Master's Thesis

Comparative analysis of hierarchical approximate singular value decompositions for low-rank approximation of large structured matrices

in partial fulfillment of the requirements for the degree of
Master of Science Scientific Computing

submitted by

Arya Prasetya
Matriculation Number : 0473184

21.07.2025

Technische Universität Berlin

First Supervisor & Reviewer: Prof. Dr. Christian Mehl
Second Reviewer: Dr. Michael Karow
Second Supervisor: Art Pelling

I hereby declare that the thesis submitted is my own, unaided work, completed without any external help. Only the sources and resources listed were used. All passages taken from the sources and aids used, either unchanged or paraphrased, have been marked as such.

Where generative AI tools were used, I have indicated the product name, manufacturer, the software version used, as well as the respective purpose (e.g. checking and improving language in the texts, systematic research). I am fully responsible for the selection, adoption, and all results of the AI-generated output I use.

I have taken note of the Principles for Ensuring Good Research Practice at TU Berlin dated 8 March 2017. https://www.tu.berlin/en/working-at-tu-berlin/important-documents/guidelinesdirectives/principles-for-ensuring-good-research-practice

I further declare that I have not submitted the thesis in the same or similar form to any other examination authority.

Berlin, 21.07.2025

. . . . . .

(Signature [Arya Prasetya])

Abstract

Low-rank approximations have become increasingly important in model order reduction, machine learning, and signal processing for reducing the size of high-dimensional data. While significant advancements exist in large-scale singular value decomposition (SVD) algorithms, tree-structured SVD methods, despite their potential for parallelization, remains limited. We address this by developing the hierarchical approximate singular value decomposition (HASVD) framework, building on the hierarchical approximate proper orthogonal decomposition (HAPOD) method, a tree-based POD technique [15]. We propose aggregation procedures for computing the SVD of partitioned matrix blocks, along with an error-prescribing mechanism to ensure global error tolerances. With the increasing application of Hankel matrices in eigensystem realization algorithm (ERA) and their unique structure, we also develop pruning techniques to leverage these properties [34]. Initial investigations into accuracy, rank truncation capability, and runtime performance are conducted on Hankel matrices. Results show that HASVD achieves computational accuracy comparable to established SVD algorithms such as LAPACK's GESDD, although singular value inflation can arise from using HASVD with a distributed two-level bidirectional tree (TLBD) structure. Additionally, we find that choosing appropriately sized block partitions is critical to exploiting the rank reduction potential of HASVD. Pruning also provides significant speedups in the computation of Hankel matrices. Finally, we demonstrate that HASVD yields results comparable to well-established methods such as randomized SVD [12] when applied to block-Hankel matrices.

Contents

Acronyms

DIST distributed tree

ERA eigensystem realization algorithm

FID free-induction decay

HAPOD hierarchical approximate proper orthogonal decomposition

HASVD hierarchical approximate singular value decomposition

HRTF head-related transfer function

INC incremental tree

LTI linear time-invariant

POD proper orthogonal decomposition

Rand-SVD randomized SVD

SSA-LRF single spectrum analysis linear recurrent formula

SSID subspace system identification

SVD singular value decomposition

TLAI two-level alternating incremental tree

TLBD distributed two-level bidirectional tree

TLBI incremental two-level bidirectional tree

Chapter 1

Introduction

In this chapter, the objectives and motivations of the thesis are introduced and formally stated. Following this, a summary of the main content is provided to offer an overview of the structure and scope of the work.

1.1 Motivation

The low-rank approximation is a technique used to approximate a matrix by another matrix of lower rank. More precisely, given a matrix $A \in \mathbb{R}^{m\times n}$, rank-$k$ approximation of matrix A, denoted by A_k, is defined as a matrix that solves the minimization problem,

$$\min_{\text{rank}(B)=k} \|A - B\|_2, \tag{1.1}$$

where $\|\cdot\|_2$ denotes the spectral norm (see Definition 2.1.1).

Low-rank approximation has become increasingly important in model order reduction, machine learning, and signal processing [18, 24]. It is primarily used to compress high-dimensional data into lower-dimensional representations, thereby reducing both computational cost and storage requirements.

Classically, the solution to the low-rank approximation problem is given by the **Eckart-Young theorem**. Let matrix A have a singular value decomposition (SVD) (see Theorem 2.1.1) given by $A = U\Sigma V^T$, where $U \in \mathbb{R}^{m\times m}$ and $V^T \in \mathbb{R}^{n\times n}$ are matrix of singular vectors of A and Σ is the diagonal matrix of singular values of A.

Theorem 1.1.1 (Eckhart-Young theorem [7]). *If $k < r = \text{rank}(A)$ and*

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \tag{1.2}$$

where u_i is i-th column of U, v_i^T is i-th row of V^T, and σ_i is the i-th diagonal element for $i = 1, \ldots, k$ then

$$\min_{\text{rank}(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}. \tag{1.3}$$

This not-only shows that the spectral error of the best rank-k approximation of A is bounded by σ_{k+1}, but also that the best approximation of A is obtained **a posteriori** by the computation of the SVD.

A commonly used algorithm to compute the SVD of a matrix is the Golub-Reinsch method, which employs Householder transformations and QR algorithm. The computational complexity of the Golub-Reinsch method is $\mathcal{O}(mn^2)$ [7]. While this algorithm provides a numerically accurate computation, the quadratic scaling with respect to the number of columns contributes to a high computational cost. As a point of reference, even relatively small matrix sizes arising in simple simulation models of heat equations can range from 6000×7000 to 18668×13642 [23].

A rough estimate of the memory required to store such matrices in double-precision floating-point format (e.g., in a NumPy array) shows that they require approximately 300 to 2000 MB of memory, respectively.

Given the high computational and storage cost of full singular value decomposition (SVD), there is a pressing need for more efficient algorithms that can provide a direct **a priori** low-rank approximation, where one can guarantee an approximation quality without having to access the entire data [22].

State-of-the-art algorithms for large-scale SVD often utilize randomized techniques to form low-rank approximations [39, 30]. However, there is growing interest in parallel algorithms, particularly those based on divide-and-conquer techniques that decompose the problem into smaller, independently solvable SVD problems [31].

Moreover, many matrices encountered in SVD-based applications exhibit inherent structure, most notably in the form of Hankel and Toeplitz matrices (see Definition 2.1.2) [10, 34]. These structured matrices possess symmetries that can be exploited to develop useful symmetries that provides the opportunity to develop memory saving memory-efficient, matrix-free algorithm for low-rank approximations.

A method of particular interest is the hierarchical approximate proper orthogonal decomposition (HAPOD), which employs a tree structure to aggregate multiple sets of vector data into so-called POD modes, offering promising results in terms of both accuracy and computational efficiency compared to conventional proper orthogonal decomposition (POD) methods [15]. Given the close relationship between POD and SVD computations, this raises the question of whether such a hierarchical approach can be adapted for general low-rank approximation tasks.

Motivated by these insights, this thesis aims to explore structured, parallel, and a priori low-rank approximation techniques, and to develop a hierarchical SVD framework. Its performance will be systematically compared with existing methods.

1.2 Objective

The HAPOD method primarily involves aggregating sets of vectors into orthogonal modes—interpretable as singular vectors—along with their associated singular values [15], indicating a strong connection to the SVD. However, due to the specific formulation of the POD method, the left singular vectors are typically not tracked or retained. In this context, the thesis aims to identify both the commonalities and

differences between POD and SVD computations, with the goal of establishing a foundation for developing a hierarchical SVD framework.

Building on this foundation, a hierarchical framework for SVD will be developed with the objective of retaining as much information from the original matrix as possible. To support this, theoretical error bounds for the hierarchical aggregation process must be derived.

Finally, the thesis will explore various tree structures. These ranges from simple topologies to those specifically designed to exploit the inherent structure of Hankel and Toeplitz matrices. A comparative analysis of these tree structures, along with an evaluation of their approximation quality, will be conducted through numerical experiments.

1.3 Outline

In Chapter 2, we will introduce the reader to basic definitions and notations used in this thesis, as well as review the POD method. Section 2.1 introduces key definitions from linear algebra [25], presents the structure and properties of Hankel and Toeplitz matrices following [19], and provides definitions and intuitive explanations of tree structures. It is then followed by an introduction to POD in Section 2.2, which mainly follows from [16]. The POD method is strictly defined for the discrete case. To establish a connection to low-rank approximations and the SVD, an approximate POD formulation is also introduced.

Chapter 3 develops the hierarchical framework for low-rank approximations using the singular value decomposition (SVD), referred to as hierarchical approximate singular value decomposition (HASVD). Section 3.1 begins with a brief overview of the HAPOD method from [15] and demonstrates how its aggregation step forms the foundation of HASVD. This is followed by Section 3.2, where the full HASVD framework is constructed, from the aggregation procedure to the computation of low-rank approximations. Finally, Section 3.3 presents a theoretical analysis of error bounds for approximations computed with HASVD and introduces an error prescription strategy to ensure that global tolerances are guaranteed.

Chapter 4 briefly discusses practical implementations of HASVD. Section 4.1 introduces common tree structures and their properties, while Section 4.2 presents pruning techniques that exploit the symmetries of Hankel matrices, such as skew-diagonal and transpose symmetry.

In Chapter 5, we conduct numerical experiments. Section 5.1 describes the experimental setup, including matrix generation and evaluation metrics. Section 5.2 presents results concerning the accuracy, rank truncation capabilities, and runtime performance of HASVD when applied to Hankel matrices. We conclude the chapter by applying HASVD to analyze the KEMAR head-related transfer function (HRTF) measurements [6], demonstrating its applicability to real-world datasets.

Finally, Chapter 6 summarizes the thesis and its main findings, particularly the performance and behavior of HASVD in the context of Hankel matrices. We also discuss potential directions for future research to further develop the framework

introduced in this work.

Chapter 2

Preliminaries

The aim of this chapter is to introduce the theoretical tools and notations that form the foundation of this thesis. Key concepts essential to the development of the HASVD framework will be presented.

Section 2.1 provides a brief review of fundamental linear algebra, including notational conventions, properties of matrix norms, and singular value decomposition (SVD). This section also introduces Hankel and Toeplitz matrices, highlighting their structural characteristics. Additionally, a concise overview of graph theory is provided to build intuition for the tree structures employed in the hierarchical SVD framework.

Following this, Section 2.2 introduces the proper orthogonal decomposition (POD), including its approximation properties and the method of snapshots. Connections between POD and linear algebra are explored, laying the groundwork for extending hierarchical approximate proper orthogonal decomposition (HAPOD) into the SVD framework.

2.1 Basic Definitions and Theorems

2.1.1 Linear Algebra

Throughout this thesis, vectors in the real k-dimensional space $x \in \mathbb{R}^k$ as well as real matrices $A \in \mathbb{R}^{m \times n}$ will be used.

If a vector $x \in \mathbb{R}^k$ is defined, it will be defined as a column vector. Thus its transpose x^T will be a row vector,

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix} \quad x^T = \begin{pmatrix} x_1 & x_2 & \dots & x_k \end{pmatrix}.$$

Similarly, the matrix $A \in \mathbb{R}^{m \times n}$ will be defined as (2.1) and its transpose as (2.2).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{2.1}$$

$$A^T = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & & a_{m2} \\ \vdots & & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{pmatrix} \tag{2.2}$$

The SVD is a matrix decomposition that can used to identify the dominant components of a matrix. It will also be core component to the HASVD.

Theorem 2.1.1 (Singular value decomposition (SVD)[25]). *Let $A \in \mathbb{R}^{m\times n}$ with $rank(A) = k$ be given. Then there exist unitary matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ and a diagonal matrix $\Sigma_+ = diag(\sigma_1, \dots, \sigma_k)$ with $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_k > 0$ such that*

$$A = U \begin{pmatrix} \Sigma_+ & 0_{k,n-k} \\ 0_{m-k,k} & 0_{m-k,n-k} \end{pmatrix} V^T. \tag{2.3}$$

*We call $\{\sigma_j\}_{j=1}^k$ the **singular values** of A, U the **matrix of left singular vectors** and V^T the **matrix of right singular vectors**.*

It should be noted that in Theorem 2.1.1, U contains left singular vectors along its columns, while V^T contains right singular vectors along its rows,

$$U = \begin{pmatrix} u_1 & u_2 & \dots & u_m \end{pmatrix} \text{ and } V^T = \begin{pmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{pmatrix}, \tag{2.4}$$

where $u_i \in \mathbb{R}^m$ is a left singular vector for $i = 1, \dots, m$ and $v_j \in \mathbb{R}^n$ is a right singular vector for $j = 1, \dots, n$.

One can observe that applying Theorem 2.1.1 to the matrix product A^TA results in a unitary decomposition, where the square of the singular values appear as eigenvalues, and the unitary matrix V as the matrix of eigenvectors. As a consequence, the columns of V corresponds to the eigenvectors, as shown in (2.5).

$$A^TA = V \begin{pmatrix} \Sigma_+^2 & 0_{k,n-k} \\ 0_{n-k,k} & 0_{n-k,n-k} \end{pmatrix} V^T \tag{2.5}$$

As a geometric interpretation, the singular value decomposition represents the action of a matrix on the unit sphere. It is spanned by its right singular vectors, and maps it to a hyper-ellipsoid whose principal semi-axes are the singular values oriented along the left singular vectors [44].

Hence, the right singular vectors that correspond to the zero eigenvalues form a basis of the nullspace of A^TA, while those that correspond to the non-zero eigenvalues

are part of the image of A^TA. Moreover, the magnitudes of the squared singular values indicate dominant eigenvectors. A similar characteristic apply to AA^T and the matrix of left singular vectors U as its eigenvector matrix.

This is why the singular value decomposition has an important role in low-rank approximation. It is able to identify eigenvectors that contribute to the range of a matrix and therewith, to the rank of matrix A. Hence it is also known as a rank-revealing method [7].

Consequently, SVD will be employed in the development of the hierarchical framework in Chapter 3 to reveal the ranks of a matrix and potentially reduce it to its most important components, as will be demonstrated.

By Theorem 2.1.1 and the orthonormality of the singular vectors, the decomposition of matrix A can be written in terms of the sum of the individual products between the singular values, left singular vectors, and right singular vectors.

Theorem 2.1.2. *[25] Let $A \in \mathbb{R}^{m\times n}$ with rank$(A) = k$ be given. Then it can be written as*

$$A = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \tag{2.6}$$

where u_i are the i-th column of the matrix of left singular vectors U and v_i^T are the i-th row of the matrix of right singular vectors V^T as in (2.4).

Using the form in (2.6) we can see that each singular value corresponds to a component of the matrix A, where zero singular values simply does not contribute to the form of the matrix and therefore, are neglectable when representing the matrix.

To perform error analysis of decompositions and concrete bounds for the approximate SVD computed from HASVD, matrix norms will be used.

Definition 2.1.1 (Norms [25])**.** *A function $||\cdot|| : V \to \mathbb{R}$ is called a **norm** on a vector space V when the following hold:*

i) $||v|| \geq 0$ *for all* $v \in V$ *with equality if and only if* $v = 0$,

ii) $||\alpha v|| = |\alpha| \cdot ||v||$ *for all scalars* α *and* $v \in V$,

iii) $||v_1 + v_2|| \leq ||v_1|| + ||v_2||$ *for all* $v_1, v_2 \in V$.

In particular, we will be using **matrix norms** on $\mathbb{R}^{m\times n}$, namely:

- the Frobenius norm,

$$\|A\|_F := \sqrt{\sum_{i,j=1}^{m,n} |a_{ij}|^2}, \tag{2.7}$$

- and the spectral norm,

$$\|A\|_2 := \sqrt{\lambda_{\max}(A^TA)}. \tag{2.8}$$

The **Frobenius norm** is defined by the sum squares of the modulus of all the elements of a matrix, while the **spectral norm** is the maximum eigenvalue of the matrix product A^TA. Therefore, due to Theorem 2.1.1, we also know that the matrix norms can be written in terms of singular values. If a matrix $A \in \mathbb{R}^{m\times n}$ has rank k, then

$$||A||_2 = \sigma_1 \text{ and } ||A||_F = \sqrt{\sum_{i=1}^{k} \sigma_i^2}. \tag{2.9}$$

Remark 2.1.1. *Let $A \in \mathbb{R}^{m\times n}$ be a block matrix with blocks $B^{(i)} \in \mathbb{R}^{m_i\times n_i}$ for $i = 1, \ldots, jk$ such that,*

$$A = \begin{pmatrix} B^{(1)} & \cdots & B^{(k)} \\ \vdots & \ddots & \vdots \\ B^{((j-1)k+1)} & \cdots & B^{(jk)} \end{pmatrix}$$

$$= \begin{pmatrix} b_{11}^{(1)} & \cdots & b_{1n_1}^{(1)} & & b_{11}^{(k)} & \cdots & b_{1n_1}^{(k)} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ b_{m_11}^{(1)} & \cdots & b_{m_1n_1}^{(1)} & & b_{m_11}^{(k)} & \cdots & b_{m_1n_1}^{(k)} \\ & \vdots & & \ddots & & \vdots & \\ b_{11}^{((j-1)k+1)} & \cdots & b_{1n_1}^{((j-1)k+1)} & & b_{11}^{(jk)} & \cdots & b_{1n_1}^{(jk)} \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ b_{m_11}^{((j-1)k+1)} & \cdots & b_{m_1n_1}^{((j-1)k+1)} & & b_{m_11}^{(jk)} & \cdots & b_{m_1n_1}^{(jk)} \end{pmatrix}.$$

By (2.7) the Frobenius norm of matrix A is

$$\begin{aligned} ||A||_F^2 &= \sum_{i=1}^{jk} \sum_{l,p=1}^{m_i,n_i} |b_{lp}^{(i)}|^2 \\ &= \sum_{i=1}^{jk} ||B^{(i)}||_F^2. \end{aligned} \tag{2.10}$$

Remark 2.1.1 is going to be used extensively in the error bound analysis of HASVD where block matrices are formed to perform hierarchical tasks of SVD.

2.1.2 Hankel and Toeplitz matrices

We will specifically study singular value decomposition under the specific matrix structure of Hankel and Toeplitz matrices.

Definition 2.1.2 (Hankel and Toeplitz matrix [19])**.** *Let $H \in \mathbb{R}^{m\times n}$ be a matrix. If the elements of H fulfill:*

*i) $(H)_{ij} = (H)_{i'j'}$ for all $i + j = i' + j'$, then H is called a **Hankel** or **catalecticant matrix**,*

ii) $(H)_{ij} = (H)_{i'j'}$ *for all* $i-j = i'-j'$*, then* H *is called a* ***Toeplitz*** *or* ***diagonal-constant matrix****.*

Definition 2.1.2 implies that the elements of a Hankel matrix is constant along its skew-diagonals, while the elements of a Teoplitz matrix is constant along its diagonals.

It should be noted that in [19], the definitions of Hankel and Toeplitz matrices are given only for the square case. However, modern applications—particularly in system-approximation and model-order reduction—routinely employ rectangular Hankel and Toeplitz matrices. Consequently, several authors have extended the original square-matrix definitions to the rectangular setting (For example, see [40]).

Hankel and Toeplitz matrices are interesting due to its structure as it can be defined entirely by a row and a column. As a consequence, both matrices can be defined by $n+m-1$ terms. This is shown in (2.11) for the Hankel matrix and in (2.12) for the Toeplitz matrix.

$$\text{Hankel}_{mn}(a_1, a_2, \ldots, a_{n+m-1}) := \begin{pmatrix} a_1 & a_2 & a_3 & \ldots & a_n \\ a_2 & a_3 & & & a_{n+1} \\ a_3 & & \ddots & & a_{n+2} \\ \vdots & & & & \vdots \\ a_m & & \ldots & & a_{n+m-1} \end{pmatrix} \tag{2.11}$$

$$\text{Toeplitz}_{mn}(a_{m-1}, a_{m-2}, \ldots, a_0, \ldots, a_{1-n}) := \begin{pmatrix} a_0 & a_{-1} & a_{-2} & \ldots & a_{1-n} \\ a_1 & a_0 & a_{-1} & & \\ a_2 & a_1 & a_0 & & \vdots \\ \vdots & & & \ddots & a_{-1} \\ a_{m-1} & & \ldots & a_1 & a_0 \end{pmatrix} \tag{2.12}$$

By having this structure, the amount numbers that are needed to be stored, to construct a Hankel or Toeplitz matrix is significantly smaller than a general matrix, only requiring a sequence of $n+m-1$ elements. Therefore, one can have a matrix-free represenation of Hankel and Toeplitz matrices.

Specifically for square Hankel and Toeplitz matrix, there are further properties that can be observed.

Remark 2.1.2 ([19])**.** *Let* $H \in \mathbb{R}^{n\times n}$ *be a square matrix and* $J_n \in \mathbb{R}^{n\times n}$ *be an* ***exchange (backward identity) matrix****,*

$$J_n = \underbrace{\begin{pmatrix} 0 & 0 & \ldots & 0 & 1 \\ 0 & 0 & & 1 & 0 \\ \vdots & & \iddots & & \vdots \\ 0 & 1 & & 0 & 0 \\ 1 & 0 & \ldots & 0 & 0 \end{pmatrix}}_{n\times n}.$$

.

i) If H is a Hankel matrix, it is symmetric.

ii) If H is a Hankel matrix, then $T = HJ_n$ is a Toeplitz matrix.

iii) If H is a Hankel (Toeplitz) matrix, every square block is also a Hankel (Toeplitz) matrix.

The transformation in Remark 2.1.2 can be shown by noting that elements of the exchange matrix is $(J_n)_{i,j} = \delta_{i,n+1-j}$. For a Hankel matrix H, the product HJ_n gives

$$(HJ_n)_{i,k} = \sum_{j=1}^{n}(H)_{i,j}(J_n)_{j,k} = \sum_{j=1}^{n}(H)_{i,j}\delta_{j,n+1-k} = (H)_{i,n+1-k} \tag{2.13}$$

By Definition 2.1.2, we know that for elements of Hankel matrix H, the elements of a Hankel matrix H have values that depend only on $i + k$ for row i and column k,

$$(H)_{i,k} = h_{i+k} = h_{i'+k'} = (H)_{i',k'}. \tag{2.14}$$

When (2.14) is substituted into (2.13), we obtain

$$(HJ_n)_{i,k} = (H)_{i,n+1-k} = h_{i-k+n+1} = h_{i'-k'+n+1} = (H)_{i',n+1-k'} = (HJ_n)_{i',k'}. \tag{2.15}$$

By Definition 2.1.2, we observe that (2.15) describes a Toeplitz matrix, as the elements depend only on the difference $i - k$.

In addition to this, the transformation can also be extended to rectangular matrices, where we use two exchange matrices.

Remark 2.1.3 ([14]). *Let $H \in \mathbb{R}^{m\times n}$ be a rectangular Hankel matrix. Let $J_m \in \mathbb{R}^{m\times m}$ and $J_n \in \mathbb{R}^{n\times n}$ be exchange matrices. Then*

$$T := J_m H J_n \tag{2.16}$$

is a rectangular Toeplitz matrix.

From Remark 2.1.3, it is concluded that a Hankel matrix can be trasformed to a Toeplitz matrix by (2.16) or vice versa, since the exchange matrix is involutory, i.e. $J_n^2 = I_n$.

One can perform extension of a Hankel or Toeplitz matrix by adding i elements to extend i rows and j elements to extend j columns.

As an example, suppose a Hankel matrix $H \in \mathbb{R}^{m\times n}$ is given, where the matrix is defined by a sequence $(a_1, a_2, \ldots, a_{n+m-1})$, i.e. $H = \text{Hankel}_{mn}(a_1, a_2, \ldots, a_{n+m-1})$.

Suppose the columns of matrix H is extended to matrix $H' \in \mathbb{R}^{m\times(n+1)}$, where it takes the form

$$H' = \left(\begin{array}{c|c} H & \begin{matrix} a_{n+1} \\ \vdots \\ \mathbf{a_{n+m}} \end{matrix} \end{array}\right).$$

This can be done by extending the sequence defining the matrix H', where

$$H' = \text{Hankel}_{m,(n+1)}(a_1, a_2, \ldots, a_{n+m-1}, \mathbf{a_{n+m}}).$$

Similarly, if the rows of matrix H were to be extended to matrix $H'' \in \mathbb{R}^{(m+1)\times n}$,

$$H'' = \left(\begin{array}{ccc} & H & \\ \hline a_{m+1} & \cdots & \mathbf{a_{n+m}} \end{array}\right).$$

It also only requires the extension of the sequence defining the matrix H'' with the a different shape, where

$$H'' = \text{Hankel}_{(m+1),n}(a_1, a_2, \ldots, a_{n+m-1}, \mathbf{a_{n+m}}).$$

If an extensions along both rows and columns from matrix H to matrix $H''' \in \mathbb{R}^{(m+1)\times(n+1)}$ is needed, where H''' has the form

$$H''' = \left(\begin{array}{ccc|c} & & & a_{n+1} \\ & H & & \vdots \\ & & & \mathbf{a_{n+m}} \\ \hline a_{m+1} & \cdots & \mathbf{a_{n+m}} & \mathbf{a_{n+m+1}} \end{array}\right),$$

the matrix H''' can be formed by the extending the sequence with two elements, where

$$H''' = \text{Hankel}_{(m+1),(n+1)}(a_1, a_2, \ldots, a_{n+m-1}, \mathbf{a_{n+m}}, \mathbf{a_{n+m+1}}).$$

The steps above can be done iteratively for some Hankel matrix $\tilde{H} \in \mathbb{R}^{(m+i)\times(n+j)}$ as shown in the following Remark 2.1.4.

Remark 2.1.4 (Hankel matrix extension [19])**.** *Let $H \in \mathbb{R}^{m\times n}$ where*

$$H = Hankel_{mn}(a_1, a_2, \ldots, a_{n+m-1}).$$

Then there exists some $\tilde{H} \in \mathbb{R}^{(m+i)\times(n+j)}$ where

$$\tilde{H} = Hankel_{m+i,n+j}(a_1, a_2, \ldots, a_{n+m-1}, \mathbf{a_{n+m}}, \ldots, \mathbf{a_{n+m+i+j-1}}),$$

and takes the form

$$H_{m+i,n+j} = \left(\begin{array}{ccccc|ccc} & & & & & a_{n+1} & \cdots & a_{n+j} \\ & & & & & & & \vdots \\ & & H_{m,n} & & & \vdots & & \mathbf{a_{n+m}} \\ & & & & & & \iddots & \vdots \\ & & & & & \mathbf{a_{n+m}} & \cdots & \mathbf{a_{n+m+j-1}} \\ \hline a_{m+1} & & \cdots & & \mathbf{a_{n+m}} & \mathbf{a_{n+m+1}} & & \\ \vdots & & & \iddots & \vdots & & \ddots & \\ a_{m+i} & \cdots & \mathbf{a_{n+m}} & \cdots & \mathbf{a_{n+m+j-1}} & & & \mathbf{a_{n+m+i+j-1}} \end{array}\right).$$

Analogously, an extension of a Toeplitz matrix can be formed by inserting sequence of $i+j$ elements, but this will be beyond the scope of this thesis, as one can already transform Toeplitz to Hankel matrices by Remark 2.1.3.

Finally, we also consider block-Hankel and Toeplitz matrices, which is an extension of Definition 2.1.2.

Definition 2.1.3 (block-Hankel and -Toeplitz matrices [32]). *Let $H \in \mathbb{R}^{Mm\times Nn}$ be a block matrix with blocks $A_{ij} \in \mathbb{R}^{m\times n}$ for block row positions $i = 1,\dots,M$ and block column positions $j = 1,\dots,N$. If the blocks of H fulfill:*

*i) $A_{ij} = A_{i'j'}$ for all $i+j = i'+j'$, then H is called a **block-Hankel matrix**,*

*ii) $A_{ij} = A_{i'j'}$ for all $i-j = i'-j'$, then H is called a **block-Toeplitz matrix**.*

Note that block-Hankel matrices may not be Hankel matrix, since the blocks are general matrices, and hence breaks the anti-diagonal symmetry in the scalar level. Nevertheless the SVD of this matrix is the core of many subspace system identification (SSID) methods in model order reduction [34].

With this, a map of sequences of matrices can also be defined for these block matrices, where from matrices $A_1, A_2, \dots, A_{M+N-1} \in \mathbb{R}^{m\times n}$ a block-Hankel matrix can be formed by (2.17) and block-Toeplitz matrix by (2.18).

$$\mathrm{HB}_{MN}(A_1, A_2, \dots, A_{N+m-1}) := \begin{pmatrix} A_1 & A_2 & A_3 & \dots & A_N \\ A_2 & A_3 & & & A_{N+1} \\ A_3 & & \ddots & & A_{N+2} \\ \vdots & & & & \vdots \\ A_M & & \dots & & A_{M+N-1} \end{pmatrix} \tag{2.17}$$

$$\mathrm{TB}_{MN}(A_{M-1}, A_{M-2}, \dots, A_0, \dots, A_{1-n}) := \begin{pmatrix} A_0 & A_{-1} & A_{-2} & \dots & A_{1-n} \\ A_1 & A_0 & A_{-1} & & \\ A_2 & A_1 & A_0 & & \vdots \\ \vdots & & & \ddots & A_{-1} \\ A_{M-1} & & \dots & A_1 & A_0 \end{pmatrix} \tag{2.18}$$

2.1.3 Graph Theory

In addition to linear algebra, the use of graphs, namely rooted tree, will be crucial in defining hierarchical structures in HASVD.

Definition 2.1.4 (Rooted tree [15]). *Let $\mathcal{N}_T$ be a finite set, with $\rho_T \in \mathcal{N}_T$ and map $\mathcal{C}_T : \mathcal{N}_T \to \mathbb{P}(\mathcal{N}_T \setminus \{\rho_T\})$. A **rooted tree** is then a triple $T = (\mathcal{N}_T, \mathcal{C}_T, \rho_T)$ where $\mathcal{C}_T$ satisfy:*

i) $\forall \alpha, \beta \in \mathcal{N}_T : \alpha \neq \beta \implies \mathcal{C}_T(\alpha) \cap \mathcal{C}_T(\beta) = \emptyset$,

ii) $\forall \emptyset \neq X \subseteq \mathcal{N}_T \setminus \{\rho_T\}, \exists \alpha \in \mathcal{N}_T \setminus X : \mathcal{C}_T(\alpha) \cap X \neq \emptyset$, *where* ρ_T *is called the* ***root node****.*

The largest distance of the root node to the leaf node is the ***depth*** d_T *of* T*.*

For a node $\alpha \in \mathcal{N}_T$ we call the set $\mathcal{C}_T(\alpha)$, the **children nodes** of α.

In Definition 2.1.4, *i)* implies that each node is uniquely the child of one node and a particular node will not share a common child. Furthermore, *ii)* implies that there exist a parent node for subset elements and that it is connected to root node ρ_T.

Due to this one can derive some definitions of a given rooted tree T:

- The **leaf nodes** is set of nodes defined by

$$\mathcal{L}_T := \{\alpha \in \mathcal{N}_T : \mathcal{C}_T(\alpha) = \emptyset\}. \tag{2.19}$$

  This means leaf nodes consist of all the nodes that do not have any children.

- **Lower level nodes** of α is a set of nodes recursively defined by

$$\mathcal{N}_T(\alpha) := \{\alpha\} \cup \bigcup_{\beta \in \mathcal{C}_T(\alpha)} \mathcal{N}_T(\beta). \tag{2.20}$$

  This set consists of node α and the nodes contained in the lower level set of its children.

- **Parent node** $\mathcal{P}_T(\alpha)$ is defined by,

$$\mathcal{P}_T(\alpha) := \beta, \text{ such that } \alpha \in \mathcal{C}(\beta). \tag{2.21}$$

  Note that $\mathcal{P}_T(\alpha)$ is not a set of nodes, but rather a specific node associated to a member of a child node.

- In a given rooted tree T, for non-leaf nodes that is not a root node $(\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}$, the following identity holds true,

$$(\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\} = \{\alpha \in \mathcal{C}_T(\alpha') \setminus \mathcal{L}_T : \alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T\}. \tag{2.22}$$

  The identity in (2.22) states that the sef of non-leaf nodes, excluding the root node, are equivalent to the set of all nodes α which are non-leaf children of non-leaf nodes α', including the root node.

A clear visualization of sets in a rooted tree can also be seen in Figure 2.1.

Since tree traversal is required in both the HAPOD and HASVD algorithms, we define the Traverse() function function in Algorithm 1 to explore a node $\alpha \in \mathcal{N}_T$. This algorithm mirrors the functionality of the `traverse()` routine implemented in the HAPOD module of pyMOR [29]. For completeness, we also provide a brief explanation of its fundamental yet useful role.

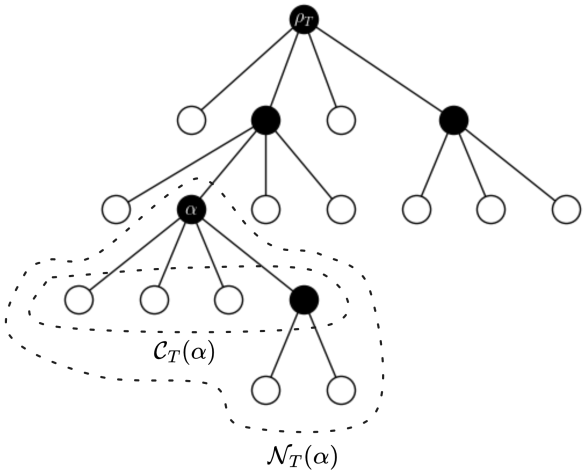


Figure 2.1: A rooted tree T with children ($\mathcal{C}_T(\alpha)$) and lower level nodes ($\mathcal{N}_T(\alpha)$) of node α indicated. White nodes indicate leaf nodes $\mathcal{L}_T$ and black nodes indicate non-leaf nodes $\mathcal{N}_T \setminus \mathcal{L}_T$.

Algorithm 1: Traverse() function

Input: Node $\alpha \in \mathcal{N}_T$ and explored set of nodes $\mathcal{S}$. *Note: α should not be in $\mathcal{S}$*
Output: Node $\beta \in \mathcal{N}_T$ or no node at all, and explored set of nodes $\mathcal{S}$.

```
if node α ∈ S then
    β :=None
    return β, S
for child α' ∈ C_T(α) do
    if α' ∉ S then
        β, S := Traverse(α', S)
        return β, S
β := α
S := {α} ∪ S
return β, S
```

Given a set of explored node $\mathcal{S}$, the function Traverse() traverses the tree and returns an unexplored node β, while also adding β to the set of explored nodes $\mathcal{S}$. Note that with the branching logic defined in the Traverse(), child nodes are given precedence over their parent nodes. As a result, a parent node will never be returned if any of its children nodes remain unexplored. This traversal order is crucial for correctly executing subtask in both HAPOD and HASVD.

These definitions, identities, and algorithms provide the foundational intuition required for understanding the notational structure of HASVD, and they will be essential in proving the validity of the HASVD error bounds and approximation guarantees discussed in Section 3.3.

2.2 Proper orthogonal decomposition

Proper orthogonal decomposition (POD) is a method that emerged in the study of turbulence and fluids. The main goal of POD is to extract basis from experimental data of system which usually has high dimensionality. By doing this, datas are decomposed into a representation of orthogonal basis which is optimal to the system, hence the name.

POD serves as the foundation for the hierarchical approximate proper orthogonal decomposition (HAPOD) method, which subsequently inspires the development of the hierarchical approximate singular value decomposition (HASVD) framework peresented herein.

While the method is synonymous in large part with Principal Component Analysis, Karhunen-Loeve Decomposition, and SVD throughout literatures [4], distinction should be made between POD and SVD so that we can be as precise as possible. Therefore this section will introduce the fundamentals of POD and its discretization for application in Linear Algebra.

2.2.1 POD basis

In this subsection, we begin by giving a concrete example of POD followed by a function analytic construction of POD as described in [16]. This description will be developed for application in matrices $A \in \mathbb{R}^n$.

As an introductory example, suppose that there is a set of real scalar fields $u_i : [0,1] \to \mathbb{R}$ for $i = 1, \dots, n$. We would like to seek an appropriate representation of members of $\{u_i(x)\}_{i=1}^n$. This can be done by projecting each member with a set of candidate basis functions $\varphi_j : [0,1] \to \mathbb{R}$ for $j \in \mathbb{N}$, leading to the basis representation

$$u_i(x) = \sum_{j=1}^{\infty} a_{ij}\varphi_j(x),$$

where $\{\varphi_j\}_{j=1}^{\infty}$ is the basis that describes each member.

This is done by assuming that $\{u_i\}_{i=1}^n$ belongs to the inner product space $L^2([0,1])$ and projecting a basis within that space. However, the choice of the basis is not unique as we can use different representation such as Fourier series and Chebyshev polynomials to obtain a basis representation, up to a domain shift [4]. This is where POD specifies an optimal choice of basis.

As mentioned earlier, POD aims to find the optimal basis in the system. By optimality, we choose $\{\varphi_i\}_{j=1}^{\infty}$ in such a way that the basis maximize the averaged projection of u_i onto φ_j, where the basis is suitably normalized. More precisely, it is formulated as an optimization problem in (2.23).

$$\max_{\varphi_j \in L^2([0,1])} \frac{\overline{|\langle u_i, \varphi_j \rangle|^2}}{||\varphi_j||^2}, ||\varphi_j|| = 1 \quad \text{for } j \in \mathbb{N} \tag{2.23}$$

In (2.23), $\overline{f(u_i)} := \frac{1}{n}\sum_{i=1}^n f(u_i)$ is the average over the entire set of the scalar fields.

It has been shown in [16] that optimization problem in (2.23) can be solved using calculus of variation using a constrained variational problem,

$$J[\varphi_j] = \overline{|\langle u_i, \varphi_j \rangle|^2} - \lambda(||\varphi_j||^2 - 1).$$

Theorem 2.2.1 (Optimality condition of POD basis [16]). *Let $\{u_i\}_{i=1}^n \subset L^2(\Omega)$ be a set of real function over a closed interval $\Omega \subset \mathbb{R}$. The optimality condition for the optimization problem,*

$$\max_{\varphi \in L^2(\Omega)} \frac{\overline{|\langle u_i, \varphi \rangle|^2}}{||\varphi||^2}, ||\varphi|| = 1$$
$$\text{for } i = 1 \dots n, j \in \mathbb{N},$$

is given by

$$\int_\Omega \overline{u_i(x)u_i(x')}\varphi(x')dx' = \lambda\varphi(x).$$

The optimality condition given for the POD basis can be expanded by the averaging sum, resulting in (2.24).

$$\frac{1}{n}\sum_{i=1}^{n} \int_\Omega u_i(x)u_i(x')\varphi_j(x')dx' = \frac{1}{n}\sum_{i=1}^{n} u_i(x) \int_\Omega u_i(x')\varphi_j(x')dx' = \lambda\varphi(x). \tag{2.24}$$

Now suppose that, instead of a continuous function over the domain, we have a discrete set of values from gridpoints $x_1, x_2, \dots, x_m$. Then, we can give a formal definition of POD for vector sets in finite-dimensional real space, which we will use throughout this thesis.

Definition 2.2.1 (Proper orthogonal decomposition (POD)). *Let $\{u_i\}_{i=1}^n$ be a finite set of vectors in $\mathbb{R}^m$. A **proper orthogonal decomposition (POD)** of $\{u_i\}_{i=1}^n$ is the decomposition of each member into a linear combination of vector set $\{\varphi_j\}_{j=1}^{\min(\{m,n\})}$*

$$u_i = \sum_{j=1}^{\min(\{m,n\})} a_{ij}\varphi_j, \tag{2.25}$$

fulfilling the optimization problem

$$\begin{aligned} &\underset{\varphi_j \in \mathbb{R}^m}{\text{maximize}} && \frac{\overline{|\langle u_i, \varphi_j \rangle|^2}}{||\varphi_j||^2} \\ &\text{subject to} && ||\varphi_j|| = 1, \end{aligned}$$

*where $a_j \in \mathbb{R}$ is coefficient of the linear combination and $\varphi_j \in \mathbb{R}^m$ is called a **POD mode**.*

Using these gridpoints, we turn (2.24) into a summation of discrete values

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{m} u_i(x_l)u_i(x_k)\varphi_j(x_k) = \lambda\varphi_j(x_l),\ l = 1 \dots m. \tag{2.26}$$

This leads to a useful interpretation of the optimality condition in (2.23), particularly in a linear algebraic sense. By transforming $n\lambda \to \lambda_j$ to some respective POD mode φ_j, we observe that (2.26) is simply a formulation of the eigenvalue problem,

$$\overbrace{\begin{pmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_n\end{pmatrix}}^{A} \underbrace{\begin{pmatrix}\mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_n^T\end{pmatrix}}_{A^T} \overbrace{\begin{pmatrix}\varphi_j(x_1) \\ \varphi_j(x_2) \\ \vdots \\ \varphi_j(x_m)\end{pmatrix}}^{\varphi_j} = \lambda_j \varphi_j,$$

where $A \in \mathbb{R}^{m\times n}$ is a matrix and $\mathbf{u}_i$ are vectors of values $\mathbf{u}_i = (u_i(x_1), u_i(x_2), \dots, u_i(x_m))^T$. Therefore, the basis that follows the optimality condition for POD can be found by solving the eigenvalue problem of a left Gramian matrix AA^T,

$$AA^T\varphi_j = \lambda_j\varphi_j. \tag{2.27}$$

Since AA^T is symmetric, we can write the operator representing the left Gramian matrix as an eigendecomposition [17]

$$AA^T = \sum_{i=1}^{m} \lambda_i \varphi_i \varphi_i^T.$$

By inserting the decomposition in (2.25) of member function u_i in terms of POD modes in finite dimensional space in (2.26), averaging for elements dependent on averaging index i can be isolated,

$$\begin{aligned}\sum_{k=1}^{m} \overline{u_i(x_l)u_i(x_k)}\varphi_j(x_k) &= \sum_{k=1}^{m} \overline{\sum_{s=1}^{m} a_{is}\varphi_s(x_l) \sum_{t=1}^{m} a_{it}\varphi_t(x_k)}\varphi_j(x_k) \\ &= \sum_{k=1}^{m}\sum_{s=1}^{m}\sum_{t=1}^{m} \varphi_s(x_l)\varphi_t(x_k)\overline{a_{is}a_{it}}\varphi_j(x_k) \\ &\overset{!}{=} \sum_{k=1}^{m}\sum_{s=1}^{m} \lambda_s\varphi_s(x_l)\varphi_s(x_k)\varphi_j(x_k) = \lambda_j\varphi_j(x_l).\end{aligned}$$

Comparing the coefficients one then obtain,

$$\overline{a_{ij}a_{ik}} = \lambda_k\delta_{jk} \quad \text{for } j,k = 1\dots m,$$

which gives us a set of stronger condition in POD modes in finite-dimensional space.

Theorem 2.2.2 (POD mode condition)**.** *Let $\{u_i\}_{i=1}^n$ be a finite set of vectors in $\mathbb{R}^m$ and $A \in \mathbb{R}^{m\times n}$ be a matrix, such that*

$$A = \begin{pmatrix}u_1 & u_2 & \dots & u_n\end{pmatrix}.$$

Then $\{\varphi_j\}_{j=1}^m$ is the set of POD modes of $\{u_i\}_{i=1}^n$ if:

i) $\{\varphi_j\}_{j=1}^m$ *is the set of eigenvectors for the eigenproblem,* $AA^T\varphi = \lambda\varphi$,

ii) $u_i = \sum_{j=1}^m a_{ij}\varphi_j$ *such that* $\overline{a_{ij}a_{ik}} = \lambda_k\delta_{jk}$ *for* $j, k = 1 \dots m$.

Condition *ii)* in Theorem 2.2.2 also states that the magnitude of λ_k reflects the size of the respective coefficient of φ_k in the linear combinations to represent the set of vectors $\{u_i\}_{i=1}^n$. This remark will be particularly important when dealing with approximate PODs in the next subsection.

It is important to note that POD's applicability extends beyond discrete gridpoints, but also for continous function. However, the scope of this thesis will be limited to discrete points and will refer the reader to Berkooz' treatment of POD for further reading [2].

2.2.2 Approximation of POD

Subsection 2.2.1 have established that the optimality condition for POD can be satisfied by the solution of an eigenvalue problem of matrix $AA^T \in \mathbb{R}^{m\times m}$. In this subsection, we will explore the connection between POD and the SVD, and introduce the concept of approximated POD.

Now, recall that by Theorem 2.1.1, the SVD for matrix $A \in \mathbb{R}^{m\times n}$ is $A = U\Sigma V^T$, where $U \in \mathbb{R}^{m\times m}$, $V^T \in \mathbb{R}^{n\times n}$, and $\Sigma \in \mathbb{R}^{m\times n}$ are the matrix of left singular vectors, matrix of right singular vectors, and singular value matrix respectively. Inserting the singular value decomposition into (2.27), we obtain

$$AA^T\varphi_j = U\Sigma V^T V\Sigma^T U^T\varphi_j = U\Sigma_m^2 U^T\varphi = \lambda_j\varphi_j.$$

In this problem, we are looking to solve an eigenvalue problem of matrix $U\Sigma_+^2 U^T$. Since U is unitary and $\Sigma_m^2 = \text{diag}(\sigma_1^2, \dots, \sigma_k^2, \underbrace{0, \dots, 0}_{m-k})$, we can deduce that the eigenvector φ_j corresponds to the the columns of the left singular vector matrix $U = (\varphi_1, \varphi_2, \dots, \varphi_m)$, with the eigenvalue λ_j corresponding to the square of singular values σ_j^2 for a given POD mode.

In Subsection 2.2.1, it was demonstrated that the modulus of eigenvalue λ_j represents the magnitudes of dominant POD modes. A similar interpretation applies for the singular values σ_j. Notably, the nomenclature used in POD methods is primarily derived from the SVD framework rather than from eigenvalue problems, a preference that is well justified.

As a comparison, the computation of the SVD of $A \in \mathbb{R}^{m\times n}$ requires memory to store $\mathcal{O}(mn)$ elements of A and involves $\mathcal{O}(mn^2)$ flops. Meanwhile, treating this as an eigenvalue problem necessitates the storage of $\mathcal{O}(mn + n^2)$ elements to store A and $Q^T AA^T Q$ respectively. The computational cost includes $\mathcal{O}(mn^2)$ flops for QR factorization of A and an additional $\mathcal{O}(n^3)$ flops for solving the eigenvalue problem (via QR algorithm) [44]. This highlights why SVD is generally preferred for computing POD modes, as it avoids forming the covariance matrix explicitly and reduces both storage and computational complexity.

As discussed in Subsection 2.1.1, the SVD forms the basis of the low-rank approximations. This naturally transfers to the approximated POD, where dominant POD modes are retained up to a given tolerance, leading to an approximate POD.

Theorem 2.2.3 (Approximate POD)**.** *Let $\{u_i\}_{i=1}^k, \{\varphi_j\}_{j=1}^l \subset \mathbb{R}^m$ be a finite set of vectors and its POD basis respectively, with its associated singular values $\{\sigma_j\}_{j=1}^l \subset \mathbb{R}$ ordered such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_l \geq 0$. For a given tolerance $\epsilon > 0$, the* ***approximate POD*** *of $\{u_i\}_{i=1}^k$ is then*

$$\tilde{u}_i = \sum_{j=1}^{n} a_{ij}\varphi_j,$$

where $n \leq l$ is chosen such that the total 2-norm approximation error of the vectors

$$\sum_{i=1}^{k} ||u_i - \tilde{u}_i||_2^2 = \sum_{i=n+1}^{l} \sigma_i^2 \leq \epsilon^2. \tag{2.28}$$

Proof. The approximation error of Theorem 2.2.3 follows directly from Eckhardt-Young Mirsky Theorem. □

To enable concise notation in subsequent discussions and algorithms, we introduce a shorthand for the approximate POD operation. Given a set of vectors $\{u\} \in \mathbb{R}^m$, we denote the corresponding set of POD mode-singular value pairs $\{(\varphi, \sigma)\} \subset (\mathbb{R}^m, \mathbb{R})$, computed with an error tolerance $\epsilon > 0$ as,

$$\{(\varphi, \sigma)\} =: \mathrm{APOD}(\{u\}, \epsilon). \tag{2.29}$$

2.2.3 Method of snapshots

We now introduce a POD method that may be useful for memory-limited computations and is particularly relevant in hierarchical POD. This approach, known as the *method of snapshots*, was introduced by Sirovich in [41].

Algorithm 2: Method of snapshots

Input: Set of vectors $\{u_i\}_{i=1}^n \subset \mathbb{R}^m$.
Output: Set of POD mode and singular value pairs $\{(\varphi_j, \sigma_j)\}_{j=1}^k \subset (\mathbb{R}^m, \mathbb{R})$.
begin
- $A \in \mathbb{R}^{m\times n} \leftarrow$ Concatenate set of vectors $\{u_i\}_{i=1}^n$;
- Form Gramian matrix product $A^TA \in \mathbb{R}^{n\times n}$;
- Solve eigenproblem $A^TA = V\Lambda V^T$;
- $\tilde{\Lambda} := \mathrm{diag}(\lambda_1, \ldots, \lambda_k) \leftarrow \Lambda$ and and $\tilde{V} := (v_1, \ldots, v_k) \in \mathbb{R}^{n\times k} \leftarrow V$, truncate $n - k$ eigenpairs in $\ker(A^TA)$;
- $(\varphi_1, \varphi_2, \ldots, \varphi_k) \leftarrow A\tilde{V}\tilde{\Lambda}^{-1}$, Compute POD modes;
- $\sigma_i \leftarrow \sqrt{\lambda_i}$ for $i = 1 \ldots k$, Compute singular values for $i = 1, \ldots, k$;

Similar to the approach used in (2.27), the POD method involves solving an eigenproblem of the Gramian matrix $A^T A$. The advantage of this approach is that it operates on an $n \times n$ matrix rather than an $m \times m$ matrix. Inserting the SVD of $A = U\Sigma V^T$, we obtain

$$A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^T U^T U\Sigma V^T = V\Sigma_n^2 V^T.$$

While computing the POD modes as an eigenvalue problem is generally avoided, the method of snapshots provides an exception due to its memory efficiency. As opposed to requiring storage of $\mathcal{O}(mn)$ elements to store SVDs, method of snapshots require only $\mathcal{O}(n^2)$ elements to store $A^T A$. This implies that method of snapshots is bound to gain a memory advantage when $m \gg n$, i.e. A is a tall-skinny matrix.

Moreover, it is important to note that the potential to save memory does not only come from the storage of $A^T A$, but also form the computation of U. Suppose A is very large matrix, then computation of U can be done incrementally row-wise where $U[i,:] = A[i,:]\tilde{V}\tilde{\Lambda}^{-1}$ for $i = 1 \dots m$.

Nevertheless, this means that aside from computing POD modes of tall-skinny matrices, method of snapshots suffers from increasing computational costs with an increasing size of vector sets, since computing the Gramian matrix requires $\mathcal{O}(n^2 m)$ flops. As one also computes the eigenvalue instead of the singular values, method of snapshots is subjected to stability issues particularly for POD modes with small singular values.

To conclude, the method of snapshots enables memory-efficient computation of POD modes and singular values for tall-skinny matrices, forming a foundation for parallelized solutions to POD problems [15].

Chapter 3

From HAPOD to HASVD

In this chapter, we develop the theoretical framework for the hierarchical approximate singular value decomposition (HASVD).

Section 3.1 introduces the hierarchical approximate proper orthogonal decomposition (HAPOD), a tree-structured POD method that that serves as the foundation for the HASVD framework [15]. Core concepts and components of HAPOD are presented to illustrate the aggregation techniques used in hierarchical SVD schemes.

Section 3.2 then provides a systematic procedure for aggregating singular vectors and computing the SVD of matrix blocks. This leads to the formulation of low-rank approximations and the complete HASVD framework.

Finally, theoretical error bounds for the approximation error incurred in the HASVD are derived in Section 3.3, and it is shown how to prescribe tolerances a priori such that the targeted global error of a computed approximate SVD is guaranteed.

3.1 Hierarchical approximate proper orthogonal decomposition

In this section, we present the hierarchical approximate proper orthogonal decomposition (HAPOD) as introduced in [15]. HAPOD is a model order reduction technique, and its terminology has been reformulated in the language of numerical linear algebra to serve as the foundation for the hierarchical SVD framework in this thesis.

As shown in Subsection 2.2.3, the method of snapshots incurs quadratic growth in row size n for matrix $A \in \mathbb{R}^{m \times n}$ when forming the Gramian matrix $A^T A$. This scaling becomes prohibitive for tall–skinny matrices and undermines memory-efficient computation.

To overcome these limitations, HAPOD employ a hierarchical approximation strategy that integrates an appropriate approximate POD method within a rooted tree. The method of snapshots discussed in Subsection 2.2.3 is one example of such method. The resulting algorithm achieves both memory efficiency and parallel scalability in the computation of POD modes.

The idea here is that computing the approximate POD modes of a given set of

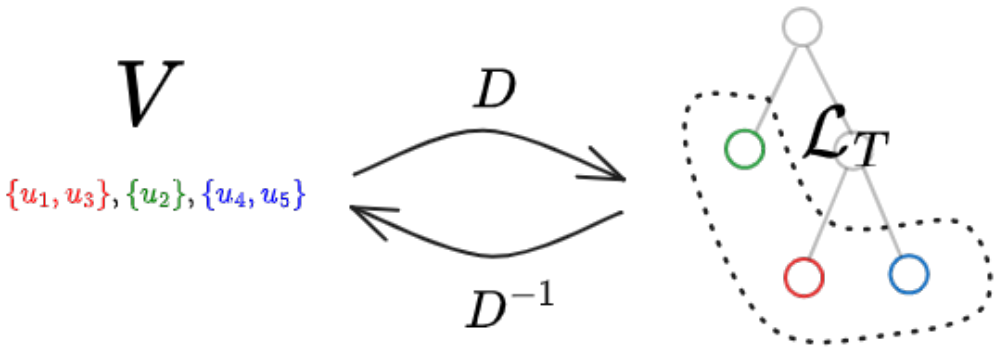


Figure 3.1: Subset-to-leaf mapping of a rooted tree with its respective seperated set of vectors.

vectors $\{u_i\}_{i=1}^n$ is done by performing subtasks involving computing multiple sets of approximate POD modes $\bigcup_{k=1}^{l}\{\varphi_{j_k}\}_{j_k=1}^{m_k}$ derived from disjoint sets of vectors $\{u_{i_k}\}_{i_k=n'_k}^{n_k} \subset \{u_i\}_{i=1}^n$ for $k = 1, \ldots, l$. This is possible because each POD modes are projections of the vector set $\{u_i\}_{i=1}^n$ in an orthonormal basis in $\mathbb{R}^m$ that fulfills Theorem 2.2.2. Any further projection in $\mathbb{R}^m$ would still lead to the initial set of vectors provided that we know which basis is dominant within a given POD .

An integral part of hierarchical approximations is the use of rooted tree to define the hierarchy on aggregating POD modes. It is also desirable to design a hierarchy of aggregation in such a way that the number of relevant POD modes are revealed as early as possible within the HAPOD computation.

To do this, a mapping from subsets of vector $\{u_i\}_{i=1}^n$ to the leaf nodes is introduced.

Definition 3.1.1 (Subset-to-leaf map)**.** *Let $\{u_i\}_{i=1}^n \subset \mathbb{R}^m$ be a finite set of vectors and $P(\{u_i\}_{i=1}^n)$ be its power set. Let $V \subset P(\{u_i\}_{i=1}^n)$ be a set of disjoint sets. Given a rooted tree T along with its set of leaf nodes $\{\alpha_k\}_{k=1}^m = \mathcal{L}_T$, a* ***subset-to-leaf map*** *is a bijective mapping $D : V \to \mathcal{L}_T$.*

Using subset-to-leaf mappings, we can assign disjoint subsets of the vector set $\{u_i\}_{i=1}^n \subset \mathbb{R}^m$ to their respective leaf nodes. This allows one to design the hierarchy for aggregating POD modes along the lowest level of the tree, as shown in Figure 3.1..

Now to aggregate POD modes along the deeper levels of the tree, we specify the steps in HAPOD in Algorithm 3.

Throughout the HAPOD process, the input to the POD computation at each non-leaf node consists of vector sets constructed by weighting the POD modes of its child nodes with their respective singular values. This ensures that the contribution of each mode is scaled according to its significance before aggregation.

The hierarchical framework is, theoretically, independent of the specific method used to perform POD computations. Nevertheless, it enables the decomposition of a given matrix into several smaller tall-skinny matrices. For example, the concatenated snapshot matrix

Algorithm 3: Hierarchical approximate POD

Input: Set of vectors $\{u_i\}_{i=1}^n \subset \mathbb{R}^m$, rooted tree T with subset-to-leaf map D, tolerances $\epsilon : \mathcal{N}_T \to \mathbb{R}^{\geq 0}$.

Output: Set of POD mode and singular value pairs $\{(\varphi_j, \sigma_j)\}_{j=1}^l \subset (\mathbb{R}^m, \mathbb{R})$.

$\mathcal{S} := \emptyset$

while *node* $\alpha \neq$*None or* $\mathcal{S} = \emptyset$ **do**

$\alpha, \mathcal{S} :=$ Traverse$(\rho_T, \mathcal{S})$ (see Algorithm 1)

preparation

if $\alpha \in \mathcal{L}_T$ *(is a leaf node)* **then**

$\{\tilde{u}\} := D^{-1}(\alpha)$, *map to subset vector.*

else

$\{\tilde{u}\} := \{\sigma_{1,\beta_k}\varphi_{1,\beta_k}, \ldots, \sigma_{l_k,\beta_k}\varphi_{l_k,\beta_k}\}_{k=1}^p$, *renormalized set of vectors from HAPOD modes and singular values of the children nodes* $\{\beta_k\}_{k=1}^p = \mathcal{C}_T(\alpha)$.

approx. POD

$(\varphi_{j,\alpha}, \sigma_{j,\alpha})_{j=1}^{l'} :=$ APOD$(\{\tilde{u}\}, \epsilon(\alpha))$, see (2.29)

return $\{(\varphi_j, \sigma_j)\}_{j=1}^l := \{(\varphi_{j,\rho_T}, \sigma_{j,\rho_T})\}_{j=1}^{l''}$, *assign final HAPOD from root node* ρ_T.

$$\overbrace{\left(\begin{array}{c} m \times n \end{array}\right)}^{\text{concatenation of } \{u_i\}_{i=1}^n} \to \left(\begin{array}{c|c|c|c} m \times n_1 & m \times n_2 & \ldots & m \times n_k \end{array}\right),$$

can be split into smaller blocks, where each block corresponds to a subset of the original vector set.

This structure allows HAPOD to leverage memory-efficient computations, particularly through the use of the method of snapshots, which is well-suited for tall-skinny matrices as discussed in Subsection 2.2.3.

Furthermore, HAPOD also has concrete error bounds and a well-defined choice of tolerances along each nodes such that one can determine the total approximation error a priori.

Theorem 3.1.1 (2-norm approximation error of HAPOD [15])**.** *Let there be a rooted tree T with subset-to-leaf map D, set of nodes $\mathcal{N}_T$, set of leaf nodes $\mathcal{L}_T$, and tolerances $\epsilon : \mathcal{N}_T \to \mathbb{R}^{\geq 0}$.*

Given $\alpha \in \mathcal{N}_T$, we denote all nodes below α as $\mathcal{N}_T(\alpha)$ and the vector set that aggregate to node α as $S_\alpha := \bigcup_{\gamma \in \mathcal{L}_T \cap \mathcal{N}_T(\alpha)} D^{-1}(\{\gamma\})$.

Let $\{(\varphi_j, \sigma_j)\}_{j=1}^l \subset (\mathbb{R}^m, \mathbb{R})$ be the approximate POD mode and singular value pairs from HAPOD such that the POD modes form

$$u \approx \sum_{j=1}^{l} a_{u,j}\varphi_j,$$

for all $u \in S_\alpha$, where $a_{u,j} \in \mathbb{R}$ is the correspoding summing coefficients for vector u.

Then the 2*-norm approximation error is*

$$\sum_{u \in S_\alpha} ||u - \sum_{j=1}^{l} a_{u,j}\varphi_j||_2^2 \leq \sum_{\gamma \in \mathcal{N}_T(\alpha)} \epsilon(\gamma)^2. \tag{3.1}$$

The error bound in (3.1) permits the aggregation of local approximation errors at any given node. Consequently, HAPOD tolerances may be prescribed a priori along the tree to guarantee a specified total error. We adopt the same strategy for singular value decompositions: by propagating and controlling local error thresholds, it becomes possible to execute partial—but rigorously bounded SVD computations of a matrix.

It has been shown that the use of HAPOD provide significant improvements to its POD counterpart, with a distributed tree (tree with depth of 1 and multiple leaf nodes) gaining computational speed to the order of 10^3. Additionally, incremental trees (tree with depth of $n-1$ and with n leaf nodes) also gain speedup, particularly in large vector sets (2000 vectors) [15].

Finally, using the SVD, we also obtain an insightful linear algebraic interpretation of matrix approximation. Consider the matrix $A \in \mathbb{R}^{m \times n}$ formed by concatenating the vectors $\{u_i\}_{i=1}^n \subset \mathbb{R}^m 0$. Suppose the set $\{u_i\}_{i=1}^n \subset \mathbb{R}^m$ is paritioned into disjoint subsets. With an appropriate choice of indices, this correspond to a horizontal partitioning of a matrix,

$$\begin{aligned} A &= \left(\begin{array}{c|c|c|c|c} A_1 & A_2 & \dots & A_{k-1} & A_k \end{array} \right) \\ &= \left(\begin{array}{c|c|c|c|c} U_1\Sigma_1 V_1^T & U_2\Sigma_2 V_2^T & \dots & U_{k-1}\Sigma_{k-1}V_{k-1}^T & U_k\Sigma_k V_k^T \end{array} \right). \end{aligned}$$

By factoring out the right singular vectors, we obtain the decomposition,

$$A = \overbrace{\begin{pmatrix} U_1\Sigma_1 & U_2\Sigma_2 & \cdots & U_k\Sigma_k \end{pmatrix}}^{\tilde{A}} \begin{pmatrix} V_1^T & & & \\ & V_2^T & & \\ & & \ddots & \\ & & & V_k^T \end{pmatrix}, \tag{3.2}$$

where $\tilde{A}$ is the concatenation of vectors as defined for non-leaf nodes in Algorithm 3.

This factorization yields a clear linear algebraic interpretation in terms of the Frobenius norm. By the unitary invariance of the Frobenius norm, it holds that $\|A\|_F = \|\tilde{A}\|_F$, indicating that the decomposition preserves the sum of squared singular values of the original matrix. Consequently, this structure ensures that low-rank approximations based on this decomposition are optimal with respect to the Frobenius norm. This observation provides the foundation for the hierarchical construction of $\tilde{A}$, which underlies the hierarchical SVD framework introduced in the next section.

3.2 Hierarchical approximate singular value decomposition

In this section, we will use HAPOD as a basis to expand the framework for SVD. This will be done by formulating an aggregation method for the matrices of left and right singular vectors and defining rules for performing aggregations in block matrices.

3.2.1 Aggregation of singular vectors

As demonstrated in the Section 3.1, factorizing the horizontal partitioning of a matrix A will result in the concatenation of normalized POD modes, denoted $\tilde{A}$ as shown in Algorithm 3, along with the block diagonal matrix $\text{diag}(V_1^T, V_2^T, \ldots, V_k^T)$. In this section, we extend this idea to develop a method for aggregating the local singular vectors from each block and constructing a global SVD of the entire matrix.

We continue the example from the previous subsection by considering the horizontal partitioning of a matrix $A \in \mathbb{R}^{m\times n}$ to k blocks of column with lengths $n_1, n_2, \ldots n_k$ for the purpose of computing a distributed HAPOD computation,

$$A = \begin{pmatrix} U_1\Sigma_1V_1^T & U_2\Sigma_2V_2^T & \cdots & U_k\Sigma_kV_k^T \end{pmatrix}.$$

Following Algorithm 3, the HAPOD of the entire matrix corresponds to a HAPOD task on the parent of each leaf node, which corresponds to computing the POD of the matrix $\tilde{A}$ in (3.2).

Using an SVD procedure, we compute

$$\tilde{A} = \begin{pmatrix} U_1\Sigma_1 & U_2\Sigma_2 & \cdots & U_k\Sigma_k \end{pmatrix} = \tilde{U}\tilde{\Sigma}\tilde{V}^T, \tag{3.3}$$

where the matrix of singular values $\tilde{\Sigma}$ preserves the singular values of the original matrix A and the matrix of left singular vectors $\tilde{U}$ is also preserved, up to a global sign ambiguity. This equivalence follows from the unitary invariance of the Frobenius norm and the fact that each product $U_i\Sigma_i$ captures the column space and magnitude of block $A_i = U_i\Sigma_iV_i^T$. Hence, the SVD of matrix $\tilde{A}$ yields singular values and left singular vectors equivalent to those of A, justifying its use in the aggregation step of HAPOD.

Moreover, as shown in (3.2), the matrix $\tilde{A}$ can be formed from the factorization of matrix A,

$$A = \begin{pmatrix} U_1\Sigma_1 & U_2\Sigma_2 & \ldots & U_k\Sigma_k \end{pmatrix} \underbrace{\begin{pmatrix} V_1^T & 0 & \cdots & 0 \\ 0 & V_2^T & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & V_k^T \end{pmatrix}}_{\hat{V}^T}.$$

Substituting the SVD of $\tilde{A}$ from (3.3), we obtain

$$A = \tilde{U}\tilde{\Sigma}\tilde{V}^T\hat{V}^T. \tag{3.4}$$

This decomposition is itself a valid SVD of A, since:

- $\tilde{U}$ match the matrix left singular vectors of A (up to a sign difference on the columns),
- $\tilde{\Sigma}$ match the matrix of singular values of A,
- and the product $\tilde{V}^T\hat{V}^T$ forms a unitary matrix.

The unitarity of $\tilde{V}^T\hat{V}^T$ is shown in (3.5), where

$$\begin{aligned}\tilde{V}^T\hat{V}^T(\tilde{V}^T\hat{V}^T)^T =& \tilde{V}^T \begin{pmatrix} V_1^T & 0 & \cdots & 0 \\ 0 & V_2^T & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & V_k^T \end{pmatrix} \begin{pmatrix} V_1 & 0 & \cdots & 0 \\ 0 & V_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & V_k \end{pmatrix} \tilde{V} \\ =& \tilde{V}^T \underbrace{\begin{pmatrix} I_{n_1} & 0 & \cdots & 0 \\ 0 & I_{n_2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & I_{n_k} \end{pmatrix}}_{I_n} \tilde{V} = \tilde{V}^T\tilde{V} = I_n.\end{aligned} \tag{3.5}$$

This confirms that the product $\tilde{V}^T\hat{V}^T$ is orthonormal, and thus $A = \tilde{U}\tilde{\Sigma}\tilde{V}^T\hat{V}^T$ represents a full SVD of the matrix A. This establishes a framework in which the right singular vectors can also be hierarchically aggregated from a partitioned matrix.

Remark 3.2.1. *Let $A \in \mathbb{R}^{m\times n}$ where $A = \begin{pmatrix} A_1 & A_2 & \cdots & A_k \end{pmatrix}$ consisting of block of columns of A, $(A_i)_{i=1}^k$, with SVDs $A_i = U_i\Sigma_i V_i^T$. Then there is a singular value decomposition of $A = U\Sigma V^T$ such that*

$$U\Sigma\tilde{V}^T = \begin{pmatrix} U_1\Sigma_1 & U_2\Sigma_2 & \cdots & U_3\Sigma_3 \end{pmatrix} \text{ and } V^T = \tilde{V}^T\hat{V}^T \tag{3.6}$$

where

$$\hat{V}^T = \begin{pmatrix} V_1^T & 0 & \cdots & 0 \\ 0 & V_2^T & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & V_k^T \end{pmatrix} \tag{3.7}$$

*is the **aggregated matrix of right singular vectors**.*

However, it should be pointed out that within a hierarchical SVD, in addition to the aggregation of the matrix of right singular vectors, one can also aggregate the matrix of the left singular vectors. This will extend the scope of hierarchical SVD not to just horizontal partitioning but also vertical partitioning of matrices.

Suppose that we have a vertical partitioning of matrix A into k rows of lengths $m_1, m_2, \dots m_k$, then we have

$$A = \begin{pmatrix} U_1\Sigma_1 V_1^T \\ U_2\Sigma_2 V_2^T \\ \vdots \\ U_k\Sigma_k V_k^T \end{pmatrix}.$$

By performing a factorization on A, we obtain

$$A = \overbrace{\begin{pmatrix} U_1 & 0 & \dots & 0 \\ 0 & U_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & U_k \end{pmatrix}}^{\hat{U}} \begin{pmatrix} \Sigma_1 V_1^T \\ \Sigma_2 V_2^T \\ \vdots \\ \Sigma_k V_k^T \end{pmatrix},$$

where $\hat{U}$ is a unitary matrix. Moreover, the remaining factor also has an SVD

$$\begin{pmatrix} \Sigma_1 V_1^T \\ \Sigma_2 V_2^T \\ \vdots \\ \Sigma_k V_k^T \end{pmatrix} = \tilde{U}\Sigma V^T.$$

Therefore, matrix A also has an SVD of the form $A = \hat{U}\tilde{U}\Sigma V^T$, since $\hat{U}\tilde{U}$ is unitary, because

$$\begin{aligned} \hat{U}\tilde{U}(\hat{U}\tilde{U})^T &= \begin{pmatrix} U_1 & 0 & \cdots & 0 \\ 0 & U_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & U_k \end{pmatrix} \underbrace{\tilde{U}\tilde{U}^T}_{I_m} \begin{pmatrix} U_1^T & 0 & \cdots & 0 \\ 0 & U_2^T & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & U_k^T \end{pmatrix} \\ &= \begin{pmatrix} I_{m_1} & 0 & \cdots & 0 \\ 0 & I_{m_2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & I_{m_k} \end{pmatrix} = I_m. \end{aligned}$$

This demonstrates that the SVD of matrix A can also be derived from smaller SVDs of the vertical partitioning of matrix A.

Remark 3.2.2. *Let $A \in \mathbb{R}^{m\times n}$ where $A = \begin{pmatrix} A_1^T & A_2^T & \cdots & A_k^T \end{pmatrix}^T$ consisting of block matrices $(A_i)_{i=1}^k$ with SVDs $A_i = U_i\Sigma_i V_i^T$. Then there is a singular value decomposition of $A = U\Sigma V^T$ such that*

$$\tilde{U}\Sigma V^T = \begin{pmatrix} V_1\Sigma_1 & V_2\Sigma_2 & \cdots & V_k\Sigma_k \end{pmatrix}^T \text{ and } U = \hat{U}\tilde{U} \tag{3.8}$$

where

$$\hat{U} = \begin{pmatrix} U_1 & 0 & \cdots & 0 \\ 0 & U_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & U_k \end{pmatrix}$$

is the ***aggregated matrix of left singular vectors****.*

The above observations on aggregations of matrices of singular vectors enables us to extend POD, which only concentrates on set of vectors (POD modes) along the column space, to computations of rows and columns.

This indicates that one can perform the hierarchical SVD of entire block matrix structures by using Remark 3.2.1 and Remark 3.2.2 in sequences of aggregation operations, which will form the basis of block aggregations as will be discussed in the next subsection.

3.2.2 Aggregation of blocks

With aggregation of singular vectors treated, we now introduce a hierarchical framework for computing the singular value decomposition (SVD) of block matrices from [45], where studies of column and row block aggregation has been studied.

In Section 3.2.1, we have shown that the SVD of a matrix A can be derived from smaller SVDs of its corresponding blocks, both for row partitioning and column partitioning of the block matrix.

This can be generalized further for arbitrary block matrices, which will be demonstrated in the following m-by-n block matrix,

$$B = \left(\begin{array}{c|cc} & A_2 & \\ A_1 & A_3 & \\ & A_4 & A_5 \\ \hline & A_6 & \\ \hline & A_7 & A_8 \end{array}\right).$$

To compute the aggregated SVD of matrix B, we proceed by performing successive aggregations along both columns and rows. Similar to the hierarchical approach used in HAPOD, the order of aggregation is defined by a rooted tree structure that encodes the dependencies and sequence of aggregation operations.

We begin the computation of the SVD of B by decomposing it to smaller SVD sub-problems. As a first step, we focus on computing the SVD of matrix A_9 which is a block that contains A_6, A_7, and A_8. Suppose now that we have computed the approximate SVDs of each block, i.e. $A_i \approx U_i \Sigma_i V_i^T$. We have

$$A_9 = \left(\begin{array}{c} U_6 \Sigma_6 V_6^T \\ \hline U_7 \Sigma_7 V_7^T \mid U_8 \Sigma_8 V_8^T \end{array}\right).$$

To obtain the SVD of A_9, we first identify adjacent blocks that can be aggregated.

Definition 3.2.1. *Let $A \in \mathbb{R}^{m\times n}$ and we define $A[i:j, k:l]$ as the block of matrix A consisting of elements from the i-th to the j-th row and from the kth column to the lth column.*

Let $A_1 = A[i_1 : j_1, k_1 : l_1]$ and $A_2 = A[i_2 : j_2, k_2 : l_2]$ be blocks of A. Then:

*i) A_1 is **row-adjacent** to A_2 if $i_1 = i_2$, $j_1 = j_2$, and $k_2 = l_1 + 1$ (or $k_1 = l_2 + 1$),*

*ii) A_1 is **column-adjacent** to A_2 if $k_1 = k_2$, $l_1 = l_2$, and $i_2 = j_1 + 1$ (or $i_1 = j_2 + 1$).*

We begin by aggregating the row-adjacent blocks A_7 and A_8 along its columns. This results in a block $A_{10} = U_{10}\Sigma_{10}V_{10}^T = U_{10}\Sigma_{10}\tilde{V}_{10}^T\hat{V}_{10}^T$, where $U_{10}\Sigma_{10}\tilde{V}_{10}^T = (U_7\Sigma_7, U_8\Sigma_8)$ is an SVD of block A_{10} and $\hat{V}_{10}^T = \mathrm{diag}(V_7^T, V_8^T)$. Finally, the approximate of A_9 is then obtained by aggregating A_6 and A_{10} along its rows. The resulting SVD is $A_9 = U_9\Sigma_9V_9^T = \hat{U}_9\tilde{U}_9\Sigma_9V_9^T$, where $\tilde{U}_9\Sigma_9V_9^T = (V_6\Sigma_6, V_{10}\Sigma_{10})^T$ and $\hat{U}_9 = \mathrm{diag}(U_6, U_{10})$. The steps can be summarized in the following procedure:

$$\left(\begin{array}{c|c} U_6\Sigma_6V_6^T & \\ \hline U_7\Sigma_7V_7^T & U_8\Sigma_8V_8^T \end{array}\right) \xrightarrow{\text{row aggregation}} \left(\begin{array}{c} U_6\Sigma_6V_6^T \\ \hline U_{10}\Sigma_{10}\tilde{V}_{10}^T\hat{V}_{10}^T \end{array}\right) \xrightarrow{\text{column aggregation}} \left(\hat{U}_9\tilde{U}_9\Sigma_9V_9^T\right).$$

Remark 3.2.3. *Let $A \in \mathbb{R}^{m\times n}$ be a block matrix and T be a rooted tree for aggregating blocks of A.*

*Let $\{A_\alpha\}_{\alpha\in\mathcal{L}_T}$ bet the set of blocks of A that is associated with the leaf nodes $\mathcal{L}_T$ of T. Such blocks are connected to the leaf nodes by a bijective mapping called the **block-to-leaf map**, $D : \{A_\alpha\}_{\alpha\in\mathcal{L}_T} \to \mathcal{L}_T$ such that*

$$D(A_\alpha) := \alpha,$$

where A_α is a block of the matrix A and α is its associated leaf node.

Let $\{A_\alpha\}_{\alpha\in\mathcal{N}_T\setminus\mathcal{L}_T}$ be blocks consisting of combinations of blocks $\{A_\alpha\}_{\alpha\in\mathcal{L}_T}$.

Then for all $\alpha \in \mathcal{N}_T \setminus \mathcal{L}_T$, $\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathcal{C}_T(\alpha)$ are its children if and only if $\{A_{\alpha_1}, \ldots, A_{\alpha_k}\}$ are row-(column-)adjacent blocks.

As shown in Remark 3.2.3, every block inside A_9 can be aggregated as long as its row size or its column sizes coincide, for column aggregation or row aggregation respectively. These sequences of rows and column aggregation can be represented as a tree as shown in Figure 3.2. Note that rooted tree of aggregations is non-unique, which means there can be different ways of constructing the tree without violating the aggregation rule.

To connect each block to a rooted tree, we use a block-to-leaf map, which helps us define where a specific block belongs in the initial approximation SVDs in the HASVD process. Afterwards, the rooted tree the provides the instruction and order of aggregation of each of the blocks.

By using block-to-leaf map and the rule in Remark 3.2.3, we can construct a tree for B as well as demonstrated in Figure 3.3.

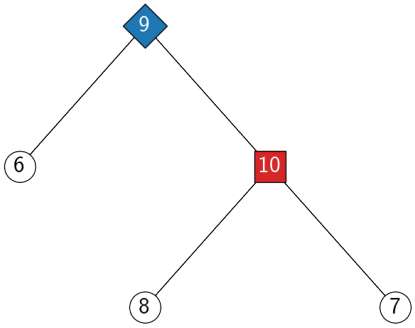


Figure 3.2: A rooted tree defining aggregation orders for block A_9, blue diamond indicates column aggregation and red square indicates row aggregation.

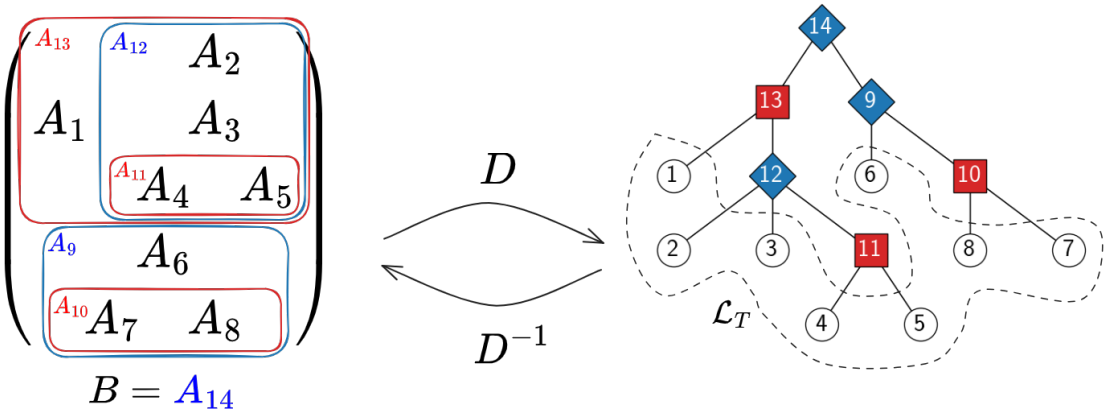


Figure 3.3: Construction of a rooted tree of aggregation for matrix B, using block adjacency rules. Block matrices are mapped to leaf by a block-to-leaf map D.

With the aggregation rules in place, we define an aggregation step AggregatedSVD() in Algorithm 4 to compute an aggregated SVD given a rooted tree T, non-leaf node $\alpha \in \mathcal{N}_T \setminus \mathcal{L}_T$, a set of SVDs of its children node $\{(U_\beta, \Sigma_\beta, V_\beta^T)\}_{\beta \in \mathcal{C}_T(\alpha)}$.

In the algorithm, we begin by preparing the aggregation matrix using equations (3.6) and (3.8), through A for the given non-leaf node $\alpha \in \mathcal{N}_T \setminus \mathcal{L}_T$.

The direction of aggregation is determined by x, where $x = 0$ denotes row aggregation and $x = 1$ denotes column aggregation. In each case preparation for aggregated SVD computations follows from Remark 3.2.1 for row aggregation, and 3.2.2 for column aggregation.

Once the aggregation matrix is prepared, the SVD of matrix A is computed. Then aggregation of the matrix of right (left) singular vectors is done for row (column) aggregations to complete the SVD step.

Algorithm 4 will play an important role in aggregating SVD between blocks systematically. While this yields a structured decomposition of the original matrix, it does not yet offer computational advantages. To address this, we take inspiration from the approximate POD method discussed in Subsection 2.2.2. In Section 3.2.3, we extend the framework to develop a full hierarchical approximate singular value decomposition (HASVD).

Algorithm 4: AggregatedSVD() - Aggregation step function

Input: Node $\alpha \in \mathcal{N}_T$, set of children SVDs $\{(U_\beta, \Sigma_\beta, V_\beta^T)\}_{\beta \in \mathcal{C}_T(\alpha)}$, rooted tree T, and aggregation direction label x.

Output: Matrix of left singular vectors of node α, $U \in \mathbb{R}^{m \times m}$, matrix of right singular vectors of node α, $V^T \in \mathbb{R}^{n \times n}$, and singular value matrix $\Sigma \in \mathbb{R}^{m \times n}$.

$A, W := (\ \ )$, initialize empty matrices for SVD and aggregation.

switch *agg. direction label* x **do**

  case 0 *(row aggregation)* **do**

    for *children nodes* $\beta \in \mathcal{C}_T(\alpha)$ **do**

      $A := \begin{pmatrix} A & U_\beta \Sigma_\beta \end{pmatrix}$, concatenate SVD matrix along rows.

      $W := \begin{pmatrix} W & 0 \\ 0 & V_\beta^T \end{pmatrix}$, concatenate aggregation matrix along diagonal.

  case 1 *(column aggregation)* **do**

    for *children nodes* $\beta \in \mathcal{C}_T(\alpha)$ **do**

      $A := \begin{pmatrix} A \\ \Sigma_\beta V_\beta^T \end{pmatrix}$, concatenate SVD matrix along columns.

      $W := \begin{pmatrix} W & 0 \\ 0 & U_\beta \end{pmatrix}$, concatenate aggregation matrix along diagonal.

$U \Sigma V^T = A$, compute SVD.

switch *node label* $x(\alpha)$ **do**

  case 0 *(row aggregation)* **do** $V^T := V^T W$;

  case 1 *(column aggregation)* **do** $U := WU$;

return U, Σ, V^T,

3.2.3 Approximation and framework

In this subsection, we aim to utilize the hierarchical framework that has been constructed in Subsection 3.2.2 and 3.2.1, and construct a procedure to reduce the problem size of the full SVD by introducing an approximate SVD that is obtained by truncating the computed SVD of the blocks up to a given tolerance.

In Theorem 2.2.3, we have an approximate POD that truncate the number of POD mode-singular value pairs to the $\{(\varphi_i, \sigma_i)\}_{i=1}^r$ where $r < \ell$ from the POD modes of vector set $\{u_i\}_{i=1}^k$. This is done by giving a tolerance $\epsilon > 0$ for the POD $\{\tilde{u}_i\}_{i=1}^k$ such that the 2-norm approximation error in (2.28) is fulfilled, given by

$$\sum_{i=1}^{k} ||u_i - \tilde{u}_i||_2^2 = \sum_{i=r+1}^{\ell} \sigma_i^2 \leq \epsilon^2, \tag{3.9}$$

where $\sigma_{r+1}, \sigma_{r+2}, \ldots, \sigma_\ell$ are the singular values of the omitted POD modes.

We propose a similar step for block matrices. Given a matrix $A \in \mathbb{R}^{m \times n}$ of rank ℓ, the rank-r approximation $A_r \in \mathbb{R}^{m \times n}$ fulfills a given tolerance $\epsilon > 0$ of the Frobenius (norm) error,

$$\|A - A_r\|_F^2 \leq \epsilon^2. \tag{3.10}$$

By Eckhart-Young Theorem (Theorem 1.1.1), the best approximation is obtained by a truncated SVD, and for the Frobenius norm, also coincides with the sum of the last $\ell - r$ squared singular values of A, similar to (3.9).

To construct this approximation, we compute the SVD of $A = U\Sigma V^T$ and truncate the matrix of singular vectors U, V^T, and matrix of singular values Σ,

$$U_r = U[1:m, 1:r], \quad \Sigma_r = \Sigma[1:r, 1:r], \quad V_r = V[1:n, 1:r], \tag{3.11}$$

such that $A_r = U_r \Sigma_r V_r^T$ is the **approximate SVD** of A.

How does this impact the hierarchical framework?

By applying truncation after computing the SVD of the individual blocks, one may reduce the size of the SVD computation of subsequent blocks in higher levels. As a demonstration, suppose we have a partitioned matrix $A \in \mathbb{R}^{m\times n}$, such that

$$A = \begin{pmatrix} A_1 & \cdots & A_k \end{pmatrix}, \tag{3.12}$$

where $A_i = U_i \Sigma_i V_i^T \in \mathbb{R}^{m\times n_i}$ are the SVD of the column blocks with $n_1 + n_2 + \cdots + n_k = n$ for $i = 1, \ldots, k$. Computing the approximate SVD of A_i, we get

$$A_{i,r_i} = U_{i,r_i} \Sigma_{i,r_i} V_{i,r_i}^T, \tag{3.13}$$

where $U_{i,r_i} \in \mathbb{R}^{m\times r_i}$ and $V_{i,r_i}^T \in \mathbb{R}^{r_i \times n_i}$ are the **approximate matrix of singular vectors**, and $\Sigma_{i,r_i} \in \mathbb{R}^{r_i \times r_i}$ is the **approximate matrix of singular values**, with $r_i < n_i$ and $r_1 + r_2 + \cdots + r_k = r < n$ for $i = 1, \ldots, k$.

Using (3.6) in the hierarchical framework as a row aggregation step for computing SVD of A, we are left to compute the SVD

$$\begin{pmatrix} U_{1,r_1}\Sigma_{1,r_1} & U_{2,r_2}\Sigma_{2,r_2} & \cdots & U_{k,r_k}\Sigma_{k,r_k} \end{pmatrix} = U_r \Sigma_r \tilde{V}_r^T \in \mathbb{R}^{m\times(r_1+r_2+\cdots+r_k)}, \tag{3.14}$$

where $U_r \in \mathbb{R}^{m\times r}$ is the r-rank approximate matrix of left singular vectors, $\Sigma_r \in \mathbb{R}^{r\times r}$ is the r-rank approximate matrix of singular values, and $\tilde{V}_r^T \in \mathbb{R}^{r\times r}$ is the intermediate r-rank matrix of left singular vectors. Since $r < n$, the truncations effectively lead to smaller problem size in the aggregation step.

By introducing the **approximate aggregated matrix of right singular vectors** $\hat{V}_r^T$ from (3.7), we obtain

$$\hat{V}_r^T = \begin{pmatrix} V_{1,r_1}^T & 0 & \cdots & 0 \\ 0 & V_{1,r_2}^T & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & V_{1,r_k}^T \end{pmatrix} \in \mathbb{R}^{(r_1+r_2+\cdots+r_k)\times(n_1+n_2+\cdots+n_k)}, \tag{3.15}$$

where $n_1 + n_2 + \cdots + n_k = n$.

Subsequently, by following (3.6) the rank-r approximation $A_r = U_r \Sigma_r V_r^T \in \mathbb{R}^{m \times n}$ and the r-rank approximate matrix of right singular vectors $V_r^T = \tilde{V}_r^T \hat{V}_r^T$ is obtained.

Note that U_r and V_r^T is no longer unitary. However, it is still isometric, i.e., $U_r^T U_r = I_r$ and $U_r U_r^T \neq I_m$, and therefore the Frobenius norm A_r relative to A is preserved up to truncation errors, leading to the approximation error,

$$\|A - A_r\|_F^2 = \sum_{i=1}^{k} \sum_{j=r_i+1}^{n_i} \sigma_{i,j}^2, \tag{3.16}$$

where $\sigma_{i,j}$ is the j-th truncated singular value of block A_i for $j = r_i + 1, \ldots, n_i$ and $i = 1, \ldots, k$.

Similar to (2.29), we introduce a shorthand for a truncation operation to use in subsequent algorithms. Given a matrix of singular vectors $\Sigma \in \mathbb{R}^{m \times n}$ obtained from computing SVD $A = U\Sigma V^T$ of rank k. The **truncation rank** $\text{TruncationRank}(\Sigma, \epsilon)$, for some $\epsilon > 0$, is defined as

$$\text{TruncationRank}(\Sigma, \epsilon) := \underset{r:\sqrt{\text{tr}(\Sigma[r:m,r:n]^T \Sigma[r:m,r:n])} < \epsilon}{\arg\max} r, \tag{3.17}$$

where $\text{tr}(\cdot)$ denotes the trace of a square matrix.

The truncation rank in (3.17) computes the largest possible r, such that sum of the smallest $k - r$ squared singular values of Σ is less than ϵ, which is equivalent to the Frobenius norm error of an r-rank approximation A_r, i.e., applying $\|A - A_r\|_F$ using (2.7).

With this we present the complete algorithmic framework of the hierarchical approximate singular value decomposition (HASVD).

The complete procedure of the HASVD is outlined in Algorithm 5. In the algorithm, we start by preparing blocks A_α for each node $\alpha \in \mathcal{N}_T$. If α is a leaf node, we map it to its corresponding block using the block-to-leaf map D. Otherwise, we prepare for an aggregation of blocks (see Algorithm 4), indicated by the **node label** $x(\alpha)$, where $x(\alpha) = 0$ indicates row aggregation and $x(\alpha) = 1$ indicates column aggregation.

After each SVD computation, an approximate SVD is obtained by truncating:

- up to r columns for matrix of left singular vectors U_α,
- up to r rows for matrix of right singular vectors V_α^T,
- and up to r columns and rows for matrix of singular vectors Σ_α.

Once all the nodes have been processed, the final approximate matrix of left singular vectors U, approximate matrix of right singular vectors V^T, and approximate matrix of singular values Σ are returned. These are associated with the root node ρ_T of the tree T.

Algorithm 5: Hierarchical approximate singular value decomposition (HASVD)

Input: Matrix $A \in \mathbb{R}^{m \times n}$, rooted tree T with block-to-leaf map D, local tolerances $\epsilon : \mathcal{N}_T \to \mathbb{R}^{\geq 0}$, and aggregation direction labels $x : \mathcal{N}_T \to \{0, 1\}$.

Output: Approximate matrix of left singular vectors $U \in \mathbb{R}^{m \times r}$, approximate matrix of right singular vectors $V^T \in \mathbb{R}^{r \times n}$, and approximate singular value matrix $\Sigma \in \mathbb{R}^{r \times r}$

$\mathcal{S} := \emptyset$, intialize explored set of nodes

while *node* $\alpha \neq$*None or* $\mathcal{S} = \emptyset$ **do**

  $\alpha, \mathcal{S} :=$ Traverse$(\rho_T, \mathcal{S})$, see Algorithm 1.

hierarchical SVD

  if $\alpha \in \mathcal{L}_T$ *(is a leaf node)* **then**

    $A_\alpha := D^{-1}(\alpha)$, map to block.

    $U_\alpha \Sigma_\alpha V_\alpha^T := A_\alpha$, compute SVD of partitioned block.

  else

    $U_\alpha, \Sigma_\alpha, V_\alpha^T :=$ AggregatedSVD$(\alpha, \{(U_\beta, \Sigma_\beta, V_\beta^T)\}_{\beta \in \mathcal{C}_T(A)}, T, x(\alpha))$, compute aggregated SVD from children nodes, see Algorithm 4.

approximate

  $r_\alpha :=$ TruncationRank$(\Sigma_\alpha, \epsilon(\alpha))$, truncate to approximate SVD using (3.13).

  $\Sigma_\alpha := \Sigma_\alpha[: r_\alpha, : r_\alpha]$

  $U_\alpha := U_\alpha[:, : r_\alpha]$

  $V_\alpha^T := V_\alpha^T[: r_\alpha, :]$

return $U := U_{\rho_T}$, $V^T := V_{\rho_T}^T$, $\Sigma := \Sigma_{\rho_T}$

Using the Traverse() function in Algorithm 1, it should be reemphasized that the tree structure determines the order of aggregation, where child nodes take precedence over parent nodes.

Furthermore, as seen in Algorithm 5, the final rank r, and by proxy the Frobenius error, depend entirely on:

- the tree structure, and
- the local tolerances ϵ.

In this regard, both aspects will be examined. The procedure for prescribing tolerances is discussed in Section 3.3, while the effect of different tree structures will be addressed in Chapter 5 on numerical experiments.

3.3 Error analysis and tolerance prescription for HASVD

In this section, we develop rigorous bounds for quantifying and controlling the approximation error of HASVD. First, we derive theoretical error bounds that relate the errors at each node of a rooted tree to the total error of the SVD of the matrix. Building on these results, we then show how to prescribe tolerances a priori so that the targeted global error is guaranteed.

Together, these two components—error bounding and tolerance prescription—provide a systematic recipe for performing partial SVD computations in large-scale or distributed settings.

3.3.1 Error bound analysis

When computing the Frobenius norm error of the approximation A_r of matrix $A \in \mathbb{R}^{m\times n}$, obtained via HASVD through a rooted tree T, we must consider two kinds of errors

- errors from computing the SVD of blocks $\{A_i\}$ of A,
- and errors obtained from the aggregation step with a particular non-leaf node $\alpha \in \mathcal{N}_T \setminus \mathcal{L}_T$.

These errors aggregate as we traverse through the rooted tree T. It is therefore important to identify the sources of these errors and understand how their magnitudes contribute to the overall Frobenius norm error bound of the approximation.

Definition 3.3.1 (Nodal error)**.** *Let $A \in \mathbb{R}^{m\times n}$ be a matrix. Given a rooted tree T that defines blocks $\{A_\alpha\}_{\alpha\in\mathcal{N}_T}$ of A with approximate SVDs of the form $A_\alpha \approx U_\alpha\Sigma_\alpha V_\alpha^T$, we define the* ***nodal error*** *of the node α as*

$$\epsilon_\alpha := ||A_\alpha - U_\alpha\Sigma_\alpha V_\alpha^T||_F. \tag{3.18}$$

Nodal errors arise either from performing standard SVDs or from aggregation steps as we traverse the tree during a HASVD computation. While the nodal errors at leaf nodes are straightforward to compute, those at non-leaf nodes are also relatively simple to evaluate.

Suppose we have a non-leaf node $\alpha \in \mathcal{N}_T \setminus \mathcal{L}_T$, then an aggregation from HASVDs of its children nodes $\beta_1, \beta_2, \ldots, \beta_l$ is performed.

If the node label is $x(\alpha) = 0$ (row aggregation), we have the nodal error

$$\|A_\alpha - U_\alpha\Sigma_\alpha V_\alpha^T\|_F = \|A_\alpha - U_\alpha\Sigma_\alpha\tilde{V}_\alpha^T\hat{V}_\alpha^T\|_F, \tag{3.19}$$

where $\tilde{V}_\alpha^T$ is the matrix of right singular vectors of the SVD of the matrix of the normalized left singular vectors and and $\hat{V}_\alpha^T = \mathrm{diag}(V_{\beta_1}, V_{\beta_2}, \ldots, V_{\beta_l})$ which follows from Remark 3.2.1. We also know that A_α can be factorized,

$$\|A_\alpha - U_\alpha\Sigma_\alpha\tilde{V}_\alpha^T\hat{V}_\alpha^T\|_F = \|\begin{pmatrix}U_{\beta_1}\Sigma_{\beta_1} & \cdots & U_{\beta_l}\Sigma_{\beta_l}\end{pmatrix}\hat{V}_\alpha^T - U_\alpha\Sigma_\alpha\tilde{V}_\alpha^T\hat{V}_\alpha^T\|_F. \tag{3.20}$$

By the unitary invariance of the Frobenius norm, $\hat{V}_\alpha^T$ can be omitted as it still contains orthogonal columns. This leads to the nodal error

$$\epsilon_\alpha = ||\begin{pmatrix}U_{\beta_1}\Sigma_{\beta_1} & \cdots & U_{\beta_l}\Sigma_{\beta_l}\end{pmatrix} - U_\alpha\Sigma_\alpha\tilde{V}_\alpha^T||_F. \tag{3.21}$$

If the node label is $x(\alpha) = 1$ (column aggregation), one can also use the unitary invariance to get the error

$$\epsilon_\alpha = \left|\left| \begin{pmatrix} \Sigma_{\beta_1} V_{\beta_1}^T \\ \vdots \\ \Sigma_{\beta_l} V_{\beta_l}^T \end{pmatrix} - \tilde{U}_\alpha \Sigma_\alpha V_\alpha^T \right|\right|_F . \tag{3.22}$$

We have proved the following result.

Lemma 3.3.1. *Let A_α be a block of $A \in \mathbb{R}^{m\times n}$ defined by a rooted tree T and a non-leaf node α. Given node label $x(\alpha)$, the nodal error is:*

$$\epsilon_\alpha = \begin{cases} \left|\left| \begin{pmatrix} U_{\beta_1}\Sigma_{\beta_1} & \cdots & U_{\beta_l}\Sigma_{\beta_l} \end{pmatrix} - U_\alpha \Sigma_\alpha \tilde{V}_\alpha^T \right|\right|_F, & x(\alpha) = 0, \\ \left|\left| \begin{pmatrix} V_{\beta_1}\Sigma_{\beta_1} & \cdots & V_{\beta_l}\Sigma_{\beta_l} \end{pmatrix}^T - \tilde{U}_\alpha \Sigma_\alpha V_\alpha^T \right|\right|_F, & x(\alpha) = 1. \end{cases}$$

As an example of computing nodal errors, we start with observing the error of a distributed HASVD.

Definition 3.3.2 (Distributed HASVD)**.** *Let T be a rooted tree with a single non-leaf node, $|\mathcal{N}_T| = 1$ and k leaf node $|\mathcal{L}_T| = k$. We call the rooted tree T a **distributed tree**.*

*Any HASVD procedure that follows the aggregation order of the rooted tree T is called a **distributed HASVD**.*

Lemma 3.3.2. *Let the approximation $U\Sigma V^T \in \mathbb{R}^{m\times n}$ of a matrix $A \in \mathbb{R}^{m\times n}$ be computed by a distributed HASVD. Given blocks $\{A_{\alpha_i}\}_{i=1}^k$ of matrix A due to column (row) partitioning and the nodal errors $\{\epsilon_{\alpha_i}\}_{i=1}^k$ of the leaf nodes, the Frobenius error of the approximation $U\Sigma V^T$ is bounded by*

$$||A - U\Sigma V^T||_F \leq \epsilon_{\rho_T} + \sqrt{\sum_{i=1}^k \epsilon_{\alpha_i}^2}, \tag{3.23}$$

where ϵ_{ρ_T} is the nodal error of the root node.

Proof. Given A, approximation $U\Sigma V^T$, and distributed tree T with root node ρ_T and leaf nodes α_i for $i = 1 \ldots k$, we have

$$A = \begin{pmatrix} A_{\alpha_1} & A_{\alpha_2} & \cdots & A_{\alpha_k} \end{pmatrix} \tag{3.24}$$

Without loss of generality, we indexed the blocks in (3.24) such that neighboring indices belong to row-adjacent matrices. Then by the triangle inequality, we have

$$
\begin{aligned}
||A - U\Sigma V^T||_F =& ||\begin{pmatrix} A_{\alpha_1} & A_{\alpha_2} & \cdots & A_{\alpha_k} \end{pmatrix} - U\Sigma V^T||_F \\
=& ||\begin{pmatrix} A_{\alpha_1} & A_{\alpha_2} & \cdots & A_{\alpha_k} \end{pmatrix} \\
& - \begin{pmatrix} U_{\alpha_1}\Sigma_{\alpha_1}V_{\alpha_1}^T & U_{\alpha_2}\Sigma_{\alpha_2}V_{\alpha_2}^T & \cdots & U_{\alpha_k}\Sigma_{\alpha_k}V_{\alpha_k}^T \end{pmatrix} \\
& + \begin{pmatrix} U_{\alpha_1}\Sigma_{\alpha_1}V_{\alpha_1}^T & U_{\alpha_2}\Sigma_{\alpha_2}V_{\alpha_2}^T & \cdots & U_{\alpha_k}\Sigma_{\alpha_k}V_{\alpha_k}^T \end{pmatrix} - U\Sigma V^T||_F \\
\leq& ||(A_{\alpha_1} - U_{\alpha_1}\Sigma_{\alpha_1}V_{\alpha_1}^T, A_{\alpha_2} - U_{\alpha_2}\Sigma_{\alpha_2}V_{\alpha_2}^T, \cdots, A_{\alpha_k} - U_{\alpha_k}\Sigma_{\alpha_k}V_{\alpha_k}^T)||_F \qquad (3.25) \\
& + ||\begin{pmatrix} U_{\alpha_1}\Sigma_{\alpha_1}V_{\alpha_1}^T & U_{\alpha_2}\Sigma_{\alpha_2}V_{\alpha_2}^T & \cdots & U_{\alpha_k}\Sigma_{\alpha_k}V_{\alpha_k}^T \end{pmatrix} - U\Sigma V^T||_F \qquad (3.26)
\end{aligned}
$$

We first consider the first term in the right-hand-side in (3.25). By Remark 2.1.1, the Frobenius norm squared of block matrices can simply be added to form the norm of the full matrix. Hence, we have

$$
||\begin{pmatrix} A_{\alpha_1} - U_{\alpha_1}\Sigma_{\alpha_1}V_{\alpha_1}^T & A_{\alpha_2} - U_{\alpha_2}\Sigma_{\alpha_2}V_{\alpha_2}^T & \cdots & A_{\alpha_k} - U_{\alpha_k}\Sigma_{\alpha_k}V_{\alpha_k}^T \end{pmatrix}||_F = \sqrt{\sum_{i=1}^{k} \epsilon_{\alpha_i}^2}.
$$

The second term in (3.26) is the reformulation of the nodal error of the root node ρ_T where, $U = U_{\rho_T}$, $\Sigma = \Sigma_{\rho_T}$, $V^T = V_{\rho_T}^T$, and A_{ρ_T} is the approximation of matrix A formed from the aggregation of blocks under column partitioning,

$$
||\underbrace{\begin{pmatrix} U_{\alpha_1}\Sigma_{\alpha_1}V_{\alpha_1}^T & U_{\alpha_2}\Sigma_{\alpha_2}V_{\alpha_2}^T & \cdots & U_{\alpha_k}\Sigma_{\alpha_k}V_{\alpha_k}^T \end{pmatrix}}_{A_{\rho_T}} - U\Sigma V^T||_F = ||A_{\rho_T} - U_{\rho_T}\Sigma_{\rho_T}V_{\rho_T}^T||_F. \qquad (3.27)
$$

By Remark 3.3.1, this term is

$$
\epsilon_{\rho_T} = \| \begin{pmatrix} U_{\alpha_1}\Sigma_{\alpha_1} & U_{\alpha_2}\Sigma_{\alpha_2} & \cdots & U_{\alpha_k}\Sigma_{\alpha_k} \end{pmatrix} - U_{\rho_T}\Sigma_{\rho_T}\tilde{V}_{\rho_T}^T\|_F,
$$

where $\tilde{V}_{\rho_T}^T$ is the matrix of right singular vectors of the approximate SVD of $\begin{pmatrix} U_{\alpha_1}\Sigma_{\alpha_1} & U_{\alpha_2}\Sigma_{\alpha_2} & \cdots & U_{\alpha_k}\Sigma_{\alpha_k} \end{pmatrix}$.

Combining both terms in (3.25) and (3.26), we have

$$
||A - U\Sigma V^T||_F \leq \epsilon_{\rho_T} + \sqrt{\sum_{i=1}^{k} \epsilon_{\alpha_i}^2}.
$$

For the case of row partitioning, we consider a partition of matrix A into row of block matrices $\{A_{\alpha_i}\}_{i=1}^k$.

Without loss of generality, blocks are indexed such that adjacent indices belong to

column-adjacent matrices. By the triangle inequality, we have

$$
\begin{aligned}
||A - U\Sigma V^T||_F &= \left|\left| \begin{pmatrix} A_{\alpha_1} \\ A_{\alpha_2} \\ \vdots \\ A_{\alpha_k} \end{pmatrix} - U\Sigma V^T \right|\right|_F \\
&\leq \left|\left| \begin{pmatrix} A_{\alpha_1} - U_{\alpha_1}\Sigma_{\alpha_1}V_{\alpha_1}^T \\ A_{\alpha_2} - U_{\alpha_2}\Sigma_{\alpha_2}V_{\alpha_2}^T \\ \vdots \\ A_{\alpha_k} - U_{\alpha_k}\Sigma_{\alpha_k}V_{\alpha_k}^T \end{pmatrix} \right|\right|_F + \left|\left| \begin{pmatrix} U_{\alpha_1}\Sigma_{\alpha_1}V_{\alpha_1}^T \\ U_{\alpha_2}\Sigma_{\alpha_2}V_{\alpha_2}^T \\ \vdots \\ U_{\alpha_k}\Sigma_{\alpha_k}V_{\alpha_k}^T \end{pmatrix} - U\Sigma V^T \right|\right|_F .
\end{aligned}
$$

Similarly, using the definition of the Frobenius norm and reformulation in terms of root nodes, we obtain (3.23). □

As shown in the proof of Lemma 3.3.2, we are able to propagate nodal errors of the leaf nodes to a higher level along with its aggregation error. Using this concept, we are able to derive two error bounds for arbitrary tree structures :

- the hierarchical error bound,
- and the flat error bound.

Theorem 3.3.1 (Hierarchical error bound)**.** *Let $A \in \mathbb{R}^{m\times n}$ be a matrix with its approximation $U\Sigma V^T$ computed using the HASVD. Given a rooted tree T that defines blocks $\{A_\alpha\}_{\alpha\in\mathcal{N}_T}$ of A and the nodal errors $\{\epsilon_\alpha\}$ of the nodes $\alpha \in \mathcal{N}_T$, the Frobenius error of the approximation $U\Sigma V^T$ of A is bounded by*

$$
||A - U\Sigma V^T||_F \leq \sum_{\alpha\in\mathcal{N}_T\setminus\mathcal{L}_T} \epsilon_\alpha + \sum_{\alpha'\in\mathcal{N}_T\setminus\mathcal{L}_T} \sqrt{\sum_{\beta\in\mathcal{C}_T(\alpha')\cap\mathcal{L}_T} \epsilon_\beta^2} =: \varepsilon(A) \tag{3.28}
$$

As shown in Theorem 3.3.1, the Frobenius norm error of the HASVD is bounded by the sum of nodal errors over all non-leaf nodes, as well as a hierarchical sum over the leaf nodes. The second term involves square roots of sums of squared nodal errors for leaf nodes grouped under the same non-leaf parent. This structure reflects the hierarchical nature of the tree, where leaf node contributions are aggregated according to their position in the tree, thereby giving the bound its hierarchical character.

We will now show a concrete example of how the error bound is constructed from a tree T that defines a HASVD computation.

In Figure 3.4, we have nodes $\rho, 2, 3, \ldots, 10$. Now suppose the HASVD computation of some matrix A with tree T produces nodal errors $\epsilon_\rho, \epsilon_2, \ldots, \epsilon_{10}$.

To compute the bound, we start by identifying the non-leaf nodes, which are nodes $\rho, 3$, and 5.

Using (3.28), we can write out the first term, which is

$$
\sum_{\alpha\in\mathcal{N}_T\setminus\mathcal{L}_T} \epsilon_\alpha = \epsilon_\rho + \epsilon_3 + \epsilon_5.
$$

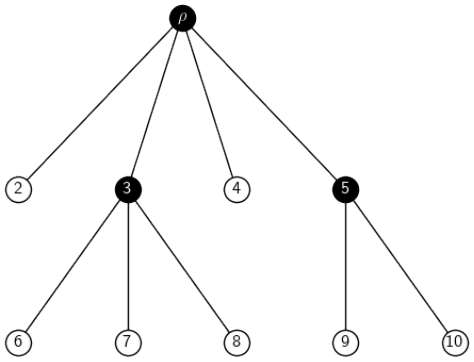


Figure 3.4: A tree with nodal errors for each node.

Afterwards, we identify and group the leaf nodes according to its parent non-leaf nodes. We first again identify the non-leaf nodes and get for the second term of (3.28) to be

$$\sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} \sqrt{\sum_{\beta \in \mathcal{C}_T(\alpha') \cap \mathcal{L}_T} \epsilon_\beta^2} = \sqrt{\sum_{\beta \in \mathcal{C}_T(\rho) \cap \mathcal{L}_T} \epsilon_\beta^2} + \sqrt{\sum_{\beta' \in \mathcal{C}_T(3) \cap \mathcal{L}_T} \epsilon_{\beta'}^2} + \sqrt{\sum_{\beta'' \in \mathcal{C}_T(5) \cap \mathcal{L}_T} \epsilon_{\beta''}^2}$$

Furthermore, we observe that the nodes 2 and 4 are children of the node ρ, nodes $6, 7$, and 8 are children of node 3, and the nodes 9 and 10 are children of node 5. Therefore, we can insert the nodal errors of the leaf nodes into (3.3.1) and obtain

$$\sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} \sqrt{\sum_{\beta \in \mathcal{C}_T(\alpha') \cap \mathcal{L}_T} \epsilon_\beta^2} = \sqrt{\epsilon_2^2 + \epsilon_4^2} + \sqrt{\epsilon_6^2 + \epsilon_7^2 + \epsilon_8^2} + \sqrt{\epsilon_9^2 + \epsilon_{10}^2}.$$

In total, the Frobenius error of the HASVD of A is bounded by

$$||A - U\Sigma V^T||_F \leq \epsilon_\rho + \epsilon_3 + \epsilon_5 + \sqrt{\epsilon_2^2 + \epsilon_4^2} + \sqrt{\epsilon_6^2 + \epsilon_7^2 + \epsilon_8^2} + \sqrt{\epsilon_9^2 + \epsilon_{10}^2}.$$

We shall now prove Theorem 3.3.1.

Proof of Theorem 3.3.1. We will prove by induction on the number of non-leaf nodes in a tree. Let $n = |\mathcal{N}_T \setminus \mathcal{L}_T| \in \mathbb{N}_0$ be the count of non-leaf nodes. We then show that for each n, the statement holds for every tree with exactly n non-leaf nodes.

For the base case $n = 0$ (single node tree), it is trivial to see that we have the nodal error of the root node ρ.

Additionally, for the case $n = 1$ (distributed tree) follows from Lemma 3.3.2.

Now assume that

$$||A - U\Sigma V^T||_F \leq \sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha + \sum_{\alpha' \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \sqrt{\sum_{\beta \in \mathcal{C}_{T'}(\alpha') \cap \mathcal{L}_{T'}} \epsilon_\beta^2} \tag{3.29}$$

is true for all rooted trees T' and $|\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}| = k-1$, that define the blockss $\{A_\alpha\}$ of A and the nodal $\{\epsilon_\alpha\}$ of the nodes $\alpha \in \mathcal{N}_{T'}$.

Suppose we now consider a rooted tree T'' with root ρ'' and and $|\mathcal{N}_{T''} \setminus \mathcal{L}_{T''}| = k$. Then we can obtain a subtree T' of T'' with $|\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}| = k-1$ by turning a non-leaf node $\beta' \in \mathcal{N}_{T''} \setminus \mathcal{L}_{T''}$ whose children are only leaf nodes into a leaf node itself (i.e. $\beta' \in \mathcal{L}_{T'}$) as shown in Figure 3.5.

Therefore, we can choose a T' such that it is obtained by picking a non-leaf node β' in T'' such that the exclusion of leaf nodes of T'' from the children of β' is empty $\mathcal{C}_{T''}(\beta') \setminus \mathcal{L}_{T''} = \emptyset$, and then omitting the children of β' to obtain T'.

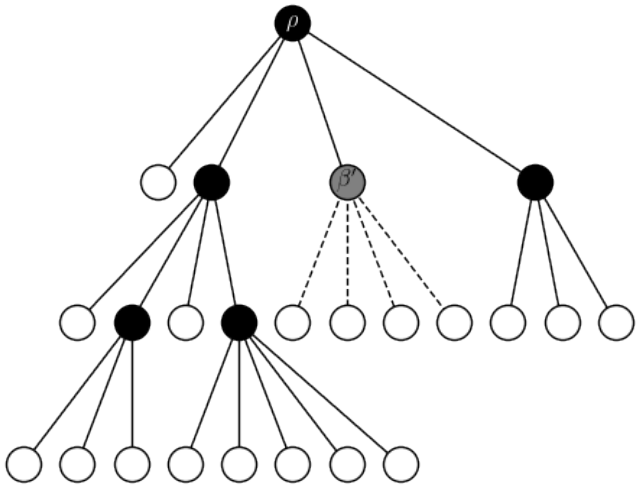


Figure 3.5: Relation between T'' and subtree T'. As shown T'' can be constructed from T' by turning leaf node β' to a parent node. The tree under β' is a distributed HASVD tree.

Since we concluded that the choice of subtree T' corresponds to turning a parent in T'' of exclusively leaf nodes into a leaf node itself, this implies that the HASVD Frobenius error of the whole matrix A in the rooted tree T'' is the HASVD Frobenius error of T' with one leaf node β' expanded.

Using the form (3.29) for T'' and inserting the error for the node β', we then obtain

$$
\begin{aligned}
||A - U\Sigma V^T||_F \leq & \sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha \\
& + \sum_{\alpha' \in (\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}) \setminus \hat{\beta}} \sqrt{\sum_{\delta \in \mathcal{C}_{T'}(\alpha') \cap \mathcal{L}_{T'}} \epsilon_\delta^2} + \sqrt{\sum_{\beta \in \mathcal{C}_{T'}(\hat{\beta}) \cap \mathcal{L}_{T'} \setminus \{\beta'\}} \epsilon_\beta^2 + \varepsilon(A_{\beta'})^2} \\
\leq & \sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha \\
& + \sum_{\alpha' \in (\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}) \setminus \hat{\beta}} \sqrt{\sum_{\delta \in \mathcal{C}_{T'}(\alpha') \cap \mathcal{L}_{T'}} \epsilon_\delta^2} + \sqrt{\sum_{\beta \in \mathcal{C}_{T'}(\hat{\beta}) \cap \mathcal{L}_{T'} \setminus \{\beta'\}} \epsilon_\beta^2} + \varepsilon(A_{\beta'}),
\end{aligned}
$$

where $\hat{\beta}$ is the parent node of β. The second line was obtained using the inequality $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for $a, b \geq 0$.

By Lemma 3.3.2, we know that the error of this block partitioning is

$$\varepsilon(A_{\beta'}) = \epsilon_{\beta'} + \sqrt{\sum_{\gamma \in \mathcal{C}_{T''}(\beta')} \epsilon_\gamma^2}.$$

Inserting the above to the Frobenius error of HASVD for rooted tree T'',

$$\begin{aligned} ||A - U\Sigma V^T||_F \leq & \sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha + \epsilon_{\beta'} + \sum_{\alpha' \in (\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}) \setminus \hat{\beta}} \sqrt{\sum_{\delta \in \mathcal{C}_{T'}(\alpha') \cap \mathcal{L}_{T'}} \epsilon_\delta^2} \\ & + \sqrt{\sum_{\beta \in (\mathcal{C}_{T'}(\hat{\beta}) \cap \mathcal{L}_{T'}) \setminus \{\beta'\}} \epsilon_\beta^2} + \sqrt{\sum_{\gamma \in \mathcal{C}_{T''}(\beta')} \epsilon_\gamma^2}. \end{aligned}$$

To merge the sums, we note that with $(\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}) \cup \{\beta'\} = \mathcal{N}_{T''} \setminus \mathcal{L}_{T''}$ (see Figure 3.5) one can combine the first two terms,

$$\sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha + \epsilon_{\beta'} = \sum_{\alpha \in \mathcal{N}_{T''} \setminus \mathcal{L}_{T''}} \epsilon_\alpha.$$

Furthermore, $(\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}) \setminus \hat{\beta} = (\mathcal{N}_{T''} \setminus \mathcal{L}_{T''}) \setminus \{\beta', \hat{\beta}\}$ since β' simply turned into a non-leaf node. Additionally, leaf nodes which are children of non-leaf nodes which are not parents of β' will remain unchanged in T''. Therefore, $\mathcal{C}_{T'}(\alpha') \cap \mathcal{L}_{T'} = \mathcal{C}_{T''}(\alpha') \cap \mathcal{L}_{T''}$ for all $\alpha' \in (\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}) \setminus \{\hat{\beta}\}$. This transforms the third term,

$$\sum_{\alpha' \in (\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}) \setminus \hat{\beta}} \sqrt{\sum_{\delta \in \mathcal{C}_{T'}(\alpha') \cap \mathcal{L}_{T'}} \epsilon_\delta^2} = \sum_{\alpha' \in (\mathcal{N}_{T''} \setminus \mathcal{L}_{T''}) \setminus \{\hat{\beta}, \beta'\}} \sqrt{\sum_{\delta \in \mathcal{C}_{T''}(\alpha') \cap \mathcal{L}_{T''}} \epsilon_\delta^2} \tag{3.30}$$

Finally, we have $(\mathcal{C}_{T'}(\hat{\beta}) \cap \mathcal{L}_{T'}) \setminus \{\beta'\} = \mathcal{C}_{T''}(\hat{\beta}) \cap \mathcal{L}_{T''}$ for the parent node $\hat{\beta}$ of β' and $\mathcal{C}_{T''}(\beta') = \mathcal{C}_{T''}(\beta') \cap \mathcal{L}_{T''}$ because children of β' are only leaf nodes, so that the final two terms become,

$$\sqrt{\sum_{\beta \in (\mathcal{C}_{T'}(\hat{\beta}) \cap \mathcal{L}_{T'}) \setminus \{\beta'\}} \epsilon_\beta^2} + \sqrt{\sum_{\gamma \in \mathcal{C}_{T''}(\beta')} \epsilon_\gamma^2} = \sqrt{\sum_{\beta \in \mathcal{C}_{T''}(\hat{\beta}) \cap \mathcal{L}_{T''}} \epsilon_\beta^2} + \sqrt{\sum_{\gamma \in \mathcal{C}_{T''}(\beta') \cap \mathcal{L}_{T''}} \epsilon_\gamma^2}. \tag{3.31}$$

The combination of (3.30) and (3.31) lead to including $\hat{\beta}$ and β' in the outer sum in (3.30),

$$\sum_{\alpha' \in \mathcal{N}_{T''} \setminus \mathcal{L}_{T''}} \sqrt{\sum_{\delta \in \mathcal{C}_{T''}(\alpha') \cap \mathcal{L}_{T''}} \epsilon_\delta^2}.$$

The total sum of the terms in the inequality is then,

$$||A - U\Sigma V^T||_F \leq \sum_{\alpha \in \mathcal{N}_{T''} \setminus \mathcal{L}_{T''}} \epsilon_\alpha + \sum_{\alpha' \in \mathcal{N}_{T''} \setminus \mathcal{L}_{T''}} \sqrt{\sum_{\beta \in \mathcal{C}_{T''}(\alpha') \cap \mathcal{L}_{T''}} \epsilon_\beta^2}. \tag{3.32}$$

Hence by induction, Theorem 3.3.1 holds for all rooted tree T with $|\mathcal{N}_T \setminus \mathcal{L}_T| = n \in \mathbb{N}_0$ that defines the partitioning of arbitrary blocks. □

Theorem 3.3.1 has shown that while there is an error bound, the structure of the nodal errors along the leaf nodes is rather complex. Where one has to keep track of the specific children of each non-leaf node and their respective leaf nodes to compute the error bound. This may be cumbersome, especially for larger trees with many non-leaf nodes that have many leaf nodes as children.

To this end, we will present the flat error bound. This error bound not only tighter, but as will be shown in Section 3.3.2 has a corresponding error prescription that can scale with its block shape along the leaf nodes.

Theorem 3.3.2 (Flat HASVD error bound)**.** *Let $A \in \mathbb{R}^{m\times n}$ be a matrix with its approximation $U\Sigma V^T$ computed using the HASVD. Given a rooted tree T that defines the blocks $\{A_\alpha\}$ of A and the nodal errors $\{\epsilon_\alpha\}$ of the nodes $\alpha \in \mathcal{N}_T$, the Frobenius error of the approximation $U\Sigma V^T$ of A is bounded by*

$$||A - U\Sigma V^T||_F \leq \sum_{\beta\in\mathcal{N}_T\setminus\mathcal{L}_T} \epsilon_\beta + \sqrt{\sum_{\gamma\in\mathcal{L}_T} \epsilon_\gamma^2} =: \varepsilon(A). \tag{3.33}$$

The error bound in Theorem 3.3.2 is a tighter bound. This can be seen by taking Figure 3.4 as another example for the flat bound.

In this case, we simply need to identify the non-leaf nodes, which are nodes $\rho, 3$, and 5, and the leaf nodes, which are nodes $2, 4, 6, 7, 8, 9$, and 10.

We can then compute the error bound as follows,

$$||A - U\Sigma V^T||_F \leq \epsilon_\rho + \epsilon_3 + \epsilon_5 + \sqrt{\epsilon_2^2 + \epsilon_4^2 + \epsilon_6^2 + \epsilon_7^2 + \epsilon_8^2 + \epsilon_9^2 + \epsilon_{10}^2}.$$

By comparing the structure of the flat error bound with the hierarchical error bound and using triangle inequality, we see that

$$\underbrace{\sqrt{\epsilon_2^2 + \epsilon_4^2 + \epsilon_6^2 + \epsilon_7^2 + \epsilon_8^2 + \epsilon_9^2 + \epsilon_{10}^2}}_{\text{flat bound}} \leq \underbrace{\sqrt{\epsilon_2^2 + \epsilon_4^2} + \sqrt{\epsilon_6^2 + \epsilon_7^2 + \epsilon_8^2} + \sqrt{\epsilon_9^2 + \epsilon_{10}^2}}_{\text{hierarchical bound}},$$

we know that this error bound is tighter.

To prepare for the proof, we start with the following simple statement in algebra.

Lemma 3.3.3. *For all $s, t, v, \epsilon \geq 0$, we have $s + \sqrt{t + (\epsilon + \sqrt{v})^2} \leq s + \epsilon + \sqrt{t + v}$.*

Proof. Since $s, t, v, \epsilon \geq 0$, we have

$$\begin{aligned}
& v && \leq t + v. \\
\implies & 2\epsilon\sqrt{v} && \leq 2\epsilon\sqrt{t+v}. \\
\implies & t + \epsilon^2 + v + 2\epsilon\sqrt{v} = t + (\epsilon + \sqrt{v})^2 && \leq t + \epsilon^2 + v + 2\epsilon\sqrt{t+v} = (\epsilon + \sqrt{t+v})^2 \\
\implies & \sqrt{t + (\epsilon + \sqrt{v})^2} && \leq \epsilon + \sqrt{t+v} \\
\implies & s + \sqrt{t + (\epsilon + \sqrt{v})^2} && \leq s + \epsilon + \sqrt{t+v}.
\end{aligned}$$

□

With the lemma above, we now have the tool to prove Theorem 3.3.2.

Proof of Theorem 3.3.2. Similar to the proof of Theorem 3.3.1, we use induction over all possible set of trees with non-leaf nodes $|\mathcal{N}_T \setminus \mathcal{L}_T| = n \in \mathbb{N}_0$.

For base case $n = 0$ (single node tree), it is trivial to see that we have the nodal error of the root node ρ.

Additionally, case $n = 1$ (distributed tree) follows from Lemma 3.3.2.

We assume that

$$||A - U\Sigma V^T||_F \leq \sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha + \sqrt{\sum_{\beta \in \mathcal{L}_{T'}} \epsilon_\beta^2}$$

holds for all rooted tree T' with root ρ' and $|\mathcal{N}_{T'} \setminus \mathcal{L}_{T'}| = k - 1$, that defines blocks $\{A_\alpha\}$ of A as well as nodal errors $\{\epsilon_\alpha\}$ of node $\alpha \in \mathcal{N}_{T'}$.

If we consider again that all rooted tree T'' with root ρ'' and and $|\mathcal{N}_{T''} \setminus \mathcal{L}_{T''}| = k$, we have that all possible T' is a subtree of some T''.

In the same manner of proof of Theorem 3.3.1, we know that the relation between T' and T'' is by turning a single leaf node $\beta' \in \mathcal{L}_{T'}$ into a parent node of a distributed HASVD tree, transforming $\epsilon_{\beta'}$ into the matrix error $\varepsilon(A_{\beta'})$.

This implies we have the error bound

$$\begin{aligned} ||A - U\Sigma V^T||_F &\leq \sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha + \sqrt{\sum_{\beta \in \mathcal{L}_{T'} \setminus \{\beta'\}} \epsilon_\beta^2 + \varepsilon(A_{\beta'})^2} \\ &= \sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha + \sqrt{\sum_{\beta \in \mathcal{L}_{T'} \setminus \{\beta'\}} \epsilon_\beta^2 + \left(\epsilon_{\beta'} + \sqrt{\sum_{\gamma \in \mathcal{C}_{T'}(\beta')} \epsilon_\gamma^2}\right)^2}. \end{aligned}$$

Using Lemma 3.3.3, we set $s := \sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha$, $t := \sum_{\beta \in \mathcal{L}_{T'} \setminus \{\beta'\}} \epsilon_\beta^2$, $\epsilon := \epsilon_{\beta'}$, and $v := \sum_{\gamma \in \mathcal{C}_{T'}(\beta')} \epsilon_\gamma^2$ and get

$$\begin{aligned} &\sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha + \sqrt{\sum_{\beta \in \mathcal{L}_{T'} \setminus \{\beta'\}} \epsilon_\beta^2 + \left(\epsilon_{\beta'} + \sqrt{\sum_{\gamma \in \mathcal{C}_{T'}(\beta')} \epsilon_\gamma^2}\right)^2} \\ \leq &\sum_{\alpha \in \mathcal{N}_{T'} \setminus \mathcal{L}_{T'}} \epsilon_\alpha + \epsilon_{\beta'} + \sqrt{\sum_{\beta \in \mathcal{L}_{T'} \setminus \{\beta'\}} \epsilon_\beta^2 + \sum_{\gamma \in \mathcal{C}_{T'}(\beta')} \epsilon_\gamma^2} \\ = &\sum_{\alpha \in \mathcal{N}_{T''} \setminus \mathcal{L}_{T''}} \epsilon_\alpha + \sqrt{\sum_{\beta \in \mathcal{L}_{T''}} \epsilon_\beta^2}, \end{aligned}$$

where the final line can be done since β' is now a non-leaf node in T'' and children of β' $\mathcal{C}_T(\beta')$ completes the set of leaf nodes of T'' from leaf nodes of T' (see Figure 3.5 for clarity).

We then also obtain the tighter bound of the HASVD of A in T'', which is

$$||A - U\Sigma V^T||_F \leq \sum_{\alpha \in \mathcal{N}_{T''} \setminus \mathcal{L}_{T''}} \epsilon_\alpha + \sqrt{\sum_{\beta \in \mathcal{L}_{T''}} \epsilon_\beta^2}.$$

Hence by induction, Theorem 3.3.2 holds for all rooted tree T with $|\mathcal{N}_T \setminus \mathcal{L}_T| = n \in \mathbb{N}_0$ that defines the partitioning of arbitrary blocks. □

3.3.2 Error tolerance prescription

It can be observed that error bounds can be derived from propagation of errors. With this, it is then useful to prescribe nodal errors appropriately such that a desired tolerance ϵ^* of HASVD of some matrix A can be achieved. Therefore, it is important to compute nodal errors of smaller SVD problems a priori such that it aggregates to the desired tolerance.

A simple but illustrative strategy for doing this is the hierarchical error prescription, described below.

Remark 3.3.1 (Hierarchical error prescription)**.** *Let $A \in \mathbb{R}^{m\times n}$ be a matrix with its HASVD $A \approx U\Sigma V^T$. Given a rooted tree T that defines blocks $\{A_\alpha\}$ of A with shapes $m_\alpha \times n_\alpha$ as well as nodal errors $\{\epsilon_\alpha\}$ of node $\alpha \in \mathcal{N}_T$, let $\epsilon^* > 0$ and $\gamma \in \mathcal{N}_T \setminus \mathcal{L}_T$.*

The nodal error tolerances ϵ_{ρ_T}, ϵ_α, ϵ_β, where $\alpha \in \mathcal{C}_T(\gamma) \setminus \mathcal{L}_T$, $\beta \in \mathcal{C}_T(\gamma) \cap \mathcal{L}_T$, is given by

$$\epsilon_{\rho_T} := \omega\epsilon^*, \quad \epsilon_\alpha := (1-\omega)\epsilon^* \frac{m_\alpha n_\alpha}{m_\gamma n_\gamma} \frac{1}{|\mathcal{N}_T \setminus \mathcal{L}_T|},$$
$$\epsilon_\beta := \frac{(1-\omega)\epsilon^*}{\sqrt{|\mathcal{C}_T(\gamma) \cap \mathcal{L}_T|}} \sum_{\delta \in \mathcal{C}_T(\gamma)\cap\mathcal{L}_T} \frac{m_\delta n_\delta}{m_\gamma n_\gamma} \frac{1}{|\mathcal{N}_T \setminus \mathcal{L}_T|},$$

where $0 \le \omega \le 1$ is an arbitrary parameter, will lead to HASVD Frobenius error

$$||A - U\Sigma V^T|| \le \epsilon^*.$$

In Remark 3.3.1 the nodal error of the root node ρ_T is scaled by a factor ω, a control parameter that can be chosen to decide how much of the error is tolerated in the root in comparison to the aggregation process in HASVD.

High value of ω implies that the approximation is mostly determined by the end of the HASVD process (in the root node), while low value of ω means that the approximation is mostly determined by the beginning of the HASVD process (in the nodes subordinate to the root).

Moreover, the shape $m_\alpha \times n_\alpha$ serves as a scaling factor for each nodal errors, where larger blocks have larger nodal errors. This is because larger blocks are more difficult to approximate, hence the larger nodal errors.

Proof of Remark 3.3.1. Using Theorem 3.3.1 and inserting the error prescription we

get

$$
\begin{aligned}
||A - U\Sigma V^T||_F \leq & \sum_{\alpha \in \mathcal{N}_T \setminus \mathcal{L}_T} \epsilon_\alpha + \sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} \sqrt{\sum_{\beta \in \mathcal{C}_T(\alpha') \cap \mathcal{L}_T} \epsilon_\beta^2} \\
= & \epsilon_{\rho_T} + \sum_{\alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}} \epsilon_\alpha + \sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} \sqrt{\sum_{\beta \in \mathcal{C}_T(\alpha') \cap \mathcal{L}_T} \epsilon_\beta^2} \\
= & \omega\epsilon^* + \sum_{\alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}} (1-\omega)\epsilon^* \frac{m_\alpha n_\alpha}{m_{\mathcal{P}(\alpha)} n_{\mathcal{P}(\alpha)}} \frac{1}{|\mathcal{N}_T \setminus \mathcal{L}_T|} \\
& + \sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} \sqrt{\sum_{\beta \in \mathcal{C}_T(\alpha') \cap \mathcal{L}_T} \frac{(1-\omega)^2 \epsilon^{*2}}{|\mathcal{C}_T(\alpha') \cap \mathcal{L}_T|} \left(\sum_{\delta \in \mathcal{C}_T(\alpha') \cap \mathcal{L}_T} \frac{m_\delta n_\delta}{m_{\alpha'} n_{\alpha'}} \frac{1}{|\mathcal{N}_T \setminus \mathcal{L}_T|} \right)^2},
\end{aligned}
$$

where $\mathcal{P}(\alpha)$ indicates the parent node of node α.

The third term in the final line can be simplified as the sum over β is actually independent of the index due to the definition of the error prescription. Which means the sum just cancels out the denominator which is the cardinality of $\mathcal{C}_T(\alpha') \cap \mathcal{L}_T$.

Therefore, we have

$$
\begin{aligned}
||A - U\Sigma V^T||_F \leq & \omega\epsilon^* + \sum_{\alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}} (1-\omega)\epsilon^* \frac{m_\alpha n_\alpha}{m_{\mathcal{P}(\alpha)} n_{\mathcal{P}(\alpha)}} \frac{1}{|\mathcal{N}_T \setminus \mathcal{L}_T|} \\
& + \sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} (1-\omega)\epsilon^* \sum_{\delta \in \mathcal{C}_T(\alpha') \cap \mathcal{L}_T} \frac{m_\delta n_\delta}{m_{\alpha'} n_{\alpha'}} \frac{1}{|\mathcal{N}_T \setminus \mathcal{L}_T|}. \\
= & \omega\epsilon^* + \frac{(1-\omega)\epsilon^*}{|\mathcal{N}_T \setminus \mathcal{L}_T|} \left[\sum_{\alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}} \frac{m_\alpha n_\alpha}{m_{\mathcal{P}(\alpha)} n_{\mathcal{P}(\alpha)}} \right. \\
& \left. + \sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} \sum_{\delta \in \mathcal{C}_T(\alpha') \cap \mathcal{L}_T} \frac{m_\delta n_\delta}{m_{\alpha'} n_{\alpha'}} \right].
\end{aligned} \tag{3.34}
$$

Using identity (2.22), summation over $\alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}$ can be transformed to summations of non-leaf children of non-leaf nodes, then we can transform the first term of the last line,

$$
\sum_{\alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}} \frac{m_\alpha n_\alpha}{m_{\mathcal{P}(\alpha)} n_{\mathcal{P}(\alpha)}} = \sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} \sum_{\delta \in \mathcal{C}_T(\alpha') \setminus \mathcal{L}_T} \frac{m_\delta n_\delta}{m_{\alpha'} n_{\alpha'}}.
$$

Consequently, the error bound is then the combination of summing of all children of α' (i.e., both leaf and non-leaf nodes). Hence, we have

$$
||A - U\Sigma V^T||_F \leq \omega\epsilon^* + \frac{(1-\omega)\epsilon^*}{|\mathcal{N}_T \setminus \mathcal{L}_T|} \sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} \sum_{\delta \in \mathcal{C}_T(\alpha')} \frac{m_\delta n_\delta}{m_{\alpha'} n_{\alpha'}}. \tag{3.35}
$$

Now observe that for any internal node α', its children (leaf or not) form a complete partition of its block. Thus, the total block shape fraction of its children satisfies

$$\sum_{\delta \in \mathcal{C}_T(\alpha')} \frac{m_\delta n_\delta}{m_{\alpha'} n_{\alpha'}} = 1.$$

Hence, the double sum over children in (3.35) reduces to

$$\sum_{\alpha' \in \mathcal{N}_T \setminus \mathcal{L}_T} 1 = |\mathcal{N}_T \setminus \mathcal{L}_T|.$$

Finally, the error simplifies to

$$\|A - U\Sigma V^T\| \leq \epsilon^* \tag{3.36}$$

as required. □

A corresponding error prescription can also be made for the tighter error bound in Remark 3.3.2.

Remark 3.3.2 (Flat error prescription)**.** *Let $A \in \mathbb{R}^{m \times n}$ be a matrix with its HASVD $A \approx U\Sigma V^T$. We define* ***branching nodes*** *of tree T as*

$$\mathcal{B}_T := \{\alpha \in \mathcal{N}_T \setminus \mathcal{L}_T : \exists \beta \in \mathcal{C}_T(\alpha) \text{ such that } \beta \in \mathcal{N}_T \setminus \mathcal{L}_T\}.$$

Given a rooted tree T that defines blocks $\{A_\alpha\}$ of A with shapes $m_\alpha \times n_\alpha$ as well as nodal errors $\{\epsilon_\alpha\}$ of node $\alpha \in \mathcal{N}_T$, let $\epsilon^* > 0$ and $\gamma \in \mathcal{N}_T \setminus \mathcal{L}_T$.

The nodal error tolerances ϵ_{ρ_T},ϵ_α, ϵ_β, where $\alpha \in \mathcal{C}_T(\gamma) \setminus \mathcal{L}_T$, $\beta \in \mathcal{C}_T(\gamma) \cap \mathcal{L}_T$, is given by

$$\epsilon_{\rho_T} := \omega\epsilon^*, \quad \epsilon_\beta := \frac{(1-\omega)\epsilon^*}{|\mathcal{B}_T| + 1} \sqrt{\frac{m_\beta n_\beta}{m_{\rho_T} n_{\rho_T}}},$$
$$\epsilon_\alpha := \frac{(1-\omega)\epsilon^*}{|\mathcal{B}_T| + 1} \frac{m_\alpha n_\alpha}{\sum_{\alpha' \in \mathcal{C}_T(\gamma) \setminus \mathcal{L}_T} m_{\alpha'} n_{\alpha'}},$$

where $0 \leq \omega \leq 1$ is an arbitrary parameter, will lead to the HASVD Frobenius error

$$||A - U\Sigma V^T|| \leq \epsilon^*.$$

One notice that the tight error prescription has a normalization factor of $|\mathcal{B}_T| + 1$ in the denominator of the nodal errors, that is dependent on number of branching nodes $\mathcal{B}_T$, which are non-leaf nodes that have at least one non-leaf child node. As an example in Figure 3.6. the branching nodes are indicated inside the dotted lines, where node 5 and 6 are excluded as they are non-leaf nodes that do not have non-leaf children.

The error prescription above now has nodal error of leaf node β scale with the shape of its respective blocks relative to $m_{\rho_T} n_{\rho_T}$, which is just the size of matrix A.

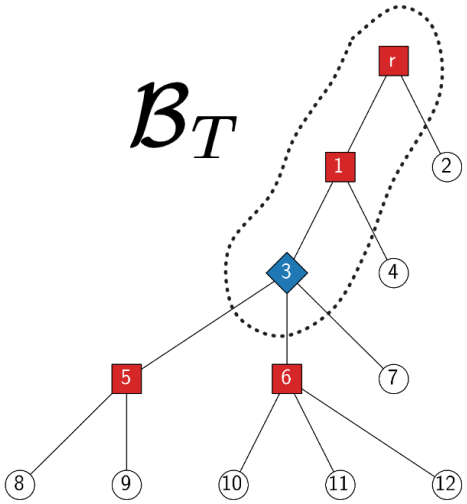


Figure 3.6: Branching nodes $\mathcal{B}_T$ of a tree T indicated inside the dotted lines.

To compensate for this, the nodal error of non-leaf node α is scaled in the denominator with the size of the blocks of its parent nodes when the leaf nodes are excluded. This is represented by the summation over $\mathcal{C}_T(\gamma) \setminus \mathcal{L}_T$ which are just sibling nodes of α that are not leaf nodes.

Proof of Remark 3.3.2. Inserting the error prescription to Theorem 3.3.2, we have

$$
\begin{aligned}
||A - U\Sigma V^T||_F \leq & \sum_{\beta \in \mathcal{N}_T \setminus \mathcal{L}_T} \epsilon_\beta + \sqrt{\sum_{\gamma \in \mathcal{L}_T} \epsilon_\gamma^2} \\
=& \omega\epsilon^* + \sum_{\beta \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}} \frac{(1-\omega)\epsilon^*}{|\mathcal{B}_T| + 1} \frac{m_\beta n_\beta}{\sum_{\beta' \in \mathcal{C}_T(\mathcal{P}(\beta)) \setminus \mathcal{L}_T} m_{\beta'} n_{\beta'}} \\
& + \sqrt{\sum_{\gamma \in \mathcal{L}_T} \left(\frac{(1-\omega)\epsilon^*}{|\mathcal{B}_T| + 1} \right)^2 \frac{m_\gamma n_\gamma}{m_{\rho_T} n_{\rho_T}}} \\
=& \omega\epsilon^* + \frac{(1-\omega)\epsilon^*}{|\mathcal{B}_T| + 1} \left(\sum_{\beta \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}} \frac{m_\beta n_\beta}{\sum_{\beta' \in \mathcal{C}_T(\mathcal{P}(\beta)) \setminus \mathcal{L}_T} m_{\beta'} n_{\beta'}} \right. \qquad (3.37) \\
& \left. + \sum_{\gamma \in \mathcal{L}_T} \frac{m_\gamma n_\gamma}{m_{\rho_T} n_{\rho_T}} \right), \qquad (3.38)
\end{aligned}
$$

where $\mathcal{P}(\beta)$ is the parent node of β.

Since the blocks associated to leaf nodes form the entire matrix A, that means summing $mn = m_{\rho_T} n_{\rho_T} = \sum_{\gamma \in \mathcal{L}_T} m_\gamma n_\gamma$ and therefore the summation term over leaf nodes in (3.38) cancels to unity.

The identity in (2.22) can be used to transform (3.37) to summation over non-leaf

children of non-leaf nodes, i.e. $\alpha \in \mathcal{C}_T(\beta) \setminus \mathcal{L}_T$ for $\beta' \in \mathcal{N}_T \setminus \mathcal{L}_T$. However, node β that are parents of exclusively leaf nodes need to be excluded, as this will give zero on both the numerator and denominator. This issue can be fixed by indexing over the branching nodes $\mathcal{B}_T$ instead of $\mathcal{N}_T \setminus \mathcal{L}_T$, giving us

$$\sum_{\beta \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}} \frac{m_\beta n_\beta}{\sum_{\beta' \in \mathcal{C}_T(\mathcal{P}(\beta)) \setminus \mathcal{L}_T} m_{\beta'} n_{\beta'}} = \sum_{\beta \in \mathcal{B}_T} \frac{\sum_{\alpha \in \mathcal{C}_T(\beta) \setminus \mathcal{L}_T} m_\alpha n_\alpha}{\sum_{\beta' \in \mathcal{C}_T(\beta) \setminus \mathcal{L}_T} m_{\beta'} n_{\beta'}} = |\mathcal{B}_T|.$$

Consequently, we have

$$||A - U\Sigma V^T||_F \leq \omega\epsilon^* + \frac{(1-\omega)\epsilon^*}{|\mathcal{B}_T| + 1}(|\mathcal{B}_T| + 1) = \epsilon^*.$$

□

To conclude, the hierarchical and the flat error prescription strategies provide systematic ways to assign local error tolerances across the nodes of a tree structure, ensuring that the overall approximation error when using HASVD remains within a predetermined prescribed tolerance ϵ^*. These error bounds will serve as guidelines for setting tolerances in the numerical experiments presented in Chapter 5.

Chapter 4

Application and schemes of HASVD

In this chapter, we will present specific application of HASVD.

Section 4.1 start with a brief introduction to simple variants of trees and partitioning schemes that can be used in the HASVD framework. This include variants inspired by HAPOD as well as its HASVD derivatives that forms the basis of more complex tree and partitioning structures.

Furthermore, in Section 4.2 we will present possible pruning schemes that can be used in the HASVD framework, particularly for Hankel and Toeplitz matrices, with a specific focus in reducing computational and memory costs. This will mostly involve pruning through symmetries found within the matrix.

4.1 HASVD Trees and Partitioning Schemes

In this section, we will discuss the different variants of trees and partitioning schemes that can be used in the HASVD framework. In Section 3.3.2, we have established that a control parameter $0 \leq \omega \leq 1$ can be used to control the tolerance of the approximate SVD. This parameter plays a crucial role in reducing the complexity of the HASVD problem. The higher we allow the error tolerance in the aggregation process in HASVD, particularly for low ω, the more we can reduce the size of the matrices involved in the computation. This reduction allows us to perform the approximate SVD on smaller matrices in the higher hierarchy of the tree, which in turn reduces the overall computational cost.

Therefore, the choice of tree structure and partitioning scheme is also important in the HASVD framework. The tree structure determines how the data is divided into smaller subproblems, while the partitioning scheme defines how the data is distributed across the tree. Different tree structures and partitioning schemes can lead to different computational costs and error bounds in the HASVD framework.

With this in mind, we will explore different tree schemes and partitioning structures that can be used in the HASVD framework. We start with the basic tree structures inspired by HAPOD applications and then explore more complex tree structures

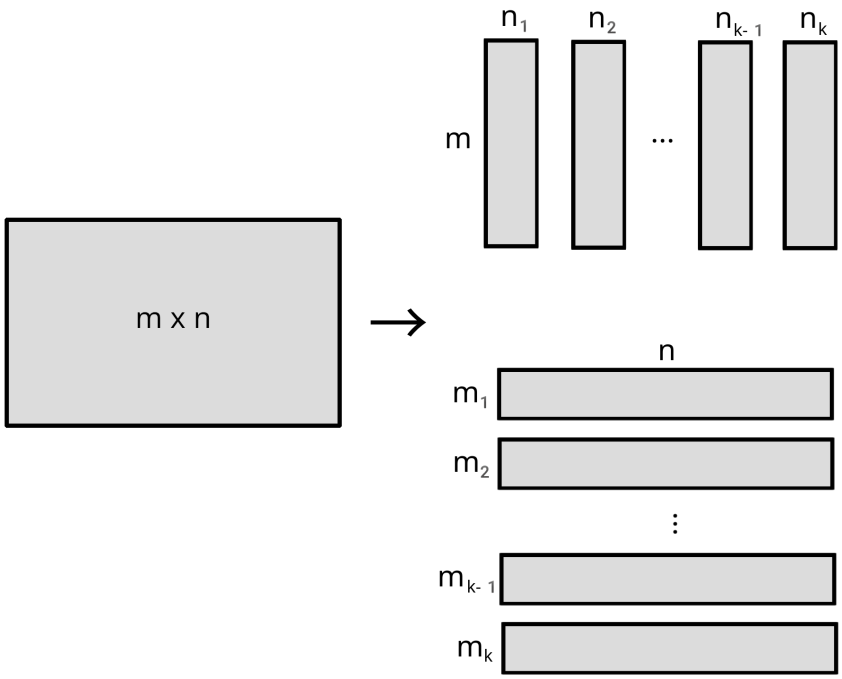


Figure 4.1: Linear column (top) and row (bottom) partitioning scheme.

that can be used to reduce the computational cost of the HASVD algorithm.

4.1.1 Linear structures

For the simplest case of partitioning, we can apply a simple column or row partitioning scheme that is similar to the one used in HAPOD [15].

This partitioning scheme is based on linearly dividing the data into exclusively row adjacent or column adjacent blocks as shown in Figure 4.1.

In this case, the associated tree structure is the:

- distributed tree (DIST),
- and the incremental tree (INC).

The structure of the trees is illustrated in Figure 4.2. In the distributed tree, the computation of approximations using HASVD is performed in a single aggregation step from the blocks associated with the leaf nodes. In contrast, the incremental tree involves multiple aggregation steps, progressively combining intermediate results along the tree.

While the distributed tree may be more efficient in terms of computational cost, as it has less amount of aggregation steps. The incremental tree may be more efficient in terms of memory usage, as it requires less memory to store the aggregated blocks in each step. For example, an incremental process can be used for data compression in an MPI (Message Passing Interface) model[1], such that each HASVD of the sub-block is computed via parallelized algorithms [15].

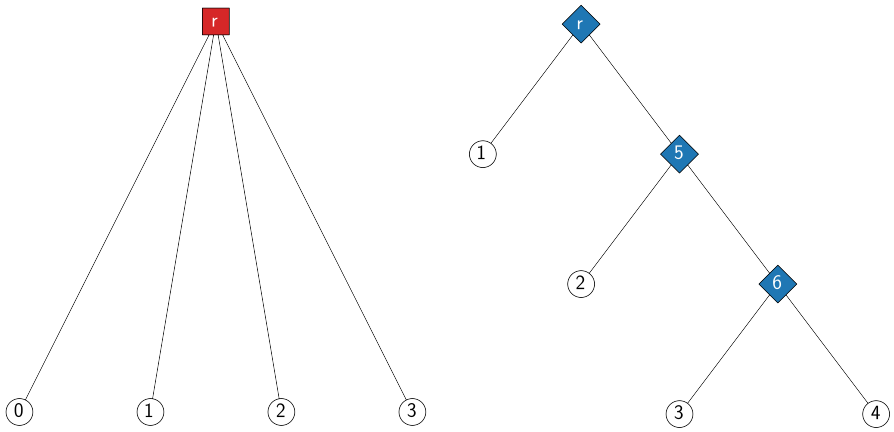


Figure 4.2: Column-distributed (left) and row-incremental (right) tree.

Using Remark 3.3.2, we can also see a comparison of the prescribed error for each tree. Suppose we have a column-patritioned matrix $A \in \mathbb{R}^{m\times n}$ divided into k blocks $A \in \mathbb{R}^{m\times n_i}$ where $i = 1, \ldots, k$ and apply the HASVD.

In the case of a distributed tree, there are only have leaf nodes and the root nodes, which means the non-root non-leaf nodes does not need to be taken into account.

Therefore, for the flat error prescription, we obtain the tolerances of each leaf nodes as

$$\epsilon_i = (1-\omega)\epsilon^* \sqrt{\frac{n_i}{n}},$$

where i is the index of the leaf node.

On the other hand, for an incremental tree, suppose that we have $i = 1, \ldots, k$ as indices of the leaf nodes and $j = k+1, \ldots, 2k-1$ as indices of the non-leaf nodes excluding the root node. Then, the tolerances for each leaf node can be expressed as

$$\epsilon_i = \frac{(1-\omega)\epsilon^*}{k-2} \sqrt{\frac{n_i}{n}}, \tag{4.1}$$

$$\epsilon_j = \frac{(1-\omega)\epsilon^*}{k-2}. \tag{4.2}$$

From (4.1) and (4.2), it is evident that the incremental tree enforces smaller error tolerances at each step of the aggregation process. As a consequence, the incremental tree may lead to a more accurate approximation of the SVD, but at the cost of more aggregation steps and thus higher computational cost.

4.1.2 Two-level bidirectional linear structures

We can extend the tree structures of Section 4.1.1 for a block partitioning scheme as indicated in Figure 4.3.

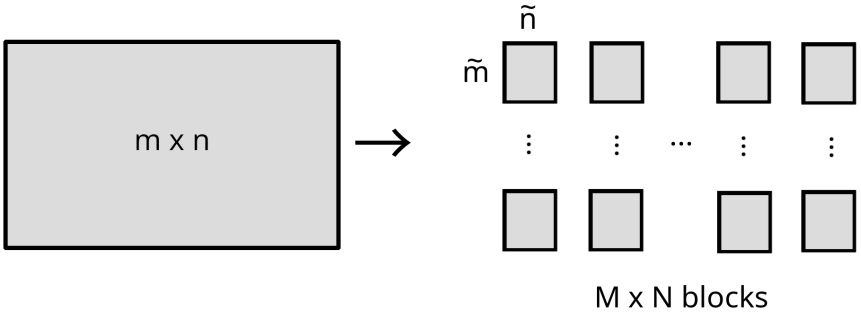


Figure 4.3: Block partitioning scheme.

In this case, we propose a two-level tree structure, where a column aggregation along the leaf nodes are followed by a row aggregation along the aggregated blocks, or vice-versa, as shown in Figure 4.4.

The proposed aggregation strategies consist of:

- the distributed two-level bidirectional tree (TLBD)
- and incremental two-level bidirectional tree (TLBI).

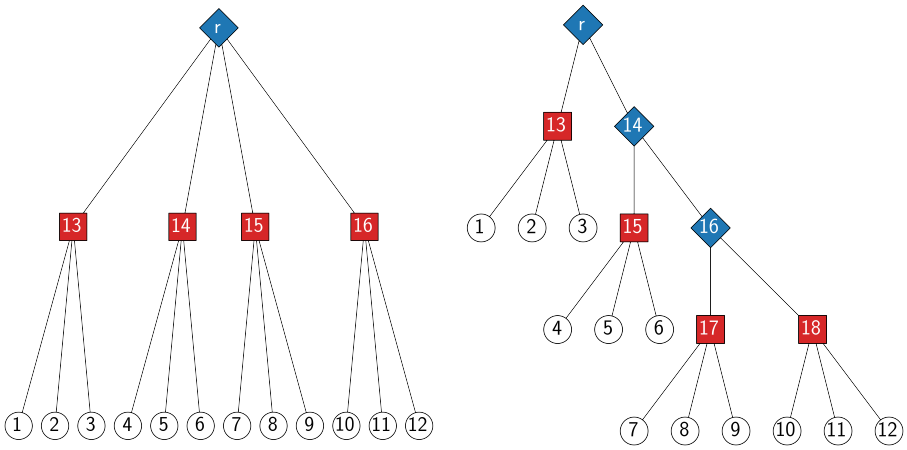


Figure 4.4: Two-level bidirectional distributed (left) and incremental (right) tree.

These tree structures are of particular interest as they allow partitioning of a matrix into smaller sub-blocks. This enables SVD approximations to be computed on smaller, more manageable portions of the matrix, thereby capturing finer structural details. Such an approach may prove especially useful in Section 4.2, where structured matrices such as Hankel and Toeplitz matrices are considered.

Suppose that we now have a matrix $A \in \mathbb{R}^{m \times n}$ divided into $M \times N$ blocks $A_i \in \mathbb{R}^{\tilde{m} \times \tilde{n}}$ for $i = 1, \ldots, MN$, we can use Remark 3.3.2 again and find the following error tolerances:

1. Distributed, where first level is horizontal and second level is vertical:

$$\epsilon_\beta = (1-\omega)\epsilon^* \sqrt{\frac{1}{MN}} \text{ for } \beta \in \mathcal{L}_T,$$
$$\epsilon_\alpha = (1-\omega)\epsilon^* \frac{1}{N} \text{ for } \alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}.$$

2. Distributed, where first level is vertical and second level is horizontal:

$$\epsilon_\beta = (1-\omega)\epsilon^* \sqrt{\frac{1}{MN}} \text{ for } \beta \in \mathcal{L}_T,$$
$$\epsilon_\alpha = (1-\omega)\epsilon^* \frac{1}{M} \text{ for } \alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \{\rho_T\}.$$

3. Incremental, where first level is horizontal and second level is vertical:

$$\epsilon_\beta = \frac{(1-\omega)\epsilon^*}{N-1} \sqrt{\frac{1}{MN}} \text{ for } \beta \in \mathcal{L}_T,$$
$$\epsilon_\alpha = \frac{(1-\omega)\epsilon^*}{N-1} \frac{1}{i} \text{ for } \alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \mathcal{B}_T \text{ for } i = 1, \ldots, N,$$
$$\epsilon_\gamma = \frac{(1-\omega)\epsilon^*}{N-1} \frac{N-i}{N-i+1} \text{ for } \gamma \in \mathcal{B}_T \setminus \{\rho_T\} \text{ for } i = 1, \ldots, N-1.$$

4. Incremental, where first level is vertical and second level is horizontal:

$$\epsilon_\beta = \frac{(1-\omega)\epsilon^*}{M-1} \sqrt{\frac{1}{MN}} \text{ for } \beta \in \mathcal{L}_T,$$
$$\epsilon_\alpha = \frac{(1-\omega)\epsilon^*}{M-1} \frac{1}{i} \text{ for } \alpha \in (\mathcal{N}_T \setminus \mathcal{L}_T) \setminus \mathcal{B}_T \text{ for } i = 1, \ldots, M,$$
$$\epsilon_\gamma = \frac{(1-\omega)\epsilon^*}{M-1} \frac{M-i}{M-i+1} \text{ for } \gamma \in \mathcal{B}_T \setminus \{\rho_T\} \text{ for } i = 1, \ldots, M-1.$$

4.2 Hankel and Toeplitz matrices in HASVD

We have established in Section 4.1 that the choice of tree structure and partitioning scheme is crucial in the HASVD framework. In this section, we will explore the application of HASVD to Hankel and Toeplitz matrices. Recurrent structures in Hankel and Toeplitz matrices make them suitable for HASVD, as we can prune computations in the aggregation process.

In this section we will develop potential strategies of exploiting the internal structures of Hankel and Toeplitz matrices. This will correspond mostly to pruning with respect to both block recurrence and transpose symmetry of the Hankel matrix.

It is important to note that by Remark 2.1.3, we can transform a Hankel matrix $H \in \mathbb{R}^{m \times n}$ to a Toeplitz matrix $T \in \mathbb{R}^{m \times n}$ (and vice versa) by a multiplication of an exchange matrix J_m and J_n, such that $T = J_m H J_n$. Therefore, we will be using a Hankel matrix to treat the structure, eventhough treatment of a Toeplitz matrix is analogous to the ones shown below.

4.2.1 Block-recurrence pruning

As reviewed in Section 2.1, the elements of Hankel matrices are constant along its skew-diagonals and elements of Toeplitz matrices are constant along its diagonals.

This inherent structural property induces symmetries in the elements and blocks within these matrices. Consequently, repeating blocks appear in both Hankel and Toeplitz matrices, which translates to computational pruning opportunities at the leaf levels in a HASVD aggregation tree.

To illustrate, we consider a Hankel matrix $H \in \mathbb{R}^{6\times 5}$, where we have a skew-diagonal symmetry,

$$H = \begin{pmatrix} a & b & \mathbf{c} & \mathbf{d} & \mathbf{e} \\ b & c & \mathbf{d} & \mathbf{e} & \mathbf{f} \\ \mathbf{c} & \mathbf{d} & \mathbf{e} & f & g \\ \mathbf{d} & \mathbf{e} & \mathbf{f} & g & h \\ e & f & g & h & i \\ f & g & h & i & j \end{pmatrix}. \tag{4.3}$$

This example exhibits a **block-wise recurrence**, where specific choices of blocks recur. For example $H[1:2, 3:5]$ recurs in $H[3:4, 1:3]$ as color coded in (4.3).

This observation motivates the identification of unique blocks when performing a linear partitioning of Hankel and Toeplitz square matrices into sub-blocks.

Suppose a Hankel matrix $A \in \mathbb{R}^{n\times n}$ is partitioned into $N \times N$ blocks. Then, can be expressed as,

$$A = \begin{pmatrix} A_1 & A_2 & A_3 & \cdots & A_N \\ A_2 & A_3 & A_4 & & A_{N+1} \\ A_3 & A_4 & A_5 & & A_{N+2} \\ \vdots & & & \ddots & \vdots \\ A_N & A_{N+1} & A_{N+2} & \cdots & A_{2N} \end{pmatrix}.$$

Computing an approximation of A via HASVD requires constructing an aggregation tree that connects each partitioned block to a corresponding leaf node via a block-to-leaf mapping.

Throughout the aggregation process, we conventionally treat each blocks as independent HASVD computations leading to the computation of N^2 approximate SVDs. However, since there are block recurrences, we can effectively reduce it to $2N$ computations. After computing each partitioned block, we can reuse approximate SVD of each blocks in the aggregation steps of the HASVD trees.

Moreover, this concept generalizes by merging individual square blocks into larger rectangular blocks. Owing to the Hankel matrix structure, such rectangular blocks recur one block row downward and one block column to the left, as depicted in Figure 4.5.

Aggregating such a matrix can then be done by using a two-level bidirectional tree. However, tagging leaf nodes that are identical we can prune tasks within the HASVD algorithm. For the example in Figure 4.5, this is demonstrated in Figure 4.6. In this case, we have node 1_i being associated to block A_1, node 2_i to block A_2, and node 3_i to block A_3.

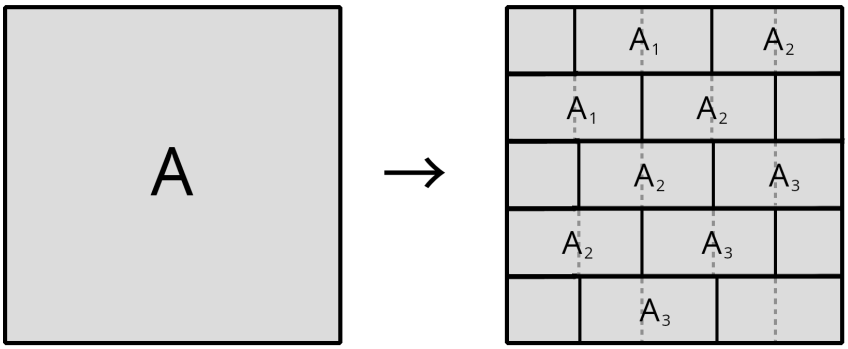


Figure 4.5: Square blocks transformed into recurring rectangular blocks.

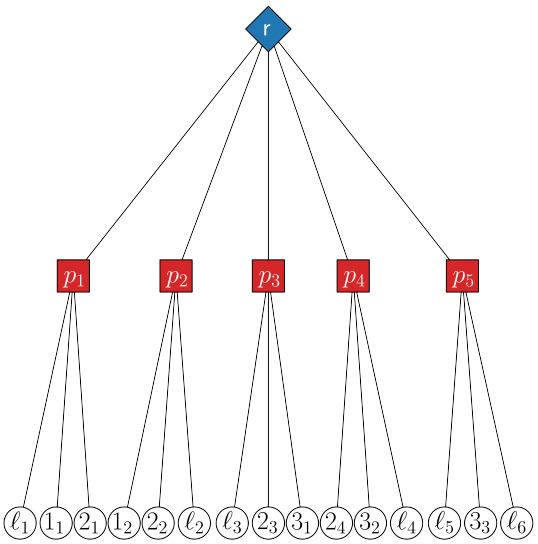


Figure 4.6: Two-level bidirectional distributed tree of rectangular recurring blocks, with numbered nodes being pruned blocks while ℓ_i are non-recurring blocks.

4.2.2 Transpositional-recurrence pruning

Now, we consider another recurrence in Hankel and Toeplitz matrices, which is the **transpositional-recurrence**. This can be seen using the same matrix in (4.3). We now have

$$H = \begin{pmatrix} a & b & \mathbf{c} & \mathbf{d} & \mathbf{e} \\ b & c & \mathbf{d} & \mathbf{e} & \mathbf{f} \\ \mathbf{c} & \mathbf{d} & e & f & g \\ \mathbf{d} & \mathbf{e} & f & g & h \\ \mathbf{e} & \mathbf{f} & g & h & i \\ f & g & h & i & j \end{pmatrix}, \tag{4.4}$$

where some blocks of H coincide exactly to the transposition of another block. In the case of (4.4), we have $H[1:2, 3:5] = H[3:5, 1:2]^T$. This type of recurrence is more useful in Hankel and Toeplitz matrices, because we are not restricted to square

blocks, but also rectangular blocks.

Similar to the treatment of block recurrence, we can reuse computations of approximate SVD of partitioned blocks to represent a transpose of another uncomputed block. In this case, suppose $B = U\Sigma V^T$ is a approximate SVD of a block in a Hankel or Toeplitz matrix, where another block in the same matrix has the relation $\tilde{B} = B^T$. We can represent SVD of $\tilde{B} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$ as,

$$\tilde{B} = \tilde{U}\tilde{\Sigma}\tilde{V}^T = V\Sigma U^T = (U\Sigma V^T)^T = B^T, \tag{4.5}$$

where $\tilde{U} = V$, $\tilde{V}^T = U^T$, and $\tilde{\Sigma} = \Sigma$.

Extending this further, we can construct a partitioning scheme for a square Hankel matrix where such a transpositional recurrence can be used. Given a square matrix $A \in \mathbb{R}^{n\times n}$, we can partition this into square blocks $D_i \in \mathbb{R}^{\frac{n}{N}\times\frac{n}{N}}$ for $i = 1, \ldots, N$ and rectangular blocks $A_j \in \mathbb{R}^{\frac{(N-j)n}{N}\times\frac{n}{N}}$ along with its transpose A_j^T for $j = 1, \ldots, N-1$. Such a partitionaing scheme can be seen in Figure 4.7

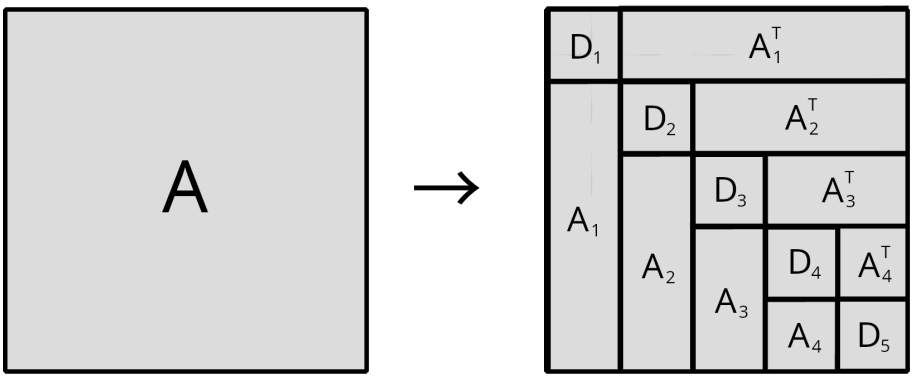


Figure 4.7: Transpositional partitioning scheme.

Along the leaf level, this can effectively reduce the number of SVD computations to $2N - 1$ computations, consisting of N blocks along the diagonals and $N - 1$ rectangular blocks on the lower triangular part of the matrix.

Additionally, we can also implement method of snapshots (as discussed in Section 2.2.3) as an SVD method within the HASVD step gain potential speedup on tall-skinny-matrices in the rectangular blocks.

Constructing a aggregation tree for this partitioning scheme would lead to a **two-level alternating incremental tree (TLAI)** as shown in Figure 4.8.

In this tree, incremental aggregation of the entire matrix is alternated between row and column aggregation. In here each nodes represented by integer $i = 1, 2, 3, 4$ are computations of the matrix A_i. On the other hand node d_j for $j = 1, \ldots, 5$ are tasks in HASVD process that is associated to the computation of approximate SVD of blocks D_j in Figure 4.7.

Execution of nodes 1^T, 2^T, 3^T, and 4^T can be pruned by using approximate SVD results form nodes 1, 2, 3, and 4. Furthermore, we can also perform method of snapshots computations on nodes $1, 2, 3, 4$ to see posibility using an $\frac{n}{N} \times \frac{n}{N}$ eigenvalue problem to compute the approximate SVD of blocks A_i for $i = 1, 2, 3, 4$.

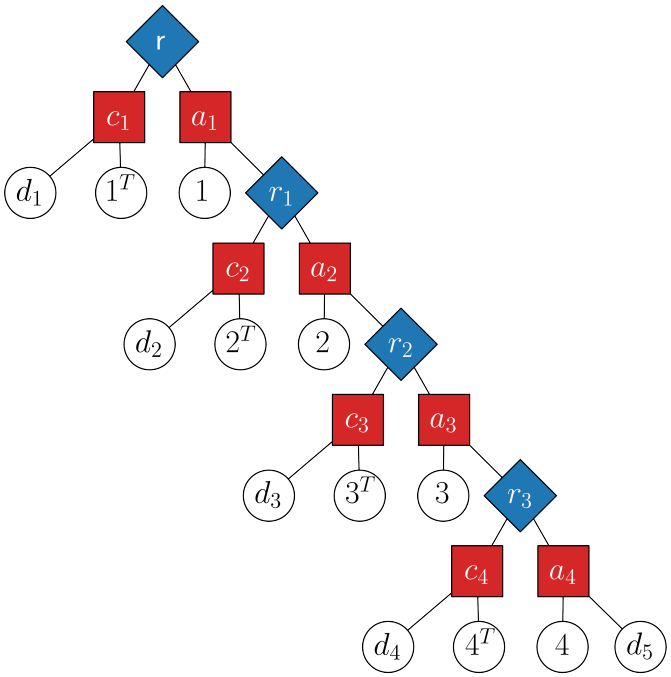


Figure 4.8: two-level alternating incremental tree (TLAI).

4.2.3 Implementation of pruning

For clarity, this section describes how pruning can be algorithmically implemented in HASVD, specifically focusing on block-recurrence pruning and transpositional-recurrence pruning.

The implementation used in this thesis follows Algorithm 6.

The key idea is to introduce a cache key map $\kappa : \mathcal{N}_T \to \mathbb{Z}$, which assigns each node in the aggregation tree a unique identifying integer. This identifier is shared among nodes that exhibit a particular symmetry of interest—such as skew-diagonal symmetry (for Hankel matrices) or transpose symmetry, as discussed in Subsections 4.2.1 and 4.2.2.

During execution, as each node $\alpha \in \mathcal{N}_T$ is explored, the algorithm checks whether its cache key $\kappa(\alpha)$ is contained in the set of previously seen keys $\mathcal{K}$.

If it is, a corresponding SVD is retrieved from the SVD cache

$$M(\kappa(\alpha)) = (U_\alpha, \Sigma_\alpha, V_\alpha^T) \in \mathcal{M}, \tag{4.6}$$

Otherwise, the node proceeds with the standard HASVD procedure, up until full SVD computation. The result is then stored in the cache $\mathcal{M}$ via a designated Store() function.

The choice of the Store() function depends on the type of symmetry being exploited. This function is critical, as it may manipulate the computed SVD before caching it.

In this thesis, we consider:

- block-recurrence store function: $\text{Store}_{\text{block}}(U, \Sigma, V^T) := (U, \Sigma, V^T)$,

- and traspositional-recurrence store function: $\text{Store}_{\text{trans}}(U, \Sigma, V^T) := (V, \Sigma, U^T)$.

which is derived from the transpose relation in (4.5).

Algorithm 6: HASVD with pruning

Input: Matrix $A \in \mathbb{R}^{m \times n}$, rooted tree T with block-to-leaf map D, local tolerances $\epsilon : \mathcal{N}_T \to \mathbb{R}^{\geq 0}$, and aggregation direction labels $x : \mathcal{N}_T \to \{0, 1\}$, cache key $\kappa : \mathcal{N}_T \to \mathbb{Z}$, cache-storing function Store().

Output: Approximate matrix of left singular vectors $U \in \mathbb{R}^{m \times r}$, approximate matrix of right singular vectors $V^T \in \mathbb{R}^{r \times n}$, and approximate singular value matrix $\Sigma \in \mathbb{R}^{r \times r}$

$\mathcal{S} := \emptyset$
$(\mathcal{K}, \mathcal{M}) := (\emptyset, \emptyset)$, Intialize SVD cache.
while *node* $\alpha \neq$*None or* $\mathcal{S} = \emptyset$ **do**
  $\alpha, \mathcal{S} := \text{Traverse}(\rho_T, \mathcal{S})$.
  if $\kappa(\alpha) \in \mathcal{K}$ **then**
    $U_\alpha, \Sigma_\alpha, V_\alpha^T := M(\kappa(\alpha))$, Access cache with key.
  else
    if $\alpha \in \mathcal{L}_T$ **then**
      $A_\alpha := D^{-1}(\alpha)$
      $U_\alpha \Sigma_\alpha V_\alpha^T := A_\alpha$
    else
      $U_\alpha, \Sigma_\alpha, V_\alpha^T := \text{AggregatedSVD}(\alpha, \{(U_\beta, \Sigma_\beta, V_\beta^T)\}_{\beta \in \mathcal{C}_T(A)}, T, x(\alpha))$
    $M(\kappa(\alpha)) := \text{Store}(U_\alpha, \Sigma_\alpha, V_\alpha^T)$, Stores SVD cache.
    $\mathcal{K} := \mathcal{K} \cup \kappa(\alpha)$
    $\mathcal{M} := \mathcal{M} \cup M(\kappa(\alpha))$
  $r_\alpha := \text{TruncationRank}(\Sigma_\alpha, \epsilon(\alpha))$
  $\Sigma_\alpha := \Sigma_\alpha[: r_\alpha, : r_\alpha]$
  $U_\alpha := U_\alpha[:, : r_\alpha]$
  $V_\alpha^T := V_\alpha^T[: r_\alpha, :]$
return $U := U_{\rho_T}$, $V^T := V_{\rho_T}^T$, $\Sigma := \Sigma_{\rho_T}$

To conclude, Algorithm 6 provides the foundation for the pruning strategies discussed in this section and will be employed in the numerical experiments involving HASVD with symmetry-exploiting tree structures such as TLBD, TLBI, and TLAI.

Chapter 5

Numerical experiment of HASVD

This chapter presents numerical experiments evaluating the HASVD under various configurations and setups. The HASVD will be investigated for three different matrix types: general matrices, Hankel matrices, and block-Hankel matrices.

The chapter is structured as follows. First, Section 5.1 provides a brief description of the experimental setup, including hardware specifications, the libraries used to conduct the experiments, and the methodology of constructing random matrices. This section also defines key metrics for presenting the results.

Section 5.2 forms the core of the chapter, presenting and analyzing the results of low-rank approximations computed by HASVD for general and Hankel matrices. The discussion encompasses approximation accuracy, rank truncation behavior, and algorithm runtime performance.

Finally, Section 5.3 presents preliminary experiments on block-Hankel matrices using real data, establishing a foundation for practical applications.

5.1 Experimental setup

5.1.1 Hardware

All the numerical experiments were performed on a machine with two 12-core Inter® Xeon® Silver 4214R CPUs with 256 RAM running on Ubuntu Linux 22.04.5 LTS provided by the Institute of Fluid Mechanics and Engineering Acoustics at TU Berlin.

5.1.2 HASVD Implementation

The HASVD algorithm was implemented in the Python Programming Language (version 3.10.12) as a package called pyHASVD.

It is developed from a forked implementation of HAPOD in the open-source MOR library pyMOR (version 2024.1.2) [29]. This is mainly done to use its pre-existing `Node()` object to construct trees as well as its established `hapod()` routine which

already has a robust concurrent task assignment system for trees made using asyncIO [36]. NumPy (version 1.26.4) was also used to compute common mathematical routines [13].

Extensions from all mentioned libraries include:

- `Node()` object → `hasvd_Node()` object: specify aggregation direction, store matrix shape, and identify symmetry.
- `hapod()` function → `hasvd()` function: extend POD to SVD methods, include core HASVD procedures, implement SVD cache, and analytic options.

The structure of the library is as follows:

- `pyhasvd.utils.svd` : HASVD and SVD routines, Analytics, and SVD cache maps.
- `pyhasvd.utils.errors` : Error prescriptions.
- `pyhasvd.utils.trees` : Utility function to construct trees, and leaf-to-block maps.
- `pyhasvd.utils.matrix` : Matrix generation and matrix-free representations.

Throughout the experiment, the NumPy SVD routine, `numpy.linalg.svd()`, is used as a benchmark. This function uses **LAPACK gesdd (GESDD)** as its backend. It is a standard SVD routine that implements a divide and conquer method, which is generally effective for computing the SVD of square matrices [35]. It also serves as the base method for SVD computation in HASVD subtasks and is used directly for full SVD computation as a comparison.

Routines found in `hasvd.utils.trees` for generating common tree structures are also developed and used in this experiment. These include:

- **distributed trees (DISTs)** with linear partitioning (`dist_hasvd_tree()`),
- **incremental trees (INCs)** with linear partitioning (`inc_hasvd_tree()`),
- **distributed two-level bidirectional trees (TLBDs)** with block partitioning (`tlbd_hasvd_tree()`),
- **incremental two-level bidirectional trees (TLBIs)** with block partitioning (`tlbi_hasvd_tree()`),
- and **two-level alternating incremental trees (TLAIs)** with block diagonal and rectangular partitioning (`tlai_hasvd_tree()`).

5.1.3 Random matrix generation

Random matrices with specified singular value characteristics are constructed for numerical experiments. In this regard, the random number generation uses the Mersenne Twister engine due to its large autocorrelation period [27, 28]. In particular, a NumPy implementation of Mersenne Twister engine (`numpy.random.MT19937()`) with seed 42 will be used, unless specified otherwise [13].

For general matrices, this can be done by specifying a sequence of singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ and constructing two unitary matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ from the orthogonalization of two random matrices of the same dimensions.

Then, a random matrix $A \in \mathbb{R}^{m\times n}$ of rank r is given by,

$$U \begin{pmatrix} \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r) & 0_{r,n-r} \\ 0_{m-r,r} & 0_{m-r,n-r} \end{pmatrix} V^T. \tag{5.1}$$

For the purpose of the numerical experiment, the singular value decay is chosen to follow a geometric progression, where for a given matrix conditioning parameter κ, the singular value σ_i has the value

$$\sigma_i = \left(\frac{1}{\kappa}\right)^{\frac{i}{r-1}} \quad \text{for } i = 0, \ldots, r-1. \tag{5.2}$$

Unlike general matrices, constructing a random Hankel matrix with a specific singular value decay requires the appropriate sequence of elements $a_1, a_2, \ldots, a_{m+n-1}$ that defines the Hankel matrix $H = \text{Hankel}_{mn}(a_1, \ldots, a_{m+n-1})$ (see (2.11)).

To this end, for a prescribed rank r two sequences has been identified to construct random Hankel matrices, namely:

- **single spectrum analysis linear recurrent formula (SSA-LRF) sequence** given by

$$a_j = \sum_{i=1}^{r} c_i a_{j-i}, \quad \text{for all } j \geq r+1, \tag{5.3}$$

  where $c_1, c_2, \ldots, c_r \in \mathbb{R}$ [9],

- and **free-induction decay (FID) signal sequence** given by

$$a_j = \sum_{i=1}^{r} c_i e^{-\tau_i j}, \quad \text{for } j = 1, 2, \ldots n+m-1, \tag{5.4}$$

  where $c_1, c_2, \ldots, c_r \in \mathbb{R}$ and $\tau_1, \tau_2, \ldots, \tau_r > 0$ [37].

To apply this to random matrix generation, free variables such as a_i for SSA-LRF sequence, and c_i and τ_i for FID sequence, are to be generated randomly. Afterwards, (5.3) and (5.4) are used to determine the value of the entire sequence.

Each of the sequence have distinct characteristics as shown in Figure 5.1. The SSA-LRF sequence is characterized by its ability to construct random Hankel matrices with remarkably accurate rank for a large enough rank prescription. However, this breaks down for small rank prescriptions. On the other hand, while it very difficult to prescribe a rank with FID sequence, its gradual decay makes it suitable to represent extremely rank-deficient matrices.

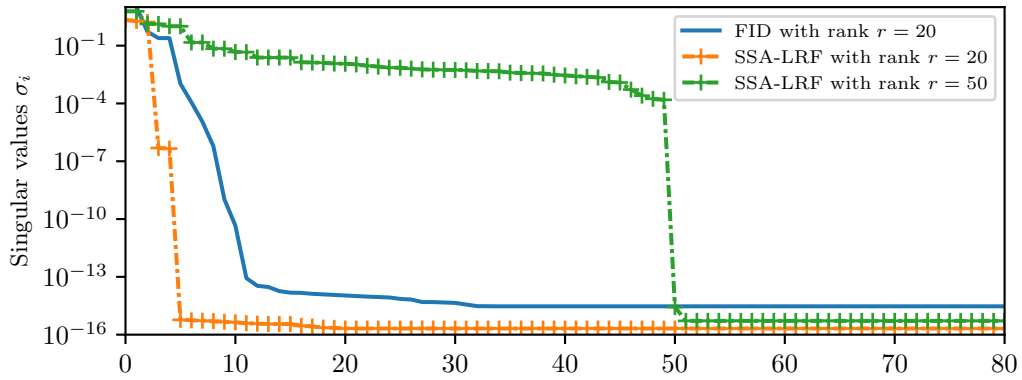


Figure 5.1: Singular value decay of random matrices using SSA-LRF and FID sequence for different rank prescriptions.

An implementation of these Hankel matrix generation was made in pyHASVD with `hasvd.utils.matrix.random_hankel()`. Furthermore, free-matrix representation `array_to_hankel()` are also made to provide storage-efficient access to block components within Hankel matrices. This is done through the information gained from the sequence $a = (a_{1,2}, \ldots, a_{n+m-1})$ defining a Hankel matrix $H = \text{Hankel}_{m,n}(a_{1,2}, \ldots, a_{n+m-1})$.

For a block in H that is defined by row indices i_1, i_2 and column indices j_1, j_2, it has elements

$$(H[i_1 : i_2, j_1 : j_2])_{kl} = a_{i_1+j_1+k+l-2},$$

and can be constructed by an offset of the sequence a,

$$H[i_1 : i_2, j_1 : j_2] = \text{Hankel}_{i_2-i_1+1, j_2-j_1+1}(\underbrace{a_{i_1+j_1}, \ldots, a_{i_2+j_2+2}}_{a'})$$

where we can create a subsequence a', given by

$$a' = (\overbrace{a_{i_1+j_1}, \ldots, a_{i_1+j_1+(i_2-i_1+1)}}^{\text{First column of } H[i_1:i_2, j_1:j_2]}, \underbrace{a_{i_2+j_1+2}, \ldots, a_{i_2+j_1+1+(j_2-j_1+1)}}_{\text{Last row of } H[i_1:i_2, j_1:j_2]}). \tag{5.5}$$

As seen in (5.5), a' can be found by first finding the coordinate of the first element in the block in the Hankel matrix and simply including the length of the Hankel sequence defining the block, i.e. $(i_2 - i_1) + 1 + (j_2 - j_1) + 1$.

This concludes the subsection on random matrix generation. All the techniques introduced here will be used to create test matrices in the numerical experiment. For a detailed discussion and analysis on random Hankel matrix generation, the reader is referred to Appendix B.

5.1.4 Metrics and measurements

There are several metrics that are used to measure performances of HASVD and analyze results.

To measure the accuracy of low-rank approximations computed by HASVD the **relative Frobenius (norm) error** is used, given by

$$\frac{\|A - U_r \Sigma_r V_r^T\|_F}{\|A\|_F}, \tag{5.6}$$

where U_r, Σ_r, and V_r^T are the approximate SVD of matrix A. The relative Frobenius error is computed using the NumPy routine `numpy.linalg.norm()`.

During data presentation, the **relative prescribed error** ϵ is used. Since error prescriptions in Subsection 3.3.2 are formulated in terms of absolute errors, it is important to note that during actual computational runs, the prescribed error ϵ^* scales with the Frobenius norm of the approximated matrix A, i.e.,

$$\epsilon^* = \epsilon \|A\|_F. \tag{5.7}$$

When measuring the rank of low-rank approximations that are computed, there are two distinct metrics.

The first is the **approximation rank** $\tilde{r}$. This is the mathematical rank of low-rank approximation $U_r \Sigma_r V_r^T$ and it is given by $r = \text{rank}(U_r \Sigma_r V_r^T)$. This value is computed using the NumPy routine `numpy.linalg.matrix_rank()`. For matrix $A \in \mathbb{R}^{m \times n}$, the default tolerance for rank cut-off of this routine is $\sigma_1 \max(m, n)u$ where σ_1 is the largest singular value of A and u is the machine epsilon ($u \approx 2.2 \cdot 10^{-16}$ for double floating point precision) [13].

The second is the **truncation rank** r_α, which denotes the number of singular values of the SVD of block A_α that are retained after truncation, based on the nodal error prescription at node $\alpha \in \mathcal{N}_T$ in some rooted tree T. This rank is computed as the length of the accepted singular values, i.e., in Python, `len(S)`.

These two variables can be further processed into interpretable metrics to analyze and quantify the performance of the HASVD algorithm under varying tree structures and parameter settings.

The **approximation ratio** $\frac{\tilde{r}}{r}$ is used to assess the accuracy of rank approximation, where r is the true rank of a matrix A, and $\tilde{r}$ is the rank of its low-rank approximation.

In terms of the ranks approximated by HASVD, this implies:

- if $\frac{\tilde{r}}{r} < 1$, the rank has been underestimated;
- if $\frac{\tilde{r}}{r} > 1$, the rank has been overestimated;
- if $\frac{\tilde{r}}{r} \approx 1$, the rank has been accurately approximated.

Thus, the closer the approximation ratio is to 1, the more accurately the HASVD algorithm approximates the true rank of a matrix.

The **truncation ratio** $\frac{r_\alpha}{n_{\text{block}}}$ is used to quantify the computational cost, where r_α is the truncation rank at node α, and $n_{\text{block}} = \min(m_\alpha, n_\alpha)$ is the smaller dimension of the block $A_\alpha \in \mathbb{R}^{m_\alpha \times n_\alpha}$.

In the context of ranks truncated by HASVD, this implies:

- if $\frac{r_\alpha}{n_{\text{block}}} = 1$, no truncation occurred;
- if $0 < \frac{r_\alpha}{n_{\text{block}}} < 1$, partial truncation occurred;
- if $\frac{r_\alpha}{n_{\text{block}}} = 0$, complete truncation occurred (i.e., the entire block is treated as noise).

Therefore, the closer the truncation ratio is to 0, the greater the reduction in problem size achieved by the HASVD algorithm.

Finally, the **runtime** refers to the duration required to execute the HASVD algorithm. It is measured using Python's standard library function `time.perf_counter()`, which offers the highest available clock resolution for accurately capturing very short computational durations. Unless stated otherwise, reported runtimes represent the average over 10 independent trials.

5.2 Results and discussion

This section presents the findings from numerical experiments evaluating the accuracy, rank truncation behavior, and computational runtime of the HASVD algorithm when applied to random general and Hankel matrices. The results are analyzed in detail to highlight how various factors, including matrix characteristics, partitioning strategies, and tree structures, influence performance and efficiency of HASVD.

5.2.1 Accuracy of general matrix approximations

To evaluate the accuracy of general matrix approximations using HASVD, several aspects are examined:

- the difference in accuracy between various type of error prescriptions,
- the effect of the control parameter ω on computational accuracy,
- and the influence of tree structures on computational accuracy.

To investigate these aspects, test matrix $A \in \mathbb{R}^{2000\times 2000}$ is prepared with rank $r = 20$, where the singular value decay follows the geometric progression in (5.2) with conditioning parameter $\kappa = 10^3$.

This investigation is limited to the distributed tree (DIST) and incremental tree (INC) with column aggregation. Therefore, the matrices are only partitioned into 20 blocks of size 100×2000 for aggregation.

Both flat and hierarchical error prescription are applied, with relative prescribed errors of $\epsilon = 1, 10^{-1}, \ldots, 10^{-15}$. Additionally, control parameters are also varied between $\omega = 0.1, 0.25, 0.5, 0.75, 0.9$.

The result shown in Figure 5.2(a) indicates that there are minimal differences in relative Frobenius error between the type of error prescriptions, while differences between tree structures are more pronounced.

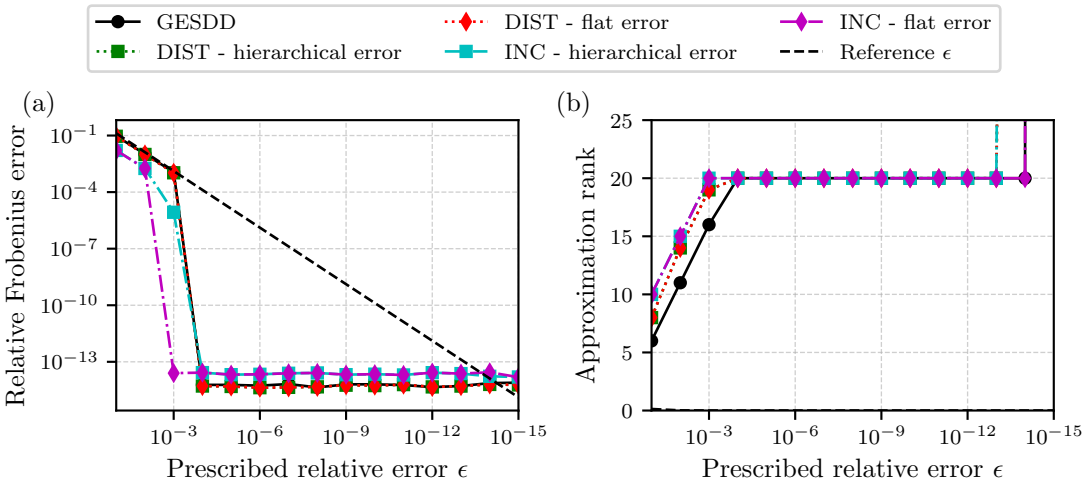


Figure 5.2: Accuracy measurements of test matrix $A \in \mathbb{R}^{2000\times2000}$, rank $r = 20$, approximated with different SVD algorithms. For HASVD, the control parameter is set $\omega = 0.1$. (a) Relative Frobenius errors. (b) Approximation ranks.

For approximating matrix A with HASVD, a sharp drop in error occurs near $\epsilon = 10^{-3}$. This is due to the conditioning of the matrix, as any $\epsilon > 10^{-3}$ simply truncates dominant singular values since the smallest conditioning is $\sigma_r = 10^{-3}\sigma_1$. As observed and expected for $\epsilon < 10^{-3}$, both HASVD and GESDD lead to relative Frobenius errors that level off at machine precision. Note that particularly for HASVD, Frobenius error will have a higher precision bound as it has the form

$$\|A - \tilde{A}\|_F = \sqrt{\sum_{i=r+1}^{n} \sigma_i^2}, \tag{5.8}$$

which means small singular values will be squared and truncated due to precision in HASVD.

It should be noted that for $\epsilon > 1/\kappa$, the smaller relative errors of approximations that are obtained through HASVD are not constrained under a fixed rank. Therefore, the fact that is below GESDD is not a violation of the Eckhardt-Young-Mirsky theorem. This is rather a characteristic of HASVD, to ensure that the final error will remain below the prescribed error. Furthermore, for small $\omega \approx 0.1$, the final nodal error for the root according to error prescriptions (i.e. Remark 3.3.2) is $\omega\epsilon\|A\|_F$, which

may also lead to a larger truncation due to small error tolerance in comparison to GESDD, which has $\epsilon\|A\|_F$.

This behavior can also be observed in the resulting rank of the approximated matrix A for each corresponding relative prescribed error ϵ in Figure 5.2(b). While HASVD may overestimate the rank of matrices even with large error tolerances, it also enables the use of larger error tolerances to obtain relatively accurate matrix ranks, given that the dominant singular value cutoff is well-defined.

Such rank overestimation may be advantageous because using larger error tolerances reduces the complexity of the approximate SVD used in HASVD subproblems, which may potentially lower computational costs.

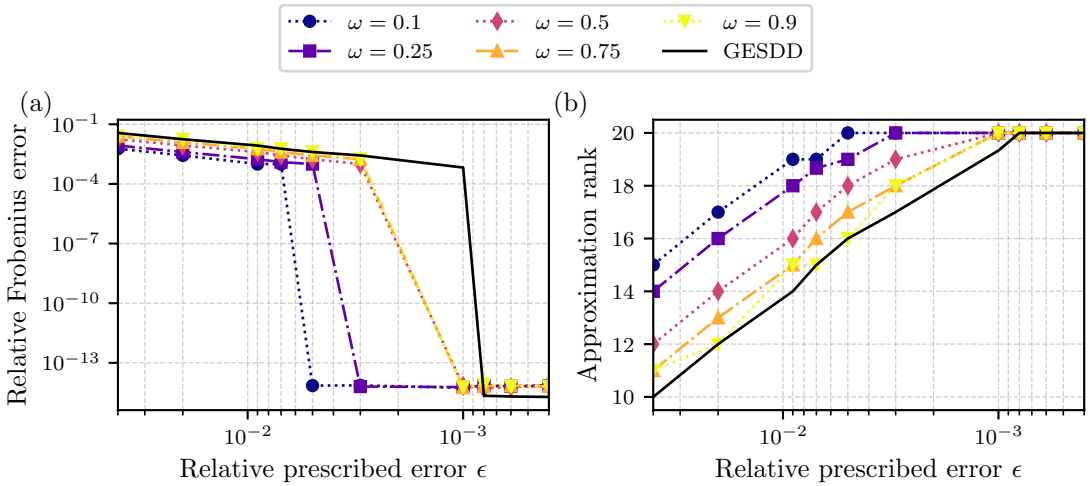


Figure 5.3: Accuracy measurements of test matrix $A \in \mathbb{R}^{2000\times2000}$, rank $r = 20$, approximated with an incremental HASVD for different control parameters ω. (a) Relative Frobenius errors. (b) Approximation ranks.

As shown in Figure 5.3, smaller control parameter values ω also appear to lead to smaller errors and, consequently, larger ranks. There is a clear pattern of convergence towards the standard SVD routines as ω approaches 1. This result is expected, since ω controls the proportion of subproblem tolerances in HASVD as discussed in Subsection 3.3.2.

If $\omega \to 1$, the tolerance prescribed in the non-root nodes, having a factor of $(1 - \omega)\epsilon\|A\|_F$, is reduced to zero. In turn, HASVD will not truncate any values up until the root node, where the error prescription will be $\omega\epsilon\|A\|_F$. HASVD effectively produces a similar result to standard SVD routine with, just with many more aggregation steps.

Finally, it can also be deduced, that in addition to low tolerance, ranks of relatively well-conditioned matrices can also be revealed with low control parameter values, which may also help in further reducing computational cost in rank revealing a matrix.

The numerical experiments demonstrate that HASVD provides accurate low-rank approximations for general matrices, with its performance strongly influenced by the choice of error prescription, tree structure, and control parameter ω. Key ob-

servations include:

- similar accuracy across different error prescriptions,
- a more pronounced effect towards accuracy from the structure of trees,
- a strong influence of the control parameter ω on truncation accuracy, with smaller values leading to tighter approximations and higher ranks,
- and the ability of HASVD to safely overestimate the rank to meet the prescribed error.

5.2.2 Accuracy of Hankel matrix approximations

Expanding upon the results of the experiment with general matrices, additional aspects are examined to evaluate the accuracy of Hankel matrix approximations using HASVD. These include:

- the accuracy of all common tree structures,
- and the effect of different singular value decay characteristics.

To prepare the experiment, test Hankel matrix $A \in \mathbb{R}^{2000\times 2000}$ is prepared using SSA-LRF and FID sequences. For the SSA-LRF sequences, a rank of $r = 400$ is used to avoid the rapid decay in Figure 5.1. To accommodate for possible variations in SSA-LRF-generated matrices, different seeds (42, 43, and 44) are provided to the Mersenne Twister engine.

All trees as mentioned in Subsection 5.1.2 are studied, with row aggregation set at the root node. For the distributed tree (DIST) and incremental tree (INC), matrix A is partitioned into 20 columns of size 2000×100. For the two-level bidirectional tree structures, TLBI and TLBD, matrix A is partitioned into 20×20 blocks with a column aggregation in the second level nodes. As for the two-level alternating incremental tree (TLAI), matrix A is partitioned into 20 diagonal square blocks, 19 standing rectangular blocks, and 19 sitting rectangular blocks (see Figure 4.7).

In the scope of this analysis, we only use the flat error prescription in Remark 3.3.2. Set of computations are observed for relative prescribed errors $\epsilon = 1, 10^{-1}, \ldots, 10^{-14}$, 10^{15} and with control parameter values $\omega = 0.1, 0.9$.

To see the complete result of the experiment, the reader is referred to Appendix C.

As shown in Figure 5.4(a), HASVD shows a similar pattern in Hankel matrices compared to measurements with general matrices in Figure 5.2. However, due to the ill-conditioning of low-rank random Hankel matrices, one can see that the sharp drop is extended to the relative prescribed error of $\epsilon = 10^{-9}$. It can also be seen that the relative Frobenius error of matrix approximations from HASVD with TLAI trees lies between the errors of the distributed TLB tree and the incremental TLB tree.

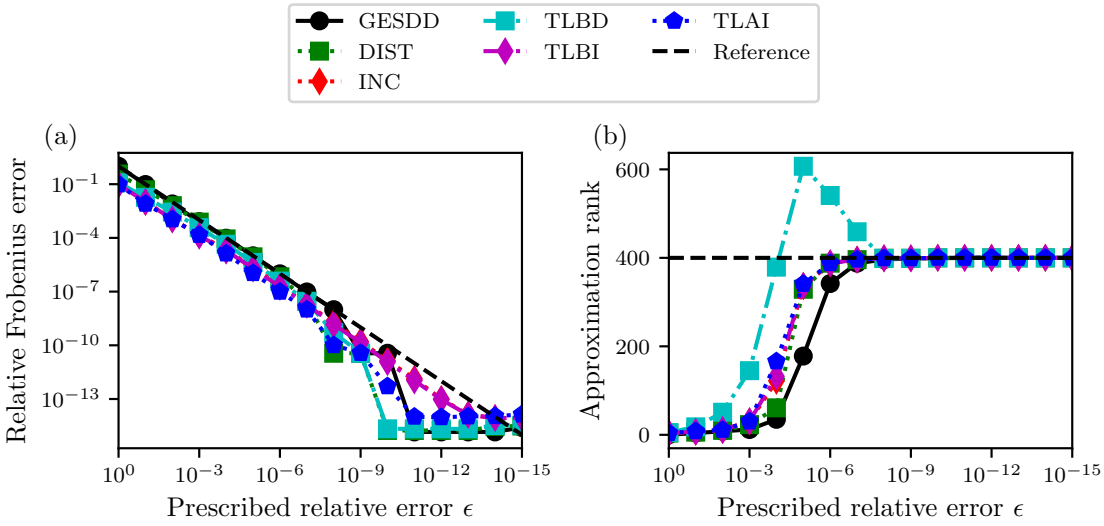


Figure 5.4: Accuracy measurements of test Hankel matrix $A \in \mathbb{R}^{2000\times 2000}$ from SSA-LRF sequence of rank $r = 400$ and seed 42, approximated with different SVD algorithms. For HASVD, the control parameter is set $\omega = 0.1$. (a) Relative Frobenius errors. (b) Approximation ranks.

The approximation ranks in Figure 5.4 (b) also shows the tendency for matrix approximations from HASVD to take a higher approximation rank to fulfill the requirement of the error prescription bound. It is also shown, that the matrix rank is generally achieved at the cutoff point of the singular value.

An interesting point of discussion is the approximation rank produced by distributed TLB trees, which shows an irregular pattern where the computed rank exceeds the expected matrix rank. This behavior consistently appears in low control parameter settings when approximating SSA-LRF-sequenced Hankel matrices using the distributed TLB tree.

This behavior seems to come from numerical precision artifacts that arises from the truncation of singular values and singular vector in a distributed TLB tree. While errors may be extremely small and generally truncated in HASVD, inflation of small singular values in low precision and round-off can occur [3]. Moreover, it is also well known that under noise-contaminated data, a Hankel matrix can become full rank [26]. Specific to a distributed tree that aggregates along both columns and rows, these effects may be introduced and amplified. For example, suppose there is matrix A, such that

$$A = \begin{pmatrix} A_1^T & \cdots & A_i^T & \cdots & A_M^T \end{pmatrix}^T, \text{ where } A_i = \begin{pmatrix} B_{1i} & \cdots & B_{Ni} \end{pmatrix}.$$

Suppose a column aggregation through truncated SVD is done to each A_i for $i = 1, \ldots, M$, due to some the large amount of rows an error matrix E_i may still be introduced after truncation such that $\tilde{A}_i = A_i + E_i = U_i \Sigma_i V_i^T + E_i$. Therefore, in the row aggregation to form ΣV^T one will effectively compute the SVD of

$$\begin{pmatrix} \Sigma_1 V_1^T \\ \vdots \\ \Sigma_i V_i^T \\ \vdots \\ \Sigma_M V_M^T \end{pmatrix} + \begin{pmatrix} U_1^T E_1 \\ \vdots \\ U_i^T E_i \\ \vdots \\ U_M^T E_M \end{pmatrix},$$

which potentially increases the error and inadvertently introduce new directions to the subspace spanned by approximate SVD of A.

To investigate this, an experiment on the control parameter and partitioning is conducted on a SSA-LRF random Hankel matrix $A \in \mathbb{R}^{2000\times 2000}$ of rank $r = 400$. The relative prescribed error is fixed to $\epsilon = 10^{-7}$. A sweep is conducted on the singular value decay of the approximate SVD computed on the root node before a truncation is applied. This sweep is done over control parameter values $\omega = 10^{-3}, 10^{-2}, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99, 0.99$ and the number of partitioning $N = 2, 4, 5, 8, 10, 16, 20, 25, 40, 50$. For the control parameter sweep, the partioning is fixed to $N = 10$ and for the partitioning sweep, the control parameter value is fixed to 0.1.

Finally, a full SVD computation of matrix A is made so that its singular valued decay can serve as a reference value by the form of a contour plot.

It is shown in Figure 5.5(b) that increasing the control parameter value up to $\omega = 0.8$, reduces the singular value decay to the actual rank. This is supported by the fact that a small value of ω corresponds to a large error tolerance in the non-root nodes, with leading factor of $(1-\omega)\epsilon^*$, as shown in Remark 3.3.2 and 3.3.1. This means there is also a larger likelihood to introduce error, as more may be truncated.

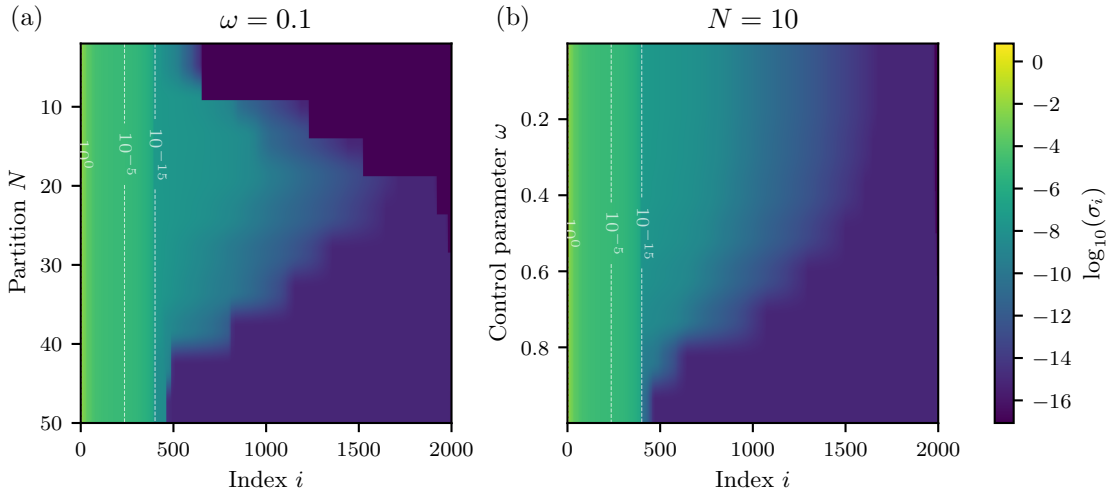


Figure 5.5: Singular value decay curves for matrix approximations of a Hankel matrix generated from an SSA-LRF sequence with seed 42, using HASVD with the distributed TLB tree and relative prescribed error $\epsilon = 10^{-7}$. Dotted lines indicate the true singular value decay of the Hankel matrix for reference. (a) Decay curves for varying numbers of partitions N. (b) Decay curves for varying control parameter values ω.

In Figure 5.5(a), the inflation of singular value relative to the partitioning number peaks at $N = 20$ and shows a dropping pattern in larger partitions. It is important to note that the sudden drops in singular value decays to the lowest precision in partition numbers $N < 30$ are not actual singular values, but rather a lack of any defined singular values. This occur since the matrix evaluated on the root node effectively has a different shape. Suppose there is a column aggregation matrix $B = \begin{pmatrix} V_1\Sigma_1 & V_2\Sigma_2 \end{pmatrix}^T = U_B\Sigma_B V_B^T \in \mathbb{R}^{m\times n}$, where $m < n$. After the column aggregation, one can form $A \approx \hat{U}U_B\Sigma V_B^T \in \mathbb{R}^{m'\times n}$ by Remark 3.2.2. Due to truncations, it may be the case that $m' > m$ since $\Sigma_i \in \mathbb{R}^{r_i\times r_i}$ will be a truncated significantly and $m = r_1 + r_2$. Therefore, even with a full SVD, information about the singular value decay is then limited to the truncation. This also means that singular value decays above $N >= 30$ barely truncate any aggregation matrix, which effectively removes the advantage of HASVD.

This shows that simultaneous aggregation of columns or rows above may cause errors to accumulate, in which larger partition may also cause a perturbation in the singular values. However, it is also important to keep in mind that with larger partitions, the individual blocks become smaller. This increases the likelihood of a block being of full rank, and therefore less likely to be truncated.

Therefore, it is important also maintain a relatively low tolerance, when dealing with Hankel matrices in the characteristic of an SSA-LRF signals to maintain the appropriate truncation value and avoid random artifacts. Furthermore, a detailed perturbation bound may have to be studied, particularly distance of singular values, as [43] indicates that distances between singular values has to be large enough to have a small relative error under additive and multiplicative perturbations.

To summarize the results from the experiments, the accuracy of HASVD on Hankel matrices reveals important sensitivities tree structure, partitioning, and control parameter settings. Several important findings are:

- due to the ill-conditioning of Hankel matrices, relative errors produced by HASVD tend to be smaller, albeit at the cost of higher approximation ranks,
- inadequate partitioning strategies, low control parameters ω, and overly relaxed tolerances ϵ can lead to singular value inflation,
- and further theoretical work, particularly on singular value perturbation bounds, may be needed to rigorously explain and control these effects.

5.2.3 Rank truncation of HASVD for Hankel matrices

In this subsection, we examine the rank truncation behavior across different tree structures. This analysis is essential for HASVD, as early truncation helps reduce intermediate problem sizes when computing low-rank approximations.

Three main truncation patterns are investigated:

- the impact of different singular value decay profiles on truncation behavior,
- the effect of partition size on rank truncation at the leaf level,

- and the differences in truncation patterns across tree structures.

To this end, Hankel matrices of size 2400×2400 are generated using both FID and SSA-LRF sequences. The FID sequence produces a severely rank-deficient matrix with $r \approx 20$, while the SSA-LRF sequences yield matrices with controlled ranks $r = 600$ and $r = 1600$.

All common trees listed in Subsection 5.1.2 are investigated. The same partitioning logic from Subsection 5.2.2 is followed here, but with different values of partioning N. The distributed trees and incremental trees divide the matrix into N row blocks respectively. The TLBI and TLBD trees use $N \times N$ block grids, while the TLAI tree forms N diagonal square blocks along with $(N-1)$ standing and sitting rectangular blocks (see Figure 4.7). The partitioning numbers are investigated for $N = 10, 3$ to study truncation behavior when the block size exceeds the matrix rank, i.e., when $600 = r < 800 = \frac{2400}{N}$.

HASVD is applied to the generated Hankel matrices for control parameter values $\omega = 0.01, 0.2, 0.4, 0.6, 0.8, 0.99$ and relative prescribed errors $\epsilon^* = 1, 10^{-1}, 10^{-2}, 10^{-3}, \ldots, 10^{-10}$. Afterwards, the approximation ratio $\frac{\tilde{r}}{r}$ and the average truncation ratio of its leaf nodes $\left\langle \frac{r_{\alpha \in \mathcal{L}_T}}{n_{\text{block}}} \right\rangle$, given by

$$\left\langle \frac{r_{\alpha \in \mathcal{L}_T}}{n_{\text{block}}} \right\rangle := \sum_{\alpha \in \mathcal{L}_T} \frac{1}{|\mathcal{L}_T|} \frac{r_\alpha}{n_{\text{block}}}, \tag{5.9}$$

are measured and computed.

Ratio profiles, as illustrated in Figure 5.6, are constructed using the approximation and truncation ratios. These profiles reveal the performance of each tree structure under varying relative prescribed errors ϵ and control parameter values ω. Ideally, one aims to choose parameters such that the approximation ratio $\frac{\tilde{r}}{r}$ remains close to 1, indicating high accuracy, while the average truncation ratio $\left\langle \frac{r_{\alpha \in \mathcal{L}T}}{n\text{block}} \right\rangle$ is kept near 0, minimizing problem size in HASVD.

To see ratio profiles of other combinations of matrices and trees, the reader is referred to Appendix E.

Figure 5.6 shows results for a Hankel matrix generated using an SSA-LRF sequence with a prescribed rank of $r = 1600$. Note that this kind of Hankel matrix exhibits a sharp singular value decay as demonstrated in Figure 5.1. Due to this property, truncation at the leaf level occurs significantly less often, and most partitioned blocks are likely to be full rank. Consequently, the potential gain from HASVD through early truncation is limited. Meaningful reductions in rank are observed only for relatively loose error tolerances, i.e., $\epsilon > 10^{-2}$. Even then, low approximation ratios are mostly seen when the control parameter ω is large.

In contrast, Figure 5.7 shows results for a Hankel matrix generated using an FID sequence. This matrix is characterized by low rank and a gradual decay of singular values (see Figure 5.1). The results indicate that relatively high approximation ratios can still be achieved, even when the truncation ratios remain low. This suggests that such a setup is particularly well-suited for HASVD.

Nevertheless, the structural differences between tree types have a significant impact

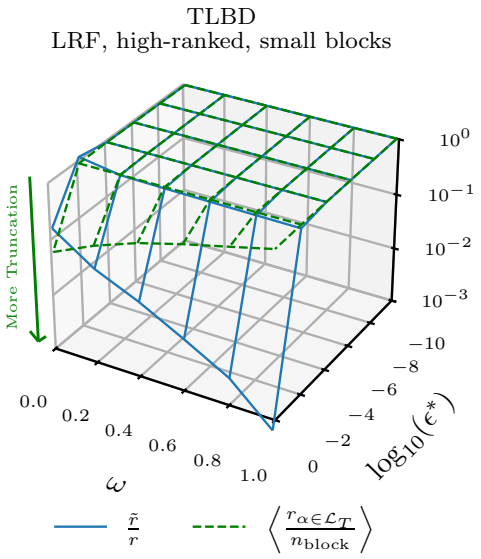


Figure 5.6: Ratio profile for an SSA-LRF-generated Hankel matrix of rank $r = 1600$, approximated using HASVD with a TLBD tree structure and partition block size 240×240, showing the effect of varying relative prescribed error ϵ and control parameter ω on approximation and truncation ratios.

on the resulting approximation rank. In this case, the TLBD tree shows more aggressive truncation at the leaf level compared to the TLBI tree. This behavior stems from the branching node term $\frac{1}{|\mathcal{B}_T|+1}$ in the flat error prescription described in Remark 3.3.2. Since the incremental tree contains more branching nodes, the tolerated nodal error at the leaves is smaller. While this reduces truncation, it also leads to different propagation of approximation errors.

As a result, rank inflation patterns similar to those in Figure 5.4 reappear for TLBD, particularly when the control parameter is small, i.e., $\omega < 0.2$.

The size of the partitioning blocks plays a significant role in the truncation of block approximations after an SVD computation in HASVD. This is demonstrated in Figure 5.8, which compares two TLAI partitionings. One uses blocks of size 800×800, where the block dimension exceeds the target rank $r = 600$, and the other uses smaller blocks of size 240×240.

When the block size is larger than the rank, the leaf nodes can capture more components of the matrix, enabling consistent truncation of block approximations across all relative prescribed errors ϵ and control parameters ω.

In contrast, smaller blocks are less likely to capture lower singular values. As a result, no truncation is observed for relative prescribed errors $\epsilon < 10^{-6}$. Although both partitioning strategies ultimately produce approximations with similar overall ranks, the use of larger blocks yields lower truncation ratios. This is preferable, as it helps reduce the computational cost of performing HASVD.

This pattern is further demonstrated when analysis are performed on the truncation ratios of each individual nodes in an HASVD run for $\omega = 0.2$ and relative prescribed error $\epsilon = 10^{-4}$ for the FID-generated Hankel matrices, as shown in Figure 5.9. As

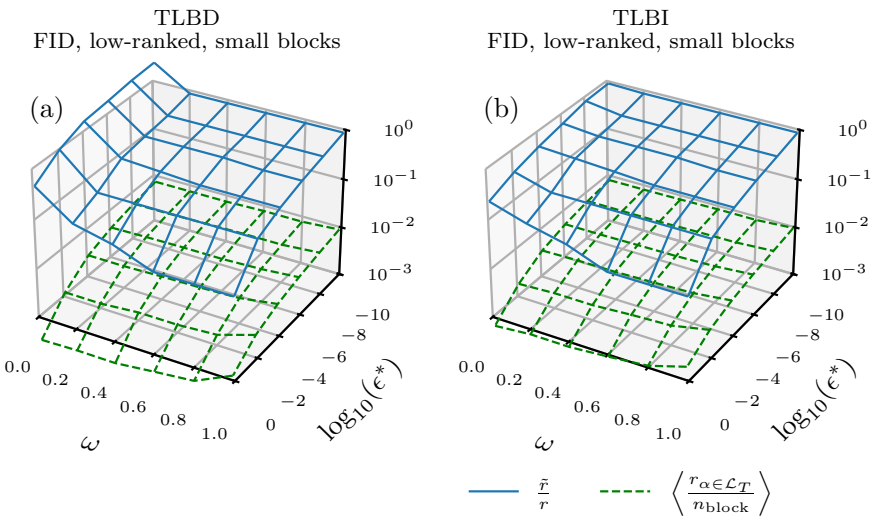


Figure 5.7: Ratio profile for an FID-generated Hankel matrix of rank $r = 21$, approximated using HASVD with (a)TLBD and (b)TLBI tree structures and partition block size 240×240.

one can see, for partitions with small blocks, there are nodes with high truncation ratios throughout the tree, which indicate that the size of the problem computed in each node are barely reduced. Whereas partitions with big blocks generally have small truncation ratio, leading to a reduced problem size withing each node.

As a general guideline, block sizes of partitions should be chosen to exceed the estimated rank of the matrix to enable effective truncation during approximation.

To conclude, the truncation behavior in HASVD is influence by the singular value decay of the matrix, the partitioning block size, and the tree structure used for aggregation. The results from the truncation and approximation ratios shows that:

- matrices with low rank and gradual singular value decay (i.e., from FID) allow for early and consistent truncation,
- matrices with steep singular value decay (i.e., from SSA-LRF) limit truncation unless the prescribed tolerance is loose,
- partitioning blocks to sizes larger than the matrix rank enable leaf nodes to better capture and truncate small singular values, reducing computational cost,
- the tree structure also plays a key role, as deeper trees like TLBI can offer better control over how truncation and approximation errors are distributed throughout the computation.

5.2.4 HASVD runtime for Hankel matrices

This subsection studies the runtime of the HASVD algorithm for Hankel matrices. Since general matrices exhibit similar behavior, the reader is referred to Appendix D for results of HASVD runtime for general matrices.

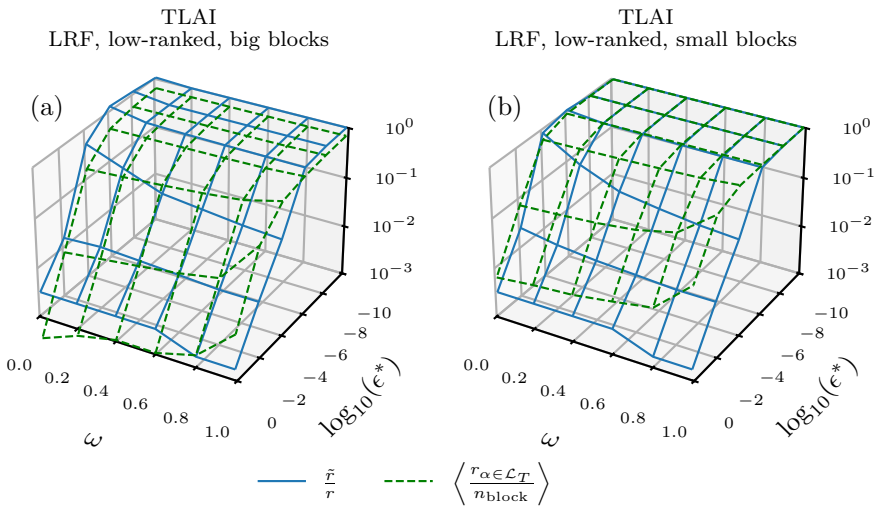


Figure 5.8: Ratio profile for an SSA-LRF-generated Hankel matrix of rank $r = 600$, approximated using HASVD with TLAI tree structure and partition block sizes of (a) 800×800 and (a) 240×240.

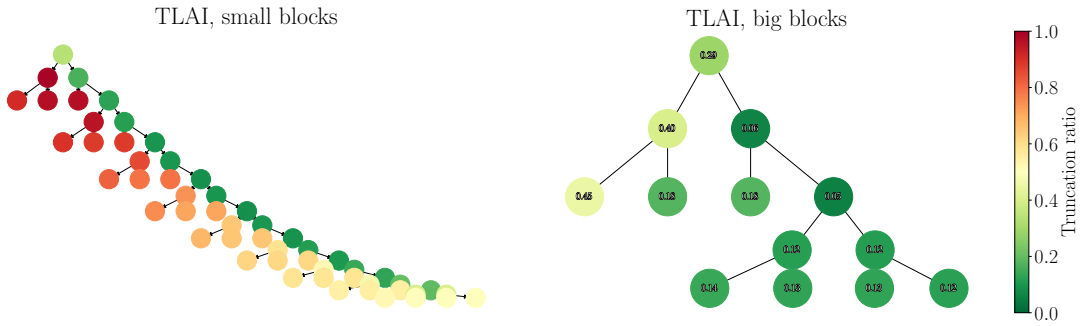


Figure 5.9: Truncation ratios of each node in a HASVD TLAI tree for an FID-generated Hankel matrix of rank $r = 21$, using partition block sizes of 800×800 and 240×240.

The focus of this analysis is on:

- runtime scaling with respect to matrix size,
- runtime scaling with respect to the partitioning number,
- and the speedup achieved on two-level matrices when using computational pruning, relative to matrix size.

To study runtime scaling with matrix size, random Hankel matrices of size $n \times n$ are generated using the FID sequence for $n = 100, 200, 500, 1000, 2000, 5000, 10000$, with the partitioning number fixed at $N = 10$.

For analyzing runtime scaling with respect to partitioning, a random Hankel matrix of size 5000×5000 is generated using the FID sequence, and partitioning numbers $N = 10, 20, 50, 100, 200, 500$ are tested.

We measure the runtime of each HASVD run using a relative prescribed error of $\epsilon = 10^{-5}$ and a control parameter value of $\omega = 0.1$. Furthermore, the runtime of GESDD is also measured as reference.

All trees described in Subsection 5.1.2 are included in the runtime scaling analysis, with row aggregation applied at the root node. For the pruning speedup investigation, a recursion key is used with the TLBI, TLBD, and TLAI trees. These trees are chosen because their partitioning structures possess symmetry, as discussed in Section 4.2.

As in Subsections 5.2.2 and 5.2.3, we follow the same partitioning logic with respect to the partitioning number N. The distributed trees and incremental trees divide the matrix into N column blocks, respectively. The TLBI and TLBD trees use $N \times N$ block grids, while the TLAI tree forms N diagonal square blocks, along with $(N-1)$ standing and sitting rectangular blocks (see Figure 4.7).

Figure 5.10(a) shows the mean runtime of GESDD and HASVD across different matrix sizes. It is clear that for small matrices, the HASVD algorithm is generally slower than GESDD, but it starts to outperform GESDD for larger sizes, specifically when $n > 500$. This trend reflects the limited truncation gains observed in Figure 5.9. With a fixed partitioning number, smaller matrices result in smaller block sizes, which can reduce the effectiveness of truncation in HASVD. As a result, the computational cost of HASVD remains high relative to GESDD in this regime.

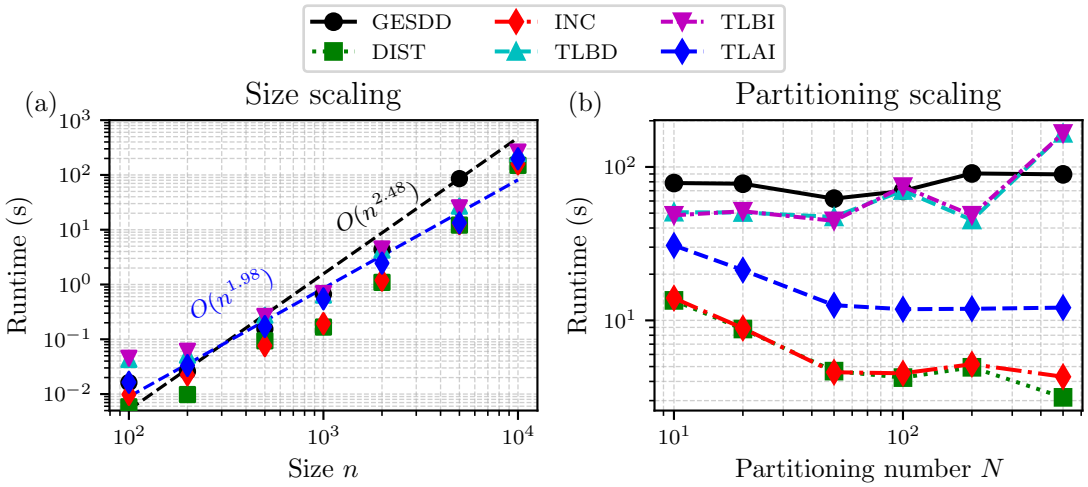


Figure 5.10: Runtime of GESDD as compared to HASVD with different trees. (a) Runtime scaling relative to size. Partitioning number is set to $N = 10$. (b) Runtime scaling relative to partitioning number. Size is set to $n = 5000$.

In the runtime scaling with respect to partitioning number shown in Figure 5.10(b), the runtime generally remains stable until very high partitioning numbers are reached. This is expected, as increasing the partitioning number leads to smaller block sizes, which as seen in Figures 5.6 and 5.9, reduces truncation ratios. Similar behavior has also been observed in runtime analyses of HAPOD, where higher partitioning numbers can lead to slower performance [15].

There is also a clear advantage of using linear partitioning as opposed to small block partitioning as shown in Figure 5.10(b). This seems to also reflect the structure of

Hankel matrices, in which FID signal sequences of sums of exponential decays are ordered along the anti-diagonals. Since the number of exponential components in FID signals reflect the rank of a Hankel matrix [38], a linear partitioning will capture more samples of the sequence and is able to identify more exponential components. This in turn allow HASVD using distributed trees (DISTs) and incremental trees (INCs) to truncate more agressively. On the other hand, small block partitioning from TLBD and TLBI trees are only able to detect a significantly smaller sample and therefore, may be required to do more, and possibly redundant SVD computations subtasks in HASVD.

The structure of two-level alternating incremental tree (TLAI) tree that combines square and rectangular block partitioning scheme seems to be reflected in its runtime, as it consistently lies between the the computational runtime of TLB trees and linear trees. This provides a promising compromise of using symmetry within the structure of Hankel matrices while being able to capture a wider range of sample in the generating sequence of the Hankel matrix.

The introduction of pruning reveals a markedly different runtime profile. Figure 5.11 illustrates that TLBD, TLBI, and TLAI trees exhibit superior performance when combined with pruning techniques in HASVD applications for Hankel matrices. As a sanity check, the error computation records show a consistent result for bounding properties, with a relative Frobenius error of around $\approx 10^{-7}$.

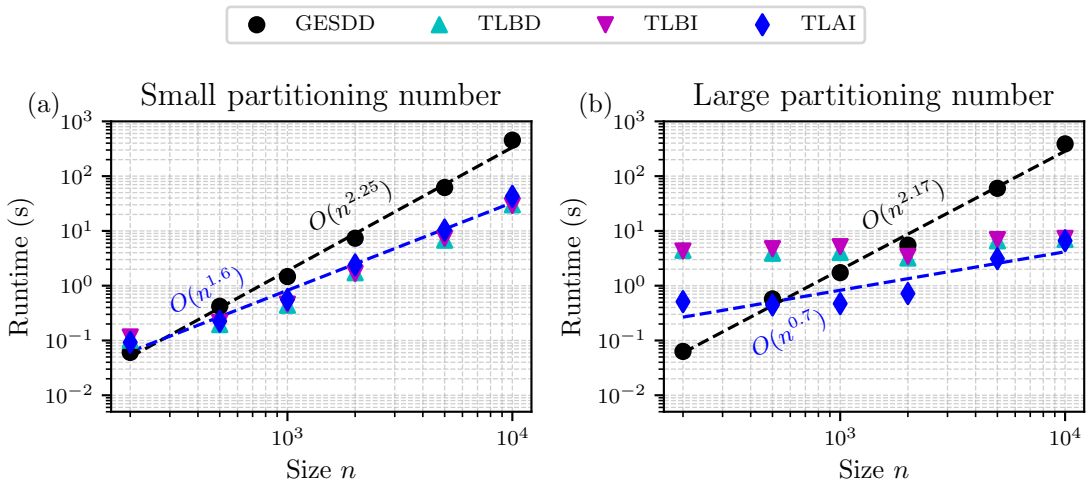


Figure 5.11: Runtime scaling relative to size of GESDD as compared to HASVD with different trees under pruning. (a) Partitioning number is set to $N = 10$. (b) Partitioning number is set to $N = 100$

As shown in Figure 5.11(a), pruning in HASVD can already improve the scaling of runtime for TLB trees, while runtime characteristic of TLAI trees remains the same for small partitionings. This is caused by the relatively larger size of blocks that has to be decomposed by HASVD performed using TLAI trees, while TLB trees can already gain an advantage by the reduction in the amount computations needed for individual blocks associated to the leaf as discussed in Subsection 4.2.1.

An increase in the partitioning number, as shown in Figure 5.11(b), results in a significant improvement in runtime. For the hardware architecture used in this

experiment, the runtime scales sublinearly for HASVD with both TLB and TLAI trees, following a trend of $\mathcal{O}(n^x)$ with $x < 1$. In particular, a speedup of up to a factor of approximately 10^2 is observed when using the TLAI tree for a matrix of size $n = 10000$.

It is worth noting that the slower runtimes for smaller matrices are due to the limited amount of truncation in small blocks, as also seen in Figure 5.10(b). Nonetheless, these results demonstrate that applying a sufficiently large partitioning number to an appropriately sized matrix can significantly enhance the effectiveness of pruning, leading to substantial reductions in computational runtime.

After this runtime experiment, the results demonstrate that thoughtful selection of tree structures, partitioning strategies, and pruning can greatly reduce the computational cost of HASVD. Key takeaways include:

- HASVD can outperform GESDD when used with appropriately sized partitions due to increased potential for rank truncation,
- linear partitioning strategies (DIST/INC) may be better suited to signal-generated Hankel matrices by enabling more aggressive truncation,
- TLAI tree provides a balanced approach, combing structural symmetry with broader block coverage,
- and computational pruning offers significant runtime improvements—especially for large matrices and high partitioning numbers—potentially achieving sub-linear scaling.

5.3 Application of HASVD to block-Hankel Matrices from HRTF Data

This section evaluates the performance of HASVD when applied to block-Hankel matrices constructed from real-world head-related transfer function (HRTF) impulse response measurements. Utilizing a comprehensive KEMAR dataset, which provides a variety of singular value decay behaviors, we investigate how effectively HASVD produces low-rank approximations in practical scenarios.

By benchmarking against established methods such as randomized SVD, we aim to demonstrate the potential of HASVD for handling large-scale, structured matrices arising in real-life signal processing applications, with particular attention to approximation accuracy and computational efficiency.

5.3.1 Hankel matrix construction using HRTF measurements

The performance of computing low-rank approximations using HASVD for block-Hankel matrices in Definition 2.1.3 is studied. Head-related transfer function (HRTF) measurements from KEMAR are employed for this investigation [6]. These datasets consist of real impulse responses of Realistic Optimus Pro 7 loudspeakers positioned

1.4 meters from a Knowles Electronics model DB-4004 output, designated as KEMAR.

The dataset contains $m = 710$ different input (source) positions, $p = 2$ output (target) positions, and $s = 256$ time samples. Each input sampling is positioned around all azimuthal angles and at elevation angles between $-\frac{2\pi}{9}$ to $\frac{\pi}{2}$ relative to the output as illustrated in Figure 5.12.

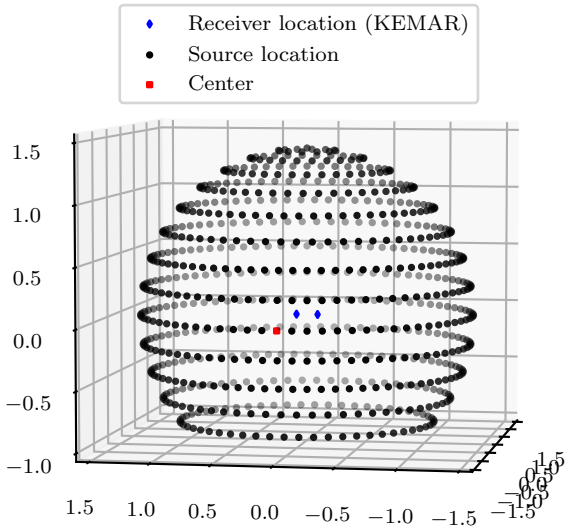


Figure 5.12: Positions of inputs and outputs in the HRTF KEMAR measurements [33].

Such a dataset of impulse responses can fully determine any causal discrete-time linear time-invariant (LTI) system with m inputs and p outputs by its impulse response matrix $h_i \in \mathbb{R}^{p\times m}$ for $i \in \mathbb{N}$ [34]. This is accomplished by computing the SVD of the block-Hankel matrix $H \in \mathbb{R}^{ps\times ms}$, given by

$$H = \begin{pmatrix} h_1 & h_2 & \cdots & h_s \\ h_2 & h_3 & \cdots & h_{s+1} \\ \vdots & \vdots & \ddots & \vdots \\ h_s & h_{s+1} & \cdots & h_{2s-1} \end{pmatrix}, \tag{5.10}$$

where $h_1, \ldots, h_s \in \mathbb{R}^{p\times m}$ are the impulse response of the samples and $h_{s+1}, \ldots, h_{2s-1} \in \mathbb{R}^{p\times m}$ are zero matrices padded to the impulse response [20]. With the KEMAR datasets, this means that the approximate SVD of a block-Hankel matrix $H \in \mathbb{R}^{512\times 181760}$ has to be computed.

Extending the investigation beyond single sequences, the rank of H corresponds in this case to the aggregate intensity peaks of the entire impulse response under Fourier transform [10]. Furthermore, computing the SVD of a block-Hankel matrix from a subset of $m' < m$ input samples yields different responses and singular value decay, as demonstrated in Figure 5.13.

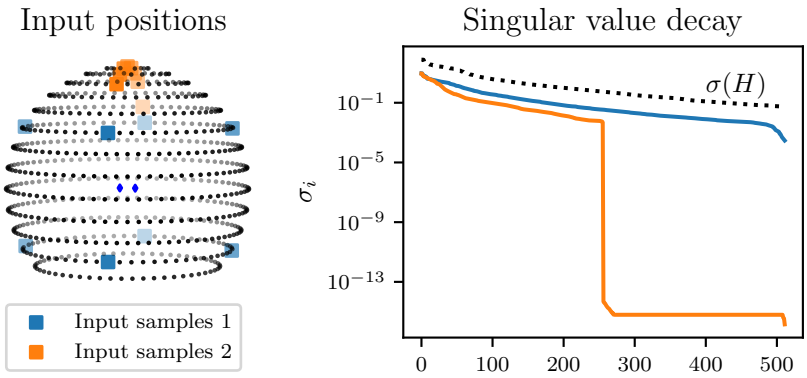


Figure 5.13: Singular value decay of H formed from entire HRTF KEMAR impulse response compared to decay formed from subsets of inputs.

It can be observed that selecting input subsets where points are concentrated along the center of the two target source generates a singular value decay that is sharper. This occurs because impulse response characteristics become more similar, resulting to similar positions of intesity peaks. On the other hand, an evenly distributed selection of subset inputs produces a more spread out decay.

Using this observation, a benchmark using HASVD is conducted on these subset samples, in addition to the entire HRTF impulse response measurement.

randomized SVD (Rand-SVD) is employed as a reference benchmark. Given a matrix A, this SVD method uses randomized sampling to select an approximate subspace from the range of matrix A and constructs a low-rank approximation of matrix A using this approximate subspace [39].

Applications of Rand-SVD includes its use for computationally efficient SVD in the ERA used to determine systems such as those described above [30]. A pyMOR implementation of Rand-SVD is utilized, which can be found in `pymor.algorithms.rand_la.randomized` Since this method approximates matrix up to a rank k, comparison is performed by first running HASVD with a relative prescribed error ϵ^*, obtaining the approximation rank, and using this rank as a target rank for Rand-SVD.

5.3.2 Application of HASVD To HRTF measurements

When dealing with block-Hankel matrices, it is generally desirable to follow the structure of the block-Hankel matrix as the partitioning scheme for HASVD, as mentioned in Subsection 4.2.1 to employ pruning technique at the block level.

However, it should be noted that a single block $h_i \in \mathbb{R}^{p\times m}$ for $i = 1, \ldots, 511$ of the KEMAR HRTF impulse response has dimensions of $p = 2$ and $m = 710$. This represents an extremely short and fat matrix. As observed in Figure 5.8, having small partitions may be detrimental for HASVD performance, as it reduces the chances of truncation.

Two options exist to address this issue. The first naive option involves expanding the partitioning scheme arbitrarily to include more blocks h_i such that each block associated to the leaf node is not $p \times m$, but rather $kp \times lm$ for $k \in \mathbb{N}$ and $l \in \mathbb{N}$.

This may permit more truncation for HASVD, but also breaks the skew-diagonal symmetry in (4.3) needed for pruning.

The second option involves expanding the block while maintaining the ratio of the partitioning dimension, i.e., $kp \times km$ for $k \in \mathbb{N}$. This approach preserves the skew-diagonal symmetry of the partitioning blocks and obtains a sufficiently large block size for truncation.

The effect of these two choices is investigated using the two subset input selection of $s = 8$ shown in Figure 5.13, which results in a block-Hankel matrix of size 512×2048. An HASVD using an incremental TLB tree is constructed with two partitioning schemes, resulting in partitions of square blocks of size 64×64 and rectangular blocks of size 16×64. This means that the square blocks does not take the skew-diagonal symmetry of the blocks in block-Hankel matrix, while the rectangular block partitioning still maintain such symmetry.

Furthermore, the HASVD runs are performed for a row aggregation in the root node, given a relative error prescription of $\epsilon^* = 10^{-5}$ and the control parameter $\omega = 0.1$.

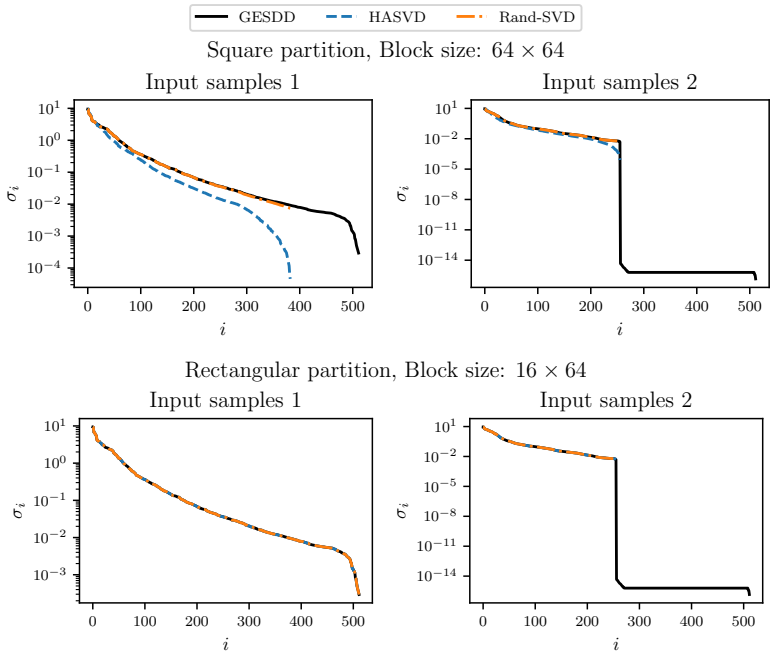


Figure 5.14: Singular value estimates of different SVD algorithms under different partitioning shape.

As shown in Figure 5.14, a significant difference in approximation quality is observed between partitioning into square blocks and partitioning into rectangular blocks. However, the stark difference in accuracy may also be amplified by the amount of blocks needed to be aggregated in the row aggregation of the square partition, similar to the numerical artifacts found from Figure 5.5.

This indicates that the square partitioning may not only remove the computational costs advantages of the skew-diagonal symmetry in pruning, but also reduce the accuracy of the computed low-rank approximation.

Finally, initial benchmark are also performed for the entire HRTF measurements

with $m = 710$. In this case, an HASVD run using distributed TLB tree is compared to Rand-SVD. A rectangular partitioning of block size 32×11360 is used, so that skew-diagonal symmetry is maintained between the blocks. With this configuration, block-recurrent pruning is used to accelerate HASVD computation.

Each run measures the mean runtime, approximation rank, and Frobenize error for relative prescribed errors of $\epsilon^* = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ and control parameter values $\omega = 0.1, 0.5$.

The results for $\omega = 0.5$ are presented in Table 5.1. It is evident that for a certain setups, HASVD shows significant speedup relative to Rand-SVD. Even in terms of relative error, low-rank approximations from HASVD performs worse only when the limit of the algorithm's precision is reached, particularly around nearly full rank conditions. It is also important to note that Rand-SVD's use of random sampling only provides the method with probabilistic error bounds [12], whereas HASVD may provide a more consistent bounding characteristic.

ϵ^*	Rank	Runtime (s)		Relative Frobenius Error	
		HASVD	Rand-SVD	HASVD	Rand-SVD
10^{-1}	139	**3.88**	55.80	$\mathbf{5.20 \cdot 10^{-2}}$	$9.03 \cdot 10^{-2}$
10^{-2}	358	**6.68**	265.81	$\mathbf{5.18 \cdot 10^{-3}}$	$1.18 \cdot 10^{-2}$
10^{-3}	507	**8.44**	501.12	$4.75 \cdot 10^{-4}$	$\mathbf{4.73 \cdot 10^{-4}}$
10^{-4}	512	**8.73**	524.06	$6.16 \cdot 10^{-15}$	$\mathbf{5.90 \cdot 10^{-15}}$
10^{-5}	512	**8.67**	533.44	$6.16 \cdot 10^{-15}$	$\mathbf{5.91 \cdot 10^{-15}}$

Table 5.1: Mean runtime and relative Frobenius errors of SVD algorithm on a 512×181760 block-Hankel based on KEMAR HRTF measurement.

Although, this brief exploration of real applications involves only a relatively small block-Hankel matrix, it already provides a glimpse of the potential of HASVD for future applications.

Chapter 6

Conclusion and Outlook

This chapter serves as the conclusion to this thesis.

Section 6.1 summarizes the theoretical and experimental work conducted on the HASVD framework and presents important takeaways for practitioners using HASVD to form low-rank approximations.

Section 6.2 outlines potential future research directions that could emerge from this study, particularly in the study of rectangular Hankel matrices, subspace perturbation, and more complex tree structures.

6.1 Summary

Throughout the course of this thesis, we aim to establish a hierarchical framework for singular value decomposition as an extension of hierarchical approximate proper orthogonal decomposition (HAPOD) [15]. At the core of this framework is a tree-based structure used to organize and aggregate blocks of SVD computations. Building on the works in [45], a systematic procedure to aggregate singular vectors using rows and column aggregations of blocks within a matrix was introduced in Section 3.2. These developments culminate in the formulation of the hierarchical approximate singular value decomposition (HASVD), where approximate SVD techniques are combined with hierarchical aggregation to facilitate progressive reduction of problem size in the HASVD procedure.

Using the property of the additivity of squared Frobenius norm and unitary invariance, we establish two error bounds relative to the nodal error of each node in an aggregation tree. These bounds serve as the foundation for both flat and hierarchical error prescriptions, enabling a priori control over the approximation errors produced by the HASVD algorithm.

Multiple tree structures have been identified. These ranges from simpler configurations like the distributed tree (DIST) and incremental tree (INC) trees to more advanced structures such as distributed two-level bidirectional tree (TLBD), incremental two-level bidirectional tree (TLBI), and two-level alternating incremental tree (TLAI). Based on these, we develop pruning techniques that exploit the skew-diagonal and transpose symmetries inherent in Hankel matrices.

The HASVD algorithm was implemented in Python by extending an existing HAPOD implementation in the pyMOR Library [13]. The challenges associated with generating random Hankel matrices were addressed, and signal models from FID and SSA-LRF sequences were employed as test cases to capture diverse singular value behaviors of Hankel matrices [37, 9].

Accuracy tests was conducted on a small 2000×2000 Hankel matrix in Subsection 5.2.2. While results shows satisfactory approximation accuracy, results also reveal the susceptibility of HASVD particularly when using TLBD in low singular values. While further investigation is needed, preliminary results indicate that an increase in the control parameter ω of the error prescription in Remark 3.3.2 and the limiting of partition may reduce these effects. Moreover, comparisons between trees also shows that the choice of error prescription, whether flat or hierarchical, is less significant compared to the choice of tree structure.

We also introduce ratio profiles to analyze the rank truncation behavior and size reduction effectiveness of HASVD. Experiments show that highly rank-deficient Hankel matrices are particularly well-suited for HASVD, and that strategic partitioning significantly enhances truncation performance.

Runtime benchmark on matrices from sizes of 100×100 to 10000×10000 was conducted on Hankel matrices. Results show that increasing the number of partitions can lead to a slowdown, confirming the importance of balanced partitioning. Pruning technique on HASVD particularly when using TLAI tree, shows a significant speedup in runtime, reaching sublinear runtime scaling.

The application of HASVD to real-world data is demonstrated through analysis of KEMAR HRTF measurements [6]. For the large 512×181760 matrix, combining pruning with strategic partitioning led to superior runtime performance, and in some cases, better approximation accuracy compared to established algorithms such as randomized SVD.

Overall, the findings demonstrate that effective application of the HASVD framework to Hankel matrices requires careful consideration of partition size and tree structure. As a general rule, partition sizes should be set slightly larger than the estimated rank for HASVD to be effective. Although the experiments are limited to moderate-sized matrices, the observed scaling properties of pruning and hierarchical aggregation suggest strong potential for parallelization and large-scale problems.

6.2 Future Research

Several key directions for future research emerge from the conclusions of this work. These are not limited to questions left unanswered due to the scope constraints of this thesis, but also include new and open questions that arise directly from the findings presented.

Firstly, while initial experiments were conducted on matrices of relatively moderate sizes, further studies involving large-scale matrices are warranted. This includes the development and evaluation of algorithms for generating large block-Hankel matrices. In this context, work on constructing block-Hankel structures

from two-dimensional FID signal sequences in nuclear magnetic resonance (NMR) spectroscopy provides a useful reference point [10].

In terms of the HASVD framework, we have established a robust error bound and prescription in Section 3.3. However, it remains important to derive tighter theoretical worst-case bounds in terms of the spectral norm, which could provide a more precise evaluation of approximation quality.

Additionally, with the foundational framework in place, several natural extensions of the HASVD method may be explored. For instance, it has been demonstrated that the matricization of tensors into block matrices preserves errors in the Frobenius norm [21], suggesting the potential for HASVD to support higher-order approximations. The method may also prove suitable for evaluating local SVDs of blocks, which is highly relevant in the context of hierarchical matrices. In such matrices, large blocks are approximated using low-rank representations, often reducing storage complexity to $\mathcal{O}(nr \log n)$ for certain partitioning strategies, where n is the matrix size and r is the rank [11].

Furthermore, in light of the singular value inflation observed in Subsection 5.2.2, a dedicated study of the low-end singular value behavior in Hankel matrices is warranted. Initial developments in solving Hankel systems via Vandermonde-Cauchy product formulations offer a promising direction [5].

Finally, the analysis in Subsection 5.2.3 indicates that partition size significantly affects rank truncation performance. This insight motivates the exploration of dynamic tree structures, where nodes and blocks may be adaptively merged during the HASVD computation. Such adaptability could improve computational efficiency. Consequently, tree-based pruning techniques also warrant further investigation.

Bibliography

[1] *MPI: A Message-Passing Interface Standard.* https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[2] G. Berkooz, P. Holmes, and J. L. Lumley, *The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows*, Annual Review of Fluid Mechanics, 25 (1993), pp. 539–575, https://doi.org/10.1146/annurev.fl.25.010193.002543.

[3] C. Boutsikas, P. Drineas, and I. C. F. Ipsen, *Small Singular Values Can Increase in Lower Precision*, SIAM Journal on Matrix Analysis and Applications, 45 (2024), pp. 1518–1540, https://doi.org/10.1137/23m1557209.

[4] A. Chatterjee, *An introduction to the proper orthogonal decomposition*, Current Science, 78 (2000).

[5] Z. Drmač, *SVD of Hankel matrices in Vandermonde-Cauchy product form*, 44 (2015), pp. 593–623.

[6] B. Gardner and K. Martin, *HRTF Measurements of a KEMAR Dummy-Head Microphone*, May 1994.

[7] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins Studies in the Mathematical Sciences, The Johns Hopkins University Press, Baltimore, fourth edition ed., 2013.

[8] N. Golyandina and A. Zhigljavsky, *Singular Spectrum Analysis for Time Series*, SpringerBriefs in Statistics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, https://doi.org/10.1007/978-3-642-34913-3.

[9] N. E. Golyandina, V. V. Nekrutkin, and A. Zhigljavsky, *Analysis of Time Series Structure: SSA and Related Techniques*, no. 90 in Monographs on Statistics and Applied Probability, Chapman & Hall / CRC, Boca Raton, Fla., 2001.

[10] D. Guo, H. Lu, and X. Qu, *A Fast Low Rank Hankel Matrix Factorization Reconstruction Method for Non-Uniformly Sampled Magnetic Resonance Spectroscopy*, IEEE Access, 5 (2017), pp. 16033–16039, https://doi.org/10.1109/access.2017.2731860.

[11] W. Hackbusch, *Hierarchical Matrices: Algorithms and Analysis*, Springer Series in Computational Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, https://doi.org/10.1007/978-3-662-47324-5.

[12] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Review, 53 (2011), pp. 217–288, https://doi.org/10.1137/090771806.

[13] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, et al., *Array programming with NumPy*, Nature, 585 (2020), pp. 357–362, https://doi.org/10.1038/s41586-020-2649-2.

[14] G. Heinig and K. Rost, *Algebraic Methods for Toeplitz-like Matrices and Operators*, vol. 13 of Operator Theory: Advances and Applications, Birkhäuser Basel, Basel, 1984, https://doi.org/10.1007/978-3-0348-6241-7.

[15] C. Himpe, T. Leibner, and S. Rave, *Hierarchical Approximate Proper Orthogonal Decomposition*, SIAM Journal on Scientific Computing, 40 (2018), pp. A3267–A3292, https://doi.org/10.1137/16M1085413.

[16] P. Holmes, J. L. Lumley, and G. Berkooz, *Proper Orthogonal Decomposition*, in Turbulence, Coherent Structures, Dynamical Systems and Symmetry, vol. 1.3, Cambridge University Press, 1 ed., Oct. 1996, pp. 86–128, https://doi.org/10.1017/CBO9780511622700.

[17] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, NY, second edition, corrected reprint ed., 2017.

[18] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, et al., *LoRA: Low-Rank Adaptation of Large Language Models*, Oct. 2021, https://doi.org/10.48550/arXiv.2106.09685, https://arxiv.org/abs/2106.09685.

[19] I. S. Iohvidov, I. Gohberg, and G. P. A. Thijsse, *Hankel and Toeplitz Matrices and Forms: Algebraic Theory*, Birkhäuser, Boston, 1982.

[20] J.-N. Juang and R. S. Pappa, *An eigensystem realization algorithm for modal parameter identification and model reduction*, Journal of Guidance, Control, and Dynamics, 8 (1985), pp. 620–627, https://doi.org/10.2514/3.20031.

[21] M. E. Kilmer and A. K. Saibaba, *Structured Matrix Approximations via Tensor Decompositions*, SIAM Journal on Matrix Analysis and Applications, 43 (2022), pp. 1599–1626, https://doi.org/10.1137/21M1418290.

[22] N. Kishore Kumar and J. Schneider, *Literature survey on low rank approximation of matrices*, Linear and Multilinear Algebra, 65 (2017), pp. 2212–2244, https://doi.org/10.1080/03081087.2016.1267104.

[23] B. Kramer and S. Gugercin, *Tangential interpolation-based eigensystem realization algorithm for MIMO systems*, Mathematical and Computer Modelling of Dynamical Systems, 22 (2016), pp. 282–306, https://doi.org/10.1080/13873954.2016.1198389.

[24] G.-Y. Lee, K.-J. Park, D.-G. Lim, and Y.-H. Park, *Model order reduction based on low-rank approximation for parameterized eigenvalue problems*

in structural dynamics, Journal of Sound and Vibration, 582 (2024), p. 118413, https://doi.org/10.1016/j.jsv.2024.118413.

[25] J. Liesen and V. Mehrmann, *Linear Algebra*, Springer Undergraduate Mathematics Series, Springer International Publishing, Cham, 2015, https://doi.org/10.1007/978-3-319-24346-7.

[26] R. K. Lim and M. Q. Phan, *State-Space System Identification with Identified Hankel Matrix*, Technical Report 3045, Princeton University, Princeton, NJ, Sept. 1998.

[27] M. Matsumoto and T. Nishimura, *Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulation, 8 (1998), pp. 3–30, https://doi.org/10.1145/272991.272995.

[28] M. Matsumoto, M. Saito, and H. Haramoto, *Pseudorandom Number Generation: Impossibility and Compromise*.

[29] R. Milk, S. Rave, and F. Schindler, *pyMOR – Generic Algorithms and Interfaces for Model Order Reduction*, SIAM Journal on Scientific Computing, 38 (2016), pp. S194–S216, https://doi.org/10.1137/15M1026614.

[30] R. Minster, A. K. Saibaba, J. Kar, and A. Chakrabortty, *Efficient Algorithms for Eigensystem Realization Using Randomized SVD*, SIAM Journal on Matrix Analysis and Applications, 42 (2021), pp. 1045–1072, https://doi.org/10.1137/20M1327616.

[31] Y. Nakatsukasa, O. Sète, and L. N. Trefethen, *The AAA Algorithm for Rational Approximation*, SIAM Journal on Scientific Computing, 40 (2018), pp. A1494–A1522, https://doi.org/10.1137/16M1106122.

[32] V. Y. Pan, *Structured Matrices and Polynomials*, Birkhäuser Boston, Boston, MA, 2001, https://doi.org/10.1007/978-1-4612-0129-8.

[33] A. J. R. Pelling, *Head-related Transfer Function Modelling from Data*. pyMOR, Aug. 2024.

[34] A. J. R. Pelling and E. Sarradj, *Efficient Forced Response Computations of Acoustical Systems with a State-Space Approach*, Acoustics, 3 (2021), pp. 581–593, https://doi.org/10.3390/acoustics3030037.

[35] G. E. Plassman, *A Survey of Singular Value Decomposition Methods and Performance Comparison of Some Available Serial Codes*, Technical Report 20050192421, Langley Research Center, Virignia, United States, July 2005.

[36] Python Software Foundation, *Asyncio — Asynchronous I/O*. The Python Software Foundation.

[37] T. Qiu, W. Liao, Y. Huang, J. Wu, et al., *An Automatic Denoising Method for NMR Spectroscopy Based on Low-Rank Hankel Model*, IEEE Transactions on Instrumentation and Measurement, 70 (2021), pp. 1–12, https://doi.org/10.1109/TIM.2021.3109743.

[38] T. Qiu, Z. Wang, H. Liu, D. Guo, and X. Qu, *Review and prospect: NMR spectroscopy denoising and reconstruction with low-rank Hankel matrices and tensors*, Magnetic Resonance in Chemistry, 59 (2021), pp. 324–345, https://doi.org/10.1002/mrc.5082.

[39] A. K. Saibaba, J. Hart, and B. Van Bloemen Waanders, *Randomized algorithms for generalized singular value decomposition with application to sensitivity analysis*, Numerical Linear Algebra with Applications, 28 (2021), p. e2364, https://doi.org/10.1002/nla.2364.

[40] R. S. Sengar, K. Chatterjee, and J. Singh, *System Approximation via Restructured Hankel Matrix*, Circuits, Systems, and Signal Processing, 40 (2021), pp. 6354–6370, https://doi.org/10.1007/s00034-021-01745-2.

[41] L. Sirovich, *Turbulence and the dynamics of coherent structures. I. Coherent structures*, Quarterly of Applied Mathematics, 45 (1987), pp. 561–571, https://doi.org/10.1090/qam/910462.

[42] G. W. Stewart, *Perturbation Theory for the Singular Value Decomposition*, Tech. Report UMIACS-TR-90-124, University of Maryland, Sept. 1990, https://arxiv.org/abs/1903/552.

[43] M. Stewart, *Perturbation of the SVD in the presence of small singular values*, Linear Algebra and its Applications, 419 (2006), pp. 53–77, https://doi.org/10.1016/j.laa.2006.04.013.

[44] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[45] V. Vasudevan and M. Ramakrishna, *A Hierarchical Singular Value Decomposition Algorithm for Low Rank Matrices*, May 2019, https://arxiv.org/abs/1710.02812.

Appendices

A PyHASVD tutorial

In this tutorial, we will walk through the process of the PyHASVD library for HASVD computations with Python. We will cover the following steps:

In the `thesis_code` folder there should be:

- `pyhasvd` - the main package folder containing the HASVD implementation.
- `requirements.txt` - a file containing the required packages for PyHASVD.
- `setup.py` - a script for installing the package.

A.1 Installation

Python 3.10 or above is required. First, create a virtual environment in the `thesis_code` folder and activate it:

```
python -m venv venv
source venv/bin/activate
```

Then, to install required packages for setuptools, you can use pip in the `thesis_code` folder:

```
pip install -r requirements.txt
pip install .
```

A.2 Basic Usage

Here is a simple example of how to use PyHASVD to perform a hierarchical approximate SVD on a Hankel matrix using different tree structures:

We first generate an SSA-LRF sequence for a matrix generation.

```
from pyhasvd.utils.matrix import mersenne_twister,lrf_sequence

m,n = (100,100) # Size of the Hankel matrix
rank = 10  # Desired rank of the Hankel matrix

rng= mersenne_twister(42) # Random number generator
seq = lrf_sequence(rank,m+n-1,rng)
```

Next we generate a incremental two-level bidirectional tree (TLBI) tree structure for the matrix partitioning.

```
from pyhasvd.utils.trees import tlbi_hasvd_tree

root_direction = 0 # 0 for row partitioning, 1 for column partitioning
```

```
M = 10  # Number of row blocks
N = 10  # Number of column blocks

block_m,block_n = (m/M, n/N) # Size of each block in the partitioning

tree = tlbi_hasvd_tree(N,M,root_direction,(m/M,n/N),rank) # Create the tree
```

Now we can set a tolerance and control parameter for a flat error prescription. This will generate a callable function `nodal_error` that computes the nodal error for each node in the tree.

```
from pyhasvd.utils.errors import flat_error
from pyhasvd.utils.trees import branch_node_count

tol = 1e-3  # Tolerance for the error prescription
omega = 0.1  # Control parameter for the error prescription

nodal_error = lambda node: flat_error(node, tol, omega, branch_node_count(tree))
```

Before we can perform the HASVD computation, we need to prepare a leaf-to-block mapping. In PyHASVD, we have a matrix-free approach, so that only the sequence is needed to generate blocks.

```
from pyhasvd.utils.trees import tlb_hankelarray_ltb_map

ltb_map = tlb_hankelarray_ltb_map(seq, M, N, m,n, root_direction)
```

Finally, we can perform the HASVD computation using the `hasvd` function. This function takes the tree, the leaf-to-block mapping, and the nodal error function as arguments.

```
from pyhasvd.utils.svd import hasvd

U,S,Vt=hasvd(tree, ltb_map, nodal_error)
```

Some advanced features of `hasvd()` include:

- `cache_map` argument, which can be used for pruning. The `pyhasvd.utils.svd.transpose_ca` can be inserted as an argument to accelerate TLAI trees in Hankel matrices.
- `svd_method` argument, which can be used to specify the SVD method to use. The default is `np.linalg.svd`, but as long as the method returns singular vectors and values in the form `(U,S,Vt)`, it can be used.
- `track_ranks` argument, returns a dictionary with the ranks of each node in the tree.

Bug: Setting large tolerances will lead to HASVD cutting off the entire matrix, resulting in an empty output. This is a known issue and will be fixed in future releases.

A.3 Benchmark

Benchmark script used can be found in `thesis_code/pyhasvd/scripts/benchmarks`. Data compilation generates `.csv` files through pandas.

B Random Hankel matrix generation

Generating low-rank Hankel matrices with a predefined rank is not as trivial as constructing a series of desired singular values and randomly choosing a singular vector.

This is due to the fact that small perturbations of a matrix may completely change its singular values, i.e. for a matrix

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1+\epsilon \end{pmatrix},$$

where $\epsilon > 0$, we have the singular vectors (both left and right),

$$V = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Given a perturbation of matrix A, we can have

$$\tilde{A} = \begin{pmatrix} 1 & \epsilon \\ \epsilon & 1 \end{pmatrix},$$

where the perturbed singular vectors are

$$\tilde{V} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

which shows that there are matrices with arbitrarily small perturbations ϵ that leads to huge changes in singular vectors [42].

With the example above, one can conclude that given a sequence of singular values, it is numerically difficult to determine the singular vectors that lead to a Hankel matrix. Therefore, it is more often the case that one models the sequence that defines a Hankel matrix, which may lead to a desired property, such as having a low rank.

The first way to achieve this is using a linear recurrent formula used for time series generation in single spectrum analysis (SSA) that has finite ranks[9, 8].

Remark B.1 (single spectrum analysis linear recurrent formula (SSA-LRF)[9])**.** *Let $H = Hankel_{mn}(a_1, a_2, \ldots, a_{n+m-1}) \in \mathbb{R}^{m\times n}$ (see (2.11)) be a Hankel matrix. Then $rank(H) = r$ if and only if the sequence $(a_1, a_2, \ldots, a_{n+m-1}) \in \mathbb{R}^{n+m-1}$ satisfies a linear recurrent relation of order r, given by,*

$$a_j = \sum_{i=1}^{r} c_i a_{j-i}, \quad \forall j \geq r+1, \tag{1}$$

where $c_1, c_2, \ldots, c_r \in \mathbb{R}$.

Such a Hankel matrix is achieved by initially choosing polynomial coefficients $c_1, \ldots, c_r \in \mathbb{R}$ and sequence elements $a_1, \ldots, a_r \in \mathbb{R}$ at random, and using (1) to construct the remaining elements of the sequence.

Although Remark B.1 states that it algebraically possible to construct a Hankel matrix with a prescribed rank, it is numerically difficult to achieve. Due to the recurrent formula in (1), the difference between values may explode or shrink as the size of a Hankel matrix increase. Consequently, the resulting singular values may also explode or shrink to zero. This explosion can be critical for numerical computations of approximate SVD since it could lead to floating point overflow.

Normalizing the sequence $a = (a_1, \ldots, a_{n+m-1})$ with the 2-norm, i.e., $a' = a/\|a\|_2$, max reduce this effect, since it will normalize the singular values such that the largest singular value is unity. However, due to the ill-conditioning of really large Hankel matrices constructed using SSA-LRF, large matrices fails to achieve the prescribed rank. This is demonstrated in Figure 1(a), where random Hankel matrices with rank $r = 50$ are constructed and its singular values are computed with LAPACK gesdd.

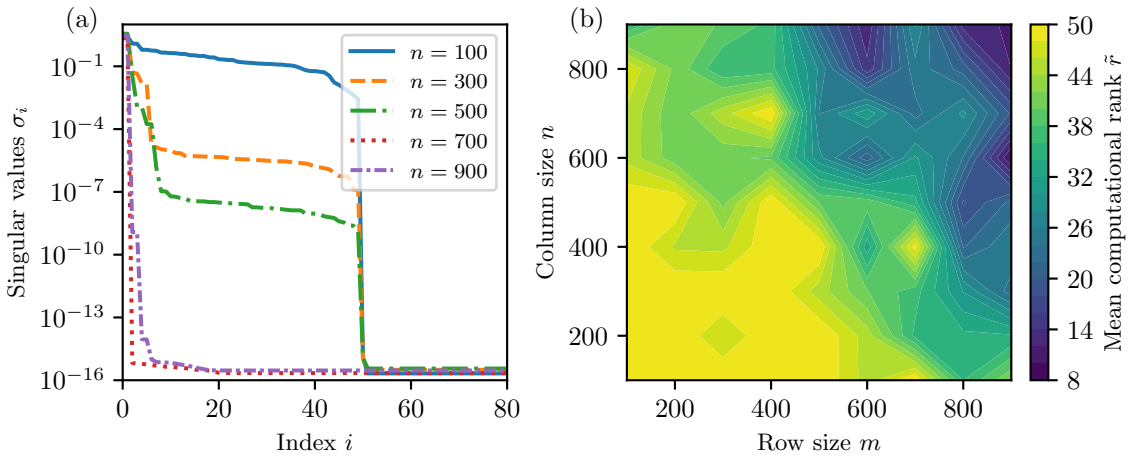


Figure 1: (a) Singular value decay of random Hankel matrices with size $n \times n$ and rank $r = 50$.(b) Mean rank at prescribed tolerance $\epsilon^* = 10^{-14}$ of Hankel matrices with size $m \times n$. Both cases are generated using normalized SSA-LRF.

In the singular value decay of SSA-LRF random Hankel matrices, it can be seen that a clear cut-off between dominant and non-dominant singular values vanish for large matrix sizes. Moreover, for large matrices, there seems to be a steep decline leading to Hankel matrices with extremely low ranks.

Similar pattern is also evident in the mean rank computations of different matrix sizes, where large matrix sizes deviate significantly from its prescribed rank. From the matrix rank computation in Figure 1 (b) it can be deduced that to obtain a reliable low-rank Hankel matrix $H \in \mathbb{R}^{m\times n}$ using SSA-LRF, the rank has to be chosen such that of $r \gtrapprox 10^{-1}m$ and $r \gtrapprox 10^{-1}n$.

Another alternative to this is a model of free-induction decay (FID) signals often used in nuclear magnetic resonance (NMR) spectroscopy. In this thesis, a non oscillating variant of the signal is used to restrict the Hankel to real values.

Remark B.2 ([37]). *Let $H = Hankel_{mn}(a_1, a_2, \ldots, a_{n+m-1}) \in \mathbb{R}^{m\times n}$ (see (2.11)) be a Hankel matrix. Then $rank(H) \leq r$ if the sequence $(a_1, a_2, \ldots, a_{n+m-1}) \in \mathbb{R}^{n+m-1}$ is given by,*

$$a_j = \sum_{i=1}^{r} c_i e^{-\tau_i j}, \tag{2}$$

for $j = 1, 2, \ldots n + m - 1$ *where* $c_1, c_2, \ldots, c_r \in \mathbb{R}$ *and* $\tau_1, \tau_2, \ldots, \tau_r > 0$.

An FID signal can be constructed using a superposition of multiple decaying exponentials. In this case, the rank bound of the Hankel matrix H corresponds to the number exponential terms in the signal which corresponds to intensity peaks in the Fourier transform of the signal [38].

While the rank properties of the Hankel matrix generated by FID signal models are only bounded by r, the elements constructed by the decaying exponentials are immediately normalized and are better conditioned to that of SSA-LRF.

This behavior can be observed in Figure 2.

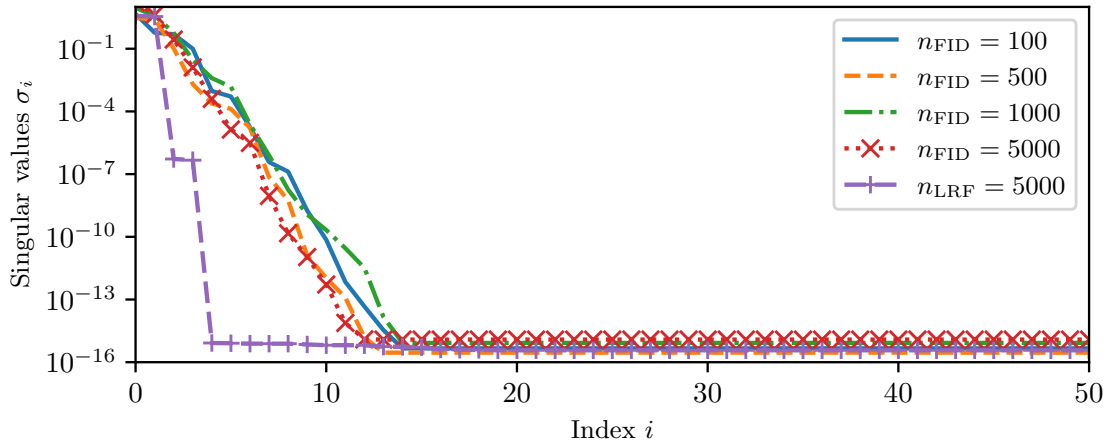


Figure 2: Singular value decay of random Hankel matrices with size $n \times n$ and rank $r = 50$ generated using FID signal modelling. A comparison with SSA-LRF generation for a 5000×5000 Hankel matrix is provided.

While FID signals does not seem to give a clear cut-off for all matrix sizes, its gradual decline allows us to obtain a matrix in which it singular values has different level of importance. This reduces the risk of truncating significant components of the matrix, which allows us to study the behavior of the approximate SVD computed by HASVD under gradual change of singular values.

C Rank and errors of randomly generated Hankel matrices

The following shows the mean relative Frobenius errors and the mean approximation rank of randomly generated Hankel matrices described in Subsection 5.2.2:

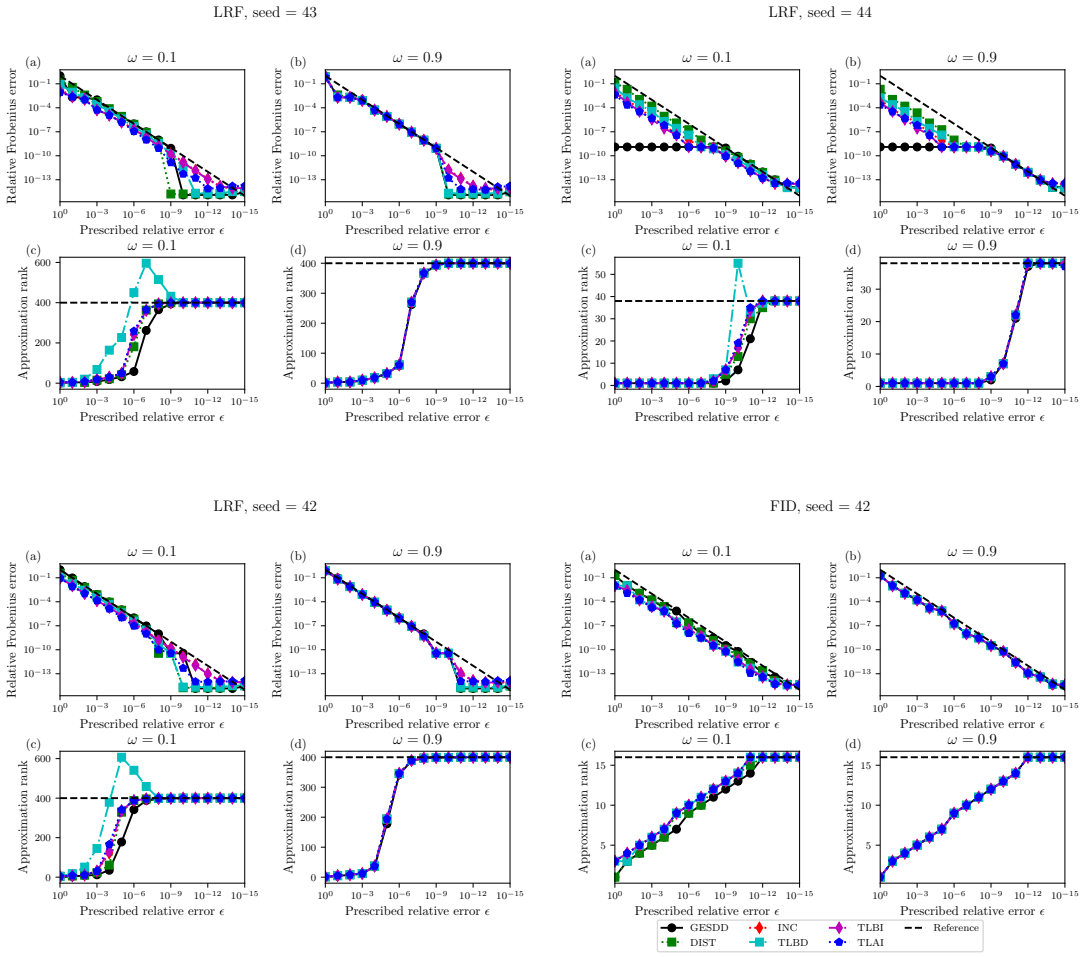


Figure 3: Different relative Frobenius errors and approximation ranks generated with FID and SSA-LRF sequences.

The characteristic of a Hankel matrix generated by FID sequence that has gradual singular value decline is reflected in the way the relative mean errors in of all SVD algorithms cannot reach a cut-off point.

Seed 44 for SSA-LRF sequence demonstrates the volatility of generating random Hankel matrices with this method, eventhough the prescribed rank is $r = 400$.

D HASVD runtime for general matrices

Presented in this appendix. is the runtime of HASVD for general matrices. Similar setup for matrix preparations are done to that of Subsection 5.2.4. While only DIST, INC, TLBD, and TLBI trees are investigated.

An interesting trend is seen in Figure 4(c,d). shows that runtime seems to improve over partitioning number $N = 100$ and then shows a decline in performance. This seems to coincide with results from HAPOD and computations with Hankel matrix in Subsection 5.2.4, where a balanced partitioning also shows similar trend [15].

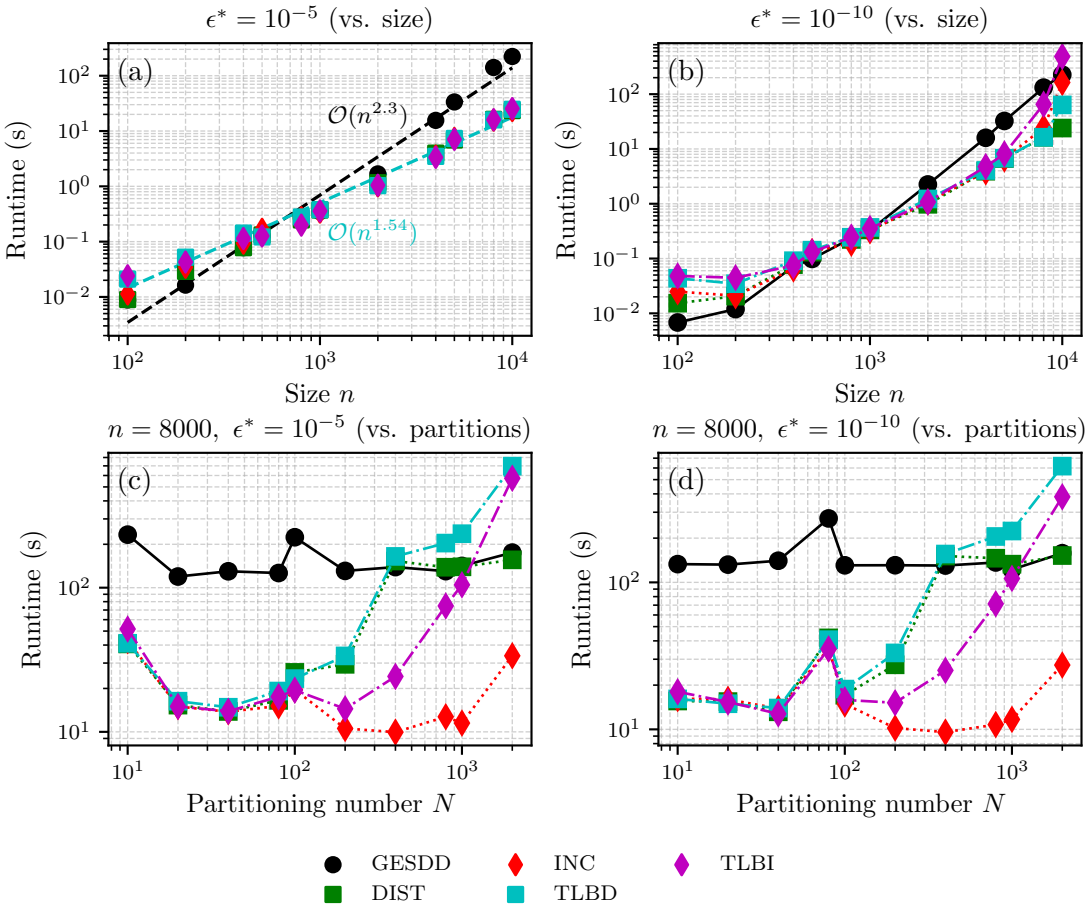
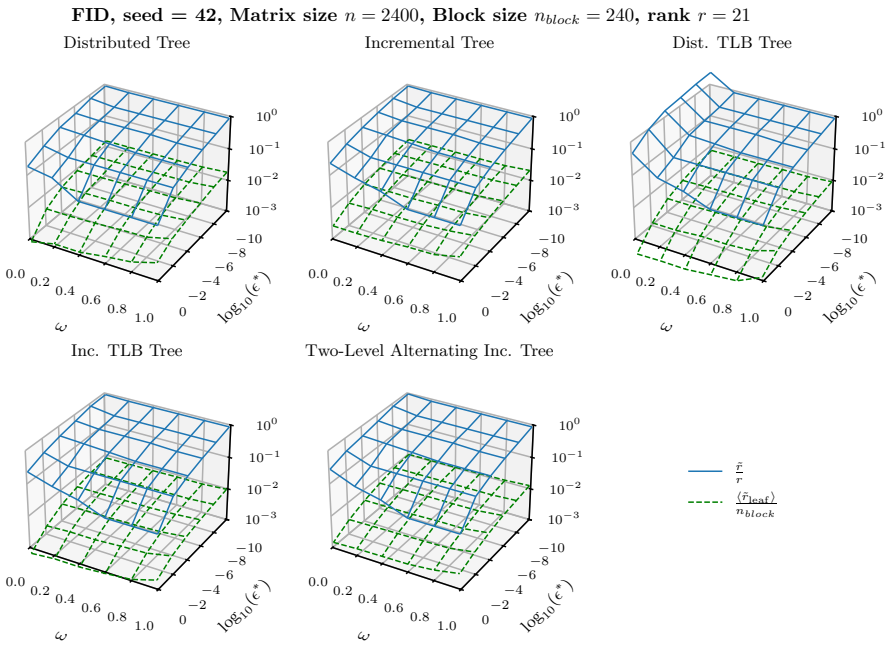
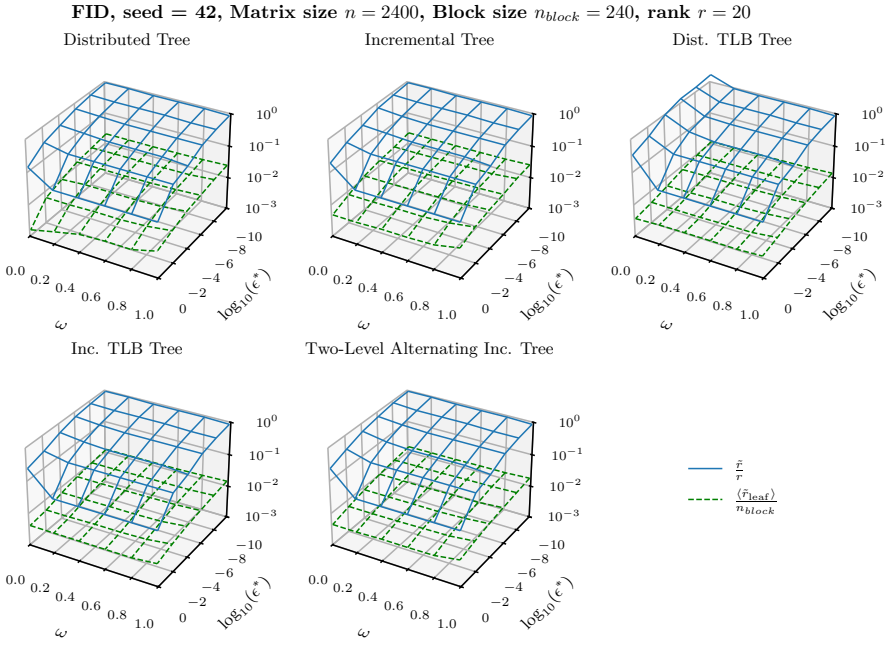


Figure 4: Runtimes in seconds(s) of matrix with conditioning $\kappa = 10^3$, $\omega = 0.1$.

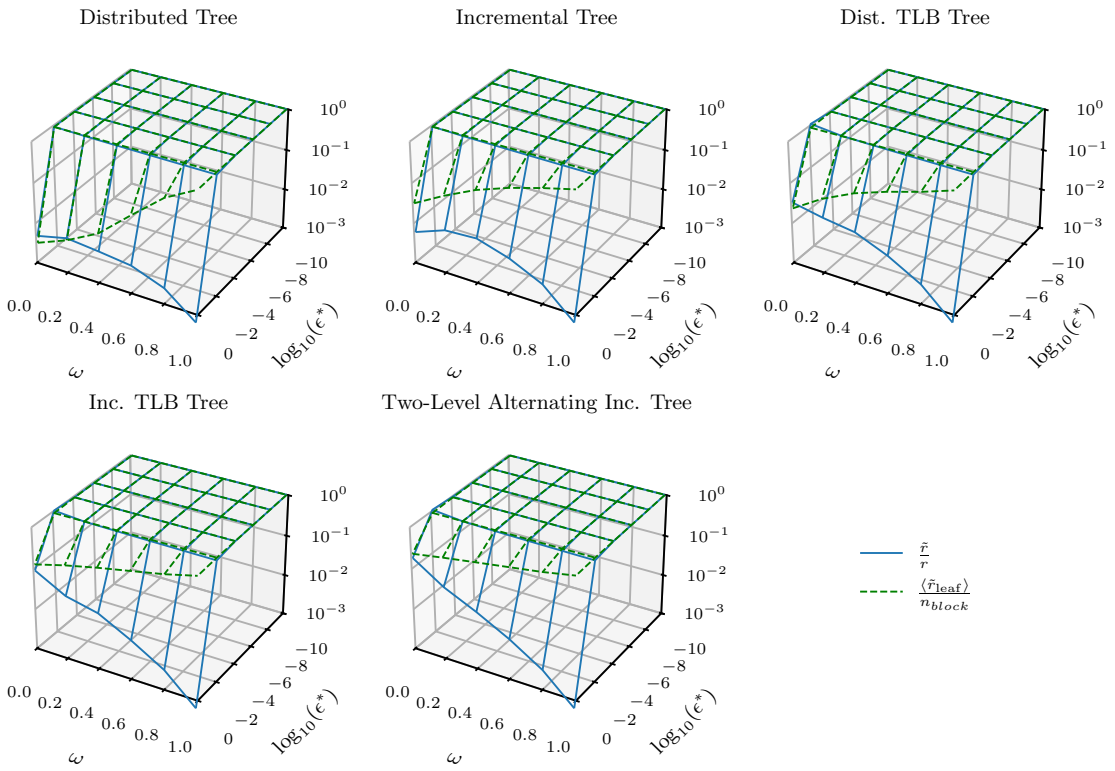
E Ratio profiles of randomly generated Hankel matrices

These ratio profiles are generated with the Mersenne Twister engine with seed of 42. The experimental setup follows from the description in Subsection 5.2.3.

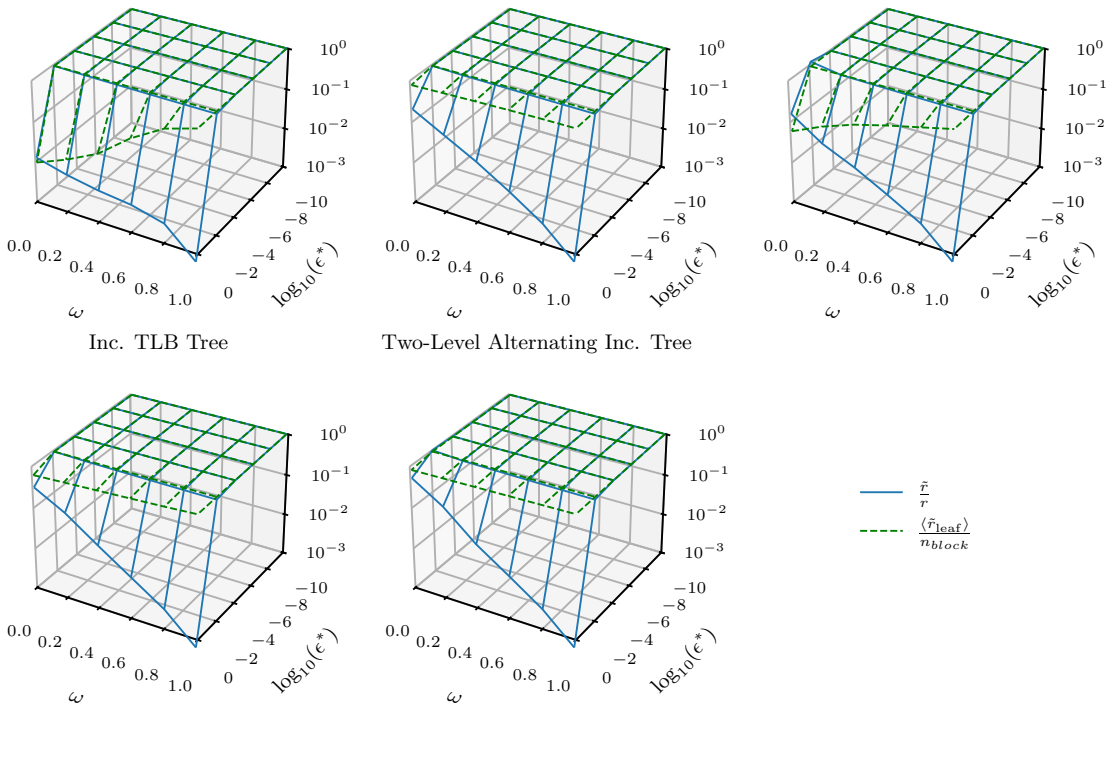




LRF, seed = 42, Matrix size $n = 2400$, Block size $n_{block} = 800$, rank $r = 1600$

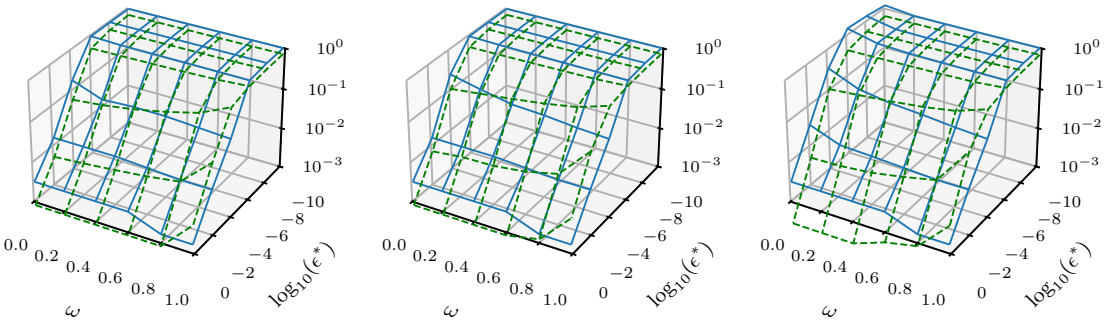


LRF, seed = 42, Matrix size $n = 2400$, Block size $n_{block} = 240$, rank $r = 1600$

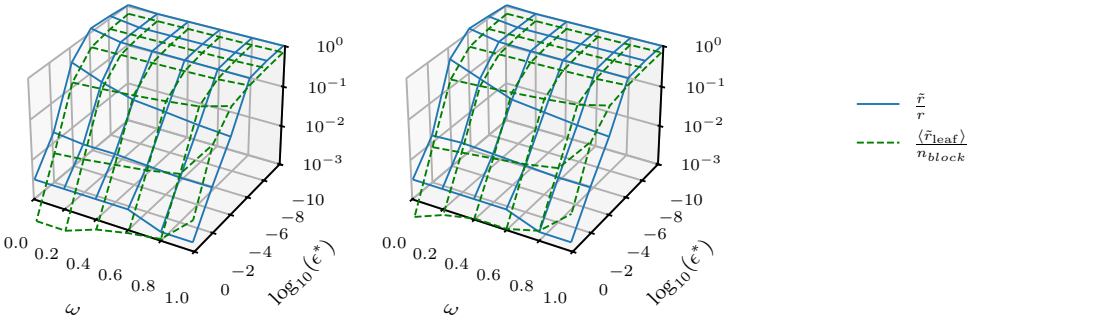


LRF, seed = 42, Matrix size $n = 2400$, Block size $n_{block} = 800$, rank $r = 600$

Distributed Tree | Incremental Tree | Dist. TLB Tree

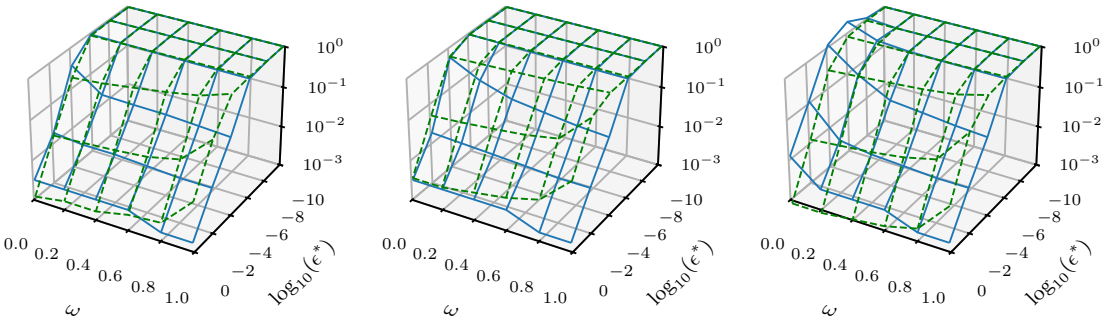


Inc. TLB Tree | Two-Level Alternating Inc. Tree



LRF, seed = 42, Matrix size $n = 2400$, Block size $n_{block} = 240$, rank $r = 600$

Distributed Tree | Incremental Tree | Dist. TLB Tree



Inc. TLB Tree | Two-Level Alternating Inc. Tree