# ABSTRACT

MINSTER, RACHEL LYNN. Randomized Algorithms for Tensors and Matrices with Applications. (Under the direction of Arvind K. Saibaba.)

This thesis is focused on developing randomized low-rank approximations for matrices and tensors, as well as applications of these algorithms to areas such as system identification and kernel interactions. There are three projects that form the core of this thesis.

In the first project, we focus on randomized low-rank approximations for tensors in the Tucker format. Many applications in data science and scientific computing involve large-scale datasets that are expensive to store and manipulate. However, these datasets possess inherent multidimensional structure that can be exploited to compress and store the dataset in an appropriate tensor format. In recent years, randomized matrix methods have been used to efficiently and accurately compute low-rank matrix decompositions. Motivated by this success, we develop randomized algorithms for tensor decompositions in the Tucker representation. Specifically, we present randomized versions of two well-known compression algorithms, namely, HOSVD and STHOSVD, and a detailed probabilistic analysis of the error in using both algorithms. We also develop variants of these algorithms that tackle specific challenges posed by large-scale datasets. The first variant adaptively finds a low-rank representation satisfying a given tolerance, which is beneficial when the target-rank is not known in advance. The second variant preserves the structure of the original tensor, and is beneficial for large sparse tensors that are difficult to load in memory. We consider several different datasets for our numerical experiments: synthetic test tensors, and realistic applications such as the compression of facial image samples in the Olivetti database and word counts in the Enron email dataset.

Our second project concerns randomized algorithms used to accelerate processes for system identification. Eigensystem Realization Algorithm (ERA) is a data-driven approach for subspace system identification and is widely used in many areas of engineering. However, the computational cost of the ERA is dominated by a step that involves the singular value decomposition (SVD) of a large, dense matrix with block Hankel structure. In this project, we develop computationally efficient algorithms for reducing the computational cost of the SVD step by using randomized subspace iteration and exploiting the block Hankel structure of the matrix. We provide a detailed analysis of the error in the identified system matrices and the computational cost of the proposed algorithms. We demonstrate the accuracy and computational benefits of our algorithms on two test problems: the first involves a partial differential equation that models the cooling of steel rails, and the second is an application from power systems engineering.

In our final project, we develop tensor-based methods for approximating low-rank kernel interactions. Kernel matrices, which are dense matrices whose entries model pairwise interactions between sets of points, appear in many applications such as integral equations and Gaussian processes. The number of interaction points can become quite large, sometimes making these

matrices prohibitively expensive to work with. We can efficiently approximate kernel matrices by representing them as rank-structured matrices, which is done by identifying and compressing off-diagonal blocks in a low-rank format. Although the rank-$r$ SVD gives the optimal rank-$r$ approximation when compressing the off-diagonal blocks, it is computationally expensive. Fast low-rank algorithms have been developed, but important challenges in computational and storage costs remain. We present our new tensor-based approach, which builds on an existing kernel-independent approach employing Chebyshev interpolation. Our approach takes the resulting block matrix, maps it to a four-dimensional tensor, compresses the tensor using new tensor compression algorithms, and maps back to a compressed block matrix. We discuss the computational costs of the proposed algorithms, and also provide extensive numerical tests that demonstrate the accuracy of the methods.

Randomized Algorithms for Tensors and Matrices with Applications

by
Rachel Lynn Minster

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Mathematics

Raleigh, North Carolina

2021

APPROVED BY:

| | |
|---|---|
| Ilse C.F. Ipsen | Agnes Szanto |
| Eric Chi | Arvind K. Saibaba<br>Chair of Advisory Committee |

## BIOGRAPHY

Rachel Minster was born in Winston-Salem, North Carolina, and later attended the University of North Carolina at Charlotte, where she received her Bachelor of Science in Mathematics and her Bachelor of Arts in German in 2016. She then moved to Raleigh to attend North Carolina State University to further pursue Mathematics, receiving her Masters degree in 2018.

## ACKNOWLEDGEMENTS

I would like to thank my advisor, Arvind Saibaba, for his help and guidance throughout my time at NC State. I have truly enjoyed working with him, and I appreciate all the time and hard work that went in to helping me through the last several years.

I would also like to thank my friends, family, and husband for supporting me throughout this entire process.

# TABLE OF CONTENTS

**LIST OF TABLES** . . . . . **vi**

**LIST OF FIGURES** . . . . . **viii**

**Chapter 1 Introduction** . . . . . **1**
1.1 Overview of the thesis . . . . . 2

**Chapter 2 Background and Notation** . . . . . **4**
2.1 Singular Value Decomposition . . . . . 4
2.2 Randomized SVD . . . . . 5
2.3 Subset Selection and Interpolatory Decompositions . . . . . 6
2.4 Tensor Notation and Preliminaries . . . . . 8
2.4.1 HOSVD/STHOSVD . . . . . 11
2.4.2 Best Approximation . . . . . 13

**Chapter 3 Randomized Low-rank Approximation Algorithms for Tucker Decompositions** . . . . . **14**
3.1 Introduction . . . . . 14
3.2 Randomized HOSVD/STHOSVD . . . . . 16
3.2.1 Algorithms . . . . . 16
3.2.2 Error Analysis . . . . . 17
3.2.3 Computational Cost . . . . . 21
3.3 Adaptive Randomized Tensor Decompositions . . . . . 22
3.4 Structure-preserving decompositions . . . . . 24
3.4.1 Algorithm . . . . . 24
3.4.2 Error Analysis . . . . . 26
3.4.3 Variants . . . . . 29
3.5 Numerical Results . . . . . 30
3.5.1 Test Problems . . . . . 30
3.5.2 Numerical Experiments . . . . . 32
3.6 Conclusion . . . . . 39
3.7 Acknowledgements . . . . . 40

**Chapter 4 Efficient Randomized Algorithms for Subspace System Identification** **41**
4.1 Introduction . . . . . 41
4.2 Background . . . . . 43
4.2.1 Eigensystem Realization Algorithm . . . . . 43
4.2.2 Hankel matrices . . . . . 46
4.2.3 Randomized SVD . . . . . 48
4.3 Randomized algorithms for Eigensystem Realization . . . . . 49
4.3.1 Randomized Eigensystem Realization Algorithm . . . . . 49
4.3.2 Randomized TERA . . . . . 52
4.4 Error Analysis . . . . . 55
4.4.1 Background and assumptions . . . . . 55
4.4.2 Main result . . . . . 57
4.4.3 Accuracy of the singular vectors . . . . . 59

4.4.4 Stability . . . 60
4.5 Numerical Results . . . 60
4.5.1 Heat Transfer . . . 61
4.5.2 Power system . . . 66
4.6 Conclusions and Future work . . . 71
4.7 Acknowledgements . . . 71

**Chapter 5 Efficient Tensor-based Approximations to Kernel Interactions . . . 72**
5.1 Introduction . . . 72
5.2 Background . . . 74
5.2.1 Kernel approximation using Chebyshev interpolation . . . 76
5.3 Tensor-based approximation algorithms . . . 79
5.3.1 Mapping to and from Tensors . . . 80
5.3.2 Method 1: Randomized Interpolatory Tensor Decomposition . . . 81
5.3.3 Method 2: Randomized Interpolatory Tensor Decomposition with Block Selection . . . 82
5.3.4 Method 3: Randomized Kronecker Product . . . 84
5.3.5 Computational Cost . . . 85
5.4 Numerical Results . . . 86
5.4.1 Standard Parameters . . . 87
5.4.2 Experiments . . . 87
5.5 Conclusions . . . 90

**Chapter 6 Conclusions . . . 91**

**BIBLIOGRAPHY . . . 93**

## LIST OF TABLES


Table 3.1 Computational Cost for the HOSVD, R-HOSVD, STHOSVD, and R-STHOSVD algorithms. The first term in each expression is the cost of computing an SVD of the mode unfoldings, and the second is the cost of forming the core tensor. . . . . 22

Table 3.2 Summary of sparse tensor examples from the FROSTT database—we include the details for both the full datasets and the condensed datasets used in our experiments. . . . . 31

Table 3.3 Runtime in seconds of the HOSVD, R-HOSVD, STHOSVD, and R-STHOSVD algorithms on the Hilbert tensor $\boldsymbol{\mathcal{X}}$ with $d = 5$, averaged over three runs. Each algorithm is run with target rank $(5, 5, 5, 5, 5)$, and the randomized algorithms use oversampling parameter $p = 5$. The STHOSVD and R-STHOSVD algorithms use the processing order $\rho = [1, 2, 3, 4, 5]$. . . 33

Table 3.4 Relative error and average runtime in seconds of R-STHOSVD on the Olivetti dataset for different processing orders. The runtime was averaged over three runs, and the R-STHOSVD used a target rank of $(20, 40, 5)$ and an oversampling parameter of $p = 5$ as inputs. . . . . 33

Table 3.5 Rank (size of the core tensor) obtained by the adaptive R-STHOSVD algorithm (Algorithm 8) with different relative error tolerances $\epsilon$ as the size of each dimension of $\boldsymbol{\mathcal{X}}$ increase. The inputs are $\epsilon$, block size $b = 1$, and processing order $\rho = [1, 2, 3]$. . . . . 34

Table 3.6 A comparison of the adaptive R-STHOSVD algorithm (Algorithm 8) to STHOSVD. We first obtained the rank of the core tensor with the requested relative error tolerance from the adaptive algorithm. Then we compared the actual error of the approximation from the adaptive R-STHOSVD to that of an STHOSVD with the same rank. The processing order for all runs was $\rho = [2, 1, 3]$. *Each STHOSVD was computed with the corresponding rank found in the second column. . . . . 35

Table 3.7 A comparison of the adaptive R-HOSVD algorithm (Algorithm 7) to HOSVD. We first obtained the rank of the core tensor with the requested relative error tolerance from the adaptive algorithm. Then we compared the actual error of the approximation from the adaptive R-HOSVD to that of an HOSVD with the same rank. *Each HOSVD was computed with the corresponding rank found in the second column. . . . . 36

Table 3.8 The relative error and runtime of both SP-STHOSVD and R-STHOSVD on the tensors defined in Table 3.2 as the target rank $(r, r, r)$ increases. The processing order was $\rho = [3, 1, 2]$, and the oversampling parameter was $p = 5$. Note that, for simplicity, the rank is the same for each mode, and that the input rank for the R-STHOSVD was $(r + p, r + p, r + p)$ so the approximations have the same size. . . . . 38

Table 3.9 The relative error and runtime of both SP-STHOSVD and R-STHOSVD on the condensed NELL-2 dataset as the target rank $(r, r, r)$ increases. The processing order was $\rho = [3, 1, 2]$, and the oversampling parameter was $p = 5$. Note that the rank is the same for each mode for simplicity, and that the input rank for the R-STHOSVD was $(r + p, r + p, r + p)$ so the approximations have the same size. . . . . 38

Table 3.10 Runtime in hours of SP-STHOSVD on the full Enron dataset with increasing rank $(r, r, r, r)$. The oversampling parameter was $p = 5$, and the processing order was $\rho = [3, 1, 2, 4]$. . . . . 39

Table 4.1 Computational complexity of the dominant step, the SVD step, in each of the four algorithms. Recall that $s$ is the number of block rows and columns in the block Hankel matrix, $m$ is the number of inputs, $\ell$ is the number of outputs, $r$ is the target rank, and $\kappa$ and $c$ are the number of iterations used in cross approximation and the dominant volume submatrix parts of the CUR-ERA algorithm. . . . . 51

Table 4.2 Computational cost of computing RandTERA . . . . . 54

Table 4.3 Details of the test problems we will use in our numerical experiments, as well as the size of the largest Hankel matrix $\boldsymbol{\mathcal{H}}_s$ that occurs for these test problems. . . . . 61

Table 4.4 Reduced number of inputs $m'$ and outputs $\ell'$ based on the singular value threshold $\epsilon$ for use in TERA and RandTERA. . . . . 68

Table 4.5 Computational time in seconds of the SVD step in the identification algorithms SVD-ERA, RandSVD-H, TERA, and RandTERA. Note for $s = 1000$, the matrix was too large to store explicitly, so we only show the run times for the other algorithms. . . . . 70

Table 5.1 Summary of the computational cost of the various algorithms. The upper table represents the computational cost of the standard approaches, and the lower table represents the computational costs associated with the tensor compression methods proposed. Note that for the lower table we need to include the cost of forming $\mathbf{F}_S, \mathbf{F}_T$ which costs $\mathcal{O}(n^2(N_s + N_t))$. Here, $n$ is the number of Chebyshev nodes, $r_t$ is the tensor target rank, $r_m$ is the matrix target rank, $p$ is the oversampling parameter, and $b$ is the number of blocks drawn from $\mathbf{M}$. . . . . 86

## LIST OF FIGURES


Figure 2.1 A Tucker representation of 3-mode tensor $\boldsymbol{\mathcal{X}}$, where the rank of the Tucker form is less than the original dimensions. . . . . 10

Figure 3.1 **Left:** Relative approximation error for 5-mode Hilbert tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{25\times25\times25\times25\times25}$ defined in (3.19), with target rank $(r, r, r, r, r)$ and oversampling parameter $p = 5$. **Right:** Actual relative error for $\boldsymbol{\mathcal{X}}$ from the R-HOSVD and R-STHOSVD algorithms compared to the calculated error bound as the target rank $(r, r, r, r, r)$ increases. Both algorithms use oversampling parameter $p = 5$, and R-STHOSVD uses the processing order $\rho = [1, 2, 3, 4, 5]$. . . . . 32

Figure 3.2 Relative error for $\boldsymbol{\mathcal{X}}$ with and without subspace iteration, compressing just the pixels (mode 2), as the target rank increases. The oversampling parameter was $p = 5$. The right hand plot was run with one step of subspace iteration. . . . . 34

Figure 3.3 Relative error for $\boldsymbol{\mathcal{X}}$ as target rank increases using the STHOSVD, compressing pixels and people (modes 2 and 1), plotted with rank given by the Adaptive R-STHOSVD (Algorithm 8) with the desired relative error tolerance $\epsilon$. Processing order was $\rho = [2, 1, *]$, and the oversampling parameter was $p = 5$. . . . . 36

Figure 3.4 Relative error for synthetic sparse tensor $\boldsymbol{\mathcal{X}}$ defined in (3.20) with $\gamma = 2, 10, 200$ and increasing target rank $(r, r, r)$. We compare the SP-STHOSVD algorithm (Algorithm 9) to the STHOSVD and R-STHOSVD algorithms with inputs of oversampling parameter $p = 5$ and processing order $\rho = [1, 2, 3]$. . . . . 37

Figure 3.5 The relative error and runtime of SP-STHOSVD, Tucker-TS [56], and the One Pass ("Streaming") algorithm [71] on the synthetic sparse tensor defined in (3.20) with increasing target rank. The processing order for SP-STHOSVD was $\rho = [1, 2, 3]$, and the oversampling parameter was $p = 5$. We used the suggested parameters for Tucker-TS and the Streaming algorithm, with a target rank of $(r + p, r + p, r + p)$ so all approximations have the same size. . . . . 39

Figure 4.1 Singular values of the block Hankel matrix $\boldsymbol{\mathcal{H}}_s$ as computed by a full SVD, SVDS-H, and RandSVD-H. We computed the first 20 singular values using SVDS-H and RandSVD-H and plotted them against the first 20 singular values computed by the full SVD. . . . . 62

Figure 4.2 Eigenvalues of the identified $\mathbf{A}_r$ matrix using SVD-ERA, SVDS-H, and RandSVD-H algorithms on the heat transfer problem. We used $s = 1000$ and a target system size of $r = 20$ as inputs for each algorithm. . . . . 62

Figure 4.3 Relative error in recreated Markov parameters compared to a full SVD ERA for both the SVDS-H and RandSVD-H algorithms. We used $s = 1000$ and a target rank of $r = 20$ as inputs. . . . . 63

Figure 4.4 Average run time of the SVD, SVDS, RandSVD, and RandSVD-H algorithms on the heat transfer problem. We averaged the run time over three runs, and a target system size of $r = 20$ as inputs for each algorithm. 64

Figure 4.5 Eigenvalues of the identified $\mathbf{A}_r$ matrix using CUR-ERA, compared to those identified by SVD and RandSVD-H on the heat transfer problem with $s = 1000$ and a target system size of $r = 20$. . . . . 64
Figure 4.6 Relative error in recreated Markov parameters compared to a full SVD-ERA for RandSVD-H, SVDS-H, and CUR-ERA. . . . . 65
Figure 4.7 Experiments with the heat transfer test problem comparing RandSVD-H to RSVDeig-H and RSVDqr-H. The top left figure shows the singular values of $\boldsymbol{\mathcal{H}}_s$ computed by all three algorithms, the top right figure shows the identified eigenvalues of $\mathbf{A}_r$ by using the randomized algorithms, and the bottom left figure shows the average run time in seconds of ERA using the three randomized algorithms. . . . . 66
Figure 4.8 Singular values of block Hankel matrix $\boldsymbol{\mathcal{H}}_s$ as computed by a full SVD and RandSVD-H versus those computed by (left) TERA with varying $\epsilon$ tolerances and (right) RandTERA with varying $\epsilon$ tolerances. . . . . 67
Figure 4.9 Eigenvalues of the identified matrix $\widehat{\mathbf{A}}_r$ from SVD-ERA compared to the identified matrix $\widehat{\mathbf{A}}_r$ from RandSVD-H and RandTERA on the power system test problem with target rank $r = 75$. . . . . 69
Figure 4.10 Relative error in the reconstructed Markov parameters using RandTERA, TERA, and RandSVD-H algorithms. The relative error is computed using the system identified using the SVD-ERA. . . . . 69
Figure 4.11 Impulse response of the original discrete-time system compared against that for the systems identified using SVD-ERA, RandSVD-H, TERA, and RandTERA and excited with impulse input at Generator 1, with responses from Generator 1 (top) and Generator 2 (bottom). . . . . 70

Figure 5.1 Visual of the bounding boxes for source points, $B_s$, and for target points, $B_t$. Note that $\mathcal{X} \subset B_s$ and $\mathcal{Y} \subset B_t$ . . . . . 75
Figure 5.2 Visual representation of mapping source and target points to Chebyshev grids, shown by steps 1 and 3. Step 2 shows computing the interactions between Chebyshev grid. Each numbered step corresponds to a matrix shown in Figure 5.3. . . . . 76
Figure 5.3 Matrices comprising the interaction matrix approximation process, corresponding to steps shown in Figure 5.2. Matrix 1 maps the source points to a Chebyshev grid, matrix 3 maps the target points to a Chebyshev grid, and matrix 2 computes the interactions between Chebyshev grids. . 78
Figure 5.4 Cosine of the angles between true and approximate left singular vectors of $\mathbf{M}_{(1)}$ as $b$ increases. . . . . 83
Figure 5.5 Box setup for our numerical experiments, where $L$ is the length of both the source and target boxes, $D$ is the distance between bottom left corners of boxes, and $\theta$ is the angle describing the placement of the target box. . 87
Figure 5.6 Relative error of SVD, RandSVD, and Methods 1, 2 and 3 with respect to increasing target rank for five different kernels $\kappa(\mathbf{x}, \mathbf{y})$ . . . . . 88
Figure 5.7 Relative error produced by SVD, RandSVD, and Methods 1, 2, and 3 with increasing target rank for four different side lengths $L$ of the source and target boxes. . . . . 89
Figure 5.8 Relative error produced by SVD, RandSVD, and Methods 1, 2, and 3 with increasing target rank for different, increasing distance between the source and target boxes. . . . . 89

Figure 5.9 Relative error produced by SVD, RandSVD, and Methods 1,2, and 3 with increasing target rank for three different values of $n$, the number of Chebyshev nodes. The left figure is $n = 10$, middle is $n = 20$, and right is $n = 45$. . . . . 90

# Chapter 1

# Introduction

Many matrices arising from different applications are large-scale, and pose significant computational challenges. Many of these large-scale matrices are challenging to store entry-wise, but have structure (e.g., low-rank, rank-structured, Hankel) that can be exploited to effectively store and compute with these matrices. For example, if the numerical rank of a matrix is significantly less than either of the matrix dimensions, a low-rank factorization can be used to efficiently store and compute with the matrix. Standard algorithms for computing the low-rank factorization, such as the singular value decomposition or rank-revealing QR factorizations, are still computationally expensive for large-scale matrices.

Another key idea behind this work is that data and operators arising from different applications are often inherently multidimensional. As an example, a series of images stored over time is a natural three-dimensional dataset. Similarly, solutions to a partial differential equation on a 3D spatial grid can be represented as a four-dimensional array. The most common low-rank algorithms only handle two-dimensional data in the form of matrices, however, so this multidimensional structure often goes unrecognized and underutilized. If we instead treat these datasets as tensors, we can exploit the multidimensional structure for a variety of benefits such as cost savings and higher accuracy.

Randomized numerical linear algebra is an emerging field, which brings together many different areas such as random matrix theory, numerical linear algebra, theoretical computer science, and probability theory. Randomized algorithms are used in increasingly diverse contexts including solving least squares and other optimization problems, computing low-rank factorizations, interpolatory decompositions, and eigenvalue problems. They have several benefits, such as ease of implementation and strong performance guarantees, which make them particularly useful for large-scale problems and high-performance computing. In this thesis, we present new, randomized, low-rank approximation algorithms for both matrices and tensors to tackle challenges in various applications.

## 1.1 Overview of the thesis

We present in this thesis three main projects concerning randomized algorithms for tensors and matrices in different applications.

**Chapter 2: Background.** Before we present our main projects, we provide general background material and results on randomized matrix algorithms and tensors in Chapter 2. This material is supplemented by chapter-specific background sections in later chapters.

**Chapter 3: Randomized Low-rank Approximation Algorithms for Tucker Decompositions.** Our first major project concerns randomized algorithms for tensor decompositions in the Tucker form. In this chapter, we present probabilistic error analysis for established randomized algorithms, namely randomized versions of HOSVD [89] and STHOSVD [17]. We also develop new randomized algorithms for tensor decompositions. Specifically, we develop a new adaptive randomized algorithm to compute a tensor decomposition when the target rank is not known. Another main contribution of this chapter is a randomized algorithm producing a low-rank decomposition where the core tensor contains entries from the original tensor. This means the core tensor preserves whatever structure, such as sparsity or non-negativity, the original tensor has. There are significant benefits of our new algorithm for sparse tensors, as the intermediate and final decompositions are able to be stored efficiently. In addition to the algorithm and decomposition, we also provide an analysis of the error and computational cost. We provide several numerical experiments testing our algorithms on synthetic tensors and real-world datasets to show the performance of the algorithms. This chapter reproduces our paper [61] and the supplementary materials to a large extent. We have also included additional numerical experiments. This is joint work with Misha E. Kilmer.

**Chapter 4: Efficient Randomized Algorithms for Subspace System Identification.** In Chapter 4, we present new randomized algorithms for subspace system identification. One of the more common identification methods used, the Eigensystem Realization Algorithm (ERA), involves an SVD of a large block Hankel matrix followed by a truncated SVD, which is used to solve a least-squares optimization problem identifying the system matrices. We aim to reduce the cost of the ERA by using randomized matrix algorithms. In the first of the new algorithms we present, we exploit the block Hankel structure of the matrix in question to efficiently compute matrix-vector products. This is then used to accelerate the standard randomized SVD, reducing computational costs further. Our second algorithm combines the above techniques with another model reduction method known as tangential interpolation. These algorithms have lower storage and computational costs compared to the ERA. In addition, we present error analysis that relates the error in the system matrix eigenvalues to the singular vectors of the block Hankel matrix. We also give additional conditions required for the stability of the identified system matrices. Finally, we present numerical experiments showing the performance of our new algorithms on

two different applications. This chapter reproduces our paper [60] in its entirety. This was joint work with Jishnudeep Kar and Aranya Chakraborrty.

**Chapter 5: Efficient Tensor-based Approximations to Kernel Interactions.** In the third project, presented in Chapter 5, we consider low-rank approximation algorithms for kernel matrices. Building on an existing kernel-independent approach [30], we develop new randomized algorithms that exploit tensor structure inherent to the approach. Specifically, we first present a method for mapping a block matrix to and from a four-dimensional tensor. This provides a framework that allows us to use tensor compression algorithms on a block matrix, which has the potential for making them more accurate and computationally efficient than standard matrix algorithms. Within this framework, we develop three main tensor approximation algorithms. In the first method, we compress each mode using a row interpolatory decomposition, which combines randomized techniques with subset selection. This results in a core tensor comprised of entries from the original tensor, and the method is more cost efficient than traditional methods. In the second algorithm, we take a similar approach as in the first, but we first subsample the tensor to further reduce computational costs. The third algorithm takes an approach that adapts the randomized HOSVD to take a Kronecker product of Gaussian random matrices instead of a single standard Gaussian random matrix. All three tensor methods reduce the computational costs of compression compared to traditional matrix algorithms. Finally, we present numerical results for a variety of kernels and problem settings which demonstrate the accuracy of our algorithms. This was also joint work with Misha E. Kilmer.

**Chapter 6: Conclusions.** Finally, in Chapter 6, we summarize our conclusions and contributions, also discussing potential future directions stemming from this work.

# Chapter 2

# Background and Notation

In this chapter, we review the singular value decomposition (SVD), which provides the basis for many of our later algorithms. We also discuss the relevant background for randomized matrix algorithms, specifically the randomized SVD. Next, we review subset selection methods, including the strong rank-revealing QR factorization (sRRQR). Finally, we introduce the necessary background information for working with tensors, and review standard algorithms for compressing tensors.

## 2.1 Singular Value Decomposition

We first review the singular value decomposition, or SVD, of a matrix. For a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ of rank $k$, the SVD of $\mathbf{X}$ is

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top,$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix whose entries are the singular values of $\mathbf{X}$, $\sigma_i$ for $i = 1, \ldots, \min\{m, n\}$, such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{m,n\}} \geq 0$. The first $k$ singular values are positive, while the remaining singular values are zero. The columns $\mathbf{u}_i \in \mathbb{R}^m$ of $\mathbf{U}$ are called the left singular vectors, and the columns $\mathbf{v}_i \in \mathbb{R}^n$ of $\mathbf{V}$ are called the right singular vectors.

In many applications, we do not require the full SVD, but rather a reduced version known as the truncated SVD. For a rank $r$, the truncated SVD $\mathbf{X}_r$ of a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ is defined as

$$\mathbf{X}_r = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top.$$

An important result involving the SVD is the Eckart-Young Theorem [28], which states that the best rank-$r$ approximation to a matrix, with $r < k = \mathsf{rank}(\mathbf{X})$, is the truncated rank-$r$ SVD of that matrix. We reproduce this theorem in the Frobenius norm here.

**Theorem 1.** *Let $\mathbf{X}_r$ be the truncated rank-r SVD of a matrix $\mathbf{X}$, i.e., $\mathbf{X}_r = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$, with*

$r < k = \mathsf{rank}(\mathbf{X})$. *Then,*

$$\min_{\mathsf{rank}(\mathbf{B})=r} \|\mathbf{X} - \mathbf{B}\|_F = \|\mathbf{X} - \mathbf{X}_r\|_F = \left( \sum_{j=r+1}^{\min\{m,n\}} \sigma_j^2 \right)^{1/2}.$$

## 2.2 Randomized SVD

It is well-known that computing the full SVD of a matrix costs $\mathcal{O}(mn^2)$, assuming $m \geq n$. When the dimensions of a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ are very large, the computational cost of a full SVD may be prohibitively expensive. Randomized SVD, popularized by [41], is a computationally efficient way to compute a rank-$r$ approximation of $\mathbf{X}$. Assuming that $\mathbf{X}$ is approximately low-rank or has singular values that decay rapidly, the randomized SVD delivers a good low-rank representation of $\mathbf{X}$. Compared to the full SVD, the randomized SVD is much more computationally efficient across a wide-range of matrices.

The randomized SVD algorithm has two distinct stages. In the first stage, or the range finding stage, we multiply the matrix $\mathbf{X}$ by a random matrix (we choose this to be standard Gaussian in this paper) to obtain a matrix $\mathbf{Y}$ with contains random linear combinations of the columns of $\mathbf{X}$. We then take a thin QR factorization to obtain a matrix $\mathbf{Q}$ whose range approximates the range of $\mathbf{X}$. By projecting $\mathbf{X}$ onto the range of $\mathbf{Q}$, we obtain the low-rank approximation $\mathbf{X} \approx \mathbf{Q}\mathbf{Q}^\top \mathbf{X}$. Then, in the second stage, or the post-processing stage, we compute a thin SVD of the much smaller matrix $\mathbf{Q}^\top \mathbf{X} = \widehat{\mathbf{U}}_{\mathbf{B}} \widehat{\mathbf{\Sigma}} \widehat{\mathbf{V}}^\top$, and compute its truncated SVD. We then compute $\widehat{\mathbf{U}} = \mathbf{Q}\widehat{\mathbf{U}}_{\mathbf{B}}$ to obtain the low-rank approximation $\widehat{\mathbf{X}} = \widehat{\mathbf{U}} \widehat{\mathbf{\Sigma}} \widehat{\mathbf{V}}^\top$. Algorithm 1 summarizes this process.

**Algorithm 1** $[\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}}, \widehat{\mathbf{V}}] = \mathrm{RandSVD}(\mathbf{X}, r, p, \mathbf{\Omega})$

**Input:** matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, target rank $r$, oversampling parameter $p \geq 0$ such that $r + p \leq \min\{m, n\}$, standard Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{n \times (r+p)}$

**Output:** $\widehat{\mathbf{U}} \in \mathbb{R}^{m \times r}$, $\widehat{\mathbf{\Sigma}} \in \mathbb{R}^{r \times r}$, and $\widehat{\mathbf{V}} \in \mathbb{R}^{n \times r}$ such that $\mathbf{X} \approx \widehat{\mathbf{U}} \widehat{\mathbf{\Sigma}} \widehat{\mathbf{V}}^\top$

1: Multiply $\mathbf{Y} \leftarrow \mathbf{X}\mathbf{\Omega}$
2: Compute thin QR factorization $\mathbf{Y} = \mathbf{Q}\mathbf{R}$
3: Form $\mathbf{B} \leftarrow \mathbf{Q}^\top \mathbf{X}$
4: Calculate thin SVD $\mathbf{B} = \widehat{\mathbf{U}}_{\mathbf{B}} \widehat{\mathbf{\Sigma}} \widehat{\mathbf{V}}^\top$
5: Form $\widehat{\mathbf{U}} \leftarrow \mathbf{Q}\widehat{\mathbf{U}}_{\mathbf{B}}(:, 1:r)$
6: Compress $\widehat{\mathbf{\Sigma}} \leftarrow \widehat{\mathbf{\Sigma}}(1:r, 1:r)$, and $\widehat{\mathbf{V}} \leftarrow \widehat{\mathbf{V}}(:, 1:r)$

The computational cost of the algorithm is

$$\text{Cost} = 2(r+p)\mathcal{O}(\mathsf{nnz}(\mathbf{X})) + \mathcal{O}(r^2(m+n)),$$

where $\mathsf{nnz}$ denotes the number of nonzeros of $\mathbf{X}$.

There is an additional version of the randomized SVD algorithm that includes subspace iterations [41, Algorithm 4.4]. The use of subspace iterations can improve the accuracy of the low-rank approximation, but it increases the computational cost and the required number of passes through the matrix. We expand on more details of this version in Algorithm 11 in Chapter 4.

An error bound for Algorithm 1 in the Frobenius norm is presented below, and we will use this result frequently in our analysis. This theorem can be found in [88, Theorem 3].

**Theorem 2.** *Let $\mathbf{X} \in \mathbb{R}^{m\times n}$, and $\mathbf{\Omega} \in \mathbb{R}^{n\times(r+p)}$ be a Gaussian random matrix. Suppose $\mathbf{Q}$ is obtained from Algorithm 1 with inputs target rank $r \leq \mathsf{rank}(\mathbf{X})$ and oversampling parameter $p \geq 2$ such that $r + p \leq \min\{m, n\}$, and let $\mathbf{B}_r$ be the rank-$r$ truncated SVD of $\mathbf{Q}^\top\mathbf{X}$. Then, the error in expectation satisfies*

$$\mathbb{E}_{\mathbf{\Omega}}\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top\mathbf{X}\|_F^2 \leq \mathbb{E}_{\mathbf{\Omega}}\|\mathbf{X} - \mathbf{Q}\mathbf{B}_r\|_F^2 \leq \left(1 + \frac{r}{p-1}\right)\sum_{j=r+1}^{\min\{m,n\}}\sigma_j^2(\mathbf{X}).$$

We will use two slightly different formulations of this theorem in our later results. Instead of $\|\mathbf{X} - \mathbf{Q}\mathbf{B}_r\|_F^2$, we will use $\|\mathbf{X} - \widehat{\mathbf{U}}\widehat{\mathbf{U}}^\top\mathbf{X}\|_F^2$. It is straightforward to show the equivalence between the two forms, see [65, section 5.3] for the explicit details. We will also use the following result, which uses Hölder's inequality [45, Theorem 23.10]:

$$\mathbb{E}_{\mathbf{\Omega}}\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top\widehat{\mathbf{X}}\|_F \leq \sqrt{1 + \frac{r}{p-1}}\left(\sum_{j=r+1}^{\min\{m,n\}}\sigma_j^2(\mathbf{X})\right)^{1/2}. \tag{2.1}$$

There have been several developments in recent years to the version of the randomized SVD algorithm we present here. Adaptive randomized SVD algorithms are an example of such a development. Found in [41], adaptive algorithms produce a low-rank approximation that satisfies a given error tolerance. These adaptive versions are useful in applications where the target rank is not known a priori. Other algorithms, such as the ones in RSVDPACK [80], use approaches such as an eigenvalue decomposition and a QR decomposition of the smaller matrix $\mathbf{B}$ from step 3 instead of a thin SVD to speed up the standard randomized SVD algorithm. For the most part, we use the version presented in Algorithm 1 as it is sufficient for our purposes, but these other versions could yield improved timing results.

## 2.3 Subset Selection and Interpolatory Decompositions

In some algorithms discussed in this manuscript, we need to form low-rank approximations by selecting certain rows or columns of a matrix. One example of a decomposition with subset selection is the rank-revealing QR decomposition [36]. For matrices $\mathbf{X} \in \mathbb{R}^{m\times n}$ with $m \geq n$ and

a target rank $r \leq k = \mathsf{rank}(\mathbf{X})$, a column-pivoted QR factorization gives

$$\mathbf{XP} = \mathbf{QR} = \mathbf{Q} \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{R}_{22} \end{bmatrix},$$

where $\mathbf{P} \in \mathbb{R}^{n \times n}$ is a permutation matrix, $\mathbf{Q} \in \mathbb{R}^{m \times m}$ is orthogonal, $\mathbf{R}_{11} \in \mathbb{R}^{r \times r}$ and $\mathbf{R}_{22} \in \mathbb{R}^{(m-r) \times (n-r)}$ are upper triangular, and $\mathbf{R}_{12} \in \mathbb{R}^{r \times (n-r)}$. This factorization is considered rank-revealing if $\mathbf{R}$ is formed so that the singular values of $\mathbf{R}_{11}$ are as large as possible, and the singular values of $\mathbf{R}_{22}$ are as small as possible. This is quantified by the following conditions on the singular values:

$$\sigma_{\min}(\mathbf{R}_{11}) \geq \frac{\sigma_r(\mathbf{X})}{\pi(r,n)}, \quad \sigma_{\max}(\mathbf{R}_{22}) \leq \sigma_{r+1}(\mathbf{X})\pi(r,n),$$

where $\pi(r,n)$ is some function bounded by a polynomial in $r$ and $n$. If we use the strong rank-revealing QR (sRRQR) factorization [36, Algorithm 4] with user defined parameter $\eta \geq 1$, the bounds

$$\sigma_i(\mathbf{R}_{11}) \geq \frac{\sigma_i(\mathbf{X})}{\pi(r,n)}, \qquad \sigma_j(\mathbf{R}_{22}) \leq \sigma_{r+j}(\mathbf{X}),$$

for $1 \leq i \leq r$ and $1 \leq j \leq n-r$, hold with $\pi(r,n) = \sqrt{1 + \eta^2 r(n-r)}$.

In our algorithms, we will be using row selection instead of the column selection provided by the rank-revealing QR as presented. To select well-conditioned rows of a matrix, we use column selection on the transpose of the matrix. This requires the sRRQR factorization of a short and fat matrix, i.e., a matrix with more columns than rows. We use the extension result in [13] to apply sRRQR to this type of matrix.

Specifically, consider a matrix $\mathbf{Z} \in \mathbb{R}^{m \times r}$ with orthonormal columns, where $r < m$ is the target rank. Then the strong rank-revealing QR decomposition of $\mathbf{Z}^\top$ takes the form

$$\mathbf{Z}^\top \begin{bmatrix} \mathbf{P}_1 & \mathbf{P}_2 \end{bmatrix} = \mathbf{Q} \begin{bmatrix} \mathbf{R}_1 & \mathbf{R}_2 \end{bmatrix}, \tag{2.2}$$

where $\begin{bmatrix} \mathbf{P}_1 & \mathbf{P}_2 \end{bmatrix} \in \mathbb{R}^{m \times m}$ is a permutation matrix with $\mathbf{P}_1 \in \mathbb{R}^{m \times r}$, $\mathbf{Q} \in \mathbb{R}^{r \times r}$ is orthogonal, $\mathbf{R}_1 \in \mathbb{R}^{r \times r}$ is upper triangular, and $\mathbf{R}_2 \in \mathbb{R}^{r \times (m-r)}$. The matrix $\mathbf{P}_1$ contains columns of the identity matrix, and represents the indices of well-conditioned rows of $\mathbf{Z}$ (columns of $\mathbf{Z}^\top$).

We reproduce Lemma 2.1 from [27], which contains error bounds for the sRRQR algorithm. The bounds in this lemma will be used in our analysis in Chapter 3.

**Lemma 1.** *Let $\mathbf{Z} \in \mathbb{R}^{m \times r}$ with $\mathbf{Z}^\top \mathbf{Z} = \mathbf{I}_r$. Applying [36, Algorithm 4] with target rank $r$ and tuning parameter $\eta \geq 1$ to $\mathbf{Z}^\top$ gives a submatrix $\mathbf{P}_1 \in \mathbb{R}^{m \times r}$ of $\mathbf{I}_m$ with*

$$\frac{1}{\sqrt{1 + \eta^2 r(m-r)}} \leq \sigma_j(\mathbf{P}_1^\top \mathbf{Z}) \leq 1, \quad 1 \leq j \leq r,$$

*and*

$$1 \leq \|(\mathbf{P}_1^\top \mathbf{Z})^{-1}\|_2 \leq \sqrt{1 + \eta^2 r(m-r)}.$$

We use the randomized range finder combined with subset selection to produce a low-rank approximation to a matrix $\mathbf{X}$. The resulting approximation exactly reproduces certain rows of $\mathbf{X}$, and is therefore called an interpolatory decomposition.

First, we compute a basis $\mathbf{Q}$ using the randomized range finding algorithm. Instead of computing the low-rank approximation $\mathbf{Q}\mathbf{Q}^\top\mathbf{X}$, as was done in Algorithm 1, we first identify a set of well-conditioned rows of $\mathbf{Q}$. We obtain these well-conditioned rows by using sRRQR on $\mathbf{Q}^\top$. This is implemented as a selection operator denoted by the matrix $\mathbf{P}$, which contains columns from the identity matrix. We then use the low-rank representation

$$\mathbf{X} \approx \mathbf{Q}(\mathbf{P}^\top\mathbf{Q})^{-1}\mathbf{P}^\top\mathbf{X} = \mathbf{A}\widehat{\mathbf{X}},$$

where $\mathbf{A} = \mathbf{Q}(\mathbf{P}^\top\mathbf{Q})^{-1}$ and $\widehat{\mathbf{X}} = \mathbf{P}^\top\mathbf{X}$.

While we used sRRQR here to extract rows of $\mathbf{X}$, we can also use other subset selection methods. In particular, we will use pivoted QR for a similar tensor algorithm (Algorithm 16) in Chapter 5. The matrix idea behind Algorithm 16 is exactly the same as the idea described above, using randomized range finding followed by subset selection, but we use pivoted QR instead of sRRQR for the subset selection step. The algorithm details for this matrix idea, which we call Randomized Row Interpolatory Decomposition (RRID), are shown in Algorithm 2. This approach is related to the SVD via row extraction algorithm [41, Algorithm 5.2].

**Algorithm 2** $[\mathbf{F}, \mathcal{J}] = \text{RRID}(\mathbf{X}, r, p)$

**Input:** matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$, target rank $r$, oversampling parameter $p$ such that $r+p \leq \min\{m,n\}$
**Output:** factor matrix $\mathbf{F}$, $r+p$ selected indices $\mathcal{J}$

1: Draw standard random Gaussian matrix $\mathbf{\Omega} \in \mathbb{R}^{n\times(r+p)}$
2: Multiply $\mathbf{Y} \leftarrow \mathbf{X}\mathbf{\Omega}$
3: Compute thin QR $\mathbf{Y} = \mathbf{Q}\mathbf{R}$
4: Use pivoted QR on $\mathbf{Q}^\top$ to obtain indices $\mathcal{J}$ of well-conditioned rows of $\mathbf{Q}$
5: Compute $\mathbf{F} \leftarrow \mathbf{Q}(\mathbf{Q}(\mathcal{J},:))^{-1}$

## 2.4 Tensor Notation and Preliminaries

We now discuss the necessary background for tensors. For more details, see [48] for a good review.

We denote a $d$-mode tensor as $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times\cdots\times I_d}$ with entries

$$x_{i_1,\ldots,i_d}, \qquad 1 \leq i_j \leq I_j, \qquad j = 1,\ldots,d.$$

**Tensor Components.** Tensors have mode fibers as higher order equivalents to matrix rows and columns. Just as matrices (tensors of dimension $d = 2$) have two types of fibers, columns and rows, a $d$-mode tensor has $d$ types of fibers. For third-order tensors, the mode fibers have specific names. Mode-1 fibers $\mathbf{x}_{:i_2i_3}$ are called column fibers, mode-2 fibers $\mathbf{x}_{i_1:i_3}$ are called row fibers, and mode-3 fibers $\mathbf{x}_{i_1i_2:}$ are called tube fibers. In higher dimensions, these components are just referred to as mode-$j$ fibers. After fibers have been extracted from tensors, they are just assumed to be column vectors.

Similarly to fibers, we can define slices of a tensor as two-dimensional sections, obtained by fixing all but two indices of a tensor. For a third-order tensor, horizontal slices are $\mathbf{X}_{i_1::}$, vertical slices are $\mathbf{X}_{:i_2:}$, and frontal slices are $\mathbf{X}_{::i_3}$.

**Matricization.** A tensor can be "unfolded" into a matrix by reordering the elements, and this process is known as matricization. There are $d$ different unfoldings for a $d$-mode tensor. Each mode-$j$ unfolding arranges the resulting matrix so that the columns of the matrix are the mode-$j$ fibers of the tensor. The mode-$j$ unfolding is denoted as $\mathbf{X}_{(j)} \in \mathbb{R}^{I_j \times (\prod_{k \neq j} I_k)}$ for $j = 1, \ldots, d$. As an example, consider the 3-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{2\times 2\times 2}$ with slices

$$\boldsymbol{\mathcal{X}}_{::1} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}, \quad \boldsymbol{\mathcal{X}}_{::2} = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}.$$

Then the three mode unfoldings are as follows:

$$\mathbf{X}_{(1)} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \end{bmatrix}, \quad \mathbf{X}_{(2)} = \begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{bmatrix}, \quad \mathbf{X}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}.$$

**Tensor product.** The tensor product (or mode product) is a fundamental operation for multiplying a tensor by a matrix. Given a matrix $\mathbf{A} \in \mathbb{R}^{K\times I_j}$, the mode-$j$ product of a tensor $\boldsymbol{\mathcal{X}}$ with $\mathbf{A}$ is denoted $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_j \mathbf{A}$, and has dimensions $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1\times \ldots I_{j-1}\times K\times I_{j+1}\times \cdots \times I_d}$. More specifically, the product can be expressed in terms of the entries of the tensor as

$$\boldsymbol{\mathcal{Y}}_{i_1,\ldots,i_{j-1},k,i_{j+1},\ldots,i_d} = \sum_{i_j=1}^{I_j} x_{i_1,\ldots,i_d} a_{ki_j}, \qquad 1 \leq k \leq K, \qquad j = 1, \ldots, d.$$

The tensor product can also be expressed as the product of two matrices. That is, we can write $\boldsymbol{\mathcal{Y}}_{(j)} = \mathbf{A}\mathbf{X}_{(j)}$ for $j = 1, \ldots, d$. Note that tensor products across distinct modes commute, and multiplying a tensor $\boldsymbol{\mathcal{X}}$ with $d$ matrices $\mathbf{A}_j$, $j = 1, \ldots, d$ across modes $1, \ldots, d$ respectively is written as $\boldsymbol{\mathcal{X}} \times_{j=1}^{d} \mathbf{A}_j$. The following lemma will be useful in our analysis.

**Lemma 2.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times \cdots \times I_d}$ and let $\mathbf{\Pi}_j \in \mathbb{R}^{I_j \times I_j}$ be a sequence of $d$ orthogonal projectors for*

$j = 1, 2, \ldots, d$. *Then,*

$$\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \bigtimes_{j=1}^{d} \mathbf{\Pi}_j\|_F^2 = \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}} \bigtimes_{i=1}^{j-1} \mathbf{\Pi}_i \times_j (\mathbf{I} - \mathbf{\Pi}_j)\|_F^2 \leq \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}} \times_j \mathbf{\Pi}_j\|_F^2.$$

The proof of this lemma can be found in [75, Theorem 5.1].

**Multirank and the Tucker representation.** There are many different definitions for tensor rank depending on the tensor decomposition form. For the format we will use in this thesis, the Tucker form, we consider the multirank of a tensor. The Tucker format of a tensor $\boldsymbol{\mathcal{X}}$ of rank $(r_1, \ldots, r_d)$ consists of a core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times \cdots \times r_d}$ and factor matrices $\{\mathbf{A}_j\}_{j=1}^d$ with each $\mathbf{A}_j \in \mathbb{R}^{I_j \times r_j}$ such that $\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \bigtimes_{j=1}^d \mathbf{A}_j$. We define the desired size of the core tensor, $(r_1, \ldots, r_d)$, as the target rank. For short, the Tucker representation is written as $[\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$. A three-dimensional Tucker representation where the rank of the Tucker form is smaller than the original dimensions is shown in Figure 2.1.

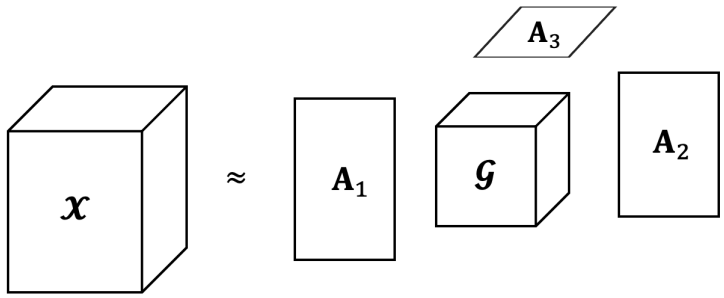


**Figure 2.1** A Tucker representation of 3-mode tensor $\boldsymbol{\mathcal{X}}$, where the rank of the Tucker form is less than the original dimensions.

Note that storing a tensor in Tucker form is beneficial as it requires less storage than a full tensor when the rank of the Tucker form is significantly less than the original dimensions. For a $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times I \times \cdots \times I}$ and target rank $(r, r, \ldots, r)$ with $r \ll I$, the cost of storing the Tucker form of $\boldsymbol{\mathcal{X}}$ is $\mathcal{O}(r^d + drI)$, compared to $\mathcal{O}(I^d)$ for a full tensor.

**Kronecker products.** The Kronecker product of two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{k \times \ell}$ is

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{mk \times n\ell}.$$

We also note some properties of Kronecker products that will be useful in our analysis, namely

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}, \qquad (\mathbf{A} \otimes \mathbf{B})^\top = \mathbf{A}^\top \otimes \mathbf{B}^\top.$$

Kronecker products are also useful for expressing tensor mode products in terms of matrix-matrix multiplications. Suppose $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_{j=1}^{d} \mathbf{A}_j$, then

$$\mathbf{Y}_{(j)} = \mathbf{A}_j \mathbf{X}_{(j)} (\mathbf{A}_d^\top \otimes \mathbf{A}_{d-1}^\top \otimes \cdots \otimes \mathbf{A}_{j+1}^\top \otimes \mathbf{A}_{j-1}^\top \otimes \cdots \otimes \mathbf{A}_1^\top). \tag{2.3}$$

### 2.4.1 HOSVD/STHOSVD

The *Higher Order SVD* (HOSVD) and *Sequentially Truncated Higher Order SVD* (STHOSVD) are two popular algorithms for computing low-rank tensor decompositions in the Tucker format. Note that, for these algorithms, the factor matrices $\mathbf{A}_j$ all have orthonormal columns.

**HOSVD.** In the HOSVD algorithm, each mode is handled separately. The factor matrix $\mathbf{A}_j$ is formed from the first $r_j$ left singular vectors of $\mathbf{X}_{(j)}$. Once all factor matrices $\mathbf{A}_j$ are found, the core tensor is formed by $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_{j=1}^{d} \mathbf{A}_j^\top$. Details of the process can be found in Algorithm 3.

**Algorithm 3** HOSVD

**Input:** $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, target rank vector $\mathbf{r} \in \mathbb{N}^d$
**Output:** $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

1: **for** $j = 1 : d$ **do**
2: Compute SVD of $\mathbf{X}_{(j)}$ to obtain $\mathbf{X}_{(j)} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$
3: Compress $\mathbf{U} \leftarrow \mathbf{U}(:, 1 : r_j)$, $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Sigma}(1 : r_j, 1 : r_j)$, and $\mathbf{V} \leftarrow \mathbf{V}(:, r_j)$
4: Set $\mathbf{A}_j \leftarrow \mathbf{U}$
5: **end for**
6: Form $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_{j=1}^{d} \mathbf{A}_j^\top$

The error in approximating $\boldsymbol{\mathcal{X}}$ using the HOSVD is bounded by the sum of the error in each mode, as shown in the following theorem, taken from [75, Theorem 5.1].

**Theorem 3.** *Let* $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ *be the rank-*$\mathbf{r}$ *approximation to* $d$*-mode tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ *using Algorithm 3. Then*

$$\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 \leq \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}} \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F^2 = \sum_{j=1}^{d} \sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{X}_{(j)}).$$

This theorem says that the error in the rank $\mathbf{r}$ approximation of the tensor $\boldsymbol{\mathcal{X}}$ computed using the HOSVD is the sum of squares of the discarded singular values from each mode unfolding. To simplify the upper bound, we introduce the notation

$$\Delta_j^2(\boldsymbol{\mathcal{X}}) \equiv \sum_{i=r_j+1}^{I_j} \sigma_i^2(\mathbf{X}_{(j)}), \qquad j = 1, \ldots, d. \tag{2.4}$$

With this notation, the error in the HOSVD satisfies $\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left(\sum_{j=1}^d \Delta_j^2(\boldsymbol{\mathcal{X}})\right)^{1/2}$.

**STHOSVD.** An alternative to the HOSVD is the *sequentially truncated HOSVD* (STHOSVD) algorithm which also produces a compressed representation in the Tucker format. STHOSVD processes the modes sequentially, which makes the order in which the modes are processed important since we may obtain different approximations by using different orders. We call the processing order $\rho$, which is a vector of length $d$. For the analysis, we will use $\rho = [1, 2, \ldots, d]$ for ease of notation.

At each stage of the STHOSVD algorithm, the core tensor $\boldsymbol{\mathcal{G}}$ is unfolded, and the factor matrix $\mathbf{A}_j$ is formed by taking the first $r_j$ left singular vectors. The new core tensor $\boldsymbol{\mathcal{G}}^{(j)}$ is obtained by projecting the previous core tensor onto the subspace spanned by the columns of $\mathbf{A}_j$. We then have a $j$-th partial approximation, defined as $\widehat{\boldsymbol{\mathcal{X}}}^{(j)} = \boldsymbol{\mathcal{G}}^{(j)} \times_{i=1}^{j} \mathbf{A}_i$. Details for this approximation process can be found in Algorithm 4.

**Algorithm 4** STHOSVD

**Input:** $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, target rank vector $\mathbf{r} \in \mathbb{N}^d$, processing order $\rho \in \mathbb{N}^d$
**Output:** $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

1: Set $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}}$
2: **for** j = 1:d **do**
3: Compute SVD of $\mathbf{X}_{(\rho_j)}$ to obtain $\mathbf{X}_{(\rho_j)} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$
4: Compress $\mathbf{U} \leftarrow \mathbf{U}(:, 1 : r_{\rho_j})$, $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Sigma}(1 : r_{\rho_j}, 1 : r_{\rho_j})$, and $\mathbf{V} \leftarrow \mathbf{V}(:, r_{\rho_j})$
5: Set $\mathbf{A}_{\rho_j} \leftarrow \mathbf{U}$
6: Update $\mathbf{G}_{(\rho_j)} \leftarrow \boldsymbol{\Sigma}\mathbf{V}^\top$ {Note: overwriting $\mathbf{G}_{(\rho_j)}$ overwrites $\boldsymbol{\mathcal{G}}$}
7: **end for**
8: $\boldsymbol{\mathcal{G}} \leftarrow \mathbf{G}_{(\rho_d)}$, in tensor form

The approximation error in this case is the sum of errors in the successive approximations, and has the same upper bound as that of HOSVD. This is shown in the following theorem, which assumes that the processing order is $\rho = [1, 2, \ldots, d]$. If a different processing order is taken, the upper bound will remain the same, so this assumption is made purely for ease of notation. The proof of this theorem can be found in [75, Theorem 6.5].

**Theorem 4.** *Let $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ be the rank-$\mathbf{r}$ approximation to $d$-mode tensor $\boldsymbol{\mathcal{X}}$ using Algorithm 4 with processing order $\rho = [1, 2, \ldots, d]$. Then,*

$$\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 = \sum_{j=1}^{d} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 \leq \sum_{j=1}^{d} \|\boldsymbol{\mathcal{X}} \times_j (\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\|_F^2 = \sum_{j=1}^{d} \Delta_j^2(\mathbf{X}_{(j)}).$$

The computational cost of the STHOSVD is lower than the HOSVD, which was established in [75] but is also reviewed in Subsection 3.2.3. Although both the error in the HOSVD and the STHOSVD satisfy the same upper bound, it is not clear which algorithm has a lower error. There is strong numerical evidence to suggest that STHOSVD typically has a lower error, although counterexamples to this claim have been found [75]. For these reasons, STHOSVD is preferable to HOSVD since it has a lower cost and the same worst case error bound. A downside to STHOSVD is that the processing order $\rho$ has to be determined in advance; some heuristics for this choice are given in [75].

### 2.4.2 Best Approximation

We would like to find an optimal rank-$\mathbf{r}$ approximation of a given tensor $\boldsymbol{\mathcal{X}}$, which we will denote as $\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}$. Let $\mathcal{S} = \{\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d} : \text{rank}(\mathbf{Y}_{(j)}) \leq r_j, j = 1, \ldots, d\}$. Then $\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}$ is an optimal tensor defined as satisfying the condition

$$\min_{\boldsymbol{\mathcal{Y}} \in \mathcal{S}} \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}\|_F = \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

The Eckart-Young theorem [28] states that an optimal rank-$r$ approximation to a matrix $\mathbf{A}$ can be constructed using the SVD truncated to rank-$r$. Unfortunately, an analog of this result for Tucker forms does not exist in higher dimensions [47]. The existence of $\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}$ is guaranteed by [38, Theorem 10.8]; however, this minimizer is not unique since Tucker representations are not unique [48, Section 4.3]. In general, computing $\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}$ requires solving an optimization problem that has no closed-form solution. In [24], the higher order orthogonal iteration (HOOI) was proposed to compute the "best" approximation by generating a sequence of iterates by repeatedly cycling through the modes sequentially. Because the HOOI algorithm requires many iterations with the tensor $\boldsymbol{\mathcal{X}}$, its implementation for large-scale tensors is challenging because of the overwhelming computational cost. Although neither the HOSVD nor the STHOSVD produce an optimal rank-$\mathbf{r}$ approximation, they do satisfy the inequality

$$\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \sqrt{d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F. \tag{2.5}$$

The proofs are available for the HOSVD (Theorem 10.3) and STHOSVD (Theorem 10.5) in [38]. The proof requires the observation that

$$\Delta_j(\boldsymbol{\mathcal{X}}) \leq \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F, \qquad j = 1, \ldots, d. \tag{2.6}$$

We highlight this inequality since it will be important for our subsequent analysis. The inequality (2.5) suggests that the outputs of the HOSVD and the STHOSVD are accurate for low dimensions and can be employed in three different ways: either as approximations to $\widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}$, as starting guesses to the HOOI algorithm, or to fit CP models.

# Chapter 3

# Randomized Low-rank Approximation Algorithms for Tucker Decompositions

## 3.1 Introduction

Tensors, or multi-way arrays, appear in a wide range of applications such as signal processing; neuroscientific applications such as Electroencephalography; data mining; seismic data processing; machine learning applications such as facial recognition, handwriting digit classification, and latent semantic indexing; imaging; astronomy; and uncertainty quantification. For example, a database of gray scale images constitutes a third order array when each image is stored as a matrix, while a numerical simulation of a system of a partial differential equations (PDEs) in three-dimensional space when tracking several parameters over time yields a five-dimensional dataset. Often, these datasets are treated as matrices rather than as tensors, suggesting that additional structure that could be leveraged for gaining insight and lowering computational cost is often underutilized and undiscovered.

A key step in processing and studying these datasets involves a compression step either to find an economical representation in memory or to find principal directions of variability. While working with tensors there are many possible formats one may consider, and each format is equipped with a different notion of compression and rank. Examples of tensor formats include CANDECOMP/PARAFAC (CP), Tucker, Hierarchical Tucker, and Tensor Train, all of which have their respective benefits (see surveys [48, 32, 21, 22]). The CP format, which represents a tensor as a sum of rank-1 outer products, gives a compact and unique (under certain conditions) representation. Tucker generally finds a better fit for data by estimating the subspace of each mode, while Hierarchical-Tucker and Tensor Train are useful for very high-dimensional tensors, i.e., tensors with a large number of modes, since the cost of storing the tensor scales exponentially with the dimension of the tensor [32]. In this chapter, we focus on the Tucker representation

which is known to have good compression properties. Given a multilinear rank $\mathbf{r}$, the Tucker form gives a representation of a tensor as a product of a core tensor and factor matrices typically having orthonormal columns. Popular algorithms for compression in the Tucker format can be found in [23, 75, 24], and a survey of approximation techniques can be found in [32]. Using these algorithms, high compression ratios can be achieved if the target rank for the Tucker approximation is small compared to the original dimensions. Even if the data is not highly compressible, representing it in the Tucker format can give insight into its principal directions [76].

In recent years, randomized matrix algorithms have gained popularity for developing low-rank matrix approximations (see reviews [41, 54, 26]). These algorithms are easy to implement, computationally efficient for a wide range of matrices (e.g. sparse matrices, matrices that can be accessed only via matrix-vector products, and dense matrices that are difficult to load in memory), and have accuracy comparable with non-randomized algorithms. There is also well-developed error analysis applicable to several classes of random matrices for randomized algorithms. Even more recently, randomized algorithms have been developed for tensor decompositions (see below for a detailed review). Motivated by this success, we analyze existing randomized tensor algorithms and propose and analyze new randomized tensor algorithms.

**Contributions and Contents.** In Section 3.2, we present analyses of randomized versions of HOSVD and STHOSVD (proposed in [89] and [17] respectively). Our contributions here include the probabilistic analysis of the randomized versions of these algorithms, as well as analysis of the associated computational costs. In Section 3.3, we present adaptive randomized algorithms to compute low-rank tensor decompositions for use in applications where the target rank is not known beforehand. In Section 3.4, we present a new randomized compression algorithm for large tensors, which produces a low-rank decomposition whose core tensor has entries taken directly from the tensor of interest. In this sense, the core tensor preserves the structure (e.g., sparsity, non-negativity) of the original tensor. For sparse tensors, our algorithm has the added benefit that the intermediate and final decompositions can be stored efficiently, thus enabling the computation of low-rank tensor decompositions of large, sparse tensors. We also provide a probabilistic error analysis of this algorithm. Finally, in Section 3.5, we test the performance of all algorithms on several synthetic tensors and real-world datasets, and discuss the performance of the proposed bounds.

**Related Work.** Several randomized algorithms have been proposed for computing low-rank tensor decompositions, e.g., Tucker format [17, 88, 51, 57, 73, 29, 89], CP format [29, 6, 10, 78], t-product [88], tensor networks [5], and Tensor Train format [17, 43]. Our work is most similar to [17, 29, 89]. The algorithm for randomized HOSVD is presented in [89], and the corresponding analysis is presented in [29] (both unpublished manuscripts). Randomized and adaptive versions of the STHOSVD were proposed and analyzed in [17], but our manuscript provides probabilistic error analysis for a different distribution of random matrices (see Section 3.2 for a justification of

our choice). To our knowledge, our proposed algorithm for producing structure-preserving tensor decompositions and the corresponding error analysis are novel. Related to this algorithm is the CUR-type decomposition for tensors proposed in [64, 25]. In contrast, our algorithm produces decompositions in which the core tensor (rather than the factor matrices in the aforementioned references) retains entries from the original tensor. This allows for the decomposition of extremely large sparse tensors as the sparsity is maintained at each step of the algorithm.

## 3.2 Randomized HOSVD/STHOSVD

In this section, we review randomized algorithms that are modified versions of the HOSVD and STHOSVD. We also develop rigorous error analysis and compare the two algorithms in terms of computational cost.

### 3.2.1 Algorithms

To obtain the randomized version of HOSVD, first proposed in [89], a full SVD of each mode unfolding is replaced with a randomized SVD of each mode unfolding to construct the factor matrix. The procedure to compute the core tensor remains unchanged. This is reflected in Algorithm 5, and we call this the R-HOSVD algorithm. The randomized version of STHOSVD

**Algorithm 5** Randomized HOSVD

**Input:** $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, target rank vector $\mathbf{r} \in \mathbb{N}^d$,
oversampling parameter $p \geq 0$ satisfying (3.1)
**Output:** $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$
**for** $j = 1 : d$ **do**
  Draw random Gaussian matrix $\boldsymbol{\Omega}_j \in \mathbb{R}^{\prod_{i \neq j} I_i \times (r_j + p)}$
  $[\widehat{\mathbf{U}}, \widehat{\boldsymbol{\Sigma}}, \widehat{\mathbf{V}}] = \text{RandSVD}(\mathbf{X}_{(j)}, r_j, p, \boldsymbol{\Omega}_j)$
  Set $\mathbf{A}_j \leftarrow \widehat{\mathbf{U}}$
**end for**
Form $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_{j=1}^{d} \mathbf{A}_j^\top$

is obtained in a similar way; at each step, the SVD of the unfolded core tensor is replaced with a randomized SVD. This is shown in Algorithm 6 (we call this R-STHOSVD) and is similar to the algorithm proposed in [17]. We briefly comment on the choice of the random matrices $\{\boldsymbol{\Omega}_j\}_{j=1}^d$. The R-HOSVD proposed in [89] uses standard Gaussian random matrices, which we also adopt in this chapter. The authors in [17] advocate $\boldsymbol{\Omega}_j$ constructed as a Khatri-Rao product of Gaussian random matrices, which is less memory intensive compare to standard Gaussian random matrices. A recent article [71] also recommends the Sparse Rademacher and Scrambled Subsampled Randomized Fourier Transform as appropriate choices for the random matrices. While the standard Gaussian random matrices may be criticized for large storage costs, we

note that these matrices need not be stored explicitly and can be generated on-the-fly, either column-wise or in appropriately sized blocks.

**Algorithm 6** Randomized STHOSVD

**Input:** $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, processing order $\rho$, target rank vector $\mathbf{r} \in \mathbb{N}^d$,
oversampling parameter $p \geq 0$ satisfying (3.5)
**Output:** $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$
Set $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}}$
**for** $j = 1 : d$ **do**
  Draw random Gaussian matrix $\boldsymbol{\Omega}_{\rho_j} \in \mathbb{R}^{z_j \times (r_{\rho_j}+p)}$, where $z_j$ is defined in (3.4)
  $[\widehat{\mathbf{U}}, \widehat{\boldsymbol{\Sigma}}, \widehat{\mathbf{V}}] = \text{RandSVD}(\mathbf{G}_{(\rho_j)}, r_{\rho_j}, p, \boldsymbol{\Omega}_{\rho_j})$
  Set $\mathbf{A}_{\rho_j} \leftarrow \widehat{\mathbf{U}}$
  Update $\mathbf{G}_{(\rho_j)} \leftarrow \widehat{\boldsymbol{\Sigma}}\widehat{\mathbf{V}}^\top$ {Note: overwriting $\mathbf{G}_{(\rho_j)}$ overwrites $\boldsymbol{\mathcal{G}}$.}
**end for**
$\boldsymbol{\mathcal{G}} \leftarrow \mathbf{G}_{(\rho_d)}$, in tensor form

### 3.2.2 Error Analysis

In the results below, we assume that the matrices $\{\boldsymbol{\Omega}_j\}_{j=1}^d$ are standard Gaussian random matrices of appropriate sizes. Here, we provide error bounds in expectation, but we can extend this analysis to develop concentration results that give insight into the tail bounds. These can be obtained by combining our analysis with the results from, e.g., Theorem 5.8 in [35].

**Theorem 5** (Randomized HOSVD)**.** *Let $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ be the output of Algorithm 5 with inputs target rank $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ satisfying $r_j \leq \mathsf{rank}(\mathbf{X}_{(j)})$ for $j = 1, \ldots, d$ and oversampling parameter $p \geq 2$ satisfying*

$$r_j + p \leq \min\{I_j, \prod_{i \neq j} I_i\}, \qquad j = 1, \ldots, d. \tag{3.1}$$

*Then, the expected error in the approximation satisfies*

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left(\sum_{j=1}^d \left(1 + \tfrac{r_j}{p-1}\right) \Delta_j^2(\boldsymbol{\mathcal{X}})\right)^{1/2} \tag{3.2}$$

$$\leq \left(d + \tfrac{\sum_{j=1}^d r_j}{p-1}\right)^{1/2} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{opt}\|_F. \tag{3.3}$$

*Proof.* Using Lemma 2 , we can write

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 \leq \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \sum_{j=1}^d \|\boldsymbol{\mathcal{X}} \times_j (\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\|_F^2 = \sum_{j=1}^d \mathbb{E}_{\boldsymbol{\Omega}_j} \|\boldsymbol{\mathcal{X}} \times_j (\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\|_F^2,$$

where the equality comes from linearity of expectations and the independence of $\mathbf{\Omega}_j$ for each mode $j$. We can unfold each term in the summation as $\|\boldsymbol{\mathcal{X}} \times_j (\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\|_F^2 = \|(\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\mathbf{X}_{(j)}\|_F^2$. Then, by applying Theorem 2, we can bound the expected value of the squared error in each mode to obtain

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 \leq \sum_{j=1}^{d}\left(1 + \frac{r_j}{p-1}\right)\Delta_j^2(\boldsymbol{\mathcal{X}}).$$

Finally, Hölder's inequality gives

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left(\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2\right)^{1/2} \leq \left(\sum_{j=1}^{d}\left(1 + \frac{r_j}{p-1}\right)\Delta_j^2(\boldsymbol{\mathcal{X}})\right)^{1/2}.$$

For the second inequality, recall that $\Delta_j^2(\boldsymbol{\mathcal{X}}) \leq \|\boldsymbol{\mathcal{X}} - \hat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F^2$ from (2.6). Thus, combined with the previous inequality, we have

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left(d + \frac{\sum_{j=1}^{d} r_j}{p-1}\right)^{1/2}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

□

To compare this result to the approximation error obtained using the HOSVD algorithm, we consider a few special cases. Let $r = \max_{1\leq j\leq d} r_j$. Then, if $p = r+1$, these bounds take the form

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \sqrt{2}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{HOSVD}}\|_F \leq \sqrt{2d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

Similarly, if we choose $p = \lceil \frac{r}{\epsilon} \rceil + 1$ for some $\epsilon > 0$, the error satisfies

$$\mathbb{E}_{\{\mathbf{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \sqrt{1+\epsilon}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{HOSVD}}\|_F \leq \sqrt{d(1+\epsilon)}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

This shows that the application of a randomized SVD in each mode of the tensor does not seriously deteriorate the accuracy compared to using an SVD. Note that the computational costs of these choices are higher.

Now consider the randomized STHOSVD approximation. For the probabilistic error analysis, it is important to note that at each intermediate step, the partially truncated core tensor is a random tensor. This is in contrast to the R-HOSVD, where we only needed to account for $\mathbf{\Omega}_j$ for each mode because the operations are independent across the modes.

For this theorem, we use the same notation introduced in Subsection 2.4.1 for the STHOSVD, in that the partially truncated core tensor at step $j$ is $\boldsymbol{\mathcal{G}}^{(j)} = \boldsymbol{\mathcal{X}} \times_{i=1}^{j} \mathbf{A}_i^\top$, giving a partial

approximation $\widehat{\boldsymbol{\mathcal{X}}}^{(j)} = \boldsymbol{\mathcal{G}}^{(j)} \times_{i=1}^{j} \mathbf{A}_i$. For convenience, given a processing order $\rho$ we define

$$z_j = \left( \prod_{i=\rho_1}^{\rho_{j-1}} r_i \right) \left( \prod_{i=\rho_{j+1}}^{\rho_d} I_i \right), \qquad j = 1, \ldots, d. \tag{3.4}$$

**Theorem 6** (Randomized STHOSVD)**.** *Let $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ be the output of Algorithm 6 with inputs target rank $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ satisfying $r_j \leq \mathsf{rank}(\mathbf{G}_{(j)}^{(j-1)})$ for $j = 1, \ldots, d$, processing order $\rho$, and oversampling parameter $p \geq 2$ and*

$$r_j + p \leq \min\{I_j, z_j\}, \qquad j = 1, \ldots, d. \tag{3.5}$$

*Then, the approximation error in expectation satisfies*

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{d}} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left( \sum_{j=1}^{d} \left( 1 + \frac{r_j}{p-1} \right) \Delta_j^2(\boldsymbol{\mathcal{X}}) \right)^{1/2} \tag{3.6}$$

$$\leq \left( d + \frac{\sum_{j=1}^{d} r_j}{p-1} \right)^{1/2} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{opt}\|_F. \tag{3.7}$$

*Proof.* We first assume that the processing order is $\rho = [1, \ldots, d]$ and then consider the general case. The first equality in Lemma 2 and the linearity of expectations together give

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{d}} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 = \sum_{j=1}^{d} \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{d}} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 = \sum_{j=1}^{d} \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j}} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2. \tag{3.8}$$

We have used the fact that the $j$-th term in the summation does not depend on the random matrices $\{\boldsymbol{\Omega}_k\}_{k>j}$. We first consider $\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j}} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2$. Since all the $\boldsymbol{\Omega}_k$'s are independent, we can write the expectation in an iterated form as

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j}} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 = \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j-1}} \left\{ \mathbb{E}_{\boldsymbol{\Omega}_j} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 \right\}.$$

The $j$-th term, which measures the difference in the sequential iterates, can be expressed as

$$\|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 = \|\boldsymbol{\mathcal{G}}^{(j-1)} \bigtimes_{i=1}^{j-1} \mathbf{A}_i \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F^2.$$

Now let

$$\mathbf{Z}_j \equiv \underbrace{\mathbf{I} \otimes \cdots \otimes \mathbf{I}}_{d-j \text{ terms}} \otimes \mathbf{A}_{j-1} \otimes \cdots \otimes \mathbf{A}_1.$$

If we unfold the difference $\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}$ along the $j$-th mode, using (2.3) we have

$$\begin{aligned} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F^2 &= \|(\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\mathbf{G}_{(j)}^{(j-1)}\mathbf{Z}_j^\top\|_F^2 \\ &\leq \|(\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\mathbf{G}_{(j)}^{(j-1)}\|_F^2. \end{aligned} \tag{3.9}$$

The inequality comes as $\mathbf{Z}_j$ has orthonormal columns for every $j$ since the factor matrices $\mathbf{A}_j$ all have orthonormal columns.

Now, let $\boldsymbol{\Gamma}_j = \mathbf{G}_{(j)}^{(j-1)} = \mathbf{X}_{(j)}\mathbf{Z}_j$ for simplicity. We take expectations and bound this last quantity in (3.9) using Theorem 2 (keeping $\{\boldsymbol{\Omega}_k\}_{k=1}^{j-1}$ fixed), as

$$\mathbb{E}_{\boldsymbol{\Omega}_j}\|(\mathbf{I} - \mathbf{A}_j\mathbf{A}_j^\top)\boldsymbol{\Gamma}_j\|_F^2 \leq \left(1 + \frac{r_j}{p-1}\right)\sum_{i=r_j+1}^{I_j}\sigma_i^2(\boldsymbol{\Gamma}_j). \tag{3.10}$$

We recall the definition and properties of Loewner partial ordering; see Section 7.7 in [42]. Let $\mathbf{M}, \mathbf{N} \in \mathbb{R}^{n\times n}$ be symmetric; $\mathbf{M} \preceq \mathbf{N}$ means $\mathbf{N} - \mathbf{M}$ is positive semidefinite. For $\mathbf{S} \in \mathbb{R}^{n\times m}$, then $\mathbf{S}^\top\mathbf{M}\mathbf{S} \preceq \mathbf{S}^\top\mathbf{N}\mathbf{S}$. Furthermore, $\lambda_i(\mathbf{M}) \leq \lambda_i(\mathbf{N})$ for $i = 1, \ldots, n$. Since $\mathbf{Z}_j$ has orthonormal columns, $\mathbf{Z}_j\mathbf{Z}_j^\top$ is a projector so that

$$\boldsymbol{\Gamma}_j\boldsymbol{\Gamma}_j^\top = \mathbf{X}_{(j)}\mathbf{Z}_j\mathbf{Z}_j^\top\mathbf{X}_{(j)}^\top \preceq \mathbf{X}_{(j)}\mathbf{X}_{(j)}^\top,$$

and the singular values of $\boldsymbol{\Gamma}_j$, which are squared eigenvalues of $\boldsymbol{\Gamma}_j\boldsymbol{\Gamma}_j^\top$, satisfy

$$\sum_{i=r_j+1}^{I_j}\sigma_i^2(\boldsymbol{\Gamma}_j) \leq \sum_{i=r_j+1}^{I_j}\sigma_i^2(\mathbf{X}_{(j)}) = \Delta_j^2(\boldsymbol{\mathcal{X}}). \tag{3.11}$$

To summarize, (3.8),(3.9),(3.10),(3.11) combined give

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F^2 \leq \sum_{j=1}^d \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j-1}}\left(1 + \frac{r_j}{p-1}\right)\Delta_j^2(\boldsymbol{\mathcal{X}}) = \sum_{j=1}^d\left(1 + \frac{r_j}{p-1}\right)\Delta_j^2(\boldsymbol{\mathcal{X}}).$$

The equality follows since the tensor $\boldsymbol{\mathcal{X}}$ is deterministic. Finally, we have by Hölder's inequality and (2.4) that

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \left(\sum_{j=1}^d\left(1 + \frac{r_j}{p-1}\right)\Delta_j^2(\boldsymbol{\mathcal{X}})\right)^{1/2} \leq \left(d + \frac{\sum_{j=1}^d r_j}{p-1}\right)^{1/2}\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{\text{opt}}\|_F.$$

In the general case, when the processing order does not equal $\rho = [1, \ldots, d]$, the proof is similar. We only need to work with the processed order, and we omit the details. □

We make several observations regarding Theorem 6. First, the upper bound for the error is the same for the R-STHOSVD as for the R-HOSVD (Theorem 5); however, once again, we note

that the performance of the two algorithms may be different. Second, this result says that the upper bound for R-STHOSVD is independent of the processing order. This means that while some processing orders may result in more accurate decompositions, every processing order has the same worst-case error bound. Our recommendation is to pick a processing order that minimizes the computational cost; see Subsection 3.2.3 for details, and Table 3.4 for numerical results. Third, the discussion following Theorem 5 regarding the choice of the oversampling parameter is applicable to the R-STHOSVD as well.

### 3.2.3 Computational Cost

We discuss the computational costs of the proposed randomized algorithms and compare them against the HOSVD and the STHOSVD algorithms. We make the following assumptions. First, we assume that the tensors are dense and our implementations take no advantage of their structure. Second, we assume that the target ranks in each dimension are sufficiently small, i.e., $r_j \ll I_j$ so that we can neglect the computational cost of the QR factorization and the truncation steps of the RandSVD algorithm. Third, we assume that the random matrices used in the algorithms are standard Gaussian random matrices. If other distributions are used, the computational cost may be lower. Finally, for the STHOSVD and R-STHOSVD algorithms, we assume that the processing order is $\rho = [1, 2, \ldots, d]$.

The computational cost of both HOSVD and STHOSVD was discussed in [75], and is reproduced in Table 3.1. In this chapter, we also provide an analysis of the computational cost of R-HOSVD and R-STHOSVD, which is summarized in Table 3.1. The table includes the costs for both a general tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ with target rank $\mathbf{r} = (r_1, r_2, \ldots, r_d)$, as well as for the special case when $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times I \times \cdots \times I}$ with target rank $\mathbf{r} = (r, r, \ldots, r)$. For ease of notation, denote the product $\prod_{k=i}^{j} I_k$ by $I_{i:j}$, and similarly $\prod_{k=i}^{j} r_k = r_{i:j}$ for $1 \leq i \leq j \leq d$. The dominant costs of each algorithm lie in computing the SVD of the unfoldings (the first term in each summation) and forming the core tensor (the second term in each summation). We can see from Table 3.1 that the savings in randomizing both algorithms is roughly $r/I$. Since by assumption $r \ll I$, the randomized algorithms are expected to be much faster. See Subsection 3.5.2 for experiments exploring this.

**Processing order.** The error in the approximation obtained using the R-STHOSVD depends on the choice of the processing order; see the numerical experiment in Subsection 3.5.2. However, Theorem 6 suggests that the worst case error is independent of the processing mode. For this reason, we choose a processing order that minimizes the computational cost. Since the dominant cost at each step $j$ is a randomized SVD with a cost of $\mathcal{O}(r_{\rho_1:\rho_j} I_{\rho_j:\rho_d})$, we can minimize this cost by choosing to process the modes in decreasing order of sizes; i.e., we process the largest modes first. Note that this is in contrast to the approach taken by [75] for the STHOSVD, in that they process in the order of increasing mode sizes to minimize the cost of the standard SVD at each step.

**Table 3.1** Computational Cost for the HOSVD, R-HOSVD, STHOSVD, and R-STHOSVD algorithms. The first term in each expression is the cost of computing an SVD of the mode unfoldings, and the second is the cost of forming the core tensor.

| Algorithm | Cost for $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ | Cost for $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times \cdots \times I}$ |
|---|---|---|
| HOSVD | $\mathcal{O}\left(\sum_{j=1}^{d} I_j I_{1:d} + \sum_{j=1}^{d} r_{1:j} I_{j:d}\right)$ | $\mathcal{O}\left(dI^{d+1} + \sum_{j=1}^{d} r^j I^{d-j+1}\right)$ |
| R-HOSVD | $\mathcal{O}\left(\sum_{j=1}^{d} r_j I_{1:d} + \sum_{j=1}^{d} r_{1:j} I_{j:d}\right)$ | $\mathcal{O}\left(drI^{d} + \sum_{j=1}^{d} r^j I^{d-j+1}\right)$ |
| STHOSVD | $\mathcal{O}\left(\sum_{j=1}^{d} I_j r_{1:j-1} I_{j:d} + \sum_{j=1}^{d} r_{1:j} I_{j+1:d}\right)$ | $\mathcal{O}\left(\sum_{j=1}^{d} r^{j-1} I^{d-j+2} + r^j I^{d-j}\right)$ |
| R-STHOSVD | $\mathcal{O}\left(\sum_{j=1}^{d} r_{1:j} I_{j:d} + \sum_{j=1}^{d} r_{1:j} I_{j+1:d}\right)$ | $\mathcal{O}\left(\sum_{j=1}^{d} r^j I^{d-j+1} + r^j I^{d-j}\right)$ |

## 3.3 Adaptive Randomized Tensor Decompositions

In the algorithms described in the previous section, we had to assume prior knowledge of the target rank. This knowledge may not be available, or may be difficult to estimate, in practice. Given a tensor $\boldsymbol{\mathcal{X}}$, it is often desirable to produce a decomposition $\widehat{\boldsymbol{\mathcal{X}}}$ that satisfies

$$\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \varepsilon \|\boldsymbol{\mathcal{X}}\|_F,$$

where $0 < \varepsilon < 1$ is a user-defined parameter. Note that there may not be a unique tensor $\widehat{\boldsymbol{\mathcal{X}}}$ that satisfies this inequality, but it is desirable to find a tensor with a small multirank that does satisfy this inequality. We first explain the adaptive randomized algorithm to find a low-rank matrix approximation, and then explain how this can be extended to the tensor case.

Several adaptive randomized range finders for matrices have been proposed in the literature [41, 86]. Given a matrix $\mathbf{X}$ and a tolerance $\varepsilon > 0$, the goal is to find a matrix $\mathbf{Q}$ with orthonormal columns that satisfies

$$\|\mathbf{X} - \mathbf{Q}\mathbf{Q}^\top \mathbf{X}\| \leq \varepsilon \|\mathbf{X}\|. \tag{3.12}$$

The number of columns of $\mathbf{Q}$ is taken to be the rank of the low rank approximation. The adaptive algorithms begin with a small number of columns of the random matrix $\boldsymbol{\Omega}$ to estimate the range $\mathbf{Q}$ and then sequentially increase the number of columns of $\boldsymbol{\Omega}$ until the matrix $\mathbf{Q}$ satisfies (3.12). In this chapter, we use a version of the adaptive randomized range finding algorithm first proposed by [59] and subsequently refined in Algorithm 2 in [86]. We use Algorithm 2 from [86] and assume that it can be invoked as $\mathbf{Q} = \text{AdaptRangeFinder}(\mathbf{X}, \varepsilon, b)$, where $\mathbf{X}$ is the matrix to be approximated, $\varepsilon$ is the requested relative error tolerance, and $b$ is a blocking integer to determine how many columns of $\boldsymbol{\Omega}$ to draw at a time.

**Adaptive R-HOSVD.** We now explain how the adaptive range finder can be used for computing tensor factorizations. For the R-HOSVD, we apply this adaptive matrix algorithm to

each mode unfolding $\mathbf{X}_{(j)}$. This gives a factor matrices $\mathbf{A}_j$ for each mode $j = 1, \ldots, d$. Given some tolerance $\varepsilon$, the approximation error $\|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \varepsilon \|\boldsymbol{\mathcal{X}}\|_F$ can be achieved if we choose the factor matrices $\mathbf{A}_j$ to satisfy

$$\|\mathbf{X}_{(j)} - \mathbf{A}_j \mathbf{A}_j^\top \mathbf{X}_{(j)}\|_F = \|\boldsymbol{\mathcal{X}} \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F \leq \varepsilon \|\boldsymbol{\mathcal{X}}\|_F / \sqrt{d}.$$

Thus, we have apportioned an equal amount of error tolerance to each mode unfolding. Combined with Lemma 2, this ensures that an overall relative error $\varepsilon$ is achieved. This approach is summarized in Algorithm 7. Suppose we want a more flexible approach, in which a different tolerance $\epsilon_j$ is chosen for mode $j = 1, \ldots, d$. This may be necessary, if we know in advance that we want to avoid compressing along some modes. In general, we may use any sequence $\epsilon_j$, so long as it satisfies $(\sum_{j=1}^d \epsilon_j^2) = \varepsilon^2$. Indeed, setting $\epsilon_j = 0$ for selected modes ensures that no compression is performed across those modes. Note that the choice $\epsilon_j = \varepsilon/\sqrt{d}$ for $j = 1, \ldots, d$ automatically satisfies this equality.

**Algorithm 7** Adaptive R-HOSVD

**Input:** $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, tolerance $\varepsilon \geq 0$, blocking integer $b \geq 1$
**Output:** $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

1: **for** $j = 1 : d$ **do**
2: $\quad \mathbf{A}_j = \text{AdaptRangeFinder}(\mathbf{X}_{(j)}, \frac{\varepsilon}{\sqrt{d}}, b)$
3: **end for**
4: Form $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_{j=1}^d \mathbf{A}_j^\top$

**Adaptive R-STHOSVD.** The same approach can be extended to the R-STHOSVD algorithm. We define the intermediate tensors $\boldsymbol{\mathcal{X}}^{(j)} = \boldsymbol{\mathcal{X}} \times_{i=1}^j \mathbf{A}_i \mathbf{A}_i^\top$ for $j = 1, \ldots, d$ and $\boldsymbol{\mathcal{X}}^{(0)} = \boldsymbol{\mathcal{X}}$. Analogously, we define the intermediate core tensor $\boldsymbol{\mathcal{G}}^{(j)} = \boldsymbol{\mathcal{X}} \times_{i=1}^j \mathbf{A}_i^\top$ with $\boldsymbol{\mathcal{G}}^{(0)} = \boldsymbol{\mathcal{G}}$. Furthermore, we choose the processing order $\rho = [1, 2, \ldots, d]$ for simplicity. In this case, we choose the factor matrices $\mathbf{A}_j$ in order to ensure that the successive iterates satisfy

$$\|\boldsymbol{\mathcal{X}}^{(j-1)} - \boldsymbol{\mathcal{X}}^{(j)}\|_F = \|\boldsymbol{\mathcal{G}}^{(j-1)} \bigtimes_{i=1}^{j-1} \mathbf{A}_j \times_j (\mathbf{I} - \mathbf{A}_j \mathbf{A}_j^\top)\|_F \leq \frac{\varepsilon}{\sqrt{d}} \|\boldsymbol{\mathcal{X}}\|_F, \quad j = 1, \ldots, d.$$

Applying the first part of Lemma 2, we obtain

$$\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}^{(d)}\|_F^2 = \sum_{j=1}^d \|\boldsymbol{\mathcal{X}}^{(j-1)} - \boldsymbol{\mathcal{X}}^{(j)}\|_F^2 \leq \varepsilon^2 \|\boldsymbol{\mathcal{X}}\|_F^2.$$

The details are provided in Algorithm 8, which uses a general processing order. As with R-HOSVD, we may elect to use the same error tolerance $\varepsilon/\sqrt{d}$ for each mode unfolding, or use a

different tolerance $\epsilon_j$ for iterate $j = 1, \ldots, d$.

**Algorithm 8** Adaptive R-STHOSVD

**Input:** $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, processing order $\rho$, tolerance $\varepsilon \geq 0$, blocking integer $b \geq 1$

**Output:** $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

1: Set $\boldsymbol{\mathcal{G}} \leftarrow \boldsymbol{\mathcal{X}}$
2: **for** $j = 1 : d$ **do**
3: $\quad \mathbf{A}_{\rho_j} = \text{AdaptRangeFinder}(\mathbf{G}_{(\rho_j)}, \frac{\varepsilon}{\sqrt{d}}, b)$
4: $\quad$ Update $\mathbf{G}_{(\rho_j)} \leftarrow \mathbf{A}_{\rho_j}^\top \mathbf{G}_{(\rho_j)}$
5: **end for**
6: $\boldsymbol{\mathcal{G}} \leftarrow \mathbf{G}_{(\rho_d)}$, in tensor format

## 3.4 Structure-preserving decompositions

In this section, we are interested in computing a low-rank decomposition in the Tucker format in which the core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{r_1 \times \cdots \times r_d}$ has entries that are explicitly taken from the original tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$. That is, $\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \bigtimes_{j=1}^{d} \mathbf{A}_j$ where $\mathbf{A}_j \in \mathbb{R}^{I_j \times r_j}$ with $j = 1, \ldots, d$ are the factor matrices (that do not necessarily have orthonormal columns). We call such a decomposition *structure-preserving* since the core tensor retains favorable properties (e.g., sparsity, nonnegativity, binary or integer counts) of the original tensor. This generalizes a related decomposition proposed for matrices in [19]. Related to the structure-preserving decompositions, prior work includes a higher-order interpolatory decomposition [64, 25, 55] of the form

$$\boldsymbol{\mathcal{X}} \approx \boldsymbol{\mathcal{G}} \bigtimes_{j=1}^{d} \mathbf{C}_j, \qquad \boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \bigtimes_{j=1}^{d} \mathbf{C}_j^\dagger,$$

where the matrices $\{\mathbf{C}_j\}_{j=1}^d$ have entries from the original tensor (specifically, columns selected from the appropriate mode-unfoldings), and $^\dagger$ represents the Moore-Penrose pseudoinverse.

### 3.4.1 Algorithm

We first explain our algorithm for a matrix $\mathbf{X}$ and then generalize it to tensors. We compute a basis $\mathbf{Q}$ using the randomized range finding algorithm. Instead of computing the low-rank approximation $\mathbf{Q}\mathbf{Q}^\top\mathbf{X}$, as was done in Algorithm 1, we first identify a set of well-conditioned rows of $\mathbf{Q}$. This is implemented as a selection operator denoted by the matrix $\mathbf{P}$, which contains columns from the identity matrix. We then use the low-rank representation

$$\mathbf{X} \approx \mathbf{Q}(\mathbf{P}^\top\mathbf{Q})^{-1}\mathbf{P}^\top\mathbf{X} = \mathbf{A}\widehat{\mathbf{X}},$$

where $\mathbf{A} = \mathbf{Q}(\mathbf{P}^\top\mathbf{Q})^{-1}$ and $\widehat{\mathbf{X}} = \mathbf{P}^\top\mathbf{X}$. We see that the matrix $\mathbf{A}$ does not have orthonormal columns, but is well-conditioned, and that the matrix $\widehat{\mathbf{X}}$ contains rows from the matrix $\mathbf{X}$ as determined by the selection operator $\mathbf{P}$. We use strong rank-revealing QR (sRRQR) factorization [36, 27] for subset selection; other possibilities are discussed in Subsection 3.4.3.

The idea behind our algorithms is the following: at each step, given the core tensor $\boldsymbol{\mathcal{G}}$ we first unfold this tensor and use a randomized range finder on the unfolding to obtain a basis $\mathbf{Q}_j$. Then, we use the sRRQR algorithm to determine the selection operator that identifies well-conditioned rows of $\mathbf{Q}_j$; we use these same selection operator to select rows of $\mathbf{G}_{(j)}$ which then determines the core tensor for the next step. The details of the algorithm are provided in Algorithm 9. A few things are worth noting. First, the algorithm is structure-preserving in two ways: the core tensor at each step contains elements from the original tensor, so that the final core tensor also has entries from the original tensor; and the low-rank approximation reproduces certain elements of the tensor exactly (in exact arithmetic). Second, in contrast to Algorithms 5 and 6, the factor matrices do not have orthonormal columns; if orthonormal columns are desired, a postprocessing step can be performed (a thin-QR factorization of each factor matrix to obtain the basis, followed by an aggregation step in which the core tensor is multiplied with all the triangular factors). Third, the resulting tensor is of rank-$(r_1 + p, \ldots, r_d + p)$, which is also in contrast to other algorithms that produce decompositions of the rank $(r_1, \ldots, r_d)$. Once again, these factors can be recompressed using a post-processing step; see, for example, [51].

**Algorithm 9** Structure-preserving STHOSVD (SP-STHOSVD)

**Input:** $d$-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, target rank vector $\mathbf{r} \in \mathbb{N}^d$,
oversampling parameter $p \geq 0$ satisfying (3.14), processing order $\rho$

**Output:** $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$

Set $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}}$
**for** $j = 1 : d$ **do**
  Draw Gaussian matrix $\boldsymbol{\Omega}_{\rho_j} \in \mathbb{R}^{z'_j \times (r_{\rho_j} + p)}$, where $z'_j$ is defined in (3.13)
  Form $\mathbf{Y} \leftarrow \mathbf{G}_{(\rho_j)}\boldsymbol{\Omega}_{\rho_j}$
  Thin QR factorization $\mathbf{Y} = \mathbf{Q}_{\rho_j}\mathbf{R}$
  Use strong RRQR on $\mathbf{Q}_{\rho_j}^\top$ with parameter $\eta = 2$

$$\mathbf{Q}_{\rho_j}^\top \begin{bmatrix} \mathbf{S}_1 & \mathbf{S}_2 \end{bmatrix} = \mathbf{Z} \begin{bmatrix} \mathbf{N}_{11} & \mathbf{N}_{12} \end{bmatrix},$$

  where $\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{S}_2 \end{bmatrix}$ is a permutation matrix, $\mathbf{Z}$ is an orthogonal matrix, and $\mathbf{N}_{11}$ is upper triangular
  Let $\mathbf{P}_{\rho_j} = \mathbf{S}_1 \in \mathbb{R}^{I_{\rho_j} \times (r_{\rho_j} + p)}$ which contains the columns from the identity matrix
  Form $\mathbf{A}_{\rho_j} = \mathbf{Q}_{\rho_j}(\mathbf{P}_{\rho_j}^\top\mathbf{Q}_{\rho_j})^{-1}$
  Update $\mathbf{G}_{(\rho_j)} \leftarrow \mathbf{P}_{\rho_j}^\top\mathbf{G}_{(\rho_j)}$
**end for**
Set $\boldsymbol{\mathcal{G}} = \mathbf{G}_{(\rho_d)}$, in tensor format

Algorithm 9 is particularly beneficial for sparse tensors. Although sparse tensors can be efficiently stored in an appropriate tensor format (e.g., [3]), a straightforward application of either Algorithm 5 or Algorithm 6 produces dense intermediate tensors that may be prohibitively expensive to store, even though the final decomposition may be economical in terms of storage costs. On the other hand, in Algorithm 9, each intermediate core tensor is sparse, and only contains entries from the original tensor. Therefore, the intermediary core tensors can be efficiently stored in the same sparse tensor format. Additionally, the sparse tensor format permits cheaper tensor and matrix product computations.

**Computational Cost.** For simplicity, we assume a processing order of $\rho = [1, 2, \ldots, d]$. The two dominant costs of Algorithm 9 for each mode are obtaining the basis $\mathbf{Q}_j$ and computing an sRRQR to determine $\mathbf{P}_j$. Let $\mathsf{nnz}(\boldsymbol{\mathcal{G}}^{(j)})$ denote the number of nonzeros in the core tensor at step $j$. Letting $\ell_j = r_j + p$, the cost of forming the product of the (unfolded) core tensor with a random matrix $\boldsymbol{\Omega}_j$, over all $d$ modes is $\mathcal{O}(\sum_{j=1}^{d} \mathsf{nnz}(\boldsymbol{\mathcal{G}}^{(j)})\ell_j)$. Computing an sRRQR of an $I_j \times \ell_j$ matrix with parameter $\eta = 2$ (parameter was called $f$ in [36]) costs $\mathcal{O}(I_j \ell_j^2)$ per mode. Combining both the dominant costs gives a total cost of $\mathcal{O}\left(\sum_{j=1}^{d} \mathsf{nnz}(\boldsymbol{\mathcal{G}}^{(j)})\ell_j + \sum_{j=1}^{d} I_j \ell_j^2\right)$. This analysis shows that the computational cost of SP-STHOSVD is significantly smaller than the other algorithms presented thus far, particularly with a sparse tensor. Even when the original tensor is dense, there is a savings in computational cost compared to STHOSVD since a full SVD is not computed, and compared to R-STHOSVD since the core tensor $\boldsymbol{\mathcal{G}}^{(j)}$ is only multiplied once per iteration. We use the processing order in Subsection 3.2.3.

### 3.4.2 Error Analysis

We now present the error analysis for Algorithm 9. There are two major difficulties here in extending the proofs of Lemmas 3 and 4. First, we have to work with an oblique projector $\boldsymbol{\Pi}_j = \mathbf{Q}_j(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1}\mathbf{P}_j^\top$, whereas in the previous analysis we used an orthogonal projector. As a consequence, we can no longer use the Pythagorean theorem to obtain Lemma 2; instead we have to employ the triangle inequality, resulting in a weaker bound. Second, we have to work with the factor matrices $\mathbf{A}_j$ which no longer have orthonormal columns. Let us define $\ell_i = r_i + p$, $g(I, r) = \sqrt{1 + 4r(I - r)}$, and $f_p(r) = \sqrt{1 + \frac{r}{p-1}}$, and given a processing order $\rho$ we also define

$$z'_j = \left(\prod_{i=\rho_1}^{\rho_{j-1}} \ell_i\right)\left(\prod_{i=\rho_{j+1}}^{\rho_d} I_i\right), \qquad j = 1, \ldots, d. \tag{3.13}$$

We are able to derive the following error bound.

**Theorem 7.** *Let $\widehat{\boldsymbol{\mathcal{X}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}_1, \ldots, \mathbf{A}_d]$ be the output of Algorithm 9 with inputs target rank $\mathbf{r} = (r_1, \ldots, r_d)$ satisfying $r_j \leq \mathsf{rank}(\mathbf{G}_{(j)}^{(j-1)})$ for $j = 1, \ldots, d$, a processing order of $\rho = [1, 2, \ldots, d]$,*

*and oversampling parameter $p \geq 2$ satisfying*

$$r_j + p < \min\{I_j, z'_j\}, \qquad j = 1, \ldots, d \tag{3.14}$$

*Furthermore, let $\eta = 2$ be the sRRQR parameter. Then,*

1. *the matrices $\{\mathbf{A}_j\}_{j=1}^d$ each contain an $\ell_j \times \ell_j$ identity matrix and*

$$1 \leq \|\mathbf{A}_j\|_2 \leq g(I_j, \ell_j), \qquad j = 1, \ldots, d,$$

2. *$\widehat{\boldsymbol{\mathcal{X}}}$ exactly reproduces $\prod_{i=1}^d \ell_i$ entries of the original tensor $\boldsymbol{\mathcal{X}}$ (in exact arithmetic), and*

3. *the expected approximation error satisfies*

$$\begin{aligned}\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F &\leq \sum_{j=1}^d \left( \prod_{k=1}^j g(I_k, \ell_k) \right) f_p(r_j) \Delta_j(\boldsymbol{\mathcal{X}}) \\ &\leq \sum_{j=1}^d \left( \prod_{k=1}^j g(I_k, \ell_k) \right) f_p(r_j) \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}_{opt}\|_F.\end{aligned}$$

*Proof.* We tackle each item individually:

**1.** Consider the factor matrices $\mathbf{A}_j = \mathbf{Q}_j(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1}$ for $j = 1, \ldots, d$. Since $\mathbf{P}_j$ contains columns from the identity matrix, it is easy to verify that $\mathbf{A}_j$ contains the identity matrix as its submatrix. The lower bound on $\|\mathbf{A}_j\|_2$ follows immediately from this fact. For the upper bound, since $\mathbf{Q}_j$ has orthonormal columns,

$$\|\mathbf{A}_j\|_2 = \|(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1}\|_2 \leq g(I_j, \ell_j). \tag{3.15}$$

The last step follows from [27, Lemma 2.1], where $\eta = 2$ is used as the tuning parameter for the sRRQR algorithm.

**2.** Let $I_j$ be the chosen index set for each $\mathbf{P}_j$, $j = 1, \ldots, d$. Then if we pick the corresponding elements from $\widehat{\boldsymbol{\mathcal{X}}}$, denoted by $\widehat{\boldsymbol{\mathcal{X}}}(I_1, \ldots, I_d)$, we have

$$\widehat{\boldsymbol{\mathcal{X}}}(I_1, \ldots, I_d) = \widehat{\boldsymbol{\mathcal{X}}} \bigtimes_{j=1}^d \mathbf{P}_j^\top = \left( \boldsymbol{\mathcal{G}} \bigtimes_{j=1}^d \mathbf{A}_j \right) \bigtimes_{j=1}^d \mathbf{P}_j^\top = \boldsymbol{\mathcal{G}}.$$

Note that the last equality comes as $\mathbf{P}_j^\top \mathbf{A}_j = (\mathbf{P}_j^\top \mathbf{Q}_j)(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1} = \mathbf{I}$ for $j = 1, \ldots, d$. Then as $\boldsymbol{\mathcal{G}}$ consists of elements of the original tensor $\boldsymbol{\mathcal{X}}$ by construction, $\widehat{\boldsymbol{\mathcal{X}}}$ reproduces $\prod_{j=1}^d \ell_j$ elements of $\boldsymbol{\mathcal{X}}$.

**3.** Next, let $\boldsymbol{\mathcal{G}}^{(j)} = \boldsymbol{\mathcal{X}} \times_{i=1}^{j} \mathbf{P}_i^\top$ denote the partially truncated core tensor after the $j$-th step of Algorithm 9. Also let $\widehat{\boldsymbol{\mathcal{X}}}^{(j)} = \boldsymbol{\mathcal{G}}^{(j)} \times_{i=1}^{j} \mathbf{A}_i$ be the $j$-th partial approximation of $\boldsymbol{\mathcal{X}}$. Then, by the triangle inequality and the linearity of expectations, we have

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}}\|_F \leq \sum_{j=1}^{d} \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d} \|\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F. \tag{3.16}$$

Consider the term $\|\hat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \hat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F$. We can simplify $\hat{\boldsymbol{\mathcal{X}}}^{(j)}$ as

$$\begin{aligned}\hat{\boldsymbol{\mathcal{X}}}^{(j)} &= \boldsymbol{\mathcal{G}}^{(j)} \bigtimes_{i=1}^{j} \mathbf{A}_i = \left(\boldsymbol{\mathcal{G}}^{(j-1)} \times_j \mathbf{P}_j^\top\right) \bigtimes_{i=1}^{j} \mathbf{A}_i \\ &= \boldsymbol{\mathcal{G}}^{(j-1)} \bigtimes_{i=1}^{j-1} \mathbf{A}_i \times_j \mathbf{A}_j \mathbf{P}_j^\top = \boldsymbol{\mathcal{G}}^{(j-1)} \bigtimes_{i=1}^{j-1} \mathbf{A}_i \times_j \boldsymbol{\Pi}_j.\end{aligned}$$

Therefore, $\hat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \hat{\boldsymbol{\mathcal{X}}}^{(j)} = \boldsymbol{\mathcal{G}}^{(j-1)} \times_{i=1}^{j-1} \mathbf{A}_i \times_j (\mathbf{I} - \boldsymbol{\Pi}_j)$. Repeated use of the submultiplicativity inequality $\|\mathbf{MN}\|_F \leq \|\mathbf{M}\|_F \|\mathbf{N}\|_2$ gives,

$$\|\hat{\boldsymbol{\mathcal{X}}}^{(j-1)} - \hat{\boldsymbol{\mathcal{X}}}^{(j)}\|_F \leq \|\boldsymbol{\mathcal{G}}^{(j-1)} \times_j (\mathbf{I} - \boldsymbol{\Pi}_j)\|_F \prod_{i=1}^{j-1} \|\mathbf{A}_i\|_2. \tag{3.17}$$

Now, observe that $\boldsymbol{\Pi}_j \mathbf{Q}_j \mathbf{Q}_j^\top = \mathbf{Q}_j \mathbf{Q}_j^\top$, implying that $\mathbf{I} - \boldsymbol{\Pi}_j = (\mathbf{I} - \boldsymbol{\Pi}_j)(\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)$. Therefore, once again using the submultiplicativity inequality,

$$\begin{aligned}\|\boldsymbol{\mathcal{G}}^{(j-1)} \times_j (\mathbf{I} - \boldsymbol{\Pi}_j)\|_F &= \|\boldsymbol{\mathcal{G}}^{(j-1)} \times_j (\mathbf{I} - \boldsymbol{\Pi}_j)(\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)\|_F \\ &\leq \|\boldsymbol{\mathcal{G}}^{(j-1)} \times_j (\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)\|_F \|\mathbf{I} - \boldsymbol{\Pi}_j\|_2.\end{aligned} \tag{3.18}$$

We note that $\boldsymbol{\Pi}_j \neq \mathbf{I}$ since $\mathsf{rank}(\boldsymbol{\Pi}_j) \leq \mathsf{rank}(\mathbf{Q}_j) = r_j + p < \min\{I_j, z_j'\}$, and $\boldsymbol{\Pi}_j \neq \mathbf{0}$ since $\mathbf{Q}_j$ has orthonormal columns and $\mathbf{P}_j^\top \mathbf{Q}_j$ is invertible. Therefore, we can use [72, Theorem 2.1] to conclude $\|\mathbf{I} - \boldsymbol{\Pi}_j\|_2 = \|\boldsymbol{\Pi}_j\|_2$. Once again using [27, Lemma 2.1]

$$\|\mathbf{I} - \boldsymbol{\Pi}_j\|_2 = \|\boldsymbol{\Pi}_j\|_2 = \|(\mathbf{P}_j^\top \mathbf{Q}_j)^{-1}\|_2 \leq g(I_j, \ell_j).$$

Combining this inequality with (3.15),(3.18),and (3.17), we have

$$\|\widehat{\boldsymbol{\mathcal{X}}}^{(j)} - \widehat{\boldsymbol{\mathcal{X}}}^{(j-1)}\|_F \leq \|\boldsymbol{\mathcal{G}}^{(j-1)} \times_j (\mathbf{I} - \mathbf{Q}_j \mathbf{Q}_j^\top)\|_F \prod_{i=1}^{j} g(I_i, \ell_i).$$

Taking expectations, and using the independence of the random matrices, we obtain

$$\begin{aligned}\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\widehat{\boldsymbol{\mathcal{X}}}^{(j)}-\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)}\|_F &= \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j-1}}\mathbb{E}_{\boldsymbol{\Omega}_j}\|\boldsymbol{\mathcal{G}}^{(j-1)}\times_j(\mathbf{I}-\mathbf{Q}_j\mathbf{Q}_j^\top)\|_F\prod_{i=1}^j g(I_i,\ell_i)\\ &= \mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j-1}}\mathbb{E}_{\boldsymbol{\Omega}_j}\|(\mathbf{I}-\mathbf{Q}_j\mathbf{Q}_j^\top)\mathbf{G}_{(j)}^{(j-1)}\|_F\prod_{i=1}^j g(I_i,\ell_i)\\ &\leq \left(\prod_{i=1}^j g(I_i,\ell_i)\right) f_p(r_j)\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j-1}}\left(\sum_{i=r_j+1}^{I_j}\sigma_i^2(\mathbf{G}_{(j)}^{(j-1)})\right)^{1/2}.\end{aligned}$$

In the last step, we have used (2.1) and kept the random matrices $\{\boldsymbol{\Omega}_k\}_{i=1}^{j-1}$ fixed. By construction $\mathbf{G}_{(j)}^{(j-1)}$ is a submatrix of the mode-unfolding $\mathbf{X}_{(j)}$. Arguing as in the proof of Theorem 6, we can show $\sum_{i=r_j+1}^{I_j}\sigma_i^2(\mathbf{G}_{(j)}^{(j-1)})\leq\Delta_j^2(\boldsymbol{\mathcal{X}})$, for $j=1,\ldots,d$ and, therefore,

$$\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^d}\|\widehat{\boldsymbol{\mathcal{X}}}^{(j)}-\widehat{\boldsymbol{\mathcal{X}}}^{(j-1)}\|_F\leq\left(\prod_{i=1}^j g(I_i,\ell_i)\right)f_p(r_j)\mathbb{E}_{\{\boldsymbol{\Omega}_k\}_{k=1}^{j-1}}\Delta_j(\boldsymbol{\mathcal{X}}).$$

Plugging this into (3.16), and using (2.6) we get the desired bound.

□

In this result, we have assumed the standard processing order $\rho=[1,\ldots,d]$. The analysis can be extended to other processing orders; however, note that in this case the upper bound derived here will depend on the processing order, which is in contrast to the bound in Theorem 6. Although the error bound in Theorem 7 can be much higher than Theorem 6, numerical results suggest that the bound is somewhat pessimistic and, in practice, Algorithm 9 produces accurate low-rank decompositions.

### 3.4.3 Variants

One can easily develop several variations of Algorithm 9 that may be beneficial in applications.

1. **SP-HOSVD.** In Algorithm 9, we handled all the modes sequentially; they can however be handled independently. For each mode $j=1,\ldots,d$, we obtain a basis $\mathbf{Q}_j$ and a selection operator $\mathbf{P}_j$. The resulting decomposition is of the form $\boldsymbol{\mathcal{X}}\approx\widehat{\boldsymbol{\mathcal{X}}}=\boldsymbol{\mathcal{G}}\times_{j=1}^d\mathbf{A}_j$ where the core tensor $\boldsymbol{\mathcal{G}}=\boldsymbol{\mathcal{X}}\times_{j=1}^d\mathbf{P}_j^\top$ and the factor matrices are of the form $\mathbf{A}_j=\mathbf{Q}_j(\mathbf{P}_j^\top\mathbf{Q}_j)^{-1}$ for $j=1,\ldots,d$. The error analysis is similar to Theorem 7 and we found that it has the same upper bound.

2. **Range finding.** We used a basic version of the randomized range finding algorithm to obtain the matrices $\mathbf{Q}_j$. Other variations are certainly possible; for example, the adaptive range finding algorithm in Section 3.3, randomized subspace iteration [41, Algorithm 4.3], or other deterministic or randomized rank-revealing decompositions.

**3. Subset selection.** In Algorithm 9, we used the sRRQR algorithm for the subset selection step. In practice, this is computationally expensive, and an alternative is to use the Column-Pivoted QR factorization. This algorithm has lower computational cost but is known to fail for certain adversarial cases [36]. Besides deterministic algorithms for subset selection, there are several randomized techniques available, such as uniform sampling and leverage score sampling that can be used instead of sRRQR.

## 3.5 Numerical Results

In this section, we study the accuracy and the computational cost of our algorithms on several synthetic and real-world tensors. Most of our results were run on a desktop with a 3.4 GHz Intel Core i7 processing unit and 16GB memory. A few experiments were run on the NCSU Mathematics Department HPC Cluster with 72GB memory. We used two tensor packages in MATLAB, namely Tensor Toolbox [3] and Tensorlab [79].

### 3.5.1 Test Problems

We briefly describe the different tensors that we use to validate our algorithms.

**1. Hilbert Tensor.** Our first test tensor is a synthetic, super-symmetric tensor (invariant under the permutation of indices), where each entry is defined as

$$\boldsymbol{\mathcal{X}}_{i_1 i_2 \ldots i_d} = \frac{1}{i_1 + i_2 + \cdots + i_d} \qquad 1 \leq i_j \leq I_j,\ j = 1, \ldots, d. \tag{3.19}$$

We call this the *Hilbert tensor*, which generalizes the Hilbert matrix ($d = 2$). When $d = 5$ and each $I_j = 25$, this tensor has $25^5 = 9,765,625$ nonzero entries. The singular values of each mode-unfolding decay rapidly, which suggests that the randomized algorithms proposed in this chapter are likely to be accurate.

**2. Synthetic Sparse tensor.** For our second example, we construct a three-dimensional sparse tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{200 \times 200 \times 200}$ as the sum of outer products as

$$\boldsymbol{\mathcal{X}} = \sum_{i=1}^{10} \frac{\gamma}{i^2}\, \mathbf{x}_i \circ \mathbf{y}_i \circ \mathbf{z}_i + \sum_{i=11}^{200} \frac{1}{i^2}\, \mathbf{x}_i \circ \mathbf{y}_i \circ \mathbf{z}_i, \tag{3.20}$$

where $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i \in \mathbb{R}^n$ are sparse vectors for all $i$, and $\circ$ denotes the outer product. The sparse vectors are all generated using the `sprand` command with 5% nonzeros each. In this instance, the tensor $\boldsymbol{\mathcal{X}}$ has $185,211$ nonzeros in total. Furthermore, $\gamma$ is a user-defined parameter which determines the strength of the gap between the first ten terms and the last terms.

**3. Olivetti Dataset.** The classification of facial images, or "tensorfaces" as popularized by [76], has two main steps. The first is a compression phase, where a higher order SVD is applied to a training images dataset, arranged as a tensor, to compute a low-rank decomposition. The second step is a classification phase, in which the decomposed tensor is used to classify images in the test dataset. We focus on the first step to efficiently decompose the tensor formed using training images from the Olivetti dataset [2]. This dataset contains 400 images ($64 \times 64$ pixels) of 40 people in 10 different poses and can be expressed as a three dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{40\times 4096\times 10}$, in which the three modes represent people, pixels, and poses, respectively.

**4. FROSTT database.** Our final test problems come from the formidable repository of sparse tensors and tools (FROSTT) database [69]. From this database, we choose two representative large, sparse tensors whose features are summarized in Table 3.2. The NELL-2 dataset [15] is

**Table 3.2** Summary of sparse tensor examples from the FROSTT database—we include the details for both the full datasets and the condensed datasets used in our experiments.

| Original Tensor | Order | Size | Nonzeros |
|---|---|---|---|
| NELL-2 | 3 | $12092 \times 9184 \times 28818$ | $76,879,419$ |
| Enron | 4 | $6066 \times 5699 \times 244268 \times 1176$ | $54,202,099$ |
| **Condensed Tensor** | **Order** | **Size** | **Nonzeros** |
| NELL-2 | 3 | $807 \times 613 \times 1922$ | $19,841$ |
| Enron | 3 | $405 \times 380 \times 9771$ | $6,131$ |

a portion of the Never Ending Language Learning knowledge base from the "Read the Web" project at Carnegie Mellon University. NELL is a machine learning system that relates different entities, creating a three-dimensional dataset whose modes represent entity, relation, and entity. The Enron dataset [67] contains word counts in emails released during an investigation by the Federal Energy Regulatory Commission. Here, the modes represent sender, receiver, word, and date, respectively.

Although our implementation of SP-STHOSVD is capable of handling both full tensors in Table 3.2, the tensors are too large to compute the approximation error. In order to be able to compute this error, we first pared down the tensors, which allows us to compare the performance of our SP-STHOSVD algorithm with other algorithms. For the NELL-2 dataset [15], we subsample every 15 elements in each mode to obtain a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{807\times 613\times 1922}$. For the Enron dataset [67], we first condense the dataset to three dimensions by summing over the fourth mode. Then we subsample this tensor by taking every 15 elements from the first two modes, and every 25 from the third mode. This results in a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{405\times 380\times 9771}$.

### 3.5.2 Numerical Experiments

We now describe the experiments performed on the test tensors introduced in the previous subsection.

#### Fixed rank

Our first experiment compares the accuracy of the HOSVD and STHOSVD algorithms with their randomized counterparts, R-HOSVD and R-STHOSVD (Algorithms 5 and 6). As inputs, we take $\boldsymbol{\mathcal{X}}$ as defined in (3.19) with $d = 5$ modes and $I_j = 25$ for $j = 1, \ldots, d$. For each algorithm, we use the target rank $(r, r, r, r, r)$, where $r$ varies from 1 to 25, and the same oversampling parameter $p = 5$ was used in every mode. Since this is a super-symmetric tensor, the processing order of modes does not affect the results, so we take the processing order $\rho = [1, 2, 3, 4, 5]$. The relative error is plotted in Figure 3.1, where we can see that the approximation error of all four algorithms is very similar and that the randomized algorithms are also highly accurate. The

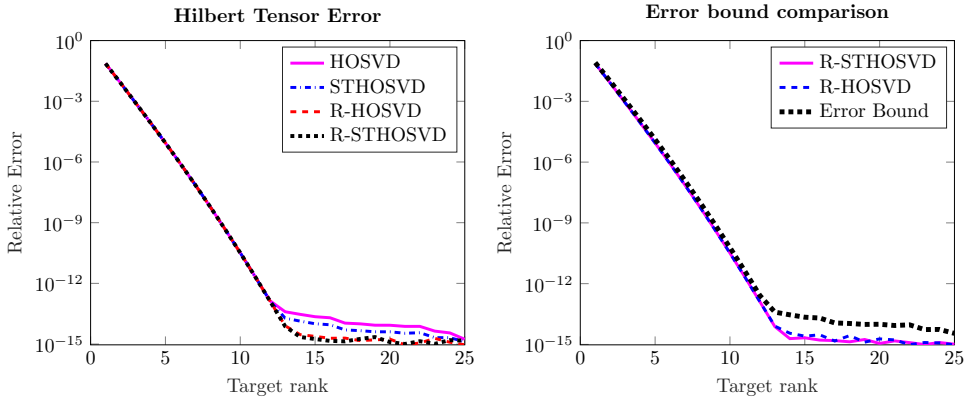


**Figure 3.1 Left:** Relative approximation error for 5-mode Hilbert tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{25\times25\times25\times25\times25}$ defined in (3.19), with target rank $(r, r, r, r, r)$ and oversampling parameter $p = 5$. **Right:** Actual relative error for $\boldsymbol{\mathcal{X}}$ from the R-HOSVD and R-STHOSVD algorithms compared to the calculated error bound as the target rank $(r, r, r, r, r)$ increases. Both algorithms use oversampling parameter $p = 5$, and R-STHOSVD uses the processing order $\rho = [1, 2, 3, 4, 5]$.

comparison of the cost in Subsection 3.2.3 implies that the proposed randomized algorithms are less expensive compared to their deterministic counterparts. To illustrate this, we report the runtime of the HOSVD, R-HOSVD, STHOSVD, and R-STHOSVD algorithms on $\boldsymbol{\mathcal{X}}$ as the size of each dimension increases. For inputs, we fixed the target rank to be $(5, 5, 5, 5, 5)$, the oversampling parameter as $p = 5$, and we used processing order $\rho = [1, 2, 3, 4, 5]$ in the sequential algorithms. The runtime in seconds, averaged over three runs, is shown in Table 3.3. The analysis of the computational cost implies that the randomized algorithms should be a factor of $I/(r + p) = 2.5$ faster than the non-randomized algorithms. This is evident in our results. Also, the sequential algorithms are significantly faster than the HOSVD/R-HOSVD

algorithms.

**Table 3.3** Runtime in seconds of the HOSVD, R-HOSVD, STHOSVD, and R-STHOSVD algorithms on the Hilbert tensor $\boldsymbol{\mathcal{X}}$ with $d = 5$, averaged over three runs. Each algorithm is run with target rank $(5, 5, 5, 5, 5)$, and the randomized algorithms use oversampling parameter $p = 5$. The STHOSVD and R-STHOSVD algorithms use the processing order $\rho = [1, 2, 3, 4, 5]$.

| | | HOSVD | R-HOSVD | STHOSVD | R-STHOSVD |
|---|---|---|---|---|---|
| | 25 | 3.0030 | 1.2609 | 0.6455 | 0.2875 |
| $I_j$ | 35 | 14.5255 | 5.5288 | 3.1974 | 1.2090 |
| $1 \leq j \leq d$ | 45 | 70.0536 | 17.3744 | 15.0629 | 3.5587 |
| | 50 | 101.4981 | 27.7725 | 21.9745 | 5.5606 |

We now compare the numerical performance of the R-HOSVD and R-STHOSVD algorithms to the theoretical bounds derived in (3.3), (3.7). Since the upper bound obtained using both the theorems is the same (see (3.2) and (3.6)), we display this bound only once. We run both algorithms with increasing target rank $(r, r, r, r, r)$, oversampling parameter $p = 5$, and processing order $\rho = [1, 2, 3, 4, 5]$. From the right panel of Figure 3.1, we can see the error in the STHOSVD and R-STHOSVD are both comparable to the theoretical bound, which shows that the analysis captures the behavior of the error quite well.

Next, we present an experiment that justifies the choice of processing order described in Subsection 3.2.3. We consider the Olivetti dataset described in Subsection 3.5.1. In Table 3.4, we compare the relative error and runtime of R-STHOSVD, averaged over three runs, for all six different processing orders $\rho$. To account for the randomness, we set the same initial seed for each run, take the target rank as $(20, 40, 5)$, and the oversampling parameter as $p = 5$. We can see that all processing orders have comparable relative errors, so we instead focus on the runtime. Our heuristic, which involves processing the modes in decreasing size per mode, has the shortest average time as expected from the analysis in Subsection 3.2.3. It is clear from the runtime that processing the largest mode first (mode 2 in this case) had the fastest runtime, and processing the smallest mode first had the slowest runtime.

**Table 3.4** Relative error and average runtime in seconds of R-STHOSVD on the Olivetti dataset for different processing orders. The runtime was averaged over three runs, and the R-STHOSVD used a target rank of $(20, 40, 5)$ and an oversampling parameter of $p = 5$ as inputs.

| Processing Order | $[1, 2, 3]$ | $[1, 3, 2]$ | $[2, 1, 3]$ | $[2, 3, 1]$ | $[3, 1, 2]$ | $[3, 2, 1]$ |
|---|---|---|---|---|---|---|
| Relative Error | 0.1652 | 0.1608 | 0.1669 | 0.1633 | 0.1576 | 0.1602 |
| Average Time (sec) | 0.0774 | 0.0933 | 0.0271 | 0.0246 | 0.1094 | 0.0952 |

Finally, we examine the effect of computing additional subspace iterations in our randomized algorithms. We apply the randomized algorithms to the Olivetti dataset [2]. The results are

described in Figure 3.2. As all the modes have different dimensions, we only compress the largest (the pixels). The first plot shows the standard algorithms, but there is a noticeable difference in the error for the randomized and standard algorithms. This can be explained by the fact that the singular values of the data do not decay sufficiently quickly. To fix this, we add one step of subspace iteration (see [41] for details) to the randomized SVD, giving the second plot. Here the difference is almost nonexistent.

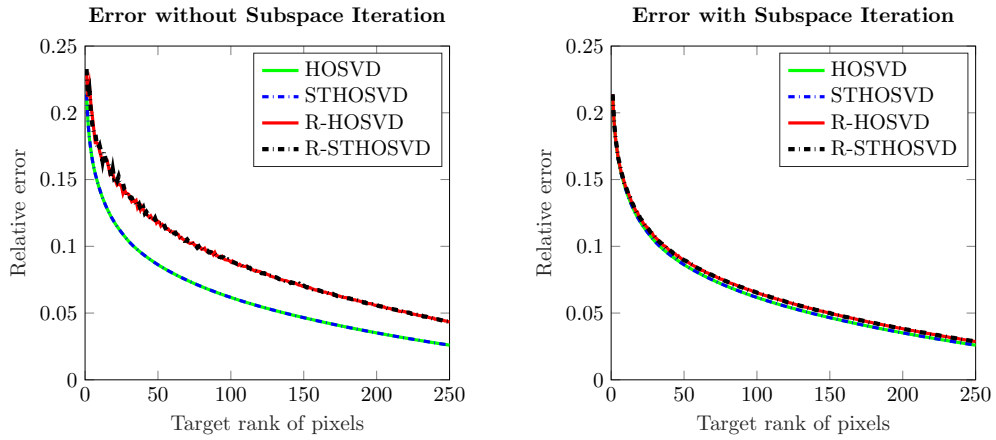


**Figure 3.2** Relative error for $\boldsymbol{\mathcal{X}}$ with and without subspace iteration, compressing just the pixels (mode 2), as the target rank increases. The oversampling parameter was $p = 5$. The right hand plot was run with one step of subspace iteration.

### Adaptive algorithms

We now demonstrate the performance of our adaptive algorithms, Algorithms 7 and 8. Taking the Hilbert tensor $\boldsymbol{\mathcal{X}}$ defined in (3.19) with $d = 3$, we give as input relative error tolerance $\epsilon$, processing order $\rho = [1, 2, 3]$, and blocking integer $b = 1$. The size of the core tensor obtained from the adaptive R-STHOSVD algorithm is shown in Table 3.5. In the $I_j = 25$, $j = 1, 2, 3$ case, we can see that the error and corresponding rank are close to those shown in the left-hand plot of Figure 3.1.

**Table 3.5** Rank (size of the core tensor) obtained by the adaptive R-STHOSVD algorithm (Algorithm 8) with different relative error tolerances $\epsilon$ as the size of each dimension of $\boldsymbol{\mathcal{X}}$ increase. The inputs are $\epsilon$, block size $b = 1$, and processing order $\rho = [1, 2, 3]$.

| | | tolerance $\epsilon$ | | | | |
|---|---|---|---|---|---|---|
| | | $10^{-3}$ | $10^{-4}$ | $10^{-5}$ | $10^{-6}$ | $10^{-7}$ |
| | 25 | $(4, 4, 4)$ | $(5, 5, 5)$ | $(6, 6, 6)$ | $(7, 7, 7)$ | $(8, 8, 8)$ |
| $I$ | 50 | $(5, 5, 5)$ | $(6, 6, 6)$ | $(7, 7, 7)$ | $(8, 8, 8)$ | $(9, 9, 9)$ |
| $1 \leq j \leq d$ | 100 | $(5, 5, 5)$ | $(6, 6, 6)$ | $(8, 8, 8)$ | $(9, 9, 9)$ | $(10, 10, 10)$ |

To show the performance of the adaptive algorithms on the Olivetti dataset, with a processing order $\rho = [2, 1, 3]$, we initialize the adaptive R-STHOSVD algorithm with different desired relative tolerances $\epsilon$, and obtain an approximate decomposition. In Table 3.6, we display the rank of this decomposition $\mathbf{r}$ (the size of the core tensor) as well as the actual relative error of the decomposition. In each instance, we observe that the resulting error of the decomposition is lower than the desired error. Next, with $\mathbf{r}$ as the target rank, we compute a low-rank decomposition using STHOSVD with the same processing order. The resulting error of this decomposition is only slightly smaller than the error in the adaptive algorithm, suggesting that the adaptive algorithm is capable of adaptively estimating the target rank.

**Table 3.6** A comparison of the adaptive R-STHOSVD algorithm (Algorithm 8) to STHOSVD. We first obtained the rank of the core tensor with the requested relative error tolerance from the adaptive algorithm. Then we compared the actual error of the approximation from the adaptive R-STHOSVD to that of an STHOSVD with the same rank. The processing order for all runs was $\rho = [2, 1, 3]$. *Each STHOSVD was computed with the corresponding rank found in the second column.

| Error tolerance $\epsilon$ | Corresponding rank $\mathbf{r}$ | Actual error | Rank-$\mathbf{r}$ STHOSVD error* |
|---|---|---|---|
| 0.25 | $(3, 10, 1)$ | 0.1995 | 0.1995 |
| 0.2 | $(10, 23, 1)$ | 0.1799 | 0.1796 |
| 0.15 | $(22, 51, 5)$ | 0.1421 | 0.1403 |
| 0.1 | $(32, 114, 8)$ | 0.0965 | 0.0946 |
| 0.05 | $(38, 237, 10)$ | 0.0400 | 0.0381 |
| 0.01 | $(40, 381, 10)$ | 0.0057 | 0.0055 |

We also recreate the results shown in Table 3.6, comparing the adaptive R-HOSVD and HOSVD algorithms. On the Olivetti dataset, we initialize the adaptive R-HOSVD algorithm with different error tolerances $\epsilon$ and obtain an approximate decomposition. In Table 3.7, we display the rank of this decomposition $\mathbf{r}$ (the size of the core tensor) as well as the actual relative error of the decomposition. In each instance, we observe that the resulting error of the decomposition is lower than the desired error. Next, with $\mathbf{r}$ as the target rank, we compute a low-rank decomposition using HOSVD. The resulting error of this decomposition is only slightly smaller than the error in the adaptive algorithm, suggesting that the adaptive algorithm is capable of adaptively estimating the target rank.

Next, we consider the relative error obtained by only compressing two modes of the tensor, namely the people and pixels (modes 1 and 2 respectively), formed using the Olivetti dataset. We only present results corresponding to the sequentially truncated algorithms here in the form of "heat" plots which display the relative error as a function of the target rank; see Figure 3.3. In general, we see that the error decreases with increasing rank for both the randomized and deterministic algorithms. We also compare the performance of the adaptive randomized STHOSVD algorithm (Algorithm 8) by displaying the target rank obtained for a given relative error $\epsilon$. For all the fixed rank algorithms discussed above, the oversampling parameter was $p = 5$,

**Table 3.7** A comparison of the adaptive R-HOSVD algorithm (Algorithm 7) to HOSVD. We first obtained the rank of the core tensor with the requested relative error tolerance from the adaptive algorithm. Then we compared the actual error of the approximation from the adaptive R-HOSVD to that of an HOSVD with the same rank. *Each HOSVD was computed with the corresponding rank found in the second column.

| Error tolerance $\epsilon$ | Corresponding rank $\mathbf{r}$ | Actual error | Rank-$\mathbf{r}$ HOSVD error* |
|---|---|---|---|
| 0.25 | $(13, 11, 3)$ | 0.1768 | 0.1709 |
| 0.2 | $(20, 23, 5)$ | 0.1560 | 0.1498 |
| 0.15 | $(27, 52, 7)$ | 0.1276 | 0.1223 |
| 0.1 | $(34, 116, 9)$ | 0.0869 | 0.0811 |
| 0.05 | $(39, 239, 10)$ | 0.0353 | 0.0329 |
| 0.01 | $(40, 382, 10)$ | 0.0057 | 0.0053 |

and the processing order was $\rho = [2, 1, *]$. The third mode is not compressed, indicated here by the asterisk.

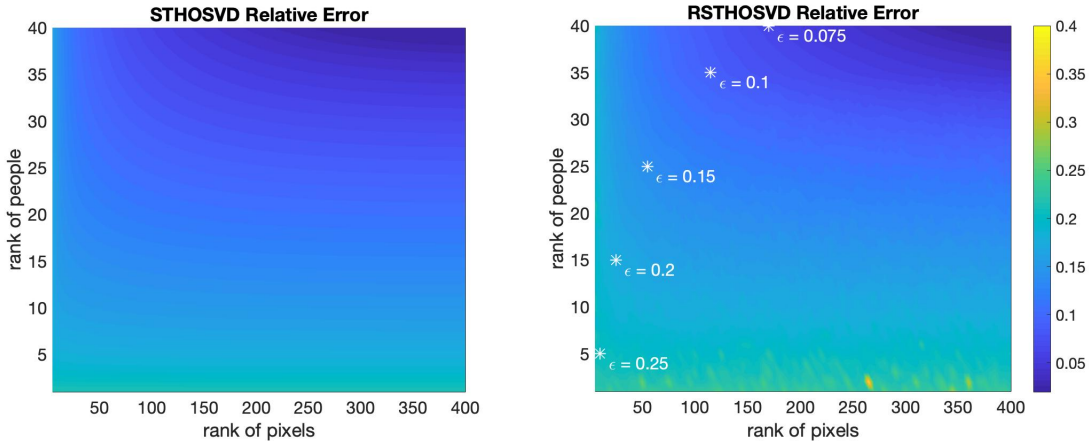


**Figure 3.3** Relative error for $\boldsymbol{\mathcal{X}}$ as target rank increases using the STHOSVD, compressing pixels and people (modes 2 and 1), plotted with rank given by the Adaptive R-STHOSVD (Algorithm 8) with the desired relative error tolerance $\epsilon$. Processing order was $\rho = [2, 1, *]$, and the oversampling parameter was $p = 5$.

### Algorithms for Sparse Tensors

We now test our randomized algorithms on sparse tensors. First, consider the synthetic sparse tensor $\boldsymbol{\mathcal{X}}$ defined in (3.20). For three different $\gamma$ values $\gamma = 2, 10, 200$, we compare the SP-STHOSVD algorithm to the STHOSVD and R-STHOSVD algorithms by plotting the relative error as the target rank $(r, r, r)$ increases. Note that we are only comparing to the sequentially truncated algorithms in Figure 3.4, since they have lower cost and comparable errors. As inputs to our test algorithms, we used oversampling parameter $p = 5$ and processing order $\rho = [1, 2, 3]$.

We can see that the error for the sparse algorithm is only slightly higher for smaller values of $r$, suggesting that the error analysis in Theorem 7 may be pessimistic.

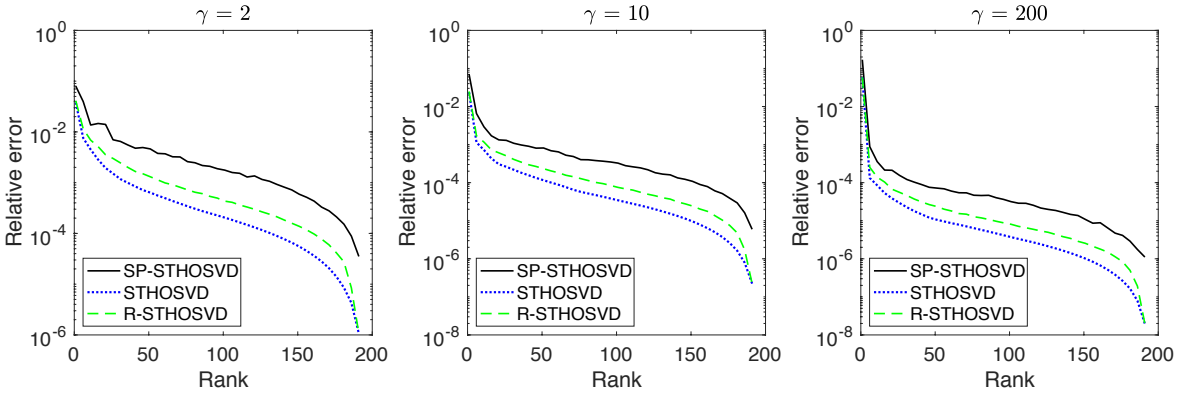


**Figure 3.4** Relative error for synthetic sparse tensor $\boldsymbol{\mathcal{X}}$ defined in (3.20) with $\gamma = 2, 10, 200$ and increasing target rank $(r, r, r)$. We compare the SP-STHOSVD algorithm (Algorithm 9) to the STHOSVD and R-STHOSVD algorithms with inputs of oversampling parameter $p = 5$ and processing order $\rho = [1, 2, 3]$.

Next, we test our SP-STHOSVD algorithm on the real-world sparse tensors. Note that this algorithm does not have a truncation step, which means that the rank of the resulting approximation given target rank $(r, r, r)$ will be $(r+p, r+p, r+p)$. To compare the approximations of SP-STHOSVD with R-STHOSVD, we use a target rank $(r+p, r+p, r+p)$ with an additional oversampling parameter $p = 5$. We ran the SP-STHOSVD and the R-STHOSVD algorithms on the condensed Enron tensor (details in Table 3.2) with processing order $\rho = [3, 1, 2]$ and increasing target rank $(r, r, r)$. The relative errors obtained are shown in Table 3.8. We can see that the error for SP-STHOSVD is higher than that of the R-STHOSVD, as is anticipated from the theory. We also compare the runtime of these two algorithms averaged over three runs to see their respective costs, which are also shown in Table 3.8. We see that the SP-STHOSVD, in addition to preserving the structure, has significantly lower computational costs.

We now show results for the structure preserving algorithms on a different dataset, namely the condensed NELL-2 dataset [15], whose details are shown in Table 3.2. Recall that we are condensing the dataset in order to compare to our other algorithms, specifically R-STHOSVD. We compared both the relative error and average runtime of the algorithms with increasing target rank $(r, r, r)$ and oversampling parameter $p = 5$. The runtime was averaged over three runs, and both algorithms used processing order $\rho = [3, 1, 2]$. We see similar results to those shown in Table 3.8, in that the relative error is slightly higher for SP-STHOSVD than for R-STHOSVD, but the runtime for SP-STHOSVD is significantly less.

We also compared our SP-STHOSVD algorithm to two recent single-pass algorithms from [56, 71]. These experiments were all run on the NCSU Mathematics Department HPC Cluster

**Table 3.8** The relative error and runtime of both SP-STHOSVD and R-STHOSVD on the tensors defined in Table 3.2 as the target rank $(r, r, r)$ increases. The processing order was $\rho = [3, 1, 2]$, and the oversampling parameter was $p = 5$. Note that, for simplicity, the rank is the same for each mode, and that the input rank for the R-STHOSVD was $(r + p, r + p, r + p)$ so the approximations have the same size.

| Enron | | | | |
|---|---|---|---|---|
| | Relative Error | | Runtime in seconds | |
| Target Rank | SP-STHOSVD | R-STHOSVD | SP-STHOSVD | R-STHOSVD |
| 20 | 0.6015 | 0.2081 | 0.4086 | 31.5615 |
| 45 | 0.3854 | 0.1259 | 0.7965 | 34.5802 |
| 70 | 0.3548 | 0.0870 | 1.3276 | 36.6431 |
| 95 | 0.2038 | 0.0632 | 2.3465 | 39.3095 |
| 120 | 0.1503 | 0.0458 | 2.8175 | 39.7169 |
| 145 | 0.0976 | 0.0332 | 3.5659 | 42.0969 |
| 170 | 0.0756 | 0.0239 | 6.2158 | 45.8429 |
| 195 | 0.0578 | 0.0180 | 6.8285 | 50.2907 |

**Table 3.9** The relative error and runtime of both SP-STHOSVD and R-STHOSVD on the condensed NELL-2 dataset as the target rank $(r, r, r)$ increases. The processing order was $\rho = [3, 1, 2]$, and the oversampling parameter was $p = 5$. Note that the rank is the same for each mode for simplicity, and that the input rank for the R-STHOSVD was $(r + p, r + p, r + p)$ so the approximations have the same size.

| NELL-2 | | | | |
|---|---|---|---|---|
| | Relative Error | | Runtime in seconds | |
| Target Rank | SP-STHOSVD | R-STHOSVD | SP-STHOSVD | R-STHOSVD |
| 30 | 0.2968 | 0.1319 | 0.5690 | 17.0642 |
| 60 | 0.2282 | 0.0914 | 0.9606 | 18.9203 |
| 90 | 0.1950 | 0.0699 | 1.5889 | 23.3303 |
| 120 | 0.1666 | 0.0573 | 2.0706 | 28.9399 |
| 150 | 0.1431 | 0.0478 | 2.1310 | 33.6867 |
| 180 | 0.1201 | 0.0417 | 2.3678 | 39.0644 |
| 210 | 0.1181 | 0.0367 | 3.0832 | 45.5227 |
| 240 | 0.1095 | 0.0326 | 3.7282 | 52.4856 |

with 72GB memory. First, we ran all three algorithms on the synthetic sparse tensor defined in (3.20) with $\gamma = 200$. The results are shown in Figure 3.5. Note that the Tucker TensorSketch (Tucker-TS) algorithm from [56] ran out of memory for target ranks higher than $(15, 15, 15)$, so we only show results for smaller ranks. We can see that the relative error is similar for both our SP-STHOSVD algorithm and the One Pass Streaming algorithm from [71], while the runtime is less for the SP-STHOSVD.

We also compared the performance of SP-STHOSVD on the Enron dataset, condensed and full, to the algorithms from [56, 71]. For both the condensed and full datasets, the other two algorithms ran out of memory for all ranks that we tried, so we only have results for SP-STHOSVD. Since we have already presented results on the condensed Enron dataset in

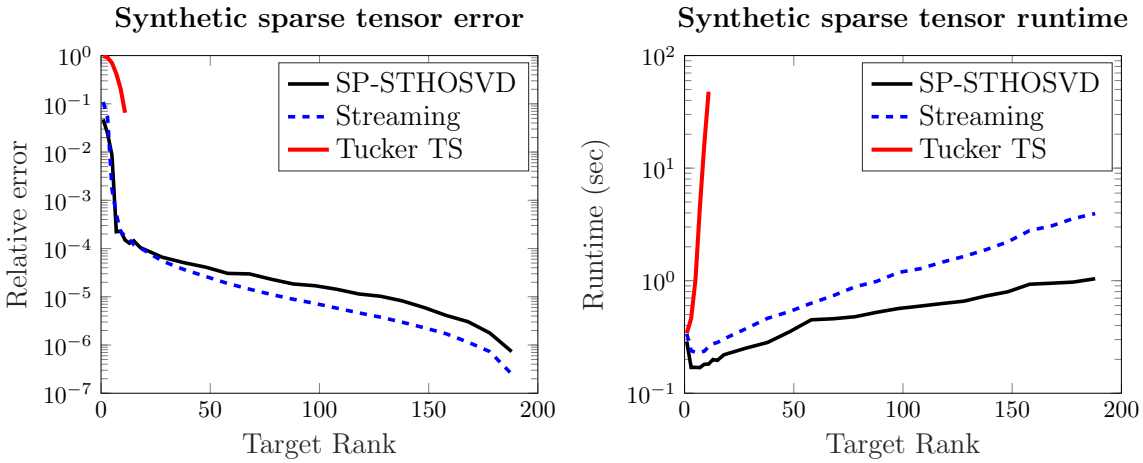


**Figure 3.5** The relative error and runtime of SP-STHOSVD, Tucker-TS [56], and the One Pass ("Streaming") algorithm [71] on the synthetic sparse tensor defined in (3.20) with increasing target rank. The processing order for SP-STHOSVD was $\rho = [1, 2, 3]$, and the oversampling parameter was $p = 5$. We used the suggested parameters for Tucker-TS and the Streaming algorithm, with a target rank of $(r + p, r + p, r + p)$ so all approximations have the same size.

Table 3.8, we just show the runtime for SP-STHOSVD on the full Enron datset in Table 3.10.

**Table 3.10** Runtime in hours of SP-STHOSVD on the full Enron dataset with increasing rank $(r, r, r, r)$. The oversampling parameter was $p = 5$, and the processing order was $\rho = [3, 1, 2, 4]$.

| $r$ | 10 | 20 | 50 | 200 |
|---|---|---|---|---|
| Runtime (hours) | 3.23 | 5.28 | 11.40 | 41.64 |

## 3.6 Conclusion

In this chapter, we developed new randomized algorithms and analysis for low-rank tensor decompositions in the Tucker form. Specifically, we proposed adaptive algorithms for problems where the target rank is not known beforehand and algorithms that preserve the structure of the original tensor. We also provided probabilistic analysis of randomized compression algorithms, R-HOSVD and R-STHOSVD, as well as for the newly proposed algorithms. We showed, through the analysis and numerical examples, that using randomized techniques still allows for accurate approximations to tensors, and that the approximation error is comparable to deterministic algorithms, with much lower computational costs.

## 3.7 Acknowledgements

We would like to thank our coauthor Misha E. Kilmer, as well as Ilse Ipsen for reading through the chapter and giving useful feedback.

# Chapter 4

# Efficient Randomized Algorithms for Subspace System Identification

## 4.1 Introduction

Linear Time Invariant (LTI) systems form one of the most important classes of dynamic systems existing in the physical world. LTI systems closely approximate the linear behavior of any nonlinear physical process around a desired operating point, and provide insights on how the system will behave in response to any small-signal change in its equilibrium state. The state-space representation of a LTI discrete-time system can be written as

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \qquad \mathbf{x}_0 = \mathbf{x}(0) \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k, \end{aligned} \tag{4.1}$$

where $n$ is the order of the system, $k = 0,\ 1,\ 2, \ldots$ is the sampling index, $\mathbf{x}_k \in \mathbb{R}^n$ is the state vector, $\mathbf{u}_k \in \mathbb{R}^m$ is the input vector, and $\mathbf{y}_k \in \mathbb{R}^\ell$ is the output vector at sampling instant $k$, with $m$ and $\ell$ being the number of inputs and outputs, respectively. The matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is referred to as the state matrix, $\mathbf{B} \in \mathbb{R}^{n\times m}$ is the input matrix, $\mathbf{C} \in \mathbb{R}^{\ell\times n}$ is the output matrix, and $\mathbf{D} \in \mathbb{R}^{\ell\times m}$ is the input-output feedthrough matrix. In the majority of practical applications, however, these four matrices may not be exactly known to the system modeler because of different kinds of model and operational uncertainties. In that case, one must estimate these matrices from sampled measurements of the input sequence $\{\mathbf{u}_k\}$ and the output sequence $\{\mathbf{y}_k\}$. This process is referred to as system identification. Since the state may be represented in different coordinate frames without changing input-output characteristics, the more formal definition of system identification is given as the process of estimating $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ up to a similarity transformation $(\mathbf{TAT}^{-1}, \mathbf{TB}, \mathbf{CT}^{-1}, \mathbf{D})$, where $\mathbf{T} \in \mathbb{R}^{n\times n}$ is an invertible matrix. As mentioned, such a transformation does not change the input-output behavior or the transfer function of the system. We assume that the system to be identified is observable and reachable [53]. There exists a vast literature on system identification with a variety of numerical algorithms,

developed for various applications such as electric power systems, process control, mechanical systems, aerospace applications, civil and architectural applications, and chemical and biological processes to name a few. For a survey of these methods, please see [1, 53, 74, 77]. In this chapter, we focus on identifying the system matrices when the inputs $\{\mathbf{u}_k\}$ are in the form of impulse functions.

One of the most common identification methods used for identifying LTI models in practice is known as the Eigensystem Realization Algorithm (ERA), which was initially proposed in [52], but can be found in many other sources such as [77]. ERA is a subset of a broader class of identification methods called subspace system identification, and is a purely data-driven approach consisting of two main steps. In the first step, this algorithm computes the singular value decomposition (SVD) of a block Hankel matrix, constructed using the impulse response data. In the second step, the truncated SVD is used to solve a least-squares optimization problem to identify the system matrices. The first step is computationally expensive since it has cubic complexity in the dimensions of the matrix. To reduce the computational cost, the tangential interpolation-based ERA (TERA) [50] was proposed. This method reduces the number of inputs and outputs, thus reducing the dimensions of the matrix whose SVD needs to be computed. Another algorithm, known as CUR-ERA [49], uses the CUR decomposition to efficiently compute the low-rank decomposition of the block Hankel matrix. While these algorithms have successfully lowered the costs, important challenges remain.

**Contributions and Content.** We propose new randomized algorithms for tackling the computational costs of ERA. The first algorithm (Section 4.3.1) accelerates the standard randomized SVD by exploiting the block Hankel structure to efficiently perform matrix-vector products. The second algorithm (Section 4.3.2) employs similar ideas as the first algorithm in combination with the TERA. The resulting algorithms are efficient both in terms of storage costs and computational costs. In Section 4.4, we derive new error bounds that provide insight into the accuracy and stability of the system matrices identified using the approximation algorithms. The error analysis is not tied to any particular algorithm and in this sense is fairly general. Finally, we demonstrate the benefits of the proposed algorithms on two different applications: first, a heat transfer problem for cooling of steel rails, and second, a dynamic model of an electric power system with multiple generators and loads.

**Comparison to related work.** While TERA works with smaller matrices, it still has cubic complexity in the number of time measurements. This is computationally demanding when the dynamics of the system are slow, and many measurements in time are needed to fully resolve the dynamics. The CUR-ERA algorithm has similar storage and computational complexity as the algorithms we propose but numerical experiments (see Section 4.5) suggest that our algorithm is more accurate. Furthermore, the randomized algorithms developed here can exploit parallelism in multiple ways; the matrix-vector products can be parallelized across multiple random vectors as well as over the input/output pairs. This makes it attractive for implementations on high

performance computing platforms. If the target rank for the low-rank decomposition is not known in advance, one can use randomized range finding algorithms [59, 87] to estimate the target rank. Two other features make our contributions attractive: first, the block Hankel structure that we exploit here can be adapted to any other matrix-free low-rank approximation algorithm, for example, based on the Lanczos bidiagonalization [68], and second, the error analysis developed here is informative for any approximation algorithm. There are other randomized algorithms for system identification [81, 85] but they tackle slightly different settings.

## 4.2 Background

In this section, we first review the Eigensystem Realization Algorithm (Section 4.2.1), the associated computational costs, and motivate the need for efficient algorithms. We also review the key ingredients needed to construct our algorithms: algorithms for storing and computing with Hankel matrices (Section 4.2.2), and randomized SVD (Section 4.2.3) for computing the low-rank factorizations.

### 4.2.1 Eigensystem Realization Algorithm

ERA was first proposed in [52], but we follow the formulation in [50]. Other references for the ERA include [77, 74]. Now, assuming the initial condition $\mathbf{x}_0 = \mathbf{0} \in \mathbb{R}^n$, and given a sequence of inputs $\{\mathbf{u}_i\}$ for $i = 0, 1, \ldots,$ the outputs observed are

$$\mathbf{y}_k = \sum_{j=0}^{k-1} \mathbf{h}_{k-j-1}\mathbf{u}_j, \quad k = 0, 1, \ldots, \tag{4.2}$$

where the matrices $\{\mathbf{h}_k\}$ are called the *Markov parameters* and are given by

$$\mathbf{h}_k = \begin{cases} \mathbf{D} & k = 0 \\ \mathbf{C}\mathbf{A}^{k-1}\mathbf{B} & k = 1, \ldots, 2s-1. \end{cases} \tag{4.3}$$

The system is excited $m$ times using impulse input excitations. That is, we take

$$\mathbf{u}_k^{(j)} = \begin{cases} \mathbf{e}_j & k = 0 \\ \mathbf{0} & k > 0 \end{cases} \qquad j = 1, \ldots, m.$$

Here $\mathbf{e}_j$ is the $j$th column of the $m \times m$ identity matrix. The outputs from each impulse excitation can be used to construct the Markov parameters $\{\mathbf{h}_k\}$. The ERA uses the Markov parameters to recover the system matrices. If data using impulse inputs is not available, then one can estimate the Markov parameters from the general input data [46].

These Markov parameters are first arranged to form the block Hankel matrix $\boldsymbol{\mathcal{H}}_s \in \mathbb{R}^{(s\ell)\times(sm)}$

defined as

$$\begin{aligned}\boldsymbol{\mathcal{H}}_s &= \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \dots & \mathbf{h}_s \\ \mathbf{h}_2 & \mathbf{h}_3 & \dots & \mathbf{h}_{s+1} \\ \vdots & \vdots & \iddots & \vdots \\ \mathbf{h}_s & \mathbf{h}_{s+1} & \dots & \mathbf{h}_{2s-1} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{CB} & \mathbf{CAB} & \dots & \mathbf{CA}^{s-1}\mathbf{B} \\ \mathbf{CAB} & \mathbf{CA}^2\mathbf{B} & \dots & \mathbf{CA}^s\mathbf{B} \\ \vdots & \vdots & \iddots & \vdots \\ \mathbf{CA}^{s-1}\mathbf{B} & \mathbf{CA}^s\mathbf{B} & \dots & \mathbf{CA}^{2s-2}\mathbf{B} \end{bmatrix}.\end{aligned} \tag{4.4}$$

Here, $s$ is a chosen parameter that determines the number of block rows and columns of $\boldsymbol{\mathcal{H}}_s$. It is known that this matrix can be factorized into the observability matrix $\boldsymbol{\mathcal{O}}_s$ and the controllability matrix $\boldsymbol{\mathcal{C}}_s$, as $\boldsymbol{\mathcal{H}}_s = \boldsymbol{\mathcal{O}}_s\boldsymbol{\mathcal{C}}_s$, where

$$\boldsymbol{\mathcal{O}}_s = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^{s-1} \end{bmatrix}, \qquad \boldsymbol{\mathcal{C}}_s = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \dots & \mathbf{A}^{s-1}\mathbf{B} \end{bmatrix}.$$

If the system is assumed to be reachable and observable, then $\mathcal{R}(\boldsymbol{\mathcal{H}}_s) = \mathcal{R}(\boldsymbol{\mathcal{O}}_s)$, where $\mathcal{R}(\cdot)$ denotes the range (or column space) of a matrix. We can obtain a basis for $\boldsymbol{\mathcal{O}}_s$ using the reduced SVD of $\boldsymbol{\mathcal{H}}_s = \mathbf{U}_n\boldsymbol{\Sigma}_n\mathbf{V}_n^\top$. Then, partition the left singular vectors as

$$\mathbf{U}_n = \begin{bmatrix} \boldsymbol{\Upsilon}_f \\ * \end{bmatrix} = \begin{bmatrix} * \\ \boldsymbol{\Upsilon}_l \end{bmatrix},$$

where $\boldsymbol{\Upsilon}_f, \boldsymbol{\Upsilon}_l \in \mathbb{R}^{(s-1)\ell \times n}$ and $*$ denote blocks that do not affect the remaining computations. We can obtain $\mathbf{A}$ using the formula $\mathbf{A} = \boldsymbol{\Upsilon}_f^\dagger \boldsymbol{\Upsilon}_l$, where $^\dagger$ denotes the Moore-Penrose inverse. Furthermore, we can obtain the output matrix $\mathbf{C}$ and the input matrix $\mathbf{B}$ using the formulas

$$\mathbf{C} = \begin{bmatrix} \mathbf{I}_\ell & \mathbf{0} \end{bmatrix} \boldsymbol{\Upsilon}_f, \qquad \mathbf{B} = \boldsymbol{\Sigma}_n\mathbf{V}_n^\top \begin{bmatrix} \mathbf{I}_m \\ \mathbf{0} \end{bmatrix}. \tag{4.5}$$

The matrix $\mathbf{D}$ can be identified as the Markov matrix $\mathbf{h}_0$ and, therefore, we will not discuss its estimation in future sections.

**Reduced order model.** In some applications, the goal is not only to identify the system but also obtain a reduced order model of the system. That is, we seek the reduced system matrices $\mathbf{A}_r \in \mathbb{R}^{r\times r}$, $\mathbf{B}_r \in \mathbb{R}^{r\times m}$, $\mathbf{C}_r \in \mathbb{R}^{\ell\times r}$ and $\mathbf{D}_r \in \mathbb{R}^{\ell\times m}$ which approximate the dynamics of the original system (4.1). To accomplish this, as before, we first compute a rank $r$ approximation to

the matrix $\boldsymbol{\mathcal{H}}_s$ as

$$\boldsymbol{\mathcal{H}}_s \approx \mathbf{U}_r \boldsymbol{\Sigma}_r \mathbf{V}_r^\top,$$

where $r \leq n$. We partition the left singular vectors $\mathbf{U}_r$ as

$$\mathbf{U}_r = \begin{bmatrix} \boldsymbol{\Upsilon}_f^{(r)} \\ * \end{bmatrix} = \begin{bmatrix} * \\ \boldsymbol{\Upsilon}_l^{(r)} \end{bmatrix},$$

such that $\boldsymbol{\Upsilon}_f^{(r)}, \boldsymbol{\Upsilon}_l^{(r)} \in \mathbb{R}^{(s-1)\ell \times r}$. Then, we compute the reduced order model $\mathbf{A}_r = \left[\boldsymbol{\Upsilon}_f^{(r)}\right]^\dagger \boldsymbol{\Upsilon}_l^{(r)}$ such it that minimizes the least squares problem

$$\min_{\widehat{\mathbf{A}} \in \mathbb{R}^{r \times r}} \|\boldsymbol{\Upsilon}_f^{(r)} \widehat{\mathbf{A}} - \boldsymbol{\Upsilon}_l^{(r)}\|.$$

The reduced order output matrix $\mathbf{C}_r$ and the input matrix $\mathbf{B}_r$ are computed using the formulas

$$\mathbf{C}_r = \begin{bmatrix} \mathbf{I}_\ell & \mathbf{0} \end{bmatrix} \mathbf{U}_r, \qquad \mathbf{B}_r = \boldsymbol{\Sigma}_r \mathbf{V}_r^\top \begin{bmatrix} \mathbf{I}_m \\ \mathbf{0} \end{bmatrix}. \tag{4.6}$$

Other papers, such as [49], use a slightly different representation for the system matrix $\mathbf{A}_r$ than the one used in [52]. In this alternate representation, $\mathbf{A}'_r$ is obtained using $\mathbf{A}'_r = \boldsymbol{\Sigma}_r^{-1/2} \boldsymbol{\Upsilon}_f^\dagger \boldsymbol{\Upsilon}_l \boldsymbol{\Sigma}_r^{1/2}$. Then, $\mathbf{B}'_r$ and $\mathbf{C}'_r$ are obtained using

$$\mathbf{C}'_r = \begin{bmatrix} \mathbf{I}_\ell & \mathbf{0} \end{bmatrix} \boldsymbol{\Upsilon}_f \boldsymbol{\Sigma}_r^{1/2}, \quad \mathbf{B}'_r = \boldsymbol{\Sigma}_r^{1/2} \mathbf{V}_r^\top \begin{bmatrix} \mathbf{I}_m \\ \mathbf{0} \end{bmatrix}.$$

Note that the two formulations are equivalent up to a similarity transformation, and this results in the same Markov parameters and input-output behavior of the system. If $r = n$, then the ERA determines the system matrices $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ and there is no model reduction step. On the other hand, if the target rank $r < n$, then the reduced order model is guaranteed stability under the conditions of [49, Theorem 3].

**Computational Cost.** We briefly review the range of possible parameters.

1. State space: In the power system applications, the dimension of the state space $n$ is around $10^2$, whereas in applications with partial differential equations (e.g., the heat transfer application in Section 4.5) this dimension can be large, i.e., $10^3 - 10^4$.

2. Input dimension: The number of inputs $m$ is in the range $1 - 10^2$.

3. Output dimension: The number of outputs $\ell$ is also in the range $1 - 10^2$. In certain applications, the number of outputs is also the dimension of the state space.

4. Sample size: When the dynamics of the system are slow, the number of samples $2s - 1$ can be large, i.e., $10^2 - 10^5$.

The dominant cost of ERA is the cost of storing and factorizing the matrix $\boldsymbol{\mathcal{H}}_s$, which is of size $s\ell \times sm$. The cost of storing the matrix $\boldsymbol{\mathcal{H}}_s$ is $\mathcal{O}(s^2\ell m)$ entries, whereas the cost of computing the SVD is $\mathcal{O}(s^3\ell m \min\{\ell, m\})$ floating point operations (flops). In the applications we consider, forming and factoring $\boldsymbol{\mathcal{H}}_s$ is expensive and is infeasible in some of the large-scale examples. The algorithms we propose are both efficient in storage and computational cost and make ERA applicable to larger problem sizes.

### 4.2.2 Hankel matrices

Structured matrices such as circulant and Hankel matrices are computationally efficient to work with since matrix-vector products (matvecs) can be accelerated using the Fast Fourier Transform (FFT). We first review circulant matrices. Circulant matrices are completely determined by their first column

$$\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_N \end{bmatrix}^\top,$$

and are diagonalized by the Fourier matrix. Let the circulant matrix $\mathbf{X}$ be defined as

$$\mathbf{X} = \begin{bmatrix} x_1 & x_N & \dots & x_2 \\ x_2 & x_1 & \dots & x_3 \\ \vdots & \vdots & \ddots & \vdots \\ x_N & x_{N-1} & \dots & x_1 \end{bmatrix},$$

and let $\mathbf{F}_N$ be the $N \times N$ Fourier matrix with entries $(\mathbf{F}_N)_{jk} = e^{2\pi i(j-1)(k-1)/N}$ where $i = \sqrt{-1}$ and $j, k = 1, \dots, N$. Then the eigenvalue decomposition of $\mathbf{X}$ is $\mathbf{X} = \mathbf{F}_N^* \mathrm{diag}(\mathbf{F}_N \mathbf{x}) \mathbf{F}_N$, where $\mathbf{x} = \mathbf{X}(:, 1)$ is the first column of $\mathbf{X}$. This means that the circulant matrix has eigenvalues $\mathbf{F}_N \mathbf{x}$. This result implies that matvecs $\mathbf{y} = \mathbf{X}\mathbf{v}$ can be performed efficiently using FFTs as $\mathbf{y} = \mathrm{IFFT}(\mathrm{FFT}(\mathbf{v}) \odot \mathrm{FFT}(\mathbf{x}))$, where $\odot$ is the elementwise product and $\mathrm{FFT}(\cdot)$ and $\mathrm{IFFT}(\cdot)$ denote the fast Fourier and inverse fast Fourier transforms respectively. The computational cost involves 2 FFTs and one inverse FFT, and can be implemented efficiently in $\mathcal{O}(N \log_2 N)$ flops.

Hankel matrices have constant entries along every anti-diagonal; that is, given the parameters $h_1, h_2, \dots, h_{2s-1}$, the Hankel matrix is

$$\mathbf{H}_s = \begin{bmatrix} h_1 & h_2 & \dots & h_s \\ h_2 & h_3 & \dots & h_{s+1} \\ \vdots & \vdots & \iddots & \vdots \\ h_s & h_{s+1} & \dots & h_{2s-1} \end{bmatrix}.$$

This corresponds to the single input/single output (SISO) case. Note that permuting a Hankel matrix with the reverse identity permutation matrix $\mathbf{J}_s$ results in a Toeplitz matrix (constant

entries along every diagonal). That is,

$$\mathbf{J}_s = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & 1 & \dots & 0 & 0 \\ 1 & 0 & \dots & 0 & 0 \end{bmatrix}, \qquad \mathbf{H}_s\mathbf{J}_s = \begin{bmatrix} h_s & h_{s-1} & \dots & h_2 & h_1 \\ h_{s+1} & h_s & \dots & h_3 & h_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ h_{2s-2} & \vdots & \dots & h_s & h_{s-1} \\ h_{2s-1} & h_{2s-2} & \dots & h_{s+1} & h_s \end{bmatrix}.$$

We can use this to our advantage while computing matrix-vector products with Hankel matrices. To compute the matrix-vector product $\mathbf{y} = \mathbf{H}_s\mathbf{v}$, we first write $\mathbf{y} = (\mathbf{H}_s\mathbf{J}_s)(\mathbf{J}_s\mathbf{v})$ so that

$$\mathbf{X}_{2s}\begin{bmatrix} \mathbf{J}_s\mathbf{v} \\ \mathbf{0}_s \end{bmatrix} = \begin{bmatrix} \mathbf{H}_s\mathbf{J}_s & \mathbf{B}_s \\ \mathbf{B}_s & \mathbf{H}_s\mathbf{J}_s \end{bmatrix}\begin{bmatrix} \mathbf{J}_s\mathbf{v} \\ \mathbf{0}_s \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ * \end{bmatrix},$$

where $*$ denotes the part of a computation which we can ignore and

$$\mathbf{B}_s = \begin{bmatrix} 0 & h_{2s-1} & \dots & h_{s+2} & h_{s+1} \\ h_1 & 0 & h_{2s-1} & \dots & h_{s+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ h_{s-2} & \vdots & \dots & 0 & h_{2s-1} \\ h_{s-1} & h_{s-2} & \dots & h_1 & 0 \end{bmatrix}.$$

This means that we can also efficiently compute matvecs with Hankel matrices by embedding it within a $2s \times 2s$ circulant matrix $\boldsymbol{X}_{2s}$ defined by the vector

$$\mathbf{x}_{2s} = \begin{bmatrix} h_s & h_{s+1} & \dots & h_{2s-1} & 0 & h_1 & h_2 & \dots & h_{s-1} \end{bmatrix}^\top.$$

Thus, only the first row and the last column need to be stored to compute matrix-vector products with the Hankel matrix $\mathbf{H}_s$. A summary of the algorithm to compute matvecs with $\mathbf{H}_s$ is given in Algorithm 10.

**Algorithm 10** $\mathbf{y} = \text{Hankel-matvec}(\mathbf{h}_c, \mathbf{h}_r, \mathbf{v})$

**Input:** last column $\mathbf{h}_c \in \mathbb{R}^s$, first row $\mathbf{h}_r \in \mathbb{R}^s$ of a Hankel matrix $\mathbf{H}_s$, vector $\mathbf{v} \in \mathbb{R}^s$
**Output:** matvec $\mathbf{y} = \mathbf{H}_s\mathbf{v} \in \mathbb{R}^s$

1: Form circulant vector $\mathbf{x} = \begin{bmatrix} \mathbf{h}_c^\top & 0 & \mathbf{h}_r(1 : \text{end} - 1) \end{bmatrix}^\top$
2: Pad the vector $\mathbf{v}$ to get $\hat{\mathbf{v}} = \begin{bmatrix} \mathbf{v} & \mathbf{0}_s \end{bmatrix}^\top$
3: Take $\mathbf{z} = \text{IFFT}(\text{FFT}(\mathbf{x}) \odot \text{FFT}(\hat{\mathbf{v}}))$.
4: Extract $\mathbf{y} = \mathbf{z}(1 : s)$

### 4.2.3 Randomized SVD

We can efficiently compute a rank $r$ approximation of an $M \times N$ matrix $\mathbf{X}$ using a randomized version of the SVD [41] (henceforth called RandSVD). The idea is to find a matrix $\mathbf{Q}$ whose range approximates that of $\mathbf{X}$. This is done by first drawing a standard Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{N\times(r+p)}$, where $r$ is the desired target rank, and $p \geq 0$ is an oversampling parameter (typically, $p \leq 20$). Then, the matrix $\mathbf{Y} = \mathbf{X\Omega}$ consists of random linear combinations of the columns of $\mathbf{X}$. This means that we can get a matrix $\mathbf{Q}$ such that $\mathcal{R}(\mathbf{X}) \approx \mathcal{R}(\mathbf{Q})$ by taking a thin QR factorization $\mathbf{Y} = \mathbf{QR}$. If $\mathbf{X}$ has singular values that decay rapidly, or $\mathsf{rank}(\mathbf{X})$ is exactly $r$, then $\mathcal{R}(\mathbf{Q})$ is a good approximation for $\mathcal{R}(\mathbf{X})$. We can then approximate $\mathbf{X}$ by the low-rank representation $\mathbf{X} \approx \mathbf{QQ}^\top\mathbf{X}$; this can then be converted into the appropriate SVD format. The procedure is summarized in Algorithm 1.

**Algorithm 11** $[\widehat{\mathbf{U}}, \widehat{\mathbf{\Sigma}}, \widehat{\mathbf{V}}]$=RandSVD$(\mathbf{X}, r, p)$

**Input:** matrix $\mathbf{X} \in \mathbb{R}^{M\times N}$ with target rank $r$, oversampling parameter $p$ such that $r + p \leq \min\{M, N\}$, and number of subspace iterations $q \geq 0$

**Output:** $\widehat{\mathbf{U}} \in \mathbb{R}^{M\times r}$, $\widehat{\mathbf{\Sigma}} \in \mathbb{R}^{r\times r}$, and $\widehat{\mathbf{V}} \in \mathbb{R}^{N\times r}$

1: Draw standard Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{N\times(r+p)}$
2: Multiply $\mathbf{Y} = (\mathbf{XX}^\top)^q\mathbf{X\Omega}$
3: Compute thin QR factorization $\mathbf{Y} = \mathbf{QR}$
4: Form $\mathbf{B} = \mathbf{Q}^\top\mathbf{X}$
5: Calculate thin SVD $\mathbf{B} = \mathbf{U_B\Sigma V}^\top$
6: Set $\widehat{\mathbf{U}} = \mathbf{QU_B}(:, 1:r)$, $\widehat{\mathbf{\Sigma}} = \mathbf{\Sigma}(1:r, 1:r)$, and $\widehat{\mathbf{V}} = \mathbf{V}(:, 1:r)$.

RandSVD is computationally beneficial compared to the full SVD. In this chapter, we use a variation of the RandSVD that uses $q \geq 0$ steps of the subspace iteration [41, Algorithm 4.4]. Note that for numerical stability, we perform orthogonalization during and in between the subspace iterations. Assuming $M \geq N$, the cost of the full SVD is $\mathcal{O}(MN^2)$. On the other hand, if the cost of a matrix-vector product with $\mathbf{X}$ (or its transpose) is $T_{\mathbf{X}}$, then the cost of the randomized SVD can be expressed as

$$\text{Cost} = (2q+1)(r+p)T_{\mathbf{X}} + \mathcal{O}(r^2(M+N)) \text{ flops.} \tag{4.7}$$

We will use RandSVD in different ways in the system identification algorithms that we derive. We have found RandSVD with $q = 0, 1, 2$ subspace iterations to be computationally efficient and sufficiently accurate for our purposes; however, there are several new randomized algorithms developed that have been reviewed in the recent paper [58]. In Section 4.5.1, we compare the performance of Algorithm 11 to other other randomized SVD algorithms.

## 4.3 Randomized algorithms for Eigensystem Realization

In this section, we derive two randomized algorithms for efficient computation of the system matrices. The first algorithm accelerates the standard randomized SVD using the block Hankel structure of $\boldsymbol{\mathcal{H}}_s$ (Section 4.3.1); the second algorithm is a randomized variant of the Tangential Interpolation-based ERA (TERA) and is applicable when the number of inputs and outputs are large.

### 4.3.1 Randomized Eigensystem Realization Algorithm

Our first approach accelerates the computation of the system matrices by combining two ingredients: first, we replace a reduced SVD of $\boldsymbol{\mathcal{H}}_s$ by a RandSVD to obtain an approximate basis for $\boldsymbol{\mathcal{O}}_s$; second, we additionally exploit the block Hankel structure of $\boldsymbol{\mathcal{H}}_s$ to accelerate the matvecs involving $\boldsymbol{\mathcal{H}}_s$ and $\boldsymbol{\mathcal{H}}_s^\top$ in the RandSVD. As we will show, each of these steps decreases the computational complexity yielding an efficient algorithm overall.

#### Block Hankel Matrices

We first explain how we exploit the block Hankel structure of the matrix $\boldsymbol{\mathcal{H}}_s$ defined in (4.4). The multiplication process for Hankel matrices can be extended to block Hankel matrices. Suppose we have to compute $\mathbf{y} = \boldsymbol{\mathcal{H}}_s\mathbf{x}$ for a given vector $\mathbf{x}$. Let us define the index sets

$$\begin{aligned} \mathcal{I}_i &= \{i, i+\ell, \ldots, i+(s-1)\ell\}, \qquad i = 1, \ldots, \ell \\ \mathcal{J}_j &= \{j, j+m, \ldots, j+(s-1)m\}, \qquad j = 1, \ldots, m. \end{aligned}$$

If we denote $\boldsymbol{\mathcal{H}}_s(\mathcal{I}_i, \mathcal{J}_j)$ as the $s \times s$ submatrix obtained by extracting the rows and the columns defined by the appropriate index sets, then it is clear that $\boldsymbol{\mathcal{H}}_s(\mathcal{I}_i, \mathcal{J}_j)$ is a Hankel matrix, and that

$$\mathbf{y}(\mathcal{I}_i) = \sum_{j=1}^{m} \boldsymbol{\mathcal{H}}_s(\mathcal{I}_i, \mathcal{J}_j)\mathbf{x}(\mathcal{J}_j), \qquad i = 1, \ldots, \ell,$$

where $\mathbf{y}(\mathcal{I}_i)$ and $\mathbf{x}(\mathcal{J}_j)$ are the $s \times 1$ vectors obtained from $\mathbf{y}$ and $\mathbf{x}$, respectively.

Algorithm 12 gives the details of the procedure described here in MATLAB-like notation. It is important to note the following points. First, we need not actually form either the full block Hankel matrix or the intermediate Hankel matrices for each block element. Since the Hankel matrices are defined by the first row and the last column, we extract these quantities from the blocks $\{\mathbf{h}_k\}_{k=1}^{2s-1}$ as and when required. Second, since each Hankel matvec costs $\mathcal{O}(s \log_2 s)$ flops, the overall cost of one matvec is $\mathcal{O}(\ell m s \log_2 s)$ flops, compared to $\mathcal{O}(\ell m s^2)$ flops using the naïve approach. Finally, we can easily adapt this algorithm to compute $\boldsymbol{\mathcal{H}}_s^\top\mathbf{x}$ as well; the main difference involves taking as inputs the transpose of the Markov parameters $\{\mathbf{h}_k^\top\}_{j=1}^{2s-1}$ instead.

**Algorithm 12** $\mathbf{y} = \text{Block-Hankel-matvec}(\{\mathbf{h}_k\}_{k=1}^{2s-1}, \mathbf{x}, s)$

**Input:** blocks $\{\mathbf{h}_k\}_{k=1}^{2s-1}$ of Hankel matrix $\boldsymbol{\mathcal{H}} \in \mathbb{R}^{s\ell \times sm}$, vector $\mathbf{x} \in \mathbb{R}^{sm}$, dimension $s \geq 1$
**Output:** matvec $\mathbf{y} = \boldsymbol{\mathcal{H}}_s \mathbf{x}$

1: **for** $i = 1 : \ell$ **do**
2:  **for** $j = 1 : m$ **do**
3:  Extract first row $\mathbf{j}_r$ and last column $\mathbf{j}_c$ as

$$\mathbf{j}_r = \begin{bmatrix} \mathbf{h}_1(i,j) & \mathbf{h}_2(i,j) & \cdots & \mathbf{h}_s(i,j) \end{bmatrix},$$

$$\mathbf{j}_c = \begin{bmatrix} \mathbf{h}_s(i,j) & \mathbf{h}_{s+1}(i,j) & \cdots & \mathbf{h}_{2s-1}(i,j) \end{bmatrix}^\top$$

4:  Compute $\hat{\mathbf{y}} = \text{Hankel-matvec}(\mathbf{j}_c, \mathbf{j}_r, \mathbf{x}(j : m : \text{end}))$
5:  Compute $\mathbf{y}(i : \ell : \text{end}) = \mathbf{y}(i : \ell : \text{end}) + \hat{\mathbf{y}}(1 : s)$
6:  **end for**
7: **end for**

### Randomized ERA

We now incorporate the block Hankel multiplication algorithm, Algorithm 12, with RandSVD in order to accelerate the system identification process. This is simple to do; every time we need to multiply $\boldsymbol{\mathcal{H}}_s$ or $\boldsymbol{\mathcal{H}}_s^\top$, we use block Hankel multiplication instead. This is beneficial in two ways: to reduce the computational cost by two orders of magnitude (one from RandSVD and one from using block Hankel structure), and to reduce storage. We need not store $\boldsymbol{\mathcal{H}}_s$ explicitly or even form the full matrix to begin with. All we need are the blocks that make up $\boldsymbol{\mathcal{H}}_s$. This is a major benefit over the standard algorithms.

First, we use the RandSVD algorithm to compute a low-rank approximation of $\boldsymbol{\mathcal{H}}_s$ with target rank $r \leq n$ to obtain $\widehat{\mathbf{U}}_r \widehat{\boldsymbol{\Sigma}}_r \widehat{\mathbf{V}}_r^\top$. As mentioned earlier, the matvecs involving $\boldsymbol{\mathcal{H}}_s$ or $\boldsymbol{\mathcal{H}}_s^\top$ are handled using Algorithm 12. Then, in the system identification phase, we partition the left singular vectors as

$$\widehat{\mathbf{U}}_r = \begin{bmatrix} \widehat{\boldsymbol{\Upsilon}}_f \\ * \end{bmatrix} = \begin{bmatrix} * \\ \widehat{\boldsymbol{\Upsilon}}_l \end{bmatrix},$$

where $\widehat{\boldsymbol{\Upsilon}}_f$ and $\widehat{\boldsymbol{\Upsilon}}_l$ are both $(s-1)\ell \times r$. The system matrices can then be recovered as $\widehat{\mathbf{A}}_r = \widehat{\boldsymbol{\Upsilon}}_f^\dagger \widehat{\boldsymbol{\Upsilon}}_l$,

$$\widehat{\mathbf{C}}_r = \begin{bmatrix} \mathbf{I}_\ell & \mathbf{0} \end{bmatrix} \widehat{\mathbf{U}}_r, \qquad \widehat{\mathbf{B}}_r = \widehat{\boldsymbol{\Sigma}}_r \widehat{\mathbf{V}}_r^\top \begin{bmatrix} \mathbf{I}_m \\ \mathbf{0} \end{bmatrix}.$$

As before, the matrix $\boldsymbol{D}_r$ is simply the first Markov parameter $\mathbf{h}_0$. We will refer to this randomized ERA that uses block Hankel multiplication as RandSVD-H. Now, we review the computational cost of this algorithm and in Section 4.2.1, we derive error bounds for the recovered system matrices.

**Computational cost.** We now examine the computational cost of the three system identification algorithms described so far, namely the ERA using a full SVD, ERA using RandSVD, and RandSVD-H. The dominant cost of each algorithm is the SVD step, so we focus our attention there. The complexity of the SVD step for these algorithms is shown in Table 4.1. Recall that the size of the matrix we are computing with is $s\ell \times sm$, the target rank for each RandSVD is $r$, and the size of the system is $n$. The cost of the full SVD is then $\mathcal{O}(s^3 m\ell \min\{\ell, m\})$ flops. To analyze the RandSVD based algorithms, we follow the analysis of cost in (4.7). For a standard RandSVD, the cost of a matvec is $\mathcal{O}(s^2\ell m)$ flops. If the matrix $\boldsymbol{\mathcal{H}}_s$ is stored explicitly as we do in our implementation, then the storage cost is $\mathcal{O}(ms^2\ell)$ entries; however, RandSVD can be implemented without storing $\boldsymbol{\mathcal{H}}_s$ explicitly, in which case the cost of storage is $\mathcal{O}(ms\ell)$ entries. When we use block Hankel structure to accelerate the RandSVD, the cost of a matvec is reduced to $\mathcal{O}(\ell ms \log_2 s)$ flops and the storage cost is also $\mathcal{O}(ms\ell)$ entries. This shows that, as anticipated, replacing the SVD with a RandSVD reduces the cost, and then exploiting the block Hankel structure reduces the cost even further. We also include, for comparison purposes, the computational and storage costs of CUR-ERA [49].

**Table 4.1** Computational complexity of the dominant step, the SVD step, in each of the four algorithms. Recall that $s$ is the number of block rows and columns in the block Hankel matrix, $m$ is the number of inputs, $\ell$ is the number of outputs, $r$ is the target rank, and $\kappa$ and $c$ are the number of iterations used in cross approximation and the dominant volume submatrix parts of the CUR-ERA algorithm.

| System ID Algorithm | Computational Cost | Storage cost |
|---|---|---|
| Full SVD | $\mathcal{O}(s^3 m\ell \min\{\ell, m\})$ | $\mathcal{O}(s^2 m\ell)$ |
| RandSVD | $\mathcal{O}(rs^2\ell m + r^2 s(\ell + m))$ | $\mathcal{O}(s^2 m\ell)$ |
| RandSVD-H | $\mathcal{O}(r\ell ms \log_2 s + r^2 s(\ell + m))$ | $\mathcal{O}(sm\ell)$ |
| CUR-ERA | $\mathcal{O}(\kappa r^3 + r^2 s\kappa c(\ell + m))$ | $\mathcal{O}(sm\ell)$ |

**CUR-ERA.** The CUR-ERA algorithm relies on the principle of finding a maximum volume sub-matrix for computing a low-rank approximation, that is, a sub-matrix of specified dimensions with the largest determinant in absolute magnitude. Finding the maximum volume sub-matrix is a combinatorial optimization problem and, hence, one has to settle for heuristics to compute a nearly maximum volume sub-matrix in a reasonable computational time. CUR-ERA uses certain heuristics for finding the cross approximation of a matrix, and finding a dominant volume submatrix. Note that in the computational cost in Table 4.1, $\kappa$ and $c$ are the number of iterations used in the cross approximation and the dominant volume submatrix, respectively. It is clear from the table that our algorithms have a comparable computational cost. Also, the randomized SVD algorithm does not involve a combinatorial optimization problem, is known to be computationally efficient and accurate for a range of problems, and has well-developed error analysis. This makes the RandSVD-H beneficial in practical applications. In addition, as we will

show in numerical experiments in Section 4.5, our algorithms are more accurate.

### 4.3.2 Randomized TERA

This approach is inspired by the tangential interpolation approach for model reduction. The goal of TERA is to reduce the dimension of the Markov parameters by projecting the parameters into a lower dimensional space. This reduces the size of the block Hankel matrix but preserves the block Hankel structure, therefore, reducing the computational cost of the SVD step [50]. We briefly review the TERA approach and describe our acceleration using RandSVD.

**TERA.** In this approach, we seek two orthogonal projection matrices

$$\begin{aligned} \mathbf{P}_1 &= \mathbf{W}_1\mathbf{W}_1^\top, \qquad \mathsf{rank}(\mathbf{W}_1) = \ell' \\ \mathbf{P}_2 &= \mathbf{W}_2\mathbf{W}_2^\top, \qquad \mathsf{rank}(\mathbf{W}_1) = m', \end{aligned}$$

where the matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ have orthonormal columns. To compute $\mathbf{P}_1$, we first arrange the Markov parameters in the matrix

$$\boldsymbol{\mathcal{H}}_w = \begin{bmatrix} \mathbf{h}_1 & \dots & \mathbf{h}_{2s-1} \end{bmatrix} \in \mathbb{R}^{\ell \times m(2s-1)}. \tag{4.8}$$

We then solve the optimization problem

$$\min_{\mathsf{rank}(\mathbf{P})=\ell'} \|\mathbf{P}\boldsymbol{\mathcal{H}}_w - \boldsymbol{\mathcal{H}}_w\|_F^2.$$

The optimal solution can be constructed using the SVD of $\boldsymbol{\mathcal{H}}_w = \mathbf{U}_w\boldsymbol{\Sigma}_w\mathbf{V}_w^\top$. We take $\mathbf{W}_1 = \mathbf{U}_w(:, 1:\ell')$; that is, we take $\mathbf{W}_1$ to be the first $\ell'$ left singular vectors of $\boldsymbol{\mathcal{H}}_w$. Similarly, to construct the matrix $\mathbf{W}_2$, we first arrange the Markov parameters into the matrix

$$\boldsymbol{\mathcal{H}}_e = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{2s-1} \end{bmatrix} \in \mathbb{R}^{\ell(2s-1) \times m}. \tag{4.9}$$

We then solve the optimization problem

$$\min_{\mathsf{rank}(\mathbf{P})=m'} \|\boldsymbol{\mathcal{H}}_e - \boldsymbol{\mathcal{H}}_e\mathbf{P}\|_F^2.$$

The optimal solution can be constructed using the SVD of $\boldsymbol{\mathcal{H}}_e = \mathbf{U}_e\boldsymbol{\Sigma}_e\mathbf{V}_e^\top$. We take $\mathbf{W}_2 = \mathbf{V}_e(:, 1:m')$; that is, we take $\mathbf{W}_2$ to be the first $m'$ right singular vectors of $\boldsymbol{\mathcal{H}}_e$. Having obtained the matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, we construct the projected Markov parameters as

$$\widetilde{\mathbf{h}}_i = \mathbf{W}_1^\top \mathbf{h}_i \mathbf{W}_2 \in \mathbb{R}^{\ell' \times m'}, \qquad i = 1, \dots, 2s-1.$$

The block Hankel matrix $\boldsymbol{\mathcal{H}}_s$ is "projected" using the matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ arranged in diagonal blocks to obtain the reduced-size block Hankel matrix

$$\widetilde{\boldsymbol{\mathcal{H}}}_s = \begin{bmatrix} \boldsymbol{W}_1^\top & & & \\ & \boldsymbol{W}_1^\top & & \\ & & \ddots & \\ & & & \boldsymbol{W}_1^\top \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \dots & \mathbf{h}_s \\ \mathbf{h}_2 & \mathbf{h}_3 & \dots & \mathbf{h}_{s+1} \\ \vdots & \vdots & \iddots & \vdots \\ \mathbf{h}_s & \mathbf{h}_{s+1} & \dots & \mathbf{h}_{2s-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{W}_2 & & & \\ & \boldsymbol{W}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{W}_2 \end{bmatrix}. \tag{4.10}$$

It is important to observe that due to this projection, the block Hankel structure is preserved, and we can express $\widetilde{\boldsymbol{\mathcal{H}}}_s$ in terms of the "projected" Markov parameters as

$$\widetilde{\boldsymbol{\mathcal{H}}}_s = \begin{bmatrix} \widetilde{\boldsymbol{h}}_1 & \widetilde{\boldsymbol{h}}_2 & \dots & \widetilde{\boldsymbol{h}}_s \\ \widetilde{\boldsymbol{h}}_2 & \widetilde{\boldsymbol{h}}_3 & \dots & \widetilde{\boldsymbol{h}}_{s+1} \\ \vdots & \vdots & \iddots & \vdots \\ \widetilde{\boldsymbol{h}}_s & \widetilde{\boldsymbol{h}}_{s+1} & \dots & \widetilde{\boldsymbol{h}}_{2s-1} \end{bmatrix} \in \mathbb{R}^{(\ell' s)\times(m' s)}. \tag{4.11}$$

If $\ell' \ll \ell$ and $m' \ll m$, then the size of the matrix $\widetilde{\boldsymbol{\mathcal{H}}}_s$ is much less than $\boldsymbol{\mathcal{H}}_s$. The dimensions $\ell'$ and $m'$ are determined by the singular value decay of the matrices $\boldsymbol{\mathcal{H}}_w$ and $\boldsymbol{\mathcal{H}}_e$. Retaining a larger number of singular vectors (that is, large $\ell'$ and $m'$) results in a more accurate approximation to $\boldsymbol{\mathcal{H}}_s$ but results in a larger matrix $\widetilde{\boldsymbol{\mathcal{H}}}_s$ and in a higher computational cost.

**Recovering system matrices.** The next steps mimic the standard ERA approach to reconstruct the system matrices but we must carefully account for the dimensions of the projected system. We compute the approximate SVD of $\widetilde{\boldsymbol{\mathcal{H}}}_s = \widetilde{\boldsymbol{U}}_r \widetilde{\boldsymbol{\Sigma}}_r \widetilde{\boldsymbol{V}}_r^\top$, and partition the left singular vectors as

$$\widetilde{\boldsymbol{U}}_r = \begin{bmatrix} \widetilde{\boldsymbol{\Upsilon}}_f \\ * \end{bmatrix} = \begin{bmatrix} * \\ \widetilde{\boldsymbol{\Upsilon}}_l \end{bmatrix}.$$

The system matrices can be recovered as

$$\widetilde{\boldsymbol{A}}_r = \widetilde{\boldsymbol{\Upsilon}}_f^\dagger \widetilde{\boldsymbol{\Upsilon}}_l \in \mathbb{R}^{r\times r}, \qquad \widetilde{\boldsymbol{C}}_r = \begin{bmatrix} \mathbf{W}_1 & \mathbf{0} \end{bmatrix} \widetilde{\boldsymbol{U}}_r, \qquad \widetilde{\boldsymbol{B}}_r = \widetilde{\boldsymbol{\Sigma}}_r \widetilde{\boldsymbol{V}}_r^\top \begin{bmatrix} \mathbf{W}_2^\top \\ \mathbf{0} \end{bmatrix}. \tag{4.12}$$

**Computational cost.** To motivate the need for accelerating TERA using RandSVD, we first review the computational cost which has three components. Computing the projection matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ involves a computational cost of $\mathcal{O}\left(s(\ell m^2 + m\ell^2)\right)$ flops. The SVD of $\widetilde{\boldsymbol{\mathcal{H}}}_s$ now costs

$$\mathcal{O}(s^3 \ell' m' \min\{\ell', m'\}) \text{ flops},$$

and, similar to earlier algorithms, the cost recovery of the system matrices is negligible compared to the cost of the SVD. Although the computational cost is significantly reduced (assuming $\ell' \ll \ell$ and $m' \ll m$), the cost of the SVD still dominates the computational cost and has cubic

scaling with $s$. We now propose a new algorithm to lower this cost.

**Algorithm 13** RandTERA

**Input:** blocks $\{\mathbf{h}_k\}_{k=1}^{2s-1}$ of the block Hankel matrix $\boldsymbol{\mathcal{H}} \in \mathbb{R}^{s\ell \times sm}$, target rank $1 \leq r \leq n$, integers $\ell' \leq \ell$, $m' \leq m$.

1: Form $\boldsymbol{\mathcal{H}}_w$ using (4.8) and compute its SVD $\mathbf{U}_w \boldsymbol{\Sigma}_w \mathbf{V}_w^\top$. Set $\mathbf{W}_1 = \mathbf{U}_w(:, 1:\ell')$
2: Form $\boldsymbol{\mathcal{H}}_e$ using (4.9) and compute its SVD $\mathbf{U}_e \boldsymbol{\Sigma}_e \mathbf{V}_e^\top$. Set $\mathbf{W}_2 = \mathbf{V}_e(:, 1:m')$
3: Form $\widehat{\mathbf{h}}_i = \mathbf{W}_1^\top \mathbf{h}_i \mathbf{W}_2$ for $i = 1, \ldots, 2s-1$.
4: Compute $[\widetilde{\boldsymbol{U}}_r, \widetilde{\boldsymbol{\Sigma}}_r, \widetilde{\boldsymbol{V}}_r]$=RandSVD($\widetilde{\boldsymbol{\mathcal{H}}}_s, r, p$). {Matvecs are computed using Algorithm 12}
5: Compute system matrices using (4.12)

**RandTERA.** It is worth pointing out that the "projected" Hankel matrix $\widetilde{\boldsymbol{\mathcal{H}}}_s$ still retains its block Hankel structure. To reduce the computational cost, we combine the following ingredients that were used previously: we use the RandSVD to compute the rank$-r$ approximation to $\widetilde{\boldsymbol{\mathcal{H}}}_s$, and the matvecs involving $\widetilde{\boldsymbol{\mathcal{H}}}_s$ can be accelerated using the block Hankel structure (as described in Section 4.3.1). The main difference is that we use the "projected" Markov parameters $\{\widetilde{\boldsymbol{h}}_i\}_{i=1}^{2s-1}$ rather than the Markov parameters $\{\mathbf{h}_i\}_{i=1}^{2s-1}$. As a result, the computational cost of the SVD step is reduced to

$$\mathcal{O}\left(r\ell' m' s \log_2 s + r^2 s(m' + \ell')\right) \text{ flops.}$$

We call this algorithm RandTERA, and a complete description of this algorithm is given in Algorithm 13. A breakdown of the computational cost of the RandTERA is given below in Table 4.2.

**Table 4.2** Computational cost of computing RandTERA

| Stage | Computational Cost (flops) |
|---|---|
| Computing $\boldsymbol{W}_1, \boldsymbol{W}_2$ | $\mathcal{O}\left(s(\ell m^2 + m\ell^2)\right)$ |
| SVD step | $\mathcal{O}\left(r\ell' m' s \log_2 s + r^2 s(m' + \ell')\right)$ |

Regarding the storage cost, since only the projected Markov parameters need to be stored, the storage cost is $\mathcal{O}(sm'\ell')$ entries, which is potentially lower than RandSVD-H which requires $\mathcal{O}(sm\ell)$ entries.

## 4.4 Error Analysis

In this section, we analyze the accuracy of RandERA in Section 4.3.1, and derive bounds on the accuracy of the recovered system matrix $\mathbf{A}_r$. Our derivation is not tied to the randomized algorithms used to compute the approximations; this makes the analysis applicable to more general settings.

### 4.4.1 Background and assumptions

We first review the necessary background information and clearly state our assumptions needed for the analysis.

**Canonical Angles.** Given $\boldsymbol{\mathcal{H}}_s = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$, partition $\mathbf{U}_r = \mathbf{U}(:, 1:r)$, the left singular vectors corresponding to the top $r$ singular values of $\boldsymbol{\mathcal{H}}_s$, as

$$\boldsymbol{U}_r = \begin{bmatrix} \boldsymbol{\Upsilon}_f \\ * \end{bmatrix} = \begin{bmatrix} * \\ \boldsymbol{\Upsilon}_l \end{bmatrix} \in \mathbb{R}^{s\ell \times r}.$$

Note that we have dropped the superscripts $(r)$, compared to Section 4.2.1, to make the notation manageable. We can recover the matrix $\boldsymbol{A}_r$ as $\boldsymbol{A}_r = \boldsymbol{\Upsilon}_f^\dagger \boldsymbol{\Upsilon}_l$. Similarly, let $\widehat{\boldsymbol{\mathcal{H}}}_s = \widehat{\mathbf{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\mathbf{V}}^\top$ be an approximation to $\boldsymbol{\mathcal{H}}_s$ with left singular vectors $\widehat{\mathbf{U}}_r$ partitioned as

$$\widehat{\boldsymbol{U}}_r = \begin{bmatrix} \widehat{\boldsymbol{\Upsilon}}_f \\ * \end{bmatrix} = \begin{bmatrix} * \\ \widehat{\boldsymbol{\Upsilon}}_l \end{bmatrix} \in \mathbb{R}^{s\ell \times r},$$

and we can compute the approximate system matrix $\widehat{\mathbf{A}}_r = \widehat{\boldsymbol{\Upsilon}}_f^\dagger \widehat{\boldsymbol{\Upsilon}}_l \in \mathbb{R}^{r \times r}$. Let the canonical angles between the subspaces $\mathcal{R}(\boldsymbol{U}_r)$ and $\mathcal{R}(\widehat{\mathbf{U}}_r)$ be denoted by

$$0 \leq \theta_1 \leq \cdots \leq \theta_{max} \leq \pi/2.$$

If we collect the angles into a diagonal matrix $\boldsymbol{\Theta}$, then $\boldsymbol{U}_r^\top \widehat{\boldsymbol{U}}_r$ has the SVD

$$\boldsymbol{U}_r^\top \widehat{\boldsymbol{U}}_r = \boldsymbol{P}(\cos \boldsymbol{\Theta})\boldsymbol{Q}^\top.$$

Let $\boldsymbol{\mathcal{P}}_{\boldsymbol{U}_r}$ denote the orthogonal projector onto $\mathcal{R}(\boldsymbol{U}_r)$; similarly, let $\boldsymbol{\mathcal{P}}_{\widehat{\boldsymbol{U}}_r}$ denote the orthogonal projector onto $\mathcal{R}(\widehat{\boldsymbol{U}}_r)$. Then, the distance between the two subspace $\mathcal{R}(\boldsymbol{U}_r)$ and $\mathcal{R}(\widehat{\boldsymbol{U}}_r)$ is given by

$$\|\boldsymbol{\mathcal{P}}_{\boldsymbol{U}_r} - \boldsymbol{\mathcal{P}}_{\widehat{\boldsymbol{U}}_r}\|_2 = \|(\boldsymbol{I} - \boldsymbol{\mathcal{P}}_{\boldsymbol{U}_r})\boldsymbol{\mathcal{P}}_{\widehat{\boldsymbol{U}}_r}\|_2 = \|\sin \boldsymbol{\Theta}\|_2 = \sin \theta_{\max}.$$

See [70, Chapter II.4] for more details on canonical angles between subspaces.

**Pseudoinverses.** We recall some facts about the perturbation of pseudoinverses [11, Section 2.2.2]. If $\boldsymbol{M}, \boldsymbol{E} \in \mathbb{R}^{n\times r}$ such that $\mathsf{rank}(\boldsymbol{M}) = \mathsf{rank}(\boldsymbol{M}+\boldsymbol{E}) = r$, then

$$\|(\boldsymbol{M}+\boldsymbol{E})^\dagger - \boldsymbol{M}^\dagger\|_2 \leq \sqrt{2}\|\boldsymbol{M}^\dagger\|_2\|(\boldsymbol{M}+\boldsymbol{E})^\dagger\|_2\|\boldsymbol{E}\|_2 \tag{4.13}$$

Furthermore, if $\|\boldsymbol{E}\|_2\|\boldsymbol{M}^\dagger\|_2 < 1$, then

$$\|(\boldsymbol{M}+\boldsymbol{E})^\dagger\|_2 \leq \frac{\|\boldsymbol{M}^\dagger\|_2}{1-\|\boldsymbol{E}\|_2\|\boldsymbol{M}^\dagger\|_2}. \tag{4.14}$$

If $\boldsymbol{A} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{B} \in \mathbb{R}^{r\times n}$ such that $\mathsf{rank}(\boldsymbol{A}) = \mathsf{rank}(\boldsymbol{B}) = r$, then by [11, Theorem 2.2.3]

$$(\boldsymbol{A}\boldsymbol{B})^\dagger = \boldsymbol{B}^\dagger\boldsymbol{A}^\dagger. \tag{4.15}$$

**Accuracy of eigenvalues.** A norm based approach, i.e., $\|\mathbf{A}-\widehat{\boldsymbol{A}}_r\|$, is not meaningful since $\mathbf{A}_r$ (and $\widehat{\boldsymbol{A}}_r$) can only be determined up to a similarity transformation. To compare the approximate system matrix $\widehat{\boldsymbol{A}}_r$ with $\mathbf{A}_r$, we compare the eigenvalues of these two matrices since the eigenvalues remain unchanged by a similarity transformation. We measure the accuracy of the eigenvalues of $\widehat{\boldsymbol{A}}$ using the spectral variation, which we now define. Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$; the spectral variation between $\boldsymbol{A}$ and $\boldsymbol{B}$ is defined as [9, Section VI.3]

$$\mathsf{sv}\left(\psi(\boldsymbol{B}), \psi(\boldsymbol{A})\right) = \max_{1\leq j\leq n} \min_{1\leq i\leq n} |\lambda_i(\boldsymbol{A}) - \lambda_j(\boldsymbol{B})|, \tag{4.16}$$

where $\psi(\cdot)$ denotes the spectrum of the matrix.

We briefly list the various assumptions that are required for our main result.

**Assumption 1.** *We assume:*

*A1. System matrix: Assume that $\boldsymbol{A}_r = \boldsymbol{\Upsilon}_f^\dagger\boldsymbol{\Upsilon}_l \in \mathbb{R}^{r\times r}$ is diagonalizable and let $\boldsymbol{W} \in \mathbb{R}^{r\times r}$ be the matrix of eigenvectors.*

*A2. Singular vectors: Assume that the subblocks $\boldsymbol{\Upsilon}_f$ of the singular vectors $\boldsymbol{U}_r$ are full rank, that is $\mathsf{rank}(\boldsymbol{\Upsilon}_f) = r$.*

*A3. Markov parameters: Assume the Markov parameters converge to $\boldsymbol{h}_i \to \mathbf{0}$ for $i > s$.*

*A4. Canonical angles: Assume that the canonical angles are sufficiently small and satisfy*

$$\eta \equiv 2\sin\theta_{\max}\|\boldsymbol{\Upsilon}_f^\dagger\|_2 < 1. \tag{4.17}$$

Assumption A1 is rather strong but can be weakened, if different perturbation results are used; see [9, Chapter VIII. 1]. Assumption A2 ensures that the least squares problem $\min_{\boldsymbol{A}} \|\boldsymbol{\Upsilon}_f\boldsymbol{A} - \boldsymbol{\Upsilon}_l\|_F$ has a unique solution. Assumption A3 is also made in [52], and is necessary to ensure the stability of the system. Finally, Assumption A4 ensures that the approximate singular vectors are sufficiently accurate.

### 4.4.2 Main result

We are ready to state our main theorem.

**Theorem 8.** *With the notation in Section 4.4.1 and Assumption 1,*

$$\mathsf{sv}\left(\psi(\widehat{\boldsymbol{A}}_r), \psi(\boldsymbol{A}_r)\right) \leq \kappa_2(\boldsymbol{W})\eta\left(1 + \frac{\sqrt{2}\|\boldsymbol{\Upsilon}_f^\dagger\|_2}{1-\eta}\right),$$

*where* $\kappa_2(\boldsymbol{W}) = \|\boldsymbol{W}\|_2\|\boldsymbol{W}^{-1}\|_2$ *is the condition number of the eigenvectors* $\boldsymbol{W}$ *of* $\mathbf{A}_r$*.*

The theorem identifies several factors that can cause large errors during the identification step. First, the eigenvectors of $\boldsymbol{W}$ may be ill-conditioned, i.e., $\kappa_2(\boldsymbol{W})$ can be large. The best case scenario is when $\boldsymbol{A}$ is normal, so that the condition number is 1. Second, the norm of the pseudoinverse $\|\boldsymbol{\Upsilon}_f^\dagger\|_2 \geq 1$ can be large. However, Assumption A3 ensures that $\lim_{s\to\infty}\|\boldsymbol{\Upsilon}_f\|_2 = 1$; see [49, Lemma 5] for a detailed argument. Finally, if the singular vectors of the low-rank approximation are not computed accurately, then the canonical angles can be large, i.e., $\sin\theta_{\max}$ can be close to 1. This can be mitigated, for example, by taking several subspace iterations. The first three assumptions are intrinsic to the system $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ and only Assumption A4 depends on the choice of the numerical method. The theorem is applicable to the randomized algorithms developed here, namely, RandERA and RandTERA. However, it is important in a larger context, since it is applicable to any approximation algorithm for ERA, including TERA.

*Proof of Theorem 8.* There are several steps involved in this proof.

**1. Introducing a similarity transformation.** Let $\boldsymbol{Z} \in \mathbb{R}^{r\times r}$ be an orthogonal matrix; we will leave the specific choice of this matrix to step 2. Since $\boldsymbol{A}_r$ is diagonalizable, using a perturbation theorem for diagonalizable matrices [9, Theorem VIII.3.1], we have

$$\mathsf{sv}\left(\psi(\widehat{\boldsymbol{A}}_r), \psi(\boldsymbol{A}_r)\right) = \mathsf{sv}\left(\psi(\boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top), \psi(\boldsymbol{A})\right) \leq \kappa_2(\boldsymbol{W})\|\boldsymbol{A}_r - \boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top\|_2. \tag{4.18}$$

The equality in the above equation is because the eigenvalues of $\widehat{\boldsymbol{A}}_r$ and $\boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top$ are the same. We first discuss the choice of $\boldsymbol{Z}$ before analyzing the error in the term $\|\boldsymbol{A}_r - \boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top\|_2$.

**2. Choosing $\boldsymbol{Z}$.** Recall that the SVD of $\boldsymbol{U}_r^\top\widehat{\boldsymbol{U}}_r$ is $\boldsymbol{P}(\cos\boldsymbol{\Theta})\boldsymbol{Q}^\top$. Using the unitary invariance of the 2-norm, $\|\boldsymbol{U}_r - \widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top\|_2 = \|\boldsymbol{U}_r\boldsymbol{Z} - \widehat{\boldsymbol{U}}_r\|_2$. Now, we choose $\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{Q}^\top$ and verify that $\boldsymbol{Z}$ is orthogonal. By the triangle inequality and the unitary invariance of the 2-norm,

$$\begin{aligned}
\|\boldsymbol{U}_r\boldsymbol{Z} - \widehat{\boldsymbol{U}}_r\|_2 &\leq \|\boldsymbol{U}_r\boldsymbol{Z} - \boldsymbol{U}_r\boldsymbol{U}_r^\top\widehat{\boldsymbol{U}}_r\|_2 + \|(\boldsymbol{I} - \boldsymbol{U}_r\boldsymbol{U}_r^\top)\widehat{\boldsymbol{U}}_r\|_2 \\
&\leq \|\boldsymbol{U}_r\boldsymbol{Z} - \boldsymbol{U}_r\boldsymbol{P}(\cos\boldsymbol{\Theta})\boldsymbol{Q}^\top\|_2 + \|(\boldsymbol{I} - \boldsymbol{U}_r\boldsymbol{U}_r^\top)\boldsymbol{\mathcal{P}}_{\widehat{\boldsymbol{U}}_r}\widehat{\boldsymbol{U}}_r\|_2 \\
&\leq \|\boldsymbol{P}\boldsymbol{Q}^\top - \boldsymbol{P}(\cos\boldsymbol{\Theta})\boldsymbol{Q}^\top\|_2 + \|(\boldsymbol{I} - \boldsymbol{\mathcal{P}}_{\boldsymbol{U}_r})\boldsymbol{\mathcal{P}}_{\widehat{\boldsymbol{U}}_r}\|_2\|\widehat{\boldsymbol{U}}_r\|_2 \\
&\leq \|\boldsymbol{I} - \cos\boldsymbol{\Theta}\|_2 + \|\sin\boldsymbol{\Theta}\|_2.
\end{aligned} \tag{4.19}$$

Since the canonical angles satisfy $0 \leq \theta_i \leq \pi/2$, we have $\cos^2\theta_i \leq \cos\theta_i$ and, therefore, for

$i = 1, \dots, r$,

$$(1 - \cos\theta_i)^2 = 1 - 2\cos\theta_i + \cos^2\theta_i \le 1 - \cos^2\theta_i = \sin^2\theta_i.$$

Therefore, $\|\boldsymbol{I} - \cos\boldsymbol{\Theta}\|_2 \le \|\sin\boldsymbol{\Theta}\|_2$. Together with (4.19), we have

$$\|\boldsymbol{U}_r - \widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top\|_2 = \|\boldsymbol{U}_r\boldsymbol{Z} - \widehat{\boldsymbol{U}}_r\|_2 \le 2\|\sin\boldsymbol{\Theta}\|_2. \tag{4.20}$$

**3. Ensuring $\mathsf{rank}(\widehat{\boldsymbol{\Upsilon}}_f) = r$.** Choose $\boldsymbol{M} = \boldsymbol{\Upsilon}_f$, which has rank $r$, and $\boldsymbol{M} + \boldsymbol{E} = \widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top$. We now show that $\boldsymbol{M} + \boldsymbol{E}$ also has rank $r$. Using Weyl's theorem on perturbation of singular values [11, Theorem 2.2.8],

$$|\sigma_r(\widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top) - \sigma_r(\boldsymbol{\Upsilon}_f)| \le \|\boldsymbol{E}\|_2 = \|\boldsymbol{\Upsilon}_f - \widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top\|_2 \le \|\boldsymbol{U}_r - \widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top\|_2 \le 2\|\sin\boldsymbol{\Theta}\|_2.$$

We have used the fact that $\boldsymbol{\Upsilon}_f$ and $\widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top$ are submatrices of $\boldsymbol{U}_r$ and $\widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top$, respectively. Since $\sigma_r(\boldsymbol{\Upsilon}_f) = 1/\|\boldsymbol{\Upsilon}_f^\dagger\|_2$, we can rearrange to get

$$\sigma_r(\widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top) \ge \frac{1}{\|\boldsymbol{\Upsilon}_f^\dagger\|_2} - 2\|\sin\boldsymbol{\Theta}\|_2 > 0,$$

since by assumption, $\eta = 2\|\sin\boldsymbol{\Theta}\|_2\|\boldsymbol{\Upsilon}_f^\dagger\|_2 < 1$. This shows that both $\widehat{\boldsymbol{\Upsilon}}_f$ and $\widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top$ have rank $r$. Therefore, by (4.15), $(\widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top)^\dagger = \boldsymbol{Z}\widehat{\boldsymbol{\Upsilon}}_f^\dagger$. Furthermore, (4.14) applies and

$$\|\boldsymbol{Z}\widehat{\boldsymbol{\Upsilon}}_f^\dagger\|_2 \le \frac{\|\boldsymbol{\Upsilon}_f^\dagger\|_2}{1 - 2\|\sin\boldsymbol{\Theta}\|_2\|\boldsymbol{\Upsilon}_f^\dagger\|_2}. \tag{4.21}$$

**4. Error in $\|\boldsymbol{A}_r - \boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top\|_2$.** Write both matrices in terms of their factors, and add and subtract the term $\boldsymbol{\Upsilon}_f^\dagger\widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top$ to get

$$\begin{aligned}\|\boldsymbol{A}_r - \boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top\|_2 &= \|\boldsymbol{\Upsilon}_f^\dagger\boldsymbol{\Upsilon}_l - \boldsymbol{\Upsilon}_f^\dagger\widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top + \boldsymbol{\Upsilon}_f^\dagger\widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top - \boldsymbol{Z}\widehat{\boldsymbol{\Upsilon}}_f^\dagger\widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top\|_2 \\ &\le \|\boldsymbol{\Upsilon}_f^\dagger\|_2\|\boldsymbol{\Upsilon}_l - \widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top\|_2 + \|\boldsymbol{\Upsilon}^\dagger - \boldsymbol{Z}\widehat{\boldsymbol{\Upsilon}}_f^\dagger\|_2\|\widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top\|_2.\end{aligned} \tag{4.22}$$

Note that, by Assumption A2, $\mathsf{rank}(\boldsymbol{\Upsilon}_f) = r$. In step 3, we showed that $\mathsf{rank}(\widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top) = r$. As before, with $\boldsymbol{M} = \boldsymbol{\Upsilon}_f$ and $\boldsymbol{M} + \boldsymbol{E} = \widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top$, (4.13) applies and

$$\|\boldsymbol{\Upsilon}_f^\dagger - \boldsymbol{Z}\widehat{\boldsymbol{\Upsilon}}_f^\dagger\|_2 \le \sqrt{2}\|\boldsymbol{\Upsilon}_f^\dagger\|_2\|\boldsymbol{Z}\widehat{\boldsymbol{\Upsilon}}_f^\dagger\|_2\|\boldsymbol{\Upsilon}_f - \widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top\|_2.$$

Plugging this into (4.22), we have

$$\|\boldsymbol{A}_r - \boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top\|_2 \le \|\boldsymbol{\Upsilon}_f^\dagger\|_2\left(\|\boldsymbol{\Upsilon}_l - \widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top\|_2 + \sqrt{2}\|\boldsymbol{Z}\widehat{\boldsymbol{\Upsilon}}_f^\dagger\|_2\|\boldsymbol{\Upsilon}_f - \widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top\|_2\right).$$

We have used the fact that $\|\widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top\|_2 \le \|\widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top\|_2 = 1$; the inequality is because $\widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top$ is a

submatrix of $\widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top$ and the equality follows since $\widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top$ has orthonormal columns. Similarly,

$$\|\boldsymbol{\Upsilon}_l - \widehat{\boldsymbol{\Upsilon}}_l\boldsymbol{Z}^\top\|_2 \le \|\boldsymbol{U}_r - \widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top\|_2, \qquad \|\boldsymbol{\Upsilon}_f - \widehat{\boldsymbol{\Upsilon}}_f\boldsymbol{Z}^\top\|_2 \le \|\boldsymbol{U}_r - \widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top\|_2,$$

so that

$$\|\boldsymbol{A}_r - \boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top\|_2 \le \|\boldsymbol{\Upsilon}_f^\dagger\|_2\|\boldsymbol{U}_r - \widehat{\boldsymbol{U}}_r\boldsymbol{Z}^\top\|_2\left(1 + \sqrt{2}\|\boldsymbol{Z}\widehat{\boldsymbol{\Upsilon}}_f^\dagger\|_2\|\right). \tag{4.23}$$

**5. Finishing the proof.** Plug (4.20) and (4.21) into (4.23) to obtain

$$\|\boldsymbol{A}_r - \boldsymbol{Z}\widehat{\boldsymbol{A}}_r\boldsymbol{Z}^\top\|_2 \le (2\|\sin\boldsymbol{\Theta}\|_2\|\boldsymbol{\Upsilon}_f^\dagger\|_2)\left(1 + \frac{\sqrt{2}\|\boldsymbol{\Upsilon}_f^\dagger\|_2}{1 - 2\|\sin\boldsymbol{\Theta}\|_2\|\boldsymbol{\Upsilon}_f^\dagger\|_2}\right).$$

Plug this bound into (4.18) to finish the proof. □

An important aspect of the proof of Theorem 8 is the choice of the orthogonal matrix $\boldsymbol{Z}$ that determines the similarity transformation. Our proof used this idea from [44, Theorem 5]. It is worth pointing out that the matrix $\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{Q}^\top$ is also the solution to the orthogonal Procrustes problem

$$\min_{\substack{\boldsymbol{Z}\in\mathbb{R}^{r\times r}\\ \boldsymbol{Z}^\top\boldsymbol{Z}=\boldsymbol{I}_r}} \|\boldsymbol{U}_r\boldsymbol{Z} - \widehat{\boldsymbol{U}}_r\|_F.$$

Therefore, the matrix $\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{Q}^\top$ is the matrix that "best rotates" the columns $\boldsymbol{U}_r$ to align with the columns of $\widehat{\boldsymbol{U}}_r$. Note, however, that in the proof we use the 2-norm instead of the Frobenius norm.

### 4.4.3 Accuracy of the singular vectors

Theorem 8 shows that the accuracy of the approximation algorithms to the ERA depend on the canonical angles between the exact and approximate subspaces $\mathcal{R}(\boldsymbol{U}_r)$ and $\mathcal{R}(\widehat{\boldsymbol{U}}_r)$. Insight into the accuracy between these subspaces can be obtained by standard perturbation theory. Suppose there are numbers $\alpha \ge 0$ and $\delta > 0$ such that

$$\sigma_{r+1}(\widehat{\boldsymbol{\mathcal{H}}}_s) \ge \alpha + \delta, \qquad \sigma_r(\mathbf{H}_s) \le \alpha.$$

Let $\widehat{\boldsymbol{U}}_r, \widehat{\boldsymbol{V}}_r$ be the matrices containing the left and right singular vectors corresponding to the first $r$ singular values of $\widehat{\boldsymbol{\mathcal{H}}}_s$, which are taken to be the diagonals of the matrix $\widehat{\boldsymbol{\Sigma}}_r$. Then, by [70, Chapter V.4, Theorem 4.4],

$$\|\sin\boldsymbol{\Theta}\|_2 \le \frac{\max\{\|\boldsymbol{\mathcal{H}}_s\widehat{\boldsymbol{V}}_r - \widehat{\boldsymbol{U}}_r\widehat{\boldsymbol{\Sigma}}_r\|_2, \|\boldsymbol{\mathcal{H}}_s^\top\widehat{\boldsymbol{U}}_r - \widehat{\boldsymbol{V}}_r\widehat{\boldsymbol{\Sigma}}_r\|_2\}}{\delta}.$$

In Section 4.3.1, the randomized subspace iteration is used to construct the low-rank approximation $\widehat{\boldsymbol{\mathcal{H}}}_s$. We can use more special purpose error bounds to quantify the accuracy of the singular vectors. Specifically, one may use techniques developed in Section 3.2 of [65]. We omit a detailed

statement of the results.

### 4.4.4 Stability

The discrete dynamical system with the system matrix $\boldsymbol{A}_r$ is stable if the eigenvalues of $\boldsymbol{A}_r$ lie inside the unit circle; that is, the spectral radius $\rho(\boldsymbol{A}_r) < 1$. Assumption A3 in Section 4.4.1 means that the Markov parameters decay after some finite time. Then, by [49, Theorem 3], there exists a positive integer $s$ such that the identified reduced order model is a stable discrete dynamical system. Theorem 8 can also be used to determine when the approximate discrete dynamical system with the system matrix $\widehat{\boldsymbol{A}}_r$ is stable.

We consider the spectral radius of $\widehat{\boldsymbol{A}}_r$, and write

$$\rho(\widehat{\boldsymbol{A}}_r) = \max_{1 \le j \le r} |\lambda_j(\widehat{\boldsymbol{A}}_r)| = |\lambda_{j^*}(\widehat{\boldsymbol{A}}_r)|,$$

for some index $j^*$. Similarly, let $i^*$ be the index such that $\lambda_{i^*}(\boldsymbol{A}_r)$ is the closest eigenvalue to $\lambda_{j^*}(\widehat{\boldsymbol{A}}_r)$. If $i^*$ and $j^*$ are not unique, break the tie in some fashion. Therefore, we have the series of inequalities

$$\rho(\widehat{\boldsymbol{A}}_r) \le |\lambda_{j^*}(\widehat{\boldsymbol{A}}_r) - \lambda_{i^*}(\boldsymbol{A}_r)| + |\lambda_{i^*}(\boldsymbol{A}_r)| \le \mathsf{sv}\left(\Psi(\widehat{\boldsymbol{A}}_r), \Psi(\boldsymbol{A}_r)\right) + \rho(\boldsymbol{A}_r).$$

By Theorem 8, the approximate discrete dynamical system is stable if

$$\kappa_2(\boldsymbol{W})\eta\left(1 + \frac{\sqrt{2}\|\boldsymbol{\Upsilon}_f^\dagger\|_2}{1-\eta}\right) < 1 - \rho(\boldsymbol{A}_r).$$

By assumption, $\eta < 1$; the above equation gives an additional condition on $\eta$ for stability. This condition can be informative for the numerical method used to compute the singular vectors $\widehat{\boldsymbol{U}}_r$.

## 4.5 Numerical Results

To test our algorithms, we consider two test problems which pose different challenges. The first test problem is generated from a LTI model of a controlled heat transfer process for optimal cooling of a steel profile [63]. The second test problem is generated from a LTI state-space model of an electrical power generation system with 50 generators [8]. The dimensions of the variables related to both test problems are shown in Table 4.3. The heat transfer test problem has a relatively large system size $n$ but a small number of inputs and outputs, while the power system test problem has a relatively small system size but a much larger number of inputs and outputs. All our experiments were run in MATLAB, and our timing experiments were run on the NCSU HPC Cluster with 72GB memory. MATLAB Code for the algorithms, test models, and some of the numerical experiments included in this section can be found at https://github.com/rlminste/RandSysID.

**Table 4.3** Details of the test problems we will use in our numerical experiments, as well as the size of the largest Hankel matrix $\boldsymbol{\mathcal{H}}_s$ that occurs for these test problems.

| System | System Size $n$ | Inputs $m$ | Outputs $\ell$ | Largest $\boldsymbol{\mathcal{H}}_s$ dim. |
|---|---|---|---|---|
| Heat Transfer | 1357 | 7 | 6 | $24,000 \times 28,000$ |
| Power System | 155 | 50 | 155 | $155,000 \times 50,000$ |

### 4.5.1 Heat Transfer

We first test our algorithms on the heat transfer test problem, which has a large system size but a relatively small number of inputs and outputs. This system also has slow dynamics, so a large value of $s$ is needed to capture the transient behavior. The model for this test problem is not in standard form, so we will need to convert it to standard form before we are able to use our algorithms. The original continuous form after spatial discretization is

$$\begin{aligned} \mathbf{E}\frac{d\mathbf{x}}{dt} &= \mathbf{A}_o\mathbf{x} + \mathbf{B}_o\mathbf{u} \\ \mathbf{y} &= \mathbf{C}_o\mathbf{x} + \mathbf{D}_o\mathbf{u}. \end{aligned}$$

To convert it to standard form, we use the Cholesky factorization of $\mathbf{E} = \mathbf{L}\mathbf{L}^\top$. Then, the continuous standard form system matrices are found by computing $\mathbf{A}_c = \mathbf{L}^{-1}\mathbf{A}_o\mathbf{L}^{-\top}$, $\mathbf{B}_c = \mathbf{L}^{-1}\mathbf{B}_o$, and $\mathbf{C}_c = \mathbf{C}_o\mathbf{L}^{-\top}$. Note that $\mathbf{D}_c = \mathbf{D}_o$ remains unchanged. The continuous-time matrices are converted into appropriate discrete-time matrices using MATLAB's `c2d` command. It is worth noting that the continuous-time matrices are only used in the construction of the Markov parameters but are not used explicitly in the system identification algorithms.

For this heat transfer problem, we will compare accuracy and speed of our algorithms that take advantage of the block Hankel structure of $\boldsymbol{\mathcal{H}}_s$, i.e., RandSVD-H (see Section 4.3.1), to the standard ERA that uses a full SVD. We also contrast these two algorithms with another algorithm we call SVDS-H, which uses the MATLAB command `svds`, but we instead use the block Hankel multiplication described in Algorithm 12.

#### Accuracy

We consider several numerical experiments to test the accuracy of our algorithms. For each experiment, we used $s = 1000$ and a target rank of $r = 20$; that is, we are performing model reduction in addition to the system identification. For RandSVD-H, we used $q = 1$ for the number of subspace iterations and $p = 20$ for the oversampling parameter. In our first experiment, we compare the first 20 singular values of the block Hankel matrix $\boldsymbol{\mathcal{H}}_s \in \mathbb{R}^{6000\times 7000}$ as computed by a full SVD to the first 20 singular values computed by the SVDS-H and RandSVD-H algorithms. The singular values are plotted in Figure 4.1; we see that they are in good agreement with each other.

Next we compare the eigenvalues of the identified $\widehat{\mathbf{A}}_r$ matrices using the SVDS-H and

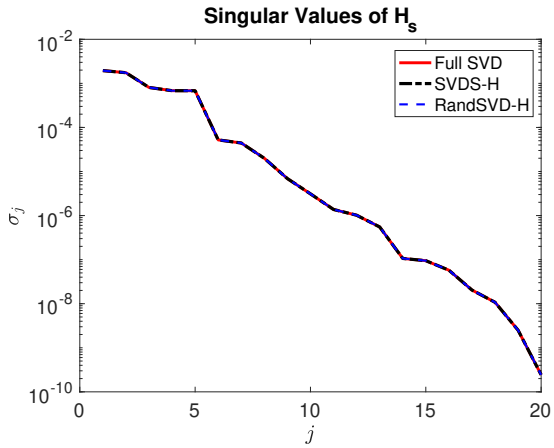


**Figure 4.1** Singular values of the block Hankel matrix $\boldsymbol{\mathcal{H}}_s$ as computed by a full SVD, SVDS-H, and RandSVD-H. We computed the first 20 singular values using SVDS-H and RandSVD-H and plotted them against the first 20 singular values computed by the full SVD.

RandSVD-H algorithms to the eigenvalues of $\mathbf{A}_r$ identified using the full SVD ERA algorithm. The results are shown in Figure 4.2, where we can see that the eigenvalues of $\widehat{\mathbf{A}}_r$ computed using SVDS-H and RandSVD-H align well with those of $\mathbf{A}_r$ computed using the full SVD accurately.

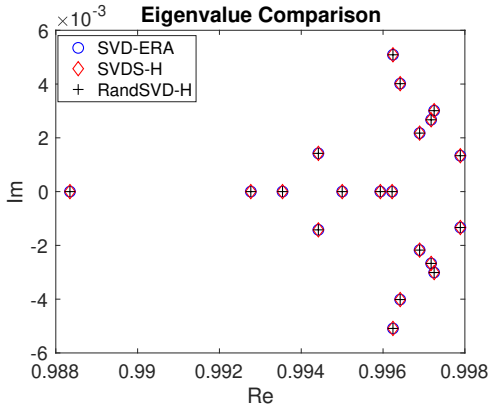


**Figure 4.2** Eigenvalues of the identified $\mathbf{A}_r$ matrix using SVD-ERA, SVDS-H, and RandSVD-H algorithms on the heat transfer problem. We used $s = 1000$ and a target system size of $r = 20$ as inputs for each algorithm.

In order to quantitatively evaluate the accuracy of the identified eigenvalues, we consider the Hausdorff distance metric, defined between the spectra of the two matrices $\mathbf{A}_r$ and $\widehat{\mathbf{A}}_r$,

$$h_d(\psi(\mathbf{A}_r), \psi(\widehat{\mathbf{A}}_r)) = \max\left\{\mathsf{sv}\left(\psi(\mathbf{A}_r), \psi(\widehat{\mathbf{A}}_r)\right), \mathsf{sv}\left(\psi(\widehat{\mathbf{A}}_r), \psi(\mathbf{A}_r)\right)\right\}. \tag{4.24}$$

Note that the Hausdorff distance is related to the spectral variation, defined in (4.16), as it measures how close two sets are to each other. We use Hausdorff distance $h_d$ instead of spectral

variation since the spectral variation of two sets depends on the order of the sets in question, while the Hausdorff distance does not. For both algorithms SVDS-H and RandSVD-H, we computed the Hausdorff distance between the spectra of $\widehat{\mathbf{A}}_r$ and $\mathbf{A}_r$ and we saw that for both algorithms $h_d \approx 10^{-14}$.

The final measure of accuracy we will consider for this test problem is the relative error in the Markov parameters. Let $\mathbf{A}_r, \mathbf{B}_r, \mathbf{C}_r$ be the system matrices identified by the full SVD-ERA, and let $\widehat{\mathbf{A}}_r, \widehat{\mathbf{B}}_r, \widehat{\mathbf{C}}_r$ be the system matrices identified using another method. The relative error in the Markov parameters is then defined as

$$M_k = \frac{\|\mathbf{C}_r \mathbf{A}_r^k \mathbf{B}_r - \widehat{\mathbf{C}}_r \widehat{\mathbf{A}}_r^k \widehat{\mathbf{B}}_r\|_2}{\|\mathbf{C}_r \mathbf{A}_r^k \mathbf{B}_r\|_2}, \quad k = 1, \ldots, 2s-1. \tag{4.25}$$

We compute this error for the identified matrices using the SVDS-H and RandSVD-H algorithms, and plot the results in Figure 4.3. Both algorithms produce comparable error in the Markov parameters, and this error is very low in both cases. These results combined with those from Figure 4.2 show that our algorithms are very accurate compared to the standard algorithms.

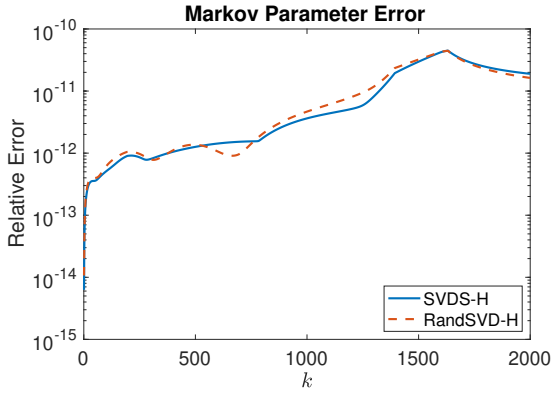


**Figure 4.3** Relative error in recreated Markov parameters compared to a full SVD ERA for both the SVDS-H and RandSVD-H algorithms. We used $s = 1000$ and a target rank of $r = 20$ as inputs.

### Timing

We now consider the average run time of the three algorithms on the heat transfer problem. The results, shown in Figure 4.4, clearly indicate that using the block Hankel structure as in SVDS-H and RandSVD-H significantly reduces the computational cost, and using a randomized algorithm reduces that cost further, as RandSVD-H is computationally the least expensive of the three. We also see from the plots that the SVDS and RandSVD-H algorithms, which both exploit the block Hankel structure, show approximately linear scaling with $s$ confirming the analysis of the computational costs.

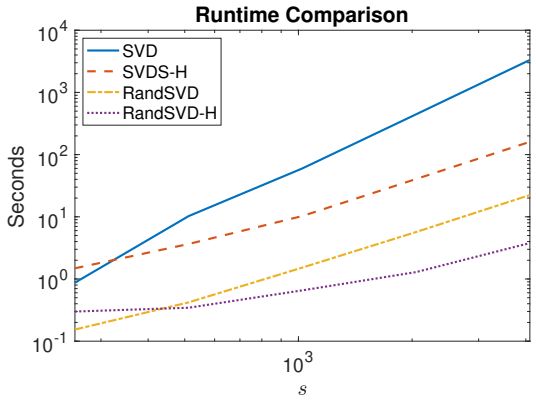


**Figure 4.4** Average run time of the SVD, SVDS, RandSVD, and RandSVD-H algorithms on the heat transfer problem. We averaged the run time over three runs, and a target system size of $r = 20$ as inputs for each algorithm.

### Comparison to CUR-ERA

The final test we consider with this application is how CUR-ERA compares to the randomized algorithms we analyzed in the previous experiments. For CUR-ERA, we used the default parameters suggested in the code provided by the authors of [49]. The maxvol tolerance is $10^{-4}$, and the iterations stop when the relative error of the subsequent iterations is less than $10^{-4}$. For our experiments, we start with the accuracy of the eigenvalues, plotted in Figure 4.5. In

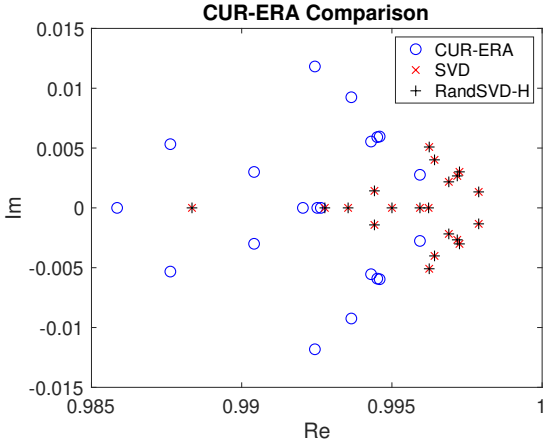


**Figure 4.5** Eigenvalues of the identified $\mathbf{A}_r$ matrix using CUR-ERA, compared to those identified by SVD and RandSVD-H on the heat transfer problem with $s = 1000$ and a target system size of $r = 20$.

this figure, we see that the eigenvalues of $\mathbf{A}_r$ identified by CUR-ERA are not as accurate as those identified by RandSVD-H. To quantify this, we compute the Hausdorff distance between the spectrum of the $\mathbf{A}_r$ matrix identified by CUR-ERA and the spectrum of the $\widehat{\mathbf{A}}_r$ matrix

identified by SVD-ERA when $s = 1000$. We find that this distance is approximately $4.0 \times 10^{-4}$, supporting the claim that CUR-ERA is less accurate than RandSVD-H (we experimented with several different choices of CUR-ERA parameters and are reporting the choice which gave the smallest error). At the same time, the CUR-ERA algorithm took 0.83 seconds compared to 0.66 seconds for RandSVD-H (averaged over three runs). Finally, we show the relative error in the Markov parameters, computed as defined in (4.25), for CUR-ERA, RandSVD-H, and SVDS-H. The results are shown in Figure 4.6, and we see that the error shown here confirms the accuracy comparison, thus far. The accuracy of CUR-ERA is lower than that obtained from either RandSVD-H or SVDS-H. In all cases, CUR-ERA took two iterations to converge.

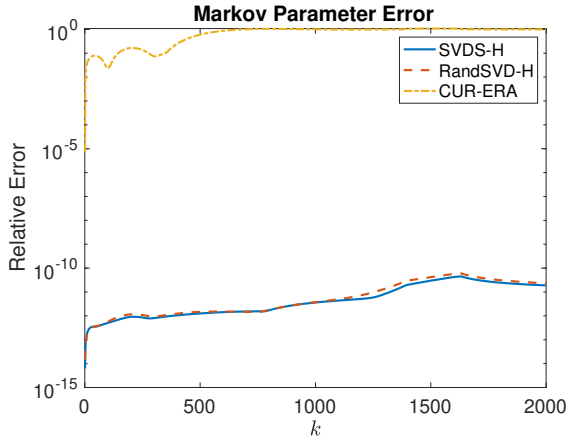


**Figure 4.6** Relative error in recreated Markov parameters compared to a full SVD-ERA for RandSVD-H, SVDS-H, and CUR-ERA.

### Comparing Randomized SVD algorithms

We now compare our algorithms with more recent randomized SVD algorithms available in the literature. Specifically, we considered Algorithms 4 and 5 from RSVDPACK [80], which are similar to Algorithm 1 but differ in how they postprocess the low-rank approximation to obtain the approximate SVD. We adapted these algorithms to incorporate block Hankel multiplication, and we denote the resulting eigenvalue decomposition based algorithm as RSVDeig-H ([80, Algorithm 4]), and the resulting QR decomposition based algorithm as RSVDqr-H ([80, Algorithm 5]). We use the same parameters for RSVDeig-H and RSVDqr-H as we do for RandSVD-H. First, we compare the singular values of $\boldsymbol{\mathcal{H}}_s$, the identified system matrix $\mathbf{A}_r$ using RandSVD-H and the two randomized algorithms RSVDeig-H and RSVDqr-H. These results are shown in Figure 4.7. We observe from the plot that all the algorithms produce similar results, both in terms of the singular values of $\boldsymbol{\mathcal{H}}_s$ and the eigenvalues of $\mathbf{A}_r$. Finally, we compare the average runtime using all the randomized algorithms. We observe that all four algorithms are competitive but

RSVDeig-H is the fastest, and for smaller $s$ values, RSVDqr-H is also faster than RandSVD-H. Therefore, since randomized SVD algorithm as described in Algorithm 1 is competitive with these variants, we continue to use it in the rest of our experiments but we note that an improvement in timing could be made by using these variants.

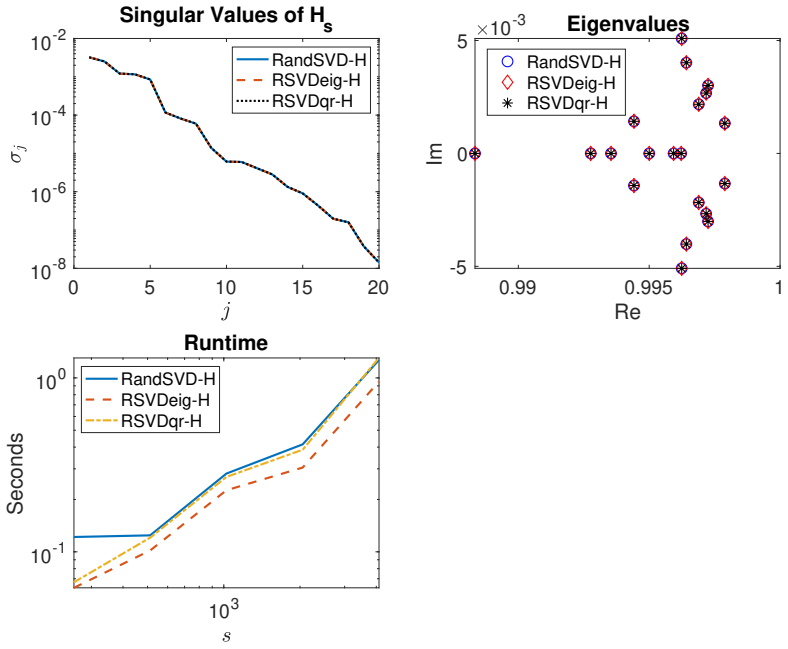


**Figure 4.7** Experiments with the heat transfer test problem comparing RandSVD-H to RSVDeig-H and RSVDqr-H. The top left figure shows the singular values of $\boldsymbol{\mathcal{H}}_s$ computed by all three algorithms, the top right figure shows the identified eigenvalues of $\mathbf{A}_r$ by using the randomized algorithms, and the bottom left figure shows the average run time in seconds of ERA using the three randomized algorithms.

### 4.5.2 Power system

For our next application, we consider the system identification of an electric power system model. Identification is an important component of power system modeling as the operating points in a power grid keep changing due to changes in loads, renewable generation profiles, and the network topology. All of this motivates the need for fast algorithms that can quickly update the small-signal model of the grid about these changing operating points from time to time [16]. The updated models, in turn, enable operators to make real-time control decisions for ensuring system stability. For the test problem, we consider a prototype model of reasonably large dimension, namely the IEEE 50-generator model with 145 buses, $n = 155$ state variables, and $m = 50$ input variables, as listed in Table 4.3. Details about the physical meaning of the states and model parameters of this test system can be found in [20]. The original IEEE

50-generator model is nonlinear. The model is, therefore, first linearized about a chosen power flow solution. The resulting LTI model is excited by 50 impulse functions injected through the excitation input terminals of the generators. All $\ell = 155$ states are measured, and are considered as outputs. The output matrix for this example is, therefore, the identity matrix.

An important feature of this problem is that it has a significantly larger number of inputs and outputs with a relatively small system size $n = 155$. Since the computational cost of RandSVD-H scales linearly with the product of the number of inputs and outputs, this suggests that RandTERA may have be computationally advantageous if the number of inputs and outputs can be reduced. Therefore, our experiments will compare RandTERA to RandSVD-H both in terms of accuracy and computational costs. For this problem, we again perform system identification as well as model reduction, taking the target rank to be $r = 75$. We found that model reduction (by using a truncated rank) is necessary for this power system test problem to achieve reasonable accuracy. If no model reduction is performed or if the target rank is too large, we encounter numerical instability in the eigenvalue reconstructions both in SVD-ERA and RandSVD-H.

### Accuracy

First, we will examine the accuracy of both RandTERA and RandSVD-H on the power system test problem. For all the experiments for testing accuracy, we will use $s = 500$, and target rank $r = 75$. For the randomized algorithms, we use an oversampling parameter $p = 20$, and $q = 1$ subspace iterations. Note that the full Hankel matrix for the accuracy experiments is of size $77500 \times 25000$.

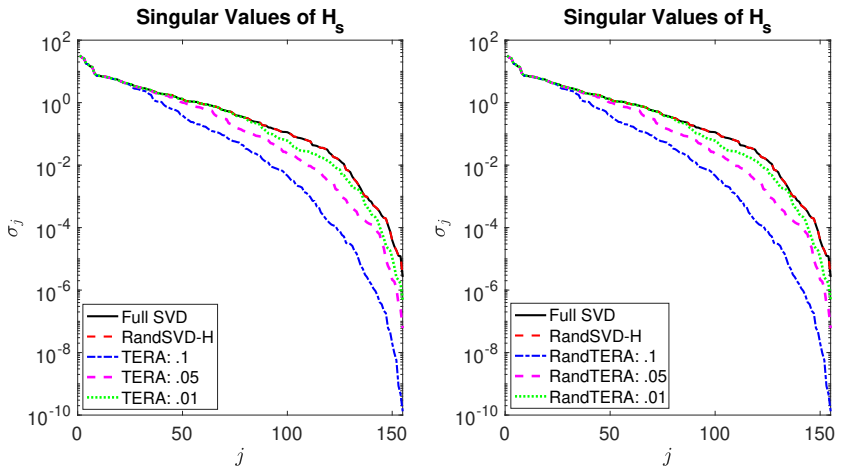


**Figure 4.8** Singular values of block Hankel matrix $\boldsymbol{\mathcal{H}}_s$ as computed by a full SVD and RandSVD-H versus those computed by (left) TERA with varying $\epsilon$ tolerances and (right) RandTERA with varying $\epsilon$ tolerances.

**Choice of $\ell'$ and $m'$.** In our first experiment, we will investigate the choice of the reduced number of inputs and outputs $\ell'$ and $m'$ for the tangential algorithms (TERA and RandTERA). To choose these parameters, we compute the singular values of $\boldsymbol{\mathcal{H}}_w$ and $\boldsymbol{\mathcal{H}}_e$ defined in (4.8), (4.9). The indices $\ell'$ and $m'$ are chosen to be the largest integers such that

$$\ell' = \arg\min_k \{1 \leq k \leq \ell \,|\sigma_k(\boldsymbol{\mathcal{H}}_w) \geq \epsilon\sigma_1(\boldsymbol{\mathcal{H}}_w)\}$$
$$m' = \arg\min_k \{1 \leq k \leq m \,|\sigma_k(\boldsymbol{\mathcal{H}}_e) \leq \epsilon\sigma_1(\boldsymbol{\mathcal{H}}_e)\}.$$

That is, they are chosen to be the smallest indices for which the singular values are within a threshold $\epsilon$ of the largest singular value. The values of $\ell'$ and $m'$ for different $\epsilon$ thresholds are shown in Table 4.4.

**Table 4.4** Reduced number of inputs $m'$ and outputs $\ell'$ based on the singular value threshold $\epsilon$ for use in TERA and RandTERA.

| $\epsilon$ | $m'$ | $\ell'$ |
|---|---|---|
| 0.1 | 14 | 18 |
| 0.05 | 21 | 33 |
| 0.01 | 30 | 59 |

**Accuracy of eigenvalues.** We first compare the accuracy of the singular values of the computed block Hankel matrix $\boldsymbol{\mathcal{H}}_s$. First, we show in Figure 4.8 the effect of different $\epsilon$ values on the accuracy of singular values computed by TERA and RandTERA. We can see that the smaller the tolerance $\epsilon$, the more accurate the singular values. Also, the singular values computed using TERA and RandTERA are less accurate compared to RandSVD-H, but the singular values computed by RandTERA are comparable with those of TERA. This shows that the randomized algorithms are accurate compared to their deterministic counterparts.

Next, we consider the eigenvalues of the identified $\widehat{\mathbf{A}}_r$ matrix using the RandSVD-H and RandTERA algorithms compared with the eigenvalues of the original $\mathbf{A}_r$ matrix. We use the parameters listed above, but we fix $\epsilon = 10^{-2}$. These results are shown in Figure 4.9. We can see that our algorithms identify the state matrix with eigenvalues very close to the true eigenvalues. To quantify how close they are, we consider the Hausdorff distance again as defined in (4.24). We computed the Hausdorff distance between the spectrum of the $\mathbf{A}_r$ matrix identified by an SVD-ERA and the spectra of the $\mathbf{A}_r$ matrices identified by RandSVD-H and RandTERA. We used $s = 500$ for all algorithms, and saw that for RandSVD-H, $h_d \approx 2.7 \times 10^{-4}$. For RandTERA, $h_d \approx 3.0 \times 10^{-3}$.

Our final measure of accuracy is the relative error in the Markov parameters as defined in (4.25). We plot this relative error in Figure 4.10, and make two observations. The first is that RandSVD-H has a much lower relative error than either TERA or RandTERA. The second

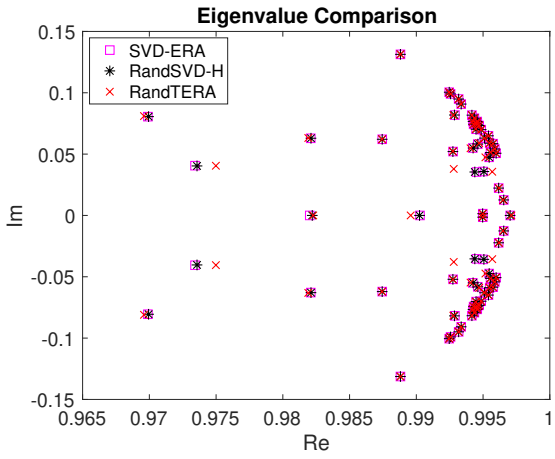


**Figure 4.9** Eigenvalues of the identified matrix $\widehat{\mathbf{A}}_r$ from SVD-ERA compared to the identified matrix $\widehat{\mathbf{A}}_r$ from RandSVD-H and RandTERA on the power system test problem with target rank $r = 75$.

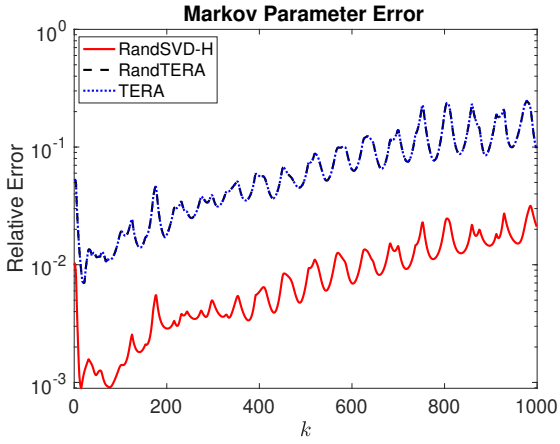


**Figure 4.10** Relative error in the reconstructed Markov parameters using RandTERA, TERA, and RandSVD-H algorithms. The relative error is computed using the system identified using the SVD-ERA.

is that the error from RandTERA is very accurate compared to the error from TERA, so the randomization is not causing a significant loss in the accuracy. We can see in Figure 4.10, that the relative error increases with increasing $k$. This is because the identification techniques using impulse response fail to identify the well damped poles (i.e., the poles whose real part is significantly lesser than 1) accurately, since information corresponding to these modes have to be obtained from the first few Markov parameters only. In Figure 4.11, we compare the impulse response of the original full, discrete-time model with that of the models identified using RandSVD-H and RandTERA (an impulse input was fed into Generator 1). The top plot shows the speed response of Generator 1 while the bottom plot shows that for Generator 2. We see that both the identified models have similar performance for Generator 1 and are close to the original trajectory, whereas in Generator 2, the trajectories from all algorithms match each

other closely, with only minor deviations from the "true" trajectory in the the first 2 seconds. The results for the other generators were observed to follow similar matching trends.

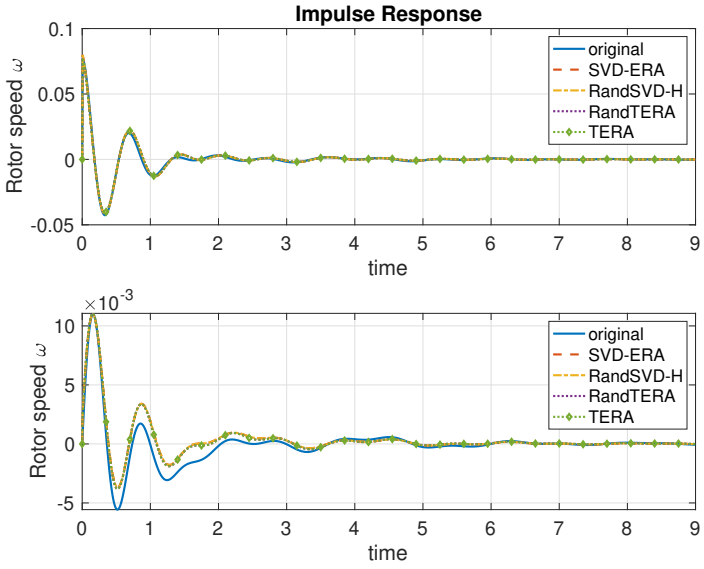


**Figure 4.11** Impulse response of the original discrete-time system compared against that for the systems identified using SVD-ERA, RandSVD-H, TERA, and RandTERA and excited with impulse input at Generator 1, with responses from Generator 1 (top) and Generator 2 (bottom).

### Timing

We next examine the computational cost of RandSVD-H and RandTERA on the power system test problem. We show in Table 4.5 the average CPU time of both of these algorithms compared to a full SVD as $s$ increases. As before, we averaged over three different runs, and used the same parameters for the randomized algorithms, $r = 75$, $p = 20$ and $q = 1$. We see that RandTERA is much cheaper compared to RandSVD-H, and both are much cheaper than a full SVD as expected. Note that for $s = 1000$, the matrix size was $155,000 \times 50,000$ and since this matrix was too large to store explicitly, we do not report the computational cost using the SVD.

**Table 4.5** Computational time in seconds of the SVD step in the identification algorithms SVD-ERA, RandSVD-H, TERA, and RandTERA. Note for $s = 1000$, the matrix was too large to store explicitly, so we only show the run times for the other algorithms.

| $s$ | size of $\boldsymbol{\mathcal{H}}_s$ | SVD | RandSVD-H | TERA | RandTERA |
|---|---|---|---|---|---|
| 500 | $77,500 \times 25,000$ | 1634.4 | 20.3 | 339.7 | 6.6 |
| 700 | $108,500 \times 35,000$ | 4292.0 | 28.4 | 795.2 | 11.7 |
| 1000 | $155,000 \times 50,000$ | N/A | 41.9 | 1983.9 | 18.0 |

## 4.6 Conclusions and Future work

We have presented two different randomized algorithms for accelerating the computational cost of system identification using ERA. The first algorithm accelerates the randomized subspace iteration by efficiently computing matrix vector products using the block Hankel structure. The second algorithm uses the previous algorithm, but on a matrix with a reduced number of inputs and outputs using tangential interpolation. The algorithms no longer have the cubic scaling with $s$, which controls the number of time steps. The error analysis relates the accuracy of the eigenvalues of the identified system matrix to the accuracy of the singular vectors of the block Hankel matrix $\boldsymbol{\mathcal{H}}_s$. Numerical experiments from different applications show that our algorithms are both accurate and computationally efficient.

There are several avenues for future work. First, the analysis in Section 4.4 is general and does not make assumptions on the specific algorithm used to compute the low-rank approximation to $\boldsymbol{\mathcal{H}}_s$. The analysis can be specialized by using specific results for the accuracy of the singular vectors, for example, following the approaches in [62, 65, 68]. Second, the paper assumes that we have access to the Markov parameters. This is not the case when it is possible to excite the system using general inputs. One possibility, as mentioned earlier, is to use the approach in [46] which may not be feasible when a large number of Markov parameters need to be estimated. Algorithms such as MOESP and N4SID (see [77, Chapter 9]) can be used for general inputs but also suffer from high computational costs. However, we can combine the block Hankel structure with randomized algorithms for the general input case as well. Such techniques which involve identification using general input can be extremely accurate in identifying the well-damped poles as they constant excite the system with a persistently exciting signal. This is an ongoing research direction.

## 4.7 Acknowledgements

We would like to thank our coauthors, Jishnudeep Kar and Aranya Chakraborrty, as well as Eric Hallman for useful comments and for generously sharing his code with us.

# Chapter 5

# Efficient Tensor-based Approximations to Kernel Interactions

## 5.1 Introduction

Kernel methods are used to model pairwise interactions among a set of points. They are popular in many applications in numerical analysis and scientific computing such as integral equations, radial basis functions, Gaussian processes, machine learning, geostatistics and spatiotemporal statistics, and Bayesian inverse problems. A major challenge in working with kernel methods is that the resulting kernel matrices are dense, and are difficult to store and compute with when the number of interaction points is large. Many fast kernel summation methods have been developed over the years including the Barnes-Hut algorithm [4], Fast Multipole Method [33], Fast Gauss Transform [34], etc.

Rank-structured matrices are a class of dense matrices that are useful for efficiently representing kernel matrices. In this approach, the kernel matrix is represented as a hierarchy of subblocks, and certain off-diagonal subblocks are approximated in a low-rank format. There are several different types of hierarchical formats for rank-structured matrices, including $\mathcal{H}$-matrices, $\mathcal{H}^2$-matrices, hierarchically semi-separable (HSS) matrices, and hierarchically off-diagonal low-rank (HODLR) matrices. For more details on hierarchical matrices, see [37, 40, 31, 12, 39, 66]. If the number of interaction points is $n$, then the matrix can be stored approximately using $\mathcal{O}(n(\log n)^\alpha)$ entries rather than $n^2$ entries, and the cost of forming matrix-vector products (matvecs) is $\mathcal{O}(n(\log n)^\beta)$ flops, where $\alpha, \beta \geq 0$ are nonnegative integers depending on the format and the algorithm for computing the rank-structured matrix. A major challenge in working with rank-structured matrices is the high cost of computing low-rank approximations.

Low-rank approximations arise due to kernel interactions corresponding to two sets of points: the source points and the target points. Assuming that the kernel is sufficiently smooth

and the source and target points are well-separated, it is reasonable to approximate the interactions between the two sets of points by a low-rank matrix. This is explained more clearly in Subsection 5.2. The matrix representing the interactions between these sets of points can be compressed using the singular value decomposition (SVD). However, the cost of the SVD can be large when the number of source and target points are large, and also because the low-rank approximations need to be computed repeatedly over several sets of source and target points. Over the years, there have been many efforts to compute efficient low-rank approximations in the context of kernel methods. While it is not our goal to do a comprehensive survey, we mention a few important works to set our work in the right context. The low-rank approximations can be generally classified into three different schemes: analytic, semi-analytic, and algebraic methods.

Analytic methods employ Taylor expansions or other kernel-dependent function expansions to approximate a kernel; the Fast Multipole Method (FMM) is an example of an analytic method. These methods are generally the most accurate, but one major downside of analytic methods is that they are specific to the kernel in question, and require the development of a new expansion for each new kernel. Semi-analytic methods instead only require evaluations of the kernel, and can be implemented in a kernel-independent manner. These methods approximate the interactions between large sets of points by computing interactions between smaller, strategically selected, sets of points. There are many examples of semi-analytic methods in recent work, including [14, 82, 18, 84, 83]. The third type of method, algebraic methods, also only require kernel evaluations. The difference between algebraic and semi-analytic is that algebraic methods work directly with the matrix-vector product of the kernel. These methods then employ numerical linear algebra methods for approximation. The most commonly used algebraic method is the adaptive cross approximation (ACA). One recent example of an algebraic method is [7], which builds on the ACA to create a more efficient method.

Our approach falls into the semi-analytic category, and builds on the black-box fast multipole method (BBFMM) low-rank approximation approach in Fong and Darve [30], which uses Chebyshev interpolation to approximate the kernel interactions. In this method, instead of computing the kernel interactions between every pair of points given, the interactions are only computed between points on Chebyshev grids. This significantly reduces the number of kernel evaluations needed. This work is also similar to the semi-analytic methods in [14, 82], as the new approaches in these papers also use some form of Chebyshev interpolation to reduce the number of kernel interactions. Our work differs in the subsequent low-rank approximation techniques, as we use tensor-based methods instead of the matrix techniques presented in these papers.

**Contributions and Contents.** In this chapter, we propose new tensor-based methods to compute efficient low-rank kernel approximations in two spatial dimensions. Our approach is based on the BBFMM approach, which uses Chebyshev interpolation, and has four steps: mapping the block matrix representing the kernel interactions between Chebyshev grids into a four dimensional tensor; using novel tensor approximation algorithms; mapping the resulting

tensor approximation to obtain a compressed block matrix; and further compressing the block matrix to obtain a low-rank matrix approximation. Though the proposed approach is kernel-independent, we demonstrate the performance of our algorithms on a wide variety of kernels as well as different source and target point configurations. The resulting algorithms are accurate and computationally efficient. One of the algorithms (Section 5.3, Method 2) dramatically cuts down the number of kernel evaluations compared to standard approaches. It is worth pointing out that the novelty in our approach lies both in the use of the tensor-based approach as well as in the tensor compression methods themselves. The use of tensor decompositions allows us to exploit the latent multidimensional structure in order to obtain efficiently computed, accurate approximations. We anticipate that this tensor-based compression approach is more broadly applicable to block matrices.

We first review, in Section 5.2, the problem setup and the black-box fast multipole method from [30]. Then, in Section 5.3, we present our new tensor-based approach described above. Finally, we show the accuracy of our algorithms in various settings in Section 5.4.

## 5.2 Background

Before we present our low-rank approximation algorithms, we review the necessary background information. We explain in detail the problem setup and the underlying assumptions, and review the Chebyshev interpolation approach at the heart of the BBFMM.

### Problem Setup and Notation

The computational bottleneck of working with rank-structured matrices involves the efficient low-rank approximation of the off-diagonal blocks. Before showing how to compute these off-diagonal blocks efficiently, we define the problem setup that we will use in the rest of this chapter. Let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_s}\}$ and $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{N_t}\}$ be $N_s$ and $N_t$ source and target points, respectively. These points are assumed to be enclosed by two boxes, $B_s$ and $B_t$, defined as

$$\begin{aligned} B_s &= [a_1, b_1] \times [a_2, b_2], \\ B_t &= [c_1, d_1] \times [c_d, d_2], \end{aligned} \tag{5.1}$$

such that $\mathcal{X} \subset B_s$ and $\mathcal{Y} \subset B_t$. These bounding boxes are plotted in Figure 5.1 for visualization. Then, for the kernel $\kappa : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$, let the interaction matrix $\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y})$ be defined as

$$\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y}) = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{y}_1) & \kappa(\mathbf{x}_1, \mathbf{y}_2) & \ldots & \kappa(\mathbf{x}_1, \mathbf{y}_{n_t}) \\ \kappa(\mathbf{x}_2, \mathbf{y}_1) & \kappa(\mathbf{x}_2, \mathbf{y}_2) & \ldots & \kappa(\mathbf{x}_2, \mathbf{y}_{N_t}) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_{N_s}, \mathbf{y}_1) & \kappa(\mathbf{x}_{N_s}, \mathbf{y}_2) & \ldots & \kappa(\mathbf{x}_{N_s}, \mathbf{y}_{N_t}) \end{bmatrix}. \tag{5.2}$$

We can only represent blocks in low-rank form if those blocks are *admissible*. There are two

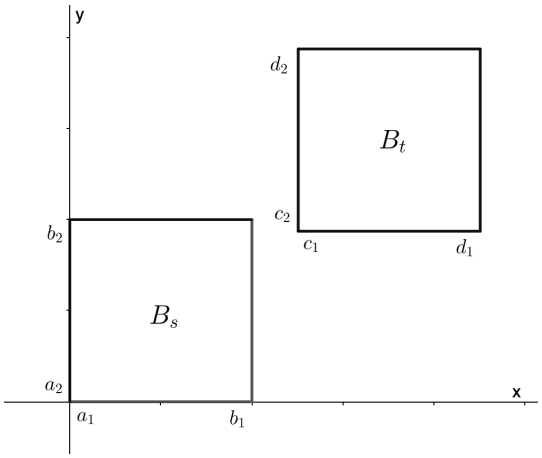


**Figure 5.1** Visual of the bounding boxes for source points, $B_s$, and for target points, $B_t$. Note that $\mathcal{X} \subset B_s$ and $\mathcal{Y} \subset B_t$

types of admissibility, weakly admissible and strongly admissible. Note that the diameters of the bounding blocks $B_s$ and $B_t$ are

$$\begin{aligned} \operatorname{diam}(B_s) &= \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}, \\ \operatorname{diam}(B_t) &= \sqrt{(d_1 - c_1)^2 + (d_2 - c_2)^2}. \end{aligned} \tag{5.3}$$

We define the distance $\operatorname{dist}(B_s, B_t)$ between these blocks as the minimum distance between any two points from $B_s$ and $B_t$. Specifically,

$$\operatorname{dist}(B_s, B_t) = \min_{\substack{(x_s, y_s) \in B_s \\ (x_t, y_t) \in B_t}} \left( \sqrt{(x_t - x_s)^2 + (y_t - y_s)^2} \right). \tag{5.4}$$

The domains $B_s$ and $B_t$ are considered *strongly admissible* if they satisfy

$$\max\{\operatorname{diam}(B_s), \operatorname{diam}(B_t)\} \leq \eta \operatorname{dist}(B_s, B_t), \tag{5.5}$$

where $\eta > 0$ is the admissibility parameter. This means two sets of points are strongly admissible if they are sufficiently well-separated. Two blocks are *weakly admissible* if there is no overlap between them.

We now formally state the problem we seek to address. Given points $\mathcal{X} \subset B_s$ and $\mathcal{Y} \subset B_t$, we seek to compute the pairwise kernel interactions between the sets of points, i.e., $\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y})$. This is expensive both from a computational and storage perspective. If we simply compute the kernel interaction for each pair of points, this requires $N_s N_t$ kernel interactions and the cost of storage is $\mathcal{O}(N_s N_t)$. As $N_s$ and $N_t$ can get quite large (for our later numerical examples $N_s = N_t = 2000$), computing $\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y})$ can become computationally challenging. Assuming that the points are well-separated so that the kernel interactions between the source and targets can be treated as a smooth function, the matrix $\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y})$ can be approximated in a low-rank format.

However, computing this low-rank approximation is computationally expensive. To this end, our goal is to compute an approximation to $\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y})$ with a cost that is linear in $N_s$ and $N_t$, which we accomplish using the black-box FMM (BBFMM) approach.

### 5.2.1 Kernel approximation using Chebyshev interpolation

In this chapter, we use the BBFMM approach to obtain a low-rank approximation to the kernel $\kappa$ [30], which is based on Chebyshev interpolation. The general idea of this method is to treat the kernel as a multidimensional function over 4 spatial variables: two variables representing the sources and two variables representing the targets. Then we use Chebyshev interpolation of the multidimensional function. Assuming the number of Chebshev grid points is $n$ for each dimension, instead of computing the pairwise interactions between the source and target points, we only need to compute $n^4$ kernel interactions. We choose $n$, the number of Chebyshev nodes, to be significantly smaller than either $N_s$ or $N_t$ so that the Chebyshev approximation is much more efficient to evaluate than the dense matrix $\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y})$. A visual representation of this process is provided in Figure 5.2 but the details will be given later in this section.

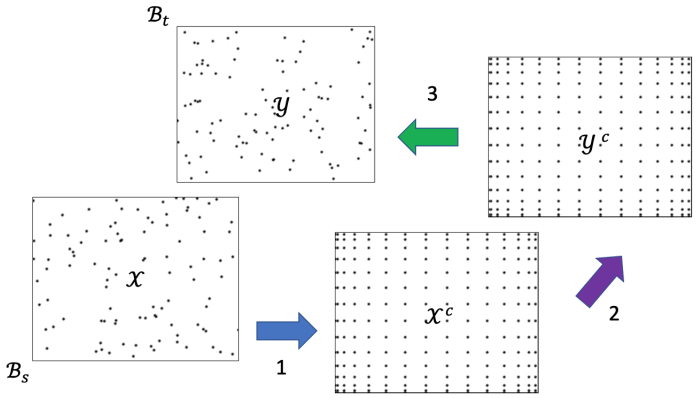


**Figure 5.2** Visual representation of mapping source and target points to Chebyshev grids, shown by steps 1 and 3. Step 2 shows computing the interactions between Chebyshev grid. Each numbered step corresponds to a matrix shown in Figure 5.3.

#### Chebyshev interpolation

It is easiest to first give details for interpolating a function $g$: $[a, b] \to \mathbb{R}$ using Chebyshev interpolation. We first construct the Chebyshev points of the first kind in the interval $[-1, 1]$; these are given by $\xi_k = \cos\left(\frac{2k-1}{2n}\pi\right)$ for $k = 1, \ldots, n$. Using the mapping $I_{[a,b]}$: $[-1, 1] \to [a, b]$ defined as

$$I_{[a,b]}(x) = (x + 1)\frac{(b - a)}{2} + a, \tag{5.6}$$

we obtain the Chebyshev points in the interval $[a, b]$ as $x_k^c = I_{[a,b]}(\xi_k)$ for $k = 1, \ldots, n$. This mapping is invertible, with the inverse $I_{[a,b]}^{-1}(x) = 2\frac{(x-a)}{(b-a)} - 1$. Recall that for a function $g(x)$ with $x \in [a, b]$, the Chebyshev interpolation polynomial of degree $n-1$ is

$$\pi_{n-1}(x) = \sum_{k=0}^{n-1} c_k T_k(I_{[a,b]}^{-1}(x)),$$

where $T_k(x) = \cos(k \cos^{-1}(x))$ is the Chebyshev polynomial of degree $k$, and $c_k$ are the coefficients of the Chebyshev polynomials. The coefficients $c_k$ can be obtained using the interpolating conditions and the discrete orthogonality of the Chebyshev polynomials. If we define an interpolating polynomial $S_n^{[a,b]}(x, y)$ as

$$S_n^{[a,b]}(x, y) = \frac{1}{n} + \frac{2}{n} \sum_{k=1}^{n-1} T_k \left( I_{[a,b]}^{-1}(x) \right) T_k \left( I_{[a,b]}^{-1}(y) \right), \tag{5.7}$$

then the polynomial $\pi_{n-1}(x)$ can also be expressed as

$$\pi_{n-1}(x) = \sum_{k=1}^{n} g(x_k^c) S_n^{[a,b]}(x_k^c, x). \tag{5.8}$$

**Kernel approximation**

This simple one-dimensional Chebyshev approximation can be extended to obtain an approximation to the kernel, $\kappa$: $\mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$. Consider two points $\mathbf{x} \in B_s$ and $\mathbf{y} \in B_t$, respectively the bounding boxes of the sources and targets. We can consider the kernel $\kappa$ as a function $f$: $\mathbb{R}^4 \to \mathbb{R}$, i.e.,

$$f(x_1, x_2, y_1, y_2) = \kappa(\mathbf{x}, \mathbf{y}). \tag{5.9}$$

Assuming the bounding boxes $B_s$ and $B_t$ are strongly admissible, the kernel can be approximated by a degenerate representation. Let $\{x_{1j}^c\}_{j=1}^n$, $\{x_{2j}^c\}_{j=1}^n$, $\{y_{1j}^c\}_{j=1}^n$, and $\{y_{2j}^c\}_{j=1}^n$ denote the Chebyshev points of the first kind over the intervals $[a_1, b_1]$, $[a_2, b_2]$, $[c_1, d_1]$ and $[c_2, d_2]$ respectively. Using Chebyshev approximation to $f$, the approximation to $\kappa$ takes the form

$$\kappa(\mathbf{x}, \mathbf{y}) \approx \sum_{i,j,k,\ell} f(x_{1i}^c, x_{2j}^c, y_{1k}^c, y_{2\ell}^c) S_n^{[a_1,b_1]}(x_{1i}^c, x_1) S_n^{[a_2,b_2]}(x_{2j}^c, x_2) S_n^{[c_1,d_1]}(y_{1k}^c, y_1) S_n^{[c_2,d_2]}(y_{2\ell}^c, y_2). \tag{5.10}$$

It is straightforward to extend it to the case where the number of Chebyshev points are different along each dimension but we avoid this for simplicity. We denote by $\mathcal{X}^c = \{\mathbf{x}_1^c, \ldots, \mathbf{x}_{n^2}^c\}$ the $n^2$ Chebyshev grid points over the source domain $B_s$, and by $\mathcal{Y}^c = \{\mathbf{y}_1^c, \ldots, \mathbf{y}_{n^2}^c\}$ the $n^2$ Chebyshev grid points over the target domain $B_t$.

We now show how this kernel approximation can be used to obtain a low-rank approximation to the matrix $\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y})$. A visual representation of this approximation is shown in Figure 5.3 and consists of three matrices. We start with matrix 2, whose entries are defined as the pairwise

kernel interactions between Chebyshev grid points $\mathcal{X}^c$ and $\mathcal{Y}^c$, the mapped source and target points, respectively. This matrix, which we call $\mathbf{M} \in \mathbb{R}^{n^2 \times n^2}$, is defined as

$$\mathbf{M} = \boldsymbol{\mathcal{K}}(\mathcal{X}^c, \mathcal{Y}^c) = \begin{bmatrix} \kappa(\mathbf{x}_1^c, \mathbf{y}_1^c) & \kappa(\mathbf{x}_1^c, \mathbf{y}_2^c) & \dots & \kappa(\mathbf{x}_1^c, \mathbf{y}_{n^2}^c) \\ \kappa(\mathbf{x}_2^c, \mathbf{y}_1^c) & \kappa(\mathbf{x}_2^c, \mathbf{y}_2^c) & \dots & \kappa(\mathbf{x}_2^c, \mathbf{y}_{n^2}^c) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_{n^2}^c, \mathbf{y}_1^c) & \kappa(\mathbf{x}_{n^2}^c, \mathbf{y}_2^c) & \dots & \kappa(\mathbf{x}_{n^2}^c, \mathbf{y}_{n^2}^c) \end{bmatrix} \in \mathbb{R}^{n^2 \times n^2}. \tag{5.11}$$

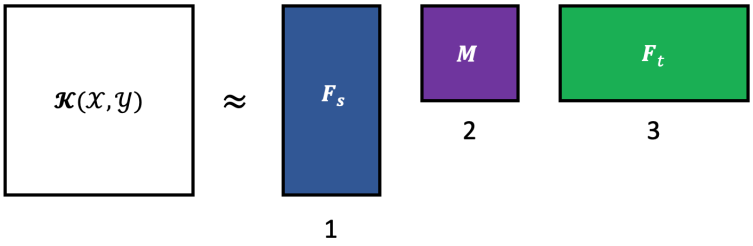


**Figure 5.3** Matrices comprising the interaction matrix approximation process, corresponding to steps shown in Figure 5.2. Matrix 1 maps the source points to a Chebyshev grid, matrix 3 maps the target points to a Chebyshev grid, and matrix 2 computes the interactions between Chebyshev grids.

The other two matrices in this approximation, represented by steps 1 and 3 in Figures 5.2 and 5.3, interpolate the source points $\mathcal{X}$ and target points $\mathcal{Y}$ to Chebyshev grids $\mathcal{X}^c$ and $\mathcal{Y}^c$, respectively. Let the matrix interpolating $\mathcal{X}$ to $\mathcal{X}^c$ be denoted as $\mathbf{F}_s$, which will have dimensions $N_s \times n^2$. Then let the matrix interpolating $\mathcal{Y}$ to $\mathcal{Y}^c$ be denoted as $\mathbf{F}_t$, which will have dimensions $N_t \times n^2$. These matrices correspond to the portions of (5.10). We compute $\mathbf{F}_s$ and $\mathbf{F}_t$ columnwise, using the Hadamard product, i.e.,

$$\mathbf{F}_s(:, j) = S_n^{[a_1, b_1]}(x_{j,1}^c, \mathbf{x}_1') \odot S_n^{[a_2, b_2]}(x_{j,2}^c, \mathbf{x}_2') \in \mathbb{R}^{N_s},$$

where $\mathbf{x}_1' \in \mathbb{R}^{N_s}$ is a vector of all the $x$-coordinates of the source points, and $\mathbf{x}_2' \in \mathbb{R}^{N_s}$ is a vector of all the $y$-coordinates of the source points. $\mathbf{F}_t$ is formed similarly, using target points $\{\mathbf{y}_i\}$ and Chebyshev grid points $\{\mathbf{y}_i^c\}$ instead, and letting $\mathbf{y}_1'$ and $\mathbf{y}_2'$ be defined as vectors of the $x$- and $y$- coordinates, respectively, of the target points. Having computed $\mathbf{F}_s$ and $\mathbf{F}_t$ in this manner, we have an approximation to the interaction matrix

$$\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y}) \approx \mathbf{F}_s \mathbf{M} \mathbf{F}_t^\top. \tag{5.12}$$

The details of this approximation process are given in Algorithm 14.

Once the approximation in (5.12) is computed using Algorithm 14, the cost of storing the low-rank approximation is $\mathcal{O}(n^2(N_s + N_t))$. This is also the cost of a matrix-vector product

**Algorithm 14** Black-box FMM

**Input:** number of nodes $n$, kernel $\kappa$, $N_s$ source points, $N_t$ target points
**Output:** $\mathbf{F}_s \in \mathbb{R}^{N_s \times n^2}, \mathbf{M} \in \mathbb{R}^{n^2 \times n^2}, \mathbf{F}_t \in \mathbb{R}^{N_t \times n^2}$

Construct Chebyshev points $\mathcal{X}^c$ and $\mathcal{Y}^c$
Compute $\mathbf{M}$ with entries $\mathbf{M}_{ij} = \kappa(\mathbf{x}_i^c, \mathbf{y}_j^c)$
**for** $j = 1 : n^2$ **do**
    Compute $\mathbf{F}_s^1(:, j) = S_n^{[a_1, b_1]}(x_{j,1}^c, \mathbf{x}_1')$
    Compute $\mathbf{F}_s^2(:, j) = S_n^{[a_2, b_2]}(x_{j,2}^c, \mathbf{x}_2')$
    Compute $\mathbf{F}_t^1(:, j) = S_n^{[c_1, d_1]}(y_{j,1}^c, \mathbf{y}_1')$
    Compute $\mathbf{F}_t^2(:, j) = S_n^{[c_2, d_2]}(y_{j,2}^c, \mathbf{y}_2')$
**end for**
Compute $\mathbf{F}_s = \mathbf{F}_s^1 \odot \mathbf{F}_s^2$
Compute $\mathbf{F}_t = \mathbf{F}_t^1 \odot \mathbf{F}_t^2$

with the approximation. While this cost is smaller than $\mathcal{O}(N_s N_t)$, it can be further reduced by compression using the SVD. We will develop new tensor-based approximation algorithms in the next sections that will reduce the computational cost required to compute these approximations.

## 5.3 Tensor-based approximation algorithms

In this section, we will describe our approach for compressing block matrices using a tensor-based framework. This framework is based on the BBFMM approach described in Section 5.2.1. There are four main steps in this process:

### Algorithm Framework

**Step 1:** Map block matrix $\mathbf{M} \in \mathbb{R}^{n^2 \times n^2}$ to tensor $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{n \times n \times n \times n}$.

**Step 2:** Compress $\boldsymbol{\mathcal{M}}$ to obtain low-rank approximation $\widehat{\boldsymbol{\mathcal{M}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}]$.

**Step 3:** Map $\widehat{\boldsymbol{\mathcal{M}}}$ back to block matrix $\widehat{\mathbf{M}}$.

**Step 4:** Recompress $\widehat{\mathbf{M}}$ using rank-$r_m$ SVD to obtain approximation $\widehat{\mathbf{M}}_r = \widehat{\mathbf{U}}\widehat{\boldsymbol{\Sigma}}\widehat{\mathbf{V}}^\top$.

We first describe the process of mapping to and from tensors, which form steps 1 and 3 of the general algorithmic framework. The three variants on this framework we will present help to reduce the computational cost compared to the SVD.

### 5.3.1 Mapping to and from Tensors

Suppose $\mathbf{M}$ is a block matrix with blocks $\mathbf{M}^{(k,\ell)} \in \mathbb{R}^{m\times n}$ defined as follows:

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}^{(1,1)} & \mathbf{M}^{(1,2)} & \dots & \mathbf{M}^{(1,q)} \\ \mathbf{M}^{(2,1)} & \mathbf{M}^{(2,2)} & \dots & \mathbf{M}^{(2,q)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{M}^{(p,1)} & \mathbf{M}^{(p,2)} & \dots & \mathbf{M}^{(p,q)} \end{bmatrix} \in \mathbb{R}^{mp\times qn}.$$

We can naturally reshape this into a four-dimensional tensor $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{m\times n\times p\times q}$ by taking each block $\mathbf{M}^{(k,\ell)}$ as slices for the third and fourth modes, i.e. $\boldsymbol{\mathcal{M}}_{:,:,k,\ell} = \mathbf{M}^{(k,\ell)}$.

Now let

$$\widehat{\boldsymbol{\mathcal{M}}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \times_4 \mathbf{D},$$

be an approximation to $\boldsymbol{\mathcal{M}}$ in the Tucker format, computed using any suitable approach. To map this approximation back to a block matrix, we extract the slices for the third and fourth modes, as

$$\begin{aligned} \widehat{\boldsymbol{\mathcal{M}}}_{:,:,k,\ell} &= \widehat{\boldsymbol{\mathcal{M}}} \times_3 \mathbf{e}_k^\top \times_4 \mathbf{e}_\ell^\top \\ &= \boldsymbol{\mathcal{G}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{e}_k^\top \mathbf{C} \times_4 \mathbf{e}_\ell^\top \mathbf{D} \\ &= \boldsymbol{\mathcal{G}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}_{k,:} \times_4 \mathbf{D}_{\ell,:}, \end{aligned} \tag{5.13}$$

where the notation $\mathbf{C}_{k,:}$ refers to the $k$-th row of factor matrix $\mathbf{C}$, and similarly for $\mathbf{D}_{\ell,:}$. Recall that, for a tensor $\boldsymbol{\mathcal{X}}$ and matrix $\mathbf{Y}$ of compatible dimensions, mode-1 tensor multiplication $\boldsymbol{\mathcal{X}} \times_1 \mathbf{Y}$ can be expressed as $\boldsymbol{\mathcal{X}} \times_1 \mathbf{Y} = \mathbf{Y}\mathbf{X}_{(1)}$. Similarly, mode-2 tensor multiplication $\boldsymbol{\mathcal{X}} \times_2 \mathbf{Y}$ can be expressed as $\boldsymbol{\mathcal{X}} \times_2 \mathbf{Y} = \mathbf{X}_{(2)}\mathbf{Y}^\top$. Thus, recognizing that $\boldsymbol{\mathcal{G}} \times_3 \mathbf{C}_{k,:} \times_4 \mathbf{D}_{\ell,:}$ is a matrix, we can then reorder the multiplication from the last line of (5.13) and rewrite the mode 1 and 2 products such that

$$\widehat{\boldsymbol{\mathcal{M}}}_{:,:,k,\ell} = \mathbf{A}(\boldsymbol{\mathcal{G}} \times_3 \mathbf{C}_{k,:} \times_4 \mathbf{D}_{\ell,:})\mathbf{B}^\top = \mathbf{A}\widehat{\mathbf{G}}^{(k,\ell)}\mathbf{B}^\top,$$

where $\widehat{\mathbf{G}}^{(k,\ell)} = \boldsymbol{\mathcal{G}} \times_3 \mathbf{C}_{k,:} \times_4 \mathbf{D}_{\ell,:}$. Thus, an approximation to the block $\mathbf{M}^{(k,\ell)}$ is

$$\mathbf{M}^{(k,\ell)} \approx \widehat{\boldsymbol{\mathcal{M}}}_{:,:,k,\ell} = \mathbf{A}\widehat{\mathbf{G}}^{(k,\ell)}\mathbf{B}^\top.$$

This allows us to form an approximation to the full block matrix

$$\mathbf{M} \approx \begin{bmatrix} \mathbf{A} & & \\ & \ddots & \\ & & \mathbf{A} \end{bmatrix} \begin{bmatrix} \widehat{\mathbf{G}}^{(1,1)} & \dots & \widehat{\mathbf{G}}^{(1,q)} \\ \vdots & \ddots & \vdots \\ \widehat{\mathbf{G}}^{(p,1)} & \dots & \widehat{\mathbf{G}}^{(p,q)} \end{bmatrix} \begin{bmatrix} \mathbf{B}^\top & & \\ & \ddots & \\ & & \mathbf{B}^\top \end{bmatrix}$$

$$= (\mathbf{I}_p \otimes \mathbf{A}) \begin{bmatrix} \widehat{\mathbf{G}}^{(1,1)} & \dots & \widehat{\mathbf{G}}^{(1,q)} \\ \vdots & \ddots & \vdots \\ \widehat{\mathbf{G}}^{(p,1)} & \dots & \widehat{\mathbf{G}}^{(p,q)} \end{bmatrix} (\mathbf{I}_q \otimes \mathbf{B}^\top).$$

To summarize, the framework we developed here allows us to take a block matrix, map it to a four-dimensional tensor, compress it in tensor form, and map the result back to an approximation of the original block matrix. This allows us to use tensor compression methods, which often are more computationally efficient than standard matrix methods, to approximate block matrices. This technique can be applied to our block matrix $\mathbf{M}$ from Subsection 5.2.1 simply by letting all dimensions $m, n, p, q$ be the number of Chebyshev nodes in each spatial dimension, $n$. We will describe in more detail how we will apply this technique to obtain an approximation for $\mathbf{M}$.

### 5.3.2 Method 1: Randomized Interpolatory Tensor Decomposition

The first compression algorithm we present follows the same steps as the Algorithm Framework at the start of the section. In step 2, to compress the tensor we use a variation of the structure-preserving randomized algorithm that we previously proposed in [61, Algorithm 5.1]. In mode 1, we apply the Randomized Row Interpolatory Decomposition (RRID) with target rank $r_t$ and oversampling parameter $p$ along each mode to obtain the low-rank approximation

$$\mathbf{M}_{(1)} \approx \mathbf{A}\mathbf{P}_1^\top \mathbf{M}_{(1)},$$

Here $\mathbf{P}_1 \in \mathbb{R}^{n \times (r_t+p)}$ has columns from the identity matrix corresponding to the index set $\mathcal{J}_1$. This process is repeated across each mode to obtain the other factor matrices $\mathbf{B}, \mathbf{C}, \mathbf{D}$ and the index sets $\mathcal{J}_2, \mathcal{J}_3, \mathcal{J}_4$. Finally, we have the low-rank Tucker representation

$$\boldsymbol{\mathcal{M}} \approx \widehat{\boldsymbol{\mathcal{M}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}], \qquad \boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{M}}(\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \mathcal{J}_4).$$

In contrast to Algorithm 9 in which we used sequential truncation, our low-rank approach can be described as the structure-preserving HOSVD algorithm. After obtaining the compressed tensor, we map our core tensor and factor matrices to block matrix approximation $\widehat{\mathbf{M}}$ as described in Section 5.3.1. Finally, to obtain an approximation of the desired rank $r_m$, we compute the rank-$r_m$ SVD of $\widehat{\mathbf{M}}$. The details of this algorithm are given in Algorithm 15.

To analyze the computational cost of this algorithm, assume that the target tensor rank is

$(r_t, r_t, r_t, r_t)$ and the target matrix rank is $r_m$. First, we discuss the cost of tensor compression. We apply the RRID algorithm to each mode unfolding of size $n \times n^3$. This costs $\mathcal{O}(r_t n^4)$ flops. After mapping the compressed tensor to a matrix of size $n(r_t + p) \times n(r_t + p)$; to compress this matrix further we use the Randomized SVD with cost $\mathcal{O}(r_m r_t^2 n^2)$ flops.

**Algorithm 15** Method 1: Randomized Interpolatory Tensor Decomposition

**Input:** block matrix $\mathbf{M}$, number of nodes $n$, tensor target rank $(r_t, r_t, r_t, r_t)$, matrix target rank $r_m$, oversampling parameter $p$ such that $r_t + p \leq n$ and $r_m + p \leq n(r_t + p)$

**Output:** approximation $\widehat{\mathbf{M}}_r = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$

1: Map matrix $\mathbf{M}$ to tensor $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{n \times n \times n \times n}$ {Step 1}
2: Compute $[\mathbf{A}, \mathcal{J}_1] = \text{RRID}(\mathbf{M}_{(1)}, r_t, p)$ {Start Step 2: Tensor Compression}
3: Compute $[\mathbf{B}, \mathcal{J}_2] = \text{RRID}(\mathbf{M}_{(2)}, r_t, p)$
4: Compute $[\mathbf{C}, \mathcal{J}_3] = \text{RRID}(\mathbf{M}_{(3)}, r_t, p)$
5: Compute $[\mathbf{D}, \mathcal{J}_4] = \text{RRID}(\mathbf{M}_{(4)}, r_t, p)$
6: Form core tensor $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{M}}(\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \mathcal{J}_4)$ {End Step 2}
7: Map $\widehat{\boldsymbol{\mathcal{M}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}]$ back to block matrix $\widehat{\mathbf{M}}$ {Step 3}
8: Apply RandSVD with target rank $r_m$ and oversampling parameter $p$ to obtain $\widehat{\mathbf{M}}_r = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$ {Step 4}

### 5.3.3 Method 2: Randomized Interpolatory Tensor Decomposition with Block Selection

Our second compression algorithm aims to further reduce the cost of computing the tensor decomposition, by working with a subsampled tensor rather than the entire tensor. Recall that we have assumed that the number of Chebyshev grid points in each spatial dimension is the same. Suppose we are given an index set $\mathcal{I}$ with cardinality $|\mathcal{I}| = b$ representing the subsampled fibers. In the first step of the algorithm, we process mode 1. We consider the subsampled tensor $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}}(:, \mathcal{I}, \mathcal{I}, \mathcal{I})$ and apply the RRID algorithm to $\mathbf{X}_{(1)}$ obtain the matrix $\mathbf{A}$ and the index set $\mathcal{J}_1$. We then have a low-rank approximation to $\boldsymbol{\mathcal{M}}$ as

$$\mathbf{M}_{(1)} \approx \mathbf{A}\mathbf{P}_1^\top \mathbf{X}_{(1)}, \qquad \boldsymbol{\mathcal{M}} \approx \boldsymbol{\mathcal{M}}(\mathcal{J}_1, :, :, :) \times_1 \mathbf{A}.$$

Here $\mathbf{P}_1 \in \mathbb{R}^{n \times r}$ has columns from the identity matrix corresponding to the index set $\mathcal{J}_1$. This process is repeated across each mode to obtain the other factor matrices $\mathbf{B}, \mathbf{C}, \mathbf{D}$ and the index sets $\mathcal{J}_2, \mathcal{J}_3, \mathcal{J}_4$. Finally, we have the low-rank Tucker representation

$$\boldsymbol{\mathcal{M}} \approx \widehat{\boldsymbol{\mathcal{M}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}], \qquad \boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{M}}(\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \mathcal{J}_4).$$

After obtaining the compressed tensor as before, we map our core tensor and factor matrices to block matrix approximation $\widehat{\mathbf{M}}$ as described in Section 5.3.1. Finally, to obtain an approximation

of the desired rank $r_m$, we compute the rank-$r_m$ SVD of $\widehat{\mathbf{M}}$. The summary of all steps for our second variation is described in Algorithm 16.

Compared with Method 1, in each mode we are working with the subsampled version of the mode unfolding rather than the entire mode-unfolding. In other words, setting $b = n$ we can recover Method 1. In numerical experiments, we found that taking $b \sim$ 3-5 is sufficient for obtaining an accurate tensor decomposition. Furthermore, as long as the subsampled indices covered the range of the indices $\{1, \ldots, n\}$, the accuracy of the tensor decomposition was not affected by the choice of indices. In practice, we chose $b = 3$ with $\mathcal{I} = \{1, \lceil n/2 \rceil, n\}$.

To examine the accuracy of Method 2, we present an experiment involving the canonical angles for Method 2 compared to Method 1. The setup for this experiment is given in Subsection 5.4.1. For mode 1 of tensor $\boldsymbol{\mathcal{M}}$, we compute the angles between the left singular vectors computed by using $\mathbf{M}_{(1)}$ and the left singular vectors computed by using $\mathbf{X}_{(1)}$ with $b = 1, 2, 3, 4$. The cosine of these angles are plotted in Figure 5.4. In the figure, we see that choosing $b = 3$ or $b = 4$ gives an accurate representation, as all the canonical angles are nearly zero for those values of $b$. We can increase $b$ to be larger in order to obtain higher accuracy if needed, but we found that $b = 3$ was sufficient for good accuracy.

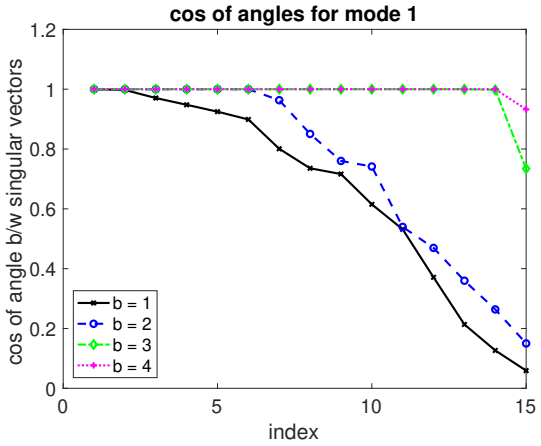


**Figure 5.4** Cosine of the angles between true and approximate left singular vectors of $\mathbf{M}_{(1)}$ as $b$ increases.

Now consider the computational cost. At the first step, the unfolded matrix $\mathbf{X}_{(1)}$ has dimensions $n \times b^3$. Applying subset selection for each mode requires $\mathcal{O}(r_t n b^3)$ flops; this is also the cost for the entire tensor decomposition. To compute the low-rank matrix decomposition, we require an additional $\mathcal{O}(r_m r_t^2 n^2)$ flops. The number of kernel evaluations are $4nb^3 + r_t^4$; in practice, this is much smaller than $n^4$ since we choose $b, r_t \ll n$.

**Algorithm 16** Method 2: Randomized Interpolatory Tensor Decomposition with Block Selection

**Input:** block matrix $\mathbf{M}$, number of nodes $n$, tensor target rank $(r_t, r_t, r_t, r_t)$, index set $\mathcal{I} \subset \{1, \ldots, n\}$ with cardinality $b$ such that $b^3 \geq r_t$, matrix target rank $r_m$, oversampling parameter $p$ such that $r_t + p \leq \min\{n, b^3\}$ and $r_m + p \leq n(r_t + p)$

**Output:** approximation $\widehat{\mathbf{M}}_r = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$

1: Map matrix $\mathbf{M}$ to tensor $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{n\times n\times n\times n}$ {Step 1}
2: Form $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}}(:, \mathcal{I}, \mathcal{I}, \mathcal{I})$ {Start Step 2: Tensor Compression}
3: Compute $[\mathbf{A}, \mathcal{J}_1] = \text{RRID}(\mathbf{X}_{(1)}, r_t, p)$
4: Form $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}}(\mathcal{I}, :, \mathcal{I}, \mathcal{I})$
5: Compute $[\mathbf{B}, \mathcal{J}_2] = \text{RRID}(\mathbf{X}_{(2)}, r_t, p)$
6: Form $\mathbf{X} = \boldsymbol{\mathcal{M}}(\mathcal{I}, \mathcal{I}, :, \mathcal{I})$
7: Compute $[\mathbf{C}, \mathcal{J}_3] = \text{RRID}(\mathbf{X}_{(3)}, r_t, p)$
8: Form $\mathbf{X} = \boldsymbol{\mathcal{M}}(\mathcal{I}, \mathcal{I}, \mathcal{I}, :)$
9: Compute $[\mathbf{D}, \mathcal{J}_4] = \text{RRID}(\mathbf{X}_{(4)}, r_t, p)$
10: Form core tensor $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{M}}(\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \mathcal{J}_4)$ {End Step 2}
11: Map $\widehat{\boldsymbol{\mathcal{M}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}]$ back to block matrix $\widehat{\mathbf{M}}$ {Step 3}
12: Apply RandSVD with target rank $r_m$ and oversampling parameter $p$ to obtain $\widehat{\mathbf{M}}_r = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$ {Step 4}

### 5.3.4 Method 3: Randomized Kronecker Product

In the third compression algorithm, we are essentially computing R-HOSVD for the tensor compression step. However, the random matrix is generated as the Kronecker product of Gaussian random matrices. More specifically, consider three $\mathbf{\Phi}, \mathbf{\Psi}, \mathbf{\Omega} \in \mathbb{R}^{n\times(r_t+p)}$ which are standard Gaussian random matrices. Here $(r_t, r_t, r_t, r_t)$ is the target tensor rank and $p$ is the oversampling parameter. In mode 1, we form the tensor

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}} \times_2 \mathbf{\Phi}^\top \times_3 \mathbf{\Psi}^\top \times_4 \mathbf{\Omega}^\top \in \mathbb{R}^{n\times(r_t+p)\times(r_t+p)\times(r_t+p)},$$

and compute the left singular vectors of the mode-1 unfolding $\mathbf{X}_{(1)}$ to obtain the factor matrix $\mathbf{A}$. We call this the Randomized Kronecker Product approach since using the definition of mode products

$$\mathbf{X}_{(1)} = \mathbf{M}_{(1)}(\mathbf{\Omega} \otimes \mathbf{\Psi} \otimes \mathbf{\Phi}) \in \mathbb{R}^{n\times(r_t+p)^3}.$$

To obtain the factor matrices $\mathbf{B}, \mathbf{C}$ and $\mathbf{D}$, we follow a similar procedure along modes 2, 3, and 4. Finally, to obtain the core tensor we compute $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{M}} \times_1 \mathbf{A}^\top \times_2 \mathbf{B}^\top \times_3 \mathbf{C}^\top \times_4 \mathbf{D}^\top$. The rest of the algorithm resembles Method 1 and is detailed in Algorithm 17. The asymptotic cost of Method 3 is the same as Method 1. However, a potential advantage over Method 1 is the reduction in the number of random entries generated. In Method 1, in using a RRID along each mode, we need to generate $n^3(r_t + p)$ random numbers, whereas in Method 3, we only need to generate $3n(r_t + p)$ random numbers.

**Algorithm 17** Method 3: Randomized Kronecker Product

**Input:** block matrix $\mathbf{M}$, number of nodes $n$, tensor target rank $(r_t, r_t, r_t, r_t)$, matrix target rank $r_m$, oversampling parameter $p$ such that $r_t + p \leq n$ and $r_m + p \leq n(r_t + p)$

**Output:** approximation $\widehat{\mathbf{M}}_r = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$

1: Map matrix $\mathbf{M}$ to tensor $\boldsymbol{\mathcal{M}} \in \mathbb{R}^{n\times n\times n\times n}$ {Step 1}
2: Draw $\mathbf{\Phi}, \mathbf{\Psi}, \mathbf{\Omega} \in \mathbb{R}^{n\times(r_t+p)}$ Standard Gaussian random matrices {Start Step 2: Tensor Compression}
3: Form $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}} \times_2 \mathbf{\Phi}^\top \times_3 \mathbf{\Psi}^\top \times_4 \mathbf{\Omega}^\top$
4: Compute thin-SVD $\mathbf{X}_{(1)} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$; Set $\mathbf{A} = \mathbf{U}(:, 1 : r_t + p)$.
5: Form $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}} \times_1 \mathbf{\Phi}^\top \times_3 \mathbf{\Psi}^\top \times_4 \mathbf{\Omega}^\top$
6: Compute thin-SVD $\mathbf{X}_{(2)} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$; Set $\mathbf{B} = \mathbf{U}(:, 1 : r_t + p)$.
7: Form $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}} \times_1 \mathbf{\Phi}^\top \times_2 \mathbf{\Psi}^\top \times_4 \mathbf{\Omega}^\top$
8: Compute thin-SVD $\mathbf{X}_{(3)} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$; Set $\mathbf{C} = \mathbf{U}(:, 1 : r_t + p)$.
9: Form $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{M}} \times_1 \mathbf{\Phi}^\top \times_2 \mathbf{\Psi}^\top \times_3 \mathbf{\Omega}^\top$
10: Compute thin-SVD $\mathbf{X}_{(4)} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$; Set $\mathbf{D} = \mathbf{U}(:, 1 : r_t + p)$.
11: Form core tensor $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{M}} \times_1 \mathbf{A}^\top \times_2 \mathbf{B}^\top \times_3 \mathbf{C}^\top \times_4 \mathbf{D}^\top$ {End Step 2}
12: Map $\widehat{\boldsymbol{\mathcal{M}}} = [\boldsymbol{\mathcal{G}}; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}]$ back to block matrix $\widehat{\mathbf{M}}$ {Step 3}
13: Apply RandSVD with target rank $r_m$ and oversampling parameter $p$ to obtain $\widehat{\mathbf{M}}_r = \widehat{\mathbf{U}}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{V}}^\top$ {Step 4}

### 5.3.5 Computational Cost

Suppose we use the SVD to compress interaction matrix $\kappa(\mathcal{X}, \mathcal{Y})$; the cost of compression is then $\mathcal{O}(\max\{N_s, N_t\} \min\{N_s, N_t\}^2)$ flops. If the number of sources and targets are large, then this cost scales cubically with the number of points. By using the BBFMM, the cost of forming the kernel approximation $\kappa(\mathcal{X}, \mathcal{Y}) \approx \mathbf{F}_S \mathbf{M} \mathbf{F}_T^\top$ becomes $\mathcal{O}(n^2(N_s + N_t) + n^4)$ flops, since forming $\mathbf{F}_S, \mathbf{F}_T$ cost $\mathcal{O}(n^2(N_s + N_t))$ flops and forming $\mathbf{M}$ costs $\mathcal{O}(n^4)$ flops. Since the cost of storage of the BBFMM approximation can be large when $n$ is large, we have proposed various techniques for compressing $\mathbf{M}$.

We now summarize the costs of the various compression techniques at our disposal. It should be noted that to obtain a low-rank approximation to $\kappa(\mathcal{X}, \mathcal{Y})$, we need to include the cost of forming $\mathbf{F}_S$ and $\mathbf{F}_T$ which cost $\mathcal{O}(n^2(N_s + N_t))$ flops. If the SVD is used for compressing $\mathbf{M}$, then the cost is $\mathcal{O}(n^6)$ flops; if RandSVD is used instead of SVD, then the cost is $\mathcal{O}(n^4 r_m)$ flops. Methods 1 and 3 both cost $\mathcal{O}\left(r_t n^4 + n^2 r_t^2 r_m\right)$ flops, whereas Method 2 costs $\mathcal{O}(b^3 n r_t + n^2 r_t^2 r_m)$ flops. Method 2 is the most computationally efficient of the proposed algorithms if $b, r_t \ll n$, and if $b = n$, Method 2 has the same cost as Methods 1 and 3. Note that Methods 1 and 3 have comparable cost to RandSVD directly computed on $\mathbf{M}$, but typically the tensor rank $r_t$ is chosen to be smaller than $r_m$, so we expect the tensor-based methods to be more efficient for the same accuracy. As mentioned earlier, Method 3 may be preferable to Method 1 since it requires generating fewer random numbers.

Another potential benefit of Method 2 is the number of kernel evaluations needed to form $\mathbf{M}$. In nearly all the other approaches, the matrix $\mathbf{M}$ needs to be formed explicitly requiring $n^4$

**Table 5.1** Summary of the computational cost of the various algorithms. The upper table represents the computational cost of the standard approaches, and the lower table represents the computational costs associated with the tensor compression methods proposed. Note that for the lower table we need to include the cost of forming $\mathbf{F}_S, \mathbf{F}_T$ which costs $\mathcal{O}(n^2(N_s + N_t))$. Here, $n$ is the number of Chebyshev nodes, $r_t$ is the tensor target rank, $r_m$ is the matrix target rank, $p$ is the oversampling parameter, and $b$ is the number of blocks drawn from $\mathbf{M}$.

| Approximation Method | Cost | Kernel Evals. | Storage |
|---|---|---|---|
| SVD on $\kappa(\mathcal{X}, \mathcal{Y})$ | $\mathcal{O}(\max\{N_s, N_t\} \min\{N_s, N_t\}^2)$ | $N_s N_t$ | $\mathcal{O}(r(N_s + N_t))$ |
| BBFMM | $\mathcal{O}(n^2(N_s + N_t) + n^4)$ | $n^4$ | $\mathcal{O}(n^4 + n^2 N_s N_t)$ |
| BBFMM + SVD | $\mathcal{O}(n^6 + n^2(N_s + N_t))$ | $n^4$ | $\mathcal{O}(r(N_s + N_t))$ |

| Compression Method | Cost | Kernel Evals. | Storage |
|---|---|---|---|
| RandSVD | $\mathcal{O}(n^4 r_m)$ | $n^4$ | $\mathcal{O}(r(N_s + N_t))$ |
| Method 1 | $\mathcal{O}\left(r_t n^4 + n^2 r_t^2 r_m\right)$ | $n^4$ | $\mathcal{O}(r(N_s + N_t))$ |
| Method 2 | $\mathcal{O}(b^3 n r_t + n^2 r_t^2 r_m)$ | $b^3 n + (r_t + p)^4$ | $\mathcal{O}(r(N_s + N_t))$ |
| Method 3 | $\mathcal{O}\left(r_t n^4 + n^2 r_t^2 r_m\right)$ | $n^4$ | $\mathcal{O}(r(N_s + N_t))$ |

kernel evaluations. In Method 2, only a portion of the tensor $\boldsymbol{\mathcal{M}}$ needs to be formed. That is, we only need $b^3 n + (r_t + p)^4$ kernel evaluations. Assuming $b = r_t = p = 5$ and $n = 30$, we only need $(b^3 n + (r_t + p)^4)/n^4 \approx 0.017$. In other words, we only need 2% of the kernel evaluations, which can be beneficial if the kernel evaluations are expensive.

There are other benefits to using our tensor-based compression method. Consider a time-dependent problem in which the source and target points are changing in time but are contained within the same bounding boxes $B_s$ and $B_t$. Although the matrices $\mathbf{F}_S, \mathbf{F}_T$ change in time, the kernel interactions in $\mathbf{M}$ do not change, and the matrix approximation $\widehat{\mathbf{M}}$ can be precomputed and deployed efficiently in the time dependent problem. This can yield significant reductions in computational cost and has low storage costs since the resulting approximations can be stored efficiently.

## 5.4 Numerical Results

In this section, we present numerical results that demonstrate the accuracy of our algorithms. We first explain the standard parameters used in most of our experiments, and then test our algorithms in various situations.

All experiments were conducted on a laptop with 8GB memory and a 2.5 GHz Intel Core i5 processor. We will first present experiments varying the choice of kernel, then experiments varying the box parameters for the source and target points, and finally an experiment testing the accuracy with different numbers of Chebyshev nodes.

### 5.4.1 Standard Parameters

For most of the experiments we use to test our algorithms, we will use the kernel $\kappa(\mathbf{x}, \mathbf{y}) = 1/\|\mathbf{x} - \mathbf{y}\|$ and $n = 30$ Chebyshev nodes. For the randomized algorithms, we use oversampling parameter $p = 5$ and no subspace iterations. For all experiments, we take target ranks $r_t = r_m$ for simplicity. For Method 2, we will take $b = 3$.

We define our source and target boxes in the following way. Let the source box have one vertex at the origin and side length $L$. The target box also has side length $L$, and let $D$ be the distance between the bottom left vertices. Finally, define $\theta$ to be the angle describing the placement of the targets box in relation to the sources box. A visual representation of this setup is in Figure 5.5. Unless otherwise stated, we will use $L = 5$, $D = 10$, and $\theta = \pi/4$. In the boxes defined in this manner, we will generate $N_s = 2000$ and $N_t = 2000$ uniformly randomly distributed source and target points, respectively. To relate these parameters back to our setup in Figure 5.1, the value of $D$ loosely corresponds to the admissibility parameter $\eta$ in that it determines how well-separated the boxes are. Similarly, $L$ is somewhat related to the diameter of each box.

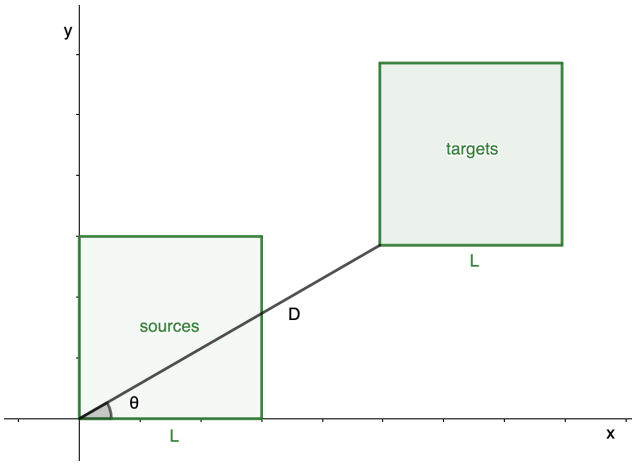


**Figure 5.5** Box setup for our numerical experiments, where $L$ is the length of both the source and target boxes, $D$ is the distance between bottom left corners of boxes, and $\theta$ is the angle describing the placement of the target box.

### 5.4.2 Experiments

In our first experiment, we consider the relative error in the approximation to $\boldsymbol{\mathcal{K}}(\mathcal{X}, \mathcal{Y})$ produced by our algorithms for various commonly used kernels. In Figure 5.6, we plot the relative error produced by using an SVD and RandSVD of matrix $\mathbf{M}$ compared to the relative error produced by our three methods (Algorithms 15, 16, 17). For each kernel, the relative errors produced by each algorithm are all very close to each other.

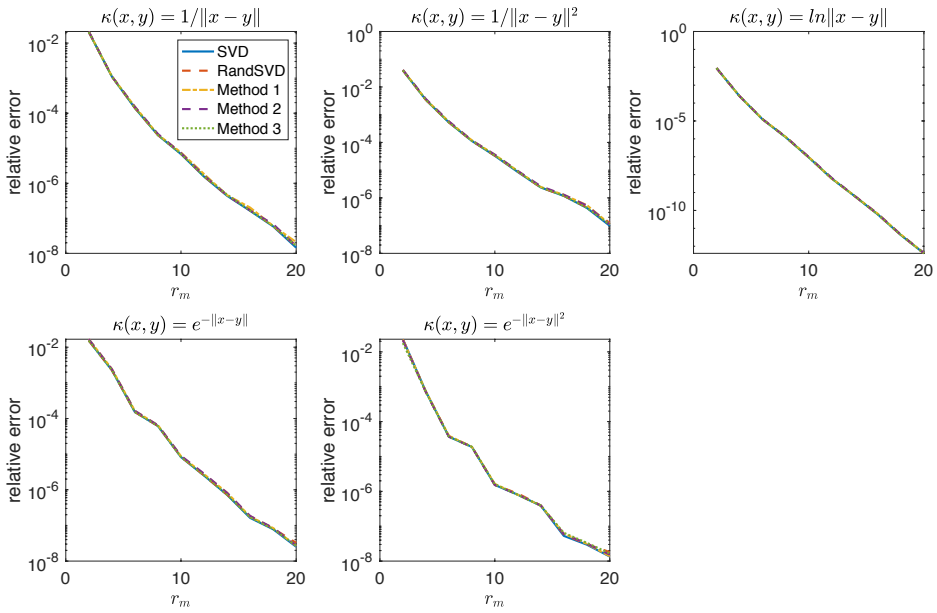


**Figure 5.6** Relative error of SVD, RandSVD, and Methods 1, 2 and 3 with respect to increasing target rank for five different kernels $\kappa(\mathbf{x}, \mathbf{y})$

Next, we change the side length $L$ of the source and target point boxes, and use the standard parameters in Subsection 5.4.1 for everything else. We keep the same number of source and target points, just increasing the size of the boxes in which we generate those points. Starting with a box side length of $L = 1$, we increase to $L = 5$, then $L = 10$, and finally $L = 30$, to examine how the relative error changes. We keep the distance fixed at $D = 50$. The relative errors produced by our three methods as well as SVD and RandSVD are shown in Figure 5.7. We note that the relative error increases as $L$ increases. This is to be expected, because as the box size increases but the number of points remains constant, the distinction between the source points and target points becomes less clear. This also means that the source and target boxes are not as well-separated for larger $L$ values. In the figure, we see that all five algorithms perform similarly to each other with the different side length values.

In this next experiment, we keep the box side length fixed at $L = 5$, and instead change the distance between source and target boxes. We still keep the number of points $N_s$ and $N_t$ the same, and we examine the relative error produced by SVD, RandSVD, and our three methods as the boxes become more well separated. This again is plotted as the target rank increases, and the results are shown in Figure 5.8. We start with the boxes $D = 7.1$ apart, and increase to $D = 10$ then $D = 15$, and finally $D = 20$. As the boxes get farther apart, the relative error decreases, supporting our assumption that the source and target points must be well-separated to produce accurate results.

Finally, we test the accuracy of our algorithms with different values of $n$, the number of Chebyshev nodes. We keep all other parameters the same as the standard parameters detailed

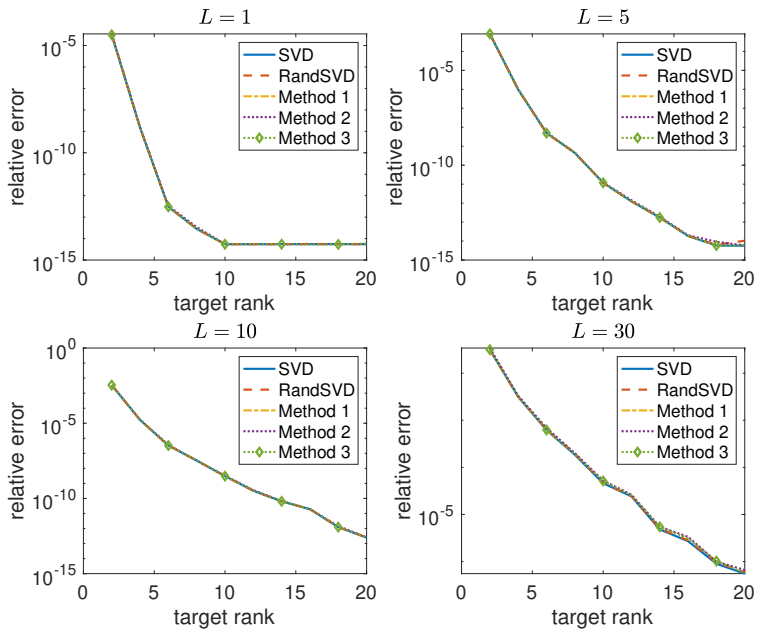


**Figure 5.7** Relative error produced by SVD, RandSVD, and Methods 1, 2, and 3 with increasing target rank for four different side lengths $L$ of the source and target boxes.

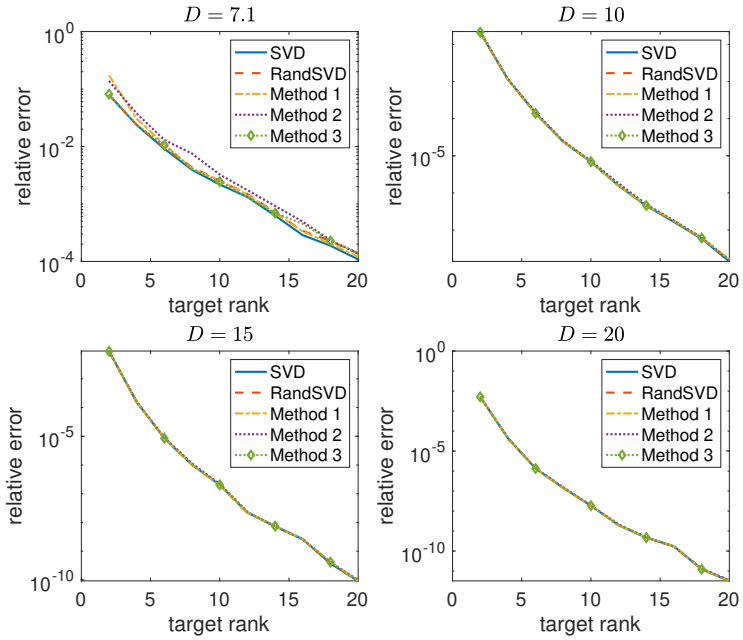


**Figure 5.8** Relative error produced by SVD, RandSVD, and Methods 1, 2, and 3 with increasing target rank for different, increasing distance between the source and target boxes.

in Subsection 5.4.1. The relative errors produced by SVD, RandSVD, and Methods 1, 2, and 3 are plotted in Figure 5.9. We see that the error decreases as the number of nodes $n$ increases,

and that again, our new algorithms perform extremely similarly to the standard matrix-based methods.

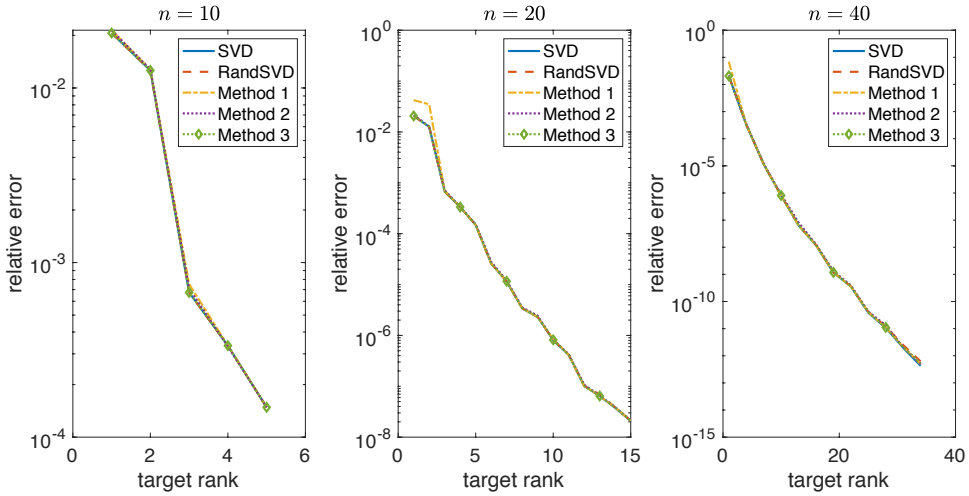


**Figure 5.9** Relative error produced by SVD, RandSVD, and Methods 1,2, and 3 with increasing target rank for three different values of $n$, the number of Chebyshev nodes. The left figure is $n = 10$, middle is $n = 20$, and right is $n = 45$.

## 5.5 Conclusions

In this chapter, we developed new, tensor-based algorithms to compute efficient low-rank kernel approximations. Our tensor-based framework built on the black-box fast multipole method with four steps. We mapped a block matrix to a four-dimensional tensor, compressed the tensor using three new tensor compression algorithms, mapped the compressed tensor back to a compressed block matrix representation, and finally compressed the block matrix to obtain a low-rank matrix approximation. The tensor compression algorithms we presented are accurate and computationally efficient, with one method significantly reducing the number of needed kernel evaluations. These benefits were shown explicitly for several different experimental conditions through our numerical experiments and our analysis of the computational complexity.

# Chapter 6

# Conclusions

In this thesis, we developed new randomized algorithms for tensors and matrices in various applications. First, we presented new randomized algorithms and analysis for low-rank tensor approximation algorithms in the Tucker format. This included adaptive algorithms for problems where the target rank is unknown as well as algorithms that preserve the structure of the original tensor. These algorithms are more computationally efficient than standard tensor compression algorithms, and we also showed that the structure preserving algorithms are particularly beneficial for large, sparse tensors. We also provided probabilistic analysis for our new algorithms as well as for previously developed algorithms, namely R-HOSVD and R-STHOSVD.

Next, we presented two new randomized algorithms that accelerate the traditional system identification algorithm, ERA. Our first algorithm efficiently computes matrix vector products by exploiting block Hankel structure. The second algorithm also takes advantage of block Hankel structure, but with a matrix that has a reduced number of inputs and outputs from tangential interpolation. These algorithms reduced the computational cost associated with traditional ERA, and we showed that they are accurate as well. We also included error analysis on the accuracy and stability of the system matrices identified using the approximation algorithms.

In our final chapter, we presented randomized tensor-based methods for approximating kernel interactions efficiently. Our methods fit into a framework where we, building on the black-box multipole method, map the resulting block matrix to a four-dimensional tensor, compress the tensor, and map back to a low-rank block matrix approximation. Within this framework, we developed three new tensor compression algorithms. The first two algorithms combine randomized techniques and subset selection to reduce computational cost of traditional methods, and the third algorithm takes a Kronecker product of random matrices in order to reduce the number of generated random entries. These algorithms are accurate and computationally efficient compared to standard matrix algorithms, and in addition to fitting within a framework for computing efficient low-rank kernel approximations, can be applicable to other situations.

**Potential future directions.** There are several potential research directions that stem from this work. First, we can extend the structure preserving decomposition idea from Chapter 3

to other tensor decomposition formats such as the tensor train decomposition. There are also many applications involving large, sparse multidimensional data where our algorithms would be helpful, such as in image processing. An open direction for system identification is working with the general input problem instead of impulse input as we covered in Chapter 4. Work is currently being done in this area. A first direction for our work with kernel methods is to provide analysis for the algorithms presented in Chapter 5. Also, constructing rank-structured matrices with our low-rank approximation algorithms is a natural extension for this work. There are also other potential applications for the tensor compression algorithms we developed as well as for the technique we use to map a block matrix to a tensor and back. One example is for approximating matrices constructed from clustering large-scale data.

# BIBLIOGRAPHY


[1] A. C. Antoulas. *Approximation of large-scale dynamical systems*, volume 6. Society for Industrial and Applied Mathematics, 2005.

[2] AT&T Laboratories at Cambridge. Olivetti database of faces. `https://cs.nyu.edu/~roweis/data.html`, 2002.

[3] B. W. Bader and T. G. Kolda. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, December 2007.

[4] J. Barnes and P. Hut. A hierarchical $\mathcal{O}(n \log n)$ force-calculation algorithm. *Nature*, 324(6096):446–449, 1986.

[5] K. Batselier, W. Yu, L. Daniel, and N. Wong. Computing low-rank approximations of large-scale matrices with the tensor network randomized SVD. *SIAM Journal on Matrix Analysis and Applications*, 39(3):1221–1244, 2018.

[6] C. Battaglino, G. Ballard, and T. G. Kolda. A practical randomized CP tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 39(2):876–901, 2018.

[7] M. Bebendorf and S. Kunis. Recompression techniques for adaptive cross approximation. *The Journal of Integral Equations and Applications*, pages 331–357, 2009.

[8] P. Benner and J. Saak. Linear-quadratic regulator design for optimal cooling of steel profiles. Technical Report SFB393/05-05, Sonderforschungsbereich 393 *Parallele Numerische Simulation für Physik und Kontinuumsmechanik*, TU Chemnitz, D-09107 Chemnitz (Germany), 2005.

[9] R. Bhatia. *Matrix analysis*, volume 169. Springer Science & Business Media, 2013.

[10] D. J. Biagioni, D. Beylkin, and G. Beylkin. Randomized interpolative decomposition of separated representations. *Journal of Computational Physics*, 281:116–134, 2015.

[11] Å. Björck. *Numerical methods in matrix computations*, volume 59. Springer, 2015.

[12] S. Börm, L. Grasedyck, and W. Hackbusch. Hierarchical matrices. *Lecture notes*, 21:2003, 2003.

[13] M. E. Broadbent, M. Brown, K. Penner, I. Ipsen, and R. Rehman. Subset selection algorithms: Randomized vs. deterministic. *SIAM undergraduate research online*, 3(01), 2010.

[14] L. Cambier and E. Darve. Fast low-rank kernel matrix factorization using skeletonized interpolation. *SIAM Journal on Scientific Computing*, 41(3):A1652–A1680, 2019.

[15] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.

[16] G. Chavan, M. Weiss, A. Chakrabortty, S. Bhattacharya, A. Salazar, and F. Ashrafi. Identification and predictive analysis of a multi-area WECC power system model using synchrophasors. *IEEE Transactions on Smart Grid*, 8(4):1977–1986, 2017.

[17] M. Che and Y. Wei. Randomized algorithms for the approximations of Tucker and the Tensor Train decompositions. *Advances in Computational Mathematics*, 45(1):395–428, 2019.

[18] C. Chen, S. Aubry, T. Oppelstrup, A. Arsenlis, and E. Darve. Fast algorithms for evaluating the stress field of dislocation lines in anisotropic elastic media. *Modelling and Simulation in Materials Science and Engineering*, 26(4):045007, 2018.

[19] H. Cheng, Z. Gimbutas, P.-G. Martinsson, and V. Rokhlin. On the compression of low rank matrices. *SIAM Journal on Scientific Computing*, 26(4):1389–1404, 2005.

[20] J. H. Chow and K. W. Cheung. A toolbox for power system dynamics and control engineering education and research. *IEEE Transactions on Power Systems*, 7(4):1559–1564, 1992.

[21] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016.

[22] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, and D. P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives. *Foundations and Trends® in Machine Learning*, 9(6):431–673, 2017.

[23] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[24] L. De Lathauwer, B. De Moor, and J. Vandewalle. On the best rank-1 and rank-$(r_1, r_2, ..., r_n)$ approximation of higher-order tensors. *SIAM journal on Matrix Analysis and Applications*, 21(4):1324–1342, 2000.

[25] P. Drineas and M. W. Mahoney. A randomized algorithm for a tensor-based generalization of the singular value decomposition. *Linear algebra and its applications*, 420(2-3):553–571, 2007.

[26] P. Drineas and M. W. Mahoney. RandNLA: randomized numerical linear algebra. *Communications of the ACM*, 59(6):80–90, 2016.

[27] Z. Drmač and A. K. Saibaba. The discrete empirical interpolation method: Canonical structure and formulation in weighted inner product spaces. *SIAM Journal on Matrix Analysis and Applications*, 39(3):1152–1180, 2018.

[28] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[29] N. B. Erichson, K. Manohar, S. L. Brunton, and J. N. Kutz. Randomized CP tensor decomposition. *Machine Learning: Science and Technology*, 1(2):025012, 2020.

[30] W. Fong and E. Darve. The black-box fast multipole method. *Journal of Computational Physics*, 228(23):8712–8725, 2009.

[31] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.

[32] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.

[33] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[34] L. Greengard and J. Strain. The fast Gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[35] M. Gu. Subspace iteration randomization and singular value problems. *SIAM Journal on Scientific Computing*, 37(3):A1139–A1173, 2015.

[36] M. Gu and S. C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.

[37] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part i: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[38] W. Hackbusch. *Tensor spaces and numerical tensor calculus*, volume 42. Springer Science & Business Media, 2012.

[39] W. Hackbusch. *Hierarchical matrices: algorithms and analysis*, volume 49. Springer, 2015.

[40] W. Hackbusch and S. Börm. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. *Computing*, 69(1):1–35, 2002.

[41] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[42] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.

[43] B. Huber, R. Schneider, and S. Wolf. A randomized Tensor Train singular value decomposition. In *Compressed Sensing and its Applications*, pages 261–290. Springer, 2017.

[44] B. Hunter and T. Strohmer. Performance analysis of spectral clustering on compressed, incomplete and inaccurate measurements. *arXiv preprint arXiv:1011.0997*, 2010.

[45] J. Jacod and P. Protter. *Probability essentials*. Springer Science & Business Media, 2012.

[46] J.-N. Juang, M. Phan, L. G. Horta, and R. W. Longman. Identification of observer/Kalman filter Markov parameters-theory and experiments. *Journal of Guidance, Control, and Dynamics*, 16(2):320–329, 1993.

[47] T. G. Kolda. A counterexample to the possibility of an extension of the Eckart–Young low-rank approximation theorem for the orthogonal rank tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 24(3):762–767, 2003.

[48] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[49] B. Kramer and A. A. Gorodetsky. System identification via CUR-factored Hankel approximation. *SIAM Journal on Scientific Computing*, 40(2):A848–A866, 2018.

[50] B. Kramer and S. Gugercin. Tangential interpolation-based eigensystem realization algorithm for MIMO systems. *Mathematical and Computer Modelling of Dynamical Systems*, 22(4):282–306, 2016.

[51] D. Kressner and L. Perisa. Recompression of Hadamard products of tensors in Tucker format. *SIAM Journal on Scientific Computing*, 39(5):A1879–A1902, 2017.

[52] S.-Y. Kung. A new identification and model reduction algorithm via singular value decomposition. In *Proc. 12th Asilomar Conf. on Circuits, Systems and Computers, Pacific Grove, CA, November, 1978*, 1978.

[53] L. Ljung. *System Identification: Theory for the user*. Prentice Hall, 1999.

[54] M. W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

[55] M. W. Mahoney, M. Maggioni, and P. Drineas. Tensor-CUR decompositions for tensor-based data. *SIAM Journal on Matrix Analysis and Applications*, 30(3):957–987, 2008.

[56] O. A. Malik and S. Becker. Low-rank Tucker decomposition of large tensors using TensorSketch. In *Advances in Neural Information Processing Systems*, pages 10096–10106, 2018.

[57] O. A. Malik and S. Becker. Fast randomized matrix and tensor interpolative decomposition using CountSketch. *Advances in Computational Mathematics*, 46(6):1–28, 2020.

[58] P.-G. Martinsson and J. Tropp. Randomized numerical linear algebra: Foundations & algorithms. *arXiv preprint arXiv:2002.01387*, 2020.

[59] P. G. Martinsson and S. Voronin. A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices. *SIAM Journal on Scientific Computing*, 38(5):S485–S507, 2016.

[60] R. Minster, A. K. Saibaba, J. Kar, and A. Chakrabortty. Efficient randomized algorithms for subspace system identification. *arXiv preprint arXiv:2003.11872*, 2020.

[61] R. Minster, A. K. Saibaba, and M. E. Kilmer. Randomized algorithms for low-rank tensor decompositions in the Tucker format. *SIAM Journal on Mathematics of Data Science*, 2(1):189–215, 2020.

[62] Y. Nakatsukasa. Accuracy of singular vectors obtained by projection-based SVD methods. *BIT Numerical Mathematics*, 57(4):1137–1152, 2017.

[63] Oberwolfach Benchmark Collection. Steel profile. hosted at MORwiki – Model Order Reduction Wiki `http://modelreduction.org/index.php/Steel_Profile`, 2005.

[64] A. K. Saibaba. HOID: higher order interpolatory decomposition for tensors based on Tucker representation. *SIAM Journal on Matrix Analysis and Applications*, 37(3):1223–1249, 2016.

[65] A. K. Saibaba. Randomized subspace iteration: Analysis of canonical angles and unitarily invariant norms. *SIAM Journal on Matrix Analysis and Applications*, 40(1):23–48, 2019.

[66] S. A. Sauter and C. Schwab. Boundary element methods. In *Boundary Element Methods*, pages 183–287. Springer, 2010.

[67] J. Shetty and J. Adibi. The Enron email dataset database schema and brief statistical report. *Information sciences institute technical report, University of Southern California*, 4, 2004.

[68] H. D. Simon and H. Zha. Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. *SIAM Journal on Scientific Computing*, 21(6):2257–2274, 2000.

[69] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. FROSTT: The formidable repository of open sparse tensors and tools. `http://frostt.io/`, 2017.

[70] G. W. Stewart and J. G. Sun. *Matrix perturbation theory*. Computer Science and Scientific Computing. Academic Press, Inc., Boston, MA, 1990.

[71] Y. Sun, Y. Guo, C. Luo, J. Tropp, and M. Udell. Low-rank Tucker approximation of a tensor from streaming data. *SIAM Journal on Mathematics of Data Science*, 2(4):1123–1150, 2020.

[72] D. B. Szyld. The many proofs of an identity on the norm of oblique projections. *Numerical Algorithms*, 42(3-4):309–323, 2006.

[73] C. E. Tsourakakis. MACH: Fast randomized tensor decompositions. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 689–700. SIAM, 2010.

[74] P. Van Overschee and B. De Moor. *Subspace identification for linear systems: Theory—Implementation—Applications*. Springer Science & Business Media, 2012.

[75] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen. A new truncation strategy for the higher-order singular value decomposition. *SIAM Journal on Scientific Computing*, 34(2):A1027–A1052, 2012.

[76] M. A. O. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pages 447–460. Springer, 2002.

[77] M. Verhaegen and V. Verdult. *Filtering and system identification: a least squares approach*. Cambridge university press, 2007.

[78] N. Vervliet and L. De Lathauwer. A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):284–295, 2016.

[79] N. Vervliet, O. Debals, L. Sorber, M. Van Barel, and L. De Lathauwer. Tensorlab 3.0. `http://www.tensorlab.net`, 2016.

[80] S. Voronin and P.-G. Martinsson. RSVDPACK: An implementation of randomized algorithms for computing the singular value, interpolative, and CUR decompositions of matrices on multi-core and GPU architectures. *arXiv preprint arXiv:1502.05366*, 2015.

[81] T. Wu, V. M. Venkatasubramanian, and A. Pothen. Fast parallel stochastic subspace algorithms for large-scale ambient oscillation monitoring. *IEEE Transactions on Smart Grid*, 8(3):1494–1503, 2016.

[82] Z. Xu, L. Cambier, F.-H. Rouet, P. L'Eplatennier, Y. Huang, C. Ashcraft, and E. Darve. Low-rank kernel matrix approximation using skeletonized interpolation with endo-or exo-vertices. *arXiv preprint arXiv:1807.04787*, 2018.

[83] X. Ye, J. Xia, and L. Ying. Analytical low-rank compression via proxy point selection. *SIAM Journal on Matrix Analysis and Applications*, 41(3):1059–1085, 2020.

[84] L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626, 2004.

[85] D. Yu and S. Chakravorty. A computationally optimal randomized proper orthogonal decomposition technique. In *2016 American Control Conference (ACC)*, pages 3310–3315. IEEE, 2016.

[86] W. Yu, Y. Gu, and Y. Li. Efficient randomized algorithms for the fixed-precision low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 39(3):1339–1359, 2018.

[87] W. Yu, Y. Gu, and Y. Li. Efficient randomized algorithms for the fixed-precision low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 39(3):1339–1359, 2018.

[88] J. Zhang, A. K. Saibaba, M. E. Kilmer, and S. Aeron. A randomized tensor singular value decomposition based on the t-product. *Numerical Linear Algebra with Applications*, 25(5):e2179, 2018.

[89] G. Zhou, A. Cichocki, and S. Xie. Decomposition of big tensors with low multilinear rank. *arXiv preprint arXiv:1412.1885*, 2014.