

# Algorithms for Matrix Approximations with Time Varying Systems

Stephan Nüßlein



TUM



Master's thesis

# Algorithms for Matrix Approximations with Time Varying Systems

Stephan Nüßlein

June 29, 2022



Chair of Data Processing  
Technische Universität München



Stephan Nüßlein. *Algorithms for Matrix Approximations with Time Varying Systems*. Master's thesis, Technische Universität München, Munich, Germany, 2022.

Supervised by Prof. Dr.-Ing. Klaus Diepold and Matthias Kissel; submitted on June 29, 2022 to the Department of Electrical and Computer Engineering of the Technische Universität München.

© 2022 Stephan Nüßlein

Chair of Data Processing, Technische Universität München, 80290 München, Germany, <http://www.ldv.ei.tum.de/>.

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

## Abstract

There are different approaches to approximate matrices using structured matrices to reduce the computational cost of matrix-vector multiplications. A possible structure are sequentially semiseparable matrices, that describe the input-output behavior of time varying systems. If time varying systems are used to approximate weight matrices from neural networks, structural parameters have to be determined. In this thesis two algorithms to obtain the structural parameters are described. The first refines an initial segmentation by optimizing the input and output dimensions of the system. The second algorithm recursively splits the subsystems in a way that makes it possible to recover permuted sequentially semiseparable matrices. In experiments, the algorithms were able to recover the structure of simple test matrices. When used to approximate weight matrices from neural networks, the algorithms were able to reduce the computational cost of the matrix approximation compared to a naive approximation.



# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>Literature Review</b>	<b>11</b>
2.1	Matrix Structures . . . . .	11
2.1.1	Semiseparable Matrices . . . . .	11
2.1.2	Hierarchical Matrices . . . . .	12
2.1.3	Sequentially Semiseparable Matrices . . . . .	13
2.2	Neural Networks . . . . .	14
<b>3</b>	<b>Background</b>	<b>17</b>
<b>4</b>	<b>Methods</b>	<b>25</b>
4.1	Matrix Approximations . . . . .	25
4.1.1	Balanced Truncation of Ordered Systems . . . . .	25
4.1.2	Computing Singular Values of Hankel matrices . . . . .	27
4.1.3	Error Bound for Matrix Approximation . . . . .	30
4.2	Cost of Computation . . . . .	32
4.3	Number of Stages . . . . .	33
4.4	Segmentation Adaptation . . . . .	36
4.4.1	Moving Boundaries . . . . .	36
4.4.2	Singular Values of Hankel Matrices when Moving Boundaries . . . . .	40
4.4.3	Optimizing Input and Output Dimensions . . . . .	43
4.5	Permutations . . . . .	45
4.5.1	Split Stage . . . . .	47
4.5.2	Matrix Segmentation . . . . .	50
<b>5</b>	<b>Experiments</b>	<b>57</b>
5.1	Segmentation Adaption . . . . .	57
5.1.1	Illustrative Example . . . . .	57
5.1.2	Weight Matrix Approximation . . . . .	59
5.2	Permutations . . . . .	63
5.2.1	Illustrative Example . . . . .	63
5.2.2	Weight Matrix Approximation . . . . .	63
<b>6</b>	<b>Discussion</b>	<b>67</b>

*Contents*

<b>7 Conclusion</b>	<b>69</b>
<b>A Matrix Approximation Error Bounds</b>	<b>71</b>
<b>B Algorithms to Move Bounds Up or Down</b>	<b>77</b>
<b>C Random Sequentially Semiseparable Matrices</b>	<b>81</b>



# 1 Introduction

The computational complexity for training and evaluating neural networks is increasing steadily [42]. As the neural networks get larger, the evaluation gets computationally more expensive. This is often problematic as computational resources are a limiting factor, especially on embedded or mobile systems. Linear layers in neural networks are usually represented with weight matrices. The evaluation of a neural network requires the calculation of a matrix-vector product, which has an algorithmic complexity in the order of  $\mathcal{O}(n^2)$  for a full unstructured matrix [17]. For large  $n$  the cost for computing the matrix-vector products will therefore dominate over the parts of the neural network whose costs grow linearly.

In other fields the quadratic growth of the computational cost for a matrix-vector product is also an issue. This led to research into structured matrices, where the cost of the matrix-vector product is of lower order. Some structured matrices are used to solve partial differential equations like  $\mathcal{H}$ -Matrices [16], others arise in different circumstances like semiseparable matrices [46], or describe time varying systems like sequentially semiseparable matrices [11]. All of them can be expressed not only as matrices but also in terms of other representations. It is possible to use these representations to construct algorithms that can compute the matrix-vector product efficiently. One example of this are circulant matrices that describe cyclic convolutions. Using the matrix describing the Discrete Fourier Transform, circulant matrices can be transformed into diagonal matrices. This allows us to obtain a faster algorithm for the product  $y = Tu$  by using the Fast Fourier Transform (FFT): We can first compute the FFT of the input, then multiply the vector elementwise with a weight vector. Finally, we use the inverse FFT to obtain the result  $y$ . In doing so, the original order of complexity of  $\mathcal{O}(n^2)$  can be reduced down to  $\mathcal{O}(n \log(n))$  [44].

Analogously, other structure can be used to represent weight matrices. A possible representation are sequentially semiseparable matrices that represent the input-output behavior of time varying systems. These systems consist of stages that can change over time. We can create a system that represents the matrix calculate the matrix-vector product using algorithms based on this representation. The input and output dimensions of the system induce an inherent segmentation of the sequentially semiseparable matrices. If certain sub blocks of the matrices are of low rank, then computing the matrix-vector product is far cheaper when using the system representation. The corresponding segmentation is often determined by an underlying physical system and therefore known. In contrast, weight matrices in neural networks do not necessarily have the same underlying structure. Even if the matrix is

## 1 Introduction

close to a sequentially semiseparable matrix the structural parameters are usually unknown. To represent the matrices, algorithms to derive these structural parameters are needed. Existing algorithms to calculate the state space representation for an arbitrary matrix require prior knowledge of the segmentation [8]. As weight matrices are usually not sequentially semiseparable, these systems are approximated to reduce the computational cost. A common approximation algorithm is the balanced truncation, that relies on the singular values of the Hankel matrices.

The goal of this thesis is to develop algorithms to represent matrices with time varying systems that do not require a prior knowledge of the segmentation. These algorithms should be able to reduce the computational cost while not increasing the approximation error. For this I guess an initial segmentation and subsequently use an algorithm to refine it. Or I start with an initial system that is split recursively. In both cases I need to compute the singular values of the Hankel matrices efficiently.

Chapter 2 reviews different matrix structures and their use in neural networks. In Section 2.1 different matrix structures related to sequentially semiseparable matrices are presented and in Section 2.2 approaches to use structured matrices in neural networks are collected. Chapter 3 introduces time varying systems, that are an essential foundation of this thesis. In Chapter 4 the methods to create a system approximating a matrix with unknown structural parameters will be presented. First, different prerequisites are discussed. In Section 4.1 an algorithm for the approximation of a system is explored and a strategy to compute the singular values of the Hankel matrices efficiently is presented. Additionally, an error bound for the matrix approximation is derived. The number of operations is derived in Section 4.2. The following Section 4.3 presents an approach to determine the number of stages for a system. Based on this, the input and output dimensions have to be determined. For this two algorithms are presented. The first algorithm, described in Section 4.4 starts with an initial guess and then refines the structure by adapting the input and output dimensions. First, an algorithm to change the input and output dimensions of a system is given. This algorithm is then used to optimize the segmentation of the matrix. The second algorithm, described in Section 4.5 starts with a simple system and then recursively refines this structure by splitting up the stages. In Chapter 5 the algorithms will be tested on random sequentially semiseparable matrices and on weight matrices. Chapter 6 will discuss the results and in Chapter 7 I will give some perspective on them.

In this thesis I use a Matlab-like notation for the indexing of vectors and matrices. Given a vector

$$a = [a_1, a_2, \dots, a_n]^T, \quad (1.1)$$

the notation  $a_{[i]}$  denotes the  $i$ -th element of  $a$ . Double points are used to denote ranges according to

$$a_{[:i]} = [a_1, \dots, a_{i-1}, a_i]^T \quad (1.2)$$

and

$$a_{[i:]} = [a_i, a_{i+1}, \dots, a_n]^\top \quad (1.3)$$

To index the last element I use the notation  $a_{[\text{end}]}$  and to index elements from the end, I use the notation

$$a_{[\text{end}-i]} = a_{n-i} \quad (1.4)$$

These notations can analogously be used for a matrix. Here  $A_{[i,j]}$  denotes the  $j$ -th element in the  $i$ -th row.

To simplify the notation, I use the shorthand notation

$$\begin{bmatrix} X \\ Y \end{bmatrix} \leftarrow Z \quad (1.5)$$

instead of

$$X \leftarrow Z_{[i,:]} \quad Y \leftarrow Z_{[i+1,:]} \quad (1.6)$$

with an index  $i$  that will be clear from the context.



## 2 Literature Review

In the next section different matrix structures are described. In Section 2.2 some approaches to use matrix structures in neural networks are presented.

### 2.1 Matrix Structures

In the following several matrix structures are presented. These have in common that they have low rank sub-matrices. These are semiseparable, hierarchical and sequentially semiseparable matrices.

#### 2.1.1 Semiseparable Matrices

Semiseparable matrices are not consistently defined in the literature. In this thesis the definitions described by Vandebril [46, 47] are used. An important differentiation are generator representable semiseparable matrices and semiseparable matrices.

**Generator Representable Semiseparable Matrix** A matrix  $S$  is a generator representable semiseparable matrix if the lower and upper triangular parts of  $S$  are taken from rank 1 matrices. This can be expressed as

$$\text{triu}(S) = \text{triu}(pq^\top) \quad (2.1)$$

$$\text{tril}(S) = \text{tril}(uv^\top) \quad (2.2)$$

Where  $\text{triu}$  is the upper triangular matrix and  $\text{tril}$  is the lower triangular matrix. The vectors  $p, q, u$  and  $v$  are called the generators. It is important to note here that the diagonal of  $S$  is both included in the lower and upper triangular matrix.

**Semiseparable Matrix** In a semiseparable matrix every subblock selected from the lower triangular part of  $S$  has rank 1. The analogous statement has to be fulfilled for the upper triangular part. This can be formalized as

$$\text{rank}(S_{[i:n, 1:i]}) \leq 1 \quad \forall_i \in \{i, \dots, n\} \quad (2.3)$$

$$\text{rank}(S_{[1:i, i:n]}) \leq 1 \quad \forall_i \in \{i, \dots, n\} \quad (2.4)$$

An extension of this matrix class are the semiseparable plus diagonal matrices.

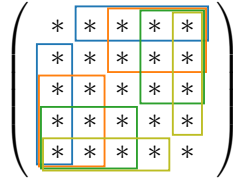


Figure 2.1: Illustration of a quasiseparable matrix

**Quasiseparable Matrix** Quasiseparable matrices are similar to the semiseparable matrices. In a quasiseparable matrix every subblock selected from the strictly lower strictly upper triangular part of  $S$  has rank 1. This can be formalized as

$$\text{rank}(S_{[i+1:n, 1:i]}) \leq 1 \quad \forall i \in \{i, \dots, n\} \quad (2.5)$$

$$\text{rank}(S_{[1:i, i+1:n]}) \leq 1 \quad \forall i \in \{i, \dots, n\} \quad (2.6)$$

A quasiseparable matrix is illustrated in Figure 2.1. All the marked submatrices have the property that their rank is 1. As the quasiseparable structure does not impose conditions on the diagonal it is more general than the semiseparable matrices.

There is a relation between invertible semiseparable matrices and invertible tridiagonal matrices [47]. The inverses of a generator representable semiseparable matrix is a irreducible tridiagonal matrix and vice versa. The inverse of a semiseparable matrix is a tridiagonal matrix and vice versa. If an invertible quasiseparable matrix is inverted, the inverse is again a quasiseparable matrix. These relations can be proven with results by Fiedler and Markham [14].

These matrix classes can also be extended for higher ranks. A matrix  $S$  is a generator representable semiseparable matrix of semiseparability rank  $r$  if there exist the matrices  $R_1$  and  $R_2$  with  $\text{rank}(R_1) = r$  and  $\text{rank}(R_2) = r$  such that

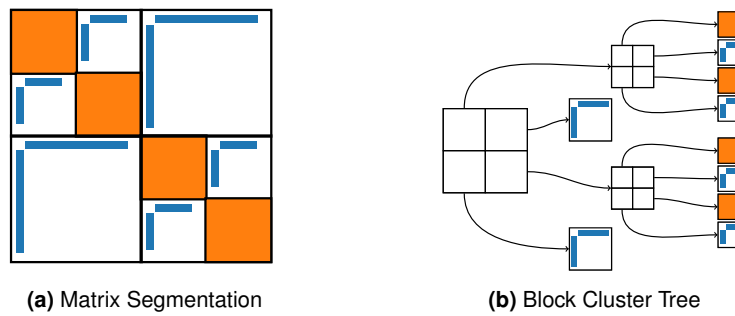
$$\text{triu}(S) = \text{triu}(R_1) \quad (2.7)$$

$$\text{tril}(S) = \text{tril}(R_2). \quad (2.8)$$

A similar definition for semiseparable matrices of semiseparability rank  $r$  is given in [46].

### 2.1.2 Hierarchical Matrices

The Hierarchical matrices ( $\mathcal{H}$ -Matrices) are a matrix structure to approximate large matrices. These were mainly introduced by Hackbusch [17] and Grasedyck [16]. A short introduction can also be found in [15]. The  $\mathcal{H}$ -Matrices were developed for the solution of PDEs. If a PDEs is solved numerically, it has to be discretized in order to obtain an approximated solution. As the discretization already introduces errors, it is advantageous to drop the requirement that the matrix representation is exact, if this



**Figure 2.2:** Illustration of the structure of a  $H$ -Matrix

results in a reduction of the computational cost. This can be done by partitioning the matrix in segments. These blocks are represented by low rank matrices. If the rank is far smaller than the size of the matrix this representation is cheaper in terms of storage and in terms of computational cost. The partitioning is done by hierarchically dividing blocks that cannot be represented using a low rank representation. To decide if a block has to be divided, an admissibility condition is introduced. This is a way to predict the representability of a matrix block. For discretizations of PDEs this admissibility condition is usually based on the geometrical distance. For other applications different admissibility conditions have to be derived.

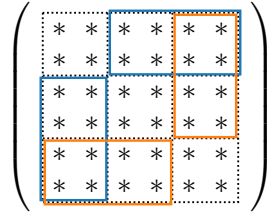
An  $\mathcal{H}$ -Matrix is shown in Figure 2.2a. In this case the matrix is divided in four blocks. If a block is admissible, it is stored as a low rank representation. These are illustrated as the white rectangles. If a block is not admissible, the block is divided in smaller subblocks. If matrices are already small and still not admissible the matrices are stored directly. In Figure 2.2a these blocks are colored orange. As the partition is done in a hierarchical fashion the matrices can be represented in a block-tree. The block-tree for the matrix in Figure 2.2a is illustrated in Figure 2.2b. The Hierarchical matrices make it possible to compute different matrix operations efficiently.

### 2.1.3 Sequentially Semiseparable Matrices

Sequentially semiseparable matrices are described in detail in the book by Dewilde and van der Veen[11]. These matrices can represent time varying systems. As the sequentially semiseparable matrices will be used in this thesis an introduction will be given in Chapter 3.

The sequentially semiseparable matrices are divided into blocks. Matrices taken from the strict lower and strict upper triangular blockmatrix have the condition that their rank is low. This is illustrated in Figure 2.3.

This segmentation in blocks makes similar to the hierarchical matrices that also have a segmentation. Unlike semiseparable matrices the sequentially semiseparable matrices do not have to be quadratic.



**Figure 2.3:** Illustration of a sequentially semiseparable matrix. The thick dotted lines mark the segmentation of the matrix. The colored lines represent submatrices with low rank.

## 2.2 Neural Networks

In this section different approaches that use structured matrices in neural networks are shortly introduced. There are two main approaches to obtain structured matrices: In one case the structure is predetermined and the parameters of the structure are trained by a regular training process [2, 9, 13, 21, 29]. Or the neural network is trained without a predetermined matrix structure and the matrices are later approximated with structured matrices [18, 23, 39, 50]. In [51] the model was retrained after the approximation. There are also approaches where the approximation is done during the training procedure like [10] or a structure is created using regularization [31, 49].

One widely known approach to reduce the complexity of neural networks is pruning. These are processes that set elements in neural network matrices to zero to reduce the computational cost [5]. This is equivalent to removing connections between neurons. An approach to remove connections from a trained network based on the Hessian of the loss function was proposed by Hassibi [18]. It is also possible to obtain sparse matrices while training. Dettmers and Zettlemoyer use an algorithm that includes pruning steps during the training after every epoch [10]. A different approach to obtain sparse matrices is including a regularization term. As an example Louizos et al. use  $L_0$  regularization [31] and Wen et al. use LASSO regularization [49].

Low rank approximations for filters in convolutional layers were proposed by Rigamonti et al. [39]. These use combinations of separable filters. Jaderberg et al. approximate existing filters using low rank filters [23]. Ioannou et al. directly learn low rank filters and combine them later [21]. In low rank and sparse decomposition a matrix is approximated with a low rank and a sparse matrix. Yu et al. represented matrices in deep models as a sum of low rank and sparse matrices [51]. The computation of the decomposition is based on an algorithm to calculate a low rank and sparse decomposition by Zhou and Tao [52]. To avoid an accumulation of errors due to the compression of multiple layers, the decomposition are not done independently. This is achieved by defining the objective function as the difference between



the outputs of the approximation for a collection of approximated input vectors and the corresponding output vectors.

The structure of Hierarchical matrices will be explained in subsection 2.1.2. These matrices have also been used in machine learning. Fan et al. used the structure of  $\mathcal{H}$ -Matrices in neural networks to solve PDEs [13]. There the structure of the neural network makes use of the different scales in the system. Hierarchical tensor decompositions were used by Wu et al. to represent weight matrices and convolutional kernels [50]. Ithapu used hierarchical factorization of covariance matrices to explore relationships between classes [22].

A different matrix structure explored in neural networks are butterfly matrices. These are products of sparse matrices that overall have a similar structure to the fast Fourier transform [29, 36]. Ailon et al. used this matrix structure to represent dense layers [2]. The butterfly structure was combined with CNNs by Li et al. [28]. This structure can be used to efficiently represent Fourier kernels. An extension of butterfly matrices are Kaleidoscope matrices (K-matrices). These were introduced by Dao et al. in [9]. K-Matrices are the product of factors of the shape  $B_a B_b^\top$ , where  $B_a$  and  $B_b$  are butterfly matrices. These can be used to represent a wide class of structured matrices. The patterns in the factors are fixed. Therefore the parameters can be learned using gradient descent. K-matrices can also be used to represent linear hand crafted structures like reprocessing steps.



### 3 Background

Sequentially semiseparable matrices can be considered as descriptions of time varying systems.<sup>1</sup>

A causal time varying system  $\Sigma_{[A,B,C,D]}$  can be described using the formulas

$$x_{k+1} = A_k x_k + B_k u_k \quad (3.1a)$$

$$y_k = C_k x_k + D_k u_k. \quad (3.1b)$$

The vectors  $u_k$  are the inputs, the vectors  $y_k$  are the outputs and the vectors  $x_k$  are the states of the system. The matrices  $A_k$ ,  $B_k$ ,  $C_k$  and  $D_k$  can vary with the time index  $k$ . This is different from time-invariant systems, where the matrices  $A$ ,  $B$ ,  $C$  and  $D$  are constant. This also means that the dimensions of the input, the output and the states can vary with time. The structure of a system is represented in Figure 3.1a. We can see that the system is structured in stages. The stage  $k$  consists of the matrices  $A_k$ ,  $B_k$ ,  $C_k$  and  $D_k$ . The matrix  $A_k$  maps the old state  $x_k$  to the next state  $x_{k+1}$ , the matrix  $B_k$  maps the current input  $u_k$  to the next state  $x_{k+1}$ . The matrix  $C_k$  maps the old state  $x_k$  to the output  $y_k$ . Finally the matrix  $D_k$  directly maps the input  $u_k$  to the output  $y_k$ . The system defined in Equation 3.1 is a causal system. This means that the output  $y_k$  only depends on the current input  $u_k$  and the previous inputs  $u_l$  with  $l < k$ . It is also possible to define an anticausal system where the output only depends on the current and the later inputs. An anticausal system is described using the formula

$$x_{k-1} = E_k x_k + F_k u_k \quad (3.2a)$$

$$y_k = G_k x_k + D_k u_k. \quad (3.2b)$$

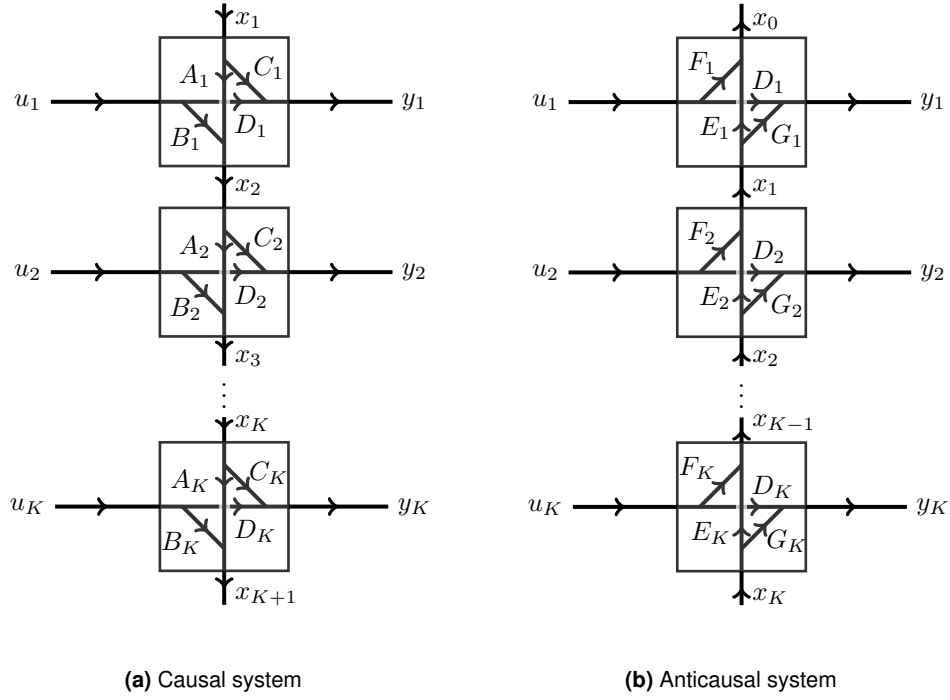
Figure 3.1b illustrates an anticausal system. The structure is analogous except for the reversed direction of the arrows for the states.

When we stack the inputs for the stages to a combined input vector  $u = [u_1^\top, \dots, u_k^\top]^\top$  and stack the outputs of all stages to a combined output vector  $y = [y_1^\top, \dots, y_k^\top]^\top$  we can describe the input-output behavior of the system using a single matrix-vector product

$$y = Tu, \quad (3.3)$$

<sup>1</sup>The naming and structure of the matrices is not consistent in the literature. In this thesis I will use the notation used in [45]. In [11] a similar notation is used, but the transfer operator is transposed. In other works like [8, 38] the causal and anticausal parts are considered jointly.

### 3 Background



**Figure 3.1:** Illustration of the structure of time varying systems.

where the matrix  $T$  is called the transfer operator. The transfer operator  $T$  for a causal system with four stages is

$$T_{\text{causal}} = \begin{bmatrix} D_1 & 0 & 0 & 0 \\ C_2 B_1 & D_2 & 0 & 0 \\ C_3 A_2 B_1 & C_3 B_2 & D_3 & 0 \\ C_4 A_3 A_2 B_1 & C_4 A_3 B_2 & C_4 B_3 & D_4 \end{bmatrix}. \quad (3.4)$$

For an anticausal system the matrix description is

$$T_{\text{anticausal}} = \begin{bmatrix} D_1 & G_1 F_2 & G_1 E_2 F_3 & G_1 E_2 E_3 F_4 \\ 0 & D_2 & G_2 F_3 & G_2 E_3 F_4 \\ 0 & 0 & D_3 & G_3 F_4 \\ 0 & 0 & 0 & D_4 \end{bmatrix}. \quad (3.5)$$

The matrices for a higher number of stages are analogous. We can see that the causal system results in a lower triangular blockmatrix whereas the anticausal system results in an upper triangular blockmatrix. If we want to represent the whole matrix we can use mixed systems. Mixed Systems can be defined as the sum of a causal and an anticausal system. These usually have the same input and output dimensions, while the state dimensions can be different for the causal and anticausal

part. When adding the systems, the  $D_k$ -matrices have to be added. As the use of two separate  $D_k$  does usually not have any benefits, the  $D_k$ -matrices for the anticausal system are usually set to 0 and can therefore be ignored. When representing matrices, the first state  $x_1$  and the last state  $x_{n+1}$  of the causal system are usually zero dimensional. This means that the system does not have an initial or remaining state. For the anticausal system this is analogous.

**Hankel Operator and Minimality** When we are interested in properties of the system, we can study the relation between the inputs, the states and the outputs. The reachability Matrix  $\mathcal{R}_k$  is the mapping from the inputs to the state  $x_k$ . For a causal system the reachability matrix for state  $x_k$  can be written as

$$\mathcal{R}_k = [\cdots \quad A_{k-1}A_{k-2}B_{k-1} \quad A_{k-1}B_{k-2} \quad B_{k-1}] \quad (3.6)$$

The mapping from the state  $x_k$  to the output is called the observability Matrix  $\mathcal{O}_k$ . For a causal system the observability matrix for state  $x_k$  can be written as

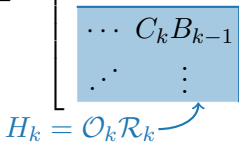
$$\mathcal{O}_k = \begin{bmatrix} C_k \\ C_{k+1}A_k \\ C_{k+2}A_{k+1}A_k \\ \vdots \end{bmatrix}. \quad (3.7)$$

When we multiply these we obtain the mapping from the inputs to the outputs. This is called the Hankel operator  $H_k$ . For a causal system this can be written as

$$H_k = \mathcal{O}_k \mathcal{R}_k = \begin{bmatrix} \cdots & C_k A_{k-1} B_{k-2} & C_k B_{k-1} \\ \cdots & C_{k+1} A_k A_{k-1} B_{k-2} & C_{k+1} A_k B_{k-1} \\ \ddots & \vdots & \vdots \end{bmatrix}. \quad (3.8)$$

The same structure can be found in the transfer operator  $T$

$$T = \begin{bmatrix} \ddots & & & & \\ \cdots & D_{k-1} & & & \\ \cdots & C_k B_{k-1} & D_k & & \\ \ddots & \vdots & \vdots & \ddots & \end{bmatrix} \quad (3.9)$$

$H_k = \mathcal{O}_k \mathcal{R}_k$  

If the rows of  $\mathcal{R}_k$  are linearly independent, the range of  $\mathcal{R}_k$  is  $\mathbb{R}_{d_k}^d$ , where  $d_k$  is the number of dimensions of  $x_k$ . This means that we can reach every state with an according input. Therefore we call a system reachable if every  $\mathcal{R}_k$  has a full row rank.

### 3 Background

If the columns of  $O_k$  are linearly independent, we can reconstruct the state  $x_k$  from the output. Therefore we call a system observable if every  $R_k$  has a full column rank.

If a system is both observable and reachable, it is called minimal. In a minimal system the state dimension cannot be further reduced without a loss of information. The rank of the Hankel operator is called the Hankelrank. If the system is minimal we have

$$\text{rank}(H_k) = \text{rank}(O_k) = \text{rank}(\mathcal{R}_k) = d_k \quad (3.10)$$

for every  $k$ . Usually the Hankel matrices are low rank matrices.. This allows us to factor the  $H_k$  in a tall and a wide matrix. This results in the factorization  $H_k = O_k \mathcal{R}_k$ .

**State Transforms** As the aforementioned factorization is not unique, the corresponding state is not uniquely defined, as well. The state can be transformed with a non-singular state transformation matrix  $S$  according to  $\tilde{x}_k = S_k x_k$ . This results in a transformed reachability matrix  $\tilde{\mathcal{R}}_k = S_k \mathcal{R}_k$  and the transformed observability matrix  $\tilde{O}_k = O_k S_k^{-1}$ . The Hankel matrix stays the same as  $\tilde{H}_k = O_k S_k^{-1} S_k \mathcal{R}_k = O_k \mathcal{R}_k = H_k$ . If a matrix factorization is used to construct a state transformation, the state transformation can be considered as moving a matrix from the reachability matrix to the observability matrix

$$O_k R_k = \underbrace{O_k A}_{\tilde{O}_k} \underbrace{B}_{\tilde{\mathcal{R}}_k} = \tilde{O}_k \tilde{\mathcal{R}}_k \quad (3.11)$$

or moving a matrix from the observability matrix to the reachability matrix.

$$O_k R_k = \underbrace{A}_{\tilde{O}_k} \underbrace{B \mathcal{R}_k}_{\tilde{\mathcal{R}}_k} = \tilde{O}_k \tilde{\mathcal{R}}_k \quad (3.12)$$

An algorithm working directly on the state space model makes it possible to reduce the state dimension of non-minimal systems, as the resulting state transformation matrix  $S_k$  does not need to be invertible.

There are three particular types of factorization that are called canonical forms. An input-normal realization has the property that the columns of each  $O_k$  are orthonormal. Whereas in an output-normal realization, all rows of each  $\mathcal{R}_k$  are orthonormal. The balanced realization results from the reduced Singular Value Decomposition (SVD)  $H_k = U_k \Sigma_k V_k^\top$ . In a balanced realization  $O_k = U_k \Sigma_k^{1/2}$  and  $\mathcal{R}_k = \Sigma_k^{1/2} V_k^\top$ . These canonical forms usually also require that the system is minimal. In this thesis the minimality is usually omitted, as the algorithms described also work on non-minimal systems and minimality is nontrivial to define for systems where the singular values decay slowly. If minimality is required, it will be explicitly stated.

**Matrix Factorization** Sequentially semiseparable matrices can also be considered as a matrix factorization. For a causal system, the matrix  $T$  can be expressed as a product  $T = T_n T_{n-1} \dots T_2 T_1$  where every  $T_k$  represents a stage. The matrix  $T_k$  is constructed according to

$$T_k = \left[ \begin{array}{c|ccc} A_k & & & B_k \\ \hline & I & & \\ & & \ddots & \\ & & & I \\ C_k & & & D_k \\ & & & & I & & \\ & & & & & \ddots & \\ & & & & & & I \end{array} \right]. \quad (3.13)$$

For a system with 3 stages this gives the factorization

$$T = \begin{bmatrix} A_3 & 0 & 0 & B_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ C_3 & 0 & 0 & D_3 \end{bmatrix} \cdots \begin{bmatrix} A_1 & B_1 & 0 & 0 \\ C_1 & D_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} A_3 A_2 A_1 & A_3 A_2 B_1 & A_3 B_2 & B_3 \\ C_1 & D_1 & 0 & 0 \\ C_2 A_1 & C_2 B_1 & D_2 & 0 \\ C_3 A_2 A_1 & C_3 A_2 B_1 & C_3 B_2 & D_3 \end{bmatrix}$$

As  $A_1$  has zero columns and  $A_3$  has zero rows, the first block-column as well as the top block-row disappear, resulting in the familiar structure shown in Equation 3.4.

An analogous factorization is possible for anticausal systems. Here the ordering is reversed  $T = T_1 T_2 \dots T_{n-1} T_n$

Sequentially semiseparable matrices can be added, multiplied and inverted using algorithms that work on the state space description. It is also possible to calculate QR factorization [8, 45] or URV factorization [7]. There are also a couple of other algorithms that can be used for controller design like sign iterations [38].

**Approximations** It is also possible to reduce the number of states by approximating a system  $\Sigma_{[A,B,C,D]}$  with a system  $\Sigma_{[\hat{A},\hat{B},\hat{C},\hat{D}]}$ . If the system  $\Sigma_{[A,B,C,D]}$  is minimal, this is not trivial, as a reduction of the number of states also results in a reduction of the Hankelrank. One approach is the Hankel-Norm approximation. The Hankel-Norm

$$\|T\|_H = \sup_i \|H_i\|. \quad (3.14)$$

is the supremum over the spectral norm (the matrix 2-norm) of each individual Hankel matrix. In [11] an algorithm is given for the Hankel norm approximation that computes an approximated  $\hat{T}$ . This is based on the results from Adamjan, Arov, and Kreĭn [1]. The approximated system satisfies the condition

$$\|\Gamma^{-1}(T - \hat{T})\|_H \leq 1. \quad (3.15)$$

### 3 Background

Here  $\Gamma$  is a diagonal and hermition operator. If we set  $\Gamma = I\gamma$  we obtain the simplified condition

$$\|T - \hat{T}\|_H \leq \gamma. \quad (3.16)$$

This problem has no unique solution, as the smaller singular values can be changed as long as they remain smaller than the supremum.

A second approach is balanced truncation, as described in [20, 40]. Here the state dimension  $d_k$  is reduced to a new dimension  $\hat{d}_k$ . The main idea of the balanced truncation is to remove states that result in a small output  $y$ , or need a big input  $u$  to be reached. We can calculate the norm of the output for a state  $x$  using

$$\|y\|_2^2 = y^\top y = x^\top \mathcal{O}^\top \mathcal{O} x. \quad (3.17)$$

The matrix  $\mathcal{O}^\top \mathcal{O}$  is called the Observability Gramian. Analogously, we can calculate the norm of the input required to reach a certain state using

$$\|u\|_2^2 = x^\top (\mathcal{R}\mathcal{R}^\top)^{-1} x. \quad (3.18)$$

The Reachability Gramian  $\mathcal{R}\mathcal{R}^\top$  is invertible if the system is reachable. The problems depend on the basis for the states. This basis can be changed using state transformations. The idea is now to obtain a state transformation, for which both problems are equivalent. This is the case for a balanced realization

$$\mathcal{O}^\top \mathcal{O} = \Sigma^{1/2} U^\top U \Sigma^{1/2} = \Sigma = \Sigma^{1/2} V^\top V \Sigma^{1/2} = \mathcal{R}\mathcal{R}^\top. \quad (3.19)$$

This results in a basis for the state with

$$\|y\|_2^2 \Big|_{x=e_l} = \|\mathcal{O}x\|_2^2 \Big|_{x=e_l} = \sigma_l \quad (3.20)$$

and

$$\|u\|_2^2 \Big|_{x=e_l} = \sigma_l^{-1}. \quad (3.21)$$

Here  $e_l$  is the  $l$ -th standard basis vector. This allows us to only keep the states with  $\|y\| > \epsilon^{1/2}\|x\|$  and  $\|x\| > \epsilon^{1/2}\|u\|$  for some  $\epsilon > 0$  by cutting all dimensions  $l$  with  $\sigma_l < \epsilon$ .

To obtain the reduced system the matrices of a balanced realization for every state are partitioned according to

$$A_k = \begin{bmatrix} A_{k[11]} & A_{k[12]} \\ A_{k[21]} & A_{k[22]} \end{bmatrix} \quad B_k = \begin{bmatrix} B_{k[1]} \\ B_{k[2]} \end{bmatrix} \quad (3.22a)$$

$$C = \begin{bmatrix} C_{k[1]} & C_{k[2]} \end{bmatrix} \quad D_k = D_k. \quad (3.22b)$$

The dimensions are determined by the new state dimensions. The approximated system  $\Sigma_{[\hat{A}, \hat{B}, \hat{C}, \hat{D}]}$  is then set to

$$\hat{A}_k = A_{k[11]} \quad \hat{B}_k = B_{k[1]} \quad (3.23a)$$

$$\hat{C}_k = C_{k[1]} \quad \hat{D}_k = D_k. \quad (3.23b)$$



This approach is equivalent to removing all  $\sigma_l < \epsilon$  of the SVD of the Hankel operator. Therefore we can also construct an approximation when identifying a system using the SVD by truncating the SVD as described in [43]. The error  $\|T - \hat{T}\|$  can be bound to two times the cut singular values [27]. A detailed discussion of approxiamtions is given in [3].



## 4 Methods

This chapter presents the methods and algorithms derived in this thesis. Section 4.1 explains the techniques to calculate the singular values of the Hankel matrices and how to use them to approximate matrices. The cost to compute a matrix-vector product using a time varying system is shown in Section 4.2 and used in Section 4.3 to choose the number of stages. In Section 4.4 a method to adapt the segmentation is presented and in Section 4.5 a algorithm to recover permuted sequentially semiseparable is explained.

### 4.1 Matrix Approximations

In this section more details of the balanced truncation are derived. First, the class of ordered systems is presented, which makes it possible to use the balanced truncation on certain systems that are not balanced. Next a algorithm to compute the required singular values is given. Finally an error bound for the approximation of matrices is derived.

#### 4.1.1 Balanced Truncation of Ordered Systems

In Chapter 3 the balanced truncation is defined for balanced systems. These have the property that  $\mathcal{O}_k = U_k \Sigma_{\mathcal{O}_k}$  and  $\mathcal{R}_k = \Sigma_{\mathcal{R}_k} V_k^\top$  where  $\Sigma_{\mathcal{O}_k} = \Sigma_{\mathcal{R}_k} = \Sigma_k^{1/2}$ . The balanced truncation is also possible for a wider class of realizations. In this thesis these realizations will be called *ordered*.

**Definition 1** (Ordered Realization). *A system is ordered if*

$$\mathcal{O}_k^\top \mathcal{O}_k = \Sigma_{\mathcal{O}_k}^2 \quad (4.1a)$$

$$\mathcal{R}_k \mathcal{R}_k^\top = \Sigma_{\mathcal{R}_k}^2 \quad (4.1b)$$

where  $\Sigma_{\mathcal{O}_k}$  and  $\Sigma_{\mathcal{R}_k}$  are diagonal matrices and the diagonal entries of  $\Sigma_k = \Sigma_{\mathcal{O}_k} \Sigma_{\mathcal{R}_k}$  are non increasing (i.e.  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n$ ) for every  $k$ .

An ordered system can be transformed into a balanced system using the diagonal state transformations  $S_k = \Sigma_k^{1/2} \Sigma_{\mathcal{R}_k}^{-1}$  and the inverse  $S_k^{-1} = \Sigma_{\mathcal{O}_k}^{-1} \Sigma_k^{1/2}$ . This gives

#### 4 Methods

the balanced system  $\Sigma_{[\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}]}$

$$\tilde{A}_k = S_{k+1} A_k S_k^{-1} \quad \tilde{B} = S_{k+1} B \quad (4.2a)$$

$$\tilde{C}_k = C_k S_k^{-1} \quad \tilde{D} = D \quad (4.2b)$$

*Proof.* By applying the state transformations to the observability and reachability matrices we get a balanced system

$$\mathcal{O}_k S_k^{-1} = U_k \Sigma_{\mathcal{O}_k} \Sigma_{\mathcal{O}_k}^{-1} \Sigma_k^{1/2} = U_k \Sigma_k^{1/2}$$

$$S_k \mathcal{R}_k = \Sigma_k^{1/2} \Sigma_{\mathcal{R}_k}^{-1} \Sigma_{\mathcal{R}_k} V_k^\top = \Sigma_k^{1/2} V_k^\top \quad \square$$

A ordered system can be reduced using the same segmentation as used for the balanced truncation.

*Proof.* The reduction of an ordered system is equivalent to the reduction of a balanced system if both reduced systems are equivalent. As the state transformations are diagonal matrices, the result of first transforming it to a balanced system and then reducing it using balanced truncation is equivalent to directly reducing the ordered system.

The ordered system  $\Sigma_{[A, B, C, D]}$  is first transformed to a balanced system  $\Sigma_{[\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}]}$  as described in Equation 4.2. Thereafter, the matrices are segmented as described in Equation 3.22. This results in the block matrices

$$\tilde{A}_k = \begin{bmatrix} S_{k+1[1]} A_{k[11]} S_{k[1]}^{-1} & S_{k+1[1]} A_{k[12]} S_{k[2]}^{-1} \\ S_{k+1[2]} A_{k[21]} S_{k[1]}^{-1} & S_{k+1[2]} A_{k[22]} S_{k[2]}^{-1} \end{bmatrix} \quad \tilde{B}_k = \begin{bmatrix} S_{k+1[1]} B_{k[1]} \\ S_{k+1[2]} B_{k[2]} \end{bmatrix} \quad (4.3a)$$

$$\tilde{C}_k = \begin{bmatrix} C_{k[1]} S_{k[1]}^{-1} & C_{k[2]} S_{k[2]}^{-1} \end{bmatrix} \quad \tilde{D}_k = D_k. \quad (4.3b)$$

The matrices  $S_{k[1]}$  and  $S_{k[2]}$  are obtained by segmenting the state transformations according to

$$S_k = \begin{bmatrix} S_{k[1]} & 0 \\ 0 & S_{k[2]} \end{bmatrix}. \quad (4.4)$$

The segmentation is determined by the new state dimension. This results in the reduced system  $\Sigma_{[\hat{A}, \hat{B}, \hat{C}, \hat{D}]}$  with

$$\hat{A}_k = S_{k+1[1]} A_{k[11]} S_{k[1]}^{-1} \quad \hat{B}_k = S_{k+1[1]} B_{k[1]} \quad (4.5)$$

$$\hat{C}_k = C_{k[1]} S_{k[1]}^{-1} \quad \hat{D}_k = D_k. \quad (4.6)$$

This system can also be obtained by first reducing the ordered system and then transforming it using  $S_{k+1[1]}$ .  $\square$

This allows us to use algorithms that require input normal or output normal forms, as systems can be both ordered and input normal if  $\Sigma_{\mathcal{R}_k} = I$  or both ordered and output normal if  $\Sigma_{\mathcal{O}_k} = I$ .

**Recursive factorization of  $\mathcal{O}_k$  and  $\mathcal{R}_k$**  The goal of this section is to construct an algorithm calculating the singular values of the Hankel matrices without applying the SVD on  $H_k$  or calculating  $H_k$ ,  $\mathcal{O}_k$  or  $\mathcal{R}_k$ . The idea is that we can recursively factor the observability matrix, as  $\mathcal{O}_k$  contains both the mapping from the state  $x_k$  to the output  $y_k$  and the mapping from  $x_k$  to the later outputs. The first mapping is represented by  $C_k$ . The second mapping is represented by  $\mathcal{O}_{k+1}A_k$ . This gives the recursive factorization

$$\mathcal{O}_k = \begin{bmatrix} C_k \\ \mathcal{O}_{k+1}A_k \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+1} \end{bmatrix} \begin{bmatrix} C_k \\ A_k \end{bmatrix}. \quad (4.7)$$

Analogously the reachability matrix can be factorized as

$$\mathcal{R}_k = \begin{bmatrix} A_{k-1}\mathcal{R}_{k-1} & B_{k-1} \end{bmatrix} = \begin{bmatrix} A_{k-1} & B_{k-1} \end{bmatrix} \begin{bmatrix} \mathcal{R}_{k-1} & 0 \\ 0 & I \end{bmatrix}. \quad (4.8)$$

#### 4.1.2 Computing Singular Values of Hankel matrices

The recursive factorizations of  $\mathcal{O}_k$  and  $\mathcal{R}_k$  allows us to create algorithms, that can convert realizations into input normal and output normal form. With a combination of both algorithms it is also possible to obtain the singular values of the Hankel matrices. The algorithms to convert to input and output normal forms are also given in [7]. Here a different derivation will be presented. Additionally, a way to use these algorithms to compute the SVD of the Hankel matrices will be explained.

**Conversion to input normal and reachable** An algorithm to transform the system into a reachable and input normal system is given. This means that all  $R_k$  have to fulfill  $\mathcal{R}_k^\top \mathcal{R}_k = I$ . As we do not want to calculate the reachability matrices directly, the algorithm employs the decomposition given in Equation 4.8. If  $\mathcal{R}_{k-1}^\top \mathcal{R}_{k-1} = I$  the next reachability matrix  $\mathcal{R}_k$  can be transformed such that  $\tilde{\mathcal{R}}_k^\top \tilde{\mathcal{R}}_k = I$ . We calculate the SVD of  $\begin{bmatrix} A_{k-1} & B_{k-1} \end{bmatrix}$  which is used to obtain the new reachability matrix  $\tilde{\mathcal{R}}_k$  according to

$$\mathcal{R}_k = \begin{bmatrix} A_{k-1} & B_{k-1} \end{bmatrix} \begin{bmatrix} \mathcal{R}_{k-1} & 0 \\ 0 & I \end{bmatrix} = U_k \Sigma_k \underbrace{V_k}_{\tilde{\mathcal{R}}_k} \begin{bmatrix} \mathcal{R}_{k-1} & 0 \\ 0 & I \end{bmatrix}. \quad (4.9)$$

This gives the state transformation  $S_k = U_k \Sigma_k$ . The usage of the reduced SVD also removes states that are not reachable. The product with the inverse  $S_k^{-1}$  is implicitly calculated by computing the SVD. We can set the matrices  $\tilde{A}_k$  and  $\tilde{B}_k$  to  $[\tilde{A}_k, \tilde{B}_k] = V_k^\top$ . The matrices  $A_k$  and  $B_k$  have to be updated with the state transformation  $S_k$ . The algorithm is formalized in Algorithm 1

The result is input normal and reachable.

#### 4 Methods

**Input:** System  $\Sigma_{[A,B,C,D]}$   
Tolerance  $\text{tol}_r$

**Output:** Input normal system  $\Sigma_{[A,B,C,D]}$

**for**  $k \leftarrow 1$  **to**  $K - 1$  **do**

$U, \sigma, V^\top \leftarrow \text{reducedSVD}([A_k, B_k], \epsilon = \text{tol}_r)$

$[A_k, B_k] \leftarrow V^\top$

$A_{k+1} \leftarrow A_{k+1}U \text{diag}(\sigma)$

$C_{k+1} \leftarrow C_{k+1}U \text{diag}(\sigma)$

**end for**

**Algorithm 1:** Conversion to input normal system.

*Proof.* Input normality can be proven by induction:

**Base case:** As  $x_1$  is zero dimensional  $R_1$  vanishes and  $\tilde{\mathcal{R}}_1 \tilde{\mathcal{R}}_1^\top = I$ .

**Induction step:** From  $\tilde{\mathcal{R}}_{k-1} \tilde{\mathcal{R}}_{k-1}^\top = I$  follows that  $\tilde{\mathcal{R}}_k \tilde{\mathcal{R}}_k^\top = I$  as

$$\tilde{\mathcal{R}}_k \tilde{\mathcal{R}}_k^\top = V_k^\top \begin{bmatrix} \tilde{\mathcal{R}}_{k-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \tilde{\mathcal{R}}_{k-1}^\top & 0 \\ 0 & I \end{bmatrix} V_k = V_k^\top \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} V_k = I \quad (4.10)$$

This proves  $\tilde{\mathcal{R}}_k \tilde{\mathcal{R}}_k^\top = I$  for all  $k \in [1, \dots, K - 1]$  by induction. Reachability directly follows from  $\mathcal{R}_k \mathcal{R}_k^\top = I$ .  $\square$

**Conversion to output normal and observable** The algorithm to transform the system into an observable and output normal system is analogous. Here all  $\mathcal{O}_k$  have to fulfill  $\mathcal{O}_k \mathcal{O}_k^\top = I$ . We use the decomposition given in Equation 4.7 to avoid a computation of  $\mathcal{O}_k$ . If  $\mathcal{O}_{k+1} \mathcal{O}_{k+1}^\top = I$ , the algorithm transforms  $\mathcal{O}_k$  such that  $\tilde{\mathcal{O}}_{k+1} \tilde{\mathcal{O}}_{k+1}^\top = I$  by calculating the SVD of  $[C_k^\top A_k^\top]^\top$ . By inserting the SVD we can obtain the new observability matrix according to

$$\mathcal{O}_k = \begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+1} \end{bmatrix} \begin{bmatrix} C_k \\ A_k \end{bmatrix} = \underbrace{\begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+1} \end{bmatrix}}_{\tilde{\mathcal{O}}_k} U_k \Sigma_k V_k^\top \quad (4.11)$$

This results in the state transformation  $S_k = \Sigma_k V_k^\top$ . Here the SVD is also in a reduced form, which results in a removal of all non observable states. The state transformation with  $S_k^{-1}$  is done implicitly by the SVD. The matrices  $A_{k-1}$  and  $C_{k-1}$  are updated with the state transformation  $S_k$

This gives the Algorithm 2. The result is output normal and observable.

*Proof.* Output normality can be proven by induction:

**Base case:** As  $x_{K+1}$  is zero dimensional  $\mathcal{O}_{k+1}$  vanishes and  $\tilde{\mathcal{O}}_{K+1} \tilde{\mathcal{O}}_{K+1}^\top = I$

**Input:** System  $\Sigma_{[A,B,C,D]}$   
 Tolerance  $\text{tol}_o$

**Output:** Output normal system  $\Sigma_{[A,B,C,D]}$

**for**  $k \leftarrow K$  **downto** 2 **do**

$U, \sigma, V^\top \leftarrow \text{reducedSVD} \left( \begin{bmatrix} C_k \\ A_k \end{bmatrix}, \epsilon = \text{tol}_o \right)$

$\begin{bmatrix} C_k \\ A_k \end{bmatrix} \leftarrow U$

$A_{k-1} \leftarrow \text{diag}(\sigma) V^\top A_{k+1}$

$B_{k-1} \leftarrow \text{diag}(\sigma) V^\top C_{k+1}$

**end for**

**Algorithm 2:** Conversion to output normal system.

**Induction step:** From  $\tilde{\mathcal{O}}_{k+1}^\top \tilde{\mathcal{O}}_{k+1} = I$  follows that  $\tilde{\mathcal{O}}_k^\top \tilde{\mathcal{O}}_k = I$  as

$$\tilde{\mathcal{O}}_k^\top \tilde{\mathcal{O}}_k = U_k^\top \begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+1}^\top \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+1} \end{bmatrix} U_k = U_k^\top \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} U_k = I \quad (4.12)$$

This proves  $\tilde{\mathcal{O}}_k^\top \tilde{\mathcal{O}}_k = I$  for all  $k \in [K-1, \dots, 1]$  by induction. Reachability directly follows from  $\mathcal{O}_k \mathcal{O}_k^\top = I$ .  $\square$

**Obtaining sigmas** By converting an input normal system to an output normal system it is possible to calculate the singular values of  $H$ . This uses the factorization in Equation 4.7 to get

$$H = \mathcal{O}_k \mathcal{R}_k = \begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+1} \end{bmatrix} \begin{bmatrix} C_k \\ A_k \end{bmatrix} \mathcal{R}_k. \quad (4.13)$$

As the algorithm started with an input normal system,  $\mathcal{R}_k \mathcal{R}_k^\top = I$  is still fulfilled. And due to the fact the algorithm has transformed the states  $x_i$  for  $i > k$  to output normal form,  $\mathcal{O}_{k+1}^\top \mathcal{O}_{k+1} = I$  is already fulfilled. When the SVD is applied to the matrix  $\begin{bmatrix} C_k^\top & A_k^\top \end{bmatrix}^\top$  this results in the SVD of  $H$

$$H = \underbrace{\begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+1} \end{bmatrix}}_{U_k} \underbrace{\Sigma_k V_k^\top \mathcal{R}_k}_{\check{V}_k^\top} = \check{U}_k \Sigma_k \check{V}_k^\top \quad (4.14)$$

*Proof.* To prove that  $H = \check{U}_k \Sigma_k \check{V}_k^\top$  is the SVD we have to prove that  $\check{U}_k^\top \check{U}_k = I$  and  $\check{V}_k^\top \check{V}_k = I$ . If  $\mathcal{O}_{k+1}$ ,  $\mathcal{R}_k$ ,  $U_k$  and  $V$  are orthogonal the proof directly follows. For non-square matrices we have  $\check{V}_k^\top \check{V}_k = V_k^\top \mathcal{R}_k \mathcal{R}_k^\top V_k = V_k^\top I V_k = I$  and  $U_k^\top U_k = I$  has been proven in Equation 4.12  $\square$

## 4 Methods

As the resulting system is ordered, the system can be approximated using balanced truncation.

*Proof.* The condition in Equation 4.1a directly follows from the output normality  $\tilde{O}_k^\top \tilde{O}_k = I$ . The condition in Equation 4.1b follows from the input normality of the intermediate system by

$$\tilde{\mathcal{R}}_k \tilde{\mathcal{R}}_k^\top = \Sigma_k V_k \mathcal{R}_k \mathcal{R}_k^\top V_k^\top \Sigma_k = \Sigma_k^2 \quad (4.15)$$

As  $\Sigma_k$  results from an SVD, the singular values are nonnegative and in decreasing order. Therefore the system is ordered.  $\square$

This algorithm makes it possible to compute the singular values of a system without explicitly computing the Hankel matrices. Therefore, the algorithm is also a fast way to reduce the system to a minimal system. The tolerance for the first transformation  $\text{tol}_i$  is usually set to a value close to the machine precision to avoid errors due to the removed states. The tolerance for the second transformation  $\text{tol}_o$  is then set to the wanted  $\epsilon$ .

It is also possible to first convert the system to an output normal system and calculate the SVD when converting it to an input normal system. The proof is analogous. The proof and the algorithms for the anticausal case are omitted for brevity.

### 4.1.3 Error Bound for Matrix Approximation

In the following I give an upper bound for the error of the matrix approximation with the balanced truncation. The algorithm given in [8] makes it possible to create a mixed system representing an arbitrary matrix. The mixed system can be approximated by separately approximating the causal part  $T_c$  and the anticausal part  $T_a$ . This results in the transfer matrix of the approximated system

$$\tilde{T} = \tilde{T}_c + \tilde{T}_a \quad (4.16)$$

The error for the approximation of the causal and anticausal system can be bounded using the results from [27]. Using this bound one can obtain the trivial bound

$$\|T - \tilde{T}\| \leq \|T_c - \tilde{T}_c\| + \|T_a - \tilde{T}_a\|. \quad (4.17)$$

I derive a tighter bound for the matrix approximation. We start by considering a case where we only approximate the  $k$ -th state of the causal system and the  $k-1$ -th state of the anticausal system.

In this case the Hankel matrix of the causal system  $H_k$  and the Hankel matrix of the anticausal system  $H_{k-1}^*$ ,



$$T = \begin{bmatrix} \ddots & \vdots & \begin{matrix} \vdots \\ \vdots \\ \vdots \end{matrix} \\ \cdots & D_{k-1} & \begin{matrix} G_{k-1}F_k \\ \vdots \\ \vdots \end{matrix} \cdots \\ \cdots & C_k B_{k-1} & D_k \cdots \\ \begin{matrix} \vdots \\ \vdots \\ \vdots \end{matrix} & \vdots & \ddots \end{bmatrix}, \quad (4.18)$$

$H_k = \mathcal{O}_k \mathcal{R}_k$  (pointing to the bottom-left block) and  $H_{k-1}^* = \mathcal{O}_{k-1}^* \mathcal{R}_{k-1}^*$  (pointing to the top-right block)

are replaced with the low rank approximation  $\hat{H}_k$  and  $\hat{H}_{k-1}^*$ . The low rank approximation is constructed taking the first  $\hat{d}_k$  columns of  $\mathcal{O}_k$  and the first  $\hat{d}_k$  rows of  $\mathcal{R}_k$ . Analogously the anticausal Hankel matrix is approximated. These approximations are based on the SVD of  $H_k$  and  $H_{k-1}^*$  because a balanced system is used. According to the Schmidt–Mirsky Theorem the approximation is optimal for the spectral norm  $\|X\|$  and the Frobenius norm  $\|X\|_F$  [4].

This results in the approximated system with the transfer matrix  $\hat{T}^{(k)}$ . The approximation error is the norm of the matrix

$$T - \hat{T}^{(k)} = \begin{bmatrix} 0 & H_{k-1}^* - \hat{H}_{k-1}^* \\ H_k - \hat{H}_k & 0 \end{bmatrix}. \quad (4.19)$$

The SVD of this matrix can be obtained by combining the SVD of  $H_k - \hat{H}_k$  and  $H_{k-1}^* - \hat{H}_{k-1}^*$  to

$$T - \hat{T}^{(k)} = \begin{bmatrix} U_k \Sigma_k V_k^\top & U_{k-1}^* \Sigma_{k-1}^* V_{k-1}^{*\top} \end{bmatrix} = \begin{bmatrix} U_{k-1}^* & \\ & U_k \end{bmatrix} \begin{bmatrix} \Sigma_{k-1}^* & \\ & \Sigma_k \end{bmatrix} \begin{bmatrix} V_k^\top & V_{k-1}^{*\top} \end{bmatrix}$$

and then permuting the matrices such that the singular values are ordered in decreasing order. The matrices  $\Sigma_k$  and  $\Sigma_{k-1}^*$  contain the singular values corresponding to the cut states. Using the SVD, the spectral norm is

$$\|T - \hat{T}^{(k)}\| = \max \left( \|H_k - \hat{H}_k\|, \|H_{k-1}^* - \hat{H}_{k-1}^*\| \right) \quad (4.20)$$

and the Frobenius norm is

$$\|T - \hat{T}^{(k)}\|_F = \sqrt{\|H_k - \hat{H}_k\|_F^2 + \|H_{k-1}^* - \hat{H}_{k-1}^*\|_F^2} \quad (4.21)$$

If multiple states are approximated, the error from the individual approximations have to be combined.

In general

$$\hat{T} \neq T - \sum_{i=2}^K T - \hat{T}^{(i)} \quad (4.22)$$

## 4 Methods

is true. Nevertheless the total error in the spectral norm is bounded by

$$\|T - \hat{T}\| \leq \sum_{i=2}^K \|T - \hat{T}^{(i)}\| \quad (4.23)$$

and for the Frobenius norm by

$$\|T - \hat{T}\|_F \leq \sum_{i=2}^K \|T - \hat{T}^{(i)}\|_F. \quad (4.24)$$

*Proof.* The proof relies on the fact that  $\|T - \hat{T}^{(k)}\|$  is equal or larger than the actual change if multiple stages are approximated. The details can be found in Appendix A  $\square$

For the spectral norm this results in the upper bound

$$\|T - \hat{T}\| \leq \sum_{i=2}^K \epsilon \leq (K - 1)\epsilon \quad (4.25)$$

for the balanced truncation of a system with  $K$  stages, using an approximation parameter  $\epsilon$ , as all singular values of  $T - \hat{T}^{(k)}$  are smaller than  $\epsilon$  for all  $k \in 1, \dots, K - 1$ .

## 4.2 Cost of Computation

When using Time Varying Systems to approximate weight matrices, we are mainly interested in the cost of computing the product  $y = Tu$  using time varying systems. The cost is the number of Floating-Point Operations (FLOPs). The number of parameters is also important as this determines the required memory.

The cost depends on the state dimensions  $d_k$ , the input dimensions  $m_k$  and the output dimensions  $p_k$ .

First, the cost for a causal stage is derived: Without considering additions, the cost  $C_k$  for the stage  $k$  is

$$C_k = d_{k+1}d_k + d_{k+1}m_k + p_kd_k + p_km_k. \quad (4.26)$$

We can also include the additions and get the cost

$$C'_k = d_{k+1}(2d_k - 1) + d_{k+1}(2m_k - 1) + p_k(2d_k - 1) + p_k(2m_k - 1). \quad (4.27)$$

For anticausal system, the formulas are identical, except for the indexing of the state. Here symbols with a \* denote quantities for the anticausal system. Without additions this gives the cost for one stage

$$C_k^* = d_{k-1}^*d_k^* + d_{k-1}^*m_k + p_kd_k^* + p_km_k. \quad (4.28)$$

If the additions are included this results in the cost

$$C_k'^* = d_{k-1}^*(2d_k^* - 1) + d_{k-1}^*(2m_k - 1) + p_k(2d_k - 1) + p_k(2m_k - 1). \quad (4.29)$$

When considering mixed systems the costs for the causal and the anticausal system are added. As the  $D$ -matrices for the anticausal part are not needed the according FLOPs are irrelevant. This gives the cost for a mixed stage

$$C_k = d_{k+1}d_k + d_{k-1}^*d_k^* + (d_{k+1} + d_{k-1}^*)m_k + (d_k + d_k^*)p_k + p_k m_k. \quad (4.30)$$

Including the additions, the cost is

$$C_k' = d_{k-1}(2d_k - 1) + d_{k-1}^*(2d_k^* - 1) + (d_{k-1} + d_{k+1}^*)(2m_k - 1) + p_k(2d_k - 1) + p_k(2d_k^* + 1) + p_k(2m_k - 1). \quad (4.31)$$

The total cost is the sum over all stages

$$C = \sum_{k=1}^K C_k. \quad (4.32)$$

The number of parameters is equal to the number of multiplications. In the following I will only use the number of multiplications.

### 4.3 Number of Stages

To choose the appropriate number of stages  $K$ , I approximate how the cost depends on the number of stages. For now the dimensions  $d$ ,  $d^*$ ,  $p$  and  $m$  are considered as constant.

Then the cost in Equation 4.32 can be simplified to the multiplication

$$C = K(d^2 + d^{*2} + (d + d^*)m + (d + d^*)p + pm). \quad (4.33)$$

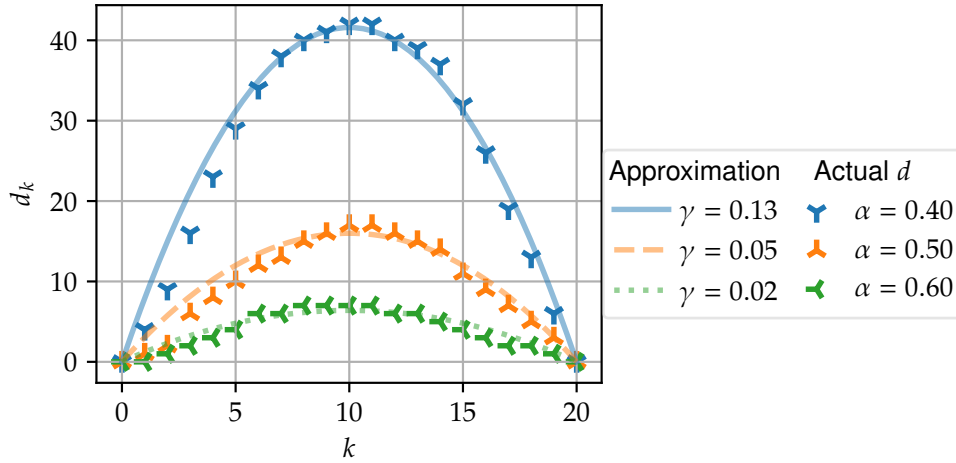
If the total number of inputs  $M$  and the total number of outputs  $P$  are divisible by  $K$ , the input and output sizes are related to the number of stages by  $m = M/K$  and  $p = P/K$ . This results in

$$C = K(d^2 + d^{*2} + (d + d^*)\frac{M}{K} + (d + d^*)\frac{P}{K} + \frac{PM}{K^2}) \quad (4.34)$$

$$= K(d^2 + d^{*2}) + (d + d^*)M + (d + d^*)P + \frac{1}{K}PM. \quad (4.35)$$

This relation describes how the general computational cost depends on the number of stages. This approximation is not helpful if we want to approximate matrices, as in these cases the state dimensions are usually not constant. Usually the state

#### 4 Methods



**Figure 4.1:** Approximated degree of a matrix from the *Mobilenet V2* model [41]. The actual degree is the number of singular values with  $\sigma > \alpha \|A\|_H$  for every causal Hankel matrix. Here  $\alpha$  is a hyperparameter that determines the accuracy. Additionally approximations of the rank with suitable parameters  $\gamma$  are plotted.

dimensions are small for the first and last states and larger for the states in the middle. Here, different approximations of the state dimensions have to be used. I used an approximation based on the properties of the singular values and their relation to the Frobenius norm. The singular values are related to the Frobenius norm by  $\|H\|_F = \sqrt{\sum \sigma_i^2}$  [4, p. 67]. If we suppose that the elements of  $H$  have a constant absolute value, then  $\|H\|_F^2 \propto \text{size}(H)$ . This means that the Frobenius norm increases with the size of the matrix, which in turn should also result in increasing singular values. Finally  $d_k$  is the number of singular values larger than a certain threshold value. The number of states is not directly related to the 2-norm of the singular values as it depends on their distribution and the threshold value. Regardless the approximation  $\check{d}_k \propto \|H_k\|_F^2$  fits the behavior of the considered matrices fairly well as illustrated in Figure 4.1. I approximate the degree using the relation

$$\check{d}_k = \gamma \text{size}(H_k) \frac{1}{\min(P, M)} \quad (4.36)$$

$$= \gamma (K - k + 1)(k - 1) \frac{PM}{K^2 \min(P, M)}. \quad (4.37)$$

The variable  $\gamma$  is a proportionality factor that represents the required accuracy. If a high accuracy is wanted, this results in a higher number of states. This can be modeled with a larger  $\gamma$ . If we do not need a high accuracy, we can use fewer states. This is modeled with a smaller  $\gamma$ . The factor  $1/\min(P, M)$  is used to normalize the state dimensions in such a way that for  $\gamma = 1$  the approximation is compatible to

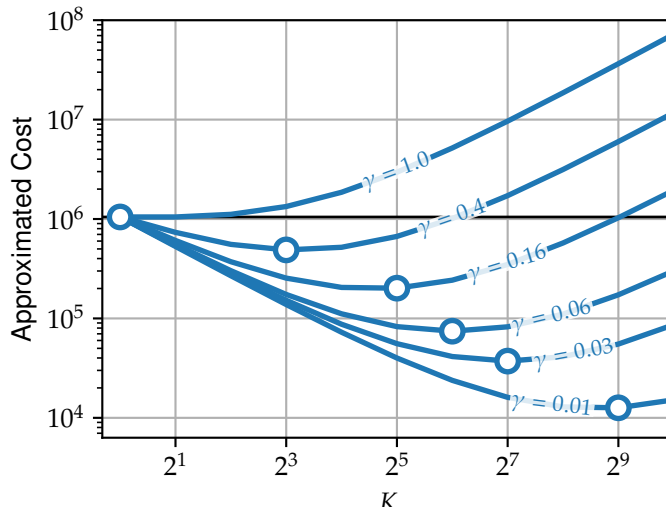
the maximum rank of  $H_k$ . As I suppose that the statedimensions for the anticausal case are equivalent to the statedimensions for the causal case, I can calculate the cost using

$$C = \sum_{k=1}^K \left( 2d_k d_{k+1} + \frac{2M d_{k+1}}{K} + \frac{2P d_k}{K} + \frac{MP}{K^2} \right). \quad (4.38)$$

As the matrix can be transposed,  $\min(P, M) = M$  can be assumed without loss of generality. By inserting the approximation from Equation 4.36 and transforming the sums using Faulhaber's formula [25] the cost is transformed to

$$C = \frac{KP^2\gamma^2}{15} + \frac{MP\gamma}{3} + \frac{P^2\gamma}{3} + \frac{MP - \frac{P^2\gamma^2}{3}}{K} + \frac{-\frac{MP\gamma}{3} - \frac{P^2\gamma}{3}}{K^2} + \frac{4P^2\gamma^2}{15K^3}. \quad (4.39)$$

This makes it possible, to calculate the cost for different values for the parameter  $\gamma$ . In Figure 4.2 this is done for an example with  $M = N = 2^{10}$ .



**Figure 4.2:** Approximated cost for different numbers of stages. The system has  $M = P = 2^{10}$  input and output dimensions. The cost is approximated for different values for the parameter  $\gamma$ . The optimal number of stages for every  $\gamma$  is marked with a circle.

We can see that for one stage the cost is equivalent to the cost of a simple matrix product, as the whole matrix is stored as one  $D$ -matrix. For  $\gamma = 1$  the cost only increases for higher number of stages. For  $\gamma \ll 1$  the cost first decreases before increasing later. The figure shows that the optimal number of stages  $\hat{K}$  increases as  $\gamma$  decreases. For very small  $\gamma$  I get  $\hat{K} = 1/2M$ . For more reasonable values like  $\gamma = 0.16$  the optimal number of stages is  $\hat{K} = 2^{-5}M$ . This gives an educated guess on the optimal number of stages  $K$  for a weight matrix.

## 4 Methods


The approximation has the downsides, that the parameter  $\gamma$  has to be estimated and the input and output dimensions are modeled as constant, which is not true for the tested systems. It is also possible to approximate the state dimension based on the maximum possible rank of the Hankel matrix. The rank has to be lower than  $\min(\text{width}(H_k), \text{height}(H_k))$ . This approximation might have some benefits for very tall or wide matrices, as this approximation is able to represent ranks that are not symmetric in  $k$ .

### 4.4 Segmentation Adaptation


In this section the segmentation of the matrix is changed. This means adapting the input and output dimensions. First I will describe how to move the boundaries. In Subsection 4.4.2 the algorithm is extended such that makes it possible to calculate the singular values of the Hankel matrices. In Subsection 4.4.3 a strategy to improve the segmentation is presented.

#### 4.4.1 Moving Boundaries

In this subsection algorithms to move the boundaries of a causal system are introduced. This is done by changing the segmentation of the input and output. When we change the segmentation of the input and output, this also changes the segmentation of the transfer matrix  $T$ . A change of the division between the inputs  $u_k$  and  $u_{k+1}$  moves the  $k$ -th vertical boundary left or right according to

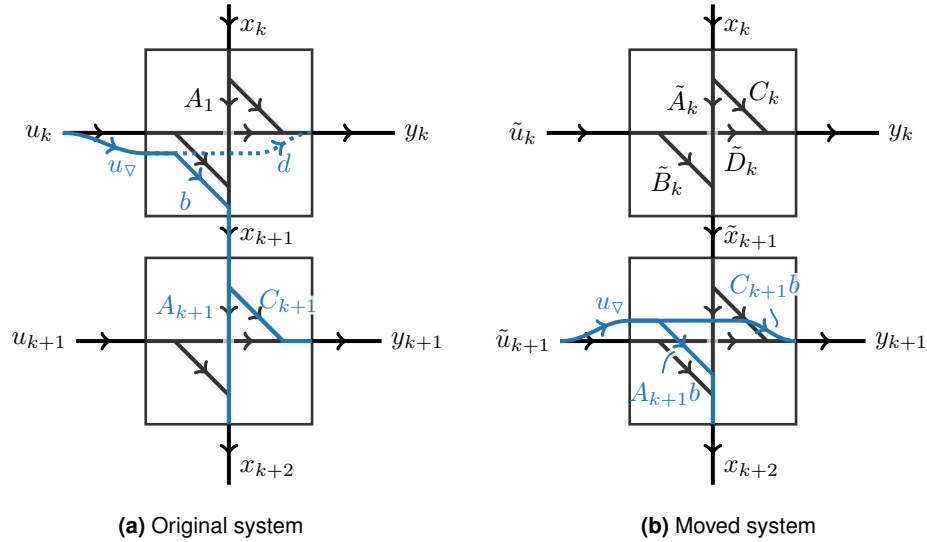
$$\begin{bmatrix} \vdots \\ y_k \\ y_{k+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & & & & & \\ \cdots & & D_k & & & \\ \cdots & & B_k C_{k+1} & D_{k+1} & & \\ \ddots & & \vdots & \vdots & \ddots & \end{bmatrix} \begin{bmatrix} \vdots \\ u_k \\ u_{k+1} \\ \vdots \end{bmatrix}. \quad (4.40)$$


A change of the division between the outputs  $y_k$  and  $y_{k+1}$  analogously moves the  $k$ -th horizontal boundary up or down according to

$$\begin{bmatrix} \vdots \\ y_k \\ y_{k+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & & & & & \\ \cdots & & D_k & & & \\ \cdots & & B_k C_{k+1} & D_{k+1} & & \\ \ddots & & \vdots & \vdots & \ddots & \end{bmatrix} \begin{bmatrix} \vdots \\ u_k \\ u_{k+1} \\ \vdots \end{bmatrix}. \quad (4.41)$$


The movement of the boundaries changes the parts of the matrix that can be represented with a causal system. When we move boundaries to the left or down,

some connections are no longer possible in a causal system. These will be dropped. Conversely when we move boundaries to the right or up, new connections will be established. For a mixed system these connections can be moved to the anticausal system and vice versa. The algorithms are constructed such that they also preserve the minimality of the system.



**Figure 4.3:** Illustration of a system where a boundary is moved to the left.

```

Input: System  $\Sigma_{[A,B,C,D]}$ , Index  $k$ , Tolerance  $\text{tol}$ 
Output: System  $\Sigma_{[A,B,C,D]}$  with changed input dimensions
     $b \leftarrow B_k[:, \text{end}]$ 
     $D_k \leftarrow D_k[:, : \text{end}-1]$ 
     $B_k \leftarrow B_k[:, : \text{end}-1]$ 
     $D_{k+1} \leftarrow [C_{k+1}b, D_{k+1}]$ 
     $B_{k+1} \leftarrow [A_{k+1}b, B_{k+1}]$ 
     $U, \sigma, V^\top \leftarrow \text{SVD}([A_k, B_k], \text{econ} = \text{True})$ 
    if  $\sigma_{[\text{end}]} < \text{tol}$  then
         $U \leftarrow U[:, : \text{end}-1]; \sigma \leftarrow \sigma[:, : \text{end}-1]; V \leftarrow V[:, : \text{end}-1]$ 
         $[A_k, B_k] \leftarrow \text{diag}(\sigma)V^\top$ 
         $A_{k+1} \leftarrow A_{k+1}U$ 
         $C_{k+1} \leftarrow C_{k+1}U$ 
    end if
    
```

**Algorithm 3:** Move boundary between  $u_k$  and  $u_{k+1}$  to the left.

## 4 Methods

**Move Left** First we move the  $k$ -th vertical boundary to the left. This means that the last element of the input  $u_k$  becomes the first element in  $u_{k+1}$ . This moved input is designated as  $u_{\nabla}$ . The altered connections are highlighted in Figure 4.3.

The vector  $b$  is the last column of  $B_k$  and describes the connection from  $u_{\nabla}$  to  $x_{k+1}$ . The vector  $d$  is the last column of  $D_k$  and describes the connection from  $u_{\nabla}$  to  $y_k$ . These vectors are removed from the matrices resulting in  $\check{B}_k$  and  $\check{D}$ . We can see that the connection  $d$  in the original system cannot be realized with a causal system as this would mean a connection from  $\tilde{u}_{k+1}$  to  $y_k$  in the transformed system. This also means that the parts of the transfer matrix  $T$  that can be represented using the causal system change. The dropped connection is illustrated using a dotted line in Figure 4.3. Now we have to add the the input  $u_{\nabla}$  to the next stage. The connection from the input  $u_{\nabla}$  to the output  $y_{k+1}$  can be incorporated by adding the vector  $C_{k+1}b$  to  $D_{k+1}$  as a new column. Analogously the connection from  $u_{\nabla}$  to the state  $x_{k+2}$  can be incorporated by attaching the vector  $A_{k+1}b$  to  $B_{k+1}$ .

After these changes, the mapping from the combined inputs  $[u_k^{\top} u_{k+1}^{\top}]^{\top}$  to the state  $x_{k+2}$  remains unchanged. Therefore the following states are still reachable and minimal. As the last column of  $\mathcal{R}_{k+1}$  is removed, the state  $x_{k+1}$  might become non reachable. This is the case if the matrix  $[A_b \check{B}_k]$  is no longer of full rank. We can check this using the SVD. If the last singular value is equal to zero, one state-dimension has to be removed. This can be done using a state transformation based on the reduced SVD. The matrices  $[\tilde{A}_b \tilde{B}_k]$  are set to  $\text{diag}(\sigma)V^{\top}$  and the following stages are transformed according to  $\tilde{A}_{k+1} = A_{k+1}U$  and  $\tilde{C}_{k+1} = C_{k+1}U$ . This restores the minimality of the system.

The pseudocode to move the boundary to the left is given in Algorithm 3.

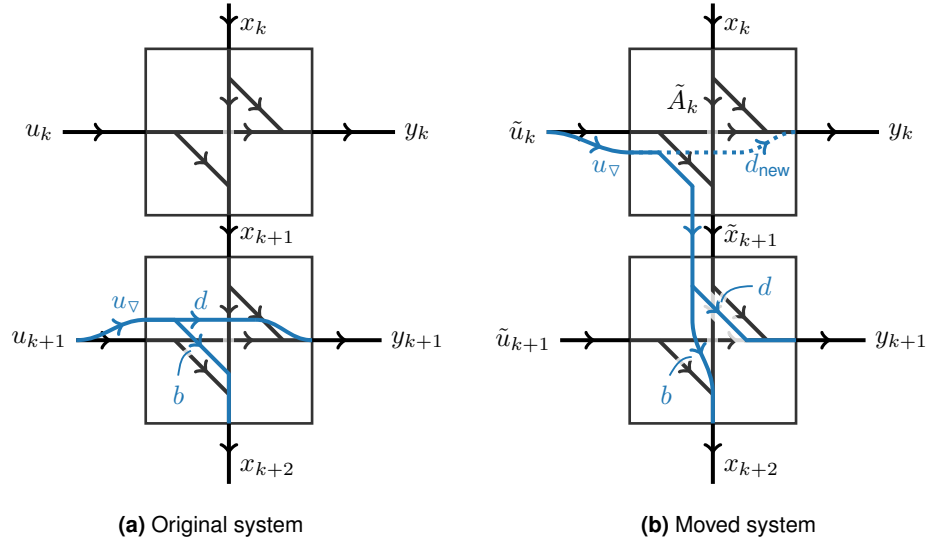
**Move Right** It is also possible to reverse these changes. When we move the  $k$ -th vertical boundary to the right, the input  $u_{\nabla}$  is moved from  $u_{k+1}$  to  $\tilde{u}_k$ . The connection from  $u_{\nabla}$  is moved from the current stage to the previous stage. We can do this by directly routing the input to an additional state dimension. To attach the input  $u_{\nabla}$  to the end of the state vector  $x_{k+1}$ , we set

$$\check{B}_k = \begin{bmatrix} B_k & 0 \\ 0 & 1 \end{bmatrix} \quad (4.42)$$

This makes the input  $u_{\nabla}$  available in the next stage. Now the connections can be reconnected. For this we remove first row  $d$  of  $D_{k+1}$  and attach it to  $C_{k+1}$  as the new last row. Analogously the first row  $b$  of  $B_{k+1}$  is removed and attached to the end of  $A_{k+1}$ . This is illustrated in Figure 4.4.

After these changes, the mapping from the combined inputs  $[u_k^{\top} u_{k+1}^{\top}]^{\top}$  to the state  $x_{k+2}$  remains unchanged. Therefore the following states are still reachable and minimal. This trivial approach results in a new observability matrix  $\tilde{\mathcal{O}}_{k+1}$  and a reachability matrix  $\tilde{\mathcal{R}}_{k+1}$  for the state  $x_{k+1}$ . The state  $x_k$  is still reachable, as the





**Figure 4.4:** Illustration of a system where a boundary is moved to the right.

additional statedimension is directly connected to the input  $u_{\nabla}$ . The observability matrix has one additional column

$$\tilde{\mathcal{O}}_{k+1} = \begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+2} \end{bmatrix} \begin{bmatrix} C_{k+1} & d \\ A_{k+1} & b \end{bmatrix}. \quad (4.43)$$

If the last column is not linearly independent, the system is no longer observable as  $\mathcal{O}_{k+1}$  does not have full column rank. This is the case if

$$\begin{bmatrix} d \\ b \end{bmatrix} \in \text{range} \left( \begin{bmatrix} C_{k+1} \\ A_{k+1} \end{bmatrix} \right). \quad (4.44)$$

If the vector is in the range then a state  $m$  with the property

$$\begin{bmatrix} d \\ b \end{bmatrix} = \begin{bmatrix} C_{k+1} \\ A_{k+1} \end{bmatrix} m \quad (4.45)$$

exists. This also means that no new state dimension is required. In this case the vector  $m$  is appended to the matrix  $B_k$ . The algorithm uses the SVD of  $[C_{k+1}^{\top}, A_{k+1}^{\top}]^{\top}$  to check the relations. This uses the intermediate vector

$$a = U^{\top} \begin{bmatrix} d \\ b \end{bmatrix} \quad (4.46)$$

If the vector  $[d^{\top}, b^{\top}]^{\top}$  is in the range, the norm of

$$\|a_{[r+1:]}\| = 0. \quad (4.47)$$

## 4 Methods

<p><b>Input:</b> System <math>\Sigma_{[A,B,C,D]}</math>, Index <math>k</math>, Tolerance <math>\text{tol}</math></p> <p><b>Output:</b> System <math>\Sigma_{[A,B,C,D]}</math> with changed input dimensions</p> <p><math>b \leftarrow B_{k+1}[:,1]</math></p> <p><math>d \leftarrow D_{k+1}[:,1]</math></p> <p><math>B_{k+1} \leftarrow B_{k+1}[:,2:]</math></p> <p><math>D_{k+1} \leftarrow D_{k+1}[:,2:]</math></p> <p><math>U, \sigma, V^\top \leftarrow \text{SVD}\left(\begin{bmatrix} C_{k+1} \\ A_{k+1} \end{bmatrix}\right)</math></p> <p><math>r \leftarrow \text{count}(\sigma &gt; \text{tol}_i)</math></p> <p><math>a \leftarrow U^\top \begin{bmatrix} d \\ b \end{bmatrix}</math></p> <p><b>if</b> <math>\ a_{[r+1:]}\  &gt; \text{tol}</math> <span style="float: right;">▷ not in range</span></p> <p style="padding-left: 20px;"><math>B_k \leftarrow \begin{bmatrix} B_k &amp; 0 \\ 0 &amp; 1 \end{bmatrix}</math></p> <p style="padding-left: 20px;"><math>A_k \leftarrow \begin{bmatrix} A_k \\ 0 \end{bmatrix}</math></p> <p style="padding-left: 20px;"><math>A_{k+1} \leftarrow [A_{k+1}, b]</math></p> <p style="padding-left: 20px;"><math>C_{k+1} \leftarrow [C_{k+1}, d]</math></p> <p><b>else</b></p> <p style="padding-left: 20px;"><math>m \leftarrow V_{[:,r]} \text{diag}(\sigma_{[r]})^{-1} a_{[r]}</math></p> <p style="padding-left: 20px;"><math>B_k \leftarrow [B_k \quad m]</math></p> <p><b>end if</b></p> <p><math>D_k \leftarrow [D_k \quad d_{\text{new}}]</math></p>
---

**Algorithm 4:** Move boundary between  $u_k$  and  $u_{k+1}$  to the right.

Here  $r$  is the number of nonzero singular values. The vector  $m$  is computed with

$$m = V_{[:,r]} \text{diag}(\sigma_{[r]})^{-1} a_{[r]}. \quad (4.48)$$

This is equivalent to calculating the vector  $m$  using the pseudoinverse. Additionally the vector  $d_{\text{new}}$  is attached to  $D_k$  to represent the new connection from  $u_{k+1}$  to  $y_\nabla$ . This connection is drawn with a dotted line in Figure 4.4.

The algorithms to move the horizontal boundaries up or down are described in Appendix B

### 4.4.2 Singular Values of Hankel Matrices when Moving Boundaries

In this section the ideas from Section 4.1 are used to calculate the singular values for a moved system. The main idea is that this algorithm iterates over the different stages and moves them if needed. This algorithm not only calculates the moved system but also returns the singular values. Algorithm 3 and Algorithm 4 are only

able to move the boundaries by a single dimension. The algorithm derived here can move the boundaries by arbitrary distances, as long as the other boundaries are not crossed.

I will describe the algorithm to move the bounds left or right in this subsection. The algorithms for moving up or down will be omitted for brevity. A reference implementation of the algorithm can be found in the supplementary code. The algorithm starts with an output normal system and transforms it into an ordered input normal system, by iterating over the stages from  $k = 1$  up to  $K$ .

**Move Left** When moving the boundary between  $u_k$  and  $u_{k+1}$  to the left the distance  $l$ , the last  $l$  columns of  $\mathcal{R}_{k+1}$  are removed. Based on this we have the new Hankel matrix

$$\tilde{H}_{k+1} = \tilde{\mathcal{O}}_{k+1} \tilde{\mathcal{R}}_{k+1} = \mathcal{O}_{k+1} \mathcal{R}_{k+1}[:, \text{end}-l]. \quad (4.49)$$

Using the decomposition from Equation 4.8 this results in

$$\tilde{H}_{k+1} = \mathcal{O}_{k+1} [A_k \quad B_{k[:, \text{end}-l]}] \begin{bmatrix} \mathcal{R}_k & 0 \\ 0 & I \end{bmatrix}. \quad (4.50)$$

If  $\mathcal{O}_{k+1}^\top \mathcal{O}_{k+1} = I$  and  $\mathcal{R}_k \mathcal{R}_k^\top = I$  then the singular values of the SVD of  $[A_k \quad B_{k[:, \text{end}-l]}]$  are the singular values of  $H_k$ . This is the case as the initial system is output normal and the previous stages were already transformed to input normal form. Based on the SVD, the stage can be transformed such that  $\mathcal{R}_{k+1} \mathcal{R}_{k+1}^\top = I$ . This also gives an ordered realization. The pseudocode can be found in Algorithm 5.

**Input:** System  $\Sigma_{[A,B,C,D]}$  with  $\mathcal{O}_{k+1}^\top \mathcal{O}_{k+1} = I$  and  $\mathcal{R}_k \mathcal{R}_k^\top = I$ ,  
 Index  $k$ , Distance to move  $l$ , Tolerance  $\epsilon$

**Output:** System  $\Sigma_{[A,B,C,D]}$  with changed input dimensions and  $\mathcal{R}_{k+1} \mathcal{R}_{k+1}^\top = I$ ,  
 Singular values  $\sigma$  of  $\tilde{H}_{k+1}$

```

b ← Bk[:,end-l+1:]
Dk ← Dk[:,end-l]
Bk ← Bk[:,end-l]
Dk+1 ← [Ck+1b, Dk+1]
Bk+1 ← [Ak+1b, Bk+1]
U, σ, V⊤ ← reducedSVD([Ak, Bk], ε = tol)
[Ak, Bk] ← V⊤
Ak+1 = Ak+1U diag(σ)
Ck+1 = Ck+1U diag(σ)
    
```

**Algorithm 5:** Move boundary between  $u_{k-1}$  and  $u_k$  to the left by  $l$  inputs.

#### 4 Methods

**Move Right** When the bound is moved right by the distance  $l$ , this results in  $l$  additional columns in  $\mathcal{R}_{k+1}$ . Additional states are also required in most cases. Using the decomposition from Equation 4.8 and the structure from Equation 4.43 the new Hankel matrix can be expressed as

$$\tilde{H}_{k+1} = \begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+2} \end{bmatrix} \begin{bmatrix} C_{k+1} & d \\ A_{k+1} & b \end{bmatrix} \begin{bmatrix} A_k \\ 0 \end{bmatrix} \begin{bmatrix} B_k & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathcal{R}_k & 0 \\ 0 & I \end{bmatrix} \quad (4.51)$$

If  $A_k$  and  $B_k$  are transformed to input normal form analogous to Algorithm 1 this can be reduced to

$$\tilde{H}_{k+1} = \begin{bmatrix} I & 0 \\ 0 & \mathcal{O}_{k+2} \end{bmatrix} \begin{bmatrix} C_{k+1} & d \\ A_{k+1} & b \end{bmatrix} \begin{bmatrix} \mathcal{R}_{k+1} & 0 \\ 0 & I \end{bmatrix}. \quad (4.52)$$

The reachability matrix  $\mathcal{R}_{k+1}$  fulfills  $\mathcal{R}_{k+1}\mathcal{R}_{k+1}^\top = I$ . Therefore the singular values of the SVD of  $\begin{bmatrix} C_{k+1} & d \\ A_{k+1} & b \end{bmatrix}$  are the singular values of the Hankel matrix. Using the SVD the system can also be reduced to a minimal system. To preserve the input normality  $A_k$  and  $B_k$  are multiplied with  $V^\top$ . The combined matrices  $[C_k^\top \ A_k^\top]^\top$  are set to  $U\Sigma$ . This results in the Algorithm 6.

**Input:** System  $\Sigma_{[A,B,C,D]}$  with  $\mathcal{O}_{k+1}^\top \mathcal{O}_{k+1} = I$  and  $\mathcal{R}_k \mathcal{R}_k^\top = I$ ,  
Index  $k$ , Distance to move  $l$ , Tolerance  $\text{tol}$

**Output:** System  $\Sigma_{[A,B,C,D]}$  with changed input dimensions and  $\mathcal{R}_{k+1} \mathcal{R}_{k+1}^\top = I$ ,  
Singular values  $\sigma$  of  $\tilde{H}_{k+1}$

$\Sigma_{[A,B,C,D]} \leftarrow \text{input normal}_{k+1}(\Sigma_{[A,B,C,D]}) \quad \triangleright \text{Transform state } x_{k+1}$

$b \leftarrow B_{k+1}[:, :l]$

$d \leftarrow D_{k+1}[:, :l]$

$B_{k+1} \leftarrow B_{k+1}[:, l+1:]$

$D_{k+1} \leftarrow D_{k+1}[:, l+1:]$

$U, \sigma, V^\top \leftarrow \text{reducedSVD}\left(\begin{bmatrix} C_{k+1} & d \\ A_{k+1} & b \end{bmatrix}, \epsilon = \text{tol}\right)$

$B_k \leftarrow V^\top \begin{bmatrix} B_k & 0 \\ 0 & I \end{bmatrix}$

$A_k \leftarrow V^\top \begin{bmatrix} A_k \\ 0 \end{bmatrix}$

$\begin{bmatrix} C_{k+1} \\ B_{k+1} \end{bmatrix} \leftarrow U \text{diag}(\sigma)$

$D_k \leftarrow [D_k \ d_{\text{new}}]$

**Algorithm 6:** Move boundary between  $u_k$  and  $u_{k+1}$  to the right by  $l$  inputs.

Analogous algorithms can be derived for the anticausal part.

### 4.4.3 Optimizing Input and Output Dimensions

In order to obtain the optimal segmentation of a system, the optimization problem

$$\min_{\text{segmentation}} f(\Sigma_{[A,B,C,D]}), \quad (4.53)$$

needs to be solved. Here  $f(\Sigma_{[A,B,C,D]})$  is some objective function. The problem is discrete and, in general nonconvex, which makes it hard for to solve.

To get a solution of the optimization problem, an iterative algorithm is used. For this, both the input dimensions and the output dimensions have to be changed. This algorithm alternately changes the input and output dimensions.

The general strategy is to start with an input normal causal system in combination with an output normal anticausal system. First, we adapt the dimensions of the outputs. For every boundary, the algorithm computes a preliminary system in which the boundary is either moved down, not moved, or moved up. For these three options the objective function is calculated. Then the system with the lowest objective function is adopted. This is done for every boundary. By doing this using the algorithms described in the previous subsection, the systems are transformed into a causal input normal system and an anticausal output normal system. These algorithms also compute the singular values of the Hankel matrices. This makes it possible to use these in the objective function. After the output dimensions are adapted, we adapt the input dimensions. Again, the adaptation is done by computing three preliminary systems were the bounds are either moved to the left, not moved, or moved to the right. Then the option with the lowest objective function is used. Thereby, the system is transformed back to an input normal causal system in combination with an output normal anticausal system. This makes it possible to immediately continue with an adaption of the output dimensions.

The pseudocode for the adaptation of the segmentation is given in Algorithm 7.

To avoid that the algorithm gets stuck in spurious local minima and to improve the speed, the algorithm does not change the segmentation by a fixed search distance  $l$ , but uses different search distances for every iteration of the algorithm. If the matrices are non square it might also be advantageous to use a different search distances for the outputs and the input.

For most of the tests I used search distances of the form  $l_i = \lceil l_0 2^{-i} \rceil$ . This gives a structure that is similar to a tree. As the optimal segmentation of the input depends on the segmentation of the output, that is also subject to change, the sequence is constructed such that there is not a single way to reach a certain movement of a boundary.

If moving the boundary by the search distance would mean crossing the next boundary, the boundary is only moved to the position of the next boundary.

The objective function for the output from Algorithm 7  $\Sigma_{[\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}, \tilde{E}, \tilde{F}, \tilde{G}]}$  is lower than or equal to the objective function for the original system  $\Sigma_{[A, B, C, D, E, F, G]}$ . For

#### 4 Methods

<p><b>Input:</b> Mixed system <math>\Sigma_{[A,B,C,D,E,F,G]}</math> with input normal causal system and output normal anticausal system sequences <math>l^{\text{out}}</math> and <math>l^{\text{in}}</math></p> <p><b>Output:</b> Ordered mixed system <math>\Sigma_{[\tilde{A},\tilde{B},\tilde{C},\tilde{D},\tilde{E},\tilde{F},\tilde{G}]}</math> with optimized input and output dimensions</p> <p><b>for</b> <math>i \leftarrow 1</math> <b>to</b> Number of iterations <b>do</b></p> <p style="padding-left: 20px;"><b>for</b> <math>k \leftarrow K - 1</math> <b>downto</b> 1 <b>do</b></p> <p style="padding-left: 40px;">▷ Iterate over boundaries and transform causal system to output normal and anticausal to input normal</p> <p style="padding-left: 40px;"><math>\Sigma_d \leftarrow \text{move down}(\Sigma)</math> by <math>\min(l_i^{\text{out}}, p_{k+1})</math></p> <p style="padding-left: 40px;"><math>\Sigma_n \leftarrow \text{no move}(\Sigma)</math></p> <p style="padding-left: 40px;"><math>\Sigma_u \leftarrow \text{move up}(\Sigma)</math> by <math>\min(l_i^{\text{out}}, p_k)</math></p> <p style="padding-left: 40px;"><math>\Sigma \leftarrow \underset{\Sigma \in \{\Sigma_d, \Sigma_n, \Sigma_u\}}{\text{argmin}} f(\Sigma)</math></p> <p style="padding-left: 20px;"><b>end for</b></p> <p style="padding-left: 20px;"><b>for</b> <math>k \leftarrow 1</math> <b>to</b> <math>K - 1</math> <b>do</b></p> <p style="padding-left: 40px;">▷ Iterate over boundaries and transform causal system back to input normal and anticausal to output normal</p> <p style="padding-left: 40px;"><math>\Sigma_r \leftarrow \text{move right}(\Sigma)</math> by <math>\min(l_i^{\text{in}}, m_{k+1})</math></p> <p style="padding-left: 40px;"><math>\Sigma_n \leftarrow \text{no move}(\Sigma)</math></p> <p style="padding-left: 40px;"><math>\Sigma_l \leftarrow \text{move left}(\Sigma)</math> by <math>\min(l_i^{\text{in}}, m_k)</math></p> <p style="padding-left: 40px;"><math>\Sigma \leftarrow \underset{\Sigma \in \{\Sigma_r, \Sigma_n, \Sigma_l\}}{\text{argmin}} f(\Sigma)</math></p> <p style="padding-left: 20px;"><b>end for</b></p> <p style="padding-left: 20px;"><b>if</b> <math>\Sigma</math> did not change <b>then</b></p> <p style="padding-left: 40px;">Exit</p> <p style="padding-left: 20px;"><b>end if</b></p> <p><b>end for</b></p>
---

**Algorithm 7:** Optimization of segmentation.

every objective function that is invariant under state transformations. This means

$$f(\Sigma_{[\tilde{A},\tilde{B},\tilde{C},\tilde{D},\tilde{E},\tilde{F},\tilde{G}]}) \leq f(\Sigma_{[A,B,C,D,E,F,G]}). \quad (4.54)$$

*Proof.* The sequence of objective functions for the intermediate systems  $f(\Sigma^{(1)}), \dots, f(\Sigma^{(N)})$  produced by the algorithm is non-increasing. When the output dimensions are adapted this is due to

$$f(\Sigma^{(l+1)}) = \min(f(\Sigma_d^{(l)}), f(\Sigma^{(l)}), f(\Sigma_u^{(l)})) \leq f(\Sigma^{(l)}). \quad (4.55)$$

The analogous argument

$$f(\Sigma^{(l+1)}) = \min(f(\Sigma_r^{(l)}), f(\Sigma^{(l)}), f(\Sigma_l^{(l)})) \leq f(\Sigma^{(l)}) \quad (4.56)$$

holds when the input dimensions are adapted.  $\square$

**Objective function** To recover the segmentation, the goal is to reduce the rank of the Hankel matrices. The rank itself is not a suitable objective function, as the algorithm has to exactly meet the segmentation. Therefore we use the nuclear norm  $\|H\|_*$ . The nuclear norm is defined as the sum of the singular values

$$\|X\|_* = \sum_{i=1}^r \sigma_i. \quad (4.57)$$

Liu and Vandenberghe use the nuclear norm get low rank solutions in optimization problems [30]. Dividing it with the spectral norm  $\|X\|$  results in the lower bound of the rank  $\text{rank}(X) \geq \|X\|_* / \|X\|$  as described by Recht et al. in [37]. This results in the objective function

$$f_{\text{nuc}}(\Sigma) = \sum_{\text{all } H} \frac{\|H\|_*}{\|H\|}. \quad (4.58)$$

When we compute  $f_{\text{nuc}}$  for Hankel matrices taken out of weight matrices, we can see that  $f_{\text{nuc}}(H) \approx \sqrt{\text{size}(H)}$ . This means that the algorithm would minimize the size of the Hankel matrices. In this case, the boundaries tend to move outwards, as this minimizes the size of the Hankel matrices.

As a different objective function I used the number of multiplications required to calculate the matrix-vector product  $y = Tu$  in the state space. Based on the Equation 4.30, this gives the objective function

$$f_{\text{FLOP}} = \sum_{k=1}^K d_{k+1}d_k + d_{k-1}^*d_k^* + (d_{k+1} + d_{k-1}^*)m_k + (d_k + d_k^*)p_k + p_k m_k. \quad (4.59)$$

Here the number of states is determined by counting the singular values of the Hankel matrices that are bigger than a certain threshold value. The threshold value has to be known in advance. This objective function does not minimize the approximation error, but Equation 4.25 gives an upper bound of  $K\epsilon$ .

## 4.5 Permutations

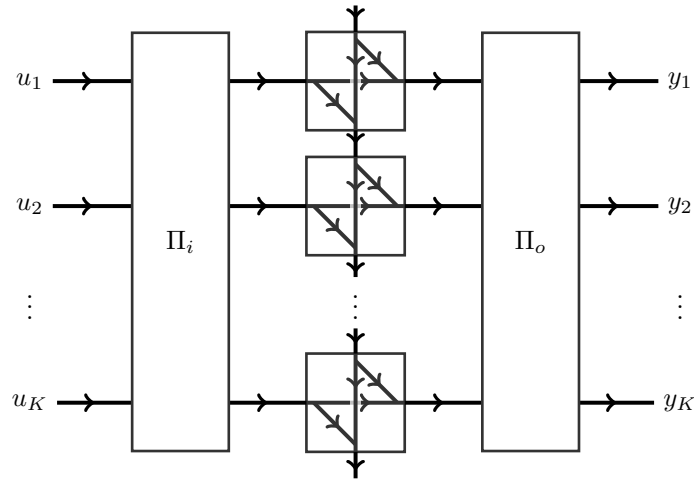
In this section the matrix is represented using a permutation of the inputs, a time varying system, and a permutation of the output.

This means that the matrix  $M$  is represented as the product

$$M = \Pi_o T \Pi_i. \quad (4.60)$$

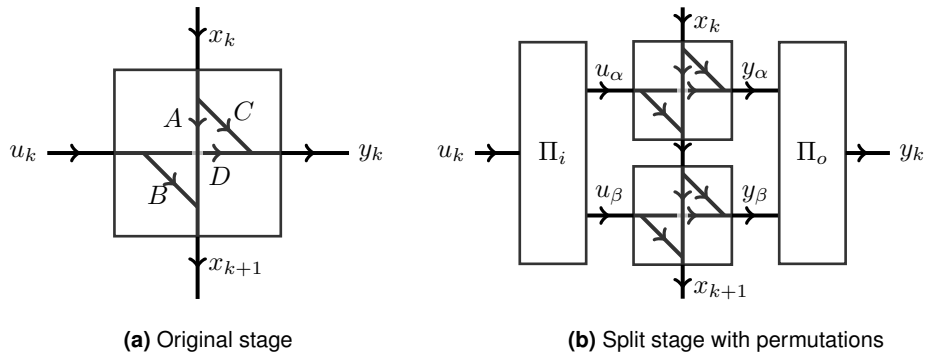
Here  $T$  is the transfer operator of the time varying system and  $\Pi_i$  and  $\Pi_o$  are permutation matrices. This is equivalent to permuting the columns and rows of the matrix  $M$  before representing it as a time varying system. This can lead to a reduction of

#### 4 Methods



**Figure 4.5:** Illustration of system with permuted inputs and outputs.

the Hankel rank and therefore the state dimensions as demonstrated by Diepold et al. in [12]. In this section I introduce an algorithm that makes it possible to permute the rows and columns of the represented matrix, such that state dimensions are reduced. The algorithm does not recover the whole permutation in one step but uses a strategy similar to divide and conquer by recursively splitting the problem up in smaller subproblems and recovering the permutation by combining the results from these individual subproblems. First an initial system consisting of only one stage is created. In this case  $T = D_1$ . Then the Algorithm splits the stages into two stages as illustrated in Figure 4.6. When this is done the inputs and outputs are permuted



**Figure 4.6:** Illustration of splitting a stage.

and segmented. These new stages are later split using the same strategy. The permutations can be combined, resulting in the permutation matrices  $\Pi_i$  and  $\Pi_o$ .

The order of the inputs inside the input vectors  $u_k$  and the order of the outputs



inside the output vectors  $y_k$  is not important. The states of the balanced realization depend on the SVD of  $H_k$ . The SVD of  $H_k$  is invariant under permutation of the rows with  $\Pi_R$  and permutation of the columns with  $\Pi_C$  as the new factorization

$$\Pi_R U \Sigma V^T \Pi_C = \tilde{U} \Sigma \tilde{V}^T \quad (4.61)$$

is again a SVD.

*Proof.* To be a valid SVD, we have to prove that  $\tilde{U}$  and  $\tilde{V}$  are orthogonal. This is shown by the forward multiplication

$$\tilde{U}^T \tilde{U} = U^T \Pi_C^T \Pi_C U = U^T U = I. \quad (4.62)$$

This uses the fact that  $\Pi^T \Pi = I$  for a permutation matrix  $\Pi$ . Analogously

$$\tilde{V}^T \tilde{V} = V^T \Pi_C \Pi_C^T V = V^T V = I \quad (4.63)$$

is true. As the matrix  $\Sigma$  stems from a SVD, the singular values are nonnegative and are ordered decreasingly.  $\square$

This means that all permutations that preserve the grouping of the inputs and outputs are equivalent. Therefore, the algorithm does not have to recover a unique ordering of the inputs and outputs but only has to group the inputs and outputs.

The algorithm to split a stage is described in Subsection 4.5.1. It uses the same strategy as described in [7]. Before the stage is split, the inputs and outputs of the stage must be grouped. The required algorithm to partition the inputs and outputs is presented in Subsection 4.5.2.

### 4.5.1 Split Stage

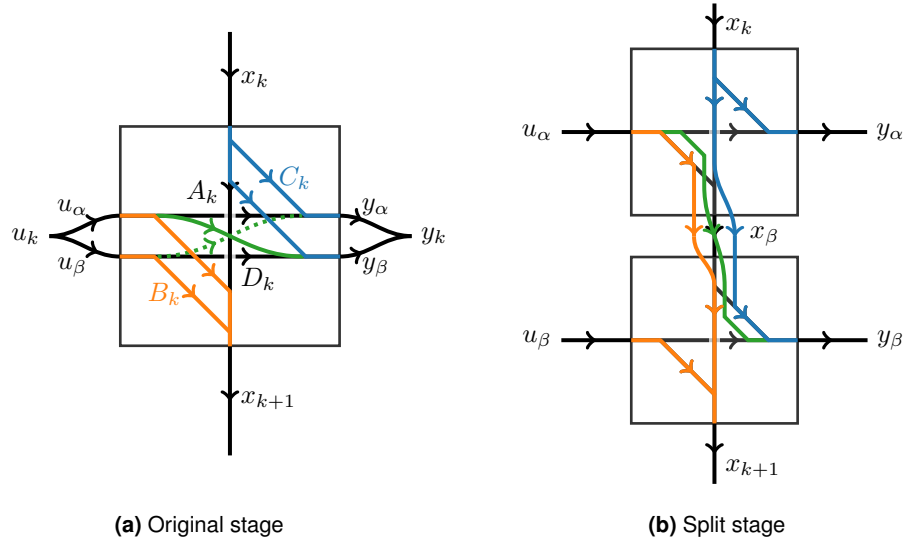
We want to split the stage  $k$  into the stages  $\alpha$  and  $\beta$ , as illustrated in Figure 4.7. Again, we have connections that are no longer possible in a causal system. These have to be included in the anticausal part. In Figure 4.7 the connections are represented using a dotted line. The resulting system should have the same input-output behavior. Therefore, the relations

$$A_k = A_\beta A_\alpha \quad B_k = \begin{bmatrix} B_{k[1]} & B_{k[2]} \end{bmatrix} = \begin{bmatrix} A_\beta B_\alpha & B_\beta \end{bmatrix} \quad (4.64a)$$

$$C_k = \begin{bmatrix} C_{k[1]} \\ C_{k[2]} \end{bmatrix} = \begin{bmatrix} C_\alpha \\ C_\beta A_\alpha \end{bmatrix} \quad D_k = \begin{bmatrix} D_{k[1,1]} & D_{k[1,2]} \\ D_{k[2,1]} & D_{k[2,2]} \end{bmatrix} = \begin{bmatrix} D_\alpha & 0 \\ C_\beta A_\alpha & D_\beta \end{bmatrix} \quad (4.64b)$$

have to be fulfilled. This immediately gives us most of the matrices for the new stages

#### 4 Methods



**Figure 4.7:** Illustration of splitting a stage.

Now we want to compute the remaining matrices. The new realization should be minimal. For this we consider the Hankel matrix associated with the new intermediate state  $x_\beta$ . The matrix can be factored into

$$H_\beta = \mathcal{O}_\beta \mathcal{R}_\beta = \begin{bmatrix} C_\beta \\ C_{k+1}A_\beta \\ C_{k+2}A_{k+1}A_\beta \\ \vdots \end{bmatrix} [\cdots \ A_\alpha A_{k-1}B_{k-2} \ A_\alpha B_{k-1} \ B_\alpha] \quad (4.65)$$

By factoring  $\mathcal{O}_\beta$  and  $\mathcal{R}_\beta$  using Equations 4.7 and 4.8 respectively we obtain

$$H_\beta = \begin{bmatrix} I \\ \mathcal{O}_{k+1} \end{bmatrix} \begin{bmatrix} C_\beta \\ A_\beta \end{bmatrix} [A_\alpha \ B_\alpha] \begin{bmatrix} \mathcal{R}_k \\ I \end{bmatrix} \quad (4.66)$$

Multiplying the inner of the three products results in the matrix

$$\begin{bmatrix} C_\beta \\ A_\beta \end{bmatrix} [A_\alpha \ B_\alpha] = \begin{bmatrix} C_\beta A_\alpha & C_\beta B_\alpha \\ A_\beta A_\alpha & A_\beta B_\alpha \end{bmatrix} \quad (4.67)$$

With the Equation 4.64 we can equate this matrix to

$$\begin{bmatrix} C_\beta A_\alpha & C_\beta B_\alpha \\ A_\beta A_\alpha & A_\beta B_\alpha \end{bmatrix} = \begin{bmatrix} C_{k[2]} & D_{k[2,1]} \\ A_k & B_{k[1]} \end{bmatrix}. \quad (4.68)$$

By computing the SVD we can now get the matrices

$$\begin{bmatrix} C_k & D_{k[1,0]} \\ A_k & B_k \end{bmatrix} = U \Sigma V^\top = \begin{bmatrix} C_\beta \\ A_\beta \end{bmatrix} [A_\alpha \ B_\alpha]. \quad (4.69)$$

Based on this we get the missing matrices for the new stages.

$$\begin{bmatrix} C_\beta \\ A_\beta \end{bmatrix} = U\Sigma^{1/2} \quad (4.70)$$

and

$$\begin{bmatrix} A_\alpha & B_\alpha \end{bmatrix} = \Sigma^{1/2}V^\top. \quad (4.71)$$

If  $O_{k+1}^\top O_{k+1} = 1$  and  $R_k R_k^\top = 1$  the singular values calculated by the SVD are also the singular values of  $H_\beta$ .

*Proof.* The SVD of the Hankel matrix is

$$H_\beta = \begin{bmatrix} I & \\ & O_{k+1} \end{bmatrix} \begin{bmatrix} C_\beta \\ A_\beta \end{bmatrix} \begin{bmatrix} A_\alpha & B_\alpha \end{bmatrix} \begin{bmatrix} \mathcal{R}_k & \\ & I \end{bmatrix} \quad (4.72)$$

$$= \underbrace{\begin{bmatrix} I & \\ & O_{k+1} \end{bmatrix}}_{\tilde{U}} U \Sigma V^\top \underbrace{\begin{bmatrix} \mathcal{R}_k & \\ & I \end{bmatrix}}_{\tilde{V}^\top}. \quad (4.73)$$

with the property that

$$\tilde{U}^\top \tilde{U} = \begin{bmatrix} I & \\ & O_{k+1} \end{bmatrix} \begin{bmatrix} I & \\ & O_{k+1}^\top \end{bmatrix} = \begin{bmatrix} I & \\ & I \end{bmatrix} = I \quad (4.74)$$

and

$$\tilde{V}^\top \tilde{V} = \begin{bmatrix} \mathcal{R}_k & \\ & I \end{bmatrix} \begin{bmatrix} \mathcal{R}_k^\top & \\ & I \end{bmatrix} = \begin{bmatrix} I & \\ & I \end{bmatrix} = I \quad (4.75)$$

□

To run efficiently, the algorithm requires a way to store the system in such a way that we can easily compute the different normal forms. For this we store the singular values separately from the reduced matrices  $\tilde{A}$ ,  $\tilde{B}$  and  $\tilde{C}$ .

If we have a balanced system  $\Sigma_{[A,B,C,D]}$  the matrices  $\tilde{A}$ ,  $\tilde{B}$ ,  $\tilde{C}$  and  $\tilde{D}$  are constructed such that

$$A_k = \Sigma_{k+1}^{1/2} \tilde{A}_k \Sigma_k^{1/2} \quad B_k = \Sigma_{k+1}^{1/2} \tilde{B}_k \quad (4.76a)$$

$$C_k = \tilde{C}_k \Sigma_k^{1/2} \quad D = \tilde{D}. \quad (4.76b)$$

As  $\Sigma_k$  is a diagonal matrix we only need to store the vector of the diagonal entries. Also, products with  $\Sigma_k$  can be efficiently computed by scaling the columns if the matrix is multiplied from the right or scaling the rows if the matrix is multiplied from the left. Analogously the products with the inverses or the square roots can be computed. Then the input normal system can be computed according to

$$\check{A}_k = \tilde{A}_k \Sigma_k \quad \check{B}_k = \tilde{B}_k \quad (4.77a)$$

$$\check{C}_k = \tilde{C}_k \Sigma_k \quad \check{D}_k = \tilde{D}. \quad (4.77b)$$

## 4 Methods

*Proof.* We have to prove that  $\check{\mathcal{R}}_k \check{\mathcal{R}}_k^\top = I$  for all  $k$ . As the original system is balanced we know that  $\mathcal{R}_k = \Sigma_k^{1/2} V_k^\top$ . The new reachability matrix is  $\check{\mathcal{R}} = \Sigma^{-1/2} \mathcal{R}_k = V_k^\top$ . As  $V_k$  is orthogonal,  $\check{\mathcal{R}}_k \check{\mathcal{R}}_k^\top = I$ .  $\square$

Analogously the output normal system is

$$\check{A}_k = \Sigma_{k+1} \tilde{A}_k \quad \check{B}_k = \Sigma_{k+1} \tilde{B}_k \quad (4.78a)$$

$$\check{C}_k = \tilde{C}_k \quad \check{D}_k = D. \quad (4.78b)$$

*Proof.* We have to prove that  $\check{\mathcal{O}}_{k+1}^\top \check{\mathcal{O}}_{k+1} = I$  for all  $k$ . As the original system is balanced we know that  $\mathcal{O}_{k+1} = U_{k+1} \Sigma_{k+1}^{1/2}$ . The new observability matrix is  $\check{\mathcal{O}}_{k+1} = \mathcal{O}_{k+1} \Sigma_{k+1}^{-1/2} = U_{k+1}$ . As  $U_k$  is orthogonal,  $\check{\mathcal{O}}_{k+1}^\top \check{\mathcal{O}}_{k+1} = I$ .  $\square$

This makes it possible to transform the stage  $k$  such that  $\mathcal{R}_k^\top \mathcal{R}_k = I$  and  $\mathcal{O}_{k+1}^\top \mathcal{O}_{k+1} = I$  are fulfilled at the same time. The matrices  $A_k, B_k, C_k$  and  $D$  are

$$A_k = \Sigma_{k+1} \tilde{A} \Sigma_k \quad B_k = \Sigma_{k+1} \tilde{B} \quad (4.79a)$$

$$C_k = \tilde{C} \Sigma_k \quad D_k = \tilde{D}. \quad (4.79b)$$

To make it possible to later split the resulting stages, the  $\Sigma$ s have to be removed from the matrices after splitting the stage. In the case of  $\tilde{C}_\alpha$  and  $\tilde{B}_\beta$  it also possible to use the reduced matrices  $\check{C}_k$  and  $\check{B}_k$  directly.

The Algorithm takes the stage in the form described in Equation 4.76. Then the SVD of the combined matrices are computed and the results are used to construct the new stages  $\alpha$  and  $\beta$ . These can then replace the stage  $k$ . These are again in the form described in Equation 4.76. For this the algorithm also returns the intermediate singular values  $\Sigma_\beta$ . The pseudocode is given in Algorithm 8.

### 4.5.2 Matrix Segmentation

Every time we split up a stage, we have to determine which inputs and outputs of the current stage correspond to the stage  $\alpha$  and which one correspond to the stage  $\beta$ . We want to do this in such a way that the rank of the Hankel matrices of the causal and the anticausal systems are reduced. This is again nontrivial as we have to solve a discrete optimization problem.

We need to group the rows and columns of the matrix

$$X = \begin{bmatrix} 0 & F_k & E_k \\ C_k & D_k & G_k \\ A_k & B_k & 0 \end{bmatrix}. \quad (4.80)$$

into two groups each.

<p><b>Input:</b> Stage with <math>\tilde{A}_k, \tilde{B}_k, \tilde{C}_k</math> and <math>\tilde{D}_k</math> as well as <math>\Sigma_k</math> and <math>\Sigma_{k+1}</math>  index <math>i_{in}</math> to split inputs  index <math>i_{out}</math> to split outputs  Tolerance tol</p> <p><b>Output:</b> Stages <math>\alpha</math> and <math>\beta</math> also intermediate <math>\Sigma_\beta</math>  <math>A_k \leftarrow \Sigma_{k+1} \tilde{A}_k \Sigma_k \triangleright</math> transform stage such that <math>\mathcal{R}_k^\top \mathcal{R}_k = I</math> and <math>\mathcal{O}_{k+1}^\top \mathcal{O}_{k+1} = I</math>  <math>B_k \leftarrow \Sigma_{k+1} \tilde{B}_k</math>  <math>C_k \leftarrow \tilde{C}_k \Sigma_k</math>  <math>U, s, V^\top \leftarrow \text{reducedSVD} \left( \begin{bmatrix} C_{[i_{out},:]} &amp; D_{[i_{out},:i_{in}]} \\ A &amp; B_{[:,i_{in}]} \end{bmatrix}, \epsilon = \text{tol} \right)</math>  <math>\tilde{A}_\alpha \leftarrow V_{[:,d_k]}^\top \Sigma_k^{-1}</math>  <math>\tilde{B}_\alpha \leftarrow V_{[:,d_k]}^\top</math>  <math>\tilde{C}_\alpha \leftarrow \tilde{C}_k[:,i_{out},:]</math>  <math>\tilde{D}_\alpha \leftarrow \tilde{D}_k[:,i_{out},i_{in}]</math>    <math>\tilde{A}_\beta \leftarrow \Sigma_{k+1}^{-1} U_{[\text{end}-d_{k+1}+1,:]}</math>  <math>\tilde{B}_\beta \leftarrow \tilde{B}_k[:,i_{in},:]</math>  <math>\tilde{C}_\beta \leftarrow U_{[:,\text{end}-d_{k+1},:]}</math>  <math>\tilde{D}_\beta \leftarrow \tilde{D}_k[i_{out},i_{in},:]</math>    <math>\Sigma_\beta \leftarrow \text{diag}(s)</math></p>
---

**Algorithm 8:** Split stage  $k$  into the stages  $\alpha$  and  $\beta$

When permuting the rows and columns of the  $B_k, C_k, F_k, G_k$  and  $D_k$  we obtain the matrices  $\tilde{B}_k, \tilde{C}_k, \tilde{B}_k^*, \tilde{C}_k^*$  and  $\tilde{D}_k$ . Then we can segment the resulting matrix according to

$$\begin{bmatrix} x_{k-1}^* \\ y_k \\ x_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & \tilde{H}_k & E_k \\ \tilde{C}_k & \tilde{D}_k & \tilde{G}_k \\ A_k & \tilde{B}_k & 0 \end{bmatrix} \begin{bmatrix} x_k^* \\ -u_k \\ x_k \end{bmatrix} \quad (4.81)$$

The input  $u_k$  and the output  $y_k$  are also permuted accordingly. This partitions the input  $u_k$  in the two sets  $u_\alpha$  and  $u_\beta$ . Analogously the output  $y_k$  is partitioned in the two sets  $y_\alpha$  and  $y_\beta$ . I denote all partitions that partition  $u_k$  into two sets as

## 4 Methods

$\mathcal{P}(u_k)$  and  $\mathcal{P}(y_k)$  for  $y_k$  accordingly. Here  $\tilde{H}_\beta$  and  $\tilde{H}_\alpha^*$  have the same singular values as the Hankel matrices  $H_\beta$  and  $H_\alpha^*$ . These singular values determine the state dimensions  $d_\beta$  and  $d_\alpha^*$ . To efficiently represent a matrix, both state dimensions should be minimal. Therefore we optimize the singular values of  $\tilde{H}_\beta$  and  $\tilde{H}_\alpha^*$  in some sense. The properties of the remaining parts of the matrix  $X$  are not relevant, as they do not influence the state dimensions.

Ideally the algorithm directly optimizes the singular values. For this we need a way to compute or approximate the singular values for different configurations. There are some approaches to decrease the cost for low rank updates of SVD [6]. But these still require a diagonalization step and are therefore too expensive to calculate the singular values multiple times per iteration. Also, techniques for low rank optimizations described in [30] and [37] did not prove helpful for this discrete optimization problem.

Therefore, I use two strategies to reduce the state dimensions. The first is reducing the rank of  $\tilde{H}_\beta$  and  $\tilde{H}_\alpha^*$  by minimizing the angle between the columns and rows respectively. This can be used if the resulting matrix is low rank. In the case of weight matrices all sub matrices usually have full rank. Therefore I use the second strategy that reduces the squared sum of the singular values. This is equivalent to minimizing the Frobenius norms of  $\tilde{H}_\beta$  and  $\tilde{H}_\alpha^*$ . This does not promote low rank solutions like the nuclear norm but is far easier to compute.

**Rank reduction** Reducing the rank of the Hankel matrices is equivalent to reducing the dimensions of the range of  $H$ . We could also reduce the range of  $H^\top$ . For a matrix  $M \in \mathbb{R}^{m \times n}$  these are all equivalent as

$$\text{rank}(m) = \dim(\text{range}(M)) = \dim(\text{range}(M^\top)) \quad (4.82)$$

This makes it possible to reduce the rank by partitioning the input  $u_k$  such that

$$\dim \text{range}(\tilde{H}_\beta) + \dim \text{range}(\tilde{H}_\alpha^*) \quad (4.83)$$

is reduced. At the same time we have to partition the outputs  $y_k$  such that

$$\dim \text{range}(\tilde{H}_\beta^\top) + \dim \text{range}(\tilde{H}_\alpha^{*\top}) \quad (4.84)$$

is reduced. Both problems influence each other. As we want to reduce the dimension of the range we want to make sure that the angle between the columns of the Hankel matrices are small. For this the function

$$\chi(u, v) = 1 - \sin(\angle(u, v)) = 1 - \frac{|u^\top v|}{\|u\|_2 \|v\|_2}. \quad (4.85)$$

is used. If  $u$  and  $v$  are colinear  $\chi(u, v) = 0$ .

*Proof.* If  $u$  and  $v$  are colinear then  $u = \alpha v$ . Therefore

$$\chi(\alpha v, v) = 1 - \frac{|\alpha| |v^\top v|}{|\alpha| \|v\|_2 \|v\|_2} = 1 - \frac{|\alpha| \|v\|_2^2}{|\alpha| \|v\|_2^2} = 0 \quad (4.86)$$

□

if  $u$  and  $v$  are orthogonal  $\chi(u, v) = 1$ .

*Proof.* If  $u$  and  $v$  are orthogonal then  $u^\top v = 0$ . Therefore

$$\chi(u, v) = 1 - \frac{0}{\|u\|_2 \|v\|_2} = 1 \quad (4.87)$$

□

This gives the objective function

$$f_{col} = \sum_{u,v \text{ columns of } \tilde{H}_\beta} \chi(u, v) + \sum_{u,v \text{ columns of } \tilde{H}_\alpha^*} \chi(u, v). \quad (4.88)$$

An analogous derivation results in the objective function for the rows

$$f_{row} = \sum_{u,v \text{ rows of } \tilde{H}_\beta} \chi(u, v) + \sum_{u,v \text{ rows of } \tilde{H}_\alpha^*} \chi(u, v). \quad (4.89)$$

This results in the two coupled optimization problems

$$\arg \min_{(u_\alpha, u_\beta) \in \mathcal{P}(u_k)} f_{col} + \gamma \left( \frac{\text{len}(u_\alpha)}{\text{len}(u_k)} - 0.5 \right)^2 \quad (4.90)$$

and

$$\arg \min_{(y_\alpha, y_\beta) \in \mathcal{P}(y_k)} f_{row} + \gamma \left( \frac{\text{len}(y_\alpha)}{\text{len}(y_k)} - 0.5 \right)^2. \quad (4.91)$$

These include regularization terms depending on the length of the inputs and outputs to make sure that the groups are of similar size. To solve this optimization problem, I use an iterative algorithm. At every iteration, the algorithm computes for every input in  $u_\alpha$  how much the objective function  $f_{col}$  would change if we would move the input to  $u_\beta$ . The input with the smallest value is moved to  $u_\beta$ , if it is negative. Analogously the same is done for the inputs in  $u_\beta$ . If there is neither a negative value for  $u_\alpha$  nor a negative value for  $u_\beta$ , the inputs with the smallest values are exchanged. Simultaneously the same is done with the outputs with the objective function  $f_{row}$ .

#### 4 Methods

The initial partition of  $u_k$  is obtained using spectral clustering. For this the columns of  $X$  are represented by nodes connected by weighted edges. The weight between the columns  $c_i$  and  $c_j$  is

$$w_{i,j} = \begin{cases} |c_i^\top c_j| & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \quad (4.92)$$

The spectral clustering is done with a normalized Laplacian as described in [19]. The resulting segmentation also includes additional columns that do not correspond to inputs, and are therefore already fixed. If more than half of these columns have been grouped incorrectly, the segmentation is flipped.

Analogously the initial partition of  $y_k$  is computed.

**Frobenius norm reduction** Next, we consider the case where the Hankel matrices have close to full rank. Here balanced truncation is used to approximate the system. To reduce the number of states of the approximated system, the number of singular values greater than some threshold have to be reduced. To achieve this, I reduce the sum of the squared singular values. This is equivalent to the squared Frobenius norm as  $\|H\|_F^2 = \sum \sigma_i^2$ . Because  $\|H\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n h_{ij}^2$  this results in the problem of partitioning the inputs and outputs such that the sum over the squared entries of  $\tilde{H}_\beta$  and  $\tilde{H}_\alpha^*$  is reduced. As  $\|H\|_F^2 \approx \text{size}(H)$  for the weight matrices tested, this would be equivalent to minimizing the size of  $H_\beta$  and  $H_\alpha^*$ . To avoid the trivial solution, where  $u_\alpha$  and  $y_\alpha$  or  $u_\beta$  and  $y_\beta$  are empty a regularization term  $\gamma$  is added.

This results in the discrete optimization problems

$$(u_\alpha, u_\beta), (y_\alpha, y_\beta) = \underset{\substack{(u_\alpha, u_\beta) \in \mathcal{P}(u_k) \\ (y_\alpha, y_\beta) \in \mathcal{P}(y_k)}}}{\arg \min} \|\tilde{H}_\beta\|_F^2 + \|\tilde{H}_\alpha^*\|_F^2 + \gamma \quad (4.93)$$

Here I use the regularization term

$$\gamma = \gamma_r \frac{\text{len}(u_k) + \text{len}(y_k)}{2} \left[ \left( \frac{\text{len}(u_\alpha)}{\text{len}(u_k)} - 0.5 \right)^2 + \left( \frac{\text{len}(y_\alpha)}{\text{len}(y_k)} - 0.5 \right)^2 \right] \quad (4.94)$$

To solve this optimization problem, I use the same iterative algorithm as used for the rank reduction. At every iteration the algorithm computes for every input in  $u_\alpha$  how much the objective function would change if we would move the input to  $u_\beta$ . The input with the smallest value is moved to  $u_\beta$ , if it is negative. Analogously the same is done for the inputs in  $u_\beta$ . If no value is negative for both sets, the inputs with the smallest values are flipped. The same is done with the outputs. The algorithm terminates as soon as the algorithm detects an loop.

For larger matrices multiple inputs and outputs can be flipped at every iteration to speed up the algorithm. To additionally speed up the algorithm a matrix  $W$  is



computed, containing the matrix  $D_k^{\circ 2}$  with the squared entries of  $D_k$ . Here the notation  $\square^{\circ 2}$  denotes the Hadamard power. Using this approach, the squares of the entries only have to be computed once. Additionally, the size of the matrix  $W$  can be reduced by replacing the matrices  $A_k$  and  $E_k$  with their Frobenius norms  $a_k$  and  $e_k$ . The matrices  $B_k$  and  $F_k^*$  are replaced by their column-wise vector-2-norm  $b_k^\top$  and  $f_k^\top$ . The matrices  $C_k$  and  $G_k$  are replaced by their row wise vector-2-norm  $c_k$  and  $g_k$ . This results in the matrix

$$W = \begin{bmatrix} 0 & f_k^\top & e_k \\ c_k & D_k^{\circ 2} & g_k \\ a_k & b_k^\top & 0 \end{bmatrix}. \quad (4.95)$$

The size of the matrix  $W$  is independent of the state dimensions. This technique makes it possible to apply the strategy to large matrices.



## 5 Experiments

In this section, I test how the algorithms presented in Chapter 4 behave for different matrices. To demonstrate the behavior for structured matrices I use random Hankel matrices. After this I test the algorithms on weight matrices from the *Mobilenet V2* model [41] and the *AlexNet* model [26] as available through the *torchvision* package [32]. When approximating the weight matrices, the goal is to reduce the number of floating-point operations. The number of floating-point operations is described in Section 4.2. At the same time the approximation error should stay small. To measure the approximation error, I use the spectral norm of the difference between the matrix  $M$  and the approximated matrix  $\tilde{T}$ .

The code to reproduce the experiments is available on Github<sup>1</sup>. The code uses the *tvscilib* library [24] to handle the time varying systems.

### 5.1 Segmentation Adaption

In this section the approach described in Section 4.4 is tested. Here an initial system with constant input and output dimensions is created before adapting the segmentation using Algorithm 7.

#### 5.1.1 Illustrative Example

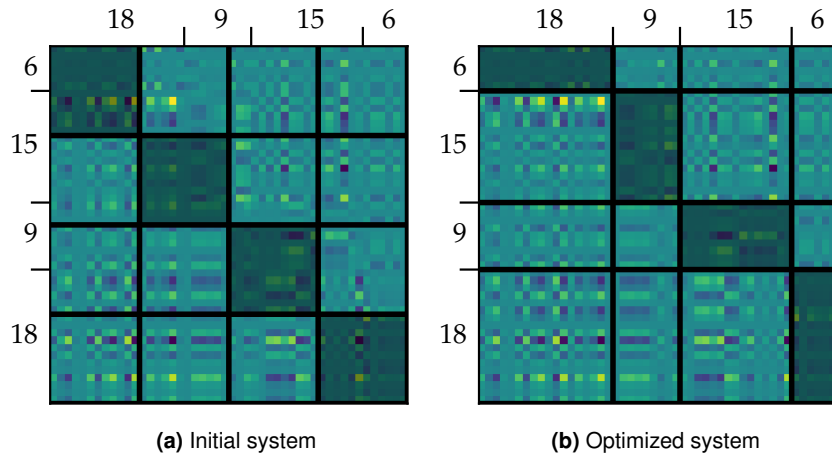
First, I demonstrate the algorithm for a sequentially semiseparable matrix  $T$ . All Hankel matrices of  $T$  are rank one. The matrix is generated from random orthogonal vectors such that the range of  $C$ ,  $D$  and  $G$  are orthogonal to each other. Also, the range of  $B^\top$ ,  $D^\top$  and  $G^\top$  are orthogonal. The properties of the matrix  $T$  are explained in more detail in Appendix C.

I represent the matrix with a time varying system without using the knowledge about the input and output dimensions. For this, I create a system with constant input and output dimensions. This system is illustrated in Figure 5.1a. Then the segmentation is adapted using Algorithm 7. As an objective function, I used the normalized nuclear norm as described in Equation 4.58. This results in the system illustrated in Figure 5.1b. We can see that the algorithm can recover the original segmentation.

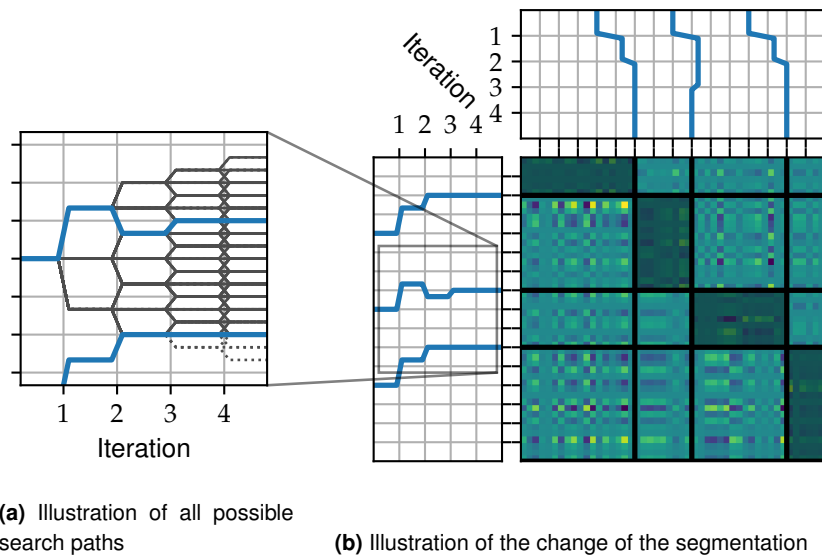
---

<sup>1</sup>The code to reproduce the plots as well as supplementary code can be found under [https://github.com/snuesslein/Matrixapprox\\_reproducibility](https://github.com/snuesslein/Matrixapprox_reproducibility)

## 5 Experiments



**Figure 5.1:** Illustration of the initial system and the optimized system. The lines mark the segmentation of the systems. The  $D_k$ -matrices are shaded in gray. The ticks show the segmentation of the matrix  $T$ . We can see that the segmentation of the optimized systems coincides with the segmentation of  $T$ .



**Figure 5.2:** Illustration of the change of the segmentation. The blue lines mark how the boundaries change as the algorithm adapts the segmentation.

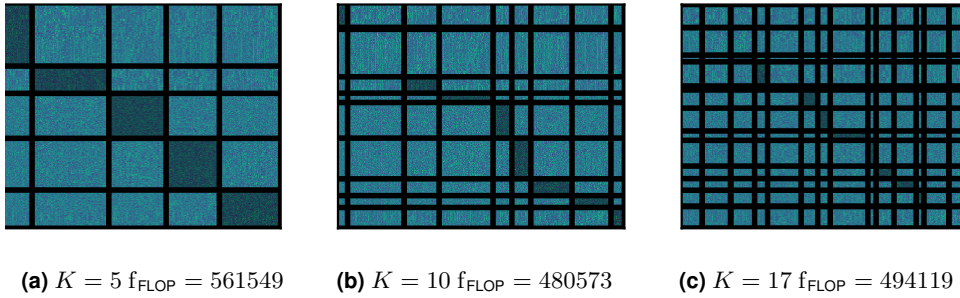
When the algorithm adapts the segmentation, it tests different segmentations and uses the one with the lowest objective function. This is done in every iteration. The search distance  $l_i$  determines which movements of boundaries are tested at every

iteration. In the first iterations a larger  $l_i$  is used. For the later iterations the  $l_i$  are smaller. I used  $l = [4, 2, 1, 1]$  as search distances for this example. The plot in Figure 5.2a shows all the positions that are possible at every iteration. We can see that in the first iterations the possible changes are larger and decrease for further iterations. We can also observe that it is possible to reach the same segmentation using different paths. The algorithm also makes sure, that the boundaries do not cross. Therefore, some paths are not possible as these would require crossing another boundary. In Figure 5.2a these are drawn with dotted lines. Figure 5.2b shows how the segmentation is changing between the iterations.

### 5.1.2 Weight Matrix Approximation

Now the algorithm is tested for weight matrices from two pretrained neural networks.

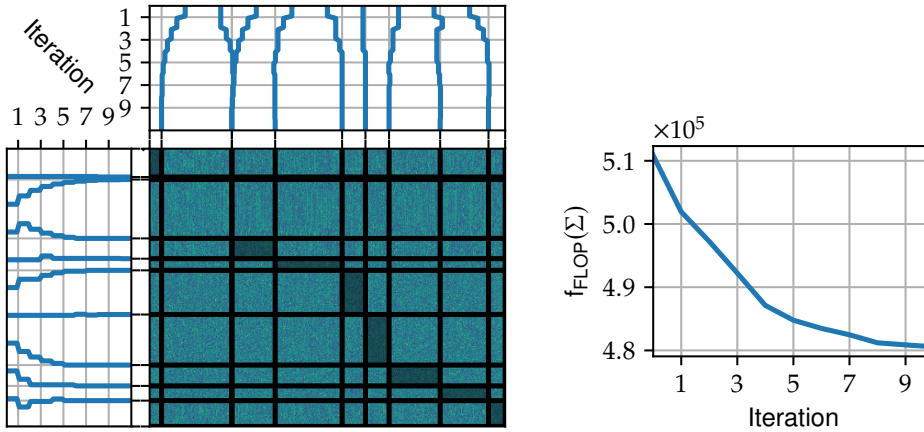
**Mobilenet V2** First a weight matrix  $M$  from the *Mobilenet V2* model is used. The size of the matrix is  $1000 \times 1280$ . I set the number of stages to  $K = 10$  based on the approximations done in Section 4.3. In Section 4.3 the input and output dimensions are presumed to be constant. Because this is not the case for the tested matrix, I also tested other  $K$ 's. As the objective function I use the number of multiplications as defined in Equation 4.59. I chose  $\epsilon = \frac{1}{4} \|M\|_H$  as the threshold value for which the segmentation is optimized. The value of  $\|M\|_H$  is calculated using the initial segmentation. I use the search sequence  $l = [30, 20, 14, 9, 6, 4, 3, 2, 2, 1]$ . This search sequence also starts with larger search distances that decrease over time.



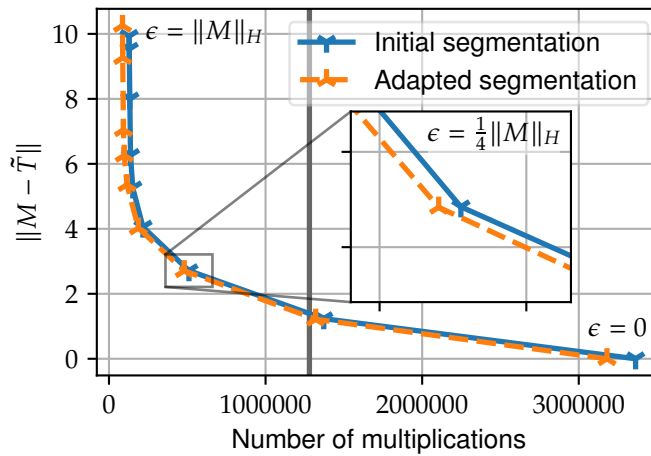
**Figure 5.3:** Matrix segmentation for different numbers of stages. The areas shaded in gray represent the  $D_k$ -matrices. For higher numbers of stages the portion represented by the  $D_k$ -Matrices gets lower.

The results for  $K = 5$ ,  $K = 10$  and  $K = 17$  are illustrated in Figure 5.3. The different numbers of stages result in different numbers of multiplications. For  $K = 10$  I get the lowest number of multiplications.

## 5 Experiments



**Figure 5.4:** Illustration of the change of the segmentation for the weight matrix from *MobileNet V2* model.



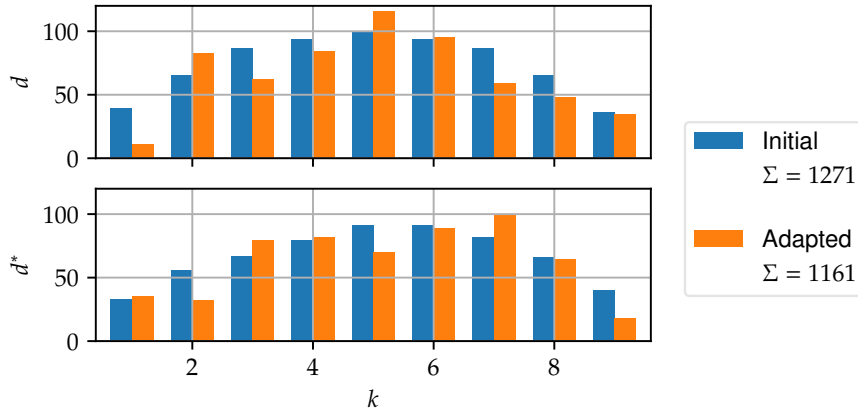
**Figure 5.5:** Plot of the approximation error with respect to the number of multiplications for different  $\epsilon$ . The black line indicates the number of multiplications for the regular matrix-vector product.

The change of the segmentation for  $K = 10$  is illustrated in Figure 5.4. We can see that the number of operations decreases, as the algorithm runs, as plotted on the right of Figure 5.4. Note that some input and output dimensions vanish. In Figure 5.4 this manifests itself as two bounds converging to the same point. This also leads to the effect that only a small part of the matrix is represented with the  $D_k$ -matrices.

The resulting system requires more multiplications to compute  $y = Tu$  as re-

quired for a regular matrix-vector multiplication. Therefore, the system complexity is reduced using balanced truncation as described in Section 4.1. This is done for evenly spaced approximation parameters  $\epsilon$  in the range from 0 to  $\|M\|_H$ . For every  $\epsilon$  the number of multiplications and the approximation error  $\|M - \tilde{T}\|$  is computed and plotted in Figure 5.5. The results for the original and the adapted segmentation are shown. For the case  $\epsilon = \frac{1}{4}\|M\|_H$  we can see that the number of multiplications decreased by 6% as shown in Figure 5.4. The approximation error  $\|M - \tilde{T}\|$  did not change due to the adaption of the segmentation.

The state dimensions of the initial and adapted systems are plotted in Figure 5.6. One can see that while some state dimensions decreased others did not decrease or even increased significantly. In total, the number of states decreased.

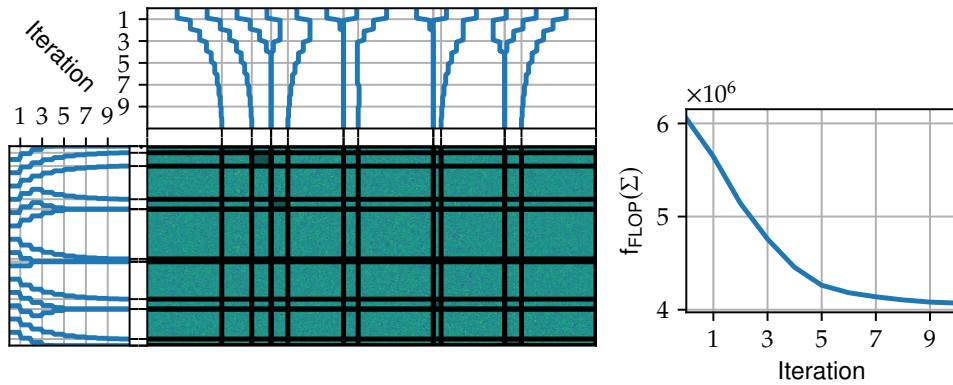


**Figure 5.6:** Plot of the causal state dimensions  $d_k$  and the anticausal state dimensions  $d_k^*$  before and after the adaptation of the segmentation for  $\epsilon = \frac{1}{4}\|M\|_H$

**AlexNet** Now a weight matrix  $M$  from the *AlexNet* model with the size  $4096 \times 9216$  is represented. For this matrix I use  $K = 15$ , based on the approximation described in Section 4.3. As the matrix is wide, I use different search distances for the inputs and outputs. For the output I use the search distances  $l = [120, 86, 62, 44, 32, 23, 16, 12, 9, 6]$ , and for the input I use  $l = [270, 193, 138, 99, 71, 51, 36, 26, 19, 14]$ . Here the threshold value is again  $\epsilon = \frac{1}{4}\|M\|_H$ , based on the initial segmentation. To speed up the computation, the system is approximated with balanced truncation before the segmentation is adapted. This is done using the threshold  $\epsilon_{\text{apr}} = \frac{1}{2}\epsilon$ . It is important to choose a smaller  $\epsilon$  for the approximation than for the adaptation.

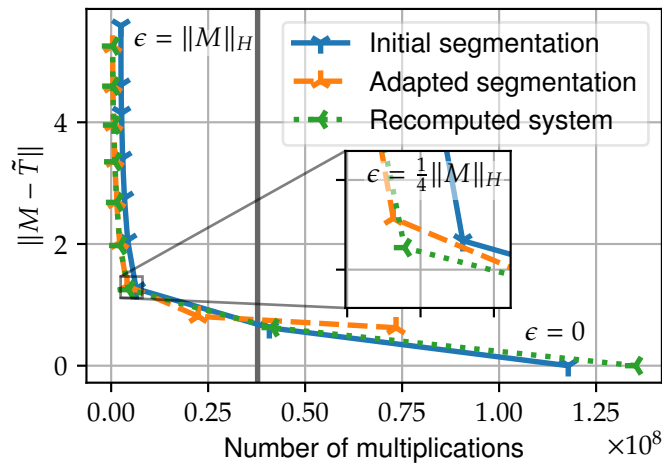
Subsequently I adapt the segmentation using Algorithm 7. The systems are again approximated using balanced truncation. After moving the bounds the number of multiplications for  $\epsilon = \frac{1}{4}\|M\|_H$  is reduced by 37%. As the adaptation of the segmen-

## 5 Experiments



**Figure 5.7:** Illustration of the change of the segmentation for the weight matrix from *AlexNet* model.

tation was not performed on the full system, but only on an approximated system, adaptation errors can be amplified. We can see in Figure 5.7 that the approxima-



**Figure 5.8:** Plot of the approximation error with respect to the number of multiplications for different  $\epsilon$ . The black line indicates the number of multiplications for the regular matrix-vector product.

tion error actually increased after applying the Algorithm 7. We can also observe that the approximation error for the adapted system does not vanish for the balanced truncation with  $\epsilon = 0$  as the system is based on an already approximated system. Therefore, a new system is computed. This system has the same input and output dimensions as the adapted system but is directly calculated from the matrix  $M$ . Subsequently this system is approximated using balanced truncation. The re-



computed system with the new borders has the lowest approximation error. The number of multiplications increases slightly for the recomputed system compared to the adapted system. The number of multiplications is still 31% less compared with the initial system for  $\epsilon = \frac{1}{4} \|M\|_H$ . For smaller  $\epsilon$ s, the recomputed system behaves similar to the initial system.

## 5.2 Permutations

In this section the algorithm described in Section 4.5 is tested. Here an initial system with one stage is created. The stage is then split up into two stages. This process is repeated until the final number of stages is reached. When the stages are split, the inputs and outputs are partitioned such that an objective function is minimized. This makes it possible to recover a permuted sequentially semiseparable matrix.

### 5.2.1 Illustrative Example

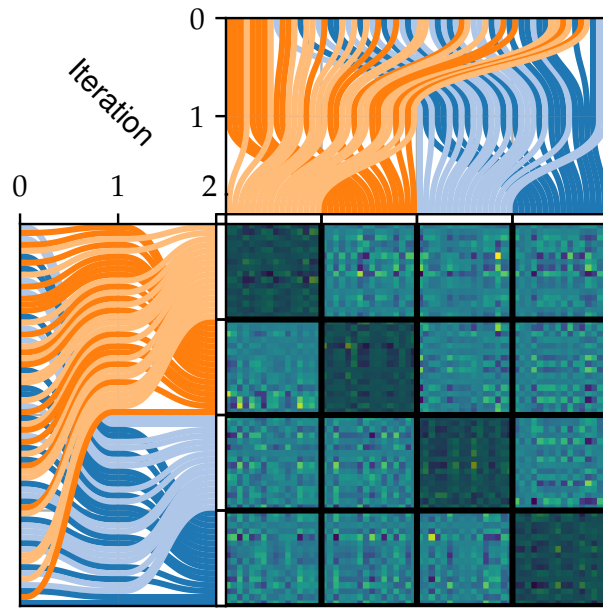
First the algorithm is tested on a random sequential semiseparable matrix  $T$ . The Hankel matrices have rank 2. The matrix is generated as described in Appendix C. The rows and columns of the matrix are then permuted with random permutations. Algorithm 8 is used to identify the system and the permutations. The result is shown in Figure 5.9. During the first iteration the algorithm splits the initial stage into two stages. We can see how the algorithm partitions the inputs and outputs into two groups and reorders them accordingly. After this, the stages are again split into four stages total. In the second iteration the inputs and outputs of each stage are grouped into two subgroups each and ordered accordingly. The algorithm is able to recover the permutation.

### 5.2.2 Weight Matrix Approximation

The algorithm is tested on weight matrices. These have the property that the Hankel matrices usually have close to full rank. As the algorithm to minimize the rank does usually increase the computational cost in this case, the Frobenius norm of the Hankel matrices is used as objective function.

**MobileNet V2** Next the algorithm is used to represent a weight matrix from the *MobileNet V2* model. Here the inputs and outputs are partitioned such that the Frobenius norm of the Hankel matrices is minimized. As the algorithm can only use numbers of stages that are powers of 2, I use  $K = 2^3 = 8$  as it is closest to  $K = 10$ . The regularization parameter is set to  $\gamma = 9 \times 10^5$ . This makes sure that the inputs and outputs are divided into two nearly equally sized parts. The resulting permuted matrix is shown in Figure 5.10.

## 5 Experiments



**Figure 5.9:** Illustration of the permuted sequentially matrix. The connections at the top and the left illustrate how the original segmentation is recovered by permuting the input and output. This is done in two iterations of the algorithm.

As a reference, a system without permutations is computed. The reference system and the permuted system are approximated using balanced truncation. In Figure 5.11 we can see that permuting the matrix has only a minor impact on the number of operations and the approximation error. For  $\epsilon = \frac{1}{4}\|M\|_H$  the reduction of the computational cost due to the permutation is  $< 1\%$ .

The state dimensions of the regular and permuted systems are plotted in Figure 5.12. One can see that while most state dimensions decreased others did increase. In total, the number of states only decreased slightly. This means that the algorithm is only partially successful in decreasing the number of states.

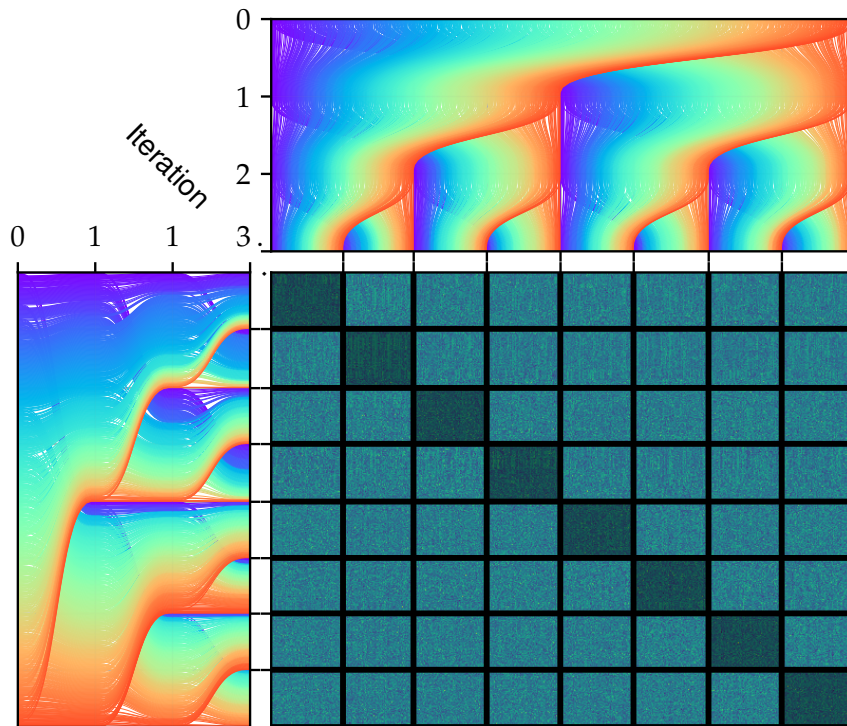


Figure 5.10: Illustration of permuted weight matrix from *Mobilenet V2* model.

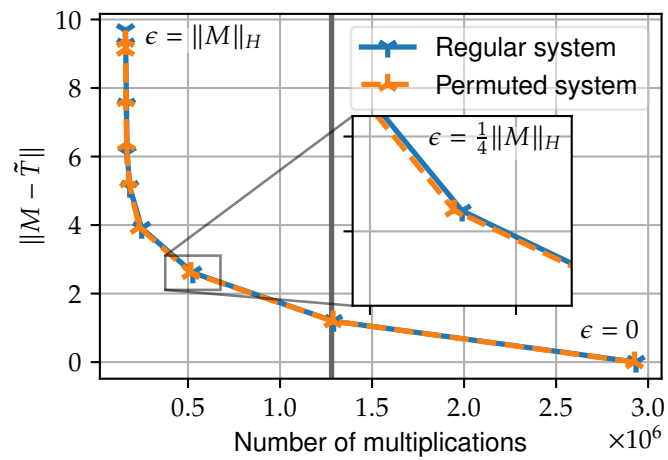
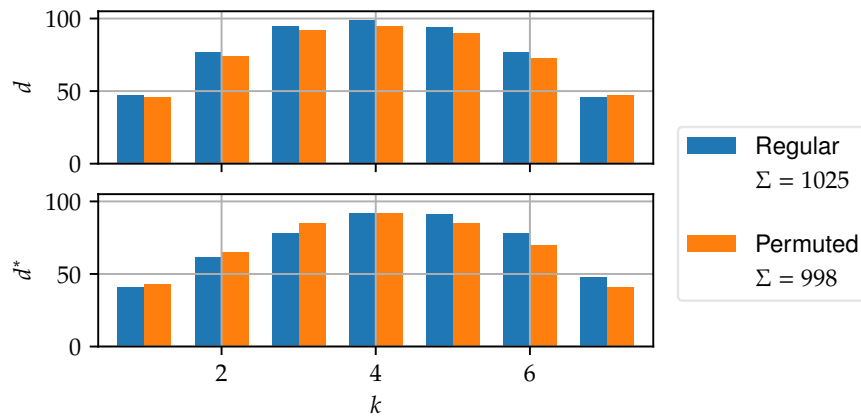


Figure 5.11: Plot of the approximation error with respect to the number of multiplications for different  $\epsilon$ . The black line indicates the number of multiplications for the regular matrix-vector product.

## 5 Experiments



**Figure 5.12:** Plot of the causal state dimensions  $d_k$  and the anticausal state dimensions  $d_k^*$  for the regular and permuted system for  $\epsilon = \frac{1}{4}\|M\|_H$

**Alexnet** The algorithm returns similar results for the weight matrix from the *AlexNet* model. For this the  $K = 2^4 = 16$  was used and  $\gamma = 6 \times 10^3$ . The number of flops is reduced by 3% if the system is approximated with  $\epsilon = \frac{1}{4}\|M\|_H$ .

## 6 Discussion

This thesis presented two algorithms which were able to recover the structure of simple test matrices based on random sequentially semiseparable matrices. When applied to the tested weight matrices, they were not able to identify a potential structure.

Regardless, optimizing the segmentation for the computational cost did lead to a noticeable reduction of the computational cost of the approximated system as seen in Section 5.1.1. The total number of state dimensions decreased, even if some state dimensions got larger. This reduction of the computational cost is possible without a significant worsening of the approximation error. As the algorithm can also run on a pre-approximated system, the segmentation of relatively large matrices like the *Alexnet* weight matrix can be computed on a standard computer. The algorithm requires that the approximation parameter for the balanced truncation is known in advance. There is no straightforward relation to determine an approximation parameter that leads to a predetermined computational cost. If the system is not pre-approximated, the adaptation also led to improvements if a different approximation parameter was used for the balanced truncation. The number of stages must also be determined in advance. The experiments demonstrate that the approximations derived in Section 4.3 can be used to choose a suitable number of stages. One notable observation is that the size of the  $D_k$ -matrices decrease. In some cases, the inputs or outputs of stages are zero-dimensional. It might be interesting to explore time varying systems that do not require  $D_k$ -matrices. Alternatively, the  $D_k$  could be represented using other matrix approximations like low rank approximations, which might further reduce the computational cost.

The second approach based on recursive splitting, tested in Section 5.2, did not return promising results. The algorithm can successfully reduce the Frobenius norm of the Hankel matrices. But this reduction of the sum of the squared singular values of the Hankel matrices did only lead to a very small reduction of the computational cost, as the number of states only decreased slightly.

The results suggest that the potential for decreasing the computational cost by reducing the number of states is limited. Focusing directly on the computational cost seems to be a better approach.

I was not able to identify sequentially semiseparable structures in the weight matrices using a simple adaptation of the segmentation. This is not surprising, as this would require that the Hankel matrices of the weight matrices are of low rank; a condition that did not show up in any experiment. It might be possible to find structures

## 6 Discussion

when allowing permutations as this has more degrees of freedom. Unfortunately, the splitting algorithm based on the minimization of the Frobenius norm was not able to do this. Here an optimization based on the nuclear norm, that promotes low rank solutions might be preferable, even if this would require some improvements to make the optimization problem computationally viable.

The fact that the Hankel matrices of the weight matrices do not have a low rank is not surprising considering other research. Research by Martin and Mahoney suggests that well trained weight matrices usually have full rank [33]. This might be explained with information theory as discussed by Papyan in [35]. The full rank property is no problem in general, as sequentially semiseparable matrices can have full rank, even if the Hankel matrices have a low rank<sup>1</sup>. I was not able to find statements if these results also hold for submatrices. Even if similar results hold true for submatrices, approximations using time varying systems might be still beneficial, if most singular values are sufficiently small. In this case the approximation error would be small, even if many nonzero singular values would have to be removed by applying balanced truncation.

---

<sup>1</sup>For a trivial example consider the identity matrix as a sequentially semiseparable matrix with full rank

## 7 Conclusion

The goal of this thesis was to develop and test algorithms that can be used to approximate matrices using time varying systems.

I developed multiple algorithms to transform time varying systems. These are also useful for other applications involving time varying systems. An important result are the techniques to obtain the singular values of the Hankel matrices without explicitly computing the Hankel matrices. The singular values are important as they are needed to approximate a system with balanced truncation. This allows an efficient approximation of arbitrary systems similar to the technique described by Chandrasekaran et al. [7]. If a system has to be reduced to a minimal system using floating-point arithmetic, then this algorithm makes it possible to do this in a well-defined fashion by truncating the small singular values. Also an error bound was derived for the approximation of matrices with balanced truncation. The error bound is similar to existing error bounds, but allows a tighter bound if a mixed system is approximated. Additionally, the error bound makes it possible to bound the error measured with the Frobenius norm. The Algorithm to split stages described by Chandrasekaran et al. in [7] was extended such that it can compute the corresponding singular values of the Hankel matrices. This makes it possible to use the algorithm to identify and later refine systems and obtain the singular values on the fly. Furthermore, algorithms to change the input and output dimensions are devised. Such algorithms are required if systems with incompatible dimensions have to be added or multiplied. The work on this thesis also led to improvements in the *tvscilib* library [24].

To devise the number of stages, the computational cost for computing the matrix-vector product with a time varying system was estimated. Using this estimation, it is possible to choose the number of stages for approximations. This estimation might also serve as a baseline for the determination of structural parameters that must be chosen for the reduction of computational cost. These are not only required if an existing matrix is approximated but can also be used as a baseline if a neural network based on time varying systems is trained from scratch.

These results were combined with optimization problems to approximate weight matrices from neural networks. The algorithm to adapt the segmentation of the matrix is able to reduce the computational cost of the system, even if no clear structure is identified. The second approach based on splitting the stages did only yield minor improvements. Some ideas used for the splitting might be interesting for other matrix structures, especially in the case of  $\mathcal{H}$ -matrices. These also rely on recur-

## 7 Conclusion

sively splitting the matrix into submatrices. When weight matrices are approximated with  $\mathcal{H}$ -matrices, this requires nonstandard admissibility conditions to determine if a matrix can be represented using a low rank approximation.

The techniques to change the segmentation can be used without increasing the approximation error measured in the spectral norm. In this thesis, I did not test how this influences the total accuracy of the neural network. The approximation error does give an indication but does not fully describe the behavior. Future research is needed to determine how the approximation and the changing of the segmentation influences the accuracy of the neural network.



## A Matrix Approximation Error Bounds

In this chapter a proof for Equation 4.23

$$\|T - \hat{T}\| \leq \sum_{i=1}^K \|T - \hat{T}^{(i)}\|. \quad (\text{A.1})$$

and for Equation 4.23

$$\|T - \hat{T}\|_F \leq \sum_{i=1}^K \|T - \hat{T}^{(i)}\|_F \quad (\text{A.2})$$

is given. To prove the bound we decompose the change due to the balanced truncation into

$$T - \hat{T} = \begin{bmatrix} \Delta_0^* \\ \check{\Delta}_1 \end{bmatrix} + \begin{bmatrix} \check{\Delta}_1^* \\ \check{\Delta}_2 \end{bmatrix} + \dots + \begin{bmatrix} \check{\Delta}_{K-1}^* \\ \Delta_K \end{bmatrix}. \quad (\text{A.3})$$

The matrix  $\Delta_k$  denotes the difference caused by the balanced truncation of the state  $k$  on the original system

$$\Delta_k = \mathcal{O}_k \mathcal{R}_k - \hat{\mathcal{O}}_k \hat{\mathcal{R}}_k = \mathcal{O}_{k[2]} \mathcal{R}_{k[2]} \quad (\text{A.4})$$

and the matrix  $\check{\Delta}_k$  denotes the difference caused by a balanced truncation of the state  $k$  if the later stages have already been truncated using balanced truncation. For the causal system this can be expressed as

$$\check{\Delta}_k = \begin{bmatrix} C_k \\ \hat{\mathcal{O}}_{k+1} [A_{k+1[11]} \quad A_{k+1[12]}] \end{bmatrix} R_{k-} \begin{bmatrix} C_k \\ \hat{\mathcal{O}}_{k+1} A_{k+1[11]} \end{bmatrix} R_{k[1]} = \begin{bmatrix} C_{k[2]} \\ \hat{\mathcal{O}}_{k+1} A_{k+1[12]} \end{bmatrix} R_{k[2]}$$

Here the matrix  $\hat{\mathcal{O}}_{k+1}$  denotes the observability matrix of the approximated system

$$\hat{\mathcal{O}}_{k+1} = \begin{bmatrix} \hat{C}_{k+1} \\ \hat{C}_{k+2} \hat{A}_{k+1} \\ \vdots \\ \hat{C}_K \hat{A}_{K-1} \dots \hat{A}_{k+1} \end{bmatrix} = \begin{bmatrix} C_{k+1[1]} \\ C_{k+2[1]} A_{k+1[11]} \\ \vdots \\ C_{K[1]} A_{K-1[11]} \dots A_{k+1[11]} \end{bmatrix} \quad (\text{A.5})$$

Using the triangle inequality we obtain the relation

$$\|T - \hat{T}\| \leq \left\| \begin{bmatrix} \Delta_0^* \\ \check{\Delta}_1 \end{bmatrix} \right\| + \left\| \begin{bmatrix} \check{\Delta}_1^* \\ \check{\Delta}_2 \end{bmatrix} \right\| + \dots + \left\| \begin{bmatrix} \check{\Delta}_{K-1}^* \\ \Delta_K \end{bmatrix} \right\|. \quad (\text{A.6})$$

## A Matrix Approximation Error Bounds

**Spectral Norm** The idea is to prove that  $\|\Delta_k\|$  is equal or larger than the actual introduced difference  $\|\check{\Delta}_k\|$ . Using the derivation from Equation 4.20 we can then bound

$$\left\| \begin{bmatrix} \check{\Delta}_k & \check{\Delta}_{k-1}^* \end{bmatrix} \right\| \leq \max \left( \|\check{\Delta}_k\|, \|\check{\Delta}_{k-1}^*\| \right) \quad (\text{A.7})$$

To prove  $\|\check{\Delta}_k\| \leq \|\Delta_k\|$  it is sufficient to prove

$$\|\check{\Delta}_k u\|_2 \leq \|\Delta_k u\|_2 \quad (\text{A.8})$$

for any vector  $u$ .

*Proof.* For this we use the definition of the spectral norm

$$\|\check{\Delta}_k\| = \max_{\|u\|_2=1} (\|\check{\Delta}_k u\|_2) \leq \max_{\|u\|_2=1} (\|\Delta_k u\|_2) = \|\Delta_k\| \quad (\text{A.9})$$

□

For the causal case we start with the equation

$$\|\Delta_k u\|_2^2 = \|\mathcal{O}_{k[2]} \mathcal{R}_{k[2]} u\|_2^2 \quad (\text{A.10})$$

By inserting the definitions from Equation 3.23a and using the factorization from Equation 4.7 we obtain

$$\left\| \begin{bmatrix} I & \\ & \mathcal{O}_{K+1} \end{bmatrix} \begin{bmatrix} C_{k[2]} \\ A_{k[12]} \\ A_{k[22]} \end{bmatrix} \mathcal{R}_{k[2]} u \right\|_2^2 \quad (\text{A.11})$$

Now we use balanced truncation of the state  $x_{k+1}$ . This analogously splits up the observability matrix  $\mathcal{O}_{k+1}$  according to

$$\mathcal{O}_{k+1} = \begin{bmatrix} \mathcal{O}_{k+1[1]} & \mathcal{O}_{k+1[2]} \end{bmatrix}. \quad (\text{A.12})$$

Using this,  $\|\Delta_k u\|_2^2$  can be expressed as

$$\left\| \begin{bmatrix} C_{k[2]} \\ \mathcal{O}_{k+1[1]} A_{k[12]} + \mathcal{O}_{k+1[2]} A_{k[22]} \end{bmatrix} \mathcal{R}_{k[2]} u \right\|_2^2. \quad (\text{A.13})$$

Using the properties of the balanced realization the norm can be split up in the sum

$$\left\| \begin{bmatrix} C_{k[2]} \\ \mathcal{O}_{k+1[1]} A_{k[12]} \end{bmatrix} \mathcal{R}_{k[2]} u \right\|_2^2 + \left\| \mathcal{O}_{k+1[2]} A_{k[22]} \mathcal{R}_{k[2]} u \right\|_2^2. \quad (\text{A.14})$$

*Proof.* The Norm can be expressed as the inner product

$$\left\| \begin{bmatrix} I \\ \mathcal{O}_{K+1} \end{bmatrix} \begin{bmatrix} C_{k[2]} \\ A_{k[12]} \\ A_{k[22]} \end{bmatrix} \mathcal{R}_{k[2]} u \right\|_2^2 \quad (\text{A.15})$$

$$= u^\top \mathcal{R}_{k[2]}^\top \begin{bmatrix} C_{k[2]}^\top & A_{k[12]}^\top & A_{k[22]}^\top \end{bmatrix} \begin{bmatrix} I \\ \mathcal{O}_{K+1}^\top \end{bmatrix} \begin{bmatrix} I \\ \mathcal{O}_{K+1} \end{bmatrix} \begin{bmatrix} C_{k[2]} \\ A_{k[12]} \\ A_{k[22]} \end{bmatrix} \mathcal{R}_{k[2]} u.$$

using the fact that the system is balanced and therefore  $\mathcal{O}_{k+1}^\top \mathcal{O}_{k+1} = \Sigma_{k+1}$

$$u^\top \mathcal{R}_{k[2]}^\top \begin{bmatrix} C_{k[2]}^\top & A_{k[12]}^\top & A_{k[22]}^\top \end{bmatrix} \begin{bmatrix} I \\ \Sigma_{K+1} \end{bmatrix} \begin{bmatrix} C_{k[2]} \\ A_{k[12]} \\ A_{k[22]} \end{bmatrix} \mathcal{R}_{k[2]} u \quad (\text{A.16})$$

By splitting  $\Sigma_{k+1}$  with the appropriate dimensions such that  $\Sigma_{k+1} = \text{diag}(\Sigma_{k+1[1]}, \Sigma_{k+1[2]})$  we can rewrite the expression as

$$u^\top \mathcal{R}_{k[2]}^\top \begin{bmatrix} C_{k[2]}^\top & A_{k[12]}^\top & A_{k[22]}^\top \end{bmatrix} \begin{bmatrix} I \\ \Sigma_{K+1[1]} \\ \Sigma_{K+1[2]} \end{bmatrix} \begin{bmatrix} C_{k[2]} \\ A_{k[12]} \\ A_{k[22]} \end{bmatrix} \mathcal{R}_{k[2]} u. \quad (\text{A.17})$$

This allows us to regroup the expression as the sum

$$= u^\top \mathcal{R}_{k[2]}^\top \begin{bmatrix} C_{k[2]}^\top & A_{k[12]}^\top \end{bmatrix} \begin{bmatrix} I \\ \Sigma_{K+1[1]} \end{bmatrix} \begin{bmatrix} C_{k[2]} \\ A_{k[12]} \end{bmatrix} \mathcal{R}_{k[2]} u \quad (\text{A.18})$$

$$+ u^\top \mathcal{R}_{k[2]}^\top A_{k[22]}^\top \Sigma_{K+1[2]} A_{k[22]} \mathcal{R}_{k[2]} u. \quad (\text{A.19})$$

It is also possible to prove this using the fact that  $\text{range}(\mathcal{O}_{k+1[1]}) \perp \text{range}(\mathcal{O}_{k+1[2]})$  and then employing  $(u+v)^\top (u+v) = u^\top u + v^\top v$  for  $u \perp v$   $\square$

Now we again use Equation 4.7 to rewrite the first summand as

$$\left\| \begin{bmatrix} C_{k[2]} \\ \mathcal{O}_{k+1[1]} A_{k[12]} \end{bmatrix} \mathcal{R}_{k[2]} u \right\|_2^2 = \left\| \begin{bmatrix} I \\ I \\ \mathcal{O}_{k+2} \end{bmatrix} \begin{bmatrix} C_{k[2]} \\ C_{k+1[1]} A_{k[12]} \\ A_{k+1[11]} \\ A_{k+1[21]} A_{k[12]} \end{bmatrix} \mathcal{R}_{k[2]} u \right\|_2^2$$

## A Matrix Approximation Error Bounds

Analogously we can again split the norm into the sum

$$= \left\| \begin{bmatrix} I & & \\ & I & \\ & & \mathcal{O}_{k+2} \end{bmatrix} \begin{bmatrix} C_{k[2]} \\ C_{k+1[1]}A_{k[12]} \\ \begin{bmatrix} A_{k+2[11]} \\ A_{k+2[21]} \end{bmatrix} A_{k[12]} \end{bmatrix} \mathcal{R}_{k[2]}u \right\|_2^2 \quad (\text{A.20})$$

$$= \left\| \begin{bmatrix} C_{k[2]} \\ C_{k+1[1]}A_{k[12]} \\ \mathcal{O}_{k+2[1]}A_{k+2[11]}A_{k[12]} + \mathcal{O}_{k+2[2]}A_{k+2[21]}A_{k[12]} \end{bmatrix} \mathcal{R}_{k[2]}u \right\|_2^2 \quad (\text{A.21})$$

$$= \left\| \begin{bmatrix} C_{k[2]} \\ C_{k+1[1]}A_{k[12]} \\ \mathcal{O}_{k+2[1]}A_{k+2[11]}A_{k[12]} + \end{bmatrix} \mathcal{R}_{k[2]}u \right\|_2^2 + \left\| \mathcal{O}_{k+2[2]}A_{k+2[21]}A_{k[12]}\mathcal{R}_{k[2]}u \right\|_2^2 \quad (\text{A.22})$$

This can be repeated until we reach the final state  $K$ . The remaining term is

$$\left\| \begin{bmatrix} C_{k[2]} \\ C_{k+1[1]}A_{k[12]} \\ C_{k+2[1]}A_{k+1[11]}A_{k[12]} \\ \vdots \\ C_{K[1]}A_{K-1[11]} \cdots A_{k+1[11]}A_{k[12]} \end{bmatrix} \mathcal{R}_{k[2]}u \right\|_2^2. \quad (\text{A.23})$$

The remaining term can be identified as

$$\left\| \begin{bmatrix} C_{k[2]} \\ \hat{\mathcal{O}}_{k+1}A_{k[12]} \end{bmatrix} \mathcal{R}_{k[2]}u \right\|_2^2 = \left\| \check{\Delta}_k u \right\|_2^2. \quad (\text{A.24})$$

This allows us to decompose  $\|\Delta_k u\|_2^2$  into the sum

$$\|\Delta_k u\|_2^2 = \|\check{\Delta}_k u\|_2^2 + \cdots + \left\| \mathcal{O}_{k+2[2]}A_{k+2[21]}A_{k[12]}\mathcal{R}_{k[2]}u \right\|_2^2 + \left\| \mathcal{O}_{k+1[2]}A_{k[22]}\mathcal{R}_{k[2]}u \right\|_2^2$$

By reordering and using the fact that the squared norms are larger or equal than 0 we obtain the relation

$$\|\check{\Delta}_k u\|_2 \leq \|\Delta_k u\|_2. \quad (\text{A.25})$$

This proves the relation from Equation A.8 for the causal matrices. The anticausal case can be proven by transposing the matrix. In this case the state 1 is approximated first as indicated in Equation A.3. The details are left as an exercise to the examiner.

**Frobenius Norm** Using Equation A.8 it is also straightforward to prove the relation for the Frobenius norm as

$$\|M\|_F^2 = \text{trace}(M^\top M) = \sum_{i=1}^N e_i^\top M^\top M e_i = \sum_{i=1}^N \|M e_i\|_2^2 \quad (\text{A.26})$$

where  $e_i$  is the  $i$ -th standard basis vector. The relation  $\|\check{\Delta}_k e_i\|_2^2 \leq \|\Delta_k e_i\|_2^2$  follows from Equation A.25 and implies that

$$\|\check{\Delta}_k\|_F \leq \|\Delta_k\|_F \quad (\text{A.27})$$

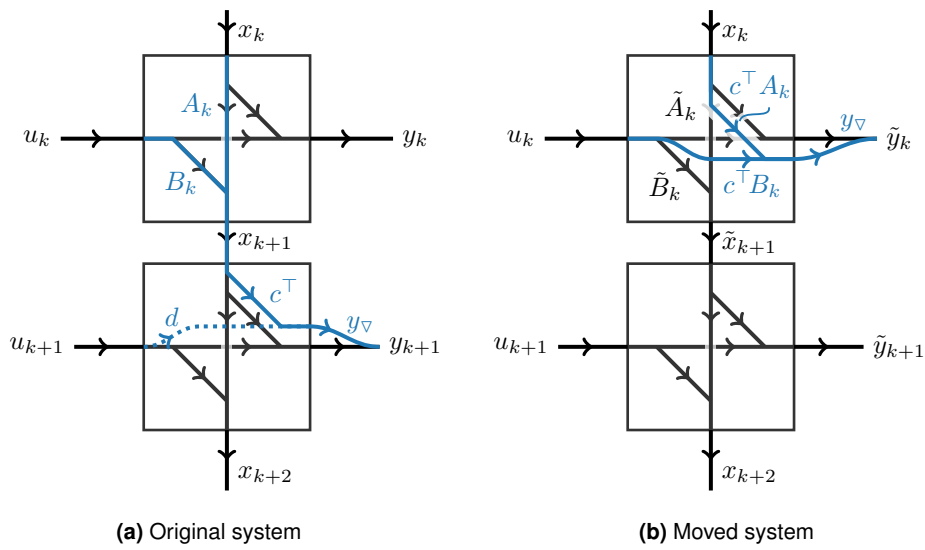
Therefore

$$\|T - \hat{T}\|_F \leq \left\| \begin{bmatrix} \check{\Delta}_1 & \Delta_0^* \end{bmatrix} \right\|_F + \left\| \begin{bmatrix} \check{\Delta}_2 & \check{\Delta}_1^* \end{bmatrix} \right\|_F + \dots + \left\| \begin{bmatrix} \Delta_K & \check{\Delta}_{K-1} \end{bmatrix} \right\|_F. \quad (\text{A.28})$$



## B Algorithms to Move Bounds Up or Down

In this Appendix the complementary algorithms to the algorithms stated in Subsection 4.4.1 are described.



**Figure B.1:** Illustration of a system where a boundary is moved down

**Move Down** A similar to Algorithm 3 can be used to move the  $k$ -th horizontal a boundary down. That means moving the first output  $y_\nabla$  from  $y_{k+1}$  to  $y_k$ . The altered connections are drawn in color in Figure B.1.

The vector  $c^\top$  is the first row of  $C_{k+1}$  and describes the connection from  $x_k$  to  $y_\nabla$ . This vector is removed from  $C_{k+1}$ . We also remove the first row from  $D_{k+1}$ . The dotted connection from  $u_{k+1}$  to the output  $y_\nabla$  is no longer possible and therefore we have to drop it. Now we attach the output  $y_\nabla$  to the output vector  $y_k$ . For this the vector  $c^\top B_k$  is added as last row to  $D_k$  and the vector  $c^\top A_k$  is added as last row to  $C_k$ .

The mapping from the state  $x_k$  to the combined outputs  $[y_k^\top y_{k+1}^\top]^\top$  remains unchanged. This means that the previous states are still observable and therefore

## B Algorithms to Move Bounds Up or Down

**Input:** System  $\Sigma_{[A,B,C,D]}$ , Index  $k$ , Tolerance  $\text{tol}$   
**Output:** System  $\Sigma_{[A,B,C,D]}$  with changed output dimensions

$$c^\top \leftarrow C_{k+1}[1,:]$$

$$D_{k+1} \leftarrow D_{k+1}[2,:]$$

$$C_{k+1} \leftarrow C_{k+1}[2,:]$$

$$D_k \leftarrow \begin{bmatrix} D_k \\ c^\top B_k \end{bmatrix}$$

$$C_k \leftarrow \begin{bmatrix} C_k \\ c^\top A_k \end{bmatrix}$$

$$U, \sigma, V^\top \leftarrow \text{SVD} \left( \begin{bmatrix} A_{k+1} \\ C_{k+1} \end{bmatrix}, \text{econ} = \text{True} \right)$$

**if**  $\sigma_{[\text{end}]} < \text{tol}$  **then**

$$U \leftarrow U[:,:\text{end}-1]; \sigma \leftarrow \sigma[:,:\text{end}-1]; V \leftarrow V[:,:\text{end}-1]$$

$$\begin{bmatrix} A_{k+1} \\ C_{k+1} \end{bmatrix} \leftarrow U \text{diag}(\sigma)$$

$$A_k \leftarrow V^\top A_k$$

$$C_k \leftarrow V^\top C_k$$

**end if**

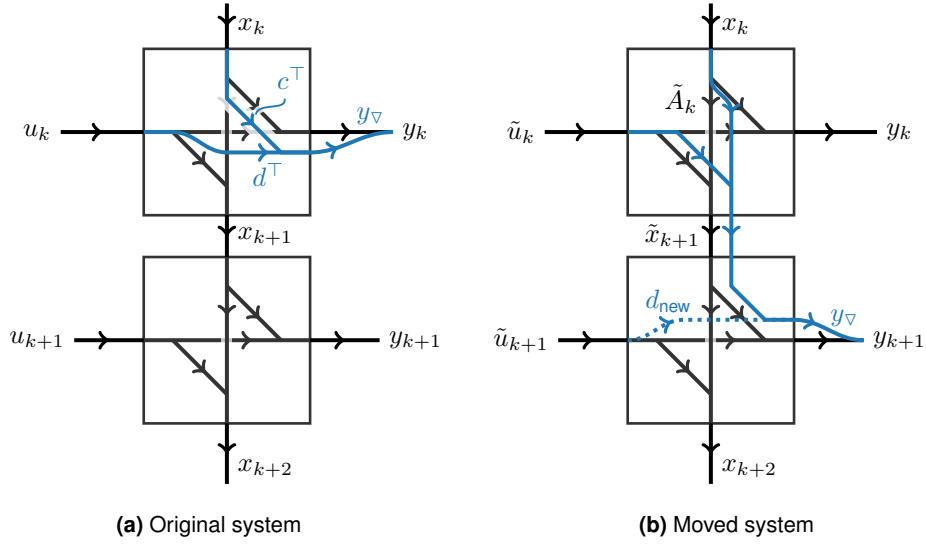
**Algorithm 9:** Move boundary between  $y_k$  and  $y_{k+1}$  down

minimal. As the algorithm removes a row from  $\mathcal{O}_{k+1}$  it might happen that the state  $x_{k+1}$  is no longer observable. This is the case if the kernel of  $\mathcal{O}_{k+1}$  is not the zero subspace. We can check this with the SVD of  $[A_{k+1}^\top, \check{C}_{k+1}^\top]^\top$ . If the last singular value is equal to zero one state dimension has to be removed. This is done by using a state transformation based on the reduced SVD. Again the inverse does not need to be calculated as it is implicitly calculated by the SVD. An algorithm to move the state is given in Algorithm 9.

**Move Up** A similar algorithm to Algorithm 4 makes it possible to move the the  $k$ -th horizontal boundary up. When moving the boundary up the output  $y_\nabla$  is removed from the output  $y_k$  and attached to the output vector  $y_{k+1}$ . Here the output  $y_\nabla$  is constructed in the stage  $k$  and appended to the state  $x_{k+1}$ . First, the last row of  $D_k$  is removed and stored in  $d^\top$  and analogously the last row of  $C_k$  is removed and stored in  $c^\top$ . The vector  $c^\top$  is appended to the matrix  $A_k$  and the vector  $d^\top$  is attached to  $B_k$ . Now we route the new statedimension to the new output. This can be done by setting

$$\check{C}_{k+1} = \begin{bmatrix} 0 & 1 \\ C_{k+1} & 0 \end{bmatrix}. \quad (\text{B.1})$$





**Figure B.2:** Illustration of a system where a boundary is moved up

This is illustrated in Figure B.2. The mapping from the state  $x_k$  to the combined outputs  $[y_k^\top y_{k+1}^\top]^\top$  remains unchanged. Therefore the previous states are still observable and minimal.

The algorithm changes the observability matrix  $\tilde{O}_{k+1}$  and reachability matrix  $\tilde{\mathcal{R}}_{k+1}$  of the state  $x_{k+1}$ . The state  $x_{k+1}$  is still observable as the added state dimension is directly connected to the output  $y_{k+1}$ . The algorithm adds a new row to the reachability matrix

$$\tilde{\mathcal{R}}_{k+1} = \begin{bmatrix} A_k & B_k \\ c^\top & d^\top \end{bmatrix} \begin{bmatrix} \mathcal{R}_k & 0 \\ 0 & I \end{bmatrix}. \quad (\text{B.2})$$

If this new row is part of the co-range of  $\mathcal{R}_{k+1}$ , then the system is no longer reachable. In this case no additional state is needed. Using a similar derivation as for the previous move we can check this using the SVD of  $[A_{k+1} B_{k+1}]^\top$ . If no new state is needed, then we can simply attach the vector

$$m = \text{pinv} \left( \begin{bmatrix} A_{k+1}^\top \\ B_{k+1}^\top \end{bmatrix} \right) \begin{bmatrix} c \\ d \end{bmatrix} \quad (\text{B.3})$$

to the matrix  $C_{k+1}$ .

The pseudocode can be found in Algorithm 10.

*B Algorithms to Move Bounds Up or Down*

**Input:** System  $\Sigma_{[A,B,C,D]}$ , Index  $k$ , Tolerance  $\text{tol}_r$   
**Output:** System  $\Sigma_{[A,B,C,D]}$  with changed output dimensions

$$c^\top \leftarrow C_k[\text{end},:]$$

$$d^\top \leftarrow D_k[\text{end},:]$$

$$C_k \leftarrow C_k[:\text{end}-1,:]$$

$$D_k \leftarrow D_k[:\text{end}-1,:]$$

$$U, \sigma, V^\top \leftarrow \text{SVD} \left( \begin{bmatrix} A_{k+1}^\top \\ B_{k+1}^\top \end{bmatrix} \right)$$

$$r \leftarrow \text{count}(\sigma > \text{tol}_i)$$

$$a \leftarrow U^\top \begin{bmatrix} c \\ d \end{bmatrix}$$

**if**  $\|a_{[r+1:]}\| > \text{tol}$  ▷ not in range

$$A_k \leftarrow \begin{bmatrix} A_k \\ c^\top \end{bmatrix}$$

$$B_k \leftarrow \begin{bmatrix} B_k \\ d^\top \end{bmatrix}$$

$$C_{k+1} \leftarrow \begin{bmatrix} 0 & 1 \\ C_{k+1} & 0 \end{bmatrix}$$

$$A_{k+1} \leftarrow \begin{bmatrix} A_{k+1} & 0 \end{bmatrix}$$

**else**

$$m \leftarrow V_{[:,r]} \text{diag}(\sigma_{[r]})^{-1} a_{[r]}$$

$$C_{k+1} \leftarrow \begin{bmatrix} m^\top \\ C_{k+1} \end{bmatrix}$$

**end if**

$$D_k \leftarrow \begin{bmatrix} d_{\text{new}} \\ D_k \end{bmatrix}$$

**Algorithm 10:** Move boundary between  $y_k$  and  $y_{k+1}$  up

## C Random Sequentially Semiseparable Matrices

In this section the algorithm to construct the sequentially semiseparable matrices used in Chapter 5 is explained. The matrices can be parameterized with the the input dimensions  $m_k$ , the output dimensions  $p_k$  and the rank of the Hankel matrices. The generated matrices have the property, that the range of  $C$ ,  $D$  and  $G$  are orthogonal to each other. Also the range of  $B^\top$ ,  $D^\top$  and  $G^\top$  are orthogonal.

To archive the orthogonality, random orthogonal matrices form the Haar distribution are used. The matrices are generated by the algorithm described in [34], that is available in *SciPy* [48]. Here we denote the group of orthogonal matrices with  $n$  dimensions as  $O(n)$ .

For every stage  $i \in 1, \dots, K$  the matrices

$$\tilde{U}_k = p_k U_k \quad \text{with } U_k \in O(p_k) \quad (\text{C.1})$$

$$\tilde{V}_k = m_k V_k \quad \text{with } V_k \in O(m_k) \quad (\text{C.2})$$

are generated. The matrices are scaled with  $p_k$  and  $m_k$  respectively to make sure that

$$\frac{\|v\|_2}{\text{len}(v)} = 1 \quad (\text{C.3})$$

for all columns of  $\tilde{V}_k$  and  $\tilde{U}_k$ . The random sequentially semipermeable matrix  $T$  with Hankel matrices of rank  $d$  is constructed form the block matrices

$$T_{[i,j]} = \begin{cases} \tilde{U}_{i[:,d]} \tilde{V}_{j[:,d]}^\top & \text{if } i < j \\ \tilde{U}_{i[d+1,2d]} \tilde{V}_{j[d+1,2d]}^\top & \text{if } i = j \\ \tilde{U}_{i[2d+1,3d]} \tilde{V}_{j[2d+1,3d]}^\top & \text{if } i > j \end{cases} \quad (\text{C.4})$$

These are then combined according to

$$T = \begin{bmatrix} T_{[1,1]} & T_{[1,2]} & \cdots & T_{[1,k]} \\ T_{[2,1]} & T_{[2,2]} & \cdots & T_{[2,k]} \\ \vdots & \vdots & \ddots & \vdots \\ T_{[K,1]} & T_{[K,2]} & \cdots & T_{[K,K]} \end{bmatrix} \quad (\text{C.5})$$



## Bibliography

- [1] V. M. Adamjan, D. Z. Arov, and M. G. Kreĭn. “Analytic properties of Schmidt pairs for a Hankel operator and the generalized Schur-Takagi problem”. In: *Mathematics of the USSR-Sbornik* 15(1) (Feb. 28, 1971). Publisher: IOP Publishing, Translated by: J. C. Lennox. ISSN: 0025-5734. DOI: 10.1070/SM1971v015n01ABEH001531.
- [2] N. Ailon, O. Leibovitch, and V. Nair. “Sparse linear networks with a fixed butterfly structure: theory and practice”. In: *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*. Uncertainty in Artificial Intelligence. ISSN: 2640-3498. PMLR, Dec. 1, 2021, pp. 1174–1184.
- [3] A. C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. Advances in Design and Control. Society for Industrial and Applied Mathematics, 2005. ISBN: 978-0-89871-871-3. DOI: 10.1137/1.9780898718713.
- [4] Z.-Z. Bai and J.-Y. Pan. *Matrix Analysis and Computations*. Other Titles in Applied Mathematics. Philadelphia: Society for Industrial and Applied Mathematics, Jan. 2021. ISBN: 978-1-61197-662-5. DOI: 10.1137/1.9781611976632.
- [5] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Gutttag. “What is the State of Neural Network Pruning?” In: *Proceedings of Machine Learning and Systems* 2 (Mar. 15, 2020), pp. 129–146.
- [6] M. Brand. “Fast low-rank modifications of the thin singular value decomposition”. In: *Linear Algebra and its Applications*. Special Issue on Large Scale Linear and Nonlinear Eigenvalue Problems 415(1) (May 1, 2006), pp. 20–30. ISSN: 0024-3795. DOI: 10.1016/j.laa.2005.07.021.
- [7] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A. J. van der Veen, and D. White. “Some Fast Algorithms for Sequentially Semiseparable Representations”. In: *SIAM Journal on Matrix Analysis and Applications* 27(2) (Jan. 2005). Publisher: Society for Industrial and Applied Mathematics, pp. 341–364. ISSN: 0895-4798. DOI: 10.1137/S0895479802405884.
- [8] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A. J. van der Veen. “Fast Stable Solver for Sequentially Semi-separable Linear Systems of Equations”. In: *High Performance Computing — HiPC 2002*. Ed. by S. Sahni, V. K. Prasanna, and U. Shukla. Berlin, Heidelberg: Springer, 2002, pp. 545–554. ISBN: 978-3-540-36265-4. DOI: 10.1007/3-540-36265-7\_51.

## Bibliography

- [9] T. Dao, N. Sohoni, A. Gu, M. Eichhorn, A. Blonder, M. Leszczynski, A. Rudra, and C. Ré. “Kaleidoscope: An Efficient, Learnable Representation For All Structured Linear Maps”. In: Eighth International Conference on Learning Representations. Apr. 2020.
- [10] T. Dettmers and L. Zettlemoyer. “Sparse Networks from Scratch: Faster Training without Losing Performance”. In: *arXiv:1907.04840 [cs, stat]* (Aug. 23, 2019). arXiv: 1907.04840.
- [11] P. Dewilde and A.-J. van der Veen. *Time-Varying Systems and Computations*. Jan. 1, 1998. ISBN: 978-0-7923-8189-1. DOI: 10.1007/978-1-4757-2817-0.
- [12] K. Diepold, P. Dewilde, and W. Bamberger. “Optic flow computation and time-varying system theory”. In: *MTNS 2004; Proceedings sixteenth international symposium on mathematical theory of networks and systems* (2004). Ed. by B. de Moor and B. Motmans. Place: Leuven Publisher: Katholieke Universiteit Leuven, pp. 1–7. ISSN: 90-5682-517-8.
- [13] Y. Fan, L. Lin, L. Ying, and L. Zepeda-Núñez. “A Multiscale Neural Network Based on Hierarchical Matrices”. In: *Multiscale Modeling & Simulation* 17(4) (Jan. 2019). Publisher: Society for Industrial and Applied Mathematics, pp. 1189–1213. ISSN: 1540-3459. DOI: 10.1137/18M1203602.
- [14] M. Fiedler and T.L. Markham. “Completing a matrix when certain entries of its inverse are specified”. In: *Linear Algebra and its Applications* 74 (Feb. 1, 1986), pp. 225–237. ISSN: 0024-3795. DOI: 10.1016/0024-3795(86)90125-4. URL: <https://www.sciencedirect.com/science/article/pii/0024379586901254> (visited on 06/27/2022).
- [15] L. Grasedyck, W. Hackbusch, and S.L. Borne. “Adaptive Geometrically Balanced Clustering of H-Matrices”. In: *Computing* 73(1) (July 1, 2004), pp. 1–23. ISSN: 1436-5057. DOI: 10.1007/s00607-004-0068-0.
- [16] L. Grasedyck. “Theorie und Anwendungen Hierarchischer Matrizen”. Accepted: 2001-07-20. Ph.D. thesis. Christian-Albrechts-Universität zu Kiel, Aug. 27, 2001.
- [17] W. Hackbusch. *Hierarchische Matrizen: Algorithmen und Analysis*. Ed. by W. Hackbusch. Berlin, Heidelberg: Springer, 2009. ISBN: 978-3-642-00222-9. DOI: 10.1007/978-3-642-00222-9.
- [18] B. Hassibi, D. Stork, and G. Wolff. “Optimal Brain Surgeon and general network pruning”. In: *IEEE International Conference on Neural Networks*. IEEE International Conference on Neural Networks. Mar. 1993, 293–299 vol.1. DOI: 10.1109/ICNN.1993.298572.

- [19] D. J. Higham, G. Kalna, and M. Kibble. “Spectral clustering and its use in bioinformatics”. In: *Journal of Computational and Applied Mathematics*. Special issue dedicated to Professor Shinnosuke Oharu on the occasion of his 65th birthday 204(1) (July 1, 2007), pp. 25–37. ISSN: 0377-0427. DOI: 10.1016/j.cam.2006.04.026.
- [20] D. Hinrichsen and A. Pritchard. “An improved error estimate for reduced-order models of discrete-time systems”. In: *IEEE Transactions on Automatic Control* 35(3) (1990), pp. 317–320.
- [21] Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, and A. Criminisi. “Training CNNs with Low-Rank Filters for Efficient Image Classification”. In: International Conference on Learning Representations. May 2, 2016. DOI: 10.13140/RG.2.1.3727.8163.
- [22] V. K. Ithapu. “Decoding the Deep: Exploring Class Hierarchies of Deep Representations Using Multiresolution Matrix Factorization”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. 2017, pp. 45–54.
- [23] M. Jaderberg, A. Vedaldi, and A. Zisserman. “Speeding up Convolutional Neural Networks with Low Rank Expansions”. In: *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. DOI: 10.5244/C.28.88.
- [24] M. Kissel, D. Stümke, and S. Nüßlein. *Time Varying Systems and Computation (TVSC) Library*. Apr. 14, 2022. URL: <https://github.com/MatthiasKi/tvsclib>.
- [25] D. E. Knuth. “Johann Faulhaber and sums of powers”. In: *Mathematics of Computation* 61(203) (1993), pp. 277–294. ISSN: 0025-5718, 1088-6842. DOI: 10.1090/S0025-5718-1993-1197512-7.
- [26] A. Krizhevsky. *One weird trick for parallelizing convolutional neural networks*. (arXiv:1404.5997). type: article. arXiv, Apr. 26, 2014. arXiv: 1404.5997[cs].
- [27] S. Lall and C. Beck. “Error-bounds for balanced model-reduction of linear time-varying systems”. In: *IEEE Transactions on Automatic Control* 48(6) (June 2003). Conference Name: IEEE Transactions on Automatic Control, pp. 946–956. ISSN: 1558-2523. DOI: 10.1109/TAC.2003.812779.
- [28] Y. Li, X. Cheng, and J. Lu. “Butterfly-Net: Optimal Function Representation Based on Convolutional Neural Networks”. In: *Communications in Computational Physics* 28(5) (June 2020), pp. 1838–1885. ISSN: 1815-2406, 1991-7120. DOI: 10.4208/cicp.OA-2020-0214.

## Bibliography

- [29] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying. “Butterfly Factorization”. In: *Multiscale Modeling & Simulation* 13(2) (Jan. 2015). Publisher: Society for Industrial and Applied Mathematics, pp. 714–732. ISSN: 1540-3459. DOI: 10.1137/15M1007173.
- [30] Z. Liu and L. Vandenberghe. “Interior-Point Method for Nuclear Norm Approximation with Application to System Identification”. In: *SIAM Journal on Matrix Analysis and Applications* 31(3) (Jan. 2010). Publisher: Society for Industrial and Applied Mathematics, pp. 1235–1256. ISSN: 0895-4798. DOI: 10.1137/090755436.
- [31] C. Louizos, M. Welling, and D. P. Kingma. “Learning Sparse Neural Networks through L<sub>0</sub> Regularization”. In: International Conference on Learning Representations. Feb. 15, 2018.
- [32] S. Marcel and Y. Rodriguez. “Torchvision the machine-vision package of torch”. In: *Proceedings of the 18th ACM international conference on Multimedia*. MM ’10. New York, NY, USA: Association for Computing Machinery, Oct. 25, 2010, pp. 1485–1488. ISBN: 978-1-60558-933-6. DOI: 10.1145/1873951.1874254.
- [33] C. H. Martin and M. W. Mahoney. “Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning”. In: *Journal of Machine Learning Research* 22(165) (2021), pp. 1–73. URL: <http://jmlr.org/papers/v22/20-410.html>.
- [34] F. Mezzadri. “How to generate random matrices from the classical compact groups”. In: *Notices of the American Mathematical Society* 54(5) (May 2007), pp. 592–604. ISSN: 0002-9920.
- [35] V. Pappas. “Traces of Class/Cross-Class Structure Pervade Deep Learning Spectra”. In: *Journal of Machine Learning Research* 21(252) (2020), pp. 1–64. ISSN: 1533-7928. URL: <http://jmlr.org/papers/v21/20-933.html>.
- [36] D. S. Parker. *Random Butterfly Transformations with Applications in Computational Linear Algebra*. 1995. URL: <http://web.cs.ucla.edu/~stott/ge/CSD-950023.pdf>.
- [37] B. Recht, M. Fazel, and P. A. Parrilo. “Guaranteed Minimum-Rank Solutions of Linear Matrix Equations via Nuclear Norm Minimization”. In: *SIAM Review* 52(3) (Jan. 2010), pp. 471–501. ISSN: 0036-1445, 1095-7200. DOI: 10.1137/070697835.
- [38] J. K. Rice. “Efficient Algorithms for Distributed Control: A Structured Matrix Approach”. Ph.D. thesis. Technische Universiteit Delft, 2010. URL: <https://repository.tudelft.nl/islandora/object/uuid%3A50460cc9-2aba-4db5-b602-5373bb0bc609>.



- [39] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua. “Learning Separable Filters”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 2754–2761.
- [40] H. Sandberg and A. Rantzer. “Balanced truncation of linear time-varying systems”. In: *IEEE Transactions on Automatic Control* 49(2) (Feb. 2004). Conference Name: IEEE Transactions on Automatic Control, pp. 217–229. ISSN: 1558-2523. DOI: 10.1109/TAC.2003.822862.
- [41] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Mar. 21, 2019.
- [42] R. Schwartz, J. Dodge, N.A. Smith, and O. Etzioni. “Green AI”. In: *Commun. ACM* 63(12) (Nov. 2020), pp. 54–63. ISSN: 0001-0782. DOI: 10.1145/3381831.
- [43] S. Shokoohi and L. M. Silverman. “Identification and model reduction of time-varying discrete-time systems”. In: *Automatica* 23(4) (July 1, 1987), pp. 509–521. ISSN: 0005-1098. DOI: 10.1016/0005-1098(87)90080-X.
- [44] G. Strang. *Computational science and engineering*. Wellesley, Ma.: Wellesley-Cambridge Press, 2007. XI, 735 S. ISBN: 978-0-9614088-1-7.
- [45] L. Tong, A.-J. van der Veen, P. Dewilde, and Y. Sung. “Blind decorrelating RAKE receivers for long-code WCDMA”. In: *IEEE Transactions on Signal Processing* 51(6) (June 2003). Conference Name: IEEE Transactions on Signal Processing, pp. 1642–1655. ISSN: 1941-0476. DOI: 10.1109/TSP.2003.811230.
- [46] R. Vandebril, M. V. Barel, G. Golub, and N. Mastronardi. “A bibliography on semiseparable matrices”. In: *CALCOLO* 42(3) (Dec. 1, 2005), pp. 249–270. ISSN: 1126-5434. DOI: 10.1007/s10092-005-0107-z.
- [47] R. Vandebril, N. Mastronardi, and M. Van Barel. *Matrix Computations and Semiseparable Matrices*. Baltimore, UNITED STATES: Johns Hopkins University Press, 2007. ISBN: 978-0-8018-9679-8.
- [48] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Í. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, and P. van Mulbregt. “SciPy 1.0: fundamental algorithms for scientific computing in Python”. In: *Nature Methods* 17(3) (Mar. 2020). Number: 3 Publisher: Nature Publishing Group, pp. 261–272. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0686-2.

## Bibliography

- [49] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. “Learning Structured Sparsity in Deep Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016.
- [50] B. Wu, D. Wang, G. Zhao, L. Deng, and G. Li. “Hybrid tensor decomposition in neural network compression”. In: *Neural Networks* 132 (Dec. 1, 2020), pp. 309–320. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2020.09.006.
- [51] X. Yu, T. Liu, X. Wang, and D. Tao. “On Compressing Deep Models by Low Rank and Sparse Decomposition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7370–7379.
- [52] T. Zhou and D. Tao. “Greedy Bilateral Sketch, Completion & Smoothing”. In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*. Artificial Intelligence and Statistics. ISSN: 1938-7228. PMLR, Apr. 29, 2013, pp. 650–658.