z_k), \quad 1 \leq i, j, k \leq N.$$

Consider a PDE $Lu = f$ on a bounded domain $\Omega \subset \mathbb{R}^d$ for $d \geq 3$ with Lipschitz smooth boundary as another example. One can then choose a desired ansatz, such as finite difference, finite element, or spectral scheme to discretize the PDE to a tensor equation

$$h(\mathcal{U}) = \mathcal{F},$$

where the tensor $\mathcal{U}$ is the solution to the discretized PDE, and h is used to represent operations described in the continuous operator L.

Researchers in a broader community use tensors in their respective fields and applications. Tensors appear in molecular dynamics and related physical sciences fields to store locations and velocities in a 3D space. Tensors are also used for more complicated data representations. For example, videos can be stored as 3D tensors where time is treated as the third dimension. In addition, tensors are useful in social sciences when they are introduced to graph

theory and network science as hypergraphs [76], which can store edges between two groups of nodes, and thus can be used to analyze interactions between two groups of people. Apart from these applications, tensors emerge in signal processing [52, 54, 72, 154], computer vision [100, 187, 215, 216, 219, 220], neuroscience [1, 25, 56, 150, 152], data mining and analysis [2, 44, 138, 139, 203], approximation theory [121], continuum mechanics [71], and differential equations [120, 127]. For a more inclusive list of application fields, we refer the readers to survey papers such as [123].

Despite the close connections between matrices and tensors, computations with tensors can be more challenging than those with matrices. A typical example is multiple ways of tensor decomposition to be introduced in Section 1.2. For readers' convenience, we present some tensor notations we shall follow throughout this thesis. We also define several tensor operations that are not common in linear algebra, but are used in multilinear algebra.

In this thesis, we use upper class letters to represent matrices and calligraphic capital letters to represent tensors. We commonly use MATLAB-style notation ":" for indices, i.e., $a:b$ represents the index set $\{a, a+1, \ldots, b\}$, and a single ":" stands for all the indices in that dimension. For example, $A(:, 3:4)$ or $A_{:,3:4}$ represents the submatrix of A that contains its third and fourth columns, and $\mathcal{Y}(:, j, :)$ represents the matrix slice of the tensor $\mathcal{Y}$ by fixing the second index to be j. We also use $\mathcal{Y}(:)$ to stack all the entries of $\mathcal{Y}$ into a single vector using column-major ordering. We use the MATLAB command "reshape" to reorganize elements of a tensor. If $\mathcal{Y} \in \mathbb{C}^{n_1 \times n_2 \times n_3}$, then $\mathrm{reshape}(\mathcal{Y}, n_1 n_2, n_3)$ returns a matrix of size $n_1 n_2 \times n_3$ formed by stacking entries according to their multi-index. Therefore, $\mathcal{Y}(:)$ and $\mathrm{reshape}(\mathcal{Y}, n_1 n_2 n_3, 1)$ are equivalent. Similarly,

if $Z \in \mathbb{C}^{n_1 n_2 \times n_3}$, then $\text{reshape}(Z, n_1, n_2, n_3)$ returns a tensor of size $n_1 \times n_2 \times n_3$. Now, consider a tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, then we have the following definitions.

Frobenius norm. The Frobenius norm of a tensor is defined similarly as the Frobenius norm of a matrix, as the square root of the sum of all its elements, i.e.,

$$\|\mathcal{X}\|_F^2 = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} |\mathcal{X}_{i_1,\ldots,i_d}|^2. \tag{1.1}$$

Double bracket. In the tensor literature, the double bracket denotes a mapping from the parametric space to the space of tensors. Specifically, it can be considered as a weighted sums of rank-1 tensors, i.e.,

$$[\![\mathcal{G}; A^{(1)}, \ldots, A^{(d)}]\!] = \sum_{i_1=1}^{r_1} \cdots \sum_{i_d=1}^{r_d} \mathcal{G}_{i_1,\ldots,i_d} A^{(1)}_{i_1} \circ \cdots \circ A^{(d)}_{i_d}, \qquad A^{(k)} \in \mathbb{C}^{n_k \times r_k}, \tag{1.2}$$

where $\mathcal{G} \in \mathbb{C}^{r_1 \times \cdots \times r_d}$ is often referred to as the core tensor and $v_1 \circ \cdots \circ v_d$ is the d-way outer-product of vectors [123].

Fibers and slices We can obtain a vector from $\mathcal{X}$ if we fix all but one indices. This vector is a mode-j fiber of $\mathcal{X}$ if the unfixed index is at the jth dimension. For example, $\mathcal{X}(:, i_2, \ldots, i_d)$ denotes a mode-1 fiber. In particular, mode-1 and mode-2 fibers can be considered as higher-dimensional analogues of matrix columns and rows respectively. Similarly, we can obtain a matrix from $\mathcal{X}$ if we fix all but two indices. Specifically for 3D tensors, the sliced matrices can be referred to as horizontal slices, lateral slices, or frontal slices if the fixed index is at the first, second, or third dimension respectively.

Flattening by reshaping. One can reorganize the entries of a tensor into a matrix without changing the column-major ordering, and this idea is fundamental to the TT decomposition. The kth unfolding of $\mathcal{X}$ is represented as

$$X_k = \text{reshape}\left(\mathcal{X}, \prod_{s=1}^{k} n_s, \prod_{s=k+1}^{d} n_s\right).$$

Flattening via matricization. Another way to flatten a tensor is to arrange the mode-j fibers to be the columns of a matrix [124], and this operation is central for the orthogonal Tucker decomposition. We denote the kth matricization of $\mathcal{X}$ by $X_{(k)} \in \mathbb{C}^{n_k \times \prod_{j \neq k} n_j}$. Since mode-1 fibers can be considered as columns of a tensor, we have $X_{(1)} = X_1$. In this thesis, for a tensor $\mathcal{X}$, matricizations are constructed so that there exists another tensor $\mathcal{Y}^j$ satisfying [53]

$$Y^j_{(1)} = X_{(j)},\ \ldots,\ Y^j_{(d-j+1)} = X_{(d)},\ Y^j_{(d-j+2)} = X_{(1)},\ \ldots,\ Y^j_{(d)} = X_{(j-1)}. \tag{1.3}$$

The k-mode product. The k-mode product of $\mathcal{X}$ with a matrix $A \in \mathbb{C}^{m \times n_k}$ is denoted by $\mathcal{X} \times_k A$, and defined elementwise as

$$(\mathcal{X} \times_k A)_{i_1,\ldots,i_{k-1},j,i_{k+1},\ldots,i_d} = \sum_{i_k=1}^{n_k} \mathcal{X}_{i_1,\ldots,i_d} A_{j,i_k}, \quad 1 \leq j \leq m. \tag{1.4}$$

This is equivalent to computing $AX_{(k)}$ and reorganizing back to a tensor.

1.2 Data sparse tensor formats

For a tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, the storage cost for all the entries takes $\prod_{j=1}^d n_j$ space, which grows exponentially with respect to dimension and size. Even in the case that $n_1 = \cdots = n_d = 2$, storing all the entries can be formidable for moderate size d. This phenomenon is referred to as "the curse of dimensionality", and it poses a significant challenge for researchers to handle high-dimensional tensors. Therefore, it is often essential to represent or approximate tensors using sparse data formats, such as low rank tensor decompositions [85, 123].

The situation for low rank tensor formats is very different from that for matrices. For a matrix $X \in \mathbb{C}^{m \times n}$, its economized singular value decomposition (SVD) is given by $X = U\Sigma V^*$, where $U \in \mathbb{C}^{m \times p}$ and $V \in \mathbb{C}^{n \times p}$ have orthonormal

columns, $*$ denotes the complex conjugate of a matrix, $\Sigma \in \mathbb{R}^{p\times p}$ is a diagonal matrix containing all the singular values $\sigma_1(X) \geq \sigma_2(X) \geq \cdots \geq \sigma_p(X) \geq 0$, and $p = \min(m, n)$. Since the SVD offers a unique way to represent each matrix X as a linear combination of vector outer products, and the vectors come from unitary matrices, one of the most essential features of the SVD is that it allows one to derive the best rank $\leq k$ approximation X_k of X with truncation:

Theorem 1.2.1 (Eckart–Young–Mirsky Theorem). *Let $X \in \mathbb{C}^{m\times n}$ with SVD $X = U\Sigma V^*$, and $Y_k \in \mathbb{C}^{m\times n}$ with $\mathrm{rank}(Y_k) = k$ for $1 \leq k \leq \min(m, n)$, then*

$$\text{(i)} \quad \sigma_{\mathrm{k+1}}(\mathrm{X}) = ||\mathrm{X} - \mathrm{X_k}||_2 \leq ||\mathrm{X} - \mathrm{Y_k}||_2,$$

$$\text{(ii)} \quad \sqrt{\sum_{\mathrm{j=k+1}}^{\mathrm{min(m,n)}} \sigma_{\mathrm{j}}^2(\mathrm{X})} = ||\mathrm{X} - \mathrm{X_k}||_{\mathrm{F}} \leq ||\mathrm{X} - \mathrm{Y_k}||_{\mathrm{F}},$$

where $X_k = U(:, 1 : k)\Sigma(1 : k, 1 : k)V(:, 1 : k)^*$.

Proof. See [81, Sec. 2.5.3]. □

In practice, one sets up a tolerance $0 < \epsilon < 1$ and intends to find an approximation $\tilde{X}$ of X such that $||X - \tilde{X}||_2 \leq \epsilon||X||_2$ or $||X - \tilde{X}||_F \leq \epsilon||X||_F$ with $\mathrm{rank}(\tilde{X})$ as small as possible. Similarly, throughout this thesis, for a tensor $\mathcal{X}$, we look for an approximation $\tilde{\mathcal{X}}$ that has low tensor ranks and satisfies

$$||\mathcal{X} - \tilde{\mathcal{X}}||_F \leq \epsilon||\mathcal{X}||_F. \tag{1.5}$$

If $\mathcal{X}$ can be well-approximated by $\tilde{\mathcal{X}}$, then dramatic storage and computational benefits can be achieved by replacing $\mathcal{X}$ by $\tilde{\mathcal{X}}$ [85, 92].

Understanding low rank tensor formats is much more complicated since there does not exist a tensor SVD. In other words, for any d-dimensional tensor $\mathcal{X}$, if $\mathcal{X}$ is represented as a linear combination of rank-1 tensors, i.e., outer

products of d vectors, then the generating vectors of each dimension cannot be orthogonal to each other. Analogously, if the factor matrices of a factorization of $\mathcal{X}$ have orthonormal columns, then the connecting tensor that mimics the singular value diagonal matrix cannot be diagonal and can only be dense. Therefore, in the tensor community, there are multiple data sparse tensor formats actively used by researchers. It is often an *ad hoc* process to choose particular low rank formats for different applications. In the rest of this section, we introduce four data-sparse tensor formats of analytical and practical importance.

1.2.1 Tensor-train format

The TT format, or also known as the matrix product state (MPS) [169], represents each tensor entry as the product of a sequence of matrices, and is used in molecular simulations [184], high-order correlation functions [125], and partial differential equation (PDE) constrained optimization [26, 60]. A tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ has TT cores $\mathcal{G}_k \in \mathbb{C}^{s_{k-1} \times n_k \times s_k}$ for $1 \leq k \leq d$, if the cores satisfy

$$\mathcal{X}_{i_1,\ldots,i_d} = \mathcal{G}_1(:, i_1, :)\mathcal{G}_2(:, i_2, :) \cdots \mathcal{G}_d(:, i_d, :), \qquad 1 \leq i_k \leq n_k.$$

Since the product of the matrices must be a scalar, we have $s_0 = s_d = 1$. We call $\boldsymbol{s} = (s_0, \ldots, s_d)$ the size of the TT cores, and it is an entry-by-entry bound on the TT rank $\boldsymbol{r} = (r_0, \ldots, r_d)$. In this way, a TT representation with TT core size $\boldsymbol{s}$ requires $p^{\mathrm{TT}} \leq \sum_{k=1}^{d} s_{k-1} s_k n_k$ degrees of freedom for storage, which is linear in mode size $\boldsymbol{n} = (n_1, \ldots, n_d)$ and order d. Figure. 1.1 illustrates a TT format with TT core size $\boldsymbol{s}$.

The TTSVD algorithm (see Algorithm 1) computes a TT format by sequentially constructing the TT cores via reshaping and SVD [164]. In this way, we

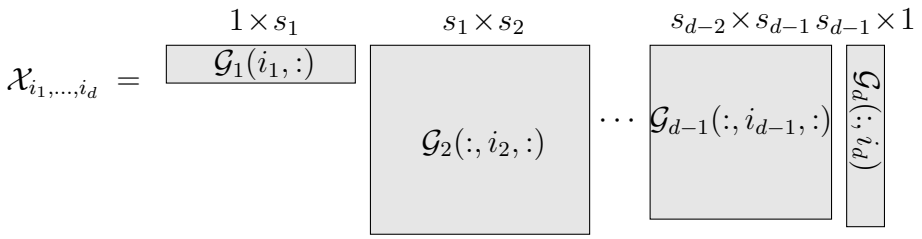

Figure 1.1: The TT format with TT core size $\boldsymbol{s} = (s_0, \ldots, s_d)$ where $s_0 = s_d = 1$. Each entry of a tensor is represented by the product of d matrices, where the kth matrix in the "train" is selected based on the value of i_k.

can use ranks of the tensor unfoldings to bound entries of the TT rank [164], and we typically set the TT core size to be smaller than the ranks of the unfolding matrices. That is,

$$r_k \leq s_k \leq \operatorname{rank}(X_k), \quad 1 \leq k \leq d-1. \tag{1.6}$$

Therefore, if the ranks of all the matrices X_k for $1 \leq k \leq d-1$ are small, then the TT format of $\mathcal{X}$ is data-sparse. In particular, if the SVDs in TTSVD are truncated to have an accuracy of $\epsilon/\sqrt{d-1}$ in the Frobenius norm, then we obtain an approximation $\tilde{\mathcal{X}}$ that satisfies (1.5) in the TT format.

Algorithm 1 TTSVD: Compute the TT format of a tensor $\mathcal{X}$.

Input: Tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, and a desired accuracy $0 < \epsilon < 1$
Output: The TT cores $\mathcal{G}_1, \ldots, \mathcal{G}_d$ of an approximation $\tilde{X}$

1: Set $s_0 = s_d = 1$.
2: Set $C = X_1$.
3: **for** $1 \leq j \leq d-1$ **do**
4: Compute SVD of $C = U\Sigma V^*$ with accuracy $\epsilon/\sqrt{d-1}$ and find rank s_j.
5: Set $\mathcal{G}_j = \operatorname{reshape}(U, s_{j-1}, n_j, s_j)$.
6: Set $C = \operatorname{reshape}(\Sigma V^*, s_j n_{j+1}, \prod_{k=j+2}^{d} n_k)$.
7: Set $\mathcal{G}_d = \operatorname{reshape}(C, s_{d-1}, n_d, s_d)$.

Fundamental tensor operations can be performed easily and efficiently on tensors in TT format. For example, the TT cores of the tensor $\mathcal{Y} = \mathcal{X} \times_j A$ are the same as those of $\mathcal{X}$ except for the jth TT corer updated to $\mathcal{G}_j \times_2 A$ [133]. Besides,

with the TT-rounding algorithm [164], which obtains a better TT format with a smaller storage cost given a rough TT approximation, one can carry out tensor addition, Frobenius norm computation, and conversion between different low rank formats [164].

1.2.2 Quantized tensor-train format

The QTT format is an extension of the TT format to low dimensional data [118, 163]. It is used vastly in solving PDEs such as integral equations [48,49,115], and design fast numerical linear algebra routines [59,114]. For a vector $v \in \mathbb{C}^{\prod_{j=1}^{d} n_j}$, its QTT format is the TT format of a reshaped tensor $\mathcal{V} = \mathrm{reshape}(v, n_1, \ldots, n_d)$. The TT rank of $\mathcal{V}$ is then referred to as the QTT rank of v. In many scenarios, $n_1 = \cdots = n_d = 2$ or 3, and classical examples include a discretization of the exponential function on a uniform grid, which has QTT rank 1 [118]. Broadly speaking, we consider the QTT format to be any matrix or tensor that contains the same elements as the original vector, as long as this transformation gives rise to low QTT rank. For example, the matrix $V = \mathrm{reshape}(v, \prod_{j=1}^{\lfloor d/2 \rfloor} n_j, \prod_{j=\lfloor d/2 \rfloor + 1}^{d} n_j)$ is another QTT format of v.

There are two ways to obtain the QTT format of a matrix A. In one way, all elements of A are reorganized into a high dimensional tensor, and TT decomposition algorithms are carried out on this single tensor. Alternatively, one can compute the QTT format of each column of A, and the TT cores for each reshaped column are treated and stored separately. In practice, one can choose the best representation technique based on the exact dataset or application problem.

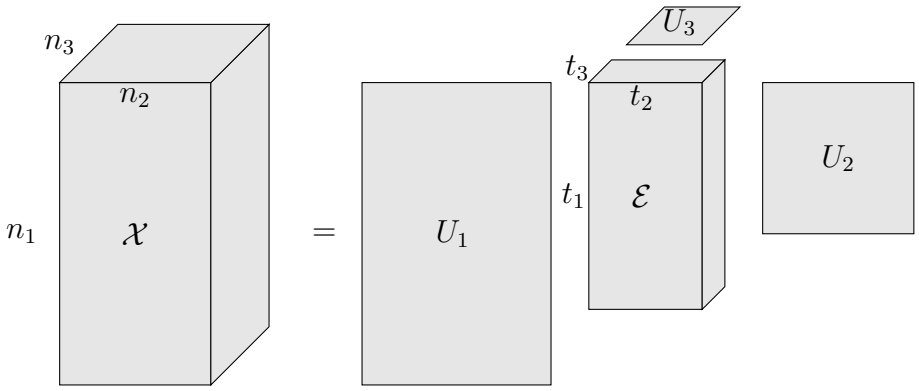

Figure 1.2: The orthogonal Tucker format with size of the factor matrices $\boldsymbol{t} = (t_1, t_2, t_3)$. The factor matrices U_1, U_2, and U_3 have orthonormal columns.

1.2.3 Orthogonal Tucker format

The orthogonal Tucker format is another well-known tensor format and is widely used in signal processing [54, 154], image processing [155, 215, 216, 218, 220], and data mining [181, 182, 202–204]. It represents a tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ with a core tensor $\mathcal{E} \in \mathbb{C}^{t_1 \times \cdots \times t_d}$ and a set of factor matrices $U_1, \ldots, U_d$ with orthonormal columns [53, 123]:

$$\mathcal{X} = [\![\mathcal{E}; U_1, \ldots, U_d]\!] = \mathcal{E} \times_1 U_1 \times_2 \cdots \times_d U_d, \qquad U_k \in \mathbb{R}^{n_k \times t_k}. \tag{1.7}$$

In this case, we call $\boldsymbol{t} = (t_1, \ldots, t_d)$ the size of the factor matrices of $\mathcal{X}$ and it provides an entry-by-entry bound on the multilinear rank $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_d)$. Such a decomposition contains $p^{\mathrm{ML}} \leq \sum_{k=1}^{d} n_k t_k + \prod_{k=1}^{d} t_k$ degrees of freedom, which is linear in size $\boldsymbol{n} = (n_1, \ldots, n_d)$, and still exponential in dimension d and thus can be infeasible for large d. Nevertheless, the Tucker format is very useful when each entry t_j is significantly smaller than the corresponding mode size n_j. Figure. 1.2 illustrates an orthogonal Tucker format of a 3D tensor with size of the factor matrices $\boldsymbol{t}$.

The higher-order SVD, or HOSVD (see Algorithm 2), can be used to compute

the orthogonal Tucker format of a given tensor [53]. By using the fact that tensor matricization ranks are bounds of the corresponding multilinear ranks

$$\ell_k \leq \operatorname{rank}(X_{(k)}), \quad 1 \leq k \leq d. \tag{1.8}$$

This algorithm utilizes an orthonormal basis of each tensor matricization as the corresponding factor matrix and computes the core tensor with these matrices. Therefore, finding the factor matrices in parallel is easy, as the matricizations are independent and can be handled simultaneously. In terms of accuracy, if the factor matrices are calculated via SVDs with $\epsilon/\sqrt{d}$ accuracy in the Frobenius norm and $0 < \epsilon < 1$, then we obtain an approximation $\tilde{\mathcal{X}}$ that satisfies 1.5 in the orthogonal Tucker format.

Algorithm 2 HOSVD: Compute the orthogonal Tucker format of a tensor $\mathcal{X}$.

Input: Tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, and a desired accuracy $0 < \epsilon < 1$
Output: Tucker core $\mathcal{E}$ and factor matrices $U_1, \ldots, U_d$ of an approximation $\tilde{X}$

1: **for** $1 \leq j \leq d$ **do**
2: Compute SVD of $X_{(j)} = U_j \Sigma_j V_j^*$ with accuracy $\epsilon/\sqrt{d}$ and find rank t_j.
3: Compute $\mathcal{E} = \mathcal{X} \times_1 U_1^* \cdots \times_d U_d^*$.

Recompressions of sub-optimal Tucker or orthogonal Tucker formats are easy to compute, since one only needs to take SVD of the factor matrices, uses the orthonormal matrix that approximates the column space as new factor matrices, and merges the rest of the information into the core tensor via tensor-matrix products. For example, if $\mathcal{Y} = \mathcal{X} \times_j A$, and $\mathcal{X} = [\![\mathcal{E}; U_1, \ldots, U_d]\!]$, then $\mathcal{Y} = [\![\mathcal{E}; U_1, \ldots, U_{j-1}, AU_j, U_{j+1}, \ldots, U_d]\!]$ and recompression shall be performed on AU_j. Other operations made possible by Tucker recompressions include the Hadamard product between two tensors [126].

1.2.4 Canonical Polyadic format

The canonical polyadic (CP) decomposition expresses a tensor as a sum of rank-1 tensors. A tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ is of rank at most r, if there are matrices $A^{(1)}, \ldots, A^{(d)}$ and a diagonal tensor D that

$$\mathcal{X} = [\![\mathcal{D}; A^{(1)}, \ldots, A^{(d)}]\!], \qquad A^{(k)} \in \mathbb{C}^{n_k \times r}, \quad \mathcal{D} \in \mathbb{C}^{r \times \cdots \times r}, \tag{1.9}$$

where the only nonzero entries of $\mathcal{D}$ are $\mathcal{D}_{i,\ldots,i}$ for $1 \leq i \leq r$. If $\mathcal{D}$ is omitted in this bracket notation, then by convention all the nonzero entries of $\mathcal{D}$ are 1. This tensor decomposition can be stored using $p^{\mathrm{CP}} \leq r + r \sum_{k=1}^{d} n_k$ degrees of freedom, but the decomposition is NP-hard to compute for worst case examples [99]. The CP decomposition in (1.9) is similar to the orthogonal Tucker decomposition with two important differences: (1) The matrices $A^{(1)}, \ldots, A^{(d)}$ in (1.9) do not need to have orthogonal columns and (2) The core tensor $\mathcal{D}$ must be diagonal. This means that (1.9) is equivalent to expressing a tensor as a sum of rank-1 tensors. However, due to the lack of orthonormality, we are not able to perform truncation to R terms and claim that this is the best rank-R approximation. Figure. 1.3 illustrates a CP format of a 3D tensor with rank at most R. In practice, the alternating least squares (ALS) algorithm [39, 97] is the workhorse to compute a CP decomposition. It allows for a desired rank R, optimizes the choice of each factor matrix while keeping others fixed, and repeats this iteration until convergence. The ALS algorithm is simple to understand and implement, but convergence can be slow, and there are no guarantees for convergence.

To obtain an upper bound on the rank of a tensor in CP format, we can take any decomposition of the form in (1.9) with a potentially large r, and see if its factor matrices $A^{(1)}, \ldots, A^{(d)}$ are themselves low rank. For example, we find

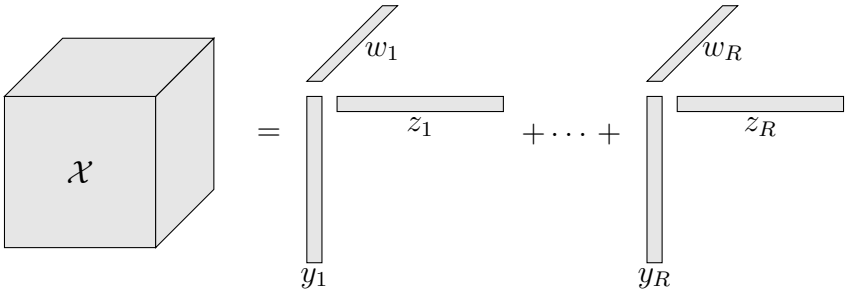

Figure 1.3: The CP format with rank at most R. The tensor $\mathcal{X}$ can be expressed as a sum of at most R rank-1 tensors.

that [129, Lem. 1]:[1]

$$\operatorname{rank}(\mathcal{X}) \leq \min_{1 \leq j \leq d} \frac{1}{r_j} \prod_{i=1}^{d} r_i, \tag{1.10}$$

where $r_i = \operatorname{rank}(A^{(i)})$ for $1 \leq i \leq d$. The bound in (1.10) is useful because it allows one to derive upper bounds on the rank of a tensor via bounds on the rank of factor matrices.

1.2.5 Computing with different tensor formats

Over the past two decades, researchers have developed multiple software packages to use low rank tensor formats in various software computing environments. For example, there are a handful of mature software packages in MATLAB and frequently used by researchers. Efficient computations with general and sparse tensors, and factored tensors in CP and Tucker formats are implemented in the Tensor Toolbox [14, 122]. The TT Toolbox [161] has fast implementations of TTSVD, TT-rounding, and tensor arithmetics in the TT format. It also includes an adaptive cross approximation (ACA) algorithm [183], which is a data-driven TT decomposition algorithm that uses a selection of

[1]Lemma 1 of [129] shows that the dimension of the vector space that spans the slices in the νth index is equal to the rank of $\mathcal{X}$. The inequality in (1.10) follows from the extra assumption that the slices are themselves low rank tensors.

the tensor entries as input. There are also Tensorlab [217] for structured data fusion, SPLATT [197] for high-performance sparse tensor factorization, and htucker [128] for hierarchical Tucker formats [84].

1.3 Why is dimensionality increase important?

The goal of compressing tensors with low rank formats is to reduce the degrees of freedom used for storage and computations, and in turn alleviate the "curse of dimensionality". However, in some scenarios, dimensionality increase provides an alternate perspective for simplifying a problem statement, leading to better-defined problem variants and more efficient algorithms.

A typical example is described in [147], where extra dimensions are introduced to remove degeneracies caused by arbitrary inputs in many geometric algorithms such as linear programming (LP-type) problems. The authors show that in order to remove degeneracies in a d-dimensional LP-type problem, one needs to increase the dimension to at least $d(1+\epsilon)$ where $\epsilon > 0$ is an arbitrary constant. Another example is to discover governing equations of nonlinear dynamical systems from data with Sparse Identification (SINDy) [36]. The original problem dimension is determined by the domain of the dynamical system, but users search through the space of functions, which is infinitely dimensional, for a handful of optimal governing functions. In this thesis, we focus on solving fractional PDEs using dimensionality increase (see Chapter 4), where an extended dimension is used to rewrite the nonlocal fractional operator into a local one [37]. As a result, our numerical solvers deal with geometries that are one dimension higher than the original formulation.

In some scenarios, dimensionality increase is a prerequisite for dimensionality reduction. The QTT tensor format (see Section 1.2.2) converts data from low dimensions into high-dimensional tensors, so that significant compressibility can be achieved on sampling special functions such as exponential functions and some trigonometric functions [118]. Additionally, Sylvester equations (see Section 1.4) can be considered as a conversion from linear systems using dimensionality increase. With this transformation, we only need to solve equations of much smaller scales. Sylvester equation solvers that aim to compute low rank factors of the solution are then dimensionality reduction techniques. With such combination of dimesionality manipulations, we can efficiently solve certain structured linear systems using available computing resources.

1.4 Sylvester equations and displacement structure

A Sylvester matrix equation for an unknown matrix X has the form

$$AX - XB^T = F, \quad A \in \mathbb{C}^{m\times m},\ B \in \mathbb{C}^{n\times n},\ F \in \mathbb{C}^{m\times n}. \tag{1.11}$$

This type of equation appear often in PDE discretizations with any Laplace-type operators [74, 193, 210, 230], and in stability analysis of dynamical systems [10, 90, 168]. In this thesis, we assume A and B in (1.11) are normal matrices, then X has a unique solution if the spectra of A and B are disjoint [195]. This can be derived by rewriting (1.11) into a linear system

$$(I_n \otimes A - B \otimes I_m)x = f, \tag{1.12}$$

where $x = X(:), f = F(:)$, I_n and I_m are identity matrices with size n and m respectively, and $\otimes$ denotes the Kronecker product of two matrices. Then disjoint spectra of A and B implies the matrix $L = I_n \otimes A - B \otimes I_m$ does not have

eigenvalue 0. In the special case that $A = B$ and F is symmetric, this equation is widely known as a Lyapunov equation. Additionally, a matrix X is said to have a displacement structure if it satisfies a Sylvester equation (1.11), and the rank of F is called the displacement rank of X. Many important matrix families in computational mathematics, including Toeplitz, Hankel, Vandermonde, and Cauchy matrices, satisfy displacement structures, with low displacement rank. In [24], the authors use this structure to explain why certain matrices (including Löwner, Pick, Cauchy, real Vandermonde, and real positive definite Hankel), are often of low numerical rank.

A Sylvester tensor equation for a tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ has the form

$$\mathcal{X} \times_1 A_1 + \cdots + \mathcal{X} \times_d A_d = \mathcal{F}, \quad \mathcal{F} \in \mathbb{C}^{n_1 \times \cdots \times n_d},\ A_j \in \mathbb{C}^{n_j \times n_j},\ 1 \leq j \leq d, \tag{1.13}$$

and $\mathcal{X}$ admits a unique solution if the Minkowski sum of the spectra of A_j for all j does not contain 0. Similarly, if a tensor $\mathcal{X}$ satisfies (1.13), then it is said to have a displacement structure.

We now describe various types of methods for Sylvester matrix equations (1.11), and we develop and use high-dimensional analogues of these solvers for Sylvester tensor equations in Chapters 2, 3, 4, and 5.

1.4.1 Direct solvers

The most naive way of solving (1.11) is to construct the matrix $L = I_n \otimes A - B \otimes I_m$, and solve (1.12). Another direct solver is feasible if computing eigendecompositions of A and B are stable. We can then use their eigenvalues and eigenvectors to convert the equation into a diagonal system, and find the so-

lution easily via direct scaling (see Algorithm 3). This algorithm is useful in scenarios that A and B are symmetric.

Algorithm 3 Eig-Sylv: Compute the solution of (1.11) with eigendecomposition.

Input: Matrices A, B, and F in (1.11)
Output: A matrix X satisfying $AX - XB^T = F$
1: Compute eigendecompositions $A = V_A D_A V_A^{-1}$ and $B = V_B D_B V_B^{-1}$.
2: Set $G = V_A^{-1} F V_B^{-T}$.
3: Compute $Y_{i,j} = G_{i,j}/((D_A)_{i,i} - (D_B)_{j,j})$ for $1 \le i \le m$ and $1 \le j \le n$.
4: Set $X = V_A Y V_B^T$.

Another efficient and stable direct method is the Bartels–Stewart algorithm [20], which uses the Schur factorizations of A and B, and computes each entry of X by solving triangular linear systems. Another benefit of this method is that it can be extended to solve generalized Sylvester matrix equations of the form

$$AXB - CXD = F, \quad A, C \in \mathbb{C}^{m\times m},\ B, D \in \mathbb{C}^{n\times n},\ F \in \mathbb{C}^{m\times n}. \tag{1.14}$$

In this setting, real QZ decomposition of the pairs (A, C) and (B, D) are computed, and the solution X can be found column by column via triangular solves (see Algorithm 4). For some special generalized Sylvester tensor equations, a generalization of this method becomes the only feasible solver.

1.4.2 The ADI method

In PDE discretizations [74, 193, 210, 230] and stability analysis of dynamical systems [10, 90, 168], the matrices A and B in (1.11) are often constructed to be sparse and/or structured, such as banded and banded plus low-rank. In these scenarios, we want solvers that utilize these structures. These include rational

Algorithm 4 Bartels–Stewart: Compute the solution of (1.14).

Input: Matrices A, B, C, D, and F in (1.14)
Output: A matrix X satisfying $AXB - CXD = F$

Compute real QZ decomposition $A = Q_1^T T_A Z_1^T$ and $C = Q_1^T T_C Z_1^T$.
Compute real QZ decomposition $B = Q_2^T T_B Z_2^T$ and $D = Q_2^T T_D Z_2^T$.
Set $G = Q_1 F Z_2$.
Set $j = n$.
while $j > 0$ **do**
    Set $H_j = G_{:,j} - T_A \sum_{k=j+1}^{n} (T_B)_{j,k} Y_{:,k} + T_C \sum_{k=j+1}^{n} (T_D)_{j,k} Y_{:,k}$.
    if $(T_D)_{j,j-1} = 0$ **then**
        Solve $((T_B)_{j,j} T_A - (T_D)_{j,j} T_C) Y_{:,j} = H_j$.
        Set $j = j - 1$.
    else
        Set $H_{j-1} = G_{:,j-1} - T_A \sum_{k=j+1}^{n} (T_B)_{j-1,k} Y_{:,k} + T_C \sum_{k=j+1}^{n} (T_D)_{j-1,k} Y_{:,k}$.
        Set $L = \begin{bmatrix} (T_B)_{j,j} T_A - (T_D)_{j,j} T_C & -(T_D)_{j,j-1} T_C \\ (T_B)_{j-1,j} T_A - (T_D)_{j-1,j} T_C & (T_B)_{j-1,j-1} T_A - (T_D)_{j-1,j-1} T_C \end{bmatrix}$.
        Solve $L \begin{bmatrix} Y_{:,j} \\ Y_{:,j-1} \end{bmatrix} = \begin{bmatrix} H_j \\ H_{j-1} \end{bmatrix}$.
        Set $j = j - 2$.
Set $X = Z_1 Y Q_2$.

Krylov subspace solvers [64, 194, 195], the alternating direction implicit (ADI) method [27, 136, 168, 179], and many others [28, 167]. In this thesis, we focus on the ADI method, which is an iterative algorithm based on rational function approximation.

The ADI algorithm consists of solving sequences of shifted linear systems with respect to A and B, and we describe the full procedure in Algorithm 5.

The shifts used in the iterations are known in many situations [74, 211]. For example, one set of shift parameters $\boldsymbol{p}$ and $\boldsymbol{q}$ can be chosen as the zeros and poles of a rational function $r \in \mathcal{R}_{k,k}$, that can achieve a quasi-optimal Zolotarev number [234] (see Sec. 2.3.1)

$$Z_k(\Lambda(A), \Lambda(B)) := \inf_{r \in \mathcal{R}_{k,k}} \frac{\sup_{z \in \Lambda(A)} |r(z)|}{\inf_{z \in \Lambda(B)} |r(z)|}, \qquad k \geq 0,$$

Algorithm 5 ADI: Given a Sylvester matrix equation (1.11), compute an approximate solution.

Input: Matrices A, B, and F in (1.11), and a desired accuracy $0 < \epsilon < 1$
Output: An approximate solution $\tilde{X}$ to $AX - XB^T = F$

1: Select 2ℓ shift parameters $p_1, \ldots, p_\ell$, and $q_1, \ldots, q_\ell$ using A and B.
2: Set $X_1 = 0$.
3: **for** $1 \leq j \leq \ell$ **do**
4: Solve $(A - q_j I_m)X_{j+1/2} = X_j(B - q_j I_n)^T + F$.
5: Solve $X_{j+1}(B - p_j I_n)^T = (A - p_j I_m)X_{j+1/2} - F$.
6: Set $\tilde{X} = X_{\ell+1}$.

where $\Lambda(A)$ and $\Lambda(B)$ are the spectra of A and B, and $\mathcal{R}_{k,k}$ is the set of rational functions of the form $s(x)/t(x)$ with polynomials s and t of degree at most k. This choice is closely related to the fact that Zolotarev numbers can be used to bound approximations of X that satisfies (1.11) [24, 193]; namely,

$$\|X - X_k\|_2 \leq Z_k(\Lambda(A), \Lambda(B))\|X\|_2, \tag{1.15}$$

and

$$\|X - X_k\|_F \leq Z_k(\Lambda(A), \Lambda(B))\|X\|_F, \tag{1.16}$$

where X_k is the best rank-k approximation of X.

1.4.3 The fADI method

Apart from structured A and B, the matrix F has low rank factorization $F = UV^*$ with $U \in \mathbb{C}^{m\times r}$ and $V \in \mathbb{C}^{n\times r}$ in many cases. Therefore, we want a solver that also obtains an approximation of X in low rank format. By expressing X_{j+1} in ADI iterations with $X_j = Z_j D_j Y_j^*$, we obtain the factored ADI (fADI) algorithm, which is described in details in Algorithm 6. The main takeaway from the fADI method is that it solves for bases of the column space and row

Algorithm 6 fADI: Given a Sylvester matrix equation (1.11), compute an approximate solution in low rank form.

Input: Matrices A, B in (1.11) U, V such that $F = UV^*$, and a desired accuracy $0 < \epsilon < 1$

Output: Factor matrices Z, D, and Y of the approximate solution $\tilde{X} = ZDY^*$ to $AX - XB^T = UV^*$

1: Select 2ℓ shift parameters $p_1, \ldots, p_\ell$, and $q_1, \ldots, q_\ell$ using A and B.
2: Solve $(A - q_1 I_m)Z_1 = U$. Let $Z = Z_1$.
3: Solve $(B - \overline{p_1} I_n)Y_1 = V$. Let $Y = Y_1$.
4: Let $D = (q_1 - p_1)I_r$.
5: **for** $1 \leq j \leq \ell - 1$ **do**
6: Set $G_j = (q_{j+1} - p_j)Z_j$, and $H_j = (\overline{p_{j+1}} - \overline{q_j})Y_j$.
7: Solve $(A - q_{j+1} I_m)Z_{j+1} = G_j$. Set $Z_{j+1} = Z_{j+1} + Z_j$ and $Z = \begin{bmatrix} Z & Z_{j+1} \end{bmatrix}$.
8: Solve $(B - \overline{p_{j+1}} I_n)Y_{j+1} = H_j$. Set $Y_{j+1} = Y_{j+1} + Y_j$ and $Y = \begin{bmatrix} Y & Y_{j+1} \end{bmatrix}$.
9: Set $D = \begin{bmatrix} D & \\ & (q_{j+1} - p_{j+1})I_r \end{bmatrix}$.

space of X independently. Therefore, in certain applications, one can only solve for a basis for column/row space and save half of the computing powers.

When Algorithm 6 terminates, the factor matrices Z, D, and Y are generally not optimal in size since they are constructed by stacking solution matrices from all iterations. There are mainly two ways to obtain ideal factors: (1) Recompressing after each iteration so that one never works with large matrices, and (2) Recompressing the final Z, D, and Y after all iterations finish, so an SVD is computed only once. In practice, one can use the problem size as a criterion for a heuristic selection process. When A and B are of moderate size, we can afford multiple SVDs so we can use (1). Comparatively when A and B are large in size, a single SVD is preferred to avoid high computation costs in each iteration. One can also choose a recompression scheme in between (1) and (2) to optimize size of SVD and number of SVD computed.

CHAPTER 2

ON THE COMPRESSIBILITY OF TENSORS

We say that a tensor is compressible if it can be approximated by a low rank tensor, in the sense of (1.5) that can be represented in a relative small number of degrees of freedom. In this chapter[1], we derive bounds on these numerical storage costs (see Sec. 1.2) for certain families of tensors in three tensor decompositions: (a) Tensor-train decomposition (see Sec. 1.2.1), (b) Orthogonal Tucker decomposition (see Sec. 1.2.3), and (c) Canonical Polyadic (CP) decomposition (see Sec. 1.2.4). In doing so, we partially justify the use of low rank tensor decompositions. Analogous theoretical results have already been derived that explicitly bound the numerical rank of matrices [24, 146, 175, 211].

We explore three methodologies to bound the compressibility of a tensor:

- **Algebraic structures:** If a tensor is constructed by sampling a multivariable function that can be expressed as a sum of products of single-variable functions, then that tensor is often compressible. Occasionally, one may have to perform algebraic manipulations to a function to explicitly reveal its desired structure, for example, by using trigonometric identities (see Sec. 2.1).
- **Smoothness:** If a tensor can be constructed by sampling a smooth function on a tensor-product grid, then that tensor is often compressible. This observation can be made rigorous by using the fact that smooth functions can be well-approximated by polynomials in a compact domain [212, Theorem 8.2] and [92, Lem. 14.5].
- **Displacement structure:** If a tensor $\mathcal{X}$ satisfies a multidimensional Sylvester

[1]This chapter is based on a paper with Alex Townsend [193]. I derived the theorems and algorithms, and was the lead author of the manuscript.

Table 2.1: Summary of the bounds of storage costs in tensor-train format of $n \times n \times n$ tensors explored in this manuscript. The numbers s_1, s_2, and s_3 are given in their corresponding sections.

Tensor	TT Storage bound	Method	Ref.
Sampled $e^{iM\pi xyz}$	$\mathcal{O}(M)$	Smoothness	Sec. 2.2.1
Sampled sum of Gaussian bumps	Implicit	Smoothness	Sec. 2.2.2
3D Hilbert tensor	$n(s_1^2 + 2s_1)$	Displacement	Sec. 2.4.1
Poisson FD soln	$n(s_2^2 + 2s_2)$	Displacement	Sec. 2.4.2
Poisson spectral soln	$n(s_3^2 + 2s_3)$	Displacement	Sec. 2.4.2

equation of the form (1.13), then this—under additional assumptions—can ensure that the tensor $\mathcal{X}$ is well-approximated by a low rank tensor. Multidimensional Sylvester equations such as (1.13) appear when discretizing certain partial differential equations with finite differences [134] and are satisfied by several classes of structured tensors [88]. For example, we show that the solution tensor $\mathcal{X} \in \mathbb{C}^{n\times n\times n}$ to $-\nabla^2 u = 1$ on $[-1, 1]^3$ with zero Dirichlet conditions can be represented up to a relative accuracy of $0 < \epsilon < 1$ in the Frobenius norm with just $\mathcal{O}(n(\log n)^2(\log(1/\epsilon))^2)$ degrees of freedom in tensor-train format, despite the solution having weak corner singularities.

The first two methodologies are considered in [92, 108, 117] and analogous results for matrices are available in the literature [175, 209]. The third methodology is used in various aspects of numerical linear algebra. For example, one can explicitly bound the singular values of matrices with displacement structure [24,211], which can make matrix-vector multiplication [93] and the solution of linear systems [83, 157] more computationally efficient. The third methodology is also related to the technique of bounding singular values using exponential sums [35, 116]. However, we are not aware of any existing literature that compress tensors with displacement structure. In this manuscript, we for-

mally provide bounds on the compressibility of such tensors and illustrate the methodologies with worked examples. Table. 2.1 summarizes our bounds on the storage cost of several special tensors.

After some experience, one can successfully identify which methodology is likely to result in the best theoretical bounds on the compressibility of a tensor. We emphasize that these three methodologies provide upper bounds using numerical tensor ranks, and do not provide a complete characterization on the compressibility of tensors. Another approach that partially explains the abundance of tensors with small storage is artificial coordinate alignment [213], though we do not know how to use this observation to derive explicit bounds on tensor ranks.

In Sec. 2.1 we study the ranks of tensors that are constructed by sampling multivariate functions that have some algebraic structure. Then, in Sec. 2.2, we consider the storage cost of tensors constructed by sampling smooth multivariate functions. Finally, in Sec. 2.3 we consider tensors that satisfy a multidimensional Sylvester equation, including a fast tensor Sylvester equation solver that exploits the compressibility of these tensors in Sec. 2.4.3.

2.1 Tensors constructed via sampling algebraically structured functions

In practice, one often encounter tensors that are sampled from multivariate functions. For example, one can take a continuous function of three variables,

$f(x, y, z)$, and sample f on a tensor grid to obtain a tensor:

$$\mathcal{X}_{ijk} = f(x_i, y_j, z_k), \qquad 1 \le i, j, k \le n,$$

where $\{x_1, \ldots, x_n\}$, $\{y_1, \ldots, y_n\}$, and $\{z_1, \ldots, z_n\}$ are sets of points.

2.1.1 Polynomials and algebraic structure

One common scenario where it is easy to spot compressible tensors is when the tensor is sampled from a polynomial. To be specific, if a tensor $\mathcal{X}$ is derived by sampling a multivariate polynomial $p(x_1, \ldots, x_d)$ of degree at most $N_j - 1$ in the variable x_j from a tensor-product grid, then one finds that $\mathcal{X}$ is highly compressible.

Lemma 2.1.1. *Let $p(x_1, \ldots, x_d)$ be a polynomial of degree at most $N_j - 1$ in the variable x_j for $1 \le j \le d$, and let $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ be the tensor constructed by sampling p, i.e.,*

$$\mathcal{X}_{i_1,\ldots,i_d} = p(x_{i_1}^{(1)}, \ldots, x_{i_d}^{(d)}), \qquad 1 \le i_j \le n_j, \qquad 1 \le j \le d,$$

where $x^{(1)}, \ldots, x^{(d)}$ are sets of $n_1, \ldots, n_d$ nodes, respectively. Then,

- $p^{\mathrm{TT}}(\mathcal{X}) \le \sum_{k=1}^{d} t_{k-1} t_k n_k$, where $t_k = \min\{\prod_{j=1}^{k} N_j, \prod_{j=k+1}^{d} N_j\}$,
- $p^{\mathrm{ML}}(\mathcal{X}) \le \sum_{k=1}^{d} n_k N_k + \prod_{k=1}^{d} N_k$, *and*
- $p^{\mathrm{CP}}(\mathcal{X}) \le r + r \sum_{k=1}^{d} n_k$, *where* $r = \min_{1 \le k \le d} \frac{1}{N_k} \prod_{j=1}^{d} N_j$.

Here, the tensor-train decomposition is constructed in the order of $x_1, \ldots, x_d$.

Proof. According to the degree assumptions on p, we can write p as

$$p(x_1, \ldots, x_d) = \sum_{q_1=0}^{N_1-1} \cdots \sum_{q_k=0}^{N_k-1} a_{q_1,\ldots,q_k}(x_{k+1}, \ldots, x_d) x_1^{q_1} \cdots x_k^{q_k}, \qquad 1 \le k \le d,$$

where $a_{q_1,\ldots,q_k}(x_{k+1},\ldots,x_d)$ is a polynomial in the variables $x_{k+1},\ldots,x_d$ and for $k+1 \leq j \leq d$, x_j has degree at most $N_j - 1$. After sampling, this means that $\operatorname{rank}(X_k) \leq \min\{\prod_{j=1}^{k} N_j, \prod_{j=k+1}^{d} N_j\}$ and the bound on $p^{\mathrm{TT}}(\mathcal{X})$ follows.

Another way to write p is

$$p(x_1,\ldots,x_d) = \sum_{j=0}^{N_k-1} b_j(x_1,\ldots,x_{k-1},x_{k+1},\ldots,x_d)x_k^j, \qquad 1 \leq k \leq d,$$

where b_j is a polynomial in $x_1,\ldots,x_{k-1},x_{k+1},\ldots,x_d$ of degree at most $N_j - 1$ in x_j. After sampling, this shows that $\operatorname{rank}(X_{(k)}) \leq N_k$ and the bound on $p^{\mathrm{ML}}(\mathcal{X})$ follows.

Finally, separating out x_d, we can also write p as

$$p(x_1,\ldots,x_d) = \sum_{q_1=0}^{N_1-1} \cdots \sum_{q_{d-1}=0}^{N_{d-1}-1} c_{q_1,\ldots,q_{d-1}}(x_d)x_1^{q_1}\cdots x_{d-1}^{q_{d-1}}, \tag{2.1}$$

where each term in (2.1) is a rank 1 tensor after sampling. We can do this to each variable and thus $\operatorname{rank}(\mathcal{X}) \leq \min_{1\leq k\leq d} \frac{1}{N_k}\prod_{j=1}^{d} N_j$. The bound on $p^{\mathrm{CP}}(\mathcal{X})$ follows. □

A special case of Lemma 2.1.1 is when the polynomial p has maximal degree of at most $N-1$ so that $N_1 = \cdots = N_d = N$.[2] We find that

- $p^{\mathrm{TT}}(\mathcal{X}) \leq \sum_{k=1}^{d} N^{2t_k-1}n_k$, where $t_k = \min\{k, d-k\}$,
- $p^{\mathrm{ML}}(\mathcal{X}) \leq N\sum_{k=1}^{d} n_k + N^d$, and
- $p^{\mathrm{CP}}(\mathcal{X}) \leq N^{d-1}\sum_{k=1}^{d} n_k + N^{d-1}$.

The important observation is that tensors constructed by sampling polynomials on a grid are highly compressible. To be specific, the storage costs of these

[2]We say that a polynomial $p_N(x_1,\ldots,x_d)$ has maximal degree $\leq N$ if p_N is a polynomial of degree at most N in all the variables x_i.

tensors, when stored in tensor-train, Tucker, or CP format, do not grow exponentially with the dimension d but linearly. In addition, in scenarios that the tensors are constructed via oversampling the polynomial, which means grid sizes n_j's are much larger than polynomial degrees N_j's, we can use much smaller degrees of freedom to represent this oversampled tensor. If one is familiar with tensor ranks, then one may notice that terms related to N_j's are upper bounds of tensor-train, multilinear, and CP ranks. These turn out to be pretty tight bounds in practice.

There is generally not a prevalent format, in the sense that its storage cost is smaller than those of the other two. Depending on the grid sizes and polynomial degrees, all formats can have the smallest storage cost. Therefore, it is case specific to choose the optimal format for a given tensor.

2.1.2 Other special cases of algebraic structure

Similar to multivariate polynomials, it is easy to spot—after some experience—the mathematical tensor ranks of tensors constructed by sampling functions that have explicit algebraic structure since each variable in the function can be thought as a fiber of the tensor. The easiest ones to spot are those tensors derived from functions that are the sums of products of single-variable functions, such as

$$f(x, y, z) = 1 + \tan(x)y + y^2z^3, \quad g(x, y, z, w) = \cos(x)\sin(y) + e^{10z}e^{100w}.$$

If $\mathcal{F}$ and $\mathcal{G}$ are tensors constructed by sampling f and g on a $n \times n \times n$ and $n \times n \times n \times n$ tensor-product grid, respectively, then the storage costs in different

formats are bounded by

$$p^{\mathrm{TT}}(\mathcal{F}) \leq 8n, \qquad p^{\mathrm{ML}}(\mathcal{F}) \leq 7n + 12, \qquad p^{\mathrm{CP}}(\mathcal{F}) \leq 9n + 3,$$
$$p^{\mathrm{TT}}(\mathcal{G}) \leq 12n, \qquad p^{\mathrm{ML}}(\mathcal{G}) \leq 8n + 16, \qquad p^{\mathrm{CP}}(\mathcal{G}) \leq 8n + 2,$$

where the tensor-train decompositions are performed in the order x, y, z, w. Other examples are functions that can be expressed with exponentials and powers, and similar examples have also been considered [118, 160].

Some special functions require reorganizations to reveal their algebraic structures. If the function is expressed with trigonometric functions, then the sampled tensor can often be low rank due to trigonometric identities. For example, consider the function $f(x, y, z) = \cos(x + y + z)$ that is a special case of the examples in [30, 166]. Since it can be written as

$$f(x, y, z) = (\cos(x)\cos(y) - \sin(x)\sin(y))\cos(z) - (\sin(x)\cos(y) + \cos(x)\sin(y))\sin(z),$$

any tensor $\mathcal{F}$ constructed by sampling f on a $n \times n \times n$ tensor-product grid satisfies

$$p^{\mathrm{TT}}(\mathcal{F}) \leq 8n, \quad p^{\mathrm{ML}}(\mathcal{F}) \leq 6n + 8, \quad p^{\mathrm{CP}}(\mathcal{F}) \leq 12n + 4.$$

In addition, [151] provides further insights on reorganizing special functions of sum of variables to reveal its algebraic structure. These examples can often be combined to build more complicated functions that result in compressible tensors. This is an $ad\ hoc$ process and requires human ingenuity to express the sampled function in a revealing form. Again, tensors constructed by sampling such algebraically structured functions on a sufficiently large tensor-product grid can be represented using a small number of degrees of freedom.

2.2 Tensors derived by sampling smooth functions

Although most functions do not have the algebraic structure specified in Sec. 2.1, tensors that are constructed by sampling smooth functions are often well approximated by compressible tensors. In light of Lemma 2.1.1, our idea to understand the compressibility of a tensor derived from sampling a function is first to approximate that function by a multivariate polynomial, which is already a routine procedure for computing with low rank approximations to multivariate functions [98].

Without loss of generality, suppose that $\mathcal{X}$ is formed by sampling a smooth function f on a tensor-product grid in $[-1,1]^d$, i.e.,

$$\mathcal{X}_{i_1,\ldots,i_d} = f(x_{i_1}^{(1)},\ldots,x_{i_d}^{(d)}), \qquad 1 \le i_k \le n_k, \qquad 1 \le k \le d, \tag{2.2}$$

where $x^{(1)},\ldots,x^{(d)}$ are sets of $n_1,\ldots,n_d$ nodes in $[-1,1]$. Our idea is to find a multivariate polynomial p of degree $\le N_j - 1$ in the variable x_j that approximates f in $[-1,1]^d$ and then set $\mathcal{Y}_{i_1,\ldots,i_d} = p(x_{i_1}^{(1)},\ldots,x_{i_d}^{(d)})$. By Lemma 2.1.1, $\mathcal{Y}$ can be represented with a small number of degrees of freedom and $\mathcal{Y}$ is an approximation to $\mathcal{X}$. In particular, we have

$$\|\mathcal{X}-\mathcal{Y}\|_F \le \left(\prod_{i=1}^{d} n_i\right)^{\frac{1}{2}} \|\mathcal{X}-\mathcal{Y}\|_{\max} \le \left(\prod_{i=1}^{d} n_i\right)^{\frac{1}{2}} \|f-p_N\|_\infty, \tag{2.3}$$

where $\|\cdot\|_\infty$ denotes the supremum norm on $[-1,1]^d$ and $\|\cdot\|_{\max}$ is the absolute maximum entry norm. Therefore, if p is a good approximation to f, then $\mathcal{Y}$ is a good approximation to $\mathcal{X}$ too. Although the error bound is good for small d, this approximation still suffers from the curse of dimensionality for large d.

One can now propose any linear or nonlinear approximation scheme to find a polynomial approximation p of f on $[-1,1]^d$. Clearly, excellent bounds on

$\|\mathcal{X} - \mathcal{Y}\|_F$ are obtained by finding a p so that

$$\|f - p\|_\infty \approx \inf_{q \in \mathcal{P}_{N_1,\ldots,N_d}} \|f - q\|_\infty,$$

where $\mathcal{P}_{N_1,\ldots,N_d}$ is the space of d-variate polynomials of maximal degree $\leq N_i - 1$ in x_i for $1 \leq i \leq d$. This best multivariable polynomial problem is often, but not always, tricky to solve directly. In those cases, near-optimal polynomial approximations are used instead. One common choice is to use p as the multivariate Chebyshev projection of f. That is,

$$p^{\text{cheb}}_{N_1,\ldots,N_d}(x_1,\ldots,x_d) = {\sum_{i_1=0}^{N_1-1}}' \cdots {\sum_{i_d=0}^{N_d-1}}' c_{i_1,\ldots,i_d} T_{i_1}(x_1)\cdots T_{i_d}(x_d),$$

$$c_{i_1,\ldots,i_d} = \left(\frac{2}{\pi}\right)^d \int_{-1}^{1}\cdots\int_{-1}^{1} \frac{f(x_1,\ldots,x_d)T_{i_1}(x_1)\cdots T_{i_d}(x_d)}{\sqrt{1-x_1^2}\cdots\sqrt{1-x_d^2}} dx_1\cdots dx_d,$$

where the primes indicate that the first term in the sum is halved and $T_k(x)$ is the Chebyshev polynomial of degree k. Importantly, $p^{\text{cheb}}_{N_1,\ldots,N_d}$ is a near-best polynomial approximation to f [212], and the error $\|f - p^{\text{cheb}}_{N_1,\ldots,N_d}\|_\infty$ can be bounded. Thus, this choice of p leads to bounds on the compressibility of $\mathcal{X}$.

Next, we give two examples that illustrate how to understand the compressibility of tensors constructed by sampling smooth functions. We consider two functions: (1) A Fourier-like function, where we use best polynomial approximation, and (2) A sum of Gaussian bumps, where we use Chebyshev approximation.

2.2.1 Fourier-like function

Consider a tensor $\mathcal{X} \in \mathbb{C}^{n\times n\times n}$ constructed by sampling the following Fourier-like function on a tensor-product grid [209]:

$$f(x,y,z) = e^{iM\pi xyz}, \qquad x,y,z \in [-1,1],$$

where $M \geq 1$ is a real parameter. While representing $\mathcal{X}$ exactly requires n^3 degrees of freedom, it can be approximated by tensors that require fewer degrees of freedom (in the tensor-train and Tucker formats). To see this, let p_{k-1}^{best} and q_{k-1}^{best} be the best minimax polynomial approximations of degree $\leq k-1$ to $\cos(M\pi t)$ and $\sin(M\pi t)$ on $[-1,1]$, respectively, and define $h_{k-1} = p_{k-1}^{\text{best}} + iq_{k-1}^{\text{best}}$. Note that $h_{k-1}(xyz)$ has maximal degree at most $k-1$ so that

$$\begin{aligned}\inf_{w_{k-1}\in\mathcal{P}_{k-1}} \sup_{x,y,z\in[-1,1]} \left|e^{iM\pi xyz} - w_{k-1}(x,y,z)\right| &\leq \sup_{x,y,z\in[-1,1]} \left|e^{iM\pi xyz} - h_{k-1}(xyz)\right| \\ &= \sup_{t\in[-1,1]} \left|e^{iM\pi t} - h_{k-1}(t)\right|,\end{aligned}$$

where $\mathcal{P}_{k-1}$ is the space of trivariate polynomials of maximal degree $\leq k-1$ and the equality follows since $t = xyz \in [-1,1]$ if $x, y, z \in [-1,1]$. Furthermore, we have $e^{iM\pi t} = \cos(M\pi t) + i\sin(M\pi t)$ and so

$$\sup_{t\in[-1,1]} \left|e^{iM\pi t} - h_{k-1}(t)\right| \leq \sup_{t\in[-1,1]} \left|\cos(M\pi t) - p_{k-1}^{\text{best}}(t)\right| + \sup_{t\in[-1,1]} \left|\sin(M\pi t) - q_{k-1}^{\text{best}}(t)\right|.$$

By the equioscillation theorem [172, Theorem 7.4], $p_{k-1}^{\text{best}} = 0$ for $k-1 \leq 2\lfloor M\rfloor - 1$ since $\cos(M\pi t)$ equioscillates $2\lfloor M\rfloor + 1$ times in $[-1,1]$. Similarly, $\sin(M\pi t)$ equioscillates $2\lfloor M\rfloor$ times in $[-1,1]$ and hence, $q_{k-1}^{\text{best}} = 0$ for $k-1 \leq 2\lfloor M\rfloor - 2$. However, for $k > 2\lfloor M\rfloor$, $\sup_{t\in[-1,1]}\left|e^{iM\pi t} - h_{k-1}(t)\right|$ decays super-geometrically to zero as $k \to \infty$. This also indicates that the error between the tensors sampled from $e^{iM\pi xyz}$ and $h_{k-1}(x,y,z)$ rapidly goes to 0 as $k \to \infty$. Hence, the numerical maximal degree, N_ϵ, of $e^{iM\pi xyz}$ satisfies $N_\epsilon/2M \to c$ for some constant $c \geq 1$ as $M \to \infty$. Lemma 2.1.1 shows that an approximant to $\mathcal{X}$ only requires $\mathcal{O}(M)$ degrees of freedom. In particular, if s_1 is the second element of the tensor-train rank of an approximant tensor to the one sampled by $e^{iM\pi xyz}$, then $s_1/(2M) \to 1$ as $M \to \infty$.

Fig. 2.1 (left) plots the ratio of the second element of the tensor-train rank,

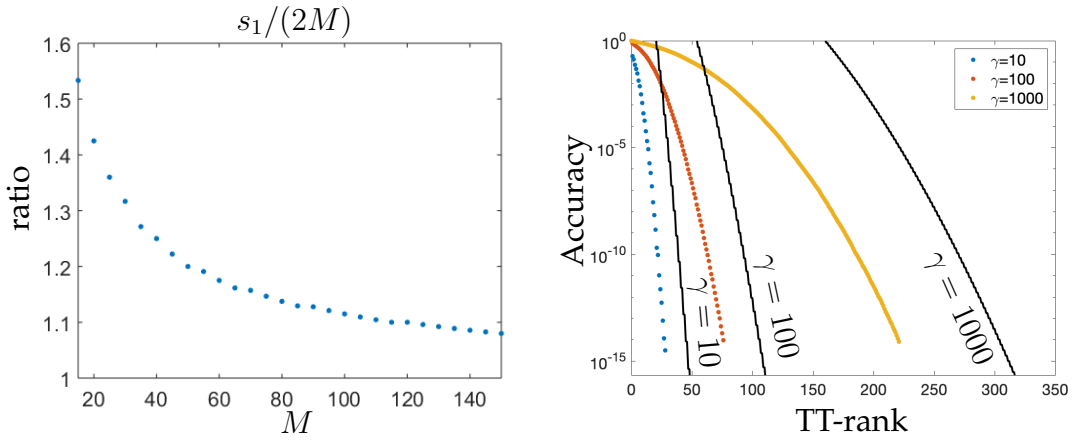

Figure 2.1: Left: The ratio of the second element of the tensor-train rank, s_1, of the tensors of size $n \times n \times n$ with $n = 600$ constructed by sampling the Fourier-like function e^{iMxyz} with $15 \leq M \leq 150$. The accuracy used to calculate the tensor-train ranks is 10^{-10}. Right: The second element, s_1, of the tensor-train rank (blue, red, and yellow dots) calculated with the TTSVD and the theoretical bounds (black lines) of $\mathcal{X} \in \mathbb{C}^{400\times400\times400}$ constructed by sampling $\sum_{j=1}^{300} e^{-\gamma((x-x_j)^2+(y-y^j)^2+(z-z_j)^2)}$ on an equispaced tensor-product grid for $\gamma = 10, 100, 1000$, where (x_j, y_j, z_j) are arbitrary centers in $[-1, 1]^3$.

s_1, of a tensor sampled from the Fourier-like function and $2M$. We observe that $s_1/(2M) \to 1$ as $M \to \infty$.

2.2.2 A sum of Gaussian bumps

Consider a tensor $\mathcal{X} \in \mathbb{C}^{n\times n\times n}$ constructed by sampling a sum of M Gaussian bumps, centered at arbitrary locations $(x_1, y_1, z_1), \ldots, (x_M, y_M, z_M)$ in $[-1, 1]^3$, i.e.,

$$f(x, y, z) = \sum_{j=1}^{M} e^{-\gamma((x-x_j)^2+(y-y_j)^2+(z-z_j)^2)}, \qquad \gamma > 0. \tag{2.4}$$

Each Gaussian bump is a separable function of three variables so, mathematically, the tensor ranks of $\mathcal{X}$ depend linearly on M. However, since the sum is a smooth function, the ranks are related to the polynomial degree required to approximate $f(x, y, z)$ in $[-1, 1]^3$ to an accuracy of $0 < \epsilon < 1$. Hence, the tensor

ranks of X depend on γ and have very mild growth in M in the storage costs.

Exponential sums have been used to approximate f in [34, Chpt. 6], but here we instead approximate it with a Chebyshev series. Due to the symmetry in x, y, and z as well as separability of each term in (2.4), we find that the Chebyshev approximation to $f(x, y, z)$ can be bounded by

$$\sup_{x,y,z\in[-1,1]} \left| f(x,y,z) - \sum_{j=1}^{M} p_\ell^j(x) q_\ell^j(y) r_\ell^j(z) \right| \leq 3M \sup_{x\in[-1,1]} \left| e^{-\gamma x^2} - h_\ell(x) \right|,$$

where p_ℓ^j, q_ℓ^j, r_ℓ^j, and h_ℓ are Chebyshev approximations of degree $\leq \ell$ to $e^{-\gamma(x-x_j)^2}$, $e^{-\gamma(y-y_j)^2}$, $e^{-\gamma(z-z_j)^2}$, and $e^{-\gamma x^2}$, respectively. An explicit Chebyshev expansion for $e^{-\gamma x^2}$ is known and given by [140, p. 32]

$$e^{-\gamma x^2} = {\sum_{j=0}^{\infty}}' (-1)^j e^{-\gamma/2} I_j(\gamma/2) T_{2j}(x),$$

where the prime on the summation indicates that the first term is halved, and $I_j(z)$ is the modified Bessel function of the first kind with parameter j [158, (10.25.2)]. This means that one can show that [77, Lem. 5]:[3]

$$h_\ell(x) = {\sum_{j=0}^{\ell}}' (-1)^j e^{-\gamma/2} I_j(\gamma/2) T_{2j}(x), \quad \sup_{x\in[-1,1]} \left| e^{-\gamma x^2} - h_\ell(x) \right| \leq 2e^{-\gamma/4} I_{\lfloor \ell/2 \rfloor + 1}(\gamma/4).$$

By Lemma 2.1.1 and (2.3), we can understand the compressibility of $\mathcal{X}$. In particular, we can find an approximant tensor whose tensor-train ranks are bounded by the smallest integer ℓ such that $6Mn^{3/2} e^{-\gamma/4} I_{\lfloor \ell/2 \rfloor + 1}(\gamma/4) \leq \epsilon$. We find it straightforward to visualize compressibility via elements of the tensor ranks and their bounds, due to the way storage costs are calculated. Fig. 2.1 (right) shows the second element of the tensor-train rank, s_1 of the approximant

[3] Unfortunately, there is a typo in [77, Lem. 5] and $I_{\ell+1}(\gamma/4)$ should be replaced by $I_{\lfloor \ell/2 \rfloor + 1}(\gamma/4)$.

tensor, along with the bound that we derived. The bounds are relatively tight when ϵ is small.

2.3 Tensors with displacement structure

We say that $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ has an $(A^{(1)}, \ldots, A^{(d)})$-displacement structure of $\mathcal{F} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ if $\mathcal{X}$ satisfies the multidimensional Sylvester equation (1.13). In this section, we show that when $A^{(1)}, \ldots, A^{(d)}$ are normal matrices with "separated" spectra and $\mathcal{F}$ is a low rank tensor, then $\mathcal{X}$ is compressible. Several classes of structured tensors (e.g., the Hilbert tensor) and the solution tensors of certain discretized partial differential equations (e.g., the discretized solution to Poisson's equation) have a displacement structure, which leads to an understanding of their compressibility.

2.3.1 Zolotarev numbers

The bounds that we derive on compressibility of tensors involve so-called Zolotarev numbers [5, 82, 234]. A Zolotarev number is a positive number between 0 and 1 defined via an infimum problem involving rational functions [234]. Namely,

$$Z_k(\Phi, \Psi) := \inf_{r \in \mathcal{R}_{k,k}} \frac{\sup_{z \in \Phi} |r(z)|}{\inf_{z \in \Psi} |r(z)|}, \qquad k \geq 0, \tag{2.5}$$

where Φ and Ψ are disjoint complex sets and $\mathcal{R}_{k,k}$ is the set of irreducible rational functions of the form $p(x)/q(x)$ with polynomials p and q of degree at most k. If Φ and Ψ are well-separated, then one finds that $Z_k(\Phi, \Psi)$ decays rapidly with

k. This is because one can construct a low degree rational function that is small on Φ and large on Ψ. If Φ and Ψ are close to each other, then typically $Z_k(\Phi, \Psi)$ decreases much more slowly with k.

Zolotarev numbers can be used to bound the singular values of matrices with displacement structure [24, Theorem 2.1]. In particular, if $X \in \mathbb{C}^{m\times n}$ with $m \geq n$ satisfies the displacement structure (1.11) with displacement rank ν, where $A \in \mathbb{C}^{m\times m}$ and $B \in \mathbb{C}^{n\times n}$ are normal matrices with spectra $\Lambda(A) \subseteq \Phi$ and $\Lambda(B) \subseteq \Psi$, i.e.,

$$AX - XB = MN^*, \qquad M \in \mathbb{C}^{m\times \nu}, \quad N \in \mathbb{C}^{n\times \nu}, \tag{2.6}$$

then the singular values of X satisfy [24, Theorem 2.1]

$$\sigma_{j+\nu k}(X) \leq Z_k(\Phi, \Psi)\sigma_j(X), \quad 1 \leq j + \nu k \leq n. \tag{2.7}$$

Roughly speaking, if $\Lambda(A)$ and $\Lambda(B)$ are well-separated and ν is small, then the singular values $\sigma_j(X)$ decrease rapidly to 0.

When working with tensors, we translate the inequalities in (2.7) into Frobenius norm error bounds so that matrix results can then be utilized.

Lemma 2.3.1. *If $X \in \mathbb{C}^{m\times n}$ is a matrix satisfying* (2.6) *and $X_{\nu k}$ is the best rank νk approximation to X, then*

$$\|X - X_{\nu k}\|_F \leq Z_k(\Phi, \Psi)\|X\|_F,$$

where $\|\cdot\|_F$ denotes the matrix Frobenius norm.

Proof. To simplify notation let $Z_k = Z_k(\Phi, \Psi)$, $r = \nu k$, $\sigma_j = \sigma_j(X)$ for $1 \leq j \leq n$, and $\sigma_j = 0$ for $j > n$. If $k = 0$, then $r = 0$ and $Z_k = 1$, so $X_r = 0$ and the

statement follows automatically. Now consider $k > 0$, note that for any $s \geq 1$ we have

$$\sum_{j=sr+1}^{(s+1)r} \sigma_j^2 = \sum_{j=1}^{r} \sigma_{j+sr}^2 \leq Z_k^2 \sum_{j=1}^{r} \sigma_{j+(s-1)r}^2 \leq \cdots \leq Z_k^{2s} \sum_{j=1}^{r} \sigma_j^2,$$

where the inequalities come from the repeated application of the bound in (2.7). Therefore, we can bound $\|X - X_r\|_F^2$ by partitioning the singular values into groups of r. That is,

$$\|X - X_r\|_F^2 = \sum_{j=r+1}^{n} \sigma_i^2 \leq \sum_{s=1}^{\infty} \sum_{j=sr+1}^{(s+1)r} \sigma_i^2 \leq \sum_{s=1}^{\infty} Z_k^{2s} \sum_{j=1}^{r} \sigma_j^2 = \frac{Z_k^2}{1 - Z_k^2} \sum_{j=1}^{r} \sigma_j^2,$$

where the last equality is obtained by summing up the geometric series. Since $\|X\|_F^2 = \sum_{j=1}^{n} \sigma_j^2$, we find that

$$\left(1 + \frac{Z_k^2}{1 - Z_k^2}\right) \|X - X_r\|_F^2 \leq \frac{Z_k^2}{1 - Z_k^2} \|X\|_F^2.$$

The result follows by rearranging. □

For $0 < \epsilon < 1$, the numerical rank of X measured in the Frobenius norm is the smallest integer, r_ϵ, such that

$$\inf_{\text{rank}(\tilde{X}) \leq r_\epsilon} \|X - \tilde{X}\|_F \leq \epsilon \|X\|_F .$$

We denote this integer by $\text{rank}_\epsilon(X)$. From Lemma 2.3.1, we find that for matrices that satisfy (2.6) with low displacement rank, we have

$$\text{rank}_\epsilon(X) \leq \nu k, \tag{2.8}$$

where k is the smallest integer so that $Z_k(\Phi, \Psi) \leq \epsilon$. Therefore, Zolotarev numbers are very useful when trying to bound the numerical rank of matrices with displacement structure. For example, for an $n \times n$ Pick matrix P_n constructed with real numbers from an inverval $[a, b]$ with $0 < a < b < \infty$, one can find that $\text{rank}_\epsilon(P_n) \leq 2\left\lceil \log(4b/a) \log(4/\epsilon)/\pi^2 \right\rceil$ [24].

2.3.2 The compressibility of tensors with displacement structure in the tensor-train format

Zolotarev numbers can also be used to understand the compressibility of tensors satisfying (1.13). From the bounds in (1.6), one finds that the numerical ranks of each unfolding provides an upper bound on all entries of the tensor-train ranks of approximant tensors. More precisely, if $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ is a tensor and $0 < \epsilon < 1$, then there exists a tensor $\tilde{\mathcal{X}}$ such that [164, Theorem 2.2]

$$||\mathcal{X} - \tilde{\mathcal{X}}||_F \leq \epsilon ||\mathcal{X}||_F, \quad \operatorname{rank}^{\mathrm{TT}}(\tilde{\mathcal{X}}) = (1, \operatorname{rank}_\delta(X_1), \ldots, \operatorname{rank}_\delta(X_{d-1}), 1), \quad (2.9)$$

where $\delta = \epsilon/\sqrt{d-1}$ and X_k is the kth unfolding of $\mathcal{X}$. In order to easily relate tensor-train ranks with multilinear ranks in the next subsection, we choose to use $\delta = \epsilon/\sqrt{d}$.

If $\mathcal{X}$ satisfies (1.13), then by rearranging (1.13) one can show that each unfolding matrix, X_j, has a displacement structure. This is precisely $B_j X_j - X_j C_j^T = F_j$, where F_j is the jth unfolding of $\mathcal{F}$ and

$$B_j = I \otimes \cdots \otimes I \otimes A^{(1)} + \cdots + A^{(j)} \otimes I \otimes \cdots \otimes I,$$

$$C_j = -(I \otimes \cdots \otimes I \otimes A^{(j+1)} + \cdots + A^{(d)} \otimes I \otimes \cdots \otimes I).$$

From properties of the Kronecker product [185, Thm 2.5], we know that B_j and C_j are normal matrices with $\Lambda(B_j) = \Lambda(A^{(1)}) + \cdots + \Lambda(A^{(j)}) \subseteq \Phi_j$ and $\Lambda(C_j) = -(\Lambda(A^{(j+1)}) + \cdots + \Lambda(A^{(d)})) \subseteq \Psi_j$.[4] From (2.3.1) we see that for any integer k_j such that $Z_{k_j}(\Phi_j, \Psi_j) \leq \delta$, then

$$\operatorname{rank}_\delta(X_j) \leq k_j \nu_j, \qquad \nu_j = \operatorname{rank}(F_j), \qquad 1 \leq j \leq d-1.$$

[4] By $\Lambda(A) + \Lambda(B)$ we mean the Minkowski sum, formed by adding each element in $\Lambda(A)$ to each element in $\Lambda(B)$, i.e.,

$$\Lambda(A) + \Lambda(B) = \{a + b \mid a \in \Lambda(A),\ b \in \Lambda(B)\}.$$

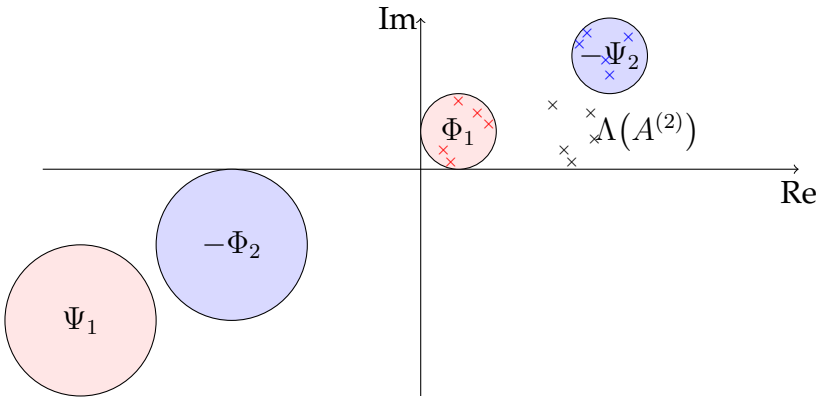

Figure 2.2: Minkowski sum separated matrices $A^{(1)}, A^{(2)}$, and $A^{(3)}$ where the colored crosses denote the spectrum of $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$, respectively. Here, $\Lambda(A^{(1)}) \subseteq \Phi_1$, $\Lambda(A^{(3)}) \subseteq -\Psi_2$, $\Lambda(A^{(1)}) + \Lambda(A^{(2)}) \subseteq \Phi_2$, and $\Lambda(A^{(2)}) + \Lambda(A^{(3)}) \subseteq -\Psi_1$. By definition, we must have that Φ_1 is disjoint from Ψ_1 (red regions), and that Φ_2 is disjoint from Ψ_2 (blue regions).

Therefore, a necessary condition to bound the numerical tensor-train ranks of $\mathcal{X}$ using this approach is that the spectra of $A^{(1)}, \ldots, A^{(d)}$ are *Minkowski sum separated*.

Definition 2.3.1. *We say that normal matrices $A^{(1)}, \ldots, A^{(d)}$ are Minkowski sum separated if there are disjoint sets Φ_j and Ψ_j so that*

$$\Lambda(A^{(1)}) + \cdots + \Lambda(A^{(j)}) \subseteq \Phi_j, \quad -(\Lambda(A^{(j+1)}) + \cdots + \Lambda(A^{(d)})) \subseteq \Psi_j, \quad 1 \leq j \leq d-1,$$

where the set additions are Minkowski sums and $\Lambda(A^{(j)})$ denotes the spectrum of $A^{(j)}$.

Figure 2.2 illustrates three Minkowski sum separated matrices $A^{(1)}, A^{(2)}$, and $A^{(3)}$ along with possible choices for the sets Φ_j and Ψ_j for $j = 1, 2$. We summarize our findings as a theorem.

Theorem 2.3.2. *Suppose $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ satisfies* (1.13)*, where $A^{(1)}, \ldots, A^{(d)}$ are Minkowski sum separated with disjoint sets Φ_j and Ψ_j for $1 \leq j \leq d-1$. Then, for a fixed $0 < \epsilon < 1$, we have*

$$p_\epsilon^{\mathrm{TT}}(\mathcal{X}) \leq \sum_{j=1}^{d} (k_{d-1}\nu_{d-1})(k_d\nu_d)n_d, \quad \nu_j = \operatorname{rank}(F_j), \quad 1 \leq j \leq d-1,$$

where F_j is the jth unfolding of $\mathcal{F}$ and k_j is an integer so that $Z_{k_j}(\Phi_j, \Psi_j) \leq \epsilon/\sqrt{d}$.

For special choices of Φ_j and Ψ_j, explicit bounds on $Z_{k_j}(\Phi_j, \Psi_j)$ are known and therefore the bounds in Theorem 2.3.2 are also explicit. Here we mention two special cases:

Intervals

If $\Lambda(A^{(j)}) \subseteq [a, b]$ for $0 < a < b < \infty$, then one can take $\Phi_j = [ja, jb]$ and $\Psi_j = [-(d-j)b, -(d-j)a]$ in Theorem 2.3.2. From [24, Cor. 4.2], we find that

$$Z_{k_j}(\Phi_j, \Psi_j) \leq 4\left[\exp\left(\frac{\pi^2}{2\log(16\gamma_j)}\right)\right]^{-2k}, \quad \gamma_j = \frac{(da + j(b-a))(db - j(b-a))}{abd^2}.$$

In particular, the following bound holds:

$$p_\epsilon^{\mathrm{TT}}(\mathcal{X}) \leq \sum_{j=1}^{d} (k_{d-1}\nu_{d-1})(k_d\nu_d)n_d, \qquad k_j = \left\lceil \frac{\log(16\gamma_j)\log(4\sqrt{d}/\epsilon)}{\pi^2} \right\rceil, \tag{2.10}$$

where $\nu_j = \mathrm{rank}(F_j)$.

Disks

If $\Lambda(A^{(j)}) \subseteq \{z \in \mathbb{C} \,:\, |z - z_0| \leq \eta\}$ for $0 < \eta < z_0$ and $z_0, \eta \in \mathbb{R}$, then one finds that $\Lambda(A^{(1)}) + \cdots + \Lambda(A^{(j)}) \subseteq \{z \in \mathbb{C} \,:\, |z - jz_0| \leq j\eta\}$ and $-(\Lambda(A^{(j+1)}) + \cdots + \Lambda(A^{(d)})) \subseteq \{z \in \mathbb{C} \,:\, |z + (d-j)z_0| \leq (d-j)\eta\}$. From [200, p. 123], we find that

$$Z_{k_j}(\Phi_j, \Psi_j) = \rho_j^{-k_j}, \qquad \rho_j = \frac{2j(d-j)\eta^2}{d^2 z_0^2 - ((d-j)^2 + j^2)\eta^2 - \sqrt{\xi_j}},$$

where $\xi_j = (d^2 z_0^2 - ((d-j)^2 + j^2)\eta^2)^2 - 4j^2(d-j)^2\eta^4$. In particular,

$$p_\epsilon^{\mathrm{TT}}(\mathcal{X}) \leq \sum_{j=1}^{d} (k_{d-1}\nu_{d-1})(k_d\nu_d)n_d, \qquad k_j = \left\lceil \log(\sqrt{d}/\epsilon)/\log(\rho_j) \right\rceil, \tag{2.11}$$

where $\nu_j = \operatorname{rank}(F_j)$.

In Sec. 2.4, we use (2.10) to bound the numerical storage cost in tensor-train format of the Hilbert tensor and the solution tensor of a discretized Poisson equation.

2.3.3 The compressibility of tensors with displacement structure in the Tucker format

From HOSVD, we know that for a tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ and accuracy level $0 < \epsilon < 1$, then there exists a tensor $\tilde{\mathcal{X}}$ such that [53]:

$$||\mathcal{X} - \tilde{\mathcal{X}}||_F \leq \epsilon ||\mathcal{X}||_F, \quad \operatorname{rank}^{\mathrm{ML}}(\tilde{\mathcal{X}}) = (\operatorname{rank}_\delta(X_{(1)}), \ldots, \operatorname{rank}_\delta(X_{(d)})),$$

where $\delta = \epsilon/\sqrt{d}$ and $X_{(j)}$ is the jth matricization of $\mathcal{X}$.

Since the first unfolding of $\mathcal{X}$ coincides with the first matricization of $\mathcal{X}$, the bound on the second element of the tensor-train rank is also a bound on the first element of the multilinear rank of $\mathcal{X}$. One can use a similar idea to bound all entries of the multilinear ranks by considering the various matricizations. However, one finds that the spectra of $A^{(1)}, \ldots, A^{(d)}$ need to be separated in a slightly different sense.

Definition 2.3.2. *We say that normal matrices $A_1, \ldots, A_d$ are Minkowski singly separated if there are disjoint sets Φ_j and Ψ_j so that*

$$\Lambda(A_j) \subseteq \Phi_j, \quad -\left(\sum_{k=1, k \neq j}^{d} \Lambda(A_k) \right) \subseteq \Psi_j, \quad 1 \leq j \leq d,$$

where the set additions are Minkowski sums and $\Lambda(A_j)$ denotes the spectrum of A_j.

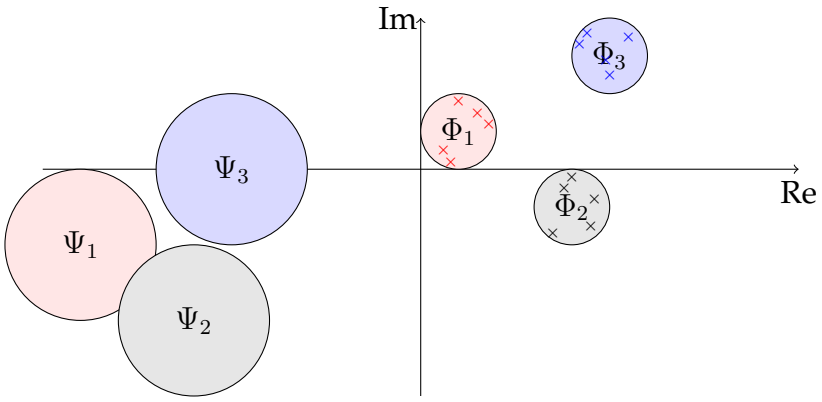

Figure 2.3: Minkowski singly separated matrices $A^{(1)}, A^{(2)}$, and $A^{(3)}$ where the colored crosses denote the spectrum of $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$, respectively. Here, $\Lambda(A^{(1)}) \subseteq \Phi_1$, $\Lambda(A^{(2)}) \subseteq \Phi_2$, $\Lambda(A^{(3)}) \subseteq \Phi_3$, $-(\Lambda(A^{(1)})+\Lambda(A^{(2)})) \subseteq \Psi_3$, $-(\Lambda(A^{(1)})+\Lambda(A^{(3)})) \subseteq \Psi_2$, and $-(\Lambda(A^{(2)}) + \Lambda(A^{(3)})) \subseteq \Psi_1$. By definition, we have that Φ_1 is disjoint from Ψ_1 (red regions), that Φ_2 is disjoint from Ψ_2 (gray regions), and that Φ_3 is disjoint from Ψ_3 (blue regions).

Figure 2.3 illustrates the spectra of Minkowski singly separated matrices $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ along with their enclosed sets and Minkowski sums of the sets. Under this separation condition, we have the following theorem:

Theorem 2.3.3. *Suppose $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ satisfies* (1.13)*, where $A^{(1)}, \ldots, A^{(d)}$ are Minkowski singly separated with disjoint sets Φ_j and Ψ_j for $1 \leq j \leq d$. Then, for a fixed $0 < \epsilon < 1$, we have*

$$p_\epsilon^{\mathrm{ML}}(\mathcal{X}) \leq \sum_{j=1}^{d} n_j k_j \mu_j + \prod_{j=1}^{d} k_j \mu_j, \qquad \mathrm{rank}^{\mathrm{ML}}(\mathcal{F}) = (\mu_1, \ldots, \mu_d),$$

where k_j is an integer so that $Z_{k_j}(\Phi_j, \Psi_j) \leq \epsilon/\sqrt{d}$.

Proof. One can bound all the entries of the multilinear rank vector of $\mathcal{X}$ by the second entry of the tensor-train rank vector of the tensors $\mathcal{Y}^1, \ldots, \mathcal{Y}^d$ (see (1.3)). Due to the way $\mathcal{Y}^j$ is constructed, it can be shown that $\mathcal{Y}^j$ satisfies

$$\mathcal{Y}^j \times_1 A^{(j)} + \cdots + \mathcal{Y}^j \times_{d-j+1} A^{(d)} + \mathcal{Y}^j \times_{d-j+2} A^{(1)} + \cdots + \mathcal{Y}^j \times_d A^{(j-1)} = \mathcal{H}^j,$$

where $H^j_{(1)} = F_{(j)}, \ldots, H^j_{(d-j+1)} = F_{(d)}, H^j_{(d-j+2)} = F_{(1)}, \ldots, H^j_{(d)} = F_{(j-1)}$ and $\mathcal{H}^j$ is constructed from $\mathcal{F}$ in the same way that $\mathcal{Y}^j$ is constructed from $\mathcal{X}$. The result follows from Theorem 2.3.2 as the jth element of the multilinear rank of $\mathcal{X}$ is bounded above by the bound of the second entry of the tensor-train rank of $\mathcal{Y}^j$. □

As before, explicit bounds on the compressibility in Tucker format can be obtained from Theorem 2.3.3 by special choices of Φ_j and Ψ_j such as when they are intervals or disks.

Intervals

If $\Lambda(A^{(j)}) \subseteq [a, b]$ for $0 < a < b < \infty$, then one can take $\Phi_j = [a, b]$ and $\Psi_j = [-(d-1)b, -(d-1)a]$. Therefore, we find that [24, Cor. 4.2]

$$p_\epsilon^{\mathrm{ML}}(\mathcal{X}) \leq k \sum_{j=1}^{d} n_j \mu_j + k^d \prod_{j=1}^{d} \mu_j, \qquad k = \left\lceil \frac{\log(16\gamma)\log(4\sqrt{d}/\epsilon)}{\pi^2} \right\rceil,$$

where $\gamma = (da + (b-a))(db - (b-a))/(abd^2)$ and $\mathrm{rank}^{\mathrm{ML}}(\mathcal{F}) = (\mu_1, \ldots, \mu_d)$.

Disks

If $\Lambda(A^{(j)}) \subseteq \{z \in \mathbb{C} \,:\, |z - z_0| \leq \eta\}$ for $0 < \eta < z_0$ and $z_0, \eta \in \mathbb{R}$, then one can take $\Phi_j = \{z \in \mathbb{C} \,:\, |z - z_0| \leq \eta\}$ and $\Psi_j = \{z \in \mathbb{C} \,:\, |z + (d-1)z_0| \leq (d-1)\eta\}$. From [200, p. 123], we find that

$$p_\epsilon^{\mathrm{ML}}(\mathcal{X}) \leq k \sum_{j=1}^{d} n_j \mu_j + k^d \prod_{j=1}^{d} \mu_j, \qquad k = \left\lceil \log(\sqrt{d}/\epsilon)/\log(\rho) \right\rceil,$$

where $\rho = (2(d-1)\eta^2)/(d^2 z_0^2 - ((d-1)^2+1)\eta^2 - \sqrt{\xi})$, $\xi = (d^2 z_0^2 - ((d-1)^2 + 1)\eta^2)^2 - 4(d-1)^2\eta^4$, and $\mathrm{rank}^{\mathrm{ML}}(\mathcal{F}) = (\mu_1, \ldots, \mu_d)$.

2.4 Worked examples of tensors with displacement structure

Here, we give two examples that illustrate how to use the displacement structure of a tensor to understand its compressibility. Since the bounds in tensor-train format and Tucker format are related through ranks, we only show results for the tensor-train format. As in the previous examples, we use the second element of the tensor-train rank and its bound to visualize the compressibility. We consider two tensors: (1) The 3D Hilbert tensor and (2) The solution tensor of a Poisson equation.

2.4.1 The 3D Hilbert tensor

Consider the Hilbert tensor $\mathcal{H} \in \mathbb{C}^{n\times n\times n}$ defined by

$$\mathcal{H}_{ijk} = \frac{1}{i+j+k-2}, \qquad 1 \le i,j,k \le n.$$

This tensor is analogous to the notoriously ill-conditioned Hilbert matrix [65, 106]. It is easy to verify that the tensor possesses the following displacement structure:

$$\mathcal{H} \times_1 D + \mathcal{H} \times_2 D + \mathcal{H} \times_3 D = \mathcal{S},$$

where $\mathcal{S}$ is the tensor of all ones and D is a diagonal matrix with $D_{ii} = i - \frac{2}{3}$. Thus, $\operatorname{rank}(\mathcal{S}) = 1$ and the ranks of the unfoldings of $\mathcal{S}$ are all 1.

Since the spectrum of D is contained in $[\frac{1}{3}, \frac{3n-2}{3}]$, (2.10) tells us that for any $0 < \epsilon < 1$ we have

$$p_\epsilon^{\mathrm{TT}}(\mathcal{H}) \le n(s_1^2 + 2s_1), \qquad s_1 = \left\lceil \frac{1}{\pi^2} \log\left(\frac{16n(2n-1)}{3n-2}\right) \log\left(\frac{4\sqrt{3}}{\epsilon}\right) \right\rceil. \tag{2.12}$$

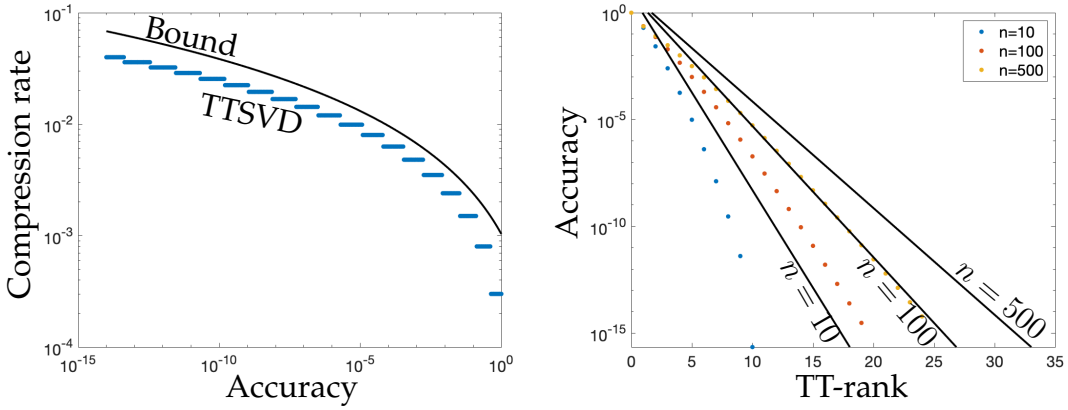

Figure 2.4: The compressibility of the 3D Hilbert tensor in tensor-train format. Left: The ratio between the storage cost for representing a $100 \times 100 \times 100$ Hilbert tensor in a tensor-train format calculated using TTSVD algorithm and 100^3 (blue dots), along with our theoretical bound on the compression rate (black line). Right: The second element of the tensor-train rank, s_1, (dots) and the theoretical bound (black lines) for $n = 10, 100,$ and 500.

That is, $s_1 = \mathcal{O}(\log n \log(1/\epsilon))$ and means that the $n \times n \times n$ Hilbert tensor can be stored, up to an accuracy of ϵ in the Frobenius norm, in just $\mathcal{O}(n(\log n)^2(\log(1/\epsilon))^2)$ degrees of freedom. Figure 2.4 (left) shows the compressibility of $\mathcal{H}$ with $n = 100$ by computing the ratio of the storage costs using tensor-train format and explicit storage. Our theoretical results bound the savings well. Figure 2.4 (right) shows the compressibility of $\mathcal{H}$ by plotting s_1 and its bound in (2.12) for different values of n. The actual tensor-train ranks of $\mathcal{H}$ are computed with TTSVD [164].

2.4.2 Tensor solution of a discretized Poisson equation

Tensor decompositions can be incorporated into efficient solvers of partial differential equations [15, 31, 113, 119, 165, 205]. Displacement structure arises for the solution tensor when one discretizes a Laplace operator, or any Laplace-like operator. Here, consider the 3D Poisson equation on $[-1, 1]^3$ with zero Dirichlet

conditions, i.e.,

$$-\left(u_{xx}+u_{yy}+u_{zz}\right)=f \text{ on } \Omega=[-1,1]^3, \qquad u|_{\partial\Omega}=0. \tag{2.13}$$

If one writes down a second-order finite difference discretization of (2.13) on an $n\times n\times n$ equispaced grid, then one obtains the following multidimensional Sylvester equation

$$\mathcal{X}\times_1 K+\mathcal{X}\times_2 K+\mathcal{X}\times_3 K=\mathcal{F}, \qquad K=-\frac{1}{h^2}\begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix},$$

where $h=2/n$ and $\mathcal{F}_{ijk}=f(ih-1,jh-1,kh-1)$ for $1\le i,j,k\le n-1$. The solution tensor $\mathcal{X}$ is unknown and for large n, one assumes that $\mathcal{X}_{ijk}\approx u(ih-1,jh-1,kh-1)$ for $1\le i,j,k\le n-1$ is a reasonably good approximation. The eigenvalues of K are given by $4/h^2\sin^2(\pi k/(2n))$ for $1\le k\le n$ with $h=2/n$ [134, (2.23)]. Since $(2/\pi)x\le\sin x\le 1$ for $x\in[0,\pi/2]$ and $h=2/n$, the eigenvalues of K are contained in the interval $[1,n^2]$.

We are interested in understanding the compressibility of $\mathcal{X}$ in tensor-train format when $f=1$. Since $\Lambda(K)\subseteq[1,n^2]$ and $\operatorname{rank}^{\mathrm{TT}}(\mathcal{F})=(1,1,1,1)$, (2.10) gives

$$p_\epsilon^{\mathrm{TT}}(\mathcal{X})\le n(s_1^2+2s_1), \quad s_1=\left\lceil\frac{1}{\pi^2}\log\left(\frac{16(n^2+2)(2n^2+1)}{9n^2}\right)\log\left(\frac{4\sqrt{3}}{\epsilon}\right)\right\rceil. \tag{2.14}$$

Figure 2.5 (left) shows the second element of the tensor-train rank, s_1, and the bound of the approximate solution tensor to the Poisson equation via finite difference discretization.

One wonders if there is also a fast Poisson solver for spectral discretizations. This turns out to be feasible with a carefully constructed ultraspherical spectral

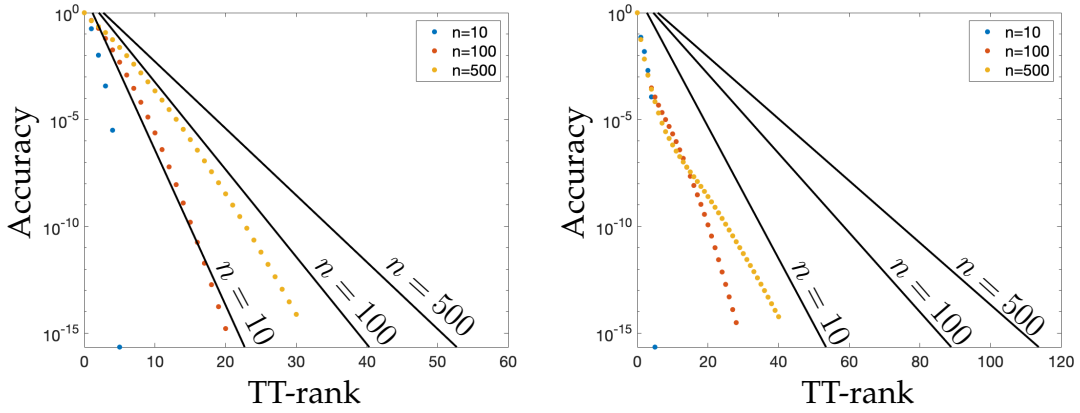

Figure 2.5: Left: The second element of the tensor-train rank, s_1, (blue, red, and yellow dots) of the finite difference solution to $-(u_{xx} + u_{yy} + u_{zz}) = 1$ on $[-1, 1]^3$ with zero Dirichlet conditions, and the theoretical bound in (2.14) (black lines). Right: The second element of the tensor-train rank, s_1, (blue, red, yellow dots) of the ultraspherical spectral solution to $-(u_{xx} + u_{yy} + u_{zz}) = 1$ on $[-1, 1]^3$ with zero Dirichlet conditions, and the theoretical bound in (2.16) (black lines).

discretization. The Poisson equation can be discretized to a tensor equation as [74]:

$$\mathcal{X} \times_1 A^{-1} + \mathcal{X} \times_2 A^{-1} + \mathcal{X} \times_3 A^{-1} = \mathcal{G}, \tag{2.15}$$

where

$$u(x, y, z) = (1 - x^2)(1 - y^2)(1 - z^2) \sum_{p=0}^{n} \sum_{q=0}^{n} \sum_{r=0}^{n} \mathcal{X}_{pqr} \tilde{C}_p^{(3/2)}(x) \tilde{C}_q^{(3/2)}(y) \tilde{C}_r^{(3/2)}(z),$$

$$f(x, y, z) = \sum_{p=0}^{n} \sum_{q=0}^{n} \sum_{r=0}^{n} \mathcal{F}_{pqr} \tilde{C}_p^{(3/2)}(x) \tilde{C}_q^{(3/2)}(y) \tilde{C}_r^{(3/2)}(z),$$

$\tilde{C}_k^{(3/2)}$ is the degree k orthonormalized ultraspherical polynomial with parameter $\frac{3}{2}$ [158, Table 18.3.1], $\mathcal{G} = \mathcal{F} \times_1 M^{-1} \times_2 M^{-1} \times_3 M^{-1}$, $A = D^{-1}M$, D is a diagonal matrix, M and A are both symmetric pentadiagonal matrices, and the spectrum of A satisfies $\Lambda(A) \in [-1, \ -1/(30n^4)]$. If $f = 1$, (2.10) gives

$$p_\epsilon^{\mathrm{TT}}(\mathcal{X}) \leq n(s_1^2 + 2s_1), \ s_1 = \left\lceil \frac{1}{\pi^2} \log\left(\frac{16(30n^4 + 2)(60n^4 + 1)}{270n^4} \right) \log\left(\frac{4\sqrt{3}}{\epsilon} \right) \right\rceil. \tag{2.16}$$

Figure 2.5 (right) shows the second element of the tensor-train rank, s_1, and the bound of the approximate solution tensor to the Poisson equation via ultraspherical spectral discretization. This spectral discretization indicates that the $n \times n \times n$ tensor discretization of the solution can be approximated with only $\mathcal{O}(dn(\log n)^2(\log(1/\epsilon))^2)$ degrees of freedom. This is a significant reduction in the cost of storing the solution, with a relatively straightforward decomposition. Comparatively, one can achieve $\mathcal{O}(d \log n \log(1/\epsilon))$ with quantics tensor formats [110, 118], but their structures are more complicated and are not as simple to use as the tensor-train format.

Some special functions can be well-approximated by exponential sums of the form

$$S_k(x) = \sum_{j=1}^{k} \alpha_j e^{-t_j x}, \qquad \alpha_j, t_j \in \mathbb{R},$$

and these approximant can be used to represent the solution to PDEs with Laplace-like operators [83, 116]. In [116], the author uses exponential sums to show that the solution tensor to several 3D elliptic PDEs can be approximated with $\mathcal{O}(dn(\log n)^2(\log(1/\epsilon))^2)$ degrees of freedom. In this scenario, the Laplacian inverse operator can be approximated with a low CP rank tensor. In general, both exponential sum approximation and Zolotarev numbers can be used to bound the kth singular value of matrices with displacement structure and capture the geometric decay, but the Zolotarev bound tends to be cleaner and does not involve an algebraic factor related to k [211, (34),(35)].

2.4.3 Solving for tensors in compressed formats

Since the proof of Theorem 2.3.2 and Theorem 2.3.3 are constructive, we can use their implicit algorithms to solve 3D Sylvester tensor equations of the form:

$$\mathcal{X} \times_1 A^{(1)} + \mathcal{X} \times_2 A^{(2)} + \mathcal{X} \times_3 A^{(3)} = \mathcal{F}, \tag{2.17}$$

where $A^{(1)} \in \mathbb{C}^{n_1 \times n_1}$, $A^{(2)} \in \mathbb{C}^{n_2 \times n_2}$, $A^{(3)} \in \mathbb{C}^{n_3 \times n_3}$, and $\mathcal{F} \in \mathbb{C}^{n_1 \times n_2 \times n_3}$. We can compute approximate solutions to (2.17) efficiently in tensor-train or Tucker format when $\mathcal{F}$ is a low rank tensor and the spectra of $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ are well-separated. In particular, if $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ are Minkowski sum separated, and the unfoldings F_1 and F_2 of $\mathcal{F}$ have low rank decompositions $F_1 = W_1 Z_1^*$, and $F_2 = W_2 Z_2^*$ with rank r_1 and r_2, respectively, then we can solve for $\mathcal{X}$ in tensor-train format.

The tensor-train factors of $\mathcal{X}$ obtained by the TTSVD algorithm are orthogonal matrices for the column and row spaces of unfoldings of X. For example, the first tensor-train factor U_1 of $\mathcal{X}$ can be found as a matrix with orthonormal columns spanning the column space of the first unfolding X_1. Since $\mathcal{X}$ satisfies (2.17), we find that X_1 satisfies the Sylvester equation

$$A^{(1)} X_1 + X_1 (I \otimes A^{(2)} + A^{(3)} \otimes I)^T = W_1 Z_1^*. \tag{2.18}$$

We can use the factored alternating direction implicit (fADI) method to solve (2.18) for a matrix V_1 such that $X_1 = V_1 D_1 Y_1^*$ [27]. One can then use the QR decomposition of V_1, i.e., $V_1 = U_1 R_1$, to calculate the first tensor-train core U_1.

Second and third tensor-train factors can be computed by finding matrices with orthonormal columns for the column and row spaces associated to C_2,

where $C_2 = \text{reshape}(R_1 D_1 Y_1^*, r_1 n_2, n_3)$. It can be shown that C_2 satisfies the Sylvester equation

$$(I \otimes (U_1^* A^{(1)} U_1) + A^{(2)} \otimes I) C_2 + C_2 (A^{(3)})^T = (I \otimes U_1^*) W_2 Z_2^*.$$

One can, again, use fADI to solve for a low rank decomposition of C_2, i.e., $C_2 = V_2 D_2 Y_2^*$. This low rank decomposition can be compressed by performing a QR factorization of V_2 and Y_2 and then doing a SVD to obtain $C_2 \approx U_2 \Sigma T_2^*$, where U_2 and T_2 are matrices with r_2 orthonormal columns and Σ is a diagonal matrix. In this way, the second tensor-train factor is $U_2 = \text{reshape}(U_2, [r_1, n_2, r_2])$ and the third factor $U_3 = \Sigma T_2^*$. Although the fADI method requires the solution of shifted linear systems with $I \otimes (U_1^* A^{(1)} U_1) + A^{(2)} \otimes I$, the Kronecker product structure allows one to reshape these linear systems into Sylvester equations, which can themselves be solved with the alternating direction implicit (ADI) method [27]. Specifically, to solve the linear system:

$$(I \otimes (U_1^* A^{(1)} U_1) + A^{(2)} \otimes I - \alpha I) y = b,$$

where α is a constant denoting the shift, we can rewrite it into a matrix equation:

$$(U_1^* A^{(1)} U_1 - \frac{\alpha}{2} I) Y + Y (A^{(2)} - \frac{\alpha}{2} I) = B, \tag{2.19}$$

where $Y = \text{reshape}(y, [r_1, n_2])$, and $B = \text{reshape}(b, [r_1, n_2])$. This Sylvester matrix equation is solvable as U_1 is orthogonal, and $A^{(1)}$ and $A^{(2)}$ have distinct spectrum. We can use the ADI method to solve the matrix equation efficiently. This means that one can completely avoid solving a huge linear system. As a result, if $n_1 = n_2 = n_3 = n$, $0 < \epsilon < 1$ is desired accuracy, and solving shifted linear systems of $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ takes T complexity, then solving (2.19) takes $\mathcal{O}(T \log n \log(1/\epsilon))$, and thus the Sylvester matrix equation solver has a complexity of $\mathcal{O}((s_1 + s_2) T \log n \log(1/\epsilon) + s_2 T (\log n)^2 (\log(1/\epsilon))^2)$. When s_1, s_2, and T are

small, we have an efficient solver. In summary, the ADI-based tensor Sylvester equation solver is described in Algorithm 7.

Algorithm 7 A 3D Sylvester equation (2.17) solver that solves the solution in TT form.

1: Use fADI to solve for the column space Z_1 of X_1 that satisfies $A^{(1)}X_1+X_1(I\otimes A^{(2)}+A^{(3)}\otimes I)^T=F_1=M_1N_1^*$.
2: Perform a QR decomposition, $Z_1=U_1R_1$, and let $U_1=U_1(:,1:s_1)$ if $R_1(s_1+1,s_1+1)$ is small enough.
3: Use fADI to solve for $C_2=Z_2D_2Y_2^*$ where C_2 satisfies $(I\otimes(U_1^*A^{(1)}U_1)+A^{(2)}\otimes I)C_2+C_2(A^{(3)})^T=(I\otimes U_1^*)F_2=(I\otimes U_1^*)M_2N_2^*$.
4: Find a low rank decomposition of $C_2\approx U_2\Sigma T_2^*$ using Z_2, D_2 and Y_2, and denote the rank by s_2.
5: Let $U_2=\mathrm{reshape}(U_2,[s_1,n_2,s_2])$.
6: Let $U_3=\Sigma T_2^*$.
7: The solution $\mathcal{X}$ is in the tensor-train form with cores U_1, U_2, and U_3.

Similarly, if all matricizations of $\mathcal{F}$ are low rank, and $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ are Minkowski singly separated, then we can solve for the solution in orthogonal Tucker format via HOSVD [53]. Each factor matrix of $\mathcal{X}$ is a matrix with orthonormal columns that span the column space of the matricization of $\mathcal{X}$, which satisfies the Sylvester equation:

$$A^{(j)}X_{(j)}+X_{(j)}(I\otimes A^{(i)}+A^{(k)}\otimes I)^T=F_{(j)},$$

where

$$i=\begin{cases}1, & j=3,\\ j+1, & j=1,2,\end{cases}\qquad k=\begin{cases}3, & j=1,\\ j-1, & j=2,3.\end{cases}$$

If solving shifted linear systems with $A^{(1)}$, $A^{(2)}$, and $A^{(3)}$ is fast, then we can use fADI to solve for the orthogonal column space of $X_{(j)}$, and use a direct method, such as a 3D Bartels–Stewart algorithm to solve for the core tensor [20].

2.4.4 Poisson equation solver

Consider the example of Poisson equation in Section 2.4.2 with ultraspherical discretization (2.15). Since A is a penta-diagonal matrix, we can solve shifted linear systems with A^{-1} in $\mathcal{O}(n)$ time using Thomas algorithm. In addition, (2.16) indicates that s_1 and s_2 are of order $\mathcal{O}(\log n \log(1/\epsilon))$. Therefore, we can obtain a fast Poisson equation solver that computes the solution in tensor-train or orthogonal Tucker format. In tensor-train format, the complexity is $\mathcal{O}(n(\log n)^3(\log(1/\epsilon))^3)$, where $0 < \epsilon < 1$ is the accuracy.

Figure 2.6 shows the running time of different discretized Poisson solvers. The dashed line represents the direct solver that converts (2.16) into a huge linear system via Kronecker product. The dash-dot line represents the eigendecomposition-based solver described in Algorithm 3. The solid line represents our fADI-based tensor-train solver. We can see as n gets large, our algorithm is the winner.[5]

Similarly, we can bound the solution of (2.15) in Tucker format with $\mathcal{O}((\log n)^3(\log(1/\epsilon))^3 + n(\log n)^2(\log(1/\epsilon))^2)$ degrees of freedom and obtain the solution efficiently using fADI and ADI with the Tucker solver described in the previous section. If one is interested in the solution in CP format, then the exponential sum method in [116] (also, see Section 2.4.2), can solve (2.15) via low rank truncation.

[5]The fADI solver is implemented in C++, while the direct and the eigensolvers are implemented in MATLAB. However, both backslash linear system solver and eigendecomposition are carried out in LAPACK, so our comparison of the three solvers is still fair. All timings are performed in MATLAB R2019a on the super computer of Cornell's Math department, with 20 physical GPUs, 250GB of RAM, and 1TB of hard-drive memory.

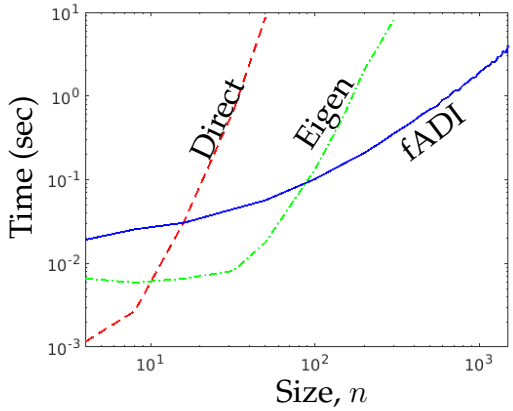

Figure 2.6: The execution time of a direct solver (dashed line), eigensolver (dash-dot line), and our fADI solver (solid line) of the spectrally discretized Poisson equation $-(u_{xx} + u_{yy} + u_{zz}) = 1$ on $[-1, 1]^3$ with zero Dirichlet conditions with discretization size n with $4 \leq n \leq 1500$.

CHAPTER 3

PARALLEL ALGORITHMS FOR COMPUTING THE TENSOR-TRAIN DECOMPOSITION

In this chapter[1], we connect dimensionality reduction techniques with high-performance computing by developing algorithms to compute the TT format of a tensor in parallel. To date, researchers have designed parallel tensor algorithms to exploit modern computing architectures and handle larger tensors emerging in applications. There are parallel algorithms for computing CP [135, 197], Tucker, and hierarchical Tucker decomposition [11, 16, 86, 109]. Subsequent operations can also be done in parallel in various tensor formats, especially tensor contractions [198], and operations in TT format [50]. However, despite some current work based on hierarchical tree structure [89], regularized least squares problem satisfied by each core [43], and multiple SVDs on tensor slices [221], parallel TT decomposition has received less attention, perhaps, due to the sequential nature of TTSVD (see Algorithm 1).

In this chapter, we show that the column spaces of tensor unfoldings (see section 1.1) are connected by the TT cores (see section 1.2.1). This result can also be derived from [50, Eqn.(2.3)] with a few extra steps. Using our more explicit formulation, we develop new parallel algorithms that are scalable, stable, and accurate to compute TT formats of tensors. In particular, we distribute tensor information across several processors and ask each of them to contribute to computing the TT cores. We design parallel algorithms for various tensor input types:

[1]This chapter is based on a paper with Maximilian Ruth and Alex Townsend [192]. I derived the theorems and designed the algorithms, and was the lead author of the manuscript.

- **Parallel-TTSVD**: Previous TT decomposition methods such as TTSVD [164] and TT-cross approximation [162] are sequential algorithms that require the entire tensor as input. In each iteration, both algorithms find one TT core by decomposing a specific matrix and use this core to determine the matrix in the next iteration. Based on the fact that there is a connection between the column space of various reshapes of a tensor (see section 1.1), we design an algorithm to compute the TT cores simultaneously by computing an orthonormal basis for the column space of each tensor unfolding via SVD.
- **Parallel Streaming TT Sketching (PSTT)**: Since SVD in Parallel-TTSVD can be computationally expensive, we can use randomized linear algebra to find orthonormal bases that approximate the column space of tensor unfoldings. This algorithm is inspired by matrix sketching [96], Tucker sketching [206], randomized algorithms for CP and Tucker format [141], TT sketching in a sequential manner [41], and TT rounding with randomized algorithms [51]. Sketching algorithms are ideal for streaming data, where it is infeasible to store the tensor in cache. We show a two-sided version, PSTT2, has a storage cost as low as $\mathcal{O}(n^{\lfloor d/2 \rfloor})$. Moreover, PSTT2-onepass, a one-pass variant of PSTT2, uses only a single evaluation of each tensor entry, and is the most efficient in numerical experiments.
- **TT2Tucker** and **Tucker2TT**: An orthonormal basis of the column space of the second unfolding of each TT core allows us to get a Tucker format of the given tensor fast. This method can be treated as a simplified version of the extended TT format in [58, Sec. 4.1] and also used in [23]. However, we derive it from an explicit relation between tensor matricizations and TT cores of the same tensor. Conversely, given a tensor in Tucker format, we can obtain its TT cores through the Tucker factor matrices and the TT cores of its Tucker core.

- **TT-fADI**: Tensors also arise as the solutions of Sylvester tensor equations (1.13). If $\mathcal{F}$ is provided in its TT format, and $A^{(1)}, \ldots, A^{(d)}$ have fast shifted linear system solvers and desired spectral properties, then we can find $\mathcal{X}$ in TT format via the factored alternating direction implicit (fADI) method that solves Sylvester matrix equations [27].
- **Implementations in message passing interface (MPI)**: We implement our algorithms in a distributed memory framework using OpenMPI in C. Each process is responsible for streaming part of the tensor and storing part of the intermediate calculations. We use well-established linear algebra packages to optimize our codes, including matrix multiplications in BLAS3, and QR and SVD in LAPACKE.

The chapter is organized as follows. In section 3.1, we consider computing the TT decomposition of a given tensor in parallel, where we have access to any entry. Then, we provide scalability and complexity analysis of our algorithms and demonstrate their performance on synthetic datasets in section 3.2. Finally, in section 3.3, we obtain the TT format of implicitly known tensors, given as solutions of Sylvester tensor equations.

3.1 Parallel TT approximations from other tensor formats

In this section, we focus on describing parallel algorithms to compute a TT approximation of a tensor $\mathcal{X}$ when we have access to all its entries. We consider three scenarios: (1) we can afford to store the whole tensor in cache (see section 3.1.1), (2) we can only afford to store a proportion of its entries in cache (see section 3.1.2), and (3) the Tucker format of $\mathcal{X}$ is known and can be stored in

cache (see section 3.1.3).

3.1.1 Parallel TT decomposition with SVD

The derivation of the parallel TT decomposition starts with the analogy between TTSVD and HOSVD. Roughly speaking, HOSVD follows a "compress-then-combine" approach, which compresses all matricizations first and then computes the core tensor. This makes HOSVD for computing the Tucker decomposition naturally parallelizable. Comparatively, for the TT format, TTSVD has a sequential nature that alternates between reshaping and compressing. Here, we design a "compress-then-combine" algorithm for computing a TT approximation. We compress all the unfoldings first and then combine the resulting matrices to obtain the TT cores.

We first show that the orthonormal bases of the column space of all tensor unfoldings are related for a d-dimensional tensor. An alternative way to derive this result is to use [50, Eqn (2.3) & Appendix A].

Theorem 3.1.1. *Let $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, and $X_j \in \mathbb{C}^{(\prod_{i=1}^{j} n_i) \times (\prod_{i=j+1}^{d} n_i)}$ be its jth flattening for $1 \leq j \leq d-1$. If $X_j = U_j V_j^*$ with $U_j \in \mathbb{C}^{(\prod_{i=1}^{j} n_i) \times r_j}$, $V_j \in \mathbb{C}^{(\prod_{i=j+1}^{d} n_i) \times r_j}$, $r_j \leq \min(\prod_{i=1}^{j} n_i, \prod_{i=j+1}^{d} n_i)$, and U_j has orthonormal columns, then for $1 \leq k \leq d-2$, there exist matrices $W_k \in \mathbb{C}^{r_k \times n_{k+1} r_{k+1}}$ such that*

$$\operatorname{reshape}\left(U_{k+1}, \prod_{i=1}^{k} n_i, n_{k+1} r_{k+1}\right) = U_k W_k.$$

Proof. To proceed with the proof, we use the fact that for $1 \leq i \leq d-2$, each column of X_{i+1} consists of n_{i+1} consecutive columns of X_i. This is true for any $d \geq 3$ so it suffices to show the statement holds when $\mathcal{X} \in \mathbb{C}^{n_1 \times n_2 \times n_3}$.

For notational simplicity, we denote the frontal slices of $\mathcal{X}$ by $X_f^{(j)} = \mathcal{X}(:,:,j) \in \mathbb{C}^{n_1 \times n_2}$ for $1 \leq j \leq n_3$ and the lateral slices by $X_\ell^{(k)} = \mathcal{X}(:,k,:) \in \mathbb{C}^{n_1 \times n_3}$ for $1 \leq k \leq n_2$. Then, by construction, we have

$$X_1 = \begin{bmatrix} X_f^{(1)} & \cdots & X_f^{(n_3)} \end{bmatrix}, \quad X_2 = \begin{bmatrix} X_\ell^{(1)} \\ \vdots \\ X_\ell^{(n_2)} \end{bmatrix}.$$

The columns of the frontal slices and those of the lateral slices are mode-1 fibers of the tensor $\mathcal{X}$, so we can find the same column in the frontal and lateral slices. That is,

$$\left(X_\ell^{(k)}\right)_j = \left(X_f^{(j)}\right)_k = \mathcal{X}(:,k,j), \quad 1 \leq j \leq n_3, \quad 1 \leq k \leq n_2,$$

where $\left(X_\ell^{(k)}\right)_j$ denotes the jth column of $X_\ell^{(k)}$, and similarly for $\left(X_f^{(j)}\right)_k$. Given $X_1 = U_1 V_1^*$, we can write X_2 as

$$X_2 = \begin{bmatrix} U_1 \left(V_1^{(1)}\right)^* \\ \vdots \\ U_1 \left(V_1^{(n_2)}\right)^* \end{bmatrix} = (I \otimes U_1) Z^*,$$

where $V_1^{(i)}$ is a submatrix that contains rows $i, i+n_2, \ldots, i+(n_3-1)n_2$ of V_1 for $1 \leq i \leq n_2$, and '$\otimes$' is the Kronecker product of two matrices.

Since U_1 has orthonormal columns, $I \otimes U_1$ has orthonormal columns, and $Z^* = ST^*$, where $S \in \mathbb{C}^{r_1 n_2 \times r_2}$ has orthonormal columns and $T \in \mathbb{C}^{n_3 \times r_2}$. Since by assumption $X_2 = U_2 V_2^*$, without loss of generality, we can assume that $U_2 = (I \otimes U_1)S$; otherwise, there is an orthogonal matrix $Y \in \mathbb{C}^{r_2 \times r_2}$ such that $U_2 = (I \otimes U_1)SY$, $T = TY$, and $V_2 = V_2 Y$. By reshaping $U_2 = (I \otimes U_1)S$, we find

$$\mathrm{reshape}(U_2, n_1, n_2 r_2) = U_1 W, \quad W = \mathrm{reshape}(S, r_1, n_2 r_2),$$

which proves the statement for $d = 3$. □

One may notice that for $1 \leq k \leq d-2$, $\text{reshape}(W_k, r_k, n_{k+1}, r_{k+1})$ is the size of the $(k+1)$st core in a TT format, where W_k is defined in Theorem 3.1.1. In the next theorem, we show the accuracy of the approximation if we construct a tensor with TT cores given by $\text{reshape}(W_k, r_k, n_{k+1}, r_{k+1})$.

Theorem 3.1.2. *Let $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, and $0 < \epsilon < 1$. Suppose further that for $1 \leq j \leq d-1$, each unfolding admits $\|X_j - U_j V_j^*\|_F \leq \frac{\epsilon}{\sqrt{d-1}} \|\mathcal{X}\|_F$, where U_j has orthonormal columns. Then, the tensor $\tilde{\mathcal{X}}$ constructed by TT cores $\mathcal{G}_1 = U_1$, $\mathcal{G}_d = V_{d-1}^*$, and*

$$\mathcal{G}_{k+1} = \text{reshape}\left(U_k^* \,\text{reshape}\left(U_{k+1}, \prod_{i=1}^{k} n_i, n_{k+1} r_{k+1}\right), r_k, n_{k+1}, r_{k+1}\right), \tag{3.1}$$

for $1 \leq k \leq d-2$, satisfies $\|\mathcal{X} - \tilde{\mathcal{X}}\|_F \leq \epsilon \|\mathcal{X}\|_F$.

Proof. Since the TT cores are computed with $U_1, \ldots, U_{d-1}$, we can express each element of $\tilde{\mathcal{X}}$ with the same matrices:

$$\begin{aligned}
\tilde{\mathcal{X}}_{i_1, i_2 \ldots, i_d} &= \mathcal{G}_1(i_1, :)\mathcal{G}_2(:, i_2, :) \cdots \mathcal{G}_d(:, i_d) \\
&= U_1(i_1, :) U_1^* U_2((i_2 - 1)n_1 + 1 : i_2 n_1, :) \cdots \\
&\quad U_{d-2}^* U_{d-1} \left(\left((i_{d-1} - 1) \prod_{k=1}^{d-2} n_k + 1 \right) : \left(i_{d-1} \prod_{k=1}^{d-2} n_k \right), : \right) U_{d-1}^* X_{d-1}(:, i_d).
\end{aligned}$$

This indicates that $\tilde{X}_{d-1}$, the $(d-1)$th unfolding of $\tilde{X}$, is an approximation of X_{d-1} with $(d-1)$ layers of projection using U_j for $1 \leq j \leq d-1$. Therefore, to show the error of this approximation, we need to consider the error generated by each layer of projection. Now, let $Y_{d-1} = U_{d-1} U_{d-1}^* X_{d-1}$ be the approximation of X_{d-1}, and construct two sequences of matrices $(\tilde{Y}_{d-1}, \cdots, \tilde{Y}_2)$ and $(Y_{d-2}, \cdots, Y_1)$. For $1 \leq j \leq d-2$, we use $\tilde{Y}_{j+1} = \text{reshape}\left(Y_{j+1}, \prod_{k=1}^{j} n_k, \prod_{k=j+1}^{d} n_k\right)$ to reshape the previous approximation to a matrix compatible in size with X_j, and $Y_j = U_j U_j^* \tilde{Y}_{j+1}$ to apply another layer of

approximation with U_j. Then,

$$\begin{aligned}
\|\mathcal{X} - \tilde{\mathcal{X}}\|_F^2 &= \|Y_1 - X_1\|_F^2 \\
&= \|U_1 U_1^* \tilde{Y}_2 - X_1\|_F^2 \\
&= \|U_1 U_1^* (\tilde{Y}_2 - X_1) + (U_1 U_1^* - I) X_1\|_F^2 \\
&= \|U_1 U_1^* (\tilde{Y}_2 - X_1)\|_F^2 + \|(U_1 U_1^* - I) X_1\|_F^2 \\
&\leq \|\tilde{Y}_2 - X_1\|_F^2 + \|(U_1 U_1^* - I) X_1\|_F^2 \\
&= \|Y_2 - X_2\|_F^2 + \|(U_1 U_1^* - I) X_1\|_F^2,
\end{aligned}$$

where the fourth equality holds since $(U_1 U_1^* (\tilde{Y}_2 - X_1))^* ((U_1 U_1^* - I) X_1) = 0$, the inequality holds since $U_1 U_1^*$ is an orthogonal projection, and the last equality holds as reshaping preserves Frobenius norm. Following this argument, by induction on $\|Y_j - X_j\|_F^2$ for $1 \leq j \leq d-1$, we have

$$\begin{aligned}
\|\mathcal{X} - \tilde{\mathcal{X}}\|_F^2 &\leq \sum_{k=1}^{d-1} \|(I - U_k U_k^*) X_k\|_F^2 \\
&= \sum_{k=1}^{d-1} \|(I - U_k U_k^*)(X_k - U_k V_k^*)\|_F^2 \\
&\leq \sum_{k=1}^{d-1} \|X_k - U_k V_k^*\|_F^2 \leq \epsilon^2 \|\mathcal{X}\|_F^2.
\end{aligned} \tag{3.2}$$

□

Theorem 3.1.2 provides an algorithm to compute a tensor $\tilde{\mathcal{X}}$ in TT format that approximates $\mathcal{X}$ (see Algorithm 8). It is also simple to observe that Algorithm 8 can be performed in parallel, since the unfoldings of a tensor are independent.

Since the unfoldings have different sizes, the amount of work assigned to each processor in Algorithm 8 varies. Roughly speaking, processors that deal with X_j when j is close to $\lfloor d/2 \rfloor$ have the most computationally expensive

Algorithm 8 Parallel-TTSVD: Given a tensor, compute an approximant tensor in TT format using SVD in parallel.

Input: A tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ and a desired accuracy $0 < \epsilon < 1$
Output: TT cores $\mathcal{G}_1, \ldots, \mathcal{G}_d$ of an approximant $\tilde{\mathcal{X}}$

1: **for** $1 \leq j \leq d-1$ **do**
2: Compute a rank r_j approximation of the jth flattening of $\mathcal{X}$ in truncated SVD form so that $\|X_j - U_j \Sigma_j V_j^*\|_F \leq \epsilon \|\mathcal{X}\|_F / \sqrt{d-1}$.
3: **for** $1 \leq k \leq d-2$ **do**
4: Calculate $W_{k+1} = U_k^* \operatorname{reshape}(U_{k+1}, \prod_{i=1}^{k} n_i, n_{k+1} r_{k+1})$.
5: Set $\mathcal{G}_{k+1} = \operatorname{reshape}(W_{k+1}, r_k, n_{k+1}, r_{k+1})$.
6: Set $\mathcal{G}_1 = U_1$ and $\mathcal{G}_d = \Sigma_{d-1} V_{d-1}^*$.

SVD. In practice, one can replace the SVD in Algorithm 8 with the randomized SVD [96], in which case the computational complexity on each processor is $\mathcal{O}(r_j \prod_{i=1}^{d} n_i)$ where r_j is the rank of the jth unfolding. In this scenario, Algorithm 8 is ideal when all the r_j's are equal so that the computation is evenly distributed across all the processors.

However, when one implements Algorithm 8 in a shared memory computing environment, such as on a multi-threaded computer, a copy of the entire tensor $\mathcal{X}$ needs to be made on each processor. This might not be feasible in many cases due to the size of the tensor. Therefore, Algorithm 8 is not practical in these scenarios. Together with Theorem 3.1.2, they mainly serve for theoretical purposes to illustrate the effectiveness of Theorem 3.1.1, and build the cornerstone for more practical algorithms such as sketching discussed in the next section.

3.1.2 Parallel TT Sketching

When we cannot afford to store an individual copy of the tensor on each processor, we cannot use SVD as it requires all the tensor entries to be in cache. Instead, we may only be able to read a small portion of the entries of $\mathcal{X}$ at a time before discarding.

A common idea in this scenario for large matrices and tensors is sketching, where information about the matrix or tensor is obtained via matrix-vector multiplications. This idea is used for computing low-rank approximations of matrices [96], Tucker decomposition on tensors [141,206], and TT decomposition [41]. In particular, SVDs in TTSVD can be replaced by sketching and a randomized range finder [41]. Here, we develop a parallel TT sketching algorithm based on Theorem 3.1.1 (see Algorithm 9). Since we want a truncated QR decomposition to reveal the rank of a given matrix, we use the column pivoted QR (CPQR) [40]. This is a so-called "two-pass" algorithm since $\mathcal{X}$ is used twice: the first time to compute an orthonormal basis of the column space of each unfolding, and the second time to compute the last TT core. The implementation details of finding the orthonormal basis are in section 3.2.2.

Since Q_j has orthonormal columns for $1 \leq j \leq d-1$, (3.2) provides an error bound for Theorem 9.

Theorem 3.1.3. *Let $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, $\boldsymbol{r}$ be the desired TT core size, and $p \geq 2$. The approximation $\tilde{\mathcal{X}}$ computed in Theorem 9 satisfies*

$$\mathbb{E}\left[\|\mathcal{X} - \tilde{\mathcal{X}}\|_F^2\right] \leq \sum_{j=1}^{d-1} \left(1 + \frac{r_j}{p-1}\right) \left(\sum_{k=r_j+1}^{M_j} \sigma_k^2(X_j)\right), \tag{3.3}$$

where $\sigma_k(X_j)$ is the kth singular value of X_j and $M_j = \min(\prod_{i=1}^{j} n_i, \prod_{i=j+1}^{d} n_i)$.

Algorithm 9 PSTT: Given a tensor, compute an approximant tensor in TT format using sketching.

Input: A tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, TT core size $\boldsymbol{r}$, and an oversampling parameter p

Output: TT cores $\mathcal{G}_1, \ldots, \mathcal{G}_d$ of an approximant $\tilde{\mathcal{X}}$

1: **for** $1 \leq j \leq d-1$ **do**
2: Generate $\Phi_j \in \mathbb{R}^{\left(\prod_{k=j+1}^{d} n_k\right) \times (r_j + p)}$ with i.i.d. standard Gaussian entries.
3: Calculate $S_j = X_j \Phi_j$, where X_j is the jth flattening of $\mathcal{X}$
4: Compute a CPQR of S_j to obtain Q_j with orthonormal cols and set $Q_j = Q_j(:, 1 : r_j)$.
5: **for** $1 \leq k \leq d-2$ **do**
6: Calculate $W_{k+1} = Q_k^* \operatorname{reshape}(Q_{k+1}, \prod_{i=1}^{k} n_i, n_{k+1} r_{k+1})$.
7: Set $\mathcal{G}_{k+1} = \operatorname{reshape}(W_{k+1}, r_k, n_{k+1}, r_{k+1})$.
8: Set $\mathcal{G}_1 = Q_1$, and $\mathcal{G}_d = Q_{d-1}^* X_{d-1}$.

Further assume that $p \geq 4$, then for all $u, t \geq 1$,

$$\|\mathcal{X} - \tilde{\mathcal{X}}\|_F^2 \leq \sum_{j=1}^{d-1} \left((1 + t\sqrt{12 r_j / p}) \left(\sum_{k=r_j+1}^{M_j} \sigma_k^2(X_j) \right)^{\frac{1}{2}} + ut \frac{e\sqrt{r_j + p}}{p+1} \sigma_{r_j+1}(X_j) \right)^2, \tag{3.4}$$

with failure probability at most $5t^{-p} + 2e^{-u^2/2}$.

Proof. The expectation bound in (3.3) follows from (3.2) and [96, Thm 10.5]. The probability bound in (3.4) follows from (3.2) and [96, Thm 10.7]. □

If the accuracy in Theorem 3.1.2 is considered as a baseline, then (3.3) implies that the error is within a constant factor of the baseline for moderate p with high probability. We can understand the probability bound as two parts. The sum of squares of the "tail" singular values corresponds with the expected approximation error in (3.3). Analogously, the $(j+1)$st singular value of each unfolding in the second term is related to the deviation above the mean.

A bottleneck of Theorem 9 is the size of the dimension reduction maps

(DRMs) Φ_j, which grow exponentially with d. In practice, we can substitute them with Khatri-Rao products of smaller DRMs [207]. In other words, for $1 \le j \le d-1$, instead of using $\Phi_j \in \mathbb{R}^{\prod_{k=j+1}^{d} n_k \times (r_j+p)}$ with independent and identically distributed (i.i.d.) standard Gaussian entries, we use $\Psi_d^{(j)} \odot \cdots \odot \Psi_{j+1}^{(j)}$, where '$\odot$' denotes the Khatri-Rao product, and $\Psi_k^{(j)} \in \mathbb{R}^{n_k \times (r_j+p)}$ has i.i.d. standard Gaussian entries for $j+1 \le k \le d$. If we partition X_j into $\prod_{k=j+2}^{d} n_k$ subblocks each of size $\prod_{k=1}^{j} n_k \times n_{j+1}$ as $X_j = \begin{bmatrix} (X_j)_1 & \cdots & (X_j)_{\prod_{k=j+2}^{d} n_k} \end{bmatrix}$, then

$$X_j \left(\Psi_d^{(j)} \odot \cdots \odot \Psi_{j+1}^{(j)} \right) = \sum_{\ell_d=1}^{n_d} \cdots \sum_{\ell_{j+2}=1}^{n_{j+2}} (X_j)_{I(\ell_{j+2},\ldots,\ell_d)} \Psi_{j+1}^{(j)} D_{\ell_d} \cdots D_{\ell_{j+2}}, \tag{3.5}$$

where $I(\ell_{j+2}, \ldots, \ell_d) = \sum_{k=j+3}^{d} (\ell_k - 1) \prod_{p=j+2}^{k-1} n_p + (\ell_{j+2} - 1)$ denotes the index of one subblock, and D_{ℓ_q} is a diagonal matrix whose diagonal elements are the elements on row ℓ_q of $\Psi_q^{(j)}$ for $j+2 \le q \le d$. In other words, the product of a tensor unfolding and a DRM is converted to a combination of subblocks of the unfolding. Algorithm 9 with the Khatri-Rao DRMs gives slightly less accurate approximations in practice, but the storage cost is only linear in $\boldsymbol{n}$ and d.

When the tensor $\mathcal{X}$ is too large to access its elements for a second time in line 8 of Algorithm 9, we design a "one-pass" (or "single-pass") algorithm that computes $\mathcal{G}_d$ without using X_{d-1} directly [206]. To be specific, we generate a new dimension reduction map $\Psi_{d-1} \in \mathbb{R}^{\left(\prod_{k=1}^{d-1} n_k\right) \times (r_{d-1}+p)}$ with i.i.d. standard Gaussian entries and compute $T_{d-1} = \Psi_{d-1}^* X_{d-1}$ in lines 2 and 3 when $j = d-1$. Then,

$$T_{d-1} \approx \Psi_{d-1}^* Q_{d-1} Q_{d-1}^* X_{d-1} = (\Psi_{d-1}^* Q_{d-1}) \mathcal{G}_d. \tag{3.6}$$

In this way, we have $\mathcal{G}_d \approx (\Psi_{d-1}^* Q_{d-1})^\dagger T_{d-1}$, where the pseudo-inverse exists with probability 1. We use PSTT-onepass to denote this single-pass version of Algorithm 9. Since all TT cores except the last one are obtained as in PSTT, we

can determine the expected error of PSTT-onepass by combining (3.3) and the error of using single-pass approximation on the last TT core.

Theorem 3.1.4. *Let $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$, $\boldsymbol{r}$ be the desired TT core size, and $p \geq 2$. The approximation $\tilde{\tilde{\mathcal{X}}}$ computed by PSTT-onepass satisfies*

$$\mathbb{E}\left[\|\mathcal{X} - \tilde{\tilde{\mathcal{X}}}\|_F^2\right] \leq \sum_{j=1}^{d-2}\left(1 + \frac{r_j}{p-1}\right)\sum_{k=r_j+1}^{M_j}\sigma_k^2(X_j) + \left(1 + \frac{r_{d-1}}{p-1}\right)^2 \sum_{k=r_{d-1}+1}^{M_{d-1}}\sigma_k^2(X_{d-1}),$$

where $\sigma_k(X_j)$ is the kth singular value of X_j and $M_j = \min(\prod_{i=1}^{j} n_i, \prod_{i=j+1}^{d} n_i)$ for $1 \leq j \leq d-1$.

Proof. As in the proof of Theorem 3.1.2, we build two sequences of matrices to understand our approximation. Let $Z_{d-1} = Q_{d-1}(\Psi_{d-1}^* Q_{d-1})^\dagger \Psi_{d-1}^* X_{d-1}$, and for each $1 \leq j \leq d-2$, let $\tilde{Z}_{j+1} = \text{reshape}\left(Z_{j+1}, \prod_{k=1}^{j-1} n_k, \prod_{k=j}^{d} n_k\right)$ and $Z_j = Q_j Q_j^* \tilde{Z}_{j+1}$. Then, from (3.2) we find that

$$\|\mathcal{X} - \tilde{\tilde{\mathcal{X}}}\|_F^2 \leq \sum_{j=1}^{d-2} \|(I - Q_j Q_j^*)X_j\|_F^2 + \|Z_{d-1} - X_{d-1}\|_F^2.$$

So, we only need to bound the second term on the right hand side, which represents the error of using single-pass approximation on the last TT core. Let $E_{d-1} = (\Psi_{d-1}^* Q_{d-1})^\dagger \Psi_{d-1}^* X_{d-1}$, then

$$\begin{aligned}
\|Z_{d-1} - X_{d-1}\|_F^2 &= \|Q_{d-1}E_{d-1} - X_{d-1}\|_F^2 \\
&= \|Q_{d-1}Q_{d-1}^* Q_{d-1}E_{d-1} - Q_{d-1}Q_{d-1}^* X_{d-1} + Q_{d-1}Q_{d-1}^* X_{d-1} - X_{d-1}\|_F^2 \\
&= \|Q_{d-1}E_{d-1} - Q_{d-1}Q_{d-1}^* X_{d-1}\|_F^2 + \|Q_{d-1}Q_{d-1}^* X_{d-1} - X_{d-1}\|_F^2 \\
&= \|E_{d-1} - Q_{d-1}^* X_{d-1}\|_F^2 + \|(I - Q_{d-1}Q_{d-1}^*)X_{d-1}\|_F^2,
\end{aligned}$$

where the third equality holds since $Q_{d-1}Q_{d-1}^*$ and $I - Q_{d-1}Q_{d-1}^*$ are orthogonal

projectors. We are left to bound $\|E_{d-1} - Q_{d-1}^* X_{d-1}\|_F^2$. Plugging in E_{d-1}, we have

$$\begin{aligned}\|E_{d-1} - Q_{d-1}^* X_{d-1}\|_F^2 &= \|(\Psi_{d-1}^* Q_{d-1})^\dagger \Psi_{d-1}^* X_{d-1} - Q_{d-1}^* X_{d-1}\|_F^2 \\ &= \|(\Psi_{d-1}^* Q_{d-1})^\dagger \Psi_{d-1}^* X_{d-1} - (\Psi_{d-1}^* Q_{d-1})^\dagger (\Psi_{d-1}^* Q_{d-1}) Q_{d-1}^* X_{d-1}\|_F^2 \\ &= \|(\Psi_{d-1}^* Q_{d-1})^\dagger \Psi_{d-1}^* (I - Q_{d-1} Q_{d-1}^*) X_{d-1}\|_F^2.\end{aligned}$$

The error bound follows from [206, Lemma B.1]. □

The CPQR in line 4 of Algorithm 9 can be expensive when j is large, and this still remains an issue when we use the "single-pass" algorithm. It is simple to notice that Theorem 3.1.1 also holds for the row spaces of the unfoldings of $\mathcal{X}$. Let $d_* = \lceil \frac{d}{2} \rceil$. Then in practice, we compute Q_j, orthonormal bases for the column spaces of X_j when $j < d_*$, and P_j, orthonormal bases for the row spaces of X_j when $j > d_*$. In this way, the TT cores $\mathcal{G}_j$ for $j \neq d_*$ can be calculated similarly using lines 6 and 7 of Algorithm 9, and the TT core in the middle $\mathcal{G}_{d_*}$ needs one extra step

$$\mathcal{G}_{d_*} = \text{reshape}\left(\mathcal{X}, \prod_{j=1}^{d_*-1} n_j, n_{d_*}, \prod_{j=d_*+1}^{d} n_j\right) \times_1 Q_{d_*-1}^* \times_3 P_{d_*+1}^*. \tag{3.7}$$

We call this variation PSTT2 to indicate that both column spaces and row spaces of tensor unfoldings are utilized, and the accuracy bounds in Theorem 3.1.3 continue to hold for PSTT2. Moreover, one can design PSTT2-onepass, a one-pass version of PSTT2, by carrying out an extra sketching step for the middle unfolding X_{d_*}. The sketching step can be performed on either the row or column space of X_{d_*}, and a pseudo-inverse follows it as in (3.6) to obtain $\mathcal{G}_{d_*}$.

3.1.3 Parallel TT and orthogonal Tucker conversion

Some applications, including signal processing [54], computer vision [216], and chemical analysis [105], construct and manipulate tensor data in the Tucker format. If one wants to explore latent structures in the TT format, a common approach is to convert the tensor back to the original format and then perform a TT decomposition. This method has a major drawback: it needs to explicitly construct the tensor in the original format, which ignores the low-rank structure one intends to utilize in both TT and Tucker format. Here, we develop a method to directly approximate a tensor $\mathcal{X}$ in orthogonal Tucker format by another tensor $\tilde{\mathcal{X}}$ in TT format.

If $\mathcal{X}$ has an orthogonal Tucker format (1.7), and $\mathcal{H}_1, \ldots, \mathcal{H}_d$ are the TT cores of $\mathcal{G}$, then the TT cores of the tensor $\mathcal{E} \times_j A_j$ are $\mathcal{H}_1, \ldots, \mathcal{H}_{j-1}, \mathcal{H}_j \times_2 A_j, \mathcal{H}_{j+1}, \ldots, \mathcal{H}_d$. In this way, (1.7) is equivalent to updating each TT core of $\mathcal{G}$ independently with the Tucker factor matrices. Therefore, we can use these updated cores as the TT cores of an approximation $\tilde{\mathcal{X}}$ (see Algorithm 10).

Algorithm 10 Tucker2TT: Given the Tucker decomposition of a tensor, compute an approximant tensor in TT format.

Input: The Tucker core $\mathcal{E}$ and factor matrices $A_1, \ldots, A_d$ of a tensor $\mathcal{X}$ (see (1.7))
Output: The TT cores $\mathcal{T}_1, \ldots, \mathcal{T}_d$ of an approximant tensor $\tilde{\mathcal{X}}$

1: Perform a parallel TT decomposition on $\mathcal{E}$ and get TT cores $\mathcal{H}_1, \ldots, \mathcal{H}_d$.
2: **for** $1 \leq j \leq d$ **do**
3: Set $\mathcal{T}_j = \mathcal{H}_j \times_2 A_j$.

As the Tucker core, $\mathcal{E}$ is much smaller in size than the original tensor $\mathcal{X}$, so line 1 of Algorithm 10 is computationally cheaper than forming $\mathcal{X}$ explicitly and computing a TT decomposition. In addition, the parallel TT decomposition to be used in line 1 depends on whether there is a prescribed accuracy of $0 < \epsilon < 1$ or a prescribed TT rank $\boldsymbol{r}$. Finally, =since all A_j's have orthonormal columns,

and

$$X_j = (A_j \otimes \cdots \otimes A_1) E_j (A_d \otimes \cdots \otimes A_{j+1})^T,$$

we find that $\operatorname{rank}(X_j) = \operatorname{rank}(E_j)$ for $1 \leq j \leq d-1$. Therefore, the accuracy of Algorithm 10 depends on the parallel TT decomposition in line 1.

If one is provided with a tensor $\mathcal{X}$ in TT format, then it is also possible to find an approximation $\tilde{\mathcal{X}}$ in orthogonal Tucker format. By explicit calculation, one finds that X_j, the jth unfolding of the tensor $\mathcal{X}$ for $1 \leq j \leq d$, can be computed with the unfoldings of the TT cores:

$$X_j = \prod_{i=1}^{j-1} \left(I_{\prod_{k=i+1}^{j} n_k} \otimes (G_i)_2 \right) (G_j)_2 (G_{j+1})_1 \prod_{i=j+2}^{d} \left((G_i)_1 \otimes I_{\prod_{k=j+1}^{i-1} n_k} \right), \tag{3.8}$$

where I_n is the identity matrix of size $n \times n$, and $(G_i)_p$ is the pth unfolding of $\mathcal{G}_i$ for $1 \leq p \leq 2$. Then, rewriting (3.8) gives $X_j = P_j (G_j)_2 Q_j$ and

$$P_j = I_{n_j} \otimes \left(\prod_{i=1}^{j-1} \left(I_{\prod_{k=i+1}^{j-1} n_k} \otimes (G_i)_2 \right) \right), \quad Q_j = \prod_{i=j+1}^{d} \left((G_i)_1 \otimes I_{\prod_{k=j+1}^{i-1} n_k} \right).$$

We find that

$$\begin{aligned} \operatorname{reshape}(\mathcal{X}, \prod_{k=1}^{j-1} n_k, n_j, \prod_{k=j+1}^{d} n_k) = \mathcal{G}_j \times_1 & \left(\prod_{i=1}^{j-1} \left(I_{\prod_{k=i+1}^{j-1} n_k} \otimes (G_i)_2 \right) \right) \\ & \times_3 \left(\prod_{i=j+1}^{d} \left((G_i)_1 \otimes I_{\prod_{k=j+1}^{i-1} n_k} \right) \right). \end{aligned}$$

We can therefore find relationships between matricizations of $\mathcal{X}$ and those of $\mathcal{G}_j$:

$$X_{(j)} = (G_j)_{(2)} \left[\left(\prod_{i=j+1}^{d} (G_i)_1 \otimes I_{\prod_{k=j+1}^{i-1} n_k} \right) \otimes \left(\prod_{i=1}^{j-1} I_{\prod_{k=i+1}^{j-1} n_k} \otimes (G_i)_2 \right)^T \right],$$

where $X_{(j)}$ is the jth matricization of $\mathcal{X}$ and $(G_j)_{(2)}$ is the second matricization of $\mathcal{G}_j$. When $(G_1)_2, \ldots, (G_{j-1})_2$ have linearly independent columns and $(G_{j+1})_1, \ldots, (G_d)_1$ have linearly independent rows, the column space of $X_{(j)}$ and $(G_j)_{(2)}$ match. This criterion can be satisfied when the TT core size is the TT

rank of $\mathcal{X}$. In this way, since Tucker decomposition algorithms such as HOSVD or Tucker sketching use orthonormal bases of column spaces of $X_{(j)}$ as factor matrices, we can find an orthogonal Tucker approximation of $\mathcal{X}$ simply by replacing $X_{(j)}$ by $(G_j)_{(2)}$. We summarize this procedure in Algorithm 11, and use a particular version of HOSVD (see [23, Alg. 1]). This algorithm is also equivalent to the computing routines introduced in [58, Sec. 4.1], with simpler tensor notations.

Algorithm 11 TT2Tucker: Given a tensor in TT format, compute an approximant tensor in orthogonal Tucker format.

Input: The TT cores $\mathcal{T}_1, \ldots, \mathcal{T}_d$ of a tensor $\mathcal{X}$
Output: The TT cores $\mathcal{H}_1, \ldots, \mathcal{H}_d$ of the Tucker core $\mathcal{E}$, and factor matrices $A_1, \ldots, A_d$ of $\tilde{\mathcal{X}}$
1: **for** $1 \leq j \leq d$ **do**
2: Compute A_j with orthonormal columns that approximates column space of $(T_j)_{(2)}$.
3: Calculate $\mathcal{H}_j = \mathcal{T}_j \times_2 A_j^*$.

This algorithm is parallelizable since the TT cores of $\mathcal{X}$ are independent. Users need to either prescribe a desired accuracy or a multilinear rank to discover orthonormal bases of column spaces in step 2. HOSVD guarantees the performance of this algorithm if we use SVD, or [206, Thm. 5.1] if we use sketching. Compared to HOSVD or Tucker sketching, Algorithm 11 is faster due to the known TT cores. As a result, memory costs and computation powers can be significantly reduced.

3.2 Complexity Analysis and Numerical Examples

In this section, we model the computational and spatial cost of the proposed methods for TT decomposition, and compare these models with numerical experiments[2]. For simplicity, we assume that the tensor $\mathcal{X}$ of dimension d is "square," meaning that $n_i = n$ for all $1 \leq i \leq d$ and $r_i = r$ for all $1 \leq i \leq d-1$.

We focus on the two-sided sketching methods PSTT2 and PSTT2-onepass defined at the end of section 3.1.2, since both the SVD based and the one-sided PSTT algorithms have poor spatial complexity. In addition, the SVD algorithm is much slower than the counterparts with sketching. As a baseline, we compare the results to a modified version of Algorithm 5.1 in [41], which is referred to in this manuscript as Serial Streaming TT Sketching (SSTT).

In section 3.2.1, we discuss the computational environment and software used to implement our methods. Next, in section 3.2.2, we discuss how PSTT2, PSTT2-onepass, and SSTT are modified to be performed in a distributed memory environment. In section 3.2.3, we discuss the improvements of PSTT2 and PSTT2-onepass over previous serial algorithms in terms of spatial complexity. Finally, in section 3.2.4 we discuss time complexity of these algorithms. To illustrate the practicality of our algorithms, we show numerical experiments in section 3.2.3 and section 3.2.4.

[2]For codes, see `https://github.com/SidShi/Parallel_TT_sketching`

3.2.1 Computational Details

All experiments are performed in C, using the OpenMPI implementation of MPI for parallelization. All subroutines take advantage of LAPACKE and BLAS to vectorize large linear algebra operations, such as matrix-matrix multiplication and QR factorization. Experiments are performed on a machine with eight 12-core compute nodes, each consisting of two Intel Xeon E5-2620 v3 processors with 32 GB per node. For each trial, we measure the relative Frobenius error of the TT approximation, and achieve $\epsilon < 10^{-10}$ (see (1.5)). For our experiments, we assume that the ranks are known *a priori*. Nevertheless, when they are not known, one can implement an adaptive algorithm such as Algorithm 5.2 in [41].

To measure the memory improvements of the two-sided methods, we compare the total allocated memory using the gperftools implementation of TC-Malloc. In particular, the memory is measured on each core and then averaged. Transient memory allocations by MPI are ignored in the overall memory measurement.

Throughout the following sections, we refer to some standard MPI functions to describe the parallel algorithm. These primarily include

- *Send:* the operation of sending some array of memory from one core to another.
- *Receive:* the operation of receiving the memory sent by *send*.
- *Reduce:* the operation of summing matrices from a group of cores, used to summarize information gained by individual cores.

To avoid confusion in the following sections, we use "core" to refer to a single

core of the CPU. This is different from a "TT core" in the TT format.

3.2.2 Parallel Tensor Sketching

We first discuss the process of parallelizing SSTT, PSTT2, and PSTT2-onepass. We focus on the parallel sketching step, which is responsible for most of the time and storage. The predominance of the sketching step is emblematic of the "compress-then-combine" approach, as most of the computational work happens when compressing. The other steps needed for a fully parallel algorithm are discussed at the end of the section.

The parallel algorithms have a structure that is reminiscent of dense matrix-matrix multiplication algorithms such as Cannon's algorithm [38] or SUMMA [214]. These algorithms perform the operation $C = AB$ by dividing A, B, and C into submatrices and distributing these submatrices across multiple cores. In this way, the overall spatial complexity is reduced.

For the parallel implementation of the TT sketching algorithms, a natural extension is to partition an unfolding X_j into submatrices, and then proceed with standard matrix-matrix multiplications for the sketch step (see Algorithm 9)

$$S_i = X_i \Phi_i. \tag{3.9}$$

However, a computational hurdle with this approach is that we need to sketch multiple unfoldings during the same computational step since we want as few passes of the tensor as possible. As such, we aim for a guarantee that if M is a submatrix of X_i, then some reshaping of M is also a submatrix of $X_{i'}$ for some $i \neq i'$. In this way, parallel linear algebra algorithms work equally well for all

unfoldings.

For this reason, we introduce the notion of a *sub-tensor*, which is a multilinear generalization of the submatrix. Sub-tensors have previously been used for computing the Tucker decomposition in parallel [16] as well as hierarchical sub-tensor decompositions [66]. They have also been proposed specifically for the matricized-tensor times Khatri-Rao product (MTTKRP) in [17,18], which is similar to the algorithm we develop in this section. In each dimension k, we partition the tensor index $1 \leq k_i \leq n$ into P_i equal "chunks" (or in the case that P_i does not divide into n, nearly equal chunks). In MATLAB notation, we have that $\mathcal{Y}_j$ is a sub-tensor of $\mathcal{X}$ if

$$\mathcal{Y}_j = \mathcal{X}\left(1 + \frac{n(j_1-1)}{P_1} : \frac{nj_1}{P_1}, 1 + \frac{n(j_2-1)}{P_2} : \frac{nj_2}{P_2}, \ldots, 1 + \frac{n(j_d-1)}{P_d} : \frac{nj_d}{P_d}\right),$$

where $\mathbf{j} = (j_1, \ldots, j_d)$ is a multi-index specifying the target sub-tensor. Defining $P = \prod_{k=1}^{d} P_k$, it is clear that the memory needed to store $\mathcal{Y}_j$ is a factor of $1/P$ smaller than the memory needed to store $\mathcal{X}$. We also notice that any unfolding of $\mathcal{Y}_j$ is a submatrix of an unfolding of $\mathcal{X}$, although those submatrices are not necessarily contiguous. Lastly, for notational simplicity, we use the column-major block index

$$j = j_1 + (j_2 - 1)P_1 + \cdots + (j_d - 1)P_1 \ldots P_{d-1},$$

and in this way, we have $\mathcal{Y}_{\mathbf{j}} = \mathcal{Y}_j$.

The sub-tensor gives an efficient method to store one part of the multiplication in (3.9). To be specific, the matrix Φ_i can be stored in d or fewer matrices of size $\mathcal{O}(nr)$ via the Khatri-Rao DRMs in (3.5). This is a small enough cost that each core can store every Φ_i. As a result, the only thing we need to distribute is the "sketch" S_i. In fact, the matrices S_i dominate the memory complexity of

PSTT2 and PSTT2-onepass (see section 3.2.3), and are therefore important to distribute. We distribute them by splitting S_i into *sub-sketches* $S_{i,k}$, with the column sub-sketches defined by

$$S_{i,k} = \text{reshape}(S_i, \underbrace{n, \ldots, n}_{d-L}, r) \left(1 + \frac{n(k_1 - 1)}{P_1} : \frac{nk_1}{P_1}, \ldots, 1 + \frac{n(k_{d-i} - 1)}{P_{d-i}} : \frac{nk_{d-i}}{P_{d-i}}, :\right). \tag{3.10}$$

We note that sketches for finding row spaces are performed by simply using X_j^T instead of X_j in (3.9). Row sub-sketches and the other necessary steps can be immediately derived from the same reasoning.

Using the notions of sub-tensors and sub-sketches, we can expand (3.9) into an explicit form with the Khatri-Rao DRMs Ψ_ℓ as $S_{i,k} = \sum_{j_{i+1}=1}^{P_{i+1}} \cdots \sum_{j_d=1}^{P_d} S_{i,k,j}$ with

$$\begin{aligned} S_{i,k,j} =& \text{reshape} \left(\mathcal{Y}_j, \frac{n}{P_1}, \ldots, \frac{n}{P_i}, \prod_{\ell=i+1}^{d} \frac{n}{P_\ell}\right) \times_{i+1} \\ & \left(\Psi_d^{(i)} \left(1 + \frac{n(j_d - 1)}{P_d} : \frac{nj_d}{P_d}, :\right) \odot \cdots \odot \Psi_{i+1}^{(i)} \left(1 + \frac{n(j_{i+1} - 1)}{P_{i+1}} : \frac{nj_{i+1}}{P_{i+1}}, :\right)\right)^T, \end{aligned} \tag{3.11}$$

where $S_{i,k,j}$ is the contribution of the sub-tensor $\mathcal{Y}_j$ to the sub-sketch $S_{i,k}$, under the condition that $\mathbf{j}(1:i) = \mathbf{k}$.

Now, we write down the parallel sketching procedure, which is a subroutine (see Algorithm 12) for PSTT2, PSTT2-onepass, and SSTT. For generality, we assume that the input to each algorithm is some function f that takes as input the index of the tensor $(\ell_1, \ldots, \ell_d)$, and outputs a value of the tensor $\mathcal{X}_{\ell_1,\ldots,\ell_d}$. This function allows us to load parts of the tensor into memory without necessarily loading the full tensor. We also assume that the time it takes to load a single element τ_f does not depend on the element or the number of elements loaded concurrently. Often, loading or computing the tensor via f is the dominant time

cost of the algorithm. In practice, f can be a function that reads data from a file, or evaluates a given function. In the former case, the assumption that f can be called independently from individual nodes may be false, as there are likely bottlenecks to loading data at some maximum bandwidth. At the beginning of the algorithm, if the tensor is stored completely in memory, then one can assume that $\tau_f = 0$. In this situation, our algorithm only has a constant (but not asymptotic) improvement in memory costs, and the fastest algorithm for computing the TT decomposition is parallelized SSTT.

Algorithm 12 outputs full sketches S_i for multiple values of i, and each of the sketches is stored as sub-sketches across all cores. As a consequence of the requirement that each sub-sketch is distributed, we must have that P is at least C^2. To see this, apply PSTT2 to a tensor with $n > C$. The algorithm performs a column sketch on the first unfolding, and a row sketch on the last unfolding. For the first unfolding, the sketch matrix is divided into sub-sketches according to only P_1, implying that $P_1 \geq C$ for each core to receive at least one sub-sketch. A similar argument for the last row sketch implies that $P_d \geq C$ as well, so $P \geq P_1 P_d \geq C^2$. As a result, we choose the partition to be concentrated only on P_1 and P_d in practice. Comparatively, if $n < C$, it is impossible to evenly distribute the sketch, so we extend the partition to include at least m dimensions until $n^m > C$. That is, we require $P_1 \cdots P_m \geq C$ and $P_d \cdots P_{d-m+1} \geq C$, so that the mth column and row sketches are distributed. When the tensor is stored fully in memory, one can simply assume that each core already holds P/C sub-tensors, and that $\tau_f = 0$ for the analysis in the following sections.

Due to the requirement that $P \geq C^2$, we make the additional assumption that C divides into both P_1 and P_d evenly (or, if $n < C$, C divides $P_1 \cdots P_m$ and

$P_d \cdots P_{d-m+1}$). This ensures that the algorithm is load-balanced, meaning each core is responsible for sketching P/C sub-tensors, and storing a factor of $1/C$ of the number of sub-sketches $S_{i,j}$. By default, we assume that the sub-sketches are distributed in "column-major" order, i.e., the cores each store sub-sketches $S_{i,j}$ that are consecutive in j. To contribute the sub-sketch to the core holding $S_{i,k}$, it is necessary to add an additional send/receive communication step. This algorithm is convenient for PSTT2 and PSTT2-onepass, as all sub-sketches can be computed with a single stream of a given sub-tensor.

Algorithm 12 Find multiple sketches of a tensor in parallel.

Input: Tensor oracle f, sketch dimensions $\mathbf{i}$, Khatri-Rao DRMs $\Psi_i^{(k)}$
Output: Sketches S_i for all $i \in \mathbf{i}$

```
Initialize S_{i,k} to zero for each i ∈ i, distributed among the available cores
parallel for 1 ≤ j ≤ P do
    Load Y_j into memory via f
    for i in i do
        Compute S_{i,k,j} via (3.11)
        Send S_{i,k,j} to owner of S_{i,k}
        for each S_{i,k,j'} received do
            Add S_{i,k,j'} to S_{i,k}
```

When $P = C$, the tensor is already loaded, and we only compute a single sketch (i.e., $\mathbf{i}$ has length 1), Algorithm 12 is equivalent to Algorithm 2 in [17] for computing the MTTKRP. The only difference is we assume that the factor matrices $\Psi^{(k)}$ are assessible by all cores. This assumption allows for one fewer communication step, and only sacrifices a factor of $\mathcal{O}(rn)$ in spatial complexity per core. These conditions apply in the algorithm for SSTT, as only a single sketch needs to be performed at a time. For PSTT2 and PSTT2-onepass, we require multiple sketches and $P \geq C^2$ for load balance, so the algorithms diverge. We illustrate the use of Algorithm 12 on a simple and small tensor in the next example.

Example. *Consider the tensor $\mathcal{X} \in \mathbb{R}^{2\times2\times2}$, and we wish to sketch the column space of the first unfolding ($\mathbf{i} = \{1\}$) with rank $r = 1$, a number of cores $C = 2$, and the partition $\mathbf{P} = \{C, 1, C\}$. Here, we explain with no oversampling, i.e., $p = 0$, but in practice we use $p = 2$. The entries of $\mathcal{X}$ can be written as*

$$\mathcal{X}(:,:,1) = \left[\begin{array}{cc} X_{111} & X_{121} \\ \hline X_{211} & X_{221} \end{array}\right], \qquad \mathcal{X}(:,:,2) = \left[\begin{array}{cc} X_{112} & X_{122} \\ \hline X_{212} & X_{222} \end{array}\right],$$

where we have put in lines to designate how the tensor is partitioned. We assign the first core to stream $j = 1$ ($X(1,:,1)$) and $j = 3$ ($X(1,:,2)$), and the second core to stream $j = 2$ ($X(2,:,1)$) and $j = 4$ ($X(2,:,2)$). We then multiply each tensor against the DRMs

$$\Psi_1^{(2)} = \left[\begin{array}{c} \Psi_{1,1}^{(2)} \\ \hline \Psi_{1,2}^{(2)} \end{array}\right], \qquad \Psi_1^{(3)} = \left[\begin{array}{c} \Psi_{1,1}^{(3)} \\ \hline \Psi_{1,2}^{(3)} \end{array}\right],$$

where include a line to emphasize that $\Psi^{(3)}$ is partitioned according to P_3. Each core is assumed to know the full matrices $\Psi^{(2)}$ and $\Psi^{(3)}$ at the start of the algorithm. Before the loop, core 1 initializes $S_{1,1} = [0]$ and core 3 initializes $S_{1,2} = [0]$.

In the first loop of the iteration, core 1 streams $j = 1$ and core 3 streams $j = 3$, where core 1 calculates $S_{1,1,1}$ and core 2 calculates $S_{1,2,2}$ by the multiplications

$$S_{1,1,1} = \begin{bmatrix} X_{111} & X_{121} \end{bmatrix} \begin{bmatrix} \Psi_{1,1}^{(3)}\Psi_{1,1}^{(2)} \\ \Psi_{1,1}^{(3)}\Psi_{1,2}^{(2)} \end{bmatrix}, \qquad S_{1,2,2} = \begin{bmatrix} X_{211} & X_{221} \end{bmatrix} \begin{bmatrix} \Psi_{1,1}^{(3)}\Psi_{1,1}^{(2)} \\ \Psi_{1,1}^{(3)}\Psi_{1,2}^{(2)} \end{bmatrix}.$$

Because core 1 owns $S_{1,1}$ and core 2 owns $S_{1,2}$, there is no communication at this step. Core 1 adds $S_{1,1,1}$ to $S_{1,1}$ and core 2 adds $S_{1,2,2}$ to $S_{1,2}$. In the next step, core 1 streams $j = 3$ and core 2 streams $j = 4$, and they perform the multiplications

$$S_{1,1,3} = \begin{bmatrix} X_{112} & X_{122} \end{bmatrix} \begin{bmatrix} \Psi_{1,2}^{(3)}\Psi_{1,1}^{(2)} \\ \Psi_{1,2}^{(3)}\Psi_{1,2}^{(2)} \end{bmatrix}, \quad S_{1,2,4} = \begin{bmatrix} X_{212} & X_{222} \end{bmatrix} \begin{bmatrix} \Psi_{1,2}^{(3)}\Psi_{1,1}^{(2)} \\ \Psi_{1,2}^{(3)}\Psi_{1,2}^{(2)} \end{bmatrix}.$$

Again, there is no communication to be done, so core 1 adds $S_{1,1,3}$ to $S_{1,1}$ and core 3 adds $S_{1,2,4}$ to $S_{1,2}$. This is the end of the algorithm.

One can confirm that the output is the same as the full unfolding multiplication:

$$\begin{bmatrix} S_{1,1} \\ S_{1,2} \end{bmatrix} = \left[\begin{array}{cc|cc} X_{111} & X_{121} & X_{112} & X_{122} \\ \hline X_{211} & X_{221} & X_{212} & X_{222} \end{array}\right] \left[\begin{array}{c} \Psi^{(3)}_{1,1}\Psi^{(2)}_{1,1} \\ \Psi^{(3)}_{1,1}\Psi^{(2)}_{1,2} \\ \hline \Psi^{(3)}_{1,2}\Psi^{(2)}_{1,1} \\ \Psi^{(3)}_{1,2}\Psi^{(2)}_{1,2} \end{array}\right].$$

Note that no communication is required for the first sketch since: (i) we assume every core knows every DRM in full, and (ii) core ℓ streams the sub-tensor "rows" that correspond to the sketch it owns, i.e., in the order $\mathcal{Y}_\ell, \mathcal{Y}_{\ell+C}, \ldots, \mathcal{Y}_{\ell+C(C-1)}$. As a result, there is no communication for column sketches. However, communication is still required for row sketches, so this fact primarily benefits SSTT.

In all three algorithms, the step after Algorithm 12 is to find an orthonormal basis for $S_{i,k}$, except for the middle sketch of PSTT2-onepass. For this purpose, we use the skinny QR algorithm in [29], which is fast and has a low memory overhead in comparison to the sketching step. These sketches are then converted to TT cores, and we assume that each processor has enough memory to store a full tensor train. Other steps for specific algorithms that need to be parallelized are:

- **SSTT**: The first step is to find an orthonormal basis for the column space using Algorithm 12 with $\mathbf{i} = 1$ and skinny QR, and use this basis as the first TT core $\mathcal{G}_1$. Because G_1 only has nr entries, it can be stored in each core to reduce the overall required communication. Then, we compute $Z = \mathcal{G}_1^T X_1$ and obtain sub-sketches of Z with Algorithm 12. These sketching and multiplication steps are repeated until all TT cores are obtained, and we distribute

all sub-sketches among the cores. Overall, the second pass of streaming takes a significant amount of time, and the largest storage contribution comes from storing the first calculated Z.

- **PSTT2**: To obtain all but the middle TT core, it is necessary to multiply orthonormal bases of column/row spaces against each other via (3.1), which can be easily rewritten in terms of sub-tensors. Since the orthonormal bases are much smaller than the tensor itself, communicating sub-sketches avoids high communication costs. In the end, the middle TT core is obtained via another streaming loop to perform (3.7), which adds significant computational time to the overall algorithm.
- **PSTT2-onepass**: All TT cores but the middle one are obtained as in PSTT2. Then, the middle TT core is computed with two matrix-matrix multiplications and a small least-squares problem. These are rewritten in terms of sub-sketches, and are cheap as they use already compressed data.

3.2.3 Memory Complexity

In this section, we analyze the memory complexity of our algorithms. We give estimates of the memory costs of a sub-tensor, a TT representation, and the three algorithms SSTT, PSTT2, and PSTT2-onepass. The asymptotic costs are then compared to measured total memory allocation per core from numerical experiments, showing PSTT2 and PSTT2-onepass have lower overall memory requirements, especially for high-dimensional tensors.

Throughout the section, we focus on trials using the Hilbert tensor, defined

d	3	5	9
$\mathbf{n}$	$960, 960, 960$	$96, 96, 96, 96, 96$	$12, 12, 12, 12, 12, 12, 12, 12, 12$
$\mathbf{r}$	$1, 25, 25, 1$	$1, 17, 18, 18, 17, 1$	$1, 12, 18, 18, 19, 19, 18, 18, 12, 1$
$\mathbf{P}$	$96, 1, 96$	$96, 1, 1, 1, 96$	$12, 6, 1, 1, 1, 1, 1, 6, 12$
tensor size	6.59 GB	60.8 GB	38.4 GB
TT size	4.94 MB	0.710 MB	0.197 MB
sub-tensor size	0.769 MB	6.79 MB	7.60 MB

Table 3.1: Dimensions and memory sizes of the three tensors used for profiling. The 'tensor size' line is calculated through the formula $8n^d/2^{30}$, and the 'TT size' and 'sub-tensor size' lines are measured via heap profiling. One can confirm that the sub-tensor sizes are nearly a factor of $1/P$ off from the tensor size, with discrepancies due to auxiliary information stored in the sub-tensor data structure. The ranks r are determined using asymptotic formulas from [193]. The choices of $\mathbf{P}$ align with our assumption of load balancing, so the sub-tensor sizes are equal for every core. A full TT is allocated on every core as well.

as

$$\mathcal{X}_{i_1,\ldots,i_d} = \frac{1}{1 - d + i_1 + \cdots + i_d}, \quad 1 \leq i_j \leq n_j, \quad 1 \leq j \leq d.$$

It is known that this tensor can be accurately approximated by a numerically low TT rank tensor, and the TT ranks can be estimated a priori [193]. Also, it is apparent that the total memory allocated does not depend on actual values of the tensor, but only on the dimension d, sizes $\mathbf{n}$, and ranks $\mathbf{r}$. Therefore, even though the Hilbert tensor is an artificial example, the memory results generalize to real-world tensors of similar sizes and ranks. We report numerical experiments for $d = 3$, $d = 5$, and $d = 9$ Hilbert tensors with sizes $\mathbf{n}$, ranks $\mathbf{r}$, and partitions $\mathbf{P}$ given in Table 3.1. The dimensions are chosen to represent a range of practical tensor dimensions. The partitions are chosen so that $P = 96^2$, i.e., they match the minimum value of P for PSTT2 and PSTT2-onepass discussed in section 3.2.2 for $C = 96$.

For each core, the number of stored entries of a sub-tensor and a TT format are

$$M_{\text{sub-tensor}} = n^d/P, \qquad M_{\text{TT}} = (d-2)r^2 n + 2rn,$$

where we use capital 'M's to denote all spatial costs. For the actual memory required, these quantities can be multiplied by the memory of a single entry. We note that neither of the above costs depend explicitly on the number of cores C, assuming $P > C$. The memory requirements for the three Hilbert examples are given in Table 3.1. We see that while the tensors have sizes in the gigabyte range, each of the cores only stores on the order of megabytes for the given values of $\mathbf{P}$. In addition, if the tensor is distributed in memory at the beginning of the algorithm, the amount of storage for the sub-tensor is $M_{\text{sub-tensor}} = n^d/C$, and this is always the dominant cost.

One can then express the memory costs of the individual algorithms as:

- **SSTT**: As mentioned in section 3.2.2, the predominant memory cost comes from storing the first calculation $Z = \mathcal{G}_1^T X_1$, which has rn^{d-1} entries. We distribute the entries among all cores, so the asymptotic memory cost of SSTT is

$$M_{\text{SSTT}} = \mathcal{O}(rn^{d-1}/C). \tag{3.12}$$

- **PSTT2**: The only major storage cost is to store the sketches. Each column sketch has a per-core memory cost of

$$M_{S_i} = \mathcal{O}(n^i r/C + (d-i)nr),$$

  where the two terms are the memory needed for S_i and the random Khatri-Rao DRMs. One needs to be careful that the above expression holds when $\mathbf{P}$ is large enough in the correct dimensions so that S_j can be split into C different sub-sketches. In practice, we choose $\mathbf{P}$ that the divisions are concentrated near P_1 and P_d as in Table 3.1. This allows all column and row sketches to be distributed among the cores. We choose the middle index $d_* = \lceil d/2 \rceil$ so that

the storage is balanced among row and column sketches, and thus the overall storage of PSTT2 is

$$M_{\text{PSTT2}} = \mathcal{O}\left(rn^{\lfloor d/2 \rfloor}/C + drn\right). \tag{3.13}$$

When $d > 4$ and $C \ll n$, the first term dominates the second, improving upon SSTT by a factor of $n^{\lceil d/2 \rceil - 1}$. When $d = 3$, the second term – the storage cost of the DRMs – exceeds that of the actual sketch matrices. Nevertheless, this complexity is better than that of SSTT, which depends on n^2.

- **PSTT2-onepass**: The storage cost of PSTT2-onepass is different from that of PSTT2 only because of the final middle sketch. Hence, the complexity is

$$M_{\text{PSTT2-onepass}} = \mathcal{O}\left(rn^{\lceil d/2 \rceil}/C\right). \tag{3.14}$$

When d is even, this spatial complexity is asymptotically the same as (3.13). When d is odd, however, the two complexities differ. This is most apparent for $d = 3$, when (3.14) matches (3.12), leading to neither algorithm being better in storage.

In practice, all our algorithms require some extra memory for the communicated quantities. However, this cost depends on $1/P$, which is by assumption less than $1/C$, and is thus absorbed in our asymptotic statements.

In Figure 3.1, we see the memory allocated per-core as a function of the total number of cores C for the three different Hilbert tensor TT computations, discounting the allocations of the sub-tensor and the TT. We discount these allocations because each algorithm requires the same allocation of memory for these components. The rest of the allocation is dominated by the size of the sketches themselves. For PSTT2, the memory allocated for sketches is comparable to the sub-tensor size in Table 3.1, but for SSTT and PSTT2-onepass the cost

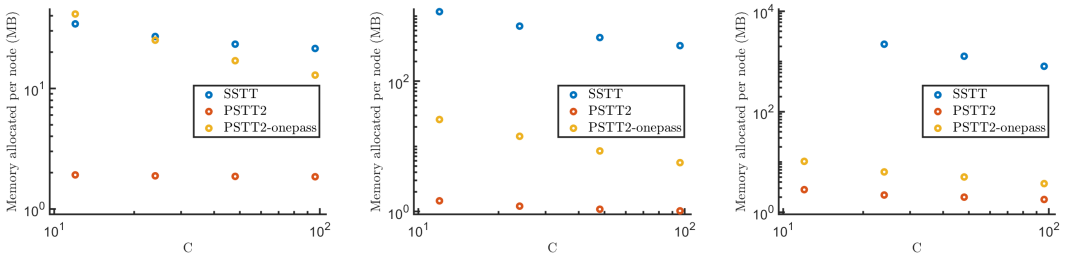

Figure 3.1: Measured memory allocated per-core for (left) $d = 3$, (middle) $d = 5$, and (right) $d = 9$ Hilbert tensors, discounting the memory allocated for sub-tensors and the TT. The right plot is missing a single data point for SSTT with $C = 12$ because the $d = 9$ tensor is too large to store in memory. See Table 3.1 for additional details of each experiment.

of sketching is larger in comparison. For each algorithm and each Hilbert tensor, we use $C = 12$, $C = 24$, $C = 48$, and $C = 96$. We see that PSTT2 reliably has the smallest memory overhead, and the improvement over SSTT increases as the dimension increases. This agrees with the asymptotic expectations of (3.13) and (3.12). We see that PSTT2-onepass also improves upon SSTT for all but the $d = 3$ trial, where the two asymptotic spatial complexities agree. Moreover, the $d = 9$, $C = 12$ SSTT trial data is missing due to an out-of-memory error. This occurs because the first sketch has $r_1 = n_1$, leading to no compression on the first step of SSTT. As a result, the first Z calculated needs to store the full tensor of 38.4 GB in memory, which is over the 32 GB limit of a single node.

As a function of C, the PSTT2 memory profiles are relatively flat due to the importance of storing sketch matrices. Specifically, PSTT2 storage costs are more influenced by the cost of storing DRMs and other similarly sized allocations, especially for small d. These costs are constant per-core, whereas the sub-sketch storage costs are distributed evenly across the cores as much as possible.

3.2.4 Time Complexity

The major cost of our algorithms is streaming via the function f, which we assume to be proportional to the size of the tensor. Because streaming is evenly split amongst the cores, the complexity to stream the tensor once is

$$T_{\text{stream}} = \tau_f n^d / C, \tag{3.15}$$

where capital 'T's are used to designate time complexities. From numerical experiments, we find that (3.15) takes a significant portion of the computation time in the $C = 1$ setting with no communication. We also want to remark that PSTT2-onepass has half the cost of streaming since it only streams the tensor once.

Even when the function evaluation is cheap, the time to multiply the DRMs against the sub-tensors can still be expensive. There are $d-1$ sketches for PSTT2, each with a time cost of $\mathcal{O}(rn^d/C)$, and this leads to a total sketching time of $\mathcal{O}(drn^d/C)$. PSTT2-onepass has a single more sketch to compute, but the complexity is the same as d becomes large. Comparatively, SSTT has a leading order complexity of $\mathcal{O}(rn^d/C)$, as only the first sketch and calculation of Z involve multiplications with the full tensor. For this reason, SSTT is expected be faster for high-dimensional tensors, at the expense of an increased storage cost.

For our experiments, we begin by comparing the single-core timing of the three presented algorithms to the standard TTSVD algorithm in TT-Toolbox written in MATLAB [161]. In the case of a single core with only a single partition ($P = C = 1$), SSTT exactly matches the algorithm presented in [41], where the authors observe speedups of around a factor of 10 for multiple large, low-rank tensors. To test that we have a similar speedup, we factor the $d = 3$ and

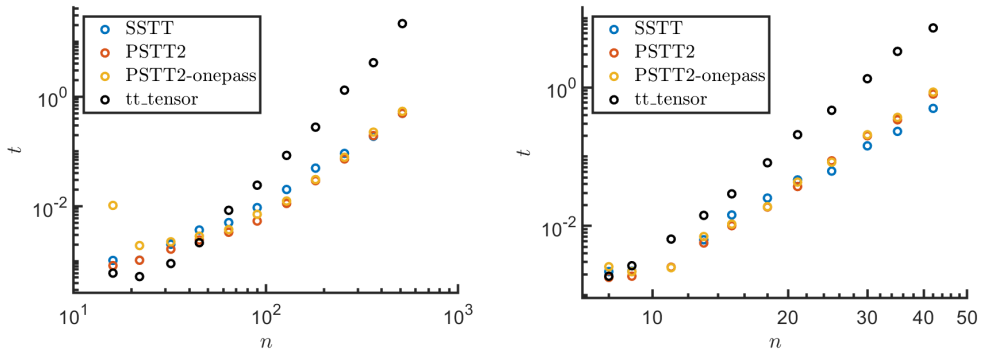

Figure 3.2: A timing comparison between the presented algorithms SSTT, PSTT2, and PSTT2-onepass to the TT-Toolbox package function `tt_tensor` [161] on the random tensor with $C = 1$ and $P = 1$. In each plot, the dimension d is held fixed with $d = 3$ (left) and $d = 5$ (right), and the tensor size n varies. The rank for $d = 3$ is held fixed at $r = 10$ and the rank for $d = 5$ is held at $r = 8$.

$d = 5$ tensors, both with the TT-Toolbox command `tt_tensor` and our implementations of SSTT, PSTT2, and PSTT2-onepass for varying values of n (see Figure 3.2). With a given rank, we construct a random tensor from its TT representation, which is consisted of i.i.d. standard Gaussian entries. For the $d = 3$ tensors, we choose $r = 10$, and for $d = 5$, we choose $r = 8$. For these tests, we omit the time of streaming for ease of comparison to [41], i.e., $\tau_f = 0$. The single-core experiments are performed on a personal computer with an Intel Core i7-7700 CPU with 3.60 GHz and 16 GB RAM.

In Figure 3.2, we see that all three of the sketching algorithms are faster than TT-Toolbox for large n, since the SVD steps in TT-Toolbox use a version of SVD that has a time complexity of $\mathcal{O}(n^{d+1})$ for the first unfolding. This leads to a speedup on the order of n/r for the sketching algorithms. For $d = 3$ and $n = 512$, SSTT is faster than tt_tensor by a factor of 43.5. For $d = 5$ and $n = 42$, SSTT is faster than tt_tensor by a factor of 14.5. For large n, all three sketching algorithms have about the same cost for $d = 3$, and for $d = 5$ SSTT is faster than PSTT2 and

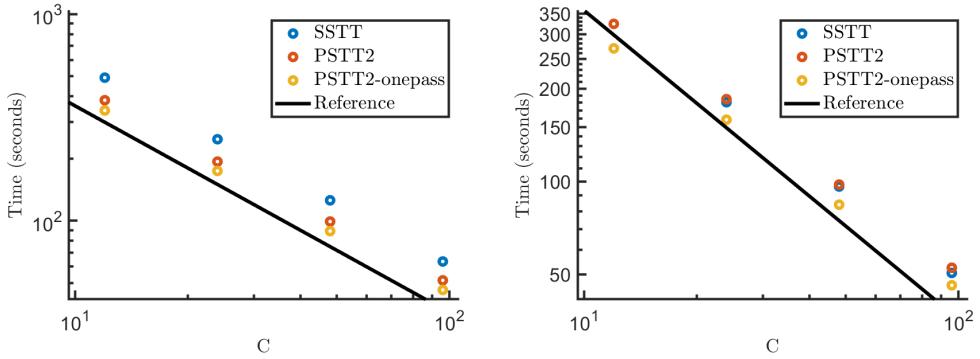

Figure 3.3: Strong scaling for the Hilbert tensor (left) $d = 5$, and (right) (b) $d = 9$. On each plot, a black line with slope -1 is shown for comparison (see Table 3.1 for details).

PSTT2-onepass by a factor of about 1.6.

Figure 3.3 shows the strong scaling timing results for the same $d = 5$ and $d = 9$ Hilbert tensors in the memory experiments (see Table 3.1). The slopes of the strong scaling times are near -1 for all algorithms and both values of d, i.e. the time nearly scales as $1/C$. This agrees with both (3.15) and the scaling of multiplying against the tensor. We see that PSTT2 and SSTT take comparable amounts of time for $d = 9$ and PSTT2 is moderately faster for $d = 5$. However, the speedup is only by a constant factor, and is difficult to predict. In comparison, PSTT2-onepass is significantly faster than the other two algorithms, due to the single streaming loop. For $d = 5$ and $C = 96$, PSTT2-onepass is faster than PSTT2 by a factor of 1.11 and faster than SSTT by a factor of 1.37. For $d = 9$, PSTT2-onepass is faster than PSTT2 by a factor of 1.10 and SSTT by 1.14.

3.2.5 Communication Cost

Now, we turn to a discussion of the communication cost. From inspection of Algorithm 12, we see that the multi-sketching step has dP sends and receives in the worst case scenario. However, since the core that calculates $S_{i,k,j}$ also stores $S_{i,k}$, not every communication is necessary. As noted in Example 3.2.2, the first sketch of SSTT can be performed with zero communication when the cores stream rows of the first unfolding. SSTT still requires communication for the QR decomposition, but the bandwidth cost for each core is only $\mathcal{O}(\log(C)r^2)$. For PSTT2 and PSTT2-onepass, this strategy of streaming rows of unfoldings can also eliminate the communication for column space sketches, but the cost of communicating row sketches remains. The worst case bandwidth cost is when every row sub-sketch obtained by a given core is communicated. Each core streams P/C sub-sketches, and each sub-sketch has a size of $\mathcal{O}(rn^{\lfloor d/2 \rfloor}/C)$ for PSTT2 and $\mathcal{O}(rn^{\lceil d/2 \rceil}/C)$ for PSTT2-onepass, leading to the worst case bandwidth costs

$$B_{\text{PSTT2}} = \mathcal{O}(rdn^{\lfloor d/2 \rfloor}P/C^2), \qquad B_{\text{PSTT2-onepass}} = \mathcal{O}(rdn^{\lceil d/2 \rceil}P/C^2). \tag{3.16}$$

In each of our examples in Table 3.1, we choose $P = C^2$ for $C = 96$, the largest considered value of C. The cost of communication is approximately proportional to the size of a sketch times the number of sketches. For trials with $C = 12$, (3.16) predicts a quadratic slowdown by a factor of 16, but this amount still leaves the bandwidth costs asymptotically much smaller than the costs of the matrix multiplications for large n. However, if we increase P, the cost of communication dominates, so (3.16) reveals a balance between memory efficiency and communication efficiency. As more messages are sent, there is an increased cost due to latency and synchronization as well. As a result, streaming

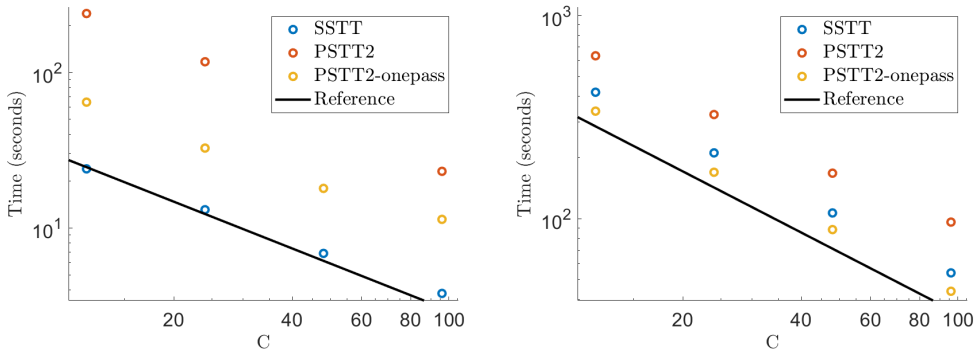

Figure 3.4: Strong scaling for the $d = 3$ Hilbert tensor (left) and $d = 3$ Gaussian bumps tensor (right). On each plot, a black line with slope -1 is shown for comparison (see Table 3.1 for details).

larger sub-tensors is generally preferable when there is sufficient space available in memory. The cost in (3.16) is pessimistic when compared to the bounds for MTTKRP [18], and we have not solved the problem on how to optimally distribute sub-tensors when one is interested in computing sketches of every unfolding.

Figure 3.4 (left) shows the $d = 3$ Hilbert tensor in Table 3.1 where PSTT2 and PSTT2-onepass are slower than SSTT due to high communication costs relative to the tensor size. While the bandwidth cost in (3.16) is still predicted to be small relative to the matrix multiplications, the latency and synchronization costs from communicating frequently cause a significant slowdown of the algorithm. However, we emphasize that these communications are still relatively unimportant when the evaluation of a tensor element is expensive, as can be seen in an experiment with a Gaussian mixture model, with the function defined as

$$\mathcal{X}_{i_1,i_2,i_3} = \sum_{j=1}^{N} e^{-\gamma\left((x_1-\xi_j)^2+(x_2-\eta_j)^2+(x_3-\zeta_j)^2\right)}, \qquad x_j = \frac{2i_j}{n_j} - 1,$$

where ξ_j, η_j, and ζ_j are the center coordinates of the jth Gaussian and γ controls the width of the Gaussians. Figure 3.4 (right) shows the result when $N = 100$,

$\gamma = 10$, and the centers are uniformly random numbers between -1 and 1. Because each calculation of a tensor entry takes the evaluation of N Gaussians, the evaluation is significantly longer than that of a Hilbert tensor. Although PSTT2 is still slower than SSTT, PSTT2-onepass becomes the fastest option since avoiding the second set of function evaluations gains more than the added communications.

3.3 Solve Sylvester tensor equations in TT format

Many tensors in practice are known implicitly as solutions of tensor equations. For example, the discretized solution of a multivariable PDE might satisfy a tensor equation.

In this section, we focus on computing an approximation $\tilde{\mathcal{X}}$ in TT format of a tensor $\mathcal{X} \in \mathbb{C}^{n_1 \times \cdots \times n_d}$ that satisfies the Sylvester tensor equation in (1.13). This type of algebraic relation appears when discretizing Poisson's equation of a tensor-product domain with either a finite difference scheme [134] or a spectral method [193]. We describe an algorithm that solves the 3D Sylvester tensor equation of the form

$$\begin{gathered} \mathcal{X} \times_1 A + \mathcal{X} \times_2 B + \mathcal{X} \times_3 C = \mathcal{F}, \quad \mathcal{F} \in \mathbb{C}^{n_1 \times n_2 \times n_3}, \\ A, B, C \text{ normal}, \quad A \in \mathbb{C}^{n_1 \times n_1}, \quad B \in \mathbb{C}^{n_2 \times n_2}, \quad C \in \mathbb{C}^{n_3 \times n_3}, \end{gathered} \tag{3.17}$$

and extensions of this algorithm to solve d dimensional Sylvester tensor equations are described at the end of this section. To ensure a unique solution of (3.17), we assume that $\lambda_i(A) + \lambda_j(B) + \lambda_k(C) \neq 0$, for $1 \leq i \leq n_1$, $1 \leq j \leq n_2$, and $1 \leq k \leq n_3$, where $\lambda_i(A)$ is an eigenvalue of A [195]. We further suppose that $\mathcal{F}$ is given in the TT format with TT rank $(1, r_1, r_2, 1)$ and cores $\mathcal{G}_1, \mathcal{G}_2$, and

$\mathcal{G}_3$. Our goal is to compute an approximate solution $\tilde{\mathcal{X}}$ to (3.17) in the TT format. Since Sylvester equations can be converted to a linear system via vectorizing $\mathcal{X}$ and $\mathcal{F}$, our proposed solver is one way to solve a specific type of high dimensional linear system where the right-hand-side is provided in the TT format. For algorithms to solve such general linear systems, we refer the reader to alternating minimal energy methods [62] and TT-GMRES [61].

Since TT cores of a tensor are analogues of factor matrices in a matrix decomposition, our key idea is to convert (3.17) into several Sylvester matrix equations and use fADI. If we reshape $\mathcal{X}$ and $\mathcal{F}$ to their 1st unfolding, respectively, then (3.17) becomes

$$AX_1 + X_1(I \otimes B + C \otimes I)^T = F_1 = (G_1)_1(G_2)_1((G_3)_2 \otimes I), \tag{3.18}$$

where $(G_1)_1$ is the 1st unfolding of $\mathcal{G}_1$, $(G_2)_1$ is the 1st unfolding of $\mathcal{G}_2$, and $(G_3)_2$ is the 2nd unfolding of $\mathcal{G}_3$. Similarly, we get another Sylvester matrix equation by reshaping $\mathcal{X}$ and $\mathcal{F}$ to their 2nd unfolding:

$$(I \otimes A + B \otimes I)X_2 + X_2C^T = F_2 = (I \otimes (G_1)_1)(G_2)_2(G_3)_2, \tag{3.19}$$

where $(G_2)_2$ is the 2nd unfolding of $\mathcal{G}_2$.

Algorithm 7 in section 2.4.3 (ST21) describes a way to compute the solution of (3.17) in TT format. It starts by computing the first TT core U_1 as an orthonormal basis of the column space of X_1 from (3.18), and then finds the other two TT cores via solving:

$$(I \otimes \tilde{A} + B \otimes I)Y + YC^T = (I \otimes \tilde{G}_1)(G_2)_2(G_3)_2, \tag{3.20}$$

where $\tilde{A} = U_1^* A U_1$, and $\tilde{G}_1 = U_1^*(G_1)_1$. (3.20) is similar to (3.19) but (3.20) has two disadvantages: (1) Parallelism when finding TT cores is not exploited, as

the two fADI loops are carried out in an order, (2) $\tilde{A}$ is a dense matrix, which may have a higher cost of solving shifted linear systems than that of A. In this way, we want an algorithm that, in certain scenarios, can solve (3.18) and (3.19) simultaneously and only solve shifted linear systems with A, B, and C.

With fADI, we can obtain U_1 and U_2, orthonormal bases of the column space of X_1 and X_2, from (3.18) and (3.19) independently. In the meantime, solving (3.19) yields the solution $X_2 = U_2 DY^*$ and we can use DY^* as the third TT core of the solution $\mathcal{X}$. As a result, Theorem 3.1.1 allows us to compute all three TT cores with only one extra matrix-matrix multiplication.

In general, we need two sets of shift parameters to solve (3.18) and (3.19). The Zolotarev number associated with (3.18) is

$$Z_k(\Lambda(A), \Lambda(-B) + \Lambda(-C)) := \inf_{r \in \mathcal{R}_{k,k}} \frac{\sup_{z \in \Lambda(A)} |r(z)|}{\inf_{z \in \Lambda(-B)+\Lambda(-C)} |r(z)|}, \qquad k \geq 0, \quad (3.21)$$

where '+' is the Minkowski sum of two sets. Similarly, the Zolotarev number corresponded to (3.19) is

$$Z_k(\Lambda(A) + \Lambda(B), \Lambda(-C)) := \inf_{r \in \mathcal{R}_{k,k}} \frac{\sup_{z \in \Lambda(A)+\Lambda(B)} |r(z)|}{\inf_{z \in \Lambda(-C)} |r(z)|}, \qquad k \geq 0. \quad (3.22)$$

We find that the number

$$L_k(\Lambda(A), \Lambda(B), \Lambda(C)) := \inf_{r \in \mathcal{R}_{k,k}} \frac{\sup_{z \in \Lambda(A) \cup [\Lambda(A)+\Lambda(B)]} |r(z)|}{\inf_{z \in \Lambda(-C) \cup [\Lambda(-B)+\Lambda(-C)]} |r(z)|}, \qquad k \geq 0,$$

is an upper bound for both (3.21) and (3.22), so that bounds in (1.16) can be acquired for both X_1 and X_2:

$$\|X_i - (X_i)_k\|_F \leq L_k(\Lambda(A), \Lambda(B), \Lambda(C)) \|\mathcal{X}\|_F, \qquad i = 1, 2.$$

Therefore, we can choose the same shift parameters when we use fADI on (3.18) and (3.19), which means U_1 and U_2 can be found in a single set of iterations.

Since we do not need U_2 in a low-rank format, we can recover U_2 column-by-column in a way introduced in ST21 by using ADI (Algorithm 5) on reshapes of the columns. We summarize the 3D Sylvester equation solver in Algorithm 13.

Algorithm 13 TT-fADI: Given a 3D Sylvester tensor equation (3.17), compute an approximate solution in TT format with three cores computed almost-simultaneously.

Input: Matrices A, B, and C, TT cores $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$ and TT ranks r_1 and r_2 of $\mathcal{F}$, and desired accuracy $0 < \epsilon < 1$

Output: TT cores $\mathcal{H}_1, \mathcal{H}_2$, and $\mathcal{H}_3$ of an approximate solution $\tilde{\mathcal{X}}$

1: Use the spectra of A, B, and C to find shift parameter arrays $\boldsymbol{p}$ and $\boldsymbol{q}$ of length ℓ.
2: Solve $(A - q_1 I_{n_1})Z_1 = (G_1)_1$. Let $Z = Z_1$.
3: Solve $((I_{n_2} \otimes A + B \otimes I_{n_1}) - q_1 I_{n_1 n_2})W_1 = (I_{n_2} \otimes (G_1)_1)(G_2)_2$. Let $W = W_1$.
4: Solve $(-C - \overline{p_1} I_{n_3})Y_1 = (G_3)_2^T$. Let $Y = Y_1$.
5: Let $D = (q_1 - p_1)I_{r_2}$.
6: **for** $1 \le j \le \ell - 1$ **do**
7: Set $R_j = (q_{j+1} - p_j)Z_j$, $U_j = (q_{j+1} - p_j)W_j$, and $V_j = (\overline{p_{j+1}} - \overline{q_j})Y_j$.
8: Solve $(A - q_{j+1} I_{n_1})Z_{j+1} = R_j$. Set $Z_{j+1} = Z_{j+1} + Z_j$ and $Z = \begin{bmatrix} Z & Z_{j+1} \end{bmatrix}$.
9: Solve $((I_{n_2} \otimes A + B \otimes I_{n_1}) - q_{j+1} I_{n_1 n_2})W_{j+1} = U_j$. Set $W_{j+1} = W_{j+1} + W_j$ and $W = \begin{bmatrix} W & W_{j+1} \end{bmatrix}$.
10: Solve $(-C - \overline{p_{j+1}} I_{n_3})Y_{j+1} = V_j$. Set $Y_{j+1} = Y_{j+1} + Y_j$ and $Y = \begin{bmatrix} Y & Y_{j+1} \end{bmatrix}$.
11: Set $D = \begin{bmatrix} D & \\ & (q_{j+1} - p_{j+1})I_{r_2} \end{bmatrix}$.
12: Recompress W, D, and Y to get $\|\tilde{W}\tilde{D}\tilde{Y}^* - WDY^*\| \le \epsilon \|WDY^*\|$.
13: Set $W = \tilde{W}, D = \tilde{D}$, and $Y = \tilde{Y}$, and s_2 to be the rank.
14: Compute a CPQR of Z to obtain U_1 with orthonormal cols and set $U_1 = U_1(:, 1\!:\!s_1)$ if $U_1(s_1 + 1, s_1 + 1) \le \epsilon$.
15: Calculate $T = U_1^* \mathrm{reshape}(W, n_1, n_2 s_2)$.
16: Set $\mathcal{H}_1 = U_1$, $\mathcal{H}_2 = \mathrm{reshape}(T, s_1, n_2, s_2)$, and $\mathcal{H}_3 = DY^*$.

We demonstrate Algorithm 13 with a simple example. Consider the equation

$$\mathcal{X} \times_1 A + \mathcal{X} \times_2 A + \mathcal{X} \times_3 A = \mathcal{F}, \quad A \in \mathbb{R}^{n \times n},\ \mathcal{F} \in \mathbb{R}^{n \times n \times n}, \tag{3.23}$$

where A is a diagonal matrix with diagonal elements $a_j \in [-1, -1/(30n)]$ for $1 \le j \le n$, and $\mathcal{F}$ has TT rank $(1, \lfloor n/4 \rfloor, 2, 1)$ with all three TT cores consisting of i.i.d. uniform random numbers in $(0, 1)$. Figure 3.5 shows the running time of

three Sylvester equation solvers. The green line represents the algorithm ST21. The blue line represents a direct solver, which computes each element of $\mathcal{X}$ by $\mathcal{X}_{i,j,k} = \mathcal{F}_{i,j,k}/(a_i + a_j + a_k)$ for $1 \leq i, j, k \leq n$, and performs TT decomposition on $\mathcal{X}$. This algorithm has complexity $\mathcal{O}(n^3)$. The red line represents Algorithm 13. We can see when $n \geq 100$, Algorithm 13 is the fastest. With $n = 350$, Algorithm 13 is 4 times faster than ST21, and almost 6 times faster than the direct solver. The performance of ST21 is affected since s_1, the size of the first TT core of the solution $\mathcal{X}$, can be close to n despite the fact that $s_1 = \mathcal{O}(\log n)$ [193]. Therefore, the cost of solving shifted linear systems of $\tilde{A}$ in (3.20) is significantly higher than that of A.

It is straightforward to extend Algorithm 13 to solve d dimensional Sylvester tensor equations (1.13). Since Algorithm 13 can be rewritten into $d-1$ Sylvester matrix equations for each unfolding of $\mathcal{X}$, we can find a universal set of shift parameters by optimizing the number

$$L_k(\Lambda(A^{(1)}), \ldots, \Lambda(A^{(d)})) := \inf_{r \in \mathcal{R}_{k,k}} \frac{\sup_{z \in \bigcup_{j=1}^{d-1} [\sum_{\ell=1}^{j} \Lambda(A^{(\ell)})]} |r(z)|}{\inf_{z \in \bigcup_{j=2}^{d} [\sum_{\ell=j}^{d} \Lambda(-A^{(\ell)})]} |r(z)|}, \qquad k \geq 0,$$

where the summation is Minkowski sum of sets. Then, we can use fADI to solve for orthonormal bases $U_1, \ldots, U_{d-1}$ for column spaces of unfoldings of $\mathcal{X}$ and the final TT core. In particular, for $2 \leq j \leq d-1$, the reshape of each column of U_j satisfies a j-dimensional Sylvester equation with a full-rank right-hand-side tensor so we can use existing solvers to recover the columns. We then can perform $d-2$ multiplications to obtain all the TT cores with Theorem 3.1.1. Furthermore, to avoid solving high-dimensional Sylvester equations for columns of U_j, we can incorporate the two-sided idea in section 3.1.2 to solve for both orthonormal bases for column and row spaces of unfoldings of $\mathcal{X}$. This helps reduce the computation costs in practice.

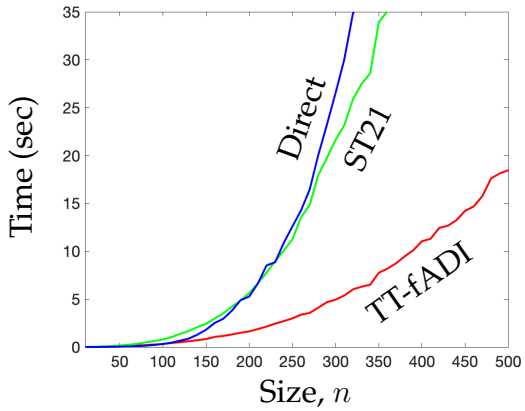

Figure 3.5: The execution time of direct solver (blue), ST21 (green), and Algorithm 13 (red) to solve (3.23) with size of the problem $10 \leq n \leq 500$.

CHAPTER 4

SPECTRAL, TENSOR AND DOMAIN DECOMPOSITION METHODS FOR FRACTIONAL PDES

In this chapter[1], we develop new spectral solvers for ractional partial differential equations (PDEs) on simple geometries. Fractional PDEs have recently received a tremendous amount of attention, which can be attributed to the flexibility of fractional operators in capturing long-range effects, due to their nonlocal nature. In addition, they have fewer regularity requirements than their classical counterparts. In particular, the fractional Laplacian has been successfully used as a regularizer in imaging science in place of the total variation regularization [7,8]. Moreover, the fractional Helmholtz equation was derived in [225], using first principle arguments combined with a constitutive relation, to model geophysical electromagnetism. Other applications include: Quasi-geostrophic flow [47], phase field models [4,7], porous media [55], and quantum mechanics [130].

4.1 Introduction

Motivated by these applications, we introduce a new approach to solve the fractional PDE

$$\begin{aligned} (-\Delta)^s u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{4.1}$$

where $\Omega \subset \mathbb{R}^n$ is a bounded open domain with boundary $\partial\Omega$, and $s \in (0,1)$ is the fractional exponent. The operator $(-\Delta)^s$ denotes the spectral fractional Lapla-

[1]This chapter is based on a paper with Harbir Antil and Drew Kouri [191]. I derived the discretization, developed the algorithms, ran the numerical experiments, and was the lead author in writing the paper.

cian, whose rigorous definition will be provided in Section 4.1.1. The nonlocality of the fractional Laplacian makes it challenging to realize in practice [37,201]. However, several approaches exist. For example, the authors in [199] use a spectral discretization in space and discuss computing the spectrum of the Laplacian to realize the fractional Laplacian. Unfornately, computing the spectrum of an operator in general domains can be expensive. Whereas, the authors in [32] discuss an alternative approach based on the so-called Kato formula. Here, spatial discretization is carried out using the finite element method. For the standard problem such as (4.1), the Kato formula approach is desirable, but is delicate, especially when lower-order terms are present [225].

Another popular approach is based on the so-called Caffarelli–Silvestre extension. This extension was originally introduced by Caffarelli and Silvestre in [37] for the case of unbounded Ω and was extended to bounded domains by Stinga and Torrea in [201]. The main idea is to rewrite the nonlocal problem (4.1) as a local problem in one additional spatial dimension (i.e., the domain of the resulting local problem is $\Omega \times (0, \infty)$ with dimension $n + 1$). This idea was exploited in [156], where the authors introduced a finite element method (FEM) to solve the extended problem on a bounded domain $\Omega \times (0, R)$ with $R < \infty$. The choice of R in [156] is motivated by the fact that the solution in the extended direction decays exponentially. Recently, the authors in [19, 149] introduced hp-FEM that reduces the complexity of the $(n+1)$-dimensional problem to *log-linear* with respect to the number of degrees of freedom in Ω. We also mention that an interesting hybrid FEM-spectral method based on approximation of the Laplacian eigenvalues was considered in [3].

An additional issue with the Caffarelli–Silvestre extension is that the solu-

tion in the extended dimension suffers from low regularity when $s \neq 1/2$. As a result, spectral methods do not provide the typical exponential convergence. This has been thoroughly investigated in [42], where the authors use generalized Laguerre functions as a basis in the extended dimension to solve a weak formulation of (4.1). To overcome this regularity issue and improve the convergence rate, the authors apply an enrichment technique for spectral methods. The convergence rate improves, as the enrichment terms are included, but the matrices for the discretized problem are generally dense. In addition, the system becomes ill-conditioned, and if there are too many singular terms included, the convergence rate can deteriorate significantly due to ill-conditioning. Finally, we mention extensive work on numerical methods for fractional PDEs with different fractional derivatives such as Riemann Liouville. See, for example [233], and the review article [137].

The goal of this paper is to introduce a new efficient and robust spectral method to solve fractional PDEs based on the Caffarelli–Silvestre extension. For simplicity, we present our method for $n = 2$ on two types of domains: disks and rectangles. By utilizing a tensor equation solver, our method is easily generalized to even higher dimensions. For example, as shown in our numerical examples in Section 4.4, our method directly applies to $n = 3$ on a cube. In addition, with spherical polar coordinates, our method can solve the problem on a ball. Our spectral method utilizes ultraspherical, Chebyshev, and Fourier polynomial discretizations. The resulting discrete systems are tensor equations, which we solve with a direct solver based on the real Schur decomposition. For the disk domain, we rewrite the extended problem in polar coordinates and use the Double Fourier Sphere (DFS) method [230] to overcome the singular behavior when the extended dimension $z = 0$. Using DFS, $z = 0$ is no longer treated

as a boundary. Afterward, we consider two cases: (i) $z^{1/s}$ can be approximated well by a polynomial; (ii) $z^{1/s}$ cannot be approximated well by a polynomial. The former case occurs, for example, when $1/s$ is an integer. In the latter case, we employ piecewise polynomials over subdomains. In this manuscript, we focus on fewer subdomains with higher polynomial degrees for simpler systems to solve. One can alternatively choose to use more subdomains with lower polynomial degrees [13]. For our numerical results, we use `chebfun` [63] to automatically generate the subdomains. We then solve the extended PDE directly using ultraspherical, Fourier, and Chebyshev spectral discretizations for the radius, angle, and extended direction. For rectangular domains, we spectrally discretize the extended PDE with ultraspherical polynomials in the space domain and Chebyshev polynomials for the extended direction. In addition, when $z^{1/s}$ is approximated by a piecewise polynomial, we design a domain decomposition solver that handles each piece independently. As a result, this solver is easily parallelized. The convergence analysis for ultraspherical spectral methods can be found in [159]. In practice, we use the polynomial coefficients of the solution to determine if the solver has converged. Specifically, when a coefficient falls below a given threshold (e.g., we choose 10^{-10} in this paper), we terminate the spectral method and use the result as the discrete approximation of the solution to the fractional PDE.

In all numerical examples and for all values of s, we observe exponential convergence with just a few degrees of freedom. In this sense, the potential benefits of our method are clear. Additionally, the proposed approach has several other advantages over the existing spectral methods. We can combine our spectral methods on rectangles with the ultraspherical spectral element method (UltraSEM) [73] to develop solvers on polygonal domains with unstructured

quadrilateral or triangular meshes. To be specific, we can partition the general domain Ω into quadrilateral subdomains, convert them into rectangles with a change of variables, and apply our solver. As a result, the total number of degrees of freedom is the sum of those used on each subdomain. Finally, when $n = 2$, we emphasize that our method can be generalized to solve fractional PDEs of the form:

$$\begin{aligned} \mathcal{L}^s u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{4.2}$$

where the operator $\mathcal{L}$ is the general elliptic operator $\mathcal{L}u = -\nabla \cdot (A\nabla u) + cu$. Here, $A \in \mathbb{R}^{2\times 2}$ is matrix function that is symmetric and positive definite and $0 \leq c \in \mathbb{R}$ is a function. This is feasible due to [73]. However, [73] only considers the case with $n = 2$, motivating our restriction to $n = 2$ in (4.2).

The remainder of this chapter is organized as follows. We first review preliminary results on fractional PDEs (see Section 4.1.1) as well as the polynomial notation (see Section 4.1.2) that we use. In Section 4.2, we introduce the direct solver for the disk domain. In Section 4.3, we solve the extended PDE on rectangles both directly and with a parallel domain-decomposition solver. Finally, in Section 4.4, we demonstrate our method on two applications: solving the fractional elliptic PDE on the cube, and solving a fractional PDE-constrained optimal control problem.

4.1.1 Preliminary results

Let $\Omega \subset \mathbb{R}^n$ be a bounded open set with Lipschitz boundary $\partial\Omega$. Let $-\Delta$ be the $L^2(\Omega)$ realization of the standard Laplace operator with zero Dirichlet boundary conditions. It then follows that $-\Delta$ has compact resolvent and its eigenvalues

can be arranged as $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k \leq \cdots$ with $\lim_{k\to\infty} \lambda_k = \infty$. Let us denote by $\varphi_k \in H_0^1(\Omega)$ the eigenfunctions corresponding to λ_k. These eigenfunctions form an orthonormal basis of $L^2(\Omega)$.

For $s \geq 0$, we let

$$\mathbb{H}^s(\Omega) := \left\{ u = \sum_{k=1}^{\infty} u_k \varphi_k \;\middle|\; \|u\|^2_{\mathbb{H}^s(\Omega)} := \sum_{k=1}^{\infty} \lambda_k^s u_k^2 < \infty, \;\text{ where }\; u_k = \int_\Omega u \varphi_k dx \right\}.$$

For a relation between $\mathbb{H}^s(\Omega)$ and the classical fractional-order Sobolev space $H^s(\Omega)$, we refer to [9,57], and the references therein. We shall denote the dual of $\mathbb{H}^s(\Omega)$ by $\mathbb{H}^{-s}(\Omega)$. Specifically in this manuscript, we focus on $0 < s < 1$. Now, to define the fractional Laplacian: $(-\Delta)^s$ is the mapping

$$(-\Delta)^s : \mathbb{H}^s(\Omega) \to \mathbb{H}^{-s}(\Omega)$$

defined for all $u \in \mathbb{H}^s(\Omega)$ by

$$(-\Delta)^s u = \sum_{k=1}^{\infty} \lambda_k^s u_k \varphi_k.$$

To solve (4.1), we use the Caffarelli–Silvestre extension [37, 201]. This requires introducing an extension variable ζ, and the change of variable $z = \left(\frac{\zeta}{2s}\right)^{2s}$. Then, the resulting problem aims to find $U : \Omega \times [0,\infty) \to \mathbb{R}$ that satisfies the following equation

$$\begin{aligned} \Delta_x U + z^\alpha U_{zz} &= 0 && \text{in } \Omega \times (0,\infty), \\ U(x,z) &= 0 && \text{on } \partial\Omega \times (0,\infty), \\ \partial_\nu U(x,0) &= d_s f(x) && \text{on } \Omega \times \{0\}, \end{aligned} \tag{4.3}$$

where $\alpha = 2 - \frac{1}{s}$, and $d_s = s^{2s-1}\frac{\Gamma(1-s)}{\Gamma(s)}$. Here, Δ_x denotes the Laplacian with respect to the original domain Ω and U_{zz} denotes the second derivative with respect to the extended dimension z. After solving for U, we can recover the solution to (4.1) as $u(x) = U(x,0)$.

In general, approximating functions by polynomials on an unbounded domain is a challenging problem. Motivated by the fact that the solution U in the z-direction decays exponentially [156] (also confirmed by our numerical experiments), we consider the following truncated problem:

$$\begin{aligned} z^{(1/s)}\Delta_x U + z^2 U_{zz} &= 0 && \text{in } \Omega \times (0,R), \\ U(x,z) &= 0 && \text{on } \partial\Omega \times (0,R), \\ U(x,R) &= 0 && \text{on } \Omega \times \{R\}, \\ \partial_\nu U(x,0) &= d_s f(x) && \text{on } \Omega \times \{0\}, \end{aligned} \tag{4.4}$$

where $R > 0$ is the truncation parameter. In our numerical experiments, the choice of R is motivated by [156]. In particular, we set $R = \mathcal{O}(\log(\mathrm{DoF}_\Omega))$ for the rectangular domains, where DoF_Ω is the total degrees of freedom used for Ω. Experimentally, we notice that for the disc domain it is more appropriate to choose $R = \mathcal{O}(\mathrm{DoF}_\Omega^{1/3})$. We emphasize that the additional variable z introduced by the extension requires that we solve a problem of one dimension higher. In particular, although Ω is chosen to be rectangles or disks in this paper, we must solve (4.4) in hexahedron or cylinders.

4.1.2 Ultraspherical Polynomial Basis and Spectral Methods

Ultraspherical (or Gegenbauer) polynomials are a special family of polynomials that are usually denoted by $C_n^{(\lambda)}(x)$. Here, $\lambda > 0$ is a coefficient and n is the polynomial degree of x [158, Table 18.3.1]. They are orthogonal on the interval $(-1,1)$ with respect to the weight function $w(x) = (1-x^2)^{\lambda-1/2}$ and satisfy the

three-term recurrence [158, Table 18.9.1]

$$\begin{aligned} C_0^{(\lambda)}(x) &= 1, \\ C_1^{(\lambda)}(x) &= 2\lambda x, \\ C_{n+1}^{(\lambda)}(x) &= \frac{2(n+\lambda)}{n+1} x C_n^{(\lambda)}(x) - \frac{n+2\lambda-1}{n+1} C_{n-1}^{(\lambda)}(x). \end{aligned} \tag{4.5}$$

For notational convenience, we use $\tilde{C}_n^{(\lambda)}(x)$ to denote the $L^2(-1,1)$-normalized ultraspherical polynomials with respect to the weight $w(x)$. Two well-known classes of polynomials, Chebyshev polynomials of the second kind and Legendre polynomials, are both special cases of ultraspherical polynomials, with coefficient $\lambda = 1$ and $\lambda = 1/2$, respectively.

4.2 Spectral Discretization for Fractional PDEs on a Disk

In this section, we solve (4.4) on a disk domain. Without loss of generality, we let Ω be the unit circle. Otherwise, we can easily convert to this problem by scaling with the radius. We use polar coordinates to rewrite (4.4) as

$$\begin{aligned} z^{1/s}\left(U_{\rho\rho} + \frac{1}{\rho}U_\rho + \frac{1}{\rho^2}U_{\theta\theta}\right) + z^2 U_{zz} &= 0 && \text{in } [0,1)\times[-\pi,\pi]\times(0,R), \\ U(1,\theta,z) &= 0 && \text{on } [-\pi,\pi]\times(0,R), \\ U(\rho,\theta,R) &= 0 && \text{on } [0,1)\times[-\pi,\pi], \\ \partial_\nu U(\rho,\theta,0) &= d_s f(\rho,\theta) && \text{on } [0,1)\times[-\pi,\pi]. \end{aligned} \tag{4.6}$$

To avoid the singularity at $\rho = 0$, we use the DFS method [230] to extend to $\rho \in (-1, 1)$ by setting

$$\tilde{U}(\rho,\theta,z) = \begin{cases} U(\rho,\theta,z) & (\rho,\theta,z) \in [0,1) \times [-\pi,\pi] \times (0,R) \\ U(-\rho,\theta+\pi,z) & (\rho,\theta,z) \in (-1,0] \times [-\pi,\pi] \times (0,R) \end{cases},$$

$$\tilde{f}(\rho,\theta) = \begin{cases} f(\rho,\theta) & (\rho,\theta) \in [0,1) \times [-\pi,\pi] \\ f(-\rho,\theta+\pi) & (\rho,\theta) \in (-1,0] \times [-\pi,\pi] \end{cases}.$$

Notice that both $\tilde{U}$ and $\tilde{f}$ are now continuous at $\rho = 0$, which leads to simpler spectral discretization with smaller polynomial degrees. From this point, we work directly with these "doubled" functions so that the singularity at $\rho = 0$ does not require additional consideration. The disk domain allows us to assume that both the solution $\tilde{U}$ and the function $\tilde{f}$ have Fourier expansions:

$$\tilde{U}(\rho,\theta,z) \approx \sum_{k=-m/2}^{m/2-1} \tilde{U}_k(\rho,z) e^{ik\theta} \qquad \text{and} \qquad \tilde{f}(\rho,\theta) \approx \sum_{k=-m/2}^{m/2-1} \tilde{f}_k(\rho) e^{ik\theta}.$$

In this way, we can decouple (4.6) into differential equations for each Fourier mode:

$$\begin{aligned} z^{1/s}\left((\tilde{U}_k)_{\rho\rho} + \frac{1}{\rho}(\tilde{U}_k)_\rho - \frac{k^2}{\rho^2}\tilde{U}_k\right) + z^2(\tilde{U}_k)_{zz} = 0 \qquad & \text{in } (-1,1)\times(0,R), \\ \tilde{U}_k(\pm 1, z) = 0 \qquad & \text{on } (0,R), \\ \tilde{U}_k(\rho, R) = 0 \qquad & \text{on } (-1,1), \\ \partial_\nu \tilde{U}_k(\rho,0) = d_s \tilde{f}_k(\rho) \quad & \text{on } (-1,1). \end{aligned} \tag{4.7}$$

Following [74, § 4.1.2], we assume that the ansatz for $\tilde{U}_k$ is given by

$$\tilde{U}_k(\rho,z) = (1-\rho^2)\rho^{\min(|k|,2)}\tilde{V}_k(\rho,z), \tag{4.8}$$

where the term $(1-\rho^2)$ incorporates the boundary condition $\tilde{U}_k(\pm 1, z) = 0$. Then, we solve for $\tilde{V}_k$. The choice of the above ansatz only imposes partial

regularity on $\tilde{U}_k$, and we refer to [74, § 4.1.2] for a detailed discussion on why it is challenging to impose full regularity.

Since the fractional exponent s can be any number between 0 and 1, the function $z^{1/s}$ makes the development of solvers for (4.7) a challenging task for our spectral method. To overcome this difficulty, we develop different solvers for varied values of s.

4.2.1 Polynomial Approximation of $z^{1/s}$

We first consider the case in which $s \in (0, 1)$ is such that the function $z \mapsto z^{(1/s)}$ can be approximated accurately by a polynomial of low degree. For the discretized problem, the multiplication by $z^{(1/s)}$ is transformed into a matrix-matrix product. Consequently, the polynomial degree is directly related to the discretization size, and the definition of "low degree" in the above statement is related to the size of the discretized system one is capable of solving. For example, one may interpret "low degree" to mean "degree less than 50". In this case, any value of s such that $1/s$ is an integer between 1 and 50 falls into this class.

To make the spectral discretization easier, we make the change-of-variables

$w = \frac{2}{R}z - 1 \in (-1, 1)$. In this way, the PDE in (4.7) becomes:

$$\begin{aligned}
\left(\frac{R(w+1)}{2}\right)^{1/s}\left((\tilde{U}_k)_{\rho\rho} + \frac{1}{\rho}(\tilde{U}_k)_\rho - \frac{k^2}{\rho^2}\tilde{U}_k\right) + (w+1)^2(\tilde{U}_k)_{ww} = 0 \quad & \text{in } (-1,1)^2,\\
\tilde{U}_k(\pm 1, w) = 0 \quad & \text{on } (-1,1),\\
\tilde{U}_k(\rho, 1) = 0 \quad & \text{on } (-1,1),\\
\partial_\nu \tilde{U}_k(\rho, -1) = Rd_s\tilde{f}_k(\rho)/2 \quad & \text{on } (-1,1),
\end{aligned} \tag{4.9}$$

and we solve for

$$\tilde{U}_k(\rho, w) = (1-\rho^2)\rho^{\min(|k|,2)}\tilde{V}_k(\rho, w). \tag{4.10}$$

Conventionally, one spectrally discretizes $\tilde{V}_k$ with Chebyshev polynomials for both ρ and w, but the matrices used to represent differentiation and function multiplication in the discretized equations are dense and hard to manipulate. Instead, we set

$$\tilde{V}_k(\rho, w) = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1} X_{ij}^{(k)}\tilde{C}_i^{(3/2)}(\rho)T_j(w), \quad \tilde{f}_k(\rho) = \sum_{i=0}^{n_1-1} F_i^{(k)}\tilde{C}_i^{(3/2)}(\rho), \tag{4.11}$$

where $T_j(w)$ is the Chebyshev polynomial of the first kind of degree j, $X^{(k)}$ is the matrix of coefficients of $\tilde{V}_k$ in the $\tilde{C}^{(3/2)}$ basis, and $F^{(k)}$ is the vector of coefficients of $\tilde{f}_k$ in the Chebyshev basis. Although $\tilde{C}^{(3/2)}$ is an uncommon polynomial basis, it is easy and efficient to transform coefficients in the $\tilde{C}^{(3/2)}$ basis to Chebyshev coefficients [74]. Therefore, users of the solver do not need to know about the special ultraspherical polynomial basis.

In this way, (4.10) indicates that we have three cases:

- If $|k| \geq 2$, then $\tilde{U}_k = \rho^2(1-\rho^2)\tilde{V}_k$, and (4.7) becomes:

$$\begin{aligned}\left(\frac{R}{2}\right)^{1/s}(w+1)^{1/s}[\rho^2(1-\rho^2)(\tilde{V}_k)_{\rho\rho} + \rho(5-9\rho^2)(\tilde{V}_k)_\rho + (4-k^2+(k^2-16)\rho^2)\tilde{V}_k],\\ +\rho^2(1-\rho^2)(w+1)^2(\tilde{V}_k)_{ww} = 0 \qquad \text{in } (-1,1)^2,\\ \tilde{V}_k(\rho,1) = 0 \qquad \text{on } (-1,1),\\ \rho^2(1-\rho^2)\partial_\nu \tilde{V}_k(\rho,-1) = d_s R\tilde{f}_k(\rho)/2 \qquad \text{on } (-1,1).\end{aligned} \tag{4.12}$$

Following [74] on operations related to $\tilde{C}^{(3/2)}$ and [159] on operations related to Chebyshev polynomials, (4.12) can be discretized to the following matrix equations:

$$\begin{aligned}(M_{\rho^2}D + 5M_\rho M D_u + (14-k^2)M - 10I)X^{(k)}(S_{2,0}B_1)^T + (M_{\rho^2}M)X^{(k)}(B_2D_2)^T = 0\\ (M_{\rho^2}M)X^{(k)}L^T = H^{(k)},\end{aligned} \tag{4.13}$$

where D is a diagonal matrix representing the second derivatives of $(1-\rho^2)\tilde{C}^{(3/2)}(\rho)$, M is a symmetric penta-diagonal matrix with 0 super and sub diagonals representing the operation of multiplying $\tilde{C}^{(3/2)}(\rho)$ by $(1-\rho^2)$, $M_{\rho^2} = I - M$ represents the operation of multiplying $\tilde{C}^{(3/2)}(\rho)$ by ρ^2, M_ρ is a tridiagonal matrix derived from the three-term recurrence (4.5) to represent the operation of multiplying $\tilde{C}^{(3/2)}(\rho)$ by ρ, D_u is a dense matrix representing the first derivatives of $\tilde{C}^{(3/2)}(\rho)$, but MD_u is tridiagonal [158, (18.9.8) & (18.9.19)], B_1 represents the operation of multiplying the Chebyshev polynomials $T(w)$ by $(R/2)^{1/s}(1+w)^{1/s}$, B_2 represents the operation of multiplying $C^{(2)}(x)$—ultraspherical polynomials with coefficient 2—by $(1+w)^2$, D_2 represents the second derivative of the Chebyshev basis, $S_{2,0}$ converts the Chebyshev basis to the $C^{(2)}$ basis,

$$L = \begin{bmatrix} T_0'(-1) & T_1'(-1) & \dots \\ T_0(1) & T_1(1) & \dots \end{bmatrix},$$

and $H^{(k)}$ is a matrix with two columns:

$$H^{(k)} = \begin{bmatrix} -d_s R F^{(k)}/2 & 0 \end{bmatrix}.$$

One can find details and visualizations of these operational matrices in [74, 210].

- If $|k| = 1$, then $\tilde{U}_k = \rho(1-\rho^2)\tilde{V}_k$. Using the same matrix notation, the discretized matrix equation is:

$$\begin{aligned} (M_\rho D + 3MD_u - 6M_\rho)X^{(k)}(S_{2,0}B_1)^T + (M_\rho M)X^{(k)}(B_2D_2)^T = 0 \\ (M_\rho M)X^{(k)}L^T = H^{(k)}. \end{aligned} \tag{4.14}$$

- If $k = 0$, then $\tilde{U}_k = (1-\rho^2)\tilde{V}_k$, and the discretized version of (4.7) is:

$$\begin{aligned} (M_{\rho^2} D + M_\rho M D_u - 2M_{\rho^2})X^{(0)}(S_{2,0}B_1)^T + (M_{\rho^2} M)X^{(0)}(B_2D_2)^T = 0 \\ MX^{(0)}L^T = H^{(0)}. \end{aligned} \tag{4.15}$$

We can merge the linear equation with the linear constraint into one matrix equation in all three cases, see for instance [210] for a similar approach. It is desirable to keep the structure and the sparsity of the matrices related to ultraspherical discretization while solving the equation. However, due to the variable s, we do not know the spectrum of the matrices associated with the Chebyshev basis, which is essential if we want to use fast iterative solvers such as ADI (Algorithm 5). Therefore, we treat all matrices as general dense matrices and solve all matrix equations with the Bartels–Stewart algorithm (Algorithm 4). Nevertheless, QZ decompositions on sparse penta-diagonal matrices are cheaper and more stable than general dense matrices. Thus, we gain from using ultraspherical polynomials. The solutions $X^{(k)}$ can be transformed to the Chebyshev coefficient matrix for both variables, and then they form the frontal slices of the solution $\mathcal{Z}$ to the discretized DFS version of (4.6). Finally, the coefficient matrix

that represents the solution of (4.1) can be calculated via

$$\mathcal{Z} \times_2 \begin{bmatrix} T_0(-1) & T_1(-1) & \cdots \end{bmatrix}.$$

We summarize this solver in Algorithm 8.

Algorithm 14 Fractional PDE solver on the unit disk — Polynomial approximation of $z^{1/s}$

1: **Input**: The coefficient matrix F of f after DFS extension in Chebyshev and Fourier bases.
2: **Output**: The coefficient matrix W of the solution u in Chebyshev and Fourier bases.
3: Convert the columns of F into coefficients in $\tilde{C}^{(3/2)}$ basis.
4: **while** $\max_i X^{(k)}_{i,n_2} > 10^{-10}$ for any k in (4.11) **do**
5: Increase n_2
6: Solve (4.13), (4.14), and (4.15) for $X^{(k)}$.
7: Stack all $X^{(k)}$ in the tube direction to form $\mathcal{Z}$.
8: Calculate $W = \mathcal{Z} \times_2 \begin{bmatrix} T_0(-1) & T_1(-1) & \ldots \end{bmatrix}$.
9: Convert the columns of W into coefficients in Chebyshev basis.

Remark (Numerical convergence). As a test, consider $s = 1/2$ so that the extended PDE is a Laplace equation on a cylinder. When $f = J_0(s_{01}\rho)$, where J_0 is the first Bessel function of the first kind, and s_{01} is the first nonzero root of J_0, the solution is

$$U = \frac{1}{s_{01}\cosh(s_{01}R)} J_0(s_{01}\rho)\sinh(s_{01}R(1-z)/2).$$

Figure 4.1 shows the reduction of approximation error through our adaptive solver.

4.2.2 Piecewise Polynomial Approximation of $z^{1/s}$

The more challenging case is when $z^{1/s}$ is not well-approximated by a polynomial. For these values of s, we approximate $z^{1/s}$ with piecewise polynomials,

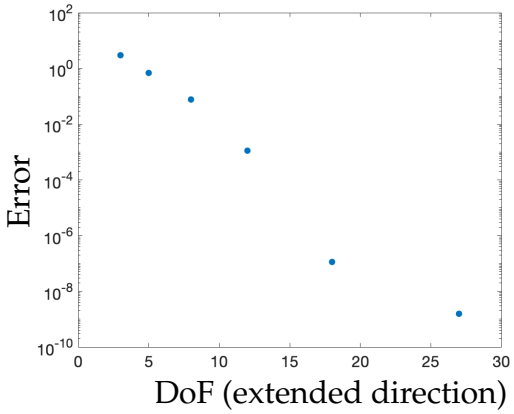

Figure 4.1: Solving fractional PDE for $f = J_0(s_{01}\rho)$ and $s = 1/2$ on the unit disk with our spectral solver. By increasing the degrees of freedom in the extended direction, the error between the analytic and numerical solutions decreases.

i.e.,

$$z^{1/s} \approx p_i(z), \quad z_i \le z \le z_{i+1}, \tag{4.16}$$

where p_i is a polynomial for $0 \le i \le \ell - 1$, $z_0 = 0$, and $z_\ell = R$. Then, on the interval (z_i, z_{i+1}), we have that

$$p_i(z)\left((\tilde{U}_k^{(i)})_{\rho\rho} + \frac{1}{\rho}(\tilde{U}_k^{(i)})_\rho - \frac{k^2}{\rho^2}\tilde{U}_k^{(i)}\right) + z^2(\tilde{U}_k^{(i)})_{zz} = 0 \quad \text{in } (-1,1)\times[-\pi,\pi]\times(z_i, z_{i+1}),$$

$$\tilde{U}_k^{(i)}(\pm 1, z) = 0 \quad \text{on } (z_i, z_{i+1}),$$

with bottom and top boundary conditions

$$\begin{cases} \tilde{U}_k^{(0)}(\rho, z_1) = \tilde{\phi}_1(\rho), \\ \partial_\nu \tilde{U}_k^{(0)}(\rho, z_0) = d_s \tilde{f}_k(\rho), \\ \partial_\nu \tilde{U}_k^{(0)}(\rho, z_1) = \tilde{\psi}_1(\rho), \end{cases} \qquad \begin{cases} \tilde{U}_k^{(\ell-1)}(\rho, z_{\ell-1}) = \tilde{\phi}_{\ell-1}(\rho), \\ \tilde{U}_k^{(\ell-1)}(\rho, R) = 0, \\ \partial_\nu \tilde{U}_k^{(\ell-1)}(\rho, z_{\ell-1}) = -\tilde{\psi}_{\ell-1}(\rho), \end{cases}$$

and

$$\begin{cases} \tilde{U}_k^{(i)}(\rho, z_i) = \tilde{\phi}_i(\rho), \\ \tilde{U}_k^{(i)}(\rho, z_{i+1}) = \tilde{\phi}_{i+1}(\rho), \\ \partial_\nu \tilde{U}_k^{(i)}(\rho, z_i) = -\tilde{\psi}_i(\rho), \\ \partial_\nu \tilde{U}_k^{(i)}(\rho, z_{i+1}) = \tilde{\psi}_{i+1}(\rho), \end{cases} \qquad 1 \le i \le l-2, \tag{4.17}$$

where $\tilde{\phi}_i$ and $\tilde{\psi}_i$ are implicitly defined. We use these implicit boundary conditions to ensure that $\tilde{U}_i$ can form an overall continuous solution. On intersection surfaces, solutions on the two sides have identical Dirichlet conditions and opposite Neumann conditions.

On each interval, we perform a change of variables to produce similar PDEs as (4.9). These PDEs can be discretized into $m\ell$ matrix equations. We solve on all intervals simultaneously by combining the matrix equations corresponding to the same Fourier mode into one equation. We use the matrix notation from Section 4.2.1 and get the following three matrix equations:

- If $|k| \geq 2$, then we have:

$$\begin{aligned}(M_{\rho^2}D + 5M_\rho M D_u + (14-k^2)M - 10I)Y^{(k)}E_1^T + (M_{\rho^2}M)Y^{(k)}E_2^T = 0\\ (M_{\rho^2}M)Y^{(k)}B_z^T = H^{(k)}.\end{aligned} \tag{4.18}$$

- If $|k| = 1$, then we have:

$$\begin{aligned}(M_\rho D + 3MD_u - 6M_\rho)Y^{(k)}E_1^T + (M_\rho M)Y^{(k)}E_2^T = 0\\ (M_\rho M)Y^{(k)}B_z^T = H^{(k)}.\end{aligned} \tag{4.19}$$

- If $k = 0$, then we have:

$$\begin{aligned}(M_{\rho^2}D + M_\rho M D_u - 2M_{\rho^2})Y^{(0)}E_1^T + (M_{\rho^2}M)Y^{(0)}E_2^T = 0\\ MY^{(0)}B_z^T = H^{(0)}.\end{aligned} \tag{4.20}$$

In the matrix equations above, we set

$$Y^{(k)} = \begin{bmatrix} X_0^{(k)} & \cdots & X_{\ell-1}^{(k)} \end{bmatrix},$$

where $X_j^{(k)}$ is the kth Fourier mode solution on (z_j, z_{j+1}). The first column of $H^{(k)}$ is $-d_s R\tilde{f}_k(\rho)/2$ and all other columns are zero. E_1 and E_2 are block diagonal matrices with diagonal blocks $S_{2,0}B_i$ and $K_i D_2$, respectively, where B_i

represents the multiplication of p_i and K_i represents the multiplication of the Chebyshev basis by $(w + \frac{z_{i+1}+z_i}{z_{i+1}-z_i})^2$. In addition, we set

$$B_z = \begin{bmatrix} P_{-1} & & & & \\ \frac{2}{z_1}P_1 & -\frac{2}{z_2-z_1}P_{-1} & & & \\ Q_1 & -Q_{-1} & & & \\ & \frac{2}{z_2-z_1}P_1 & -\frac{2}{z_3-z_2}P_{-1} & & \\ & Q_1 & -Q_{-1} & & \\ & & & \ddots & \\ & & & & Q_1 \end{bmatrix},$$

where $P_{-1} = [T_0'(-1) \;\; T_1'(-1) \cdots]$, $P_1 = [T_0'(1) \;\; T_1'(1) \cdots]$, $Q_{-1} = [T_0(-1) \; T_1(-1) \cdots]$, and $Q_1 = [T_0(1) \; T_1(1) \cdots]$. Then, we can form X_0 from $Y^{(k)}$ and obtain the coefficient matrix of the solution u of (4.1).

By construction, the first two columns of D_2 are 0. As a result, E_2 has 2ℓ scattered zero columns. This property is undesirable when using the solver in [210]. Instead, we permute the columns of E_2 so that the first 2ℓ columns are 0, and permute E_1, $Y^{(k)}$, B_z and $H^{(k)}$ accordingly. After solving the permuted matrix equations, we can easily obtain the original solution. We note that the two solvers are, in fact, equivalent for different values of s. The solver in this subsection can be thought of as a generalized version of the solver in Section 4.2.1, where the solution only has one piece and no implicit boundary conditions are needed. This generalized solver is described in Algorithm 9.

Remark. In our numerics, we use `Chebfun` [63] to automatically partition the extended direction and approximate $z^{1/s}$ with Chebyshev polynomials on each domain with desired accuracy level, i.e., to determine p_i and z_i in (4.16). `Chebfun`

Algorithm 15 Fractional PDE solver on the unit disk — Piecewise approximation of $z^{1/s}$

Input: The coefficient matrix F of f after DFS extension in Chebyshev and Fourier basis.
Output: The coefficient matrix W of the solution u in Chebyshev and Fourier basis.
Convert the columns of F into coefficients in $\tilde{C}^{(3/2)}$ basis.
while $\max_i \left(X_j^{(k)}\right)_{i,n_2^{(j)}} > 10^{-10}$ for any k or j **do**
    Increase $n_2^{(j)}$.
    Construct the matrices in (4.18), (4.19), and (4.20), and permute E_1, E_2, $Y^{(k)}$, B_z and $H^{(k)}$ such that the first 2ℓ columns of E_2 are zero columns while the equations still hold.
    Solve for $Y^{(k)}$.
    Stack $X_0^{(k)}$ to form $\mathcal{X}_0$.
.
Calculate $W = \mathcal{X}_0 \times_2 \begin{bmatrix} T_0(-1) & T_1(-1) & \dots \end{bmatrix}$.
Convert the columns of W into coefficients in Chebyshev basis.

only divides the domain when a high degree polynomial cannot achieve the resolution, so the piecewise polynomial approximation tends to have as few pieces as possible. From the above discretized systems, one notices that more pieces lead to more complicated matrix equations to solve. Consequently, it is beneficial to use `Chebfun` for approximations. Figure 4.2 shows the number of polynomial segments needed to approximate the map $z \mapsto z^{1/s}$ on $(0, 10)$ for varying $s \in (0, 1)$ with an accuracy of 10^{-12} using `Chebfun`. If we strive for machine precision, the numbers need to be larger. However, we found in practice that 10^{-12} gives sufficiently accurate PDE solutions. Comparatively, it is also possible to partition the z-direction into more segments so that $z^{1/s}$ can be approximated by a low degree polynomial on each interval. For this way of partitioning, we point the readers to [13] for more details.

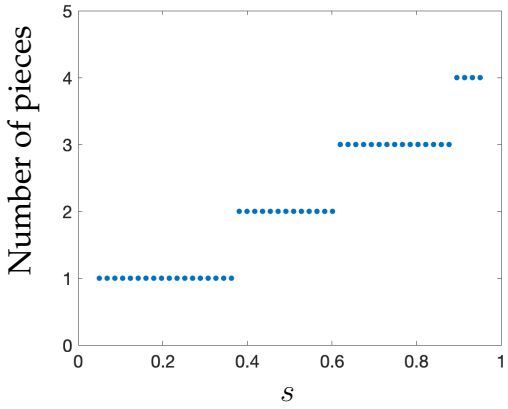

Figure 4.2: The number of segments for a piecewise polynomial approximation of $z^{1/s}$ on $(0, 10)$ with accuracy 10^{-12}.

4.3 Spectral Discretization for Fractional PDEs on a Rectangle

In this section, we solve (4.1) in the unit square $(-1, 1)^2$ by spectrally discretizing the truncated, extended PDE (4.4). General rectangles are straightforward to scale to unit squares by a change-of-variables and are thus easy to solve with the same method. It is also worth noting that the solver can be easily generalized to higher-dimensional domains. To this end, we demonstrate the solver application on a cube in Section 4.4.

We first introduce a direct solver that is similar to the solvers in Section 4.2. The direct solver, can encounter efficiency issues as the equations become large when $z^{1/s}$ needs to be approximated by a piecewise polynomial with many pieces. In those scenarios, we design a parallelizable solver using domain decomposition in Section 4.3.2.

4.3.1 Direct Solver

We first approximate $z^{1/s}$ as an ℓ-piece piecewise polynomial (4.16), with the simplest case being $\ell = 1$. On each interval (z_i, z_{i+1}), we have the PDE:

$$\begin{aligned} p_i(z)\left(U^{(i)}_{xx} + U^{(i)}_{yy}\right) + z^2 U^{(i)}_{zz} &= 0 \qquad && \text{in } (-1,1)\times(-1,1)\times(z_i, z_{i+1}), \\ U^{(i)}(\pm 1, y, z) &= 0 && \text{on } (-1,1)\times(z_i, z_{i+1}), \\ U^{(i)}(x, \pm 1, z) &= 0 && \text{on } (-1,1)\times(z_i, z_{i+1}), \end{aligned}$$

with bottom and top boundary conditions

$$\begin{cases} U^{(0)}(x,y,z_1) &= \phi_1(x,y), \\ \partial_\nu U^{(0)}(x,y,z_0) &= d_s f_k(x,y), \\ \partial_\nu U^{(0)}(x,y,z_1) &= \psi_1(x,y), \end{cases} \qquad \begin{cases} U^{(\ell-1)}(x,y,z_{\ell-1}) &= \phi_{\ell-1}(x,y), \\ U^{(\ell-1)}(x,y,R) &= 0, \\ \partial_\nu U^{(\ell-1)}(x,y,z_{\ell-1}) &= -\psi_{\ell-1}(x,y), \end{cases}$$

and

$$\begin{cases} U^{(i)}(x,y,z_i) &= \phi_i(x,y), \\ U^{(i)}(x,y,z_{i+1}) &= \phi_{i+1}(x,y), \\ \partial_\nu U^{(i)}(x,y,z_i) &= -\psi_i(x,y), \\ \partial_\nu U^{(i)}(x,y,z_{i+1}) &= \psi_{i+1}(x,y), \end{cases} \qquad 1 \le i \le l-2, \tag{4.21}$$

where ϕ_i and ψ_i are implicitly defined to ensure continuity of the entire solution. We employ the change the variables $w = \frac{2}{z_{i+1}-z_i}(z - z_i) - 1$ on (z_i, z_{i+1}), and assume the ansatz:

$$\begin{aligned} U^{(i)} &= (1-x^2)(1-y^2)\sum_{p=0}^{n_1}\sum_{q=0}^{n_2}\sum_{r=0}^{n_3^{(i)}} \mathcal{X}^{(i)}_{pqr}\tilde{C}_p^{(3/2)}(x)\tilde{C}_q^{(3/2)}(y)T_r(w), \\ f &= \sum_{p=0}^{n_1}\sum_{q=0}^{n_2} F_{pq}\tilde{C}_p^{(3/2)}(x)\tilde{C}_q^{(3/2)}(y), \end{aligned} \tag{4.22}$$

where $\tilde{C}_p^{(3/2)}(x)$ is the pth normalized ultraspherical polynomial with coefficient $3/2$, $T_r(w)$ is the rth Chebyshev polynomial, $\mathcal{X}^{(i)}$ is a 3D tensor, and F is a ma-

trix. As before, $\mathcal{X}^{(i)}$ and F contain the coefficients of $U^{(i)}$ and f in $\tilde{C}^{(3/2)}$ and Chebyshev basis, respectively. We can then write the discretized problem as a tensor equation by stacking $\mathcal{X}^{(i)}$ in the tube direction to form $\mathcal{Y}$:

$$\begin{aligned} \mathcal{Y} \times_1 A \times_3 E_1 + \mathcal{Y} \times_2 A \times_3 E_1 + \mathcal{Y} \times_1 A \times_2 A \times_3 E_2 = 0, \\ \mathcal{Y} \times_1 M \times_2 M \times_3 B_z = \mathcal{G}, \end{aligned} \tag{4.23}$$

where $A = D^{-1}M$, the first frontal slice of $\mathcal{G}$ is $-d_s RF/2$ and the remaining slices are zero, and all other matrices are defined in Section 4.2. In order to use a tensor analogue of the solver in [210], we first perform a column permutation on E_1, E_2 and B_z, and a frontal slice permutation on $\mathcal{Y}$ and $\mathcal{G}$ so that the first 2ℓ columns of E_2 are zero and (4.23) still holds.

The linear constraint $\mathcal{Y} \times_1 M \times_2 M \times_3 B_z = \mathcal{G}$ in (4.23) can be rewritten as $\mathcal{Y} \times_3 L = \mathcal{H}$, where $L = SB_z = \begin{bmatrix} I & \tilde{L} \end{bmatrix}$ such that the leftmost $2\ell \times 2\ell$ submatrix of L is the identity matrix, and $\mathcal{H} = \mathcal{G} \times_1 M^{-1} \times_2 M^{-1} \times_3 S$. Then, (4.23) can be combined into a single equation:

$$\begin{aligned} &\mathcal{Y} \times_1 A \times_3 (E_1 - (E_1)_{1:2\ell} L) + \mathcal{Y} \times_2 A \times_3 (E_1 - (E_1)_{1:2\ell} L) + \mathcal{Y} \times_1 A \times_2 A \times_3 (E_2 - (E_2)_{1:2\ell} L) \\ &\qquad = -\mathcal{H} \times_1 A \times_3 (E_1)_{1:2\ell} - \mathcal{H} \times_2 A \times_3 (E_1)_{1:2\ell} - \mathcal{H} \times_1 A \times_2 A \times_3 (E_2)_{1:2\ell}, \end{aligned} \tag{4.24}$$

where $(E_1)_{1:2\ell}$ represents the first 2ℓ columns of E_1. Since the first 2ℓ columns of E_2 are 0 and the leftmost $2\ell \times 2\ell$ submatrix of L is the identity, we know that the first 2ℓ frontal slices of $\mathcal{Y}$ do not influence the solution. Therefore, we can solve for $\mathcal{Y}_2$, which contains the rest of the frontal slices, by solving a smaller Sylvester equation:

$$\begin{aligned} &\mathcal{Y}_2 \times_1 A \times_3 R_1 + \mathcal{Y}_2 \times_2 A \times_3 R_1 + \mathcal{Y}_2 \times_1 A \times_2 A \times_3 R_2 \\ &\qquad = -\mathcal{H} \times_1 A \times_3 (E_1)_{1:2\ell} - \mathcal{H} \times_2 A \times_3 (E_1)_{1:2\ell}, \end{aligned} \tag{4.25}$$

where R_1 is the first $\left(\sum_i \left(n_3^{(i)}\right) - 2\ell\right)$ rows of $(E_1)_{2\ell+1:\sum_i \left(n_3^{(i)}\right)} - (E_1)_{1:2\ell}\tilde{L}$, and R_2 is the first $\left(\sum_i \left(n_3^{(i)}\right) - 2\ell\right)$ rows of $(E_1)_{2\ell+1:\sum_i \left(n_3^{(i)}\right)}$. After computing $\mathcal{Y}_2$, it is then straightforward to use the linear constraint to calculate the first 2ℓ frontal slices by $\mathcal{Y}_1 = \mathcal{H} - \mathcal{Y}_2 \times_3 \tilde{L}$.

Since we do not know the behavior of the spectrum of R_1 and R_2, we use a tensor analogue of the Bartels–Stewart algorithm (Algorithm 4) to solve for each column fiber of $\mathcal{Y}_2$. Finally, we recover $\mathcal{X}^{(0)}$ as the first $n_3^{(0)}$ frontal slices of $\mathcal{Y}$, convert it to the Chebyshev coefficient tensor $\mathcal{Z}$ and get the coefficient matrix of the solution of (4.1) by $\mathcal{Z} \times_3 \begin{bmatrix} T_0(-1) & T_1(-1) & \dots \end{bmatrix}$. This direct solver on a unit square is summarized in Algorithm 10.

Algorithm 16 Direct fractional PDE solver on the unit square

1: **Input**: The coefficient matrix F of f in Chebyshev basis.
2: **Output**: The coefficient matrix W of the solution u in Chebyshev basis.
3: Convert both columns and rows of F into coefficients in $\tilde{C}^{(3/2)}$ basis.
4: **while** $\max_{p,q} \left|\mathcal{X}^{(i)}_{p,q,n_3^{(i)}}\right| > 10^{-10}$ for any i in (4.22) **do**
5: Increase $n_3^{(i)}$.
6: Solve (4.25) for $\mathcal{Y}_2$ and compute $\mathcal{Y}_1 = \mathcal{H} - \mathcal{Y}_2 \times_3 \tilde{L}$.
7: Stack $\mathcal{Y}_1$ and $\mathcal{Y}_2$ in the tube direction, and form $\mathcal{X}^{(0)}$ to be the first $n_3^{(1)}$ frontal slices of $\mathcal{Y}$.
8: Calculate $W = \mathcal{X}^{(0)} \times_2 \begin{bmatrix} T_0(-1) & T_1(-1) & \cdots \end{bmatrix}$.
9: Convert both columns and rows of W into coefficients in Chebyshev basis.

Remark (Numerical convergence). As a numerical example, we consider the case that $u = \sin(\pi x)\sin(\pi y) + \sin(2\pi x)\sin(2\pi y)$, then $f = (2\pi^2)^s \sin(\pi x)\sin(\pi y) + (8\pi^2)^s \sin(2\pi x)\sin(2\pi y)$ by the spectral definition. Figure 4.3 (Left) shows the coefficient decay along the extended direction of the discretized tensor solution for different values of s. This plot demonstrates that when the algorithm terminates, the coefficient of the polynomial terms of the discretized solution is small enough. Figure 4.3 (Right) shows the accuracy improvements of our adaptive

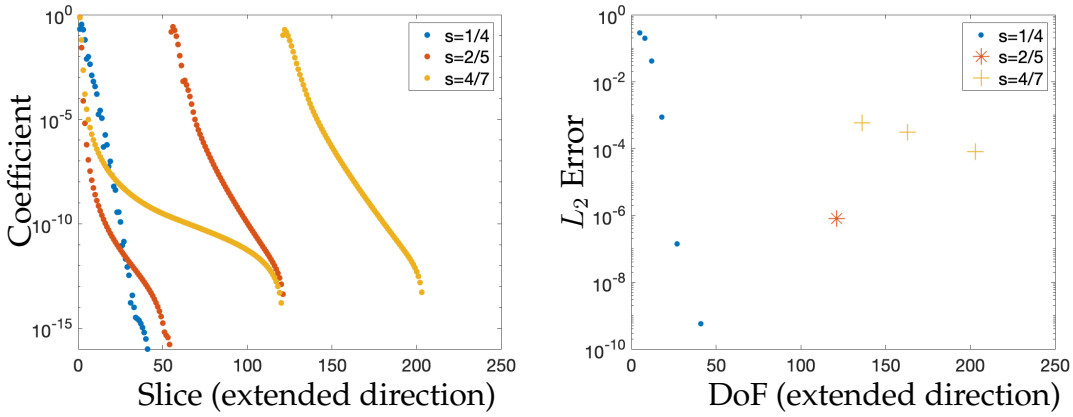

Figure 4.3: Solving the fractional PDE for $f = (2\pi^2)^s \sin(\pi x)\sin(\pi y) + (8\pi^2)^s \sin(2\pi x)\sin(2\pi y)$ and different values of s on $\Omega = (-1,1)^2$ with our spectral solver. **Left:** the largest coefficient in magnitude on each slice along the extended direction in the discretized solution, i.e., the largest $\mathcal{X}_{pqr}$ when r is fixed in (4.22). When $s = 2/5$ and $s = 4/7$, $z^{(1/s)}$ is approximated by piecewise polynomials with two pieces. The coefficient patterns show the decay of both pieces. **Right:** the accuracy achieved with different degrees of freedom in the extended direction. For $s = 1/4$, we achieve sufficient coefficient decay within five iterations. For $s = 4/7$, we require two iterations and for $s = 2/5$, the coefficients of a $28 \times 28 \times 121$ discretization decay below 10^{-13} in the first iteration, resulting in a single point on the plot.

algorithm. With the increase of polynomial degree to approximate the extended domain, we obtain a reduction of error between the numerical and analytic solution.

Remark (Numerical convergence for non-compatible datum). As another example, we consider a non-compatible case with $f = 2$ and $s < 1/2$, leading to low regularity of the solution to the fractional PDE [156, Sec. 6.3]. Figure 4.4 shows the coefficient decay along the extended direction of the discretized tensor solution for different values of s. Again, when the algorithm terminates, the coefficient of the polynomial terms of the discretized solution is small enough, and the number of coefficients to achieve this decay is smaller than that in Remark 4.3.1.

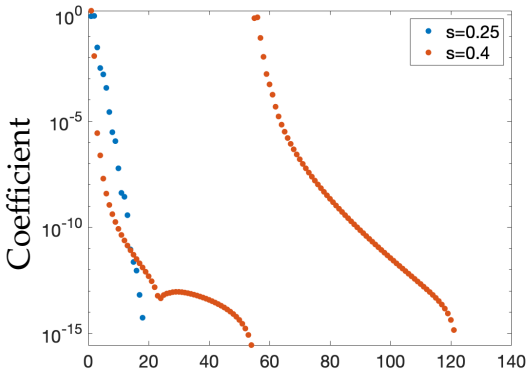

Figure 4.4: Solving the fractional PDE for $f = 2$ and different values of $s < 1/2$ on $\Omega = (-1, 1)^2$ with our spectral solver. The plot shows the largest coefficient in magnitude on each slice along the extended direction in the discretized solution, i.e., the largest $\mathcal{X}_{pqr}$ when r is fixed in (4.22). When $s = 2/5$, $z^{(1/s)}$ is approximated by a piecewise polynomial with two pieces. The coefficient patterns show the decay of both pieces.

4.3.2 Domain Decomposition Solver

In the previous section, we solved (4.1) on the unit square by jointly solving (4.21) for all segments. However, the solution of (4.1) only corresponds to $\mathcal{X}^{(0)}$, the solution on the first domain. Therefore, we design a domain decomposition solver, inspired by the hierarchical Poincaré–Steklov method [144], to solve only for $\mathcal{X}^{(0)}$.

Suppose we know the Robin boundary conditions for $\mathcal{X}^{(0)}$ and Dirichlet boundary conditions for all other $\mathcal{X}^{(i)}$. It is then straightforward to write a tensor equation for each $\mathcal{X}^{(i)}$ in the form of (4.24):

$$(\mathcal{X}^{(i)} \times_1 A + \mathcal{X}^{(i)} \times_2 A) \times_3 (C_i - (C_i)_{1:2} L_i) + \mathcal{X}^{(i)} \times_1 A \times_2 A \times_3 (M_2 - (M_2)_{1:2} L_i) \\ = -\mathcal{H}^{(i)} \times_1 A \times_3 (C_i)_{1:2} - \mathcal{H}^{(i)} \times_2 A \times_3 (C_i)_{1:2}, \tag{4.26}$$

where $C_i = S_{2,0}B_i$, $M_2 = K_i D_2$, $L_0 = N_0 \begin{bmatrix} P_{-1} \\ \\ Q_1 \end{bmatrix} = [I\ \tilde{L}_0]$, $L_i = N_i \begin{bmatrix} Q_{-1} \\ \\ Q_1 \end{bmatrix} = [I\ \tilde{L}_i]$ for $i > 1$, $\mathcal{H}^{(i)} = \mathcal{G}^{(i)} \times_1 M^{-1} \times_2 M^{-1} \times_3 N_i$, and the two frontal slices of $\mathcal{G}^{(i)}$ correspond to the boundary conditions of $\mathcal{X}^{(i)}$ that are assumed to be known. This means that, once we have $\mathcal{G}^{(i)}$, we are able to solve for $\mathcal{X}^{(i)}$.

Equation (4.26) allows us to construct a solution map $S^{(i)} \in \mathbb{R}^{n_1 n_2 n_3^{(i)} \times 2n_1 n_2}$:

$$S^{(i)} = -[(C_i - (C_i)_{1:2}L_i) \otimes I \otimes A + (C_i - (C_i)_{1:2}L_i) \otimes A \otimes I + (M_2 - (M_2)_{1:2}L_i) \otimes A \otimes A]^{-1} \\ (((C_i)_{1:2}N_i) \otimes (I \otimes A + A \otimes I)), \tag{4.27}$$

such that $\mathrm{vec}\left(\mathcal{X}^{(i)}\right) = S^{(i)}\ \mathrm{vec}\left(\mathcal{G}^{(i)}\right)$, where $\mathrm{vec}\left(\mathcal{X}^{(i)}\right)$ reshapes all elements in $\mathcal{X}^{(i)}$ to a vector. In addition, we define the Dirichlet-to-Neumann (DtN) map $K^{(i)} \in \mathbb{R}^{2n_1 n_2 \times 2n_1 n_2}$ by

$$K^{(i)} = \frac{2}{z_i - z_{i-1}} \left(\begin{bmatrix} P_{-1} \\ \\ P_1 \end{bmatrix} \otimes I \otimes I \right) S^{(i)}.$$

The map $K^{(i)}$ converts the Dirichlet boundary conditions $\mathcal{G}^{(i)}$ into the Neumann values $\mathcal{B}^{(i)}$ on the boundaries.

We solve for each column of $S^{(i)}$ by solving a tensor Sylvester equation:

$$\mathcal{S}_j^{(i)} \times_1 A \times (C_i - (C_i)_{1:2}L_i) + \mathcal{S}_j^{(i)} \times_2 A \times_3 (C_i - (C_i)_{1:2}L_i) + \mathcal{S}_j^{(i)} \times_1 A \times_2 A \times_3 (M_2 - (M_2)_{1:2}L_i) = \mathcal{W}_j^{(i)},$$

where $\mathcal{S}_j^{(i)}$ the jth column of $S^{(i)}$ reshaped to an $n_1 \times n_2 \times n_3^{(i)}$ tensor, and $\mathcal{W}_j^{(i)}$ is the jth column of $((C_i)_{1:2}N_i) \otimes (I \otimes A + A \otimes I)$ reshaped to an $n_1 \times n_2 \times n_3^{(i)}$ tensor. This suggests that the computation of $\mathcal{S}_j^{(i)}$ is a parallelizable process. We can calculate the operator $K^{(i)}$ by

$$\mathcal{K}_j^{(i)} = \frac{2}{z_i - z_{i-1}} \mathcal{S}_j^{(i)} \times_3 \begin{bmatrix} P_{-1} \\ \\ P_1 \end{bmatrix},$$

where $\mathcal{K}_j^{(i)}$ is the jth vector of $K^{(i)}$ reshaped to a $2 \times n_1 \times n_2$ tensor.

Our goal is to use the solution operator $S^{(0)}$ to solve for $\mathcal{X}^{(0)}$. Thus, we must obtain the implicit Dirichlet boundary condition. Next, we show that we can carefully merge the solution maps $S^{(i)}$ and DtN maps $K^{(i)}$. As a result, we obtain solution maps and DtN maps that work on several domains simultaneously, and these merged maps can help us find the desired boundary condition.

The boundaries for each domain consist of one upper surface and one lower surface so that we can separate the boundary conditions into two parts and the DtN operator into four parts, i.e.,

$$\operatorname{vec}\left(\mathcal{G}^{(i)}\right) = \begin{bmatrix} \mathcal{G}_v^{(i)} \\ \\ \mathcal{G}_u^{(i)} \end{bmatrix}, \quad \operatorname{vec}\left(\mathcal{B}^{(i)}\right) = \begin{bmatrix} \mathcal{B}_v^{(i)} \\ \\ \mathcal{B}_u^{(i)} \end{bmatrix}, \quad K^{(i)} = \begin{bmatrix} K_{v,v}^{(i)} & K_{v,u}^{(i)} \\ \\ K_{u,v}^{(i)} & K_{u,u}^{(i)} \end{bmatrix}.$$

Then we can follow [144] to merge the solution and DtN operators:

$$S^{(i,i+1)} = \left(K_{u,u}^{(i)} - K_{v,v}^{(i+1)}\right)^{-1} \begin{bmatrix} -K_{u,v}^{(i)} & K_{v,u}^{(i+1)} \end{bmatrix},$$

$$K^{(i,i+1)} = \begin{bmatrix} K_{v,v}^{(i)} & 0 \\ \\ 0 & K_{u,u}^{(i+1)} \end{bmatrix} + \begin{bmatrix} K_{v,u}^{(i)} \\ \\ K_{u,v}^{(i+1)} \end{bmatrix} S^{(i,i+1)},$$

so that

$$\mathcal{G}_u^{(i)} = \mathcal{G}_v^{(i+1)} = S^{(i,i+1)} \begin{bmatrix} \mathcal{G}_v^{(i)} \\ \\ \mathcal{G}_u^{(i+1)} \end{bmatrix}, \quad \begin{bmatrix} \mathcal{B}_v^{(i)} \\ \\ \mathcal{B}_u^{(i+1)} \end{bmatrix} = K^{(i,i+1)} \begin{bmatrix} \mathcal{G}_v^{(i)} \\ \\ \mathcal{G}_u^{(i+1)} \end{bmatrix}.$$

In other words, given the Dirichlet boundary conditions on the lower surface of the ith domain and the upper surface of the $(i+1)$st domain, $K^{(i,i+1)}$ enables one to compute the Neumann conditions on those surfaces, and $S^{(i,i+1)}$ allows one to calculate the Dirichlet condition of the overlapping surface.

We need $\mathcal{G}_u^{(0)}$ or $\mathcal{G}_v^{(1)}$ to compute $\mathcal{X}^{(0)}$. To achieve this, we can merge the operators for $\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(\ell-1)}$ to get $S^{(1,\ell-1)}$ and $K^{(1,\ell-1)}$, and then merge them

with $S^{(0)}$ and $K^{(0)}$ in the final step. In particular, there are two ways of merging the operators:

1. Starting from the top piece, we form $S^{(\ell-1)}$ and $K^{(\ell-1)}$. We then iterate downwards from $i = \ell - 1$ to $i = 0$, form new operators for each piece, and merge them with the operators from the previous iteration. This is merging in a sequential way.
2. Since the maps for each piece are independent, we can form the operators on all domains in parallel. Then, we merge in a hierarchical manner, merging two of them simultaneously, and these merging operations are parallelizable.

In summary, this domain decomposition solver is Algorithm 11.

Algorithm 17 Domain decomposition fractional PDE solver on the unit square

Input: The coefficient matrix F of f in Chebyshev basis.
Output: The coefficient matrix W of the solution u in Chebyshev basis.
Convert both columns and rows of F into coefficients in $\tilde{C}^{(3/2)}$ basis.
while $\max_{p,q} \left| \mathcal{X}^{(i)}_{p,q,n_3^{(i)}} \right| > 10^{-10}$ for any i in (4.26) **do**
    Increase $n_3^{(i)}$.
    Form solution and DtN maps on each domain.
    Merge the maps to get $S^{(1,\ell-1)}$ and $K^{(1,\ell-1)}$.
    Merge with $S^{(0)}$ and $K^{(0)}$ to get solution tensor $\mathcal{X}^{(0)}$ on the first domain.
.
Calculate $W = \mathcal{X}^{(0)} \times_2 \begin{bmatrix} T_0(-1) & T_1(-1) & \cdots \end{bmatrix}$.
Convert both columns and rows of W into coefficients in Chebyshev basis.

Remark. We can also use domain decomposition to construct a parallel solver for the disk domain. The solution operator $S^{(i)} \in \mathbb{C}^{n_1 n_2^{(i)} m \times 2n_1 m}$ on the ith domain takes in the Dirichlet boundary coefficients on the top and the bottom surfaces, and returns the coefficients of $\tilde{U}^{(i)}$. Although we cannot construct $S^{(i)}$ in one setting due to partial regularity, we discover that row $\left((k-1)n_1 n_2^{(i)} + 1\right)$ to row

$\left(kn_1n_2^{(i)}\right)$ of $S^{(i)}$ corresponds to the $(k-1-m/2)$th Fourier mode of the solution. In this way, we can construct these rows with the linear system converted from the generalized Sylvester equation. It is then straightforward to construct the DtN map $K^{(i)}$ from $S^{(i)}$, and we can use the hierarchical Poincaré–Steklov method described for the rectangle domain.

4.4 Numerical Example and Application to Optimal Control Problems

In this section, we present two examples. In the first example, we extend our 2D solver to 3D to solve the fractional elliptic PDE on $\Omega = (0,1)^3$. The second example is an optimal control problem with a fractional PDE constraint.

4.4.1 Fractional PDE on the Cube

We can extend our solver from Section 4.3 to solve fractional PDEs on the unit cube. Using ultraspherical polynomials for the cube dimensions and Chebyshev polynomials for the extended direction, the discretized problem is the following:

$$\begin{aligned}\mathcal{Y} \times_1 A \times_2 A \times_4 E_1 + \mathcal{Y} \times_2 A \times_3 A \times_4 E_1 + \mathcal{Y} \times_1 A \times_3 A \times_4 E_1 + \mathcal{Y} \times_1 A \times_2 A \times_3 A \times_4 E_2 = 0,\\ \mathcal{Y} \times_1 M \times_2 M \times_3 M \times_4 B_z = -d_s R\mathcal{F}/2,\end{aligned} \tag{4.28}$$

where $\mathcal{Y}$ is formed by stacking 4D tensor solutions of the discretized problem on each interval along the fourth dimension, $\mathcal{F}$ is a 3D tensor representing the

polynomial coefficients of the initial condition, and all other matrices have been defined in Section 4.3.

We solve (4.28) by merging the linear constraint into the tensor equation and using a 4D Bartels–Stewart algorithm analogue to obtain the solution directly. For a numerical example, we consider the simple case that $f = (3\pi^2)^s \sin(\pi x)\sin(\pi y)\sin(\pi z) + (12\pi^2)^s \sin(2\pi x)\sin(2\pi y)\sin(2\pi z)$ so that the analytic solution is $u = \sin(\pi x)\sin(\pi y)\sin(\pi z)+\sin(2\pi x)\sin(2\pi y)\sin(2\pi z)$. Figure 4.5 (Left) shows the coefficient decay along the extended direction of the discretized tensor solution for different values of s. This plot shows that the coefficient of the discretized solution is small enough when the algorithm finishes. Figure 4.5 (Right) shows the accuracy improvements in our adaptive algorithm when we increase the degrees of freedom by allowing higher polynomial degrees to approximate the solution in the extended direction. Similar to the problem on the square, polynomial coefficients of the discretized solution decay exponentially along the extended direction, which allows us to find an accurate numerical solution with only a few degrees of freedom.

4.4.2 Optimal Control Problem

We consider the optimization problem given by

$$\begin{aligned} &\min_{(u,q)} \left\{ J(u,q) := \frac{1}{2}\int_\Omega |u-u_d|^2 + \frac{\alpha}{2}\int_\Omega |q|^2 \right\}, \\ &\text{subject to} \qquad (-\Delta)^s u = q \quad \text{in } \Omega, \end{aligned} \tag{4.29}$$

where u_d is a given function and α is the control penalty parameter. We solve this problem via a direct solver. Specifically, we express the optimality condition

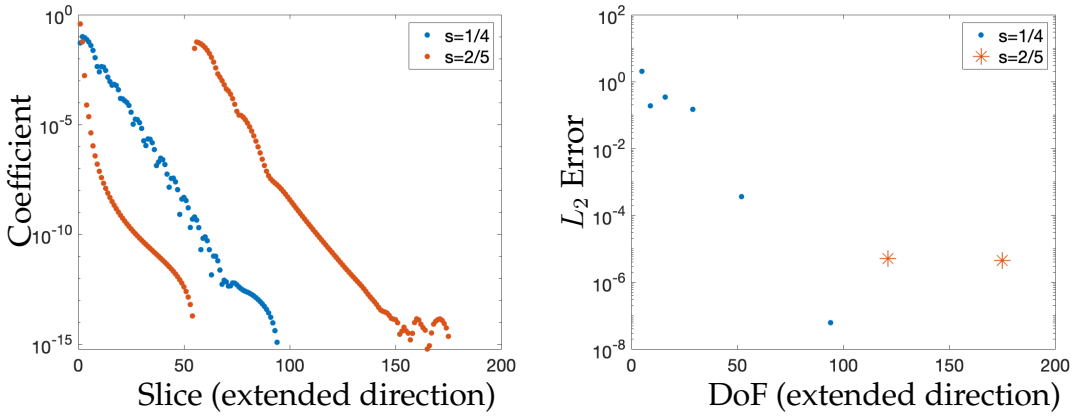

Figure 4.5: Solving the fractional PDE for $f = (3\pi^2)^s \sin(\pi x)\sin(\pi y)\sin(\pi z) + (12\pi^2)^s \sin(2\pi x)\sin(2\pi y)\sin(2\pi z)$ and different values of s on $\Omega = (-1,1)^3$ with our spectral solver. **Left:** the largest coefficient in magnitude on each slice along the extended direction in the discretized solution, i.e., the largest coefficient when the fourth index of the discretized tensor solution is fixed. When $s = 2/5$, $z^{(1/s)}$ is approximated by a piecewise polynomial with two pieces. In this case, the coefficient patterns show the decay of both pieces. **Right:** the accuracy achieved with different degrees of freedom in the extended direction. When $s = 1/4$, the algorithm performs five iterations, increasing the degrees of freedom along the extended direction to achieve sufficient decay of the coefficients. On the other hand, when $s = 2/5$, the solution admits sufficient coefficient decay in the first iteration because the adaptive solver adds additional degrees of freedom.

as two fractional PDEs, discretize both, combine them into one tensor equation and obtain the best u and z directly. In particular, we solve

$$\begin{aligned} (-\Delta)^s u &= -\frac{1}{\alpha} p, \\ (-\Delta)^s p &= u - u_d, \end{aligned} \tag{4.30}$$

where we have eliminated the so-called gradient equation. For simplicity, we consider solving (4.30) directly on the unit square. Let $\mathcal{U}$ and $\mathcal{P}$ be the coefficient tensors for the extensions of u and p, respectively. Then, $\mathcal{U}$ and $\mathcal{P}$ satisfy the

following tensor equations:

$$\begin{aligned}
\mathcal{U} \times_1 A \times_3 E_1^{(u)} + \mathcal{U} \times_2 A \times_3 E_1^{(u)} + \mathcal{U} \times_1 A \times_2 A \times_3 E_2^{(u)} &= 0, \\
\mathcal{U} \times_3 B_y^{(u)} &= \mathcal{G}^{(u)}, \\
\mathcal{P} \times_1 A \times_3 E_1^{(p)} + \mathcal{P} \times_2 A \times_3 E_1^{(p)} + \mathcal{U} \times_1 A \times_2 A \times_3 E_2^{(p)} &= 0, \\
\mathcal{P} \times_3 B_y^{(p)} &= \mathcal{G}^{(p)}, \qquad (4.31)
\end{aligned}$$

where the first frontal slice of $\mathcal{G}^{(u)}$ is $\frac{d_s z_1}{2\alpha} \mathcal{P} \times_3 \begin{bmatrix} T_0(-1) & \dots & T_{n_3^{(p)}}(-1) & 0 & \dots & 0 \end{bmatrix}$, the first frontal slice of $\mathcal{G}^{(p)}$ is $-\frac{d_s z_1}{2} \left(\mathcal{U} \times_3 \begin{bmatrix} T_0(-1) & \dots & T_{n_3^{(u)}}(-1) & 0 & \dots & 0 \end{bmatrix} - U_d \right)$, and U_d is the coefficient matrix of u_d. To distinguish between matrices used in the equations for u and p, we use superscripts (u) and (p). The matrices E_1, E_2 and B_y are defined in Section 4.3. We can then rearrange (4.31) so that we only have one tensor Sylvester equation and a linear constraint:

$$\begin{aligned}
\mathcal{Y} \times_1 A \times_3 \begin{bmatrix} E_1^{(u)} & \\ & E_1^{(p)} \end{bmatrix} + \mathcal{Y} \times_2 A \times_3 \begin{bmatrix} E_1^{(u)} & \\ & E_1^{(p)} \end{bmatrix} + \mathcal{Y} \times_1 A \times_2 A \times_3 \begin{bmatrix} E_2^{(u)} & \\ & E_2^{(p)} \end{bmatrix} &= 0, \\
\mathcal{Y} \times_3 B_y &= \mathcal{G}, \qquad (4.32)
\end{aligned}$$

where $\mathcal{Y}$ is formed by stacking U and P along the tube direction, $\mathcal{G}$ is a zero tensor except for the last frontal slice of $\frac{1}{2} d_s z_1 U_d$,

$$B_y = \begin{bmatrix} B_y^{(u)}(1) & -\frac{d_s z_1}{2\alpha} \begin{bmatrix} T_0(-1) & \dots & T_{n_3^{(p)}}(-1) & 0 & \dots & 0 \end{bmatrix} \\ B_y^{(u)}(2:\text{end}) & \\ & B_y^{(p)}(2:\text{end}) \\ \frac{d_s z_1}{2} \begin{bmatrix} T_0(-1) & \dots & T_{n_3^{(u)}}(-1) & 0 & \dots & 0 \end{bmatrix} & B_y^{(p)}(1) \end{bmatrix},$$

and $B_y^{(u)}(i)$ is the ith row of $B_y^{(u)}$. We can then solve for $\mathcal{Y}$, get $\mathcal{U}$ and $\mathcal{P}$, and calculate the coefficient tensor $\mathcal{Q}$ of q by $\mathcal{Q} = -\frac{1}{\alpha}\mathcal{P}$.

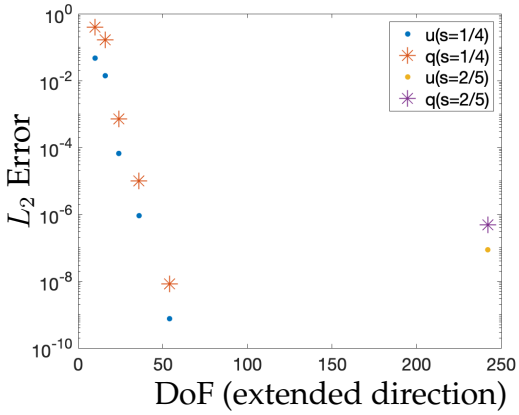

Figure 4.6: Solving the optimal control problem (4.29) with different values of s for $\alpha = 10^{-2}$, $u_d = (1 + \alpha(2\pi^2)^{2s}) \sin(\pi x) \sin(\pi y)$ and $\Omega = (-1, 1)^2$. As the degrees of freedom in the extended direction increase, the approximation of both u and q improves. When $s = 1/4$, we perform four iterations, which add more degrees of freedom along the extended direction, before our algorithm terminates. When $s = 2/5$, our first trial guarantees enough decay in the coefficients for both u and q, resulting in a single point for each on the plot.

For numerical demonstration, we take $\Omega = (-1, 1)^2$, $\alpha = 10^{-2}$, and $u_d = (1+\alpha(2\pi^2)^{2s}) \sin(\pi x) \sin(\pi y)$. The analytic solution of (4.29) with this data is $u = \sin(\pi x) \sin(\pi y)$ and $q = (2\pi^2)^s \sin(\pi x) \sin(\pi y)$. Figure 4.6 shows the performance of our adaptive direct solver. As demonstrated, our method generates accurate numerical solutions for both u and q with only a few degrees of freedom in the extended direction.

4.5 Conclusions

In this chapter, we present a spectral method that uses ultraspherical and Fourier polynomials to solve fractional Laplacian equations on square and disk domains via the Caffarelli–Silvestre extension. Based on the value of the fractional exponent s, we decompose the PDE along the extended domain. We show a direct method that finds solutions on all sub-domains through one ten-

sor equation, and a parallelizable domain decomposition solver generated from the hierarchical Poincaré–Steklov method. Numerical tests suggest that coefficients of the solutions decay exponentially along the extended direction, and we can recover accurate discretized solutions with a few degrees of freedom. Our method is easily generalized to problems of higher dimensions, such as solving fractional PDEs on cubes, and it can be used to accurately compute solutions of optimal control problems. For future work, we will develop spectral solvers for fractional operators with variable coefficients (4.2) through the extension scheme, yielding an approach that can be used to solve more general PDEs and optimal control problems as in [186].

CHAPTER 5

ELECTRON CORRELATION ENERGY COMPUTATION WITH SYLVESTER EQUATIONS

In this chapter[1], we incorporate dimensionality increase ideas and Sylvester tensor equation methods to compute electron correlation energy in computational quantum chemistry. In the past few decades, correlation has been used by researchers in quantum many-body problems to explain behaviors that are not described by density functional theory [148, 171, 180, 231]. The journey to highly accurate correlation energy is a long haul, where storage and computational inefficiency have accompanied the emergence of new accurate computing routines. We introduce numerical linear algebra and scientific computing into second-order Møller–Plesset perturbation theory (MP2) model [103, 208], one of the current most popular and accurate density functional approximation, to design fast algorithms for handling correlation.

5.1 Introduction

In quantum chemistry, a longstanding challenge is to evaluate the electron correlation energy of the electronic ground state of molecules at the lowest possible computational cost. Starting from the mean-field Hartree-Fock (HF) method,

[1]This chapter is based on a project with Abdulrahman Aldossary, Yang Liu, Zhenling Wang, Martin Head-Gordon, and Sherry Li. I started working on this project in the summer of 2021 as a research intern at Lawrence Berkeley National Laboratory (LBL), and will continue to make more progress on it as a postdoc at LBL. For this project, I derived the mathematical formulations, implemented MATLAB codes for preliminary results and C++ codes for comparison with existing methods. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of Basic Energy Sciences, Scientific Discovery through Advanced Computing (SciDAC) program.

chemists have included dependence on different forms of density and atom orbitals to achieve a higher level of accuracy during computations [101,142]. Currently, we can obtain density functional approximations with the smallest error using double hybrid density functional theory [79,142,143]. However, computation costs of these accurate methods grow drastically with respect to the system size N. For example, wave function-based improvements upon the HF method, such as MP2, coupled-clusters with singles and doubles (CCSD) [21, 22], and CCSD with correcting for perturbative triples (CCSD(T)) [174] hold the highest level of accuracy, but scale as $\mathcal{O}(N^5)$, $\mathcal{O}(N^6)$, and $\mathcal{O}(N^7)$ respectively. Therefore, the needs to reduce the scaling of these methods are well-recognized and have been research tasks for many years.

As the lowest order perturbation theory correction to HF, MP2 is free of self-interaction or delocalization errors, and provides a considerate amount of improvement in accuracy. It mainly exhibits significant advantages for stable closed shell organic and main group inorganic molecules [80, 104]. In addition, MP2 also improves the quality of optimized molecular structures and properties of closed shell molecules such as dipole moments [95, 104] and electrical polarizabilities [94]. MP2 typically experiences failures when the molecules are strongly correlated [196]. It also has drawbacks when the HF orbitals exhibit artificial symmetry-breaking even without obvious strong correlations [131], and when there is the presence of significant non-additive weak correlation effects [188]. Nevertheless, the simplicity and comparatively low scaling still mark MP2 as one of the most popular and useful methods for electron correlation.

In quantum chemistry, two types of orbital basis are often used. There is

the atomic orbital (AO) basis, where electrons in it are under the influence of only one nucleus of the atom. There is also the molecular orbital (MO) basis, where electrons in it are affected by two or more nuclei in a molecule. In general, the AO basis is an inherent property of the an atom, while an MO basis is formed by a linear combination of the AO basis. Given this formulation, MO basis of a molecule is not unique. To date, there are several successful attempts to reduce the scaling of MP2. One of them is the cutoff-based Laplace transform MP2 method [6, 12], which yields asymptotically quadratic or even linear scaling using canonical MO basis (see section 5.1.2). Another is the local density-fitting MP2 [180,226], which combines MP2 and localized MO basis (see section 5.1.2), and is shown to have linear scaling. Similar ideas have also been applied to CCSD and CCSD(T) to achieve linear scaling [91]. However, these ideas usually include many truncations during system construction, giving discontinuities when the geometry changes and unbounded errors for chemical approximations. In this chapter, we develop alternative methods for computing the correlation energy using MP2 with electrons in canonical and localized MO basis. These methods have good theoretical scaling, and also perform excellently in practice.

A major component to understanding chemical orbitals is to differentiate occupied orbitals and virtual orbitals [102]. In general, occupied orbitals are those that contain at least one electron and are used to update the mean-field potential in HF theory. The rest are then defined as virtual (or unoccupied) orbitals. Mathematically speaking, occupied and virtual orbitals lie within the solution of a nonlinear generalized eigenvalue problem [189]:

$$F(C)C = SC\epsilon, \tag{5.1}$$

where F is the Fock matrix representing effective potentials, S is the overlap

matrix indicating locations of electrons, C stores the generalized eigenvectors, and ϵ contains all the generalized eigenvalues. The lowest n_{occ} eigenvalues correspond to the occupied orbitals, and we can use their corresponding eigenvectors as coefficients to form an occupied orbital basis. As a result, the rest n_{virt} eigenvalues correspond to the virtual orbitals, and we can form a virtual orbital basis with eigenvectors in C. Due to the dependence of F on C, we need to solve this eigenvalue problem iteratively until convergence. For notational and conceptual simplicity, we refer to the final result of F and S as the Fock and overlap matrices for the remainder of this chapter. In particular, in a representation of orthogonal orbitals for both occupied and virtual basis, the overlap matrix S is the identity matrix, and thus we can gain huge computational benefits.

In the rest of this section, we briefly describe the mathematical problem we encounter to compute the correlation energy through perturbation theory, and introduce a couple of orbital bases we consider our problems in. Then, in section 5.2, we rewrite our target linear system into Sylvester tensor equations, and discuss how we interpret them in canonical and localized orbital bases. Next, we design a block low-rank solver for the canonical representation using fADI in section 5.3, and a sparsity enforcement Krylov subspace solver for the localized representation in section 5.4. Finally, we make some concluding remarks and discuss our future directions in section 5.5.

5.1.1 Mathematical formulation of correlation energy

The simplest way to capture more of the many–body wavefunction character is to use perturbation theory, i.e., Rayleigh–Schrodinger theory [228, 229]. In this

approximation, the correlation energy is:

$$E_{\text{corr}} = \mathcal{L} \cdot \mathcal{T}, \quad \mathcal{L}, \mathcal{T} \in \mathbb{R}^{n_{\text{virt}} \times n_{\text{occ}} \times n_{\text{virt}} \times n_{\text{occ}}}, \tag{5.2}$$

$$\mathcal{L}_{a,i,b,j} = (ia|jb) = \int dr \int dr' \phi_a(r)\phi_b(r) \frac{1}{|r - r'|} \psi_i(r')\psi_j(r'), \tag{5.3}$$

$$1 \leq i, j \leq n_{\text{occ}},\ 1 \leq a, b \leq n_{\text{virt}},$$

where $\mathcal{T}$ is an unknown tensor we need to solve for, n_{occ} is the number of occupied orbitals, n_{virt} is the number of virtual orbitals, $(ia|jb)$ are unsymmetrized two-electron integrals in Mulliken notation with ϕ and ψ linear combinations of spherical harmonic Gaussians [132, 153]. If $\mathcal{L}$ is reshaped to a matrix $L \in \mathbb{R}^{n_{\text{virt}} n_{\text{occ}} \times n_{\text{virt}} n_{\text{occ}}}$, L is also referred to as the electron repulsion integral (ERI) matrix. Throughout this chapter, we use a, b to denote indices for the virtual orbitals, and i, j to denote indices for the occupied orbitals. Because (5.2) is a dot product, our goal is to solve either $\mathcal{T}$, or $T = \text{reshape}(\mathcal{T}, n_{\text{virt}} n_{\text{occ}}, n_{\text{virt}} n_{\text{occ}})$, or $t = \mathcal{T}(:)$. If we assume that each orbital has two electrons, t is the solution of the linear system

$$\Delta t = 2l - k = c, \tag{5.4}$$

where $l = \mathcal{L}(:), k = \mathcal{K}(:), \mathcal{K} = (ib|ja)$, and $\Delta = \text{reshape}(\mathcal{H}, \text{n}_{\text{virt}}^2 \text{n}_{\text{occ}}^2, \text{n}_{\text{virt}}^2 \text{n}_{\text{occ}}^2)$ with

$$\begin{aligned} \mathcal{H}_{a,i,b,j,a',i',b',j'} &= S_{a,a'} F_{i,i'} S_{b,b'} S_{j,j'} + S_{a,a'} S_{i,i'} S_{b,b'} F_{j,j'} \\ &\quad - F_{a,a'} S_{i,i'} S_{b,b'} S_{j,j'} - S_{a,a'} S_{i,i'} F_{b,b'} S_{j,j'}. \end{aligned} \tag{5.5}$$

Here, to comply with notations used in quantum chemistry community, we use F_{virt} and F_{occ} to denote submatrices of the Fock matrix that correspond to the virtual and occupied orbitals respectively. Similarly, we can define S_{virt} and S_{occ}, and in fact,

$$F = \begin{bmatrix} F_{\text{occ}} & \\ & F_{\text{virt}} \end{bmatrix}, \qquad S = \begin{bmatrix} S_{\text{occ}} & \\ & S_{\text{virt}} \end{bmatrix}.$$

In addition, we use $F_{a,a'}$ and $F_{b,b'}$ to access a specific element of F_{virt}, $S_{a,a'}$ and $S_{b,b'}$ to access a specific element of S_{virt}, and similarly for the occupied orbitals. In a representation of orthogonal orbitals, (5.5) can be simplified to

$$\begin{aligned}\mathcal{H}_{a,i,b,j,a',i',b',j'} = \delta_{a,a'}F_{i,i'}\delta_{b,b'}\delta_{j,j'} + \delta_{a,a'}\delta_{i,i'}\delta_{b,b'}F_{j,j'} \\ - F_{a,a'}\delta_{i,i'}\delta_{b,b'}\delta_{j,j'} - \delta_{a,a'}\delta_{i,i'}F_{b,b'}\delta_{j,j'},\end{aligned} \tag{5.6}$$

where δ is the Dirac delta function.

It is important to emphasize that since we fix spin orthogonality, (5.4) is different from the linear system that one generates with the Hylleraas functional [107, 180, 208]. However, this simplification drops a lot of chemical complications, and we can use it as a first step to test our new algorithms. In addition, since (5.2) only consists of a dot product, we can show that

$$E_{\text{corr}} = l^T t = l^T(\Delta^{-1}c) = (\Delta^{-1}l)^T c, \tag{5.7}$$

where the last equality holds since Δ is symmetric. In other words, we can switch the role of l and c in (5.2) and (5.4), which can be beneficial when we want to use the structure of l or L to solve the linear system.

5.1.2 Different orbital bases

In this section, we review a couple of MO bases we use in this chapter. We start with the canonical orbitals [208], which diagonalize the Fock matrix F and therefore also diagonalize F_{virt} and F_{occ}. F in this scenario stores all the eigenvalues of this chemical system. The canonical orbitals are the easiest to handle with respect to (5.4), especially for orthogonal orbitals, since Δ becomes a diag-

onal matrix, and we have a formula for each element of $\mathcal{T}$:

$$\mathcal{T}_{a,i,b,j} = \frac{2(ia|jb) - (ib|ja)}{-\epsilon_a - \epsilon_b + \epsilon_i + \epsilon_j}, \tag{5.8}$$

where ϵ_i and ϵ_j are diagonal elements of F_{occ}, and ϵ_a and ϵ_b are diagonal elements of F_{virt}. However, with canonical orbitals, the ERI matrix is dense, and we need to construct each element through the double repulsion integral formula, which is an expensive procedure both in terms of memory and computations.

Another commonly used orbital basis is the localized orbital basis, which can be obtained from the canonical orbitals by standard localization procedures as proposed by Boys [33] or Pipek and Mezey [170], and is used to span localized MOs. Then, the virtual space is spanned by a basis of nonorthogonal projected AOs, which are obtained from the AO basis functions by projecting out the occupied orbital space [173]. For example, with Boys localization, the sum of spread of each orbital is minimized, and the orbital locality is thus maximized. Figure 5.1 shows the effects of Boys localization on the object $C_{10}H_{22}$. With localization, chemical information is clustered together, leading to sparsity in the ERI matrix. As a result, many elements in L vanishes, such as when i and j are far apart, or a and b are far apart. This greatly reduces memory costs and computation power of the ERI generation.

The final orbital basis that we introduce is the resolution of the identity (RI) basis, which appears in density fitting methods [112]. In short, the one-electron charge densities in (5.3) are approximated by linear expansions in an auxiliary basis set [223, 224]. Mathematically speaking, this gives us a low rank factorization of the ERI matrix

$$L = (ia|jb) \approx (ia|A)(A|B)^{-1}(B|jb), \tag{5.9}$$

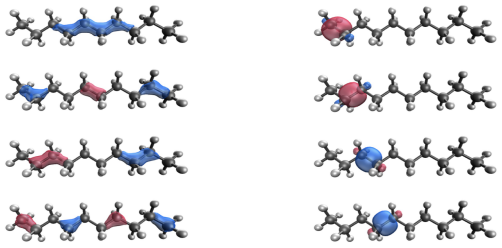

Figure 5.1: Effects of Boys localization method on $C_{10}H_{22}$. The blue and red clumps represent the canonical orbital basis (left) and the localized orbital basis (right) as distribution functions, since both are linear combinations of the AO basis, whose basis functions are commonly considered to be Gaussian distributions. We can see that with localization, each MO basis is clustered around a small region, and this helps promote sparsity in the chemical system.

where the size of the RI basis is much smaller than $n_{\text{virt}}n_{\text{occ}}$. Here, we also overload the notation a little and use $(ia|jb)$ to represent a matrix. Since L is symmetric, we can represent this low rank approximation as

$$L \approx UU^T, \tag{5.10}$$

where U is a tall-and-skinny matrix. By now, density fitting is a well-established approach, and has been applied to MP2 in canonical [223] and localized [226] representations with optimized fitting basis sets. In this chapter, we utilize density fitting by exploiting the low rank structure of L (5.10), and develop a low rank solver in a representation with the canonical orbital basis.

5.2 Sylvester equation representation

Our main contribution in this project is to adopt the dimensionality increase idea and rewrite the linear system (5.4) and (5.5) into higher-dimensional Sylvester tensor equations and then use Sylvester solvers to find t in vector, matrix, or tensor form efficiently. To be specific, since each element in Δ can be

expressed as the sum of products of elements of four matrices, we see that

$$\begin{aligned}\Delta = &- S_{\text{occ}} \otimes S_{\text{virt}} \otimes S_{\text{occ}} \otimes F_{\text{virt}} + S_{\text{occ}} \otimes S_{\text{virt}} \otimes F_{\text{occ}} \otimes S_{\text{virt}} \\ &- S_{\text{occ}} \otimes F_{\text{virt}} \otimes S_{\text{occ}} \otimes S_{\text{virt}} + F_{\text{occ}} \otimes S_{\text{virt}} \otimes S_{\text{occ}} \otimes S_{\text{virt}}.\end{aligned} \tag{5.11}$$

In this way, (5.4) can be rewritten into a 4D generalized Sylvester equation

$$\begin{aligned}&\mathcal{T} \times_1 (-F_{\text{virt}}) \times_2 S_{\text{occ}} \times_3 S_{\text{virt}} \times_4 S_{\text{occ}} + \mathcal{T} \times_1 S_{\text{virt}} \times_2 F_{\text{occ}} \times_3 S_{\text{virt}} \times_4 S_{\text{occ}} \\ &+ \mathcal{T} \times_1 S_{\text{virt}} \times_2 S_{\text{occ}} \times_3 (-F_{\text{virt}}) \times_4 S_{\text{occ}} + \mathcal{T} \times_1 S_{\text{virt}} \times_2 S_{\text{occ}} \times_3 S_{\text{virt}} \times_4 F_{\text{occ}} = \mathcal{C},\end{aligned} \tag{5.12}$$

where $\mathcal{T} = \text{reshape}(t, n_{\text{virt}}, n_{\text{occ}}, n_{\text{virt}}, n_{\text{occ}})$ and $\mathcal{C} = \text{reshape}(c, n_{\text{virt}}, n_{\text{occ}}, n_{\text{virt}}, n_{\text{occ}})$. When we want to use structures of the ERI matrix L or $C = 2L - K$, we can also rephrase (5.12) into a generalized Sylvester equation

$$\begin{aligned}&(-S_{\text{occ}} \otimes F_{\text{virt}} + F_{\text{occ}} \otimes S_{\text{virt}}) T (S_{\text{occ}} \otimes S_{\text{virt}})^T \\ &\qquad + (S_{\text{occ}} \otimes S_{\text{virt}}) T (-S_{\text{occ}} \otimes F_{\text{virt}} + F_{\text{occ}} \otimes S_{\text{virt}})^T = 2L - K = C.\end{aligned} \tag{5.13}$$

In a representation with orthogonal orbitals, we can simplify (5.12) to

$$\mathcal{T} \times_1 (-F_{\text{virt}}) + \mathcal{T} \times_2 F_{\text{occ}} + \mathcal{T} \times_3 (-F_{\text{virt}}) + \mathcal{T} \times_4 F_{\text{occ}} = \mathcal{C} \tag{5.14}$$

and (5.13) to

$$(-I \otimes F_{\text{virt}} + F_{\text{occ}} \otimes I) T + T (-I \otimes F_{\text{virt}} + F_{\text{occ}} \otimes I)^T = C. \tag{5.15}$$

In particular, since $F_{\text{virt}}, F_{\text{occ}}$, and C are all symmetric, (5.15) is also a Lyapunov equation. In this chapter, we focus on representations with orthogonal orbitals, but we mention extensions to the nonorthogonal orbital case whenever possible. Compared to solving the linear system (5.4), Sylvester equation representations avoid the construction of Δ and compute directly with Fock and overlap matrices, which significantly reduce storage and computation costs.

5.2.1 Canonical representation

When canonical MO basis is used, F_{virt} and F_{occ} are both diagonal, then there is no difference finding the solution via (5.4) or (5.14) or (5.15) as all of them collapse down to the direct scaling (5.8). Because of the one-to-one correspondence between the solution and the dense ERI matrix, ERI generation and direct evaluations take $\mathcal{O}(n_{\text{virt}}^2 n_{\text{occ}}^2)$ to accomplish.

A well-recognized approach with canonical orbital basis is the RI-MP2 method [223], which uses density fitting to help generate an approximation of the ERI matrix, and uses Laplace transform to convert the division with $d_{i,a,j,b} = -\epsilon_a - \epsilon_b + \epsilon_i + \epsilon_j$ into a finite summation. To be specific, we perform the Laplace transform

$$\frac{1}{d_{i,a,j,b}} = \int_0^\infty e^{-sd_{i,a,j,b}} ds \approx \sum_{q=1}^{N_q} w_q e^{-s_q d_{i,a,j,b}}, \tag{5.16}$$

where the approximation is a choice of a quadrature rule so that w_q represents the quadrature weights and t_q denote the discretization points. Typically in practice, $7 \leq N_q \leq 10$, so combining this truncated Laplace transform with density fitting allows us not to solve for the solution element-by-element but as a linear combination of the ERI matrix expressed with the RI basis.

The Sylvester matrix equation formulation (5.15) provides an alternative way to derive RI-MP2. Since all eigenvalues of F_{occ} and $-F_{\text{virt}}$ are negative, we can write out a closed-form solution [177] of (5.15) as

$$T = \int_0^\infty -e^{s\Lambda} C e^{s\Lambda} ds, \tag{5.17}$$

where $\Lambda = I \otimes (-F_{\text{virt}}) + F_{\text{occ}} \otimes I$. Then, it is straightforward to see that (5.17) with a numerical quadrature routine is equivalent to RI-MP2 with the approximation (5.16). Furthermore, since spectra of F_{occ} and $-F_{\text{virt}}$ are the same with

any type of basis, (5.17) remains to hold even when they are not diagonal. Therefore, one can also evaluate (5.17) numerically to obtain a solver in a representation with localized orbitals. However, the performance of this method can be greatly hindered by the computation of dense matrix exponentials.

In conclusion, existing MP2 methods with canonical orbital basis can be understood with our Sylvester equation formulation. This encourages us to develop more ways to compute electron correlation energy with Sylvester equation solvers.

5.2.2 Localized representation

When F_{occ} and $-F_{\text{virt}}$ are obtained from localized MO basis, they are dense and symmetric, so a 4D analogue of the eigen-based solver Algorithm 3 is the most straightforward algorithm for (5.14). Figure 5.2 shows a timing comparison between the eigen-based Sylvester solver and GMRES [178] to solve (5.4) on several chemical objects. We implement our Sylvester solver in MATLAB and use the MATLAB built-in GMRES routine. Table 5.1 contains the same results but we also report the exact object name and number of iterations for GMRES to converge (in parenthesis after timing). From the results, we see that using Sylvester equation solvers is more beneficial than the traditional way of solving linear systems.

Eigen-decomposition is not the dominant cost since we need to solve for every entry of the solution. As a result, this Sylvester solver has complexity $\mathcal{O}(n_{\text{virt}}^2 n_{\text{occ}}^2)$. In addition, although it shows prevalence over traditional linear system solvers, this direct eigen-based Sylvester solver is far from ideal as we

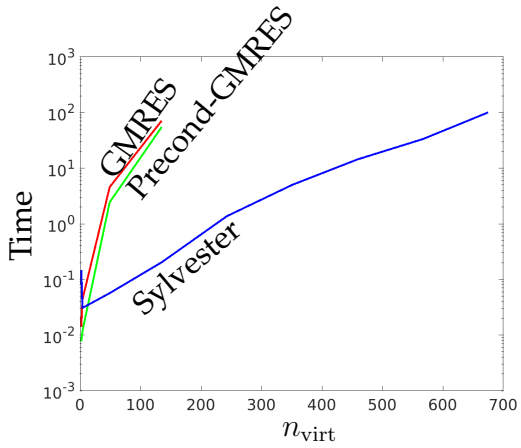

Figure 5.2: Comparison of GMRES with no preconditioner (red), GMRES with diagonal preconditioner (green) and 4D analogue of Algorithm 3 (blue) of test examples in the localized representation. The Sylvester solver is in general the fastest in small problems, and is the only solver that can handle larger problems.

Table 5.1: Summary of GMRES, preconditioned GMRES and eigen-decomposition Sylvester solver on same test objects in Figure 5.2.

Name	ch4-sto3g	C_2H_6	C_2H_6-tz	C_4H_{10}	C_6H_{14}	C_8H_{18}	$C_{10}H_{22}$	$C_{12}H_{26}$
n_{occ}	5	9	9	17	25	33	41	49
n_{virt}	4	49	135	243	351	459	567	675
GMRES	0.0262 (24)	4.4101 (42)	82.5693 (55)					
Pre-GMRES	0.0187 (7)	3.2311 (15)	60.9482 (30)					
Eigen-Sylv	0.0320	0.0488	0.1662	1.6467	5.7733	16.6457	45.4504	113.3039

neglect any structure associated with the ERI matrix. In fact, the diagonalization step tries to convert the problem back to one with canonical orbital basis, so we lose all benefits from using a localized orbital basis. Therefore, we need to design more sophisticated solvers to utilize the sparsity and data-sparsity of the ERI matrix.

In a representation with nonorthogonal orbitals, we can use a 4D analogue of the Bartels–Stewart Algorithm (Algorithm 4) to solve (5.12) directly. This algorithm also has complexity $\mathcal{O}(n_{\text{virt}}^2 n_{\text{occ}}^2)$. Similar to the orthogonal orbital case, we wish to develop solvers that exploit the structures of the ERI matrix

and achieve better complexity.

5.3 Low rank method for canonical representation

The RI-MP2 method sheds light on a solver that does not solve for each element of the solution T, but for a combination of terms obtained from the AO basis. Together with the fact that we have a low rank factorization of L from density fitting, RI-MP2 motivates us to develop a method that solves for T in a low rank representation as well.

Using the RI basis, we obtain $L = UU^T$ where the number of columns of U is roughly $3(n_{\text{virt}} + n_{\text{occ}})$ in practice. Therefore, the idea of exchanging L and C in (5.15) and (5.2) inspires us to use fADI (Algorithm 6) on the Sylvester equation

$$(-I \otimes F_{\text{virt}} + F_{\text{occ}} \otimes I)T + T(-I \otimes F_{\text{virt}} + F_{\text{occ}} \otimes I)^T = UU^T,$$

and recover the correlation energy with $E_{\text{corr}} = T \cdot C = 2T \cdot L - T \cdot K$. However, K is almost full rank, so the computation $T \cdot K$ takes $\mathcal{O}(n_{\text{virt}}^2 n_{\text{occ}}^2)$ and destroy the data sparsity we create in the matrix T. As a result, our desired solver needs to satisfy two criteria: (1) it can introduce some level of data sparsity in T, and (2) it needs to use the product with L to calculate the correlation energy.

Chemists have found that the canonical orbitals around a certain group of atoms are highly correlated, leading to low rank structures of submatrices of the ERI matrix L. For example, Figure 5.3(left) is a histogram of the numerical ranks of sub-blocks of size $n_{\text{virt}} \times n_{\text{virt}}$ of C with accuracy 10^{-5} for the chemical object C_4H_{10}. We notice that the 16 sub-blocks on the top left corner, which correspond to the carbon atoms, have low rank, and the 169 sub-blocks that correspond to

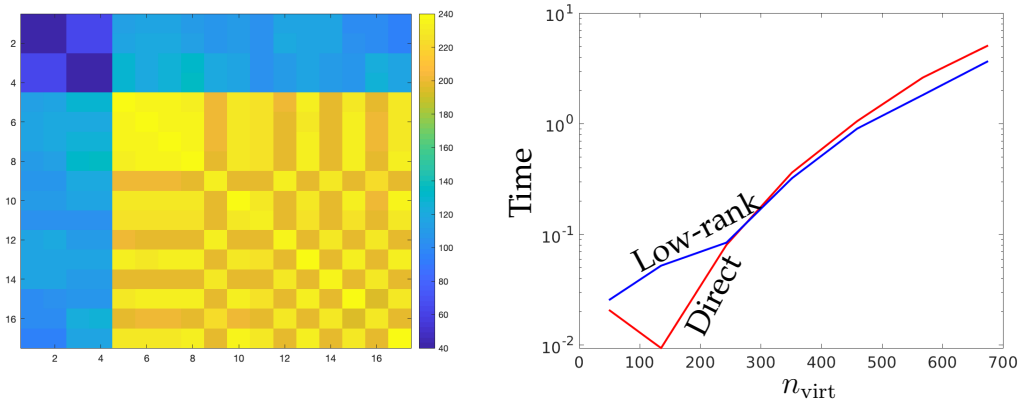

Figure 5.3: Left: The numerical rank of sub-blocks of size $n_{\text{virt}} \times n_{\text{virt}}$ of C_4H_{10}. Here, $n_{\text{virt}} = 243$ and $n_{\text{occ}} = 17$. Right: Timing comparison of the direct solver (red) and partitioned low rank solver (blue) of (5.15).

the hydrogen atoms on the bottom right corner are high-rank or full rank. This observation also holds for $n_{\text{virt}} \times n_{\text{virt}}$ sub-blocks of other chemical objects of the form C_nH_{2n+2}. In particular, we can observe n^2 low rank blocks in the top left corner.

Given this special block data sparsity structure of C, we can design a solver that separately treats low and full rank blocks. This is feasible since F_{virt} and F_{occ} are both diagonal in the canonical orbital basis, so we can partition (5.15) into an individual Sylvester equation for each sub-block:

$$(-F_{\text{virt}} + F_{i,i}I)T_{i,j} + T_{i,j}(-F_{\text{virt}} + F_{j,j}I) = C_{i,j}, \quad 1 \leq i, j \leq n_{\text{occ}}, \tag{5.18}$$

where $T_{i,j}$ and $C_{i,j}$ are the (i,j)th sub-blocks of size $n_{\text{virt}} \times n_{\text{virt}}$ of T and C respectively. For objects such as C_nH_{2n+2}, (5.18) allows one to use low rank solvers for $1 \leq i, j \leq n$, and direct method or full rank iterative method for the rest of the blocks.

We can apply fADI (Algorithm 6) to (5.18), but from practice we find that the ranks of low rank blocks $C_{i,j}$ are not very small (see Figure 5.3 (left)). As a result, using Algorithm 6 directly is not very efficient. Instead, it's straightforward to

notice that the solution $T_{i,j}$ can be computed as

$$T_{i,j} = W_{i,j} \circ C_{i,j},$$

where $\circ$ denotes the Hadamard (or entry-wise) product, and $W_{i,j}$ is the solution to the Sylvester equation

$$(-F_{\text{virt}} + F_{i,i}I)W_{i,j} + W_{i,j}(-F_{\text{virt}} + F_{j,j}I) = \mathbf{1}, \tag{5.19}$$

where $\mathbf{1}$ represents a matrix with all entries 1. In this way, (5.19) has a rank-1 right-hand-side, so solving it using Algorithm 6 is very fast. Next, suppose $W_{i,j} = \sum_{\ell=1}^{s} \sigma_\ell z_\ell h_\ell^T$ and $C_{i,j} = \sum_{k=1}^{r} \lambda_k u_k v_k^T$, where the factors z_ℓ are orthonormal, and similarly for h_ℓ, u_k, and v_k, then the solution $T_{i,j}$ can be represented as

$$T_{i,j} = \sum_{\ell=1}^{s} \sum_{k=1}^{r} \sigma_\ell \lambda_k (z_\ell \circ u_k)(h_\ell \circ v_k)^T. \tag{5.20}$$

Since $z_\ell \circ u_k$ or $h_\ell \circ v_k$ are not necessarily groups of orthonormal vectors, truncation based on the values of $\sigma_\ell \lambda_k$ does not guarantee optimal accuracy. Nevertheless, we can still apply the heuristics and truncate terms with small magnitude in this double summation. In practice, we can do two sweeps of truncation for $T_{i,j}$:

1. Truncate terms with small $\sigma_\ell \lambda_k$.
2. By Cauchy-Schwarz inequality, we can show that $||z_\ell \circ u_k|| \leq ||z_\ell|| ||u_k|| \leq 1$ and $||h_\ell \circ v_k|| \leq 1$. Therefore, for the remaining terms, we can truncate those with small $\sigma_\ell \lambda_k ||z_\ell \circ u_k|| ||h_\ell \circ v_k||$.

In this way, we take away as many terms as possible, while limiting the number of computations for vector norms. We summarize this method in Algorithm 12. This algorithm only considers Sylvester equations with diagonal matrices, and

is very similar to the fiADI algorithm [211]. Finally, we want to mention that for computational efficiency, we do not compress our final result since we do not necessarily need the factor matrices to have orthonormal columns. All we care about are low rank factorizations of the blocks, so matrix multiplications with blocks of L to compute the correlation energy are cheaper.

Algorithm 18 Low rank method for diagonal Sylvester equation of the form $LX + XR^T = U\Lambda V^T$, where the right-hand-side has a rank of moderate size.

Input: Diagonal matrices L, R, and Λ, matrices with orthonormal columns U and V, and accuracy ϵ

Output: Low rank factorization of solution matrix X

1: Use Algorithm 6 to solve $LW + WR = 1$ for $W = \sum_{\ell=1}^{s} \sigma_\ell z_\ell h_\ell^T$.
2: For all ℓ and k, form $a_{\ell,k} = z_\ell \circ u_k$ and $b_{\ell,k} = h_\ell \circ v_k$ that has $\sigma_\ell \lambda_k > \epsilon$. Denote the index set by $(\tilde{\ell}, \tilde{k})$.
3: For all index pairs $(\tilde{\ell}, \tilde{k})$, keep only terms with $\sigma_{\tilde{\ell}} \lambda_{\tilde{k}} ||a_{\tilde{\ell},\tilde{k}}|| ||b_{\tilde{\ell},\tilde{k}}|| > \epsilon$. Denote the index set by $(\hat{\ell}, \hat{k})$.
4: The solution $X = \sigma_{\hat{\ell}} \lambda_{\hat{k}} ||a_{\hat{\ell},\hat{k}}|| ||b_{\hat{\ell},\hat{k}}||$.

Fig 5.3 (Right) shows the timing result of the direct scaling method and this partitioned low rank solver on test problems in Table 5.1, represented with canonical orbital basis. We notice that as the problem size gets larger, the partitioned low rank solver has better performance. Suppose for notational simplicity that $n_{\text{virt}} = n$, then Algorithm 18 takes $\mathcal{O}(n \log n \log(1/\epsilon) + rsn)$ for each low rank sub-block if no truncation is applied in the algorithm. With near-optimal truncation, this algorithm takes roughly $\mathcal{O}(n \log n \log(1/\epsilon))$ efforts. As a result, the block partitioned solver has a better complexity than the direct scaling method, and works better in practice, especially for large problems.

The practicality of this algorithm is limited by the construction of low rank factors of blocks $C_{i,j}$ at this stage. For testing purposes, we manually compute low rank factorizations of $C_{i,j}$ and carry out our algorithm. In the future, we plan to incorporate hierarchical matrix and fast multipole method (FMM)

ideas [87, 227] to generate a low rank factorization of $C_{i,j}$ directly during evaluations of the ERI matrix. This allows us to use Algorithm 12 in real applications.

5.4 Sparsity enforcement method for localized representation

The most important feature of localized orbital basis is the introduction of sparsity in the ERI matrix. For moderate-sized test problem, we see in practice the localized ERI matrix have $\mathcal{O}(N^2)$ number of nonzeros, and this can reach $\mathcal{O}(N)$ for even larger problems. For example, Figure 5.4 (left) shows the portion of nontrivial entries with different threshold in the ERI matrix of $C_{20}H_{42}$ associated with different localized orbital bases [222]. This motivates us to develop a method to use the sparsity pattern of the ERI matrix, and optimally, we want the solution T to have the same sparsity level as C, which is easily determined by that of L.

Since (5.2) is only a dot product between the solution T and the ERI matrix L, T has the optimal sparsity pattern if its positions of nonzero entries form a subset of those of C. Unfortunately, this is rarely possible as the solution of a sparse linear system generally has a different sparsity pattern from that of the right-hand-side. However, in the linear system (5.4), if the ℓth row and column of Δ contain all zeros when the ℓth element of c is zero, then we can see that the ℓth element of the solution t is also zero. This inspires us to design a sparsity-enforced solver for (5.4) by manually setting certain columns and rows of Δ to be 0, so that nontrivial elements of t and c have the same indices. Undoubtedly, we lose some accuracy by setting nonzero elements of t to be 0, but we gain great computation efficiency. For example, the blue line of Figure 5.4 (right) shows the

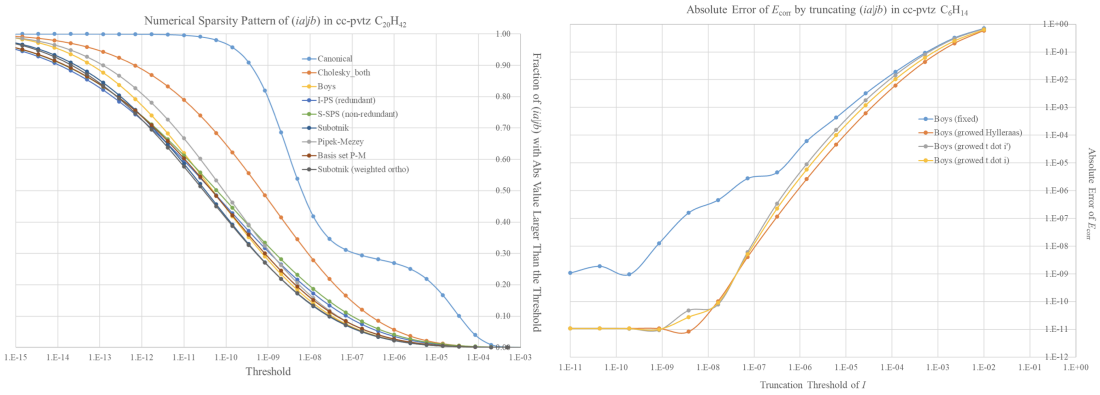

Figure 5.4: Left: The number of entries greater than a certain threshold of the ERI matrix associated with $C_{20}H_{42}$ in a variety of localized orbital basis. We can see that for this chemical object, the majority of the elements in the ERI matrix can be considered trivial. Right: The accuracy of the computed electron correlation energy when elements of L that are below a threshold, and corresponding rows and columns of Δ in (5.4) are set to 0 for the test example C_6H_{14} (blue line). For example, if we want the error of the correlation energy to be smaller than 10^{-4}, we only need to keep elements that are greater than 10^{-6} in L, and their corresponding columns and rows in Δ.

accuracy level of this sparsity-enforced method for the chemical object C_6H_{14} when we choose a different threshold to determine which elements in L to keep. We can see that for this class of object, the loss of accuracy is acceptable.

5.4.1 Removing columns and rows in Kronecker products

In the original form of Δ in (5.11) where all rows and columns are kept, matrix-vector products Δv are equivalent to tensor mode-n products

$$\mathcal{V} \times_1 (-F_{\text{virt}}) \times_2 S_{\text{occ}} \times_3 S_{\text{virt}} \times_4 S_{\text{occ}} + \mathcal{V} \times_1 S_{\text{virt}} \times_2 F_{\text{occ}} \times_3 S_{\text{virt}} \times_4 S_{\text{occ}}$$
$$+ \mathcal{V} \times_1 S_{\text{virt}} \times_2 S_{\text{occ}} \times_3 (-F_{\text{virt}}) \times_4 S_{\text{occ}} + \mathcal{V} \times_1 S_{\text{virt}} \times_2 S_{\text{occ}} \times_3 S_{\text{virt}} \times_4 F_{\text{occ}},$$

where $\mathcal{V} = \text{reshape}(v, n_{\text{virt}}, n_{\text{occ}}, n_{\text{virt}}, n_{\text{occ}})$. With orthogonal orbital basis, the products are simply

$$\mathcal{V} \times_1 (-F_{\text{virt}}) + \mathcal{V} \times_2 F_{\text{occ}} + \mathcal{V} \times_3 (-F_{\text{virt}}) + \mathcal{V} \times_4 F_{\text{occ}}.$$

Therefore, if we use Krylov subspace methods to solve (5.4), we never need to construct Δ, and can work directly with Fock and overlap matrices.

We now consider a slightly different matrix-vector multiplication Ev, where

$$E = A \otimes B \otimes C \otimes D,$$

$A \in \mathbb{R}^{n_4 \times n_4}, B \in \mathbb{R}^{n_3 \times n_3}, C \in \mathbb{R}^{n_2 \times n_2}$, and $D \in \mathbb{R}^{n_1 \times n_1}$ are symmetric matrices. Then, each of the four components in Δ can be viewed as a special case of E. If we can still represent the matrix $\bar{E}$, which is obtained via setting certain rows and columns of E to be 0, with Kronecker products of matrices related to A, B, C, and D, then we are able to perform the multiplication $\bar{E}v$ efficiently using tensor notations.

From the Kronecker product structure, we find that each column of E can be computed with columns of A, B, C, and D, i.e.,

$$E_{h(i,a,j,b)} = A_j \otimes B_b \otimes C_i \otimes D_a,$$

where A_j, B_b, C_i, D_a, and $E_{h(i,a,j,b)}$ represents the jth, bth, ith, ath, and $h(i,a,j,b) = jn_3n_2n_1 + bn_2n_1 + in_1 + a$th column of A, B, C, D, and E respectively. Here, for notational simplicity, we use the indexing routines in C and C++ that $0 \leq a \leq n_1$, $0 \leq i \leq n_2$, $0 \leq b \leq n_3$, and $0 \leq j \leq n_4$. In this way, we can express

$$\bar{E}_{h(i,a,j,b)} = \bar{A}_j \otimes \bar{B}_b \otimes \bar{C}_i \otimes \bar{D}_a,$$

where

$$\bar{A}_j = \begin{bmatrix} 0 & \cdots & 0 & A_j & 0 & \cdots & 0 \end{bmatrix}$$

is used to represent a matrix whose jth column is the only nonzero column with entries from A_j, and similarly for $\bar{B}_b$, $\bar{C}_i$, $\bar{D}_a$, and $\bar{E}_{h(i,a,j,b)}$. Then, $\bar{E}_{h(i,a,j,b)}$ corresponds to the extreme case that only the $h(i,a,j,b)$th element of v is nontrivial, so that all columns of E except the $h(i,a,j,b)$th are manually set to 0. Similarly, since E is symmetric, $\bar{E}^T_{h(i,a,j,b)}$ corresponds to a matrix that only keeps the $h(i,a,j,b)$th row of E. In this case, one can easily check that $\bar{E}^T_{h(i,a,j,b)}v$ yields the same answer as $\bar{E}^{r,c}_{h(i,a,j,b)}v$, where $\bar{E}^{r,c}_{h(i,a,j,b)}$ is a matrix whose only nontrivial entry is at the $(h(i,a,j,b), h(i,a,j,b))$ position.

For the index of each element in the vector v, we can use the function h to find its associated i, a, j, and b values, so we can use the tuple $(iajb)$ to denote a specific index. In this way, if there are M nontrivial elements in v, we can label them as $(i_1a_1j_1b_1), \ldots, (i_Ma_Mj_Mb_M)$. In practice, we can store these M tuples according to their (ij) value pair and (ab) value pair. Specifically, we store all the unique (ij) pairs from $(i_1j_1), \ldots, (i_Mj_M)$; then for each nonempty (ij) combination, we store all pairs (ab) from $(a_1b_1), \ldots, (a_Mb_M)$ that share the same (ij) values. This storage routine is based on an efficient method to generate the ERI matrix, where double integrals are only performed on certain (ij) and (ab) pairs. In this way, our target matrix-vector multiplication Ev is transformed to

$$\bar{E}^{r,c}_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)}v,$$

where $\bar{E}^{r,c}_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)}$ is originated from E by setting columns and rows not related to the indices $(i_1a_1j_1b_1), \ldots, (i_Ma_Mj_Mb_M)$ to zero. However, its closed-form expression using A, B, C, and D is very complicated, so instead, we carry out the matrix-vector product

$$\bar{E}^T_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)}v, \tag{5.21}$$

where $\bar{E}_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)}$ keeps columns $h(i_1,a_1,j_1,b_1), \ldots, h(i_M,a_M,j_M,b_M)$

of E with all others set to zero.

5.4.2 Closed-form expressions of Kronecker product matrices with columns removed

The remaining task is to relate $\bar{E}_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)}$ with A, B, C, and D through Kronecker products, so that the multiplication $\bar{E}^T_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)}v$ can be converted into tensor mode-n products of $\mathcal{V}$. According to the storage pattern of the $(iajb)$ index tuples, we obtain

$$\begin{aligned}\bar{E}_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)} &= \sum_{k=1}^{M} \bar{A}_{j_k} \otimes \bar{B}_{b_k} \otimes \bar{C}_{i_k} \otimes \bar{D}_{a_k} \\ &= \sum_{j' \in \text{unique}\, \mathcal{J}} \bar{A}_{j'} \otimes \left(\sum_{w=1}^{\zeta_{j'}} \bar{B}_{b_w} \otimes \bar{C}_{i_w} \otimes \bar{D}_{a_w} \right), \quad (5.22)\end{aligned}$$

where $\mathcal{J}$ is a set containing $j_1, \ldots, j_M$. This representation can be understood in the following way: for each unique value j' that corresponds to nontrivial elements, we find all $(i_w a_w b_w)$ tuples that share this value j'. Therefore, we must have $\sum_{j' \in \text{unique}\, \mathcal{J}} \zeta_{j'} = M$. Representation (5.22) is used to single out the matrix A, so it is most useful when we have orthogonal orbitals with $A = F_{\text{occ}}$ and $B = C = D = I$.

Similarly, when we want to single out C for the index i, we have

$$\bar{E}_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)} = \sum_{i' \in \text{unique}\, \mathcal{I}} \left(\sum_{w=1}^{\gamma_{i'}} \bar{A}_{j_w} \otimes \bar{B}_{b_w} \otimes \bar{C}_{i'} \otimes \bar{D}_{a_w} \right), \quad (5.23)$$

where $\mathcal{I}$ contains $i_1, \ldots, i_M$. This can be understood as the same way of (5.22) and $\sum_{i' \in \text{unique}\, \mathcal{I}} \gamma_{i'} = M$.

Due to the way tuples $(iajb)$ are stored, we need slightly different representations if we want to single out the matrix D:

$$\bar{E}_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)} = \sum_w \left(\bar{A}_{j_w} \otimes \bar{B}_{b_w} \otimes \bar{C}_{i_w} \otimes \bar{D}_{(\mathcal{A}|i_w,b_w,j_w)}\right), \tag{5.24}$$

where $(\mathcal{A}|i_w, b_w, j_w)$ denotes all values of a given the choice of j_w, b_w, and i_w. This is equivalent to finding all $(i_w a_\ell j_w b_w)$ tuples with the same i_w, j_w, and b_w values and group them together. For this representation, we must have $\sum_w \alpha_w = M$ where $\alpha_w = |(\mathcal{A}|i_w, b_w, j_w)|$ and $|\cdot|$ counts the cardinality of a set. Additionally, to single out the matrix B with b indices, we have

$$\bar{E}_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)} = \sum_w \left(\bar{A}_{j_w} \otimes \bar{B}_{(\mathcal{B}|i_w,j_w,a_w)} \otimes \bar{C}_{i_w} \otimes \bar{D}_{a_w}\right). \tag{5.25}$$

For this, $\sum_w \beta_w = \sum_w |(\mathcal{B}|i_w, j_w, a_w)| = M$ holds. In the next two subsections, we show how to use (5.22), (5.23), (5.24), and (5.25) in electron correlation energy computation to develop matrix-vector multiplication routines that scale linearly with respect to the number of nonzero entries.

5.4.3 Orthogonal orbital basis

In a representation with orthogonal orbital basis, three of the four matrices A, B, C, and D are identity. In this case, many multiplication processes in $\bar{E}^T_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)} v$ when using (5.22), (5.23), (5.24), and (5.25) can be simplified to selections of sub-vectors of v. In particular, we can perform the multiplication as follows:

- When $A = B = C = I$: we use (5.24). We observe that $(\bar{I}^T_j \otimes I \otimes I \otimes I)v$ is equivalent to constructing matrix $(\bar{V}^T_{(4)})_j$, i.e., this operation results in a matrix

that preserves the jth row of $V_{(4)}$. With some reshaping, we find that it has the same nonzero elements as $(\bar{V}_{(1)})_{jn_2n_3:(j+1)n_2n_3-1}$, a matrix that preserves columns jn_2n_3 to $(j+1)n_2n_3-1$ of $V_{(1)}$. Similarly, $(I \otimes \bar{I}_b^T \otimes I \otimes I)v$ preserves columns $(\ell n_3+b)n_2$ to $(\ell n_3+b+1)n_2-1$ for $0 \leq \ell \leq n_4-1$ of $V_{(1)}$, and $(I \otimes I \otimes \bar{I}_i^T \otimes I)v$ preserves (kn_2+i)th columns for $0 \leq k \leq n_3n_4-1$ of $V_{(1)}$. Finding the intersection of these preserved columns yields the only outstanding column of $V_{(1)}$ to be its $[(jn_3+b)n_2+i]$th column, and the corresponding column in the result is to multiply D with this column. Therefore, if we take the sparsity of nontrivial a indices into consideration, the complexity of multiplying (5.24) with a vector v when $A = B = C = I$ is $\mathcal{O}(\sum_w \alpha_w^2)$. Normally, α_w is a constant, so this complexity is $\mathcal{O}(M)$ in the worst case scenario when all M indices have different j, b, and i value combinations.

- When $A = C = D = I$: we use (5.25) and do the same trick as in the case of $A = B = C = I$ to perform $\bar{E}^T_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)}v$. We find that with fixed j, i, and a index, the $[(jn_2+i)n_1+a]$th column of $V_{(3)}$ is the only outstanding one. Then, the complexity of multiplying (5.25) with a vector v when $A = C = D = I$ is $\mathcal{O}(\sum_w \beta_w^2) = \mathcal{O}(M)$.
- When $A = B = D = I$: we use (5.23). We find that with fixed j, b, and a index, the $[(jn_3+b)n_1+a]$th column of $V_{(2)}$ is the only outstanding one, and the ith element of this column in the result is to multiply $\bar{C}_i^T$ with this column. According to the way we group elements together with the same value i, their corresponding multiplications can be performed together, and we can find that the complexity of multiplying (5.23) with a vector v is $\mathcal{O}(|u(\mathcal{I})|M)$, where $|u(\mathcal{I})|$ denotes the number of unique values in $\mathcal{I}$. This is again $\mathcal{O}(M)$ since $|u(\mathcal{I})|$ can be treated as a constant.
- When $B = C = D = I$: we use (5.22) and do the same trick as in the case

of $A = B = D = I$. We find that with fixed i, b, and a index, the $[(bn_2 + i)n_1 + a]$th column of $V_{(4)}$ is the only outstanding one. Then, the complexity of multiplying (5.22) with a vector v is $\mathcal{O}(|u(\mathcal{J})|M) = \mathcal{O}(M)$.

Finally, we can sum up the results from each group and achieve the final result. Clearly, the overall complexity is $\mathcal{O}(M)$.

5.4.4 Nonorthogonal orbital basis

The situation for nonorthogonal orbital basis is more complicated, but we can still think about the multiplication $\bar{E}^T_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)}v$ using (5.24), and we can find out that fixed value of j, b, i still only preserve one nonzero column. To be specific, we carry out the multiplication in four steps:

1. Perform $y = (\bar{A}^T_j \otimes I \otimes I \otimes I)v$. Since $V_{(4)} = V_3^T$, we can compute $y = \mathrm{vec}(\bar{H}_j)$ with the nontrivial column $H_j = \sum_{j' \in \text{unique } \mathcal{J}} A_{j,j'}(V_3)_{j'}$. With a different matricization, if we partition $V_{(1)}$ into blocks each with n_2n_3 columns, then the block of Y_1 that contains column jn_2n_3 to $(j+1)n_2n_3 - 1$ is a linear combination of these submatrices of $V_{(1)}$, and all other columns of Y_1 are 0.
2. Perform $z = (I \otimes I \otimes \bar{C}^T_i \otimes I)y$. Since the index i picks $(kn_2 + i)$th columns for $0 \le k \le n_3n_4 - 1$ of Y_1, then the nonzero columns of Y_1 are those with k values $jn_3 \le k \le (j+1)n_3 - 1$. In this way, we can find that Z_1 only has n_3 nonzero columns, and each nontrivial column can be calculated as $(Z_1)_{kn_2+i} = \sum_{i' \in \text{unique } (\mathcal{I}|\mathrm{j})} C_{i,i'}(Y_1)_{kn_2+i'}$ for $jn_3 \le k \le (j+1)n_3 - 1$, where $(\mathcal{I}|j)$ contains all the i values with a fixed choice of j.
3. Perform $w = (I \otimes \bar{B}^T_b \otimes I \otimes I)z$. Since the index b picks columns $(\ell n_3 + b)n_2$ to $(\ell n_3 + b + 1)n_2 - 1$ for $0 \le \ell \le n_4 - 1$ of Z_1, then the only nonzero column

of W_1 can be computed as $(W_1)_{(jn_3+b)n_2+i} = \sum_{b' \in \text{unique}\,(\mathcal{B}|\text{j,i})} B_{b,b'}(Z_1)_{(jn_3+b')n_2+i}$, where $(\mathcal{B}|j,i)$ contains all the b values with fixed j and i.

4. The final product with $\bar{D}^T_{(a_\ell|i,b,j)}$ is a straightforward multiplication.

The computation of the first three steps to compute the specific column of W_1 can be combined into:

$$(W_1)_{(jn_3+b)n_2+i} = \sum_{b' \in \text{unique}\,(\mathcal{B}|\text{j,i})} \sum_{i' \in \text{unique}\,(\mathcal{I}|\text{j})} \sum_{j' \in \text{unique}\,\mathcal{J}} B_{b,b'} C_{i,i'} A_{j,j'} (V_1)_{(j'n_3+b')n_2+i'}.$$

This contributes to one subscript w in (5.24), and thus leads to the complexity of construction of this column to be $\mathcal{O}(|u(\mathcal{J}||(\mathcal{I}|j)||(\mathcal{B}|j,i)|\alpha_w)$. Therefore, the overall complexity of all four steps is

$$\mathcal{O}(\sum_w (|u(\mathcal{J}||(\mathcal{I}|j)||(\mathcal{B}|j,i)| + \alpha_w)\alpha_w) = \mathcal{O}(M).$$

With both orthogonal and nonorthogonal orbital basis, we derive computing routines that can calculate $\bar{\Delta}^T_{h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)} v$ where v has M nonzeros at positions $h(i_1,a_1,j_1,b_1),\ldots,h(i_M,a_M,j_M,b_M)$ in $\mathcal{O}(M)$ time. This allows us to develop fast Krylov subspace methods such as GMRES or conjugate gradient (CG) [190]. What is more, because of the Kronecker product structure (5.11) of Δ, we can obtain a triangular part of Δ by using the corresponding triangular part of the matrices in (5.11). We can use the same method to obtain the diagonal of Δ. Therefore, applying Jacobi or Gauss-Seidel preconditioners in Krylov subspace methods is equivalent to solving special diagonal or triangular 4D Sylvester equations, which are much easier than solving general Sylvester equations and have complexity $\mathcal{O}(M)$.

5.5 Conclusion and future directions

In this chapter, we introduce a Sylvester tensor equation formulation into computational quantum chemistry, and use different Sylvester equation solvers to compute electron correlation energy with MP2. In particular, we show that the eigen-decomposition direct method is much more efficient than traditional linear system solvers, even though we exploit no structures in the system. We also present a block low-rank method for canonical orbital basis, and a sparsity-enforced Krylov subspace method for localized orbital basis. Our next step is to implement the sparsity-enforced method into Q-Chem [70] and compare it with RI-MP2. We want to emphasize the effectiveness of localized orbital basis in the quantum chemistry field.

Researchers also discovered the existence of hierarchical semi-separable (HSS) structure in the Fock matrix [45] and the ERI matrix [232] with localized orbital basis when the problem size is sufficiently large. Since matrices with an HSS structure have fast linear system solvers, we plan to combine it with fADI to develop a faster solver for (5.15). In addition, we will explore the interesting direction of using HSS and sparsity enforcement simultaneously on the ERI matrix. As a result, we can bridge the gap between accuracy and efficiency in electron correlation computations.

CHAPTER 6

CHEBYSHEV COEFFICIENT APPROXIMATIONS WITH QUANTIZED TENSOR-TRAIN FORMAT

Starting from the Weierstrass approximation theorem, approximating a given function with polynomials is a well-established task in mathematical analysis. With a fixed polynomial degree N, there are two major approaches for function approximation with a single piece of polynomial, i.e., a global polynomial approximation. One way is the "value-space" approximation, where we sample the function at $N+1$ points, and use them for interpolation. The other way is the "coefficient-space" approximation, where we choose a basis for polynomials of degree $\leq N$ and estimate the function with a linear combination of these polynomials. The former method requires us to store function values on a certain grid, and researchers have shown that the QTT format of the values on a uniform grid for some special functions has low rank [111, 118]. Comparatively in the latter method, we store the coefficients in a vector, but the QTT format of it is not considered in existing literature.

Chebyshev polynomials are an important family of orthogonal polynomials in numerical analysis and scientific computing [75, 145, 176, 212], and are used extensively in quadrature rules [46], and numerical solution of integral equations [67, 68] and PDEs [210]. In this chapter[1], we introduce the QTT tensor format to store coefficients of Chebyshev polynomial approximation of uni-variate analytic functions, and provide theoretical guarantees of compressibility.

[1]This chapter is based on a project with Alex Townsend, where I derived the theories.

6.1 Chebyshev polynomial approximation of functions

For $j \geq 0$, the Chebyshev polynomial of the first kind, or simply Chebyshev polynomial, with degree j is denoted by $T_j(x)$ and defined on the interval $[-1, 1]$ by

$$T_j(x) = \cos(j \arccos x), \qquad x \in [-1, 1].$$

The family of Chebyshev polynomials is orthogonal with respect to a weighted inner product

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}} dx = \begin{cases} \pi, & i = j = 0, \\ \pi/2, & i = j \geq 1, \\ 0, & i \neq j, \end{cases}$$

so Chebyshev polynomials form a natural basis for functions on $[-1, 1]$, just as one uses Fourier series for periodic functions on $[-\pi, \pi]$.

In general, there are two ways to approximate a function $f : [-1, 1] \to \mathbb{C}$ by a polynomial of degree at most N represented in the Chebyshev polynomial basis. The first polynomial approximant with great practical value is referred to as the Chebyshev interpolant, which interpolates at $N + 1$ points $(x_0^{\text{cheb}}, f_0), \ldots, (x_N^{\text{cheb}}, f_N)$ to find p_N^{interp} in the form of

$$p_N^{\text{interp}} = \sum_{k=0}^{N} c_k T_k(x),$$

such that $p_N^{\text{interp}}(x_k^{\text{cheb}}) = f_k$ for $0 \leq k \leq N$, where

$$x_k^{\text{cheb}} = \cos\left(\frac{(N-k)\pi}{N}\right), \quad 0 \leq k \leq N,$$

are called the Chebyshev points. In practice, Chebyshev interpolant p_N^{interp} offer a quasi-optimal approximation of continuous f defined on $[-1, 1]$ with bounded total variation [212, Thm. 15.1 & 15.2], and can be computed via the discrete

Chebyshev transform, which is equivalent to the type-I discrete cosine transform [78].

The other polynomial approximant that is often used for theoretical purposes is the Chebyshev projection, which is obtained by truncating the Chebyshev series of f after $N+1$ terms. To be specific, if f is Lipschitz continuous, then it has an absolutely and uniformly convergent Chebyshev series [212, Thm. 3.1] given by $f(x) = \sum_{k=0}^{\infty} a_k T_k(x)$, and the Chebyshev projection has the form

$$p_N^{\text{proj}} = \sum_{k=0}^{N} a_k T_k(x), \tag{6.1}$$

where

$$a_k = \begin{cases} \frac{1}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx, & k = 0, \\ \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx, & k \geq 1. \end{cases} \tag{6.2}$$

If f is analytic on $[-1, 1]$ and is analytically continuable to an Berstein ellipse E_ρ with $\rho > 1$, which is the open region in the complex plane bounded by the ellipse with foci ± 1 and semiminor and semimajor axis lengths summing to ρ, then it can be shown [212, Thm. 8.1] that the coefficients satisfy

$$|a_0| \leq M, \qquad |a_k| \leq 2M\rho^{-k}, \quad k \geq 1, \tag{6.3}$$

where $|f(x)| \leq M < \infty$. This exponential decay of Chebyshev coefficients implies high compressibility if QTT format is used for storage. For example, we can approximate $f(x) = \frac{1}{1+1000x^2}$ to machine precision with a Chebyshev projection (6.1) of degree 1177. Suppose its Chebyshev coefficients are stored in a vector a, then we can find that $\mathrm{reshape}(\mathrm{a}, 11, 107)$ has numerical rank 3 with machine precision as accuracy. However, (6.3) cannot be used to prove this low rank property, and we are not aware of existing literature that provides theoretical guarantees of this property for all types of analytic functions.

6.2 Analytic functions with poles

A large family of analytic functions are those with poles in the complex plane. In [69], the author provides an alternative way to compute the Chebyshev coefficients (6.2) of a function f via Cauchy integrals, i.e.,

$$a_k = \frac{1}{\pi i} \int_C \frac{f(z)dz}{\sqrt{z^2-1}\left(z \pm \sqrt{z^2-1}\right)^k}, \qquad k \geq 0, \tag{6.4}$$

where C is a contour on and within which $f(z)$ is regular, and the sign is chosen so that $|z \pm \sqrt{z^2-1}| > 1$. This leads to an expression of formulating a_j as a sum of terms related to poles and residues of f with the residue theorem, and we can then use it to bound the QTT rank of the Chebyshev coefficients of all such functions.

6.2.1 Functions with finite simple poles

We start with the case that the function f has $K < \infty$ simple poles. Following the arguments in [69, Sec. 2], we can choose the contour C in (6.4) to be any Berstein ellipse E_ρ with $\rho > 1$ that encloses all the poles. Here, we overload the notation and use E_ρ to denote the perimeter of a Berstein ellipse. If the integral around E_ρ tends to 0 as $\rho \to \infty$ for all $k > k_0$, then (6.4) becomes

$$a_k = -2 \sum_{q=1}^{K} \frac{r_q}{\sqrt{z_q^2-1}\left(z_q \pm \sqrt{z_q^2-1}\right)^k}, \qquad k > k_0, \tag{6.5}$$

where z_q is a pole of f and r_q is its associated residue. In this way, if we can approximate f with a degree N Chebyshev projection (6.1), and without loss of generality assume that $N = k_0 k_1$ for an integer k_1, then the elements of $A =$

$\mathrm{reshape}(\mathrm{a}, \mathrm{k}_0, \mathrm{k}_1)$ can be computed as

$$A_{j,\ell} = -2 \sum_{q=1}^{K} \frac{r_q}{\sqrt{z_q^2 - 1}} \frac{1}{\left(z_q \pm \sqrt{z_q^2 - 1}\right)^{(\ell-1)k_0}} \frac{1}{\left(z_q \pm \sqrt{z_q^2 - 1}\right)^{j}}, \tag{6.6}$$

for $1 \le j \le k_0$, and $2 \le \ell \le k_1$. This ensures that A has rank at most $K+1$. We can also factor $N = \tilde{k}_0 \tilde{k}_1$ for any $\tilde{k}_0, \tilde{k}_1 > k_0$, and the coefficient matrix $A = \mathrm{reshape}(\mathrm{a}, \tilde{\mathrm{k}}_0, \tilde{\mathrm{k}}_1)$ still has rank at most $K+1$. When N is not a multiple of k_0, we can increase the degree of the Chebyshev projection to $\tilde{N}$ so that $\tilde{N} = k_0 k_1$ by introducing a few higher-degree terms. This enriches the vector of coefficients to $\tilde{a} \in \mathbb{C}^{\tilde{N}}$, but does not affect the accuracy of the function approximation. In this way, the matrix $\tilde{A} = \mathrm{reshape}(\tilde{\mathrm{a}}, \mathrm{k}_0, \mathrm{k}_1)$ is a matrix of rank at most $K+1$.

6.2.2 Functions with infinitely many simple poles

When the number of poles K is very large, such as when $K > k_0$, the coefficient A is hardly compressible. In addition, in the extreme case that the function f has infinitely many simple poles, we can update (6.5) as

$$a_k = -2 \lim_{K \to \infty} \sum_{q=1}^{K} \frac{r_q}{\sqrt{z_q^2 - 1} \left(z_q \pm \sqrt{z_q^2 - 1}\right)^{k}}, \qquad k > k_0, \tag{6.7}$$

as long as no pole lies on the contour E_ρ and the integral (6.4) approaches 0 as $\rho \to \infty$, but it's hard to compress the coefficient matrix A through its mathematical rank. In these scenarios, we need to further truncate the summation in (6.5) or (6.7) to show that A has a low numerical rank.

Since there are generally no connections between poles, we can only perform a heuristic scheme to truncate (6.5) and (6.7). When $K < \infty$, we can enumerate all the poles and residues, and we can compute the terms $b_k(z_q, r_q) =$

$\frac{r_q}{\sqrt{z_q^2-1}\left(z_q\pm\sqrt{z_q^2-1}\right)^k}$ for all $1 \le q \le K$ and $1 \le k \le N$. Since $\left|z_q \pm \sqrt{z_q^2-1}\right| = \rho > 1$ from construction, $\left|b_k(z_q, r_q)\right|$ decays to 0 as k increases, and thus we need fewer terms to accurately calculate a_k for large k. Therefore, we need to determine the number of $b_k(z_q, r_q)$ terms used for $k_0 < k \le \hat{k}_0$, where $\hat{k}_0$ is a constant such that $\frac{1}{\left|z_q\pm\sqrt{z_q^2-1}\right|^{\hat{k}_0+1}}$ is below some threshold. As a result, we calculate $\left|b_{\hat{k}_0}(z_q, r_q)\right|$ for all $1 \le q \le K$, and keep those with values larger than the chosen threshold. An approximation of the numerical rank of the coefficient matrix A is thus $\tilde{K}+1$, where $\tilde{K}$ is the number of remaining poles after truncation.

The situation for an infinite number of simple poles is more complicated, since we are unable to evaluate $\left|b_k(z_q, r_q)\right|$ for all the poles. We can sort the poles with respect to the values of $\left|\sqrt{z_q^2-1}\right|$, but truncations based on these do not guarantee accurate approximations since we do not have bounds for the residues. Therefore, we can only bound the QTT rank of the Chebyshev coefficients of some special functions with infinitely many poles. For example, authors in [68,69] consider the function

$$f(x) = \frac{s}{(s^2+1) - (s^2-1)\cos\pi(\alpha+x)},$$

where $s > 1$ and $-1 \le \alpha \le 1$ are two constants. This function has poles at

$$z_q = (2q-\alpha) \pm i\beta, \quad q = 0,\ \pm 1,\ \pm 2,\ \cdots,$$

where $\beta = \frac{1}{\pi}\cosh^{-1}\frac{s^2+1}{s^2-1}$, and the residue is $-i/2\pi$ when $\mathrm{Im}\ \mathrm{z_q} > 0$, and $i/2\pi$ when $\mathrm{Im}\ \mathrm{z_q} < 0$. This means that the magnitude of the residue is independent of the function poles, and thus we can truncate (6.7) based on $\left|\sqrt{z_q^2-1}\right|$. Specifically, [69, Table 1] shows very accurate result to compute the Chebyshev coefficients with only poles $\pm i\beta$, so the coefficient matrix of this function has rank at most 3.

6.2.3 Functions with repeated poles

So far we only consider the function f to have simple poles on the complex plane, and in this section we focus on functions with poles of order more than 1, such as

$$f(x) = \frac{c}{(x - x_1)^t},$$

where c is a constant and x_1 is then a pole of x with an integer order t. We further assume that x_1 is not on $[-1, 1]$. Instead of working directly with the Chebyshev coefficients of f, we look at an alternative function $\tilde{f}(x)$ of the form

$$\tilde{f}(x) = \frac{c}{(x - x_1 - \epsilon_1) \cdots (x - x_1 - \epsilon_t)},$$

where $\epsilon_1, \ldots, \epsilon_t$ are some numbers such that $|\epsilon_1|, \ldots, |\epsilon_t| < \epsilon$ for a threshold $0 < \epsilon < 1$, and $x_1 + \epsilon_1, \ldots, x_1 + \epsilon_t \notin [-1, 1]$. $\tilde{f}(x)$ can be viewed as a perturbed version of $f(x)$ in the sense that we obtain the poles of $\tilde{f}(x)$ from those of $f(x)$ through small perturbations. This scheme of generating approximations through perturbations has been widely used in numerical linear algebra and scientific computing. In this way, we can bound

$$||f - \tilde{f}||_\infty \leq \max_{x \in [-1,1]} \left| 1 - \frac{(x - x_1)^t}{(x - x_1 - \epsilon_1) \cdots (x - x_1 - \epsilon_t)} \right| ||f||_\infty.$$

Since ϵ_ℓ can be made arbitrarily small for all $1 \leq \ell \leq t$, we have $\left| \frac{\epsilon_\ell}{x - x_1} \right| < 1$, and thus we can Taylor expand

$$\frac{x - x_1}{x - x_1 - \epsilon_\ell} = \frac{1}{1 - \frac{\epsilon_\ell}{x - x_1}} = \sum_{j=0}^{\infty} \left(\frac{\epsilon_\ell}{x - x_1} \right)^j.$$

This means that if $|\epsilon_\ell| \leq \epsilon$ for all $1 \leq \ell \leq t$, we have

$$\prod_{\ell=1}^{t} \frac{x - x_1}{x - x_1 - \epsilon_\ell} = \prod_{\ell=1}^{t} \sum_{j=0}^{\infty} \left(\frac{\epsilon_\ell}{x - x_1} \right)^j = 1 + \sum_{\ell=1}^{t} \frac{\epsilon_\ell}{x - x_1} + \mathcal{O}(\epsilon^2),$$

so that

$$\max_{x\in[-1,1]}\left|1-\prod_{\ell=1}^{t}\frac{x-x_1}{x-x_1-\epsilon_\ell}\right|\leq\left|\sum_{\ell=1}^{t}\frac{\epsilon_\ell}{x-x_1}\right|\leq t\frac{\epsilon}{|x-x_1|}.$$

This indicates that $\tilde{f}$ is an accurate approximation of f. Therefore, we can use Chebyshev coefficients of $\tilde{f}(x)$ to approximate those of $f(x)$. As a result, we transform the problem back to a function with finitely many simple poles, and we can use arguments in Section 6.2.1 for this situation.

6.3 Functions regular except at ±1 and with branch points on the real axis

From the numerators of the terms in (6.5), we can understand that the poles of an analytic function contribute much to the exponential decay (6.3), and thus result in compressibility in the coefficient matrix storage. However, this connection is unclear for other types of singularities of an analytic function, such as irregularity at ±1 and branch points on the real line. To evaluate the Cauchy integral (6.4) for these types of analytic functions, we need to choose other contours, and thus end up with different formulations for the Chebyshev coefficients.

We start with functions of the form $f(x)=(1-x)^\phi g(x)$, where $\phi>0$ is not an integer and $g(x)$ is regular at $x=1$. From [69, Sec. 6], we obtain approximations of the Chebyshev coefficients

$$a_k\approx-\frac{2^{1-\phi}g(1)\sin(\pi\phi)}{\pi k^{2\phi+1}}\Gamma(2\phi+1), \tag{6.8}$$

for all $k>k_0$, which shows a geometric decay in the Chebyshev coefficients. Similarly to our strategy in Section 6.2.1, we can express $k=(\ell-1)k_0+j$ where

j and ℓ are row and column indices of the coefficient matrix A. For special values of ϕ such as when $m = 2\phi + 1$ is an integer, we can obtain a Laurent series expansion

$$\frac{1}{((\ell-1)k_0+j)^m} = \sum_{r=0}^{\infty}(-1)^r\binom{m+r-1}{r}\frac{j^r}{(\ell-1)^{m+r}k_0^{m+r}}, \tag{6.9}$$

at $k_0 = \infty$ as long as $\left|\frac{j}{(\ell-1)k_0}\right| < 1$. With a prescribed threshold, we can truncate (6.9) to a finite summation, and obtain the numerical rank of the coefficient matrix A. Similarly, if $f(x) = (1+x)^\psi h(x)$, where $\psi > 0$ is not an integer and $h(x)$ is regular at $x = -1$, we have

$$a_k \approx -\frac{2^{1-\psi}h(-1)\sin(\pi\phi)}{\pi k^{2\psi+1}}(-1)^k\Gamma(2\psi+1), \tag{6.10}$$

and can follow the same process to obtain a low numerical rank approximation of the coefficient matrix when $2\psi+1$ is an integer. Unfortunately, we don't have results for other values of ϕ and ψ, so we leave these scenarios as a future target.

The analysis of functions with branch points on the real axis is very similar to that of functions with irregularity at ± 1. To be specific, assuming $c > 0$ is a constant, and $\phi, \psi > -1$ are any number, we can find that [69, Section. 7]

$$a_k \approx -\frac{2\sin(\pi\phi)(c^2-1)^{\phi/2}g(c)\Gamma(\phi+1)}{\pi k^{\phi+1}(c+\sqrt{c^2-1})^k}, \tag{6.11}$$

for all $k > k_0$ if $f = (c-x)^\phi g(x)$ and $g(x)$ is regular at $x = c$, and

$$a_k \approx -\frac{2\sin(\pi\psi)(c^2-1)^{\psi/2}g(-c)(-1)^k\Gamma(\psi+1)}{\pi k^{\psi+1}(c+\sqrt{c^2-1})^k}, \tag{6.12}$$

if $f = (c+x)^\psi h(x)$ and $h(x)$ is regular at $x = -c$. In both cases, the coefficients a_k contain a geometric decay component and an exponential decay component. Therefore, in special cases that ϕ and ψ are integers, the coefficient matrix A can be represented as the Hadamard product of two low rank matrices, one

generated from the term $k^{-(\phi+1)}$ or $k^{-(\psi+1)}$ as in functions irregular at ± 1, and the other generated from the term $(c+\sqrt{c^2-1})^{-k}$ as in functions with simple poles. In this way, A is itself a matrix of low numerical rank.

6.4 Entire functions

Finally, we consider functions that do not have singularities throughout the entire complex domain. Since there is no closed-form formula or approximation for Chebyshev coefficients of any entire function, we cannot conclude that all entire functions have low QTT rank. However, we show below two examples whose coefficient matrices are numerically low rank.

We first consider the exponential function $f(x) = e^x$. With some computations, we find it has a Chebyshev series expansion

$$f(x) = e^x = I_0(1)T_0(x) + 2\sum_{k=1}^{\infty} I_k(1)T_k(x),$$

where $I_k(x)$ is the kth modified Bessel function of the first kind. Since $I_k(x)$ decays with k for a fixed value of x, we can represent the three-term recurrence relation [158, (10.29.1)]

$$I_{k-1}(x) - I_{k+1}(x) = (2k/x)I_k(x),$$

as

$$\begin{bmatrix} 0 & -1 & & & & \\ 1 & -2/x & -1 & & & \\ & 1 & -4/x & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2(N-1)/x \end{bmatrix} \begin{bmatrix} I_0(x) \\ I_1(x) \\ I_2(x) \\ \vdots \\ I_{N-1}(x) \end{bmatrix} = \begin{bmatrix} -I_1(x) \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{6.13}$$

where the term $I_N(x)$ can be omitted for the last relation since it's very small. Specifically for $x = 1$, denoting $a = \begin{bmatrix} I_0(x) & \cdots & I_{N-1}(x) \end{bmatrix}^T$ and $b = \begin{bmatrix} -I_1(x) & 0 & \cdots & 0 \end{bmatrix}^T$, and assuming $N = n_1 n_2$, we can rewrite (6.13) into

$$(I_{n_2} \otimes T + D \otimes I_{n_1})a = b, \tag{6.14}$$

where $T = \begin{bmatrix} 0 & -1 & & & & \\ 1 & -2 & -1 & & & \\ & 1 & -4 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2(n_1-1) \end{bmatrix}$ and $D = \begin{bmatrix} 0 & & & \\ & -2n_1 & & \\ & & \ddots & \\ & & & -(n_2-1)n_1 \end{bmatrix}$.
Then (6.14) can be transformed into a displacement structure for $A = \mathrm{reshape(a, n_1, n_2)}$ and $B = \mathrm{reshape(b, n_1, n_2)}$:

$$TA + AD = B. \tag{6.15}$$

Since T is skew-symmetric with non-positive elements on the diagonal, all its eigenvalues are complex with non-positive real parts, which are separated from the eigenvalues of $-D$. As B has rank 1, we know from (1.15) that A is numerically low rank. Note that there is a tiny difference between A and the actual coefficient matrix A' of $f(x) = e^x$ due to the half scaling in the first term, but A' can still be approximated by a low rank matrix.

The other example that we consider is the trignometric function $f(x) = \sin(x) + \cos(x)$, which has a Chebyshev series expansion

$$f(x) = \sin(x) + \cos(x) = J_0(1)T_0(x) + 2\sum_{k=1}^{\infty}(-1)^{\lfloor k/2 \rfloor} J_k(1)T_k(x),$$

where $J_k(x)$ is the kth Bessel function of the first kind. Since we can store the patterns of the signs separately, we can use the recurrence relation [158, (10.6.1)]

to derive a displacement structure for the Bessel functions

$$\tilde{T}Y + YD = W, \tag{6.16}$$

where we consider $N = n_1 n_2$ Bessel function terms, D is the same as in (6.14),

$$Y = \text{reshape}(\begin{bmatrix} J_0(x) & \cdots & J_{N-1}(x) \end{bmatrix}, \text{n}_1, \text{n}_2),$$

$$W = \text{reshape}(\begin{bmatrix} -J_1(x) & 0 & \cdots & 0 \end{bmatrix}, \text{n}_1, \text{n}_2),$$

and

$$\tilde{T} = \begin{bmatrix} 0 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2(n_1-1) \end{bmatrix}.$$

In practice, we find all but one eigenvalues of $\tilde{T}$ are negative, with the positive eigenvalue around 0.44. As a result, there is an overlap between the interval I_1 that contains the spectra of $\tilde{T}$ and the interval I_2 that contains the spectra of D, but as N gets larger, the length of the overlap becomes comparatively much smaller than those of I_1 and I_2. Therefore, we also observe that Y is numerically low rank, which leads to a low rank approximation of the magnitudes of the Chebyshev coefficient matrix of $f(x) = \sin(x) + \cos(x)$.

6.5 Conclusion and future directions

In this chapter, we provide an alternative way that stores Chebyshev coefficients of a function approximation in the matrix format. Using expressions and approximations of Chebyshev coefficients through Cauchy integral formula, we

provide theoretical guarantees that for different types of univariate analytic functions, these coefficient matrices are mostly low rank. As a result, this QTT format of coefficient storage reduce the storage costs in numerical implementations.

Since the majority of the arguments are constructive, we plan to develop algorithms that directly compute the coefficients in the QTT format. To use the more efficient Chebyshev interpolants in practice, we will also connect these coefficient matrices with QTT format and design practical computing algorithms. We can then use these algorithms in Chebyshev spectral methods to solve PDEs, and fast evaluations of convolutions with special functions. Furthermore, with a better understanding of the geometric decay presented in Section 6.3, we will study the behaviors of QTT format of Chebyshev coefficients of differentiable functions, which have a geometric decay rate with respect to total variation and the smoothness of the function [212, Chpt. 7]. This enables us to further use QTT Chebyshev approximations in applications.

CHAPTER 7

CONCLUSIONS

The original motivation of this thesis was to explore the use of dimensionality reduction to high-dimensional problems. Due to the extensive use of tensors and low rank tensor formats in applications, in Chapter 2 and 3, we study closely on the compression rate of some tensor formats applied to certain families of tensors, and develop alternative decomposition algorithms for these formats. In Chapter 2, we try to answer some challenging yet interesting questions such as "Why are so many tensors compressible in computational mathematics?" and "What makes these tensors compressible?". In particular, we examine tensors that are generated by discretizing smooth functions, and that satisfy a specific algebraic relation called the displacement structure. We provide bounds on the compressibility of these tensors, and thus partially explaining the abundance of low rank tensors. In Chapter 3, we attempt to further develop the low rank TT format as a dimensionality reduction technique by connecting it with the blossoming field of high-performance computing. We discover the relations between different unfoldings of a tensor, and use this property as the cornerstone of our parallel TT decomposition algorithms, whose performance is illustrated by numerical experiments on high-dimensional structured tensors.

When we encountered the QTT tensor format and the Cafarelli–Silvestre extension for solving fractional PDEs, we realized the effectiveness of using dimensionality increase on low-dimensional problems, so we enriched this thesis to discuss dimensionality manipulations. We wish to offer a perspective that the dimensionality of a problem is not fixed, and we can often convert the original problem into one posed in a different dimension that might be more efficient

to solve. In this thesis, we explore two dimensionality increase approaches: (1) conversion of vectors, matrices, and tensors in linear algebra, and (2) change of domain geometries in solving fractional PDEs. In Chapter 6, we introduce the QTT format to Chebyshev polynomial approximations of analytic functions, so that the Chebyshev coefficients are stored in tensors rather than vectors for storage cost reduction. In Chapter 4, we rewrite nonlocal fractional operators on squares and disks to local operators on hexahedrons and cylinders, and we develop spectral solvers for fractional PDEs using tensor equation solvers.

An important topic that appears throughout this thesis is Sylvester equation solvers. Dimensionality increase originates from the fact that Sylvester equations can be treated as a transformation from linear systems to tensor equations. Sylvester equations normally show up in tensor displacement structures and discretizations of certain PDEs, but we also find them in computational quantum chemistry for electron correlation energy computation. This interdisciplinary connection is discussed in Chapter 5, where we use Sylvester equation solvers and related Kronecker product structures to design algorithms to compute correlation energy in canonical and localized orbital basis. In addition, dimensionality reduction arises from low rank Sylvester solvers. In Chapter 2 and 3, we extend fADI to solve Sylvester tensor equations in the TT tensor format. Our new algorithms have an optimal complexity, and illustrate efficiency when solving Poisson equations on a cube with Dirichlet boundary conditions.

Many challenges remain for tensor formats and Sylvester equations. We are curious about block low rank tensor formats such as extensions of hierarchical off-diagonal low rank (HODLR) or hierarchical semiseparable (HSS) structures for matrices, and wonder if such representations are useful in PDE solvers and

other applications. We are also interested in efficient solvers for generalized Sylvester tensor equations and nonlinear tensor equations such as algebraic Riccati equations. Furthermore, we pose some general yet very challenging questions with respect to dimensionality manipulations. For example, instead of case-by-case analysis, we want to find a systematic way to carry out dimensionality reduction or increase for any classes of problems. We also want to find patterns of problems suitable for dimensionality manipulations. All these problems present tantalizing prospects for future work.

BIBLIOGRAPHY

[1] Evrim Acar, Canan Aykut-Bingol, Haluk Bingol, Rasmus Bro, and Bülent Yener. Multiway analysis of epilepsy tensors. *Bioinf.*, 23(13):i10–i18, 2007.

[2] Evrim Acar, Seyit A Camtepe, Mukkai S Krishnamoorthy, and Bülent Yener. Modeling and multiway analysis of chatroom tensors. In *Int. Conf. Intell. Secur. Inform.*, pages 256–268. Springer, 2005.

[3] Mark Ainsworth and Christian Glusa. Hybrid finite element–spectral method for the fractional Laplacian: Approximation theory and efficient solver. *SIAM J. Sci. Comput.*, 40(4):A2383–A2405, 2018.

[4] Mark Ainsworth and Zhiping Mao. Analysis and approximation of a fractional Cahn–Hilliard equation. *SIAM J. Numer. Anal.*, 55(4):1689–1718, 2017.

[5] Naum Il'ich Akhiezer. *Elements of the theory of elliptic functions*, volume 79. Amer. Math. Soc., 1990.

[6] Jan Almlöf. Elimination of energy denominators in Møller—Plesset perturbation theory by a Laplace transform approach. *Chem. Phys. Lett.*, 181(4):319–320, 1991.

[7] Harbir Antil and Sören Bartels. Spectral approximation of fractional PDEs in image processing and phase field modeling. *Comput. Methods Appl. Math.*, 17(4):661–678, 2017.

[8] Harbir Antil, Zichao Wendy Di, and Ratna Khatri. Bilevel optimization, deep learning and fractional Laplacian regularization with applications in tomography. *Inverse Probl.*, 36(6):064001, 2020.

[9] Harbir Antil, Johannes Pfefferer, and Sergejs Rogovs. Fractional operators with inhomogeneous boundary conditions: Analysis, control, and discretization. *Commun. Math. Sci.*, 16:1395–1426, 2018.

[10] Athanasios C Antoulas. *Approximation of large-scale dynamical systems*. SIAM, 2005.

[11] Woody Austin, Grey Ballard, and Tamara G Kolda. Parallel tensor compression for large-scale scientific data. In *2016 IEEE Int. Parallel Distrib. Process. Symp.*, pages 912–922. IEEE, 2016.

[12] Philippe Y Ayala and Gustavo E Scuseria. Linear scaling second-order Møller–Plesset theory in the atomic orbital basis for large molecular systems. *J. Chem. Phys.*, 110(8):3660–3671, 1999.

[13] Ivo Babuška and BQ Guo. The h, p and hp version of the finite element method; basis theory and applications. *Adv. Eng. Software*, 15(3-4):159–174, 1992.

[14] Brett W Bader and Tamara G Kolda. Efficient MATLAB computations with sparse and factored tensors. *SIAM J. Sci. Comput.*, 30(1):205–231, 2008.

[15] Jonas Ballani and Lars Grasedyck. A projection method to solve linear systems in tensor format. *Numer. Lin. Alg. Appl.*, 20(1):27–43, 2013.

[16] Grey Ballard, Alicia Klinvex, and Tamara G Kolda. TuckerMPI: a parallel C++/MPI software package for large-scale data compression via the Tucker tensor decomposition. *ACM Trans. Math. Soft.*, 46(2):1–31, 2020.

[17] Grey Ballard, Nicholas Knight, and Kathryn Rouse. Communication lower bounds for matricized tensor times Khatri–Rao product. In *2018 IEEE Int. Parallel Distrib. Process. Symp.*, pages 557–567. IEEE, May 2018.

[18] Grey Ballard and Kathryn Rouse. General memory-independent lower bound for MTTKRP. In *Proc. 2020 SIAM Conf.Parallel Process. Sci. Comput.*, pages 1–11. SIAM, January 2020.

[19] Lehel Banjai, Jens M Melenk, Ricardo H Nochetto, Enrique Otárola, Abner J Salgado, and Christoph Schwab. Tensor FEM for spectral fractional diffusion. *Found. Comput. Math.*, 19(4):901–962, 2019.

[20] Richard H. Bartels and George W Stewart. Solution of the matrix equation $AX + XB = C$. *Commun. ACM*, 15(9):820–826, 1972.

[21] Rodney J Bartlett. Coupled-cluster theory and its equation-of-motion extensions. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.*, 2(1):126–138, 2012.

[22] Rodney J Bartlett and Monika Musiał. Coupled-cluster theory in quantum chemistry. *Rev. Mod. Phys.*, 79(1):291, 2007.

[23] Kim Batselier, Andrzej Cichocki, and Ngai Wong. Meracle: constructive

layer-wise conversion of a tensor train into a mera. *Commun. Appl. Math. Comput.*, pages 1–23, 2020.

[24] Bernhard Beckermann and Alex Townsend. Bounds on the singular values of matrices with displacement structure. *SIAM Rev.*, 61(2):319–344, 2019.

[25] Christian F Beckmann and Stephen M Smith. Tensorial extensions of independent component analysis for multisubject FMRI analysis. *Neuroimage*, 25(1):294–311, 2005.

[26] Peter Benner, Sergey Dolgov, Akwum Onwunta, and Martin Stoll. Low-rank solution of an optimal control problem constrained by random Navier-Stokes equations. *Int. J. Numer. Methods Fluids*, 92(11):1653–1678, 2020.

[27] Peter Benner, Ren-Cang Li, and Ninoslav Truhar. On the ADI method for Sylvester equations. *J. Comput. Appl. Math.*, 233(4):1035–1045, 2009.

[28] Peter Benner and Enrique S Quintana-Ortí. Solving stable generalized Lyapunov equations with the matrix sign function. *Numerical Algorithms*, 20(1):75–100, 1999.

[29] Austin R. Benson, David F. Gleich, and James Demmel. Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures. In *2013 IEEE Int. Conf. Big Data*, pages 264–272, 2013.

[30] Gregory Beylkin and Martin J Mohlenkamp. Numerical operator calculus in higher dimensions. *Proc. Natl. Acad. Sci.*, 99(16):10246–10251, 2002.

[31] Gregory Beylkin and Martin J Mohlenkamp. Algorithms for numerical analysis in high dimensions. *SIAM J. Sci. Comput.*, 26(6):2133–2159, 2005.

[32] Andrea Bonito and Joseph Pasciak. Numerical approximation of fractional powers of elliptic operators. *Math. Comp.*, 84(295):2083–2110, 2015.

[33] SF Boys. Quantum theory of atoms, molecules, and the solid state. *Academic Press, New York, NY*, page 253, 1966.

[34] Dietrich Braess. *Nonlinear approximation theory*, volume 7. Springer Science & Business Media, 2012.

[35] Dietrich Braess and Wolfgang Hackbusch. On the efficient computation of high-dimensional integrals and the approximation by exponential sums. In *Multiscale, nonlinear and adaptive approximation*, pages 39–74. Springer, 2009.

[36] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl. Acad. Sci.*, 113(15):3932–3937, 2016.

[37] Luis Caffarelli and Luis Silvestre. An extension problem related to the fractional Laplacian. *Comm. Part. Diff. Eqs.*, 32(8):1245–1260, 2007.

[38] Lynn Elliot Cannon. *A cellular computer to implement the Kalman filter algorithm*. Montana State University, 1969.

[39] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3):283–319, 1970.

[40] Tony F Chan. Rank revealing QR factorizations. *Lin. Alg. Appl.*, 88:67–82, 1987.

[41] Maolin Che and Yimin Wei. Randomized algorithms for the approximations of Tucker and the tensor train decompositions. *Adv. Comput. Math.*, 45(1):395–428, 2019.

[42] Sheng Chen and Jie Shen. An efficient and accurate numerical method for the spectral fractional Laplacian equation. *J. Sci. Comput.*, 82(1):1–25, 2020.

[43] Zhongming Chen, Kim Batselier, Johan AK Suykens, and Ngai Wong. Parallelized tensor train learning of polynomial classifiers. *IEEE Trans. Neural Networks Learn. Syst.*, 29(10):4621–4632, 2017.

[44] Peter A Chew, Brett W Bader, Tamara G Kolda, and Ahmed Abdelali. Cross-language information retrieval using PARAFAC2. In *Proc. 13th ACM SIGKDD*, pages 143–152, 2007.

[45] Jocelyn T. Chi. A scalable algorithm for approximate generalized eigenspaces of Fock Hamiltonian matrices. Technical report, Lawrence Berkeley National Lab, 2019.

[46] Charles W Clenshaw and Alan R Curtis. A method for numerical integration on an automatic computer. *Numer. Math.*, 2(1):197–205, 1960.

[47] Peter Constantin and Jiahong Wu. Behavior of solutions of 2D quasi-geostrophic equations. *SIAM J. Math. Anal.*, 30(5):937–948, 1999.

[48] Eduardo Corona, David Gorsich, Paramsothy Jayakumar, and Shravan Veerapaneni. Tensor train accelerated solvers for nonsmooth rigid body dynamics. *Appl. Mech. Rev.*, 71(5), 2019.

[49] Eduardo Corona, Abtin Rahimian, and Denis Zorin. A tensor-train accelerated solver for integral equations in complex geometries. *J. Comput. Phys.*, 334:145–169, 2017.

[50] Hussam Al Daas, Grey Ballard, and Peter Benner. Parallel algorithms for tensor train arithmetic. *SIAM J. Sci. Comput.*, 44(1):C25–C53, 2022.

[51] Hussam Al Daas, Grey Ballard, Paul Cazeaux, Eric Hallman, Agnieszka Miedlar, Mirjeta Pasha, Tim W Reid, and Arvind K Saibaba. Randomized algorithms for rounding in the tensor-train format. *arXiv preprint arXiv:2110.04393*, 2021.

[52] Lieven De Lathauwer and Bart De Moor. From matrix to tensor: Multilinear algebra and signal processing. In *Institute of mathematics and its applications conference series*, volume 67, pages 1–16. Citeseer, 1998.

[53] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000.

[54] Lieven De Lathauwer and Joos Vandewalle. Dimensionality reduction in higher-order signal processing and rank-$(R_1, R_2, \ldots, R_N)$ reduction in multilinear algebra. *Lin. Alg. Appl.*, 391:31–55, 2004.

[55] Arturo de Pablo, Fernando Quirós, Ana Rodríguez, and Juan Luis Vázquez. A fractional porous medium equation. *Adv. Math.*, 226(2):1378–1409, 2011.

[56] Maarten De Vos, Lieven De Lathauwer, Bart Vanrumste, Sabine Van Huffel, and Wim Van Paesschen. Canonical decomposition of ictal scalp EEG and accurate source localisation: Principles and simulation study. *Comput. Intell. Neurosci.*, 2007, 2007.

[57] Eleonora Di Nezza, Giampiero Palatucci, and Enrico Valdinoci. Hitchhiker's guide to the fractional Sobolev spaces. *Bull. Sci. Math.*, 136(5):521–573, 2012.

[58] Sergey Dolgov and Boris Khoromskij. Two-level QTT-Tucker format for optimized tensor calculus. *SIAM J. Matrix Anal. Appl.*, 34(2):593–623, 2013.

[59] Sergey Dolgov, Boris Khoromskij, and Dmitry Savostyanov. Superfast fourier transform using QTT approximation. *J.Fourier Anal. Appl.*, 18(5):915–953, 2012.

[60] Sergey Dolgov and Martin Stoll. Low-rank solution to an optimization problem constrained by the Navier–Stokes equations. *SIAM J. Sci. Comput.*, 39(1):A255–A280, 2017.

[61] Sergey V Dolgov. TT-GMRES: solution to a linear system in the structured tensor format. *Russ. J. Numer. Anal. Math. Model.*, 28(2):149–172, 2013.

[62] Sergey V Dolgov and Dmitry V Savostyanov. Alternating minimal energy methods for linear systems in higher dimensions. *SIAM J. Sci. Comput.*, 36(5):A2248–A2271, 2014.

[63] Tobin A Driscoll, Nicholas Hale, and Lloyd N Trefethen. Chebfun guide, 2014.

[64] Vladimir Druskin and Valeria Simoncini. Adaptive rational Krylov subspaces for large-scale dynamical systems. *Syst. Control Lett.*, 60(8):546–560, 2011.

[65] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[66] Virginie Ehrlacher, Laura Grigori, Damiano Lombardi, and Hao Song. Adaptive hierarchical subtensor partitioning for tensor compression. *SIAM J. Sci. Comput.*, 43(1):A139–A163, January 2021.

[67] David Elliott. The numerical solution of integral equations using Chebyshev polynomials. *J. Aust. Math. Soc.*, 1(3):344–356, 1960.

[68] David Elliott. A Chebyshev series method for the numerical solution of Fredholm integral equations. *Comput. J.*, 6(1):102–112, 1963.

[69] David Elliott. The evaluation and estimation of the coefficients in the Chebyshev series expansion of a function. *Math. Comput.*, 18(86):274–284, 1964.

[70] Evgeny Epifanovsky, Andrew TB Gilbert, Xintian Feng, Joonho Lee, Yuezhi Mao, Narbe Mardirossian, Pavel Pokhilko, Alec F White, Marc P Coons, Adrian L Dempwolff, et al. Software for the frontiers of quantum chemistry: An overview of developments in the Q-Chem 5 package. *J. Chem. Phys.*, 155(8):084801, 2021.

[71] JDf Eshelby. Energy relations and the energy-momentum tensor in continuum mechanics. In *Fundamental contributions to the continuum theory of evolving phase interfaces in solids*, pages 82–119. Springer, 1999.

[72] Derry FitzGerald, Matt Cranitch, and Eugene Coyle. Non-negative tensor factorisation for sound source separation. In *IEE Conf. publication*, volume 511, page 8. Citeseer, 2005.

[73] Daniel Fortunato, Nicholas Hale, and Alex Townsend. The ultraspherical spectral element method. *J. Comput. Phys.*, 436:110087, 2021.

[74] Daniel Fortunato and Alex Townsend. Fast Poisson solvers for spectral methods. *IMA J. Numer. Anal.*, 40(3):1994–2018, 2020.

[75] Leslie Fox and Ian Bax Parker. Chebyshev polynomials in numerical analysis. Technical report, 1968.

[76] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201, 1993.

[77] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586, 2018.

[78] W Morven Gentleman. Implementing Clenshaw-Curtis quadrature, II computing the cosine transformation. *Commun. ACM*, 15(5):343–346, 1972.

[79] Lars Goerigk and Stefan Grimme. Double-hybrid density functionals. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.*, 4(6):576–600, 2014.

[80] Matthew Goldey, Anthony Dutoi, and Martin Head-Gordon. Attenuated second-order Møller–Plesset perturbation theory: performance within the aug-cc-pVTZ basis. *Phys. Chem. Chem. Phys.*, 15(38):15869–15875, 2013.

[81] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[82] Andrei Aleksandrovich Gončar. Zolotarev problems connected with rational functions. *Sbornik: Mathematics*, 7(4):623–635, 1969.

[83] Lars Grasedyck. Existence and computation of low Kronecker-rank approximations for large linear systems of tensor product structure. *Comput.*, 72(3-4):247–265, 2004.

[84] Lars Grasedyck. Hierarchical singular value decomposition of tensors. *SIAM J. Matrix Anal. Appl.*, 31(4):2029–2054, 2010.

[85] Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.

[86] Lars Grasedyck and Christian Löbbert. Parallel algorithms for low rank tensor arithmetic. In *Advances in Mathematical Methods and High Performance Computing*, pages 271–282. Springer, 2019.

[87] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[88] Victor S Grigorascu and Phillip A Regalia. Tensor displacement structures and polyspectral matching. In *Fast Reliable Algorithms for Matrices with Structure*, pages 245–276. SIAM, 1999.

[89] Laura Grigori and Suraj Kumar. Parallel tensor train through hierarchical decomposition. 2020.

[90] Serkan Gugercin, Danny C Sorensen, and Athanasios C Antoulas. A modified low-rank Smith method for large-scale Lyapunov equations. *Numerical Algorithms*, 32(1):27–55, 2003.

[91] Yang Guo, Christoph Riplinger, Ute Becker, Dimitrios G Liakos, Yury Minenkov, Luigi Cavallo, and Frank Neese. Communication: An improved linear scaling perturbative triples correction for the domain based local

pair-natural orbital based singles and doubles coupled cluster method [DLPNO-CCSD(T)]. *J. Chem. Phys.*, 148(1):011101, 2018.

[92] Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*, volume 42. Springer Science & Business Media, 2012.

[93] Wolfgang Hackbusch, Boris N Khoromskij, and Eugene E Tyrtyshnikov. Hierarchical Kronecker tensor-product approximations. *J. Numer. Math.*, 13(2):119–156, 2005.

[94] Diptarka Hait and Martin Head-Gordon. How accurate are static polarizability predictions from density functional theory? An assessment over 132 species at equilibrium geometry. *Phys. Chem. Chem. Phys.*, 20(30):19800–19810, 2018.

[95] Diptarka Hait and Martin Head-Gordon. How accurate is density functional theory at predicting dipole moments? An assessment using a new database of 200 benchmark values. *J. Chem. Theory Comput.*, 14(4):1969–1981, 2018.

[96] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011.

[97] Richard A Harshman et al. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.

[98] Behnam Hashemi and Lloyd N Trefethen. Chebfun in three dimensions. *SIAM J. Sci. Comput.*, 39(5):C341–C363, 2017.

[99] Johan Håstad. Tensor rank is NP-complete. *J. Algor.*, 11(4):644–654, 1990.

[100] Tamir Hazan, Simon Polak, and Amnon Shashua. Sparse image coding using a 3D non-negative tensor factorization. In *10th IEEE Int. Conf. CV*, volume 1, pages 50–57. IEEE, 2005.

[101] Martin Head-Gordon. Quantum chemistry and molecular processes. *J. Phys. Chem.*, 100(31):13213–13225, 1996.

[102] Martin Head-Gordon, Paul E Maslen, and Christopher A White. A tensor

formulation of many-electron theory in a nonorthogonal single-particle basis. *J. Chem. Phys.*, 108(2):616–625, 1998.

[103] Martin Head-Gordon, John A Pople, and Michael J Frisch. MP2 energy evaluation by direct methods. *Chem. Phys. Lett.*, 153(6):503–506, 1988.

[104] Trygve Helgaker, Poul Jorgensen, and Jeppe Olsen. *Molecular electronic-structure theory*. John Wiley & Sons, 2014.

[105] René Henrion. N-way principal component analysis theory, algorithms and applications. *Chemom. Intell. Lab. Syst.*, 25(1):1–23, 1994.

[106] David Hilbert. Ein Beitrag zur Theorie des Le'gendre'schen Polynoms. *Acta Math.*, 18(1):155, 1894.

[107] Egil A Hylleraas. Über den Grundterm der Zweielektronenprobleme von H−, He, Li+, Be++ usw. *Zeitschrift für Physik*, 65(3):209–225, 1930.

[108] Ilgis Ibragimov and Sergej Rjasanow. Three way decomposition for the Boltzmann equation. *J. Comput. Math.*, pages 184–195, 2009.

[109] Oguz Kaya and Bora Uçar. High performance parallel algorithms for the Tucker decomposition of sparse tensors. In *2016 45th Int. Conf. Parallel Process.*, pages 103–112. IEEE, 2016.

[110] Vladimir Kazeev and Christoph Schwab. Quantized tensor-structured finite elements for second-order elliptic PDEs in two dimensions. *Numerische Math.*, 138(1):133–190, 2018.

[111] Vladimir A Kazeev and Boris N Khoromskij. Low-rank explicit QTT representation of the Laplace operator and its inverse. *SIAM J. Matrix Anal. Appl.*, 33(3):742–758, 2012.

[112] Rick A Kendall and Herbert A Früchtl. The impact of the resolution of the identity approximate integral method on modern ab initio algorithm development. *Theor. Chem. Acc.*, 97(1):158–163, 1997.

[113] Boris Khoromskij and Sergey Repin. Rank structured approximation method for quasi-periodic elliptic problems. *Comput. Methods Appl. Math.*, 17(3):457–477, 2017.

[114] Boris Khoromskij, Stefan Sauter, and Alexander Veit. Fast quadrature techniques for retarded potentials based on TT/QTT tensor approximation. *Comput. Methods Appl. Math.*, 11(3):342–362, 2001.

[115] Boris Khoromskij and Alexander Veit. Efficient computation of highly oscillatory integrals by using QTT tensor approximation. *Comput. Methods Appl. Math.*, 16(1):145–159, 2016.

[116] Boris N Khoromskij. Tensor-structured preconditioners and approximate inverse of elliptic operators in R^d. *Constr. Approx.*, 30(3):599, 2009.

[117] Boris N Khoromskij. Fast and accurate tensor approximation of a multivariate convolution with linear scaling in dimension. *J. Comput. Appl. Math.*, 234(11):3122–3139, 2010.

[118] Boris N Khoromskij. $O(d \log N)$-quantics approximation of N-d tensors in high-dimensional numerical modeling. *Constr. Approx.*, 34(2):257–280, 2011.

[119] Boris N Khoromskij. *Tensor numerical methods in scientific computing*, volume 19. Walter de Gruyter GmbH & Co KG, 2018.

[120] Boris N Khoromskij, Venera Khoromskaia, and H-J Flad. Numerical solution of the Hartree–Fock equation in multilevel tensor-structured format. *SIAM J. Sci. Comput.*, 33(1):45–65, 2011.

[121] Boris N Khoromskij and Christoph Schwab. Tensor-structured Galerkin approximation of parametric and stochastic elliptic PDEs. *SIAM J. Sci. Comput.*, 33(1):364–385, 2011.

[122] Tamara G Kolda and Brett W Bader. MATLAB tensor toolbox. Technical report, Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA . . . , 2006.

[123] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, 2009.

[124] Tamara Gibson Kolda. Multilinear operators for higher-order decompositions. Technical report, Sandia National Laboratories, 2006.

[125] Daniel Kressner, Rajesh Kumar, Fabio Nobile, and Christine Tobler. Low-

rank tensor approximation for high-order correlation functions of Gaussian random fields. *SIAM/ASA J. Uncertainty Quantif.*, 3(1):393–416, 2015.

[126] Daniel Kressner and Lana Perisa. Recompression of Hadamard products of tensors in tucker format. *SIAM J. Sci. Comput.*, 39(5):A1879–A1902, 2017.

[127] Daniel Kressner and Christine Tobler. Preconditioned low-rank methods for high-dimensional elliptic pde eigenvalue problems. *Comput. Methods Appl. Math.*, 11(3):363–381, 2011.

[128] Daniel Kressner and Christine Tobler. Algorithm 941: Htucker—a MATLAB toolbox for tensors in hierarchical Tucker format. *ACM Trans. Math. Software*, 40(3):1–22, 2014.

[129] J. B. Kruskal. Rank decomposition and uniqueness for 3-way and N-way arrays. In *Multiway Data Analysis*, pages 7–18. North-Holland, Amsterdam, 1988.

[130] Nikolai Laskin. Fractional quantum mechanics and Lévy path integrals. *Phys. Let. A*, 268(4-6):298–305, 2000.

[131] Joonho Lee and Martin Head-Gordon. Distinguishing artificial and essential symmetry breaking in a single determinant: Approach and application to the C_{60}, C_{36}, and C_{20} fullerenes. *Phys. Chem. Chem. Phys.*, 21(9):4763–4778, 2019.

[132] Michael S Lee, Paul E Maslen, and Martin Head-Gordon. Closely approximating second-order Møller–Plesset perturbation theory with a local triatomics in molecules model. *J. Chem. Phys.*, 112(8):3592–3601, 2000.

[133] Namgil Lee and Andrzej Cichocki. Fundamental tensor operations for large-scale data analysis using tensor network formats. *Multidimension. Syst. Signal Process.*, 29(3):921–960, 2018.

[134] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, volume 98. SIAM, 2007.

[135] Jiajia Li, Jee Choi, Ioakeim Perros, Jimeng Sun, and Richard Vuduc. Model-driven sparse CP decomposition for higher-order tensors. In *2017 IEEE Int. Parallel Distrib. Process. Symp.*, pages 1048–1057. IEEE, 2017.

[136] Jing-Rebecca Li and Jacob White. Low rank solution of Lyapunov equations. *SIAM J. Matrix Anal. Appl.*, 24(1):260–280, 2002.

[137] Anna Lischke, Guofei Pang, Mamikon Gulian, Fangying Song, Christian Glusa, Xiaoning Zheng, Zhiping Mao, Wei Cai, Mark M Meerschaert, Mark Ainsworth, et al. What is the fractional Laplacian? A comparative review with new results. *J. Comput. Phys.*, 404:109009, 2020.

[138] Ning Liu, Benyu Zhang, Jun Yan, Zheng Chen, Wenyin Liu, Fengshan Bai, and Leefeng Chien. Text representation: From vector to tensor. In *5th IEEE Int. Conf. DM*, pages 4–pp. IEEE, 2005.

[139] Haiping Lu, Konstantinos N Plataniotis, and Anastasios N Venetsanopoulos. A survey of multilinear subspace learning for tensor data. *Pattern Recognit.*, 44(7):1540–1551, 2011.

[140] Yudell L Luke. *Special functions and their approximations*, volume 2. Academic press, 1969.

[141] Linjian Ma and Edgar Solomonik. Fast and accurate randomized algorithms for low-rank tensor decompositions. *arXiv preprint arXiv:2104.01101*, 2021.

[142] Narbe Mardirossian and Martin Head-Gordon. Survival of the most transferable at the top of Jacob's ladder: Defining and testing the ωB97M(2) double hybrid density functional. *J. Chem. Phys.*, 148(24):241736, 2018.

[143] Jan ML Martin and Golokesh Santra. Empirical double-hybrid density functional theory: A 'third way' in between WFT and DFT. *Isr. J. Chem.*, 60(8-9):787–804, 2020.

[144] PG Martinsson. The hierarchical Poincaré-Steklov (HPS) solver for elliptic PDEs: A tutorial. *arXiv preprint arXiv:1506.01308*, 2015.

[145] John C Mason and David C Handscomb. *Chebyshev polynomials*. Chapman and Hall/CRC, 2002.

[146] Stefano Massei, Davide Palitta, and Leonardo Robol. Solving rank-structured Sylvester and Lyapunov equations. *SIAM J. Matrix Anal. Appl.*, 39(4):1564–1590, 2018.

[147] Jiří Matoušek and Petr Škovroň. Removing degeneracy may require a large dimension increase. *Theory. Compute.*, 3(1):159–177, 2007.

[148] Joseph E Mayer. Electron correlation. *Phys. Rev.*, 100(6):1579, 1955.

[149] Dominik Meidner, Johannes Pfefferer, Klemens Schürholz, and Boris Vexler. *hp*-finite elements for fractional diffusion. *SIAM Journal on Numerical Analysis*, 56(4):2345–2374, 2018.

[150] Fumikazu Miwakeichi, Eduardo Martınez-Montes, Pedro A Valdés-Sosa, Nobuaki Nishiyama, Hiroaki Mizuhara, and Yoko Yamaguchi. Decomposing EEG data into space–time–frequency components using parallel factor analysis. *Neuroimage*, 22(3):1035–1045, 2004.

[151] Martin J Mohlenkamp and Lucas Monzón. Trigonometric identities and sums of separable functions. *Math. Intell.*, 27(2):65–69, 2005.

[152] Morten Mørup, Lars Kai Hansen, Christoph S Herrmann, Josef Parnas, and Sidse M Arnfred. Parallel factor analysis as an exploratory tool for wavelet transformed event-related EEG. *Neuroimage*, 29(3):938–947, 2006.

[153] Robert S Mulliken. Report on notation for the spectra of polyatomic molecules. *J. Chem. Phys.*, 23(11):1997–2011, 1955.

[154] Damien Muti and Salah Bourennane. Multidimensional filtering based on a tensor approach. *Signal Process.*, 85(12):2338–2353, 2005.

[155] James G Nagy and Misha Elena Kilmer. Kronecker product approximation for preconditioning in three-dimensional imaging applications. *IEEE Trans. Image Process.*, 15(3):604–613, 2006.

[156] Ricardo H Nochetto, Enrique Otárola, and Abner J Salgado. A PDE approach to fractional diffusion in general domains: a priori error analysis. *Found. Comput. Math.*, 15(3):733–791, 2015.

[157] Vadim Olshevsky, Ivan Oseledets, and Eugene Tyrtyshnikov. Superfast inversion of two-level Toeplitz matrices using Newton iteration and tensor-displacement structure. In *Recent Advances in Matrix and Operator Theory*, pages 229–240. Springer, 2007.

[158] Frank WJ Olver, Daniel W Lozier, Ronald F Boisvert, and Charles W Clark.

NIST handbook of mathematical functions hardback and CD-ROM. Cambridge University Press, 2010.

[159] Sheehan Olver and Alex Townsend. A fast and well-conditioned spectral method. *SIAM Rev.*, 55(3):462–489, 2013.

[160] IV Oseledets. Constructive representation of functions in low-rank tensor formats. *Constr. Approx.*, 37(1):1–18, 2013.

[161] Ivan Oseledets, Sergey Dolgov, Vladimir Kazeev, Olga Lebedeva, and Thomas Mach. MATLAB TT-Toolbox, 2019.

[162] Ivan Oseledets and Eugene Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Lin. Alg. Appl.*, 432(1):70–88, 2010.

[163] Ivan V Oseledets. Approximation of $2^d \times 2^d$ matrices using tensor decomposition. *SIAM J. Matrix Anal. Appl.*, 31(4):2130–2145, 2010.

[164] Ivan V Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011.

[165] Ivan V Oseledets and Sergey V Dolgov. Solution of linear systems and matrix inversion in the TT-format. *SIAM J. Sci. Comput.*, 34(5):A2718–A2739, 2012.

[166] Ivan V Oseledets and Eugene E Tyrtyshnikov. Breaking the curse of dimensionality, or how to use SVD in many dimensions. *SIAM J. Sci. Comput.*, 31(5):3744–3759, 2009.

[167] Davide Palitta and Valeria Simoncini. Numerical methods for large-scale Lyapunov equations with symmetric banded data. *SIAM J. Sci. Comput.*, 40(5):A3581–A3608, 2018.

[168] Thilo Penzl. A cyclic low-rank Smith method for large sparse Lyapunov equations. *SIAM J. Sci. Comput.*, 21(4):1401–1418, 1999.

[169] David Perez-Garcia, Frank Verstraete, Michael M Wolf, and J Ignacio Cirac. Matrix product state representations. *arXiv preprint quant-ph/0608197*, 2006.

[170] János Pipek and Paul G Mezey. A fast intrinsic localization procedure

applicable for abinitio and semiempirical linear combination of atomic orbital wave functions. *J. Chem. Phys.*, 90(9):4916–4926, 1989.

[171] John A Pople, J Stephen Binkley, and Rolf Seeger. Theoretical models incorporating electron correlation. *Int. J. Quantum Chem.*, 10(S10):1–19, 1976.

[172] Michael James David Powell. *Approximation theory and methods*. Cambridge university press, 1981.

[173] Peter Pulay. Localizability of dynamic electron correlation. *Chem. Phys. Lett.*, 100(2):151–154, 1983.

[174] Krishnan Raghavachari, Gary W Trucks, John A Pople, and Martin Head-Gordon. A fifth-order perturbation comparison of electron correlation theories. *Chem. Phys. Lett.*, 157(6):479–483, 1989.

[175] JB Reade. Eigenvalues of positive definite kernels. *SIAM J. Math. Anal.*, 14(1):152–157, 1983.

[176] Theodore J Rivlin. *Chebyshev polynomials*. Courier Dover Publications, 2020.

[177] Youcef Saad. Numerical solution of large lyapunov equations. Technical report, 1989.

[178] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Compute.*, 7(3):856–869, 1986.

[179] John Sabino. Solution of large-scale Lyapunov equations via the block modified Smith methods. Technical report, 2006.

[180] S Saebo and Peter Pulay. Local treatment of electron correlation. *Annu. Rev. Phys. Chem.*, 44(1):213–236, 1993.

[181] Berkant Savas. Analyses and tests of handwritten digit recognition algorithms. *LiTH-MAT-EX-2003-01, Linkˆping University, Department of Mathematics*, 2003.

[182] Berkant Savas and Lars Eldén. Handwritten digit classification using

higher order singular value decomposition. *Pattern Recognit.*, 40(3):993–1003, 2007.

[183] Dmitry Savostyanov and Ivan Oseledets. Fast adaptive interpolation of multi-dimensional arrays in tensor train format. In *2011 Int. Workshop Multidimens. (nD) Syst.*, pages 1–8. IEEE, 2011.

[184] Dmitry V Savostyanov, SV Dolgov, JM Werner, and Ilya Kuprov. Exact NMR simulation of protein-size spin systems using tensor train formalism. *Phys. Rev. B: Condens. Matter*, 90(8):085139, 2014.

[185] Kathrin Schacke. On the Kronecker product. Master's thesis, University of Waterloo, 2004.

[186] Britta Schmitt, Boris N Khoromskij, Venera Khoromskaia, and Volker Schulz. Tensor method for optimal control problems constrained by fractional three-dimensional elliptic operator with variable coefficients. *Numer. Linear Algebra Appl.*, page e2404, 2021.

[187] Amnon Shashua and Tamir Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proc. 22nd Int. Conf. Mach. Learn.*, pages 792–799, 2005.

[188] James Shee, Matthias Loipersberger, Adam Rettig, Joonho Lee, and Martin Head-Gordon. Regularized second-order Møller–Plesset theory: A more accurate alternative to conventional MP2 for noncovalent interactions and transition metal thermochemistry for the same computational cost. *J. Phys. Chem. Lett.*, 12:12084–12097, 2021.

[189] C David Sherrill. An introduction to Hartree-Fock molecular orbital theory. *School of Chemistry and Biochemistry Georgia Institute of Technology*, 2000.

[190] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[191] Tianyi Shi, Harbir Antil, and Drew P Kouri. Spectral, tensor and domain decomposition methods for fractional pdes. *Comput. Methods Appl. Math.*, 2022.

[192] Tianyi Shi, Maximilian Ruth, and Alex Townsend. Parallel algo-

rithms for computing the tensor-train decomposition. *arXiv preprint arXiv:2111.10448*, 2021.

[193] Tianyi Shi and Alex Townsend. On the compressibility of tensors. *SIAM J. Matrix Anal. Appl.*, 42(1):275–298, 2021.

[194] Valeria Simoncini. A new iterative method for solving large-scale Lyapunov matrix equations. *SIAM J. Sci. Comput.*, 29(3):1268–1288, 2007.

[195] Valeria Simoncini. Computational methods for linear matrix equations. *SIAM Rev.*, 58(3):377–441, 2016.

[196] David W Small and Martin Head-Gordon. Post-modern valence bond theory for strongly correlated electron spins. *Phys. Chem. Chem. Phys.*, 13(43):19285–19297, 2011.

[197] Shaden Smith, Niranjay Ravindran, Nicholas D Sidiropoulos, and George Karypis. SPLATT: Efficient and parallel sparse tensor-matrix multiplication. In *2015 IEEE Int. Parallel Distrib. Process. Symp.*, pages 61–70. IEEE, 2015.

[198] Edgar Solomonik, Devin Matthews, Jeff R Hammond, John F Stanton, and James Demmel. A massively parallel tensor contraction framework for coupled-cluster computations. *J. Parallel Distrib. Comput.*, 74(12):3176–3190, 2014.

[199] Fangying Song, Chuanju Xu, and George Em Karniadakis. Computing fractional Laplacians on complex-geometry domains: algorithms and simulations. *SIAM J. Sci. Comput.*, 39(4):A1320–A1344, 2017.

[200] Gerhard Starke. Near-circularity for the rational Zolotarev problem in the complex plane. *J. Approx. Theory*, 70(1):115–130, 1992.

[201] Pablo Raúl Stinga and José Luis Torrea. Extension problem and Harnack's inequality for some fractional operators. *Comm. Part. Diff. Eqs.*, 35(11):2092–2122, 2010.

[202] Jian-Tao Sun, Hua-Jun Zeng, Huan Liu, Yuchang Lu, and Zheng Chen. CubeSVD: a novel approach to personalized web search. In *Proc. 14th Int. Conf. World Wide Web*, pages 382–390, 2005.

[203] Jimeng Sun, Spiros Papadimitriou, and S Yu Philip. Window-based tensor

analysis on high-dimensional and multi-aspect streams. In *6th Int. Conf. DM*, pages 1076–1080. IEEE, 2006.

[204] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proc. 12th ACM SIGKDD*, pages 374–383, 2006.

[205] Yifei Sun and Mrinal Kumar. Numerical solution of high dimensional stationary Fokker–Planck equations via tensor decomposition and Chebyshev spectral differentiation. *Comput. Math. Appl.*, 67(10):1960–1977, 2014.

[206] Yiming Sun, Yang Guo, Charlene Luo, Joel Tropp, and Madeleine Udell. Low-rank Tucker approximation of a tensor from streaming data. *SIAM J. Math. Data. Sci.*, 2(4):1123–1150, 2020.

[207] Yiming Sun, Yang Guo, Joel A Tropp, and Madeleine Udell. Tensor random projection for low memory dimension reduction. In *NeurIPS Workshop Relat. Represent. Learn.*, 2018.

[208] Attila Szabo and Neil S Ostlund. *Modern quantum chemistry: introduction to advanced electronic structure theory*. Courier Corporation, 2012.

[209] Alex Townsend. *Computing with Functions in Two Dimensions*. PhD thesis, University of Oxford, 2014.

[210] Alex Townsend and Sheehan Olver. The automatic solution of partial differential equations using a global spectral method. *J. Comput. Phys.*, 299:106–123, 2015.

[211] Alex Townsend and Heather Wilber. On the singular values of matrices with high displacement rank. *Lin. Alg. Appl.*, 548:19–41, 2018.

[212] Lloyd N Trefethen. *Approximation Theory and Approximation Practice*. SIAM, 2013.

[213] Lloyd N Trefethen. Cubature, approximation, and isotropy in the hypercube. *SIAM Rev.*, 59(3):469–491, 2017.

[214] R. A. Van De Geijn and J. Watts. SUMMA: scalable universal matrix multiplication algorithm. *Concurrency Pract. Expc.*, 9(4):255–274, April 1997.

[215] M Alex O Vasilescu. Human motion signatures: Analysis, synthesis, recognition. In *2002 Int. Conf. Pattern Recognit.*, volume 3, pages 456–460. IEEE, 2002.

[216] M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *Eur. Conf. Compute. Vision*, pages 447–460. Springer, 2002.

[217] Nico Vervliet, Otto Debals, and Lieven De Lathauwer. Tensorlab 3.0—Numerical optimization strategies for large-scale constrained and coupled matrix/tensor factorization. In *2016 50th Asilomar Conf. Signal Syst. Comput.*, pages 1733–1738. IEEE, 2016.

[218] Daniel Vlasic, Matthew Brand, Hanspeter Pfister, and Jovan Popovic. Face transfer with multilinear models. In *ACM SIGGRAPH 2006 Courses*, pages 24–es. 2006.

[219] Hongcheng Wang and Narendra Ahuja. Compact representation of multidimensional data using tensor rank-one decomposition. *Vectors*, 1(5), 2004.

[220] Hongcheng Wang et al. Facial expression decomposition. In *Proc. 9th IEEE Int. Conf. Compute. Vision*, pages 958–965. IEEE, 2003.

[221] Xiaokang Wang, Laurence T Yang, Yihao Wang, Lei Ren, and M Jamal Deen. Adtt: A highly efficient distributed tensor-train decomposition method for IIoT big data. *IEEE Trans. Ind. Inf.*, 17(3):1573–1582, 2020.

[222] Zhenling Wang. personal communication.

[223] Florian Weigend, Marco Häser, Holger Patzelt, and Reinhart Ahlrichs. RI-MP2: optimized auxiliary basis sets and demonstration of efficiency. *Chem. Phys. Lett.*, 294(1-3):143–152, 1998.

[224] Florian Weigend, Andreas Köhn, and Christof Hättig. Efficient use of the correlation consistent basis sets in resolution of the identity MP2 calculations. *J. Chem. Phys.*, 116(8):3175–3183, 2002.

[225] Chester J Weiss, Bart G van Bloemen Waanders, and Harbir Antil. Fractional operators applied to geophysical electromagnetics. *Geophys. J. Intern.*, 220(2):1242–1259, 2020.

[226] Hans-Joachim Werner, Frederick R Manby, and Peter J Knowles. Fast linear scaling second-order Møller-Plesset perturbation theory (MP2) using local and density fitting approximations. *J. Chem. Phys.*, 118(18):8149–8160, 2003.

[227] Christopher A White and Martin Head-Gordon. Derivation and efficient implementation of the fast multipole method. *J. Chem. Phys.*, 101(8):6593–6605, 1994.

[228] Eugene P Wigner. Application of the Rayleigh-Schrödinger perturbation theory to the hydrogen atom. *Phys. Rev.*, 94(1):77, 1954.

[229] Eugene P Wigner. On a modification of the Rayleigh-Schrödinger perturbation theory. In *Part I: Physical Chemistry. Part II: Solid State Physics*, pages 131–136. Springer, 1997.

[230] Heather Wilber, Alex Townsend, and Grady B Wright. Computing with functions in spherical and polar geometries II. the disk. *SIAM J. Sci. Comput.*, 39(3):C238–C262, 2017.

[231] Stephen Wilson. *Electron correlation in molecules*. Courier Corporation, 2014.

[232] Xin Xing, Hua Huang, and Edmond Chow. A linear scaling hierarchical block low-rank representation of the electron repulsion integral tensor. *J. Chem. Phys.*, 153(8):084119, 2020.

[233] Mohsen Zayernouri, Mark Ainsworth, and George Em Karniadakis. A unified Petrov–Galerkin spectral method for fractional PDEs. *Comput. Methods Appl. Mech. Engrg.*, 283:1545–1569, 2015.

[234] EI Zolotarev. Application of elliptic functions to questions of functions deviating least and most from zero. *Zap. Imp. Akad. Nauk. St. Petersburg*, 30(5):1–59, 1877.