Multigrid methods for high-dimensional, tensor structured continuous time Markov chains

Dissertation

Bergische Universität Wuppertal
Fakultät für Mathematik und Naturwissenschaften

eingereicht von
Sonja Sokolović, M. Sc.
zur Erlangung des Grades eines Doktors der Naturwissenschaften

Wuppertal, 12.01.2017

Die Dissertation kann wie folgt zitiert werden:

urn:nbn:de:hbz:468-20170519-115607-9
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20170519-115607-9]

Für Marcel & Maja

Contents

Chapter 1

Introduction

Continuous time Markov chains (CTMCs) are used to describe many real world processes. These processes include manufacturing systems [55], chemical reactions [1, 52], queueing networks [19, 20, 44] and many more. Many of these processes have the property that they can "naturally" be split into d subsystems. These subsystems can either interact with each other or they act within their subsystem. For the modelling of these processes this property is exploited, which then leads to a generator matrix of the form

$$A = \sum_{(s,t)} \bigotimes_i E_i^{(s,t)} - \sum_{(s,t)\neq(i,i)} \bigotimes_i D_i^{(s,t)} \in \mathbb{R}^{\prod n_i \times \prod n_i},$$

where $i, s, t \in \{1, \dots, d\}$ and $E_i^{(s,t)}, D_i^{(s,t)} \in \mathbb{R}^{n_i \times n_i}$. With this generator matrix, the so-called stationary distribution which describes the long term behaviour of the process, can be computed by solving the singular linear system

$$Ax = \mathbf{0}$$

subject to the constraint that $\boldsymbol{1}^T x = 1$.

For models with a large number of subsystems, this task is rendered difficult by the fact that the size of the generator matrix explodes, because it is given by the product of the sizes of the matrices from each subsystem. This phenomenon is known as the *curse of dimensionality*. Therefore, designing efficient iterative methods for solving this task is not enough. It is equally important to exploit the naturally given structure of the generator matrix in order to be able to store it and work with it. For this, a variety of different low-rank formats can be used, which keep the curse of dimensionality in check. Tensor Train [56, 57, 59] will be the format of choice in this thesis.

The task of this thesis is now to develop a multigrid method which exploits the tensor structure and uses techniques for efficient matrix-vector multiplications.

The challenge in this is to maintain the tensor structure across all grids, which, e.g., means serious constraints for the transfer operators that can be used. The idea for using a multigrid approach for computing the stationary distribution of a CTMC is based on the encouraging results obtained when using multigrid for discrete time Markov chains without tensor structure [6,12,17,24–26,73] and also on the nice structure of the generator matrix which motivates using an approach which is based on the graph corresponding to the matrix.

This thesis is structured as follows: In chapter 2 the definition and properties of continuous time Markov chains are discussed. Moreover, it is explained what a stationary distribution of a CTMC is and why we can compute it by solving a singular linear system with right-hand side zero. Additional assumptions under which the existence and uniqueness of a stationary distribution of a CTMC are guaranteed are also presented. Subsequently, different examples of CTMCs are shown. Therein the focus is on the construction of the corresponding generator matrices. At the end of chapter 2 the curse of dimensionality, which motivates this work, is discussed. In chapter 3 the Tensor Train format (TT) is presented and the TT-SVD algorithm, which computes a TT-representation of a vector, is given. Moreover, the definition of a TT-matrix, an adaption of the Tensor Train format to matrices, and the implementation of basic linear algebra operations (like matrix-vector multiplication) in Tensor Train formats are explained. In chapter 4 a first iterative method is considered for solving the singular system. This first method of choice is the Krylov subspace method GMRES [64]. Because of the fact that the system is singular and non-symmetric it has to be clarified under which conditions convergence of a Krylov subspace method can be guaranteed. Simple numerical examples show that GMRES can fail, but at the same time give a motivation for using a multigrid method with GMRES as smoother. Following this observation, the basic ideas and ingredients of multigrid methods are introduced. When using a multigrid method, it would be desirable to have a convergence analysis at hand which guarantees that the method always finds the stationary distribution. Even for symmetric, singular or non-symmetric, non-singular systems only few convergence results for multigrid can be found in the literature [7, 13, 65]. In chapter 5 these results are summarized and we try to adapt them to our setting. Unfortunately both approaches fail to be generalizable to the non-symmetric singular case, and we highlight the reasons for this. In chapter 6 we investigate each ingredient of a multigrid method separately, focusing on its capability to be used for tensor structured problems. Based on this, there will be different choices for each ingredient. These will be tested and judged in chapter 7 in a first series of numerical experiments. The construction of the multigrid method in chapter 6 is based on the geometric structure of the models. A strategy which generates the ingredients adaptively instead is therefore desirable. We will concentrate on the adaptive construction of the interpolation operators by generalizing the boot-

strap AMG approach [6, 11, 12, 23, 27, 43] to the tensor setting. Here the main task is to find reasonable test vectors for the least squares interpolation. Chapter 8 is therefore structured as follows: first a description of a bootstrap approach is given and afterwards the adaption to the tensor structured case is discussed. The resulting method is then tested and compared to the method from chapter 6 in further numerical experiments. In chapter 9 an alternative low-rank tensor method for finding the stationary distribution of a CTMC is presented, namely the alternating minimal energy method (AMEn) [29, 30, 47, 48]. AMEn can also be combined with the multigrid method from chapter 6 for potentially improving the coarse grid solver. In the last series of experiments the multigrid method from chapter 6 is tested versus AMEn and a combination of multigrid and AMEn. At the end of this thesis a summary of the results and developed methods is given in chapter 10, together with possible directions for further research.

Parts of the results presented in this thesis have already been published in [8, 9].

CHAPTER 2

Continuous time Markov chains

Markov chains are often referred to as "memoryless stochastic processes" and are typically used to describe systems whose state changes in time. While there are many classes of Markov chains, this thesis focuses exclusively on continuous time Markov chains (CTMCs). We therefore give a definition and an overview of important properties of CTMCs in this chapter. For a better understanding, we will present some model problems, which we will also use to test and gauge numerical methods developed in this thesis.

Our presentation in this chapter is mainly based on material from [5, 50, 66].

2.1 Basic definitions and results

Stochastic processes can be described by three sets, namely

- a state space S,
- a time space T, and
- a set of random variables $X(t) \in S, \quad t \in T$.

S and T describe the quality of the stochastic process. For example if $S \subseteq \mathbb{N}$ and $T \subseteq \mathbb{N}$, the stochastic process is discrete in space and in time. But if $T \subseteq \mathbb{R}$ is uncountable, the stochastic process is continuous in time (but still discrete in space). This case will be considered in this thesis, and we will from here on always assume $S = \{1, \ldots, N\}$ with $N \in \mathbb{N}$ and $T = [0, \infty)$.

We will now categorize continuous time Markov chains, which are particular stochastic processes with continuous time and discrete space:

Definition 2.1. The stochastic process $\{X(t) : t \in [0,\infty)\}$ on the state space $S = \{1,\dots,N\}$ is called *continuous time Markov chain* if for all $s > 0$

$$P[X(t+s) = j \mid X(u) : 0 \le u \le t] \;=\; P[X(t+s) = j \mid X(t)]. \tag{2.1}$$

The CTMC is called *homogeneous*, if for all $t \in [0,\infty)$ and $s > 0$

$$P[X(t+s) = j \mid X(t) = i] \;=\; P[X(s) = j \mid X(0) = i]. \tag{2.2}$$

Equation (2.1) is the Markov property, namely that the conditional probability of reaching a certain state only depends on the current state but not on previous states. Equation (2.2) defines a time-invariant CTMC, which is the type of CTMC considered in this thesis.

The goal is now to represent the homogeneous CTMC by a matrix. Due to the fact that T is not discrete, but continuous, we do not have the ability to describe the entry $p_{i,j}$ by the probability that a transition from state j to state i occurs, like it would be possible for a discrete time Markov chain. Instead we have to consider a time interval $[t, t+\Delta t]$. So let $p_{i,j}(\Delta t)$ be the transition probability for an observation interval of length Δt. This results in a family $P(\Delta t)$ of transition matrices, depending on the parameter Δt. To avoid the parameter dependency, we consider transition *rates* instead of probabilities.

Definition 2.2. The *transition rate* for state i and j is given by

$$a_{i,j} = \lim_{\Delta t \to 0} \frac{p_{i,j}(\Delta t)}{\Delta t}, \text{for } i \neq j \tag{2.3}$$

and

$$a_{i,i} = -\sum_{j \neq i} a_{j,i}. \tag{2.4}$$

The matrix $A = (a_{i,j}) \in \mathbb{R}^{N \times N}$ is called *transition rate matrix* or *infinitesimal generator matrix* of the CTMC.

Remark 2.3. The existence and finiteness of the limit in (2.3) can always be guaranteed and is proven in [21, Theorem II.2.5].

From Definition 2.2 we can deduce the following properties of the transition rate matrix A:

(i) $\mathbf{1}^T A = \mathbf{0}$, where $\mathbf{1}, \mathbf{0}$ are the vector of all ones and all zeros, respectively,

(ii) $a_{i,i} \leq 0$ for all i,

(iii) $a_{i,j} \geq 0$ for $i \neq j$,

(iv) A is singular (this follows from property (i)).

Although we are working with the transition rate matrix A (instead of the probability matrices $P(\Delta t)$), we are interested in probability distributions on the states of a CTMC. We will now clarify how these concepts are related.

Let $x(\Delta t) \in \mathbb{R}^N$, $\Delta t > 0$, be a probability vector, i.e., $x_i(\Delta t) = P[X(\Delta t) = i]$ for an observation interval Δt. It follows immediately that

$$0 \leq x_i(\Delta t) \leq 1 \text{ and } \sum_{i \in S} x_i(\Delta t) = 1. \tag{2.5}$$

Note that the multiplication $P(\Delta s)x(\Delta t)$, where $\Delta s > 0$, results in a probability distribution $x(\Delta s + \Delta t)$.

Particularly interesting distributions in many applications are the so-called *stationary distributions*, i.e., distributions that do not change in time. The following definition describes this kind of distribution precisely.

Definition 2.4. Let $x \in \mathbb{R}^N$ be a probability distribution on the state space S. Then x is called *stationary distribution* if for every $\Delta t > 0$

$$P(\Delta t)x = x. \tag{2.6}$$

The stationary distribution can also be computed with help of the transition rate matrix. If we consider the Taylor expansion of $p_{i,j}$ as a function of Δt with expansion point 0, it follows

$$p_{i,j}(0 + \Delta t) = p_{i,j}(0) + a_{i,j} \cdot (\Delta t - 0) + o(\Delta t).$$

Noting that $p_{i,j}(0) = 0$ gives

$$p_{i,j}(\Delta t) = a_{i,j} \cdot \Delta t + o(\Delta t). \tag{2.7}$$

Therefore, using $x(\Delta t) = P(\Delta t)x(0)$,

$$x_i(\Delta t) = x_i(0)\left(1 - \sum_{i \neq j} a_{j,i} \cdot \Delta t\right) + \left(\sum_{i \neq k} a_{i,k} \cdot x_k(0)\right)\Delta t + o(\Delta t).$$

Because of (2.4) it follows

$$x_i(\Delta t) = x_i(0) + \left(\sum_{k} a_{i,k} \cdot x_k(0)\right)\Delta t + o(\Delta t)$$

and

$$x_i'(0) = \lim_{\Delta t \to 0} \frac{x_i(\Delta t) - x_i(0)}{\Delta t} = \lim_{\Delta t \to 0} \sum_{k} a_{i,k} \cdot x_k(0) + \frac{o(\Delta t)}{\Delta t} = \sum_{k} a_{i,k} \cdot x_k(0).$$

In matrix notation we have

$$x'(0) = A \cdot x(0), \tag{2.8}$$

where we use $x'(0)$ as a symbolic expression for the vector containing the derivatives $x_i'(0)$. The values $x'(0)$ are also called the rates of change of x. From (2.8) it is obvious that if there exists a stationary distribution, then its rates of change are zero and vice versa. Summarizing we find the following result:

Lemma 2.5. *A vector $x \in \mathbb{R}^N$ is a stationary distribution of a CTMC with generator matrix A if and only if*

$$Ax = 0, \quad 0 \leq x_i \leq 1 \quad \textit{and} \quad \mathbf{1}^T x = 1. \tag{2.9}$$

So, to compute a stationary distribution of a CTMC, we have to find a non-trivial solution of a homogeneous system of linear equations. Whether such a solution of (2.9) exists and is unique depends on properties of the CTMC. One important assumption is irreducibility in the sense of the following definition.

Definition 2.6. A CTMC is *irreducible*, if there exist a $\Delta t > 0$ such that all entries of $P(\Delta t)$ are strictly positive.

So irreducibility means that for any two states i, j there is a non-zero probability that there will be a transition from i to j at some point in time.

Lemma 2.7. *If a CTMC is irreducible and has a stationary distribution x, then x is unique.*

Proof. See [50, Corollary 6.2]. □

For the existence of a solution of (2.9), irreducibility is not sufficient. We need the additional requirement that the CTMC is positive recurrent.

Definition 2.8. The hitting time of state i of a CTMC is a random variable

$$T_i = \inf\{t > 0 : X(t) = i \mid X(0) = i\}.$$

The state $i \in S$ is positive recurrent if the expectation value $E[T_i]$ is finite.

The CTMC is positive recurrent when all its states are positive recurrent.

Positive recurrence means that a state i which is visited once will be visited repeatedly. Together with irreducibility, the process will arrive at each state several times, no matter where it starts. This is the crucial property which guarantees the existence of a stationary distribution.

Lemma 2.9. *Assume that a CTMC is irreducible. Then it is positive recurrent if and only if there exists a solution to* (2.9).

Proof. See [5, Theorem 1.25]. □

Remark 2.10. From the preceding discussion it immediately follows that all entries of a stationary distribution x of an irreducible CTMC are strictly positive.

In this thesis we will concentrate on positive recurrent and irreducible CTMCs and try to compute their stationary distributions. In the following section we will introduce several model problems belonging to this class.

2.2 Model problems

In this thesis we will consider seven benchmark problems, which can be grouped into the three categories *queueing networks*, *kanban systems* and *chemical reaction*

networks. For our models we will choose the *Stochastic Automata Network (SAN)* formalism [51, 60, 61]. The reason for this is that the transition rate matrices of our CTMCs can be very large, so that storage of and computations with these matrices cannot be done in an efficient way without exploiting their structure somehow.

We will explain this formalism by considering an *overflow queueing network*, which is also our first benchmark problem. This example is taken from [53] and is a variation of a standard model problem introduced in [15, 16].

2.2.1 Overflow queueing networks

A finite number $d \in \mathbb{N}$ of queues $\mathcal{A}_i, i = 1, \ldots, d$ is given with corresponding capacities $k_i, i = 1, \ldots, d$. Each queue $\mathcal{A}_i$ describes an individual stochastic automaton in our SAN. One individual queue $\mathcal{A}_i$ is a Markov chain with $k_i + 1$ states, where state j means that currently $j - 1$ customers are waiting in the queue. So a customer can arrive at an arbitrary queue $\mathcal{A}_i$ and waits there to be served if $\mathcal{A}_i$ is not full (full means that $\mathcal{A}_i$ has reached its capacity k_i), otherwise the customer leaves the system. The arrival rate for each queue is distributed according to a Poisson process with parameter $\lambda_i, i = 1, \ldots, d$. The service time is exponentially distributed with rate $\mu_i, i = 1, \ldots, d$. By now these *local* processes in one queue $\mathcal{A}_i$ are independent from the processes in the other queues $\mathcal{A}_j, i \neq j$ and can be described by the following transition rate matrices:

$$E_i^{(i,i)} = \begin{pmatrix} -\lambda_i & \mu_i & & & 0 \\ \lambda_i & -(\lambda_i + \mu_i) & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & -(\lambda_i + \mu_i) & \mu_i \\ 0 & & & \lambda_i & -\mu_i \end{pmatrix}, i = 1, \ldots, d. \tag{2.10}$$

Note that $E_i^{(i,i)} \in \mathbb{R}^{n_i \times n_i}$ with $n_i = k_i + 1$ and that the transition rate matrix of the whole system with d independent queues is then given by

$$A_L = \sum_{i=1}^{d} A_{L_i} \tag{2.11}$$

with

$$A_{L_i} = I_1 \otimes \cdots \otimes I_{i-1} \otimes E_i^{(i,i)} \otimes I_{i+1} \otimes \cdots \otimes I_d, \tag{2.12}$$

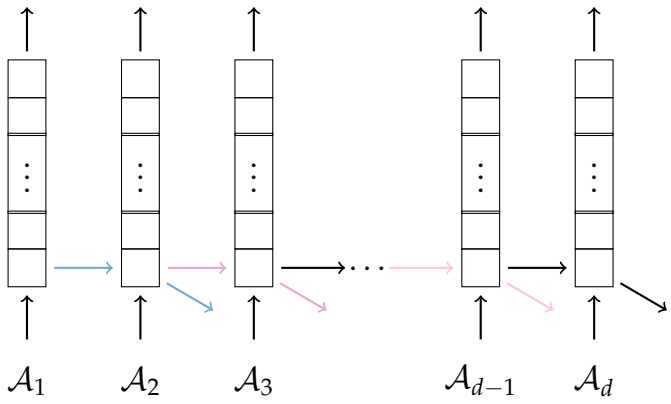


Figure 2.1: Structure of the model *overflow*.

where I_k is the identity matrix of size $n_k \times n_k$. Here we assume that the states—which can be described by d-tuples—are ordered lexicographically, which is consistent with the ordering given by the Kronecker product, see, e.g., [60, 61].

Let us now examine the situation in which the queues interact in the following way: If a customer arrives at $\mathcal{A}_i$ while $\mathcal{A}_i$ is full, the customer tries to enter the subsequent queue $\mathcal{A}_{i+1}$ instead. If this queue is also full, he leaves the system. If a customer arrives at the last queue $\mathcal{A}_d$ and it is full, he immediately leaves the system. Figure 2.1 illustrates this type of model. This interaction is a *synchronized event* in the SAN. A synchronized event is a transition in one automaton which can cause a transition in one or more other automata. These events are one of two fundamental types of interaction we consider in SANs. The other type are *functional transitions*, which we discuss in detail in the chemical reaction network problems, see section 2.2.4.

The synchronized events in which the customer enters the subsequent queue because the situation demands it, are described by the following matrices:

- The interaction is triggered if $\mathcal{A}_i$ is full and a customer arrives at $\mathcal{A}_i$, which leads to the following matrices

$$E_i^{(i,i+1)} = \begin{pmatrix} 0 & 0 & & & \\ 0 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 0 & 0 \\ & & & 0 & \lambda_i \end{pmatrix}, \; i = 1, \ldots, d-1.$$

- If $\mathcal{A}_i$ is full, $\mathcal{A}_{i+1}$ gets an additional customer, which leads to the following matrices

$$E_{i+1}^{(i,i+1)} = \begin{pmatrix} 0 & 0 & & & 0 \\ 1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 0 \\ 0 & & & 1 & 0 \end{pmatrix}, i = 1, \ldots, d-1.$$

So the interaction part of our system can be described by the matrix

$$A_S = \sum_{i=1}^{d-1} A_{S_i} \tag{2.13}$$

with

$$A_{S_i} = I_1 \otimes \cdots \otimes E_i^{(i,i+1)} \otimes E_{i+1}^{(i,i+1)} \otimes \cdots \otimes I_d. \tag{2.14}$$

Remark 2.11. Our notation for matrices appearing in an SAN description is as follows: $E_i^{(j,k)}$ describes how automaton i is influenced by a synchronized event between automaton j and automaton k. Therefore, many matrices appearing in A_{S_i} in this model are identities, as a synchronized event between $\mathcal{A}_i$ and $\mathcal{A}_{i+1}$ does not influence any other queue.

This notation also gives the reasoning for denoting the non-identity matrix describing the local transitions in (2.12) as $E_i^{(i,i)}$, as they describe events that only influence one queue.

Whereas $\boldsymbol{1}^T A_L = \mathbf{0}$, we have to add *correction terms* to A_S to arrive at this property. The correction term can be computed in the following way:

For every $A_{S_i}, i = 1, \ldots, d$ we define

$$A_{D_i} = I_1 \otimes \cdots \otimes D_i^{(i,i+1)} \otimes D_{i+1}^{(i,i+1)} \otimes \cdots \otimes I_d, \tag{2.15}$$

where $D_i^{(i,i+1)}$ and $D_{i+1}^{(i,i+1)}$ are diagonal matrices with negative diagonal entries

$$D_i^{(i,i+1)}(k,k) = -\sum_{\ell=1}^{n_i} E_i^{(i,i+1)}(\ell,k), \quad k = 1, \ldots, n_i$$

and

$$D_{i+1}^{(i,i+1)}(k,k) = -\sum_{\ell=1}^{n_i} E_{i+1}^{(i,i+1)}(\ell,k) \quad k = 1, \ldots, n_i.$$

So we can build the matrix

$$A_D = \sum_{i=1}^{d-1} A_{D_i}$$

and arrive at our transition rate matrix

$$A = A_L + A_S + A_D, \tag{2.16}$$

which describes the whole system including interactions. This separation of the three components A_L, A_S and A_D is not specific to queueing networks. We will use (2.16) also for describing all other SAN models considered in this section.

Summarizing, the SAN formalism helps us to separate the local and the interaction part of our CTMC and store them in a compact form.

Note that in general the number of summands in the local and the interaction part is independent of d, examples for this case will be shown in other model problems. So the benefit of SAN is that we can store our transition rate matrix A with complexity $\mathcal{O}((2N_{SD} + N_L) \cdot n \cdot d)$, where $n = \max_k n_k$, N_{SD} is the number of interactions and N_L is the number of local parts—instead of $\mathcal{O}(n^d)$—i.e., with complexity which is polynomial instead of exponential in d (assuming that N_{SD} and N_L depend polynomially on d, which is the case for all models we consider in this thesis).

2.2.2 Variations of the overflow queueing model

In this thesis we will look at three different kinds of overflow queueing models from [53], which differ in the interaction between the queues. The first one was described in section 2.2.1 and is illustrated in Figure 2.1 and named *overflow* in the following.

The second one is labelled *overflow_cycling*. It is analogous to the model *overflow* with the difference that if the last queue $\mathcal{A}_d$ is full and a customer arrives there, he tries to enter the first queue instead of leaving the system. In this case the matrix A_S—and therefore also the matrix A_D—in the formula of the transition rate matrix (2.16) has an additional summand, which describes the interaction between the last queue $\mathcal{A}_d$ and the first queue $\mathcal{A}_1$.

The third overflow model which we consider is named *overflow_long*. Here the customers which arrive at a full queue try to enter the subsequent queues one after the other until they find one that is not full. If they try the last queue $\mathcal{A}_d$ and detect that $\mathcal{A}_d$ is full, they leave the system. In this model the matrices A_S and A_D in (2.16) also have to be adapted. We now have interaction from $\mathcal{A}_i$ to

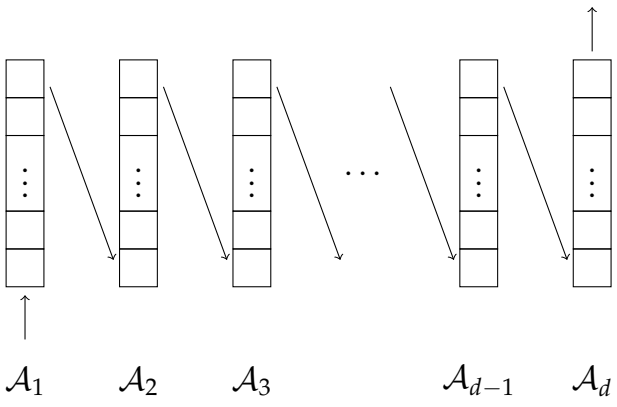


Figure 2.2: Structure of the model *simplified_kanban*.

$\mathcal{A}_j$ for all $i < j, i, j \in \{1, \ldots, d\}$, which can be described analogously to (2.14). It follows that A_S (and therefore also A_D) have $\frac{d(d-1)}{2}$ summands.

2.2.3 Kanban systems

In a kanban system, a finite number of automata d have to be passed through one after the other. These kinds of models are considered, e.g., in [14, 55]. We consider two types of kanban models here, which can also be found in the collection [53].

For visualization we think of a finite number of queues d with capacity $k_i, i = 1, \ldots, d$, which have to be passed by customers one after the other.

The first kanban model we consider is called *simplified_kanban*. In this model the customers arrive only at the first queue $\mathcal{A}_1$ and leave the system when the first queue is full or when they have passed through all queues. The service in a queue $\mathcal{A}_i$ can only be finished if the subsequent queue $\mathcal{A}_{i+1}$ is not full and the customer can thus be passed on. The arrival rate in the first queue $\mathcal{A}_1$ is (like in the overflow systems) distributed according to a Poisson process with parameter λ_1. The service time in each queue $\mathcal{A}_i$ is exponentially distributed with rate $\mu_i, i = 1, \ldots, d$.

Figure 2.2 illustrates this model. We again want to use the SAN formalism, therefore we split the local and the interaction part in this model. Note that the arrival of a customer to $\mathcal{A}_1$ is independent of the processes in the other queues. The same holds true for leaving the last queue $\mathcal{A}_d$. So local processes in this SAN only exist in the first and in the last queue and can be described by the

matrices

$$E_1^{(1,1)} = \begin{pmatrix} -\lambda_1 & & & & \\ \lambda_1 & -\lambda_1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & -\lambda_1 & \\ & & & \lambda_1 & 0 \end{pmatrix}$$

and

$$E_d^{(d,d)} = \begin{pmatrix} 0 & \mu_d & & & \\ & -\mu_d & \ddots & & \\ & & \ddots & \ddots & \\ & & & -\mu_d & \mu_d \\ & & & & -\mu_d \end{pmatrix},$$

respectively.

So the matrix A_L in (2.16) contains two summands A_{L_1} and A_{L_d} of type (2.12).

The number of interactions in this SAN is the same as for the model *overflow* described in section 2.2.1. We only have to replace the matrices $E_i^{(i,i+1)}, i = 1, \ldots, d-1$ in (2.14) (and therefore also in (2.15)) by

$$E_i^{(i,i+1)} = \begin{pmatrix} 0 & \mu_i & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & \mu_i \\ & & & & 0 \end{pmatrix}, \; i = 1, \ldots, d-1.$$

Note that $E_{i+1}^{(i,i+1)}$ in (2.14) is the same as for the model *overflow*.

The next type of kanban system is named *kanban_control*. The main difference to *kanban_simplified* is that the service process in one queue $\mathcal{A}_i$ can now be finished although the subsequent queue $\mathcal{A}_{i+1}$ is full. To visualize this, we think of a queue which at the same time acts as a waiting room with capacity k_i (think of sitting in a room where the server comes to the waiting customers, and after being served the customers can sit and wait until they can pass on to the next queue). So we now have to distinguish between customers who are in process (that means they are waiting to get served) and customers who have already

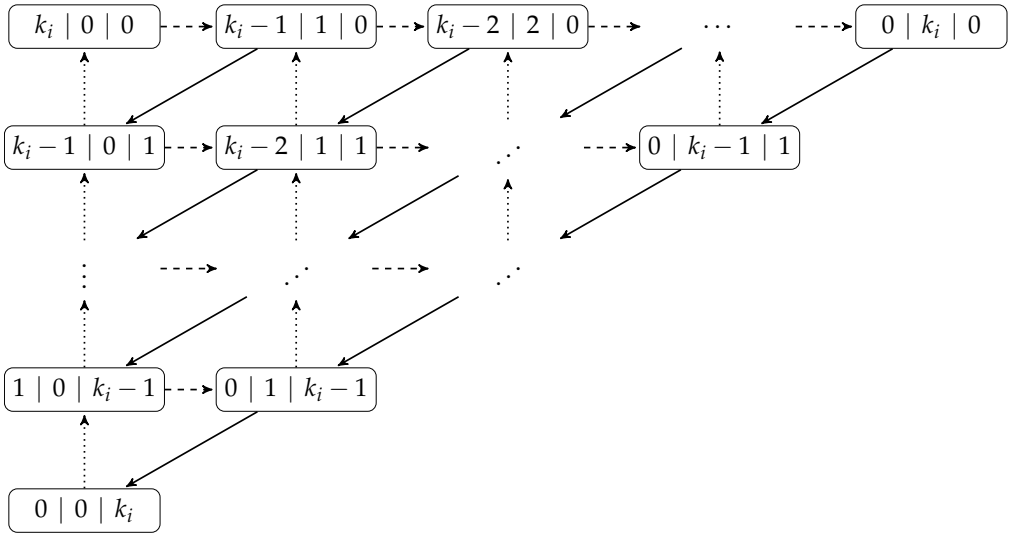


Figure 2.3: Possible transitions in the model *kanban_control* between states of a queue with k_i seats. Local transitions are depicted by solid arrows, synchronized transitions depending on the previous or subsequent queue by dashed or dotted arrows, respectively.

been processed and are waiting to get to the next queue now, which can only happen if the subsequent queue is not full.

We now want to model this CTMC as an SAN. For sake of simplicity we first do not consider the first and the last queue (although their corresponding Markov chains are easier to describe). To distinguish between the local and interaction part in our SAN, we consider the Markov chain corresponding to a queue $\mathcal{A}_i, i = 2, \ldots, d-1$. Its states can be described by three quantities:

- number of available (empty) waiting seats,
- number of customers who are in process,
- number of customers who are already processed and wait now.

Note that all three quantities always sum up to k_i. The number of states is given via the formula $\frac{(k_i+1)(k_i+2)}{2}$. In Figure 2.3 the states are ordered lexicographically along the diagonals of the grid and the different possible transitions are depicted. We can distinguish between three types of transitions: The first one is that a customer gets processed and now has to wait. This is a local process and is independent from the other queues and is marked by a solid arrow in Figure 2.3. The second one is the arrival of a new customer, which depends on the previous queue (because the customer has to pass through it). This transition is

marked by dashed arrows. The third one, depicted by dotted arrows, is that a customer leaves the queue, this is dependent on the subsequent queue (because the customer can only arrive at it if it is not full).

For the first and the last queue we only have a transition to the neighbouring queue (for the first queue it is the second one and for the last one the second to last one). In addition, the corresponding Markov chain has only $k_i + 1$ states, because we assume there are no empty seats in the first queue (whenever a customer leaves the first queue, a new customer immediately enters) and the customers served at the last queue can immediately leave.

Summarizing, in our SAN we have d local transitions and $2d - 2$ interactions.

For the matrix A_{L_i} from (2.12), the adapted matrices $E_i^{(i,i)}$ are the transposes of the adjacency matrices of the directed graph from Figure 2.3 with only the local transitions as edges, multiplied by μ_i (with negative diagonal entries to ensure column sum zero). So if we, e.g., have two seats, the corresponding matrix has the form

$$E_i^{(i,i)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\mu_i & 0 & 0 & 0 & 0 \\ 0 & \mu_i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\mu_i & 0 & 0 \\ 0 & 0 & 0 & \mu_i & -\mu_i & 0 \\ 0 & 0 & 0 & 0 & \mu_i & 0 \end{pmatrix}, \; i = 2, \ldots, d-1.$$

The matrix A_L can then be computed via (2.11) again.

For A_S from (2.16) we now have to differ between the interaction with the previous and with the next queue: This can be done for $i = 2, \ldots, d-1$ by

$$\begin{aligned} A_{S_i} \;=\; & I_1 \otimes \cdots \otimes I_{i-2} \otimes E_{i-1}^{(i,i-1)} \otimes E_i^{(i,i-1)} \otimes I_{i+1} \otimes \cdots \otimes I_d \\ + \; & I_1 \otimes \cdots \otimes I_{i-1} \otimes E_i^{(i,i+1)} \otimes E_{i+1}^{(i,i+1)} \otimes I_{i+2} \otimes \cdots \otimes I_d \end{aligned} \tag{2.17}$$

and then using the formula (2.13).

The matrices $E_i^{(i,i-1)}$ and $E_{i+1}^{(i,i+1)}$ are the transposes of the adjacency matrices of their corresponding graphs consisting of the dashed edges in Figure 2.3. Again

for two seats, we would, e.g., have

$$E_i^{(i,i-1)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \; i = 2, \ldots, d-1.$$

Analogously, the matrices $E_{i-1}^{(i,i-1)}$ and $E_i^{(i,i+1)}$ are the transposes of the adjacency matrices of their corresponding graphs consisting of the dotted edges in Figure 2.3. Continuing our example of two seats, we would, e.g., have

$$E_i^{(i,i+1)} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \; i = 2, \ldots, d-1.$$

Clearly, for queue $\mathcal{A}_1$ and $\mathcal{A}_d$ only one of the two summands in (2.17) exists.

Remark 2.12. Note that—opposed to the overflow models and the model *simplified_kanban*—the number of states in the first and last Markov chain is different from the number of states in the other Markov chains, even if all capacities $k_i \equiv k$ are identical. The size of the transition rate matrix A is then given via the formula

$$(k+1)^2 \left(\frac{(k+1)(k+2)}{2} \right)^{d-2}.$$

2.2.4 Metabolic pathways

As mentioned in section 2.2.1, we consider—besides synchronized transitions in our SAN—a different kind of transition, the so-called *functional* transitions. The rates in a functional transition are now functions, which depend either on the state of the automaton itself or on the states of the other automata. With the help of metabolic pathway systems, we will see how functional transitions may look like. We will consider two types of metabolic models, taken from [32, 52], also considered in [53]. A metabolic pathway describes chemical transformations which a substrate can go through in a specified order. We have a finite number d of substrates, which are products of these chemical transformations. Each of them has a capacity k_i, i.e., k_i is the maximum number of particles of type i

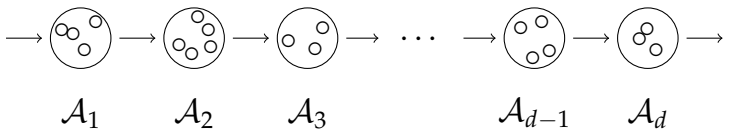


Figure 2.4: Structure of the model *directed_metab*.

which can exist in the system at the same time. These substrates represent the automata in our SAN. A transition between two automata $\mathcal{A}_i, \mathcal{A}_j$ means that the substrate i can be converted into substrate j by some chemical reaction. The rate at which the transformation of substrate i happens is given by

$$\frac{v_i \cdot m_i}{m_i + K_i - 1'}$$

where m_i is the number of particles of the ith substrate and v_i, K_i are constants. This rate is also called *flux rate*. In contrast to the overflow models in section 2.2.1 and the kanban models from section 2.2.3, this rate is not constant, but depends on the number of particles. In other words, it depends on the current state of the ith substrate and is therefore functional. Note that, additionally, the transition between two substrates is synchronized.

The first type of metabolic pathways we consider is named *directed_metab*. Figure 2.4 represents the interactions between the automata.

As for the model *simplified_kanban* in section 2.2.3, we now have processes which the particles we put in our system have to go through. So we only have two local parts—namely in the first substrate, where the particles enter the system, and in the last substrate where the particles leave the system—and $d - 1$ interactions. So the matrices A_{L_i}, A_{S_i} and A_{D_i} have the same Kronecker structure as in *simplified_kanban*, only the non-identity matrices have to be adapted. For A_{L_1} we get the matrix

$$E_1^{(1,1)} = \begin{pmatrix} -\lambda_1 & & & & \\ \lambda_1 & -\lambda_1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & -\lambda_1 & \\ & & & \lambda_1 & 0 \end{pmatrix},$$

where λ_1 is the constant influx rate. For the last substrate we have

$$E_d^{(d,d)} = \begin{pmatrix} 0 & \mu_{d_1} & & & \\ & -\mu_{d_1} & \ddots & & \\ & & \ddots & \ddots & \\ & & & -\mu_{d_{k_d-1}} & \mu_{d_{k_d}} \\ & & & & -\mu_{d_{k_d}} \end{pmatrix},$$

where

$$\mu_{i_\ell} = \frac{v_i \cdot \ell}{\ell + K_i - 1} \text{ for } i = 1, \ldots, d \text{ and } \ell = 1, \ldots, k_i.$$

As in *simplified_kanban* we only have to replace the matrices

$$E_i^{(i,i+1)}, \quad i = 1, \ldots, d-1$$

in (2.14) (and therefore also in (2.15)) by

$$E_i^{(i,i+1)} = \begin{pmatrix} 0 & \mu_{i_1} & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ddots & \mu_{i_{k_i}} \\ & & & & 0 \end{pmatrix}, \; i = 1, \ldots, d-1.$$

In this manner we have A_L, A_S and A_D and are able to compute the transition rate matrix A for *directed_metab* via formula (2.16).

The last model considered in this thesis is called *diverging_metab*. It is a variation of *directed_metab*. As Figure 2.5 shows, the second substrate in our SAN can now be transformed into two different substrates. So from this point on there are two reaction paths which are independent of each other. The computation of the transition rate matrix is analogous to the transition rate matrix in *diverging_metab*.

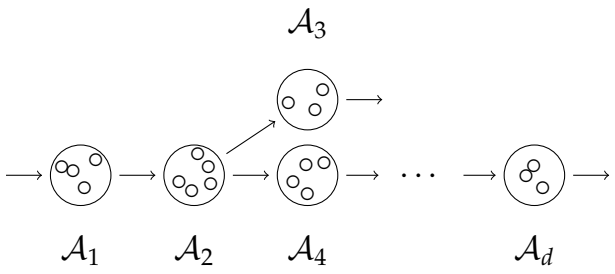


Figure 2.5: Structure of the model *diverging_metab*.

n \ d	5	6	7	8
9	59,049	531,441	4,782,969	43,046,721
17	1,419,857	24,137,569	410,338,673	6,975,757,441
33	39,135,393	1,291,467,969	42,618,442,977	1,406,408,618,241

Table 2.1: Resulting problem sizes N for various combinations of mode sizes n and dimensions d.

2.3 Curse of dimensionality

Every model in section 2.2 can be described by a matrix of the form

$$A = \sum_{(s,t)} \bigotimes_i E_i^{(s,t)} - \sum_{(s,t)\neq(i,i)} \bigotimes_i D_i^{(s,t)} \in \mathbb{R}^{\prod n_i \times \prod n_i}, \tag{2.18}$$

where $i, s, t \in \{1, \ldots, d\}$ and d is the number of Kronecker factors, also called the *dimension* of the model. The values n_i are called the *mode sizes* of the model. So if every $n_k \equiv n$ (or $n = \max_{i=1}^{d} n_i$) the matrix has a size of $N = n^d$ or ($N = \mathcal{O}(n^d)$).

Even for moderate values of d a slight increase of the mode sizes n_i leads to a huge increase of the overall system size N. To illustrate this effect we present the resulting N for different values of d and n in Table 2.1.

Thus, if we tried to solve the system (2.9) using the standard matrix representation of the transition rate matrix A, even an optimal iterative method with linear complexity $\mathcal{O}(N)$ would be infeasible already for medium-sized models (and even just storing the vector representation of the stationary distribution is not possible). To get an idea of this: a modern desktop PC with 32GB of random-access memory can store just one vector of IEEE double values of size about $4.3 \cdot 10^9$, already for a problem of size 33^7 this value is surpassed, see Table 2.1.

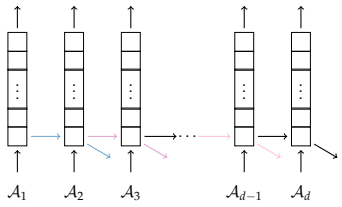
This clearly shows that one needs to use other, compressed representations (like, e.g., (2.18)) both for the transition rate matrix and for all occurring vectors, to obtain a practically usable method. This will be the topic of the next chapter.

Summary of Chapter 2:

- Homogeneous CTMCs are memoryless stochastic processes, fulfilling
$$P[X(t+s) = j | X(t) = i] \quad = \quad P[X(s) = j | X(0) = i].$$
- CTMCs can be described via a transition rate matrix A.
- We are interested in a stationary distribution, i.e., $x \in \mathbb{R}^N$ with
$$0 \leq x_i \leq 1 \text{ and } \mathbf{1}^T x = 1 \text{ and } Ax = 0.$$
- If the CTMC is irreducible and positive recurrent, there exits a unique stationary distribution.
- SAN is a modelling formalism which helps to describe A with lower storage cost.
- Storage cost in SAN form is polynomial in d (instead of exponential).
- Seven CTMCs from three categories are used as model problems:

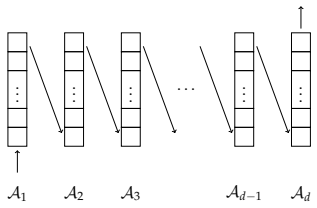
  Overflow queuing networks

  - synchronized transitions
  - one type of customers
  - customers can arrive at any queue



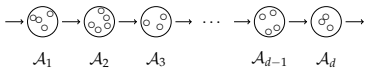
  Kanban systems

  - synchronized transitions
  - two types of customers
  - arrival only at the first queue



  Metabolic pathways

  - synchronized transitions
  - functional transition rates
  - incoming flux at the first substrate



- The transition rate matrix of all considered models has the form:
$$A = \sum_{(s,t)} \bigotimes_i E_i^{(s,t)} - \sum_{(s,t)\neq(i,i)} \bigotimes_i D_i^{(s,t)}.$$
- Models of this form suffer from the curse of dimensionality.

CHAPTER 3

Tensor Train

In section 2.3 we motivated why we have to compress our vector x and our matrix A to be able to solve the system (2.9) via iterative methods. In this thesis we work with a low-rank decomposition called *Tensor Train Decomposition* (TT-Decomposition) [57]. There also exist other tensor formats which we do not consider in this thesis, e.g., the canonical, or CP-, format [45, 46] and the hierarchical Tucker format [34, 41, 49]. For a detailed introduction into the different tensor formats and their numerical properties, we refer the reader to [39].

3.1 The TT-Decomposition

To get an idea how the TT-Decomposition works and why we benefit from it, we look at a vector $x \in \mathbb{R}^{n_1 n_2 \cdot \ldots \cdot n_d}$. The first act is to reshape this vector into a matrix X_{mat} of size $n_1 n_2 \cdot \ldots \cdot n_{\lceil d/2 \rceil} \times n_{\lceil d/2 \rceil + 1} \cdot \ldots \cdot n_d$. Instead of storing X_{mat} explicitly (as a full matrix), we can save storage if we replace it by a suitable low-rank approximation. According to the Eckart-Young theorem [31], the best rank-r approximation of a matrix can be obtained via its *singular value decomposition (SVD)*:

Theorem 3.1. *Let $A \in \mathbb{R}^{n \times m}$ be of rank R and let $1 \leq r \leq R$ and $A = U\Sigma V^T$ be a singular value decomposition of A. Then $A_r = U_r \Sigma_r V_r^T$ is the minimizer of $\|A - M\|_F$, $M \in S_r$, where $\Sigma_r = \Sigma(1\!:\!r, 1\!:\!r), U_r = U(:, 1\!:\!r), V_r = V(:, 1\!:\!r)$ and S_r is the set of real-valued matrices of rank $\leq r$.*

Proof. See [33, Theorem 2.5.3]. □

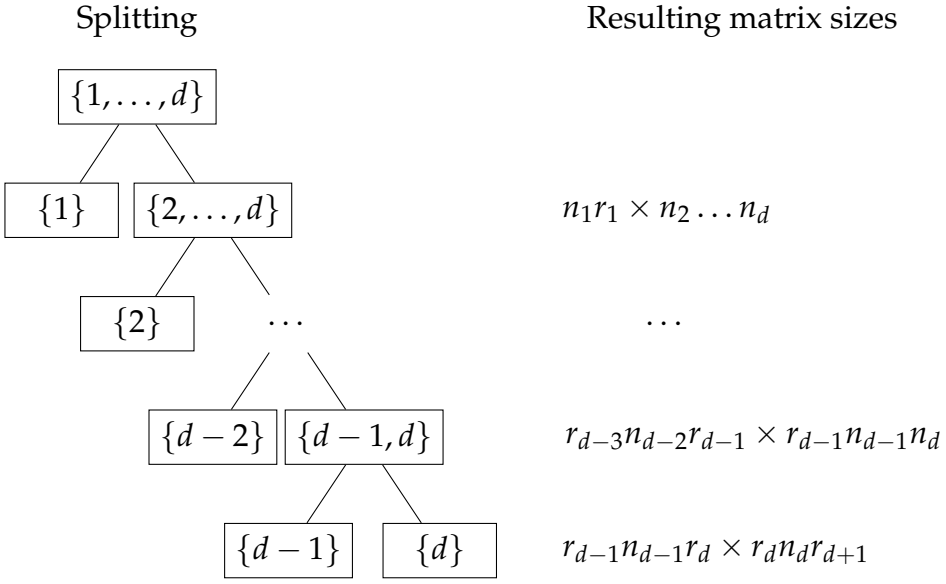


Figure 3.1: TT-Tree.

So a rank-r approximation of X_{mat} via the SVD requires storage cost of

$$\mathcal{O}(r(n_1 n_2 \cdot \ldots \cdot n_{\lceil d/2 \rceil} + n_{\lceil d/2 \rceil + 1} \cdot \ldots \cdot n_d))$$

or, if $n_k \equiv n$, of $\mathcal{O}(rn^{\lceil d/2 \rceil})$, for the vector x instead of $\mathcal{O}(n_1 n_2 \cdot \ldots \cdot n_d)$, or $\mathcal{O}(n^d)$, respectively. The computational cost for obtaining the SVD is $\mathcal{O}(N^{3/2})$ if $N = n_1 n_2 \cdot \ldots \cdot n_d$. However, this benefit is still not enough to efficiently store x if d is large, which frequently is the case in applications. Thus we compute rank-r_k SVD approximations for every dimension successively, i.e, we keep the matrix U_{r_k} after each SVD and work with the leftover matrix $\Sigma_{r_k} V_{r_k}$, where we can also do a SVD and so on. Figure 3.1 illustrates this scheme and the resulting matrix size after each decomposition. By doing so we obtain a *TT-Tensor* with *TT-ranks* r_k. In [57] this procedure is given as an algorithm called *TT-SVD*.

Before we give a formal definition of the tensor train format, we briefly motivate why tensors arise naturally in our setting. To be able to use the SVD for low-rank approximation in the preceding discussion, we reshaped the vector x into a matrix. For $d > 2$, however, it is more natural to reshape x into a tensor, i.e., a d-way array $\mathcal{X}$ of size $n_1 \times n_2 \times \cdots \times n_d$, instead, so that

$$\mathcal{X}(i_1, \ldots, i_d) = x(i_1 + (i_2 - 1)n_1 + (i_3 - 1)n_1 n_2 + \cdots + (i_d - 1)n_1 n_2 \cdot \ldots \cdot n_{d-1})$$

with $1 \leq i_k \leq n_k, k = 1, \cdots, d$. This corresponds to the MATLAB command

```
n=[n_1,n_2,...,n_d]; X=reshape(x,n);
```

There are different ways how to generalize the concept of low-rank approximation from matrices to tensors. One such concept is the TT-Decomposition:

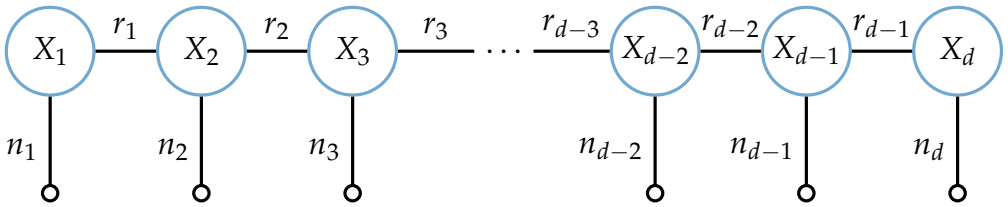


Figure 3.2: TT-Decomposition.

Definition 3.2. A tensor $\mathcal{X}$ is given in TT-format with TT-ranks $r_k, k = 1, \ldots, d$ if each entry is given by

$$\mathcal{X}(i_1, \ldots, i_d) = X_1(i_1) \cdot X_2(i_2) \cdot \ldots \cdot X_d(i_d) \tag{3.1}$$

with parameter-dependent matrices $X_k(i_k) \in \mathbb{R}^{r_{k-1} \times r_k}$ for $k = 1, \ldots, d$ (and $r_0 = r_d = 1$).

The parameter-dependent matrices $X_k(i_k)$ can also be thought of as $r_{k-1} \times n_k \times r_k$ tensors and are called *TT-cores*. These tensors are formed from the matrices U_{r_k} of the SVDs from the scheme illustrated in Figure 3.1. If a tensor $\mathcal{X}$ is given in the format (3.1), we arrive at a storage complexity $\mathcal{O}((d-2)nr^2 + 2nr)$, where $n = \max_{k=1}^{d} n_k$ and $r = \max_{k=1}^{d} r_k$.

Figure 3.2 illustrates how the cores are linked among each other through the ranks. The "free" edges determine the mode sizes of the corresponding tensor. As mentioned at the beginning of this chapter, we also want to compress the transition rate matrix $A \in \mathbb{R}^{n_1 n_2 \cdot \ldots \cdot n_d \times n_1 n_2 \cdot \ldots \cdot n_d}$. Analogously to the format (3.1) we can define a TT-Decomposition for A. Note that in this thesis we only work with square matrices so we only define the TT-Decomposition for those, but it can also be defined for general rectangular matrices.

Definition 3.3. A matrix A corresponds to a *TT-Matrix* $\mathcal{A}$, if the entries of $\mathcal{A}$ are given by

$$\mathcal{A}(i_1, i_2, \ldots, i_d; j_1, \ldots, j_d) = A_1(i_1, j_1) \cdot A_2(i_2, j_2) \cdot \ldots \cdot A_d(i_d, j_d), \tag{3.2}$$

where $A_k(i_k, j_k)$ is a $r_{A_{k-1}} \times r_{A_k}$ matrix and $1 \leq i_k, j_k \leq n_k$.

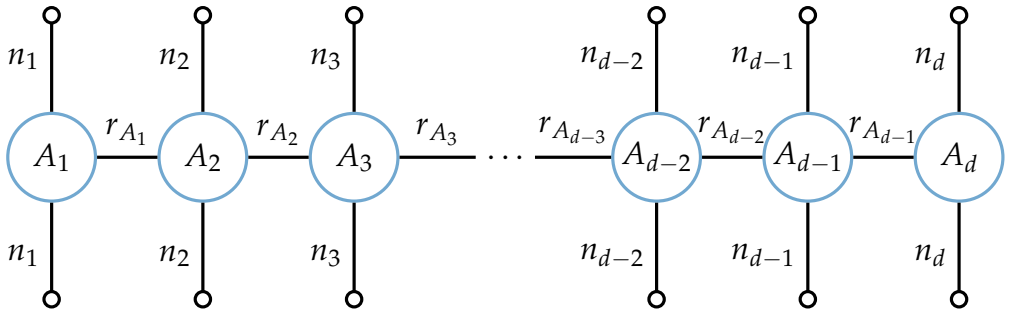


Figure 3.3: TT-Matrix-Decomposition.

Apparently from Definition 3.3 the parameter-dependent matrices (again called cores) $A_k(i_k, j_k)$ are dependent on two parameters i_k and j_k instead of one as it is the case for the matrices $X_k(i_k)$ in (3.1). Therefore, the parameter-dependent matrices $A_k(i_k, j_k)$ can be understood as $r_{A_{k-1}} \times n_k \times n_k \times r_{A_k}$ tensors, similarly to before. Note that if all ranks of a TT-matrix $\mathcal{A}$ are equal to 1, then the corresponding matrix A is given by $A = A_1 \otimes \cdots \otimes A_d$. Figure 3.3 shows the connection between the cores of a TT-Matrix.

3.2 Basic operations in TT-Format

In this section, we consider basic operations with TT-Tensors and TT-Matrices. We only briefly discuss the effect of the operations on the cores and on the ranks. Additionally, we will also list the computational costs of these operations. We start with the multiplication by a scalar $\alpha \in \mathbb{R}$. Let $\mathcal{X}$ be a TT-Tensor with cores $X_1, \ldots, X_d$ and $\alpha \in \mathbb{R}$ a scalar. Then

$$\alpha\mathcal{X}(i_1, \ldots, i_d) = (\alpha X_1(i_1)) \cdot X_2(i_2) \cdot \ldots \cdot X_d(i_d), \tag{3.3}$$

i.e., one just has to multiply one of the cores by α. Note that this operation does not have an effect on the ranks and has cost $\mathcal{O}(n_1 r_1)$. Figure 3.4 illustrates this operation.

Next, we consider the addition of two tensors $\mathcal{X}$ and $\widehat{\mathcal{X}}$. Let $\mathcal{X}, \widehat{\mathcal{X}}$ be TT-Tensors with cores $X_1, \ldots, X_d$ and $\widehat{X}_1, \ldots, \widehat{X}_d$, respectively and let $\mathcal{Y} = \mathcal{X} + \widehat{\mathcal{X}}$. Then the

cores Y_k of $\mathcal{Y}$ are given by

$$\begin{aligned} Y_1(i_1) &= \begin{pmatrix} X_1(i_1) & \widehat{X}_1(i_1) \end{pmatrix}, \\ Y_k(i_k) &= \begin{pmatrix} X_k(i_k) & 0 \\ 0 & \widehat{X}_k(i_k) \end{pmatrix}, \quad k = 2, \dots, d-1 \\ Y_d(i_d) &= \begin{pmatrix} X_d(i_d) \\ \widehat{X}_d(i_d) \end{pmatrix}. \end{aligned} \tag{3.4}$$

This operation is illustrated via Figure 3.5. Two aspects should be mentioned here. The first one is that the addition does not need any arithmetic operations. The second one is that the ranks of the result are the sums of the ranks of the summands. That this increase of the ranks is not always necessary but an effect of the specific realization (3.4) can easily be seen by adding $\mathcal{X} + \mathcal{X}$ which is equal to $2\mathcal{X}$. Multiplying a tensor by a scalar does not change the ranks, as we observed in (3.3), so the ranks of the result are unnecessarily doubled here.

The inner product of two TT-Tensors $\mathcal{X}, \widehat{\mathcal{X}}$ with ranks bounded by $r_{\mathcal{X}}, r_{\widehat{\mathcal{X}}}$, respectively, can be efficiently computed via multidimensional contraction [57, Section 4.2] in $\mathcal{O}(nd \max\{r_{\mathcal{X}}, r_{\widehat{\mathcal{X}}}\}^3)$ arithmetic operations. We do not go into details concerning the specific implementation of this operation here. As the result of this operation is just a scalar and the operands are not changed, there is no effect on the ranks. For sake of completeness we also give a schematic illustration of the result in Figure 3.6 (note that there are no free edges any more).

The last operation we consider is multiplying a TT-Matrix with a TT-Tensor. This can be done in the following way: Let $\mathcal{A}$ be a TT-Matrix and $\mathcal{X}$ a TT-Tensor, then the entries of $\mathcal{Y} = \mathcal{A}\mathcal{X}$ are given by:

$$\mathcal{Y}(i_1, \dots, i_d) = Y_1(i_1) \cdot \ldots \cdot Y_d(i_d),$$

where

$$Y_k(i_k) = \sum_{j_k} (A_k(i_k, j_k) \otimes X_k(j_k)), \tag{3.5}$$

see [57, Section 4.3]. The ranks $r_{\mathcal{Y}}$ of $\mathcal{Y}$ are bounded by $r_A r_{\mathcal{X}}$, where $r_A = \max r_{A_k}$ and $r_{\mathcal{X}} = \max r_k$. A schematic illustration of this operation is given in Figure 3.7. The arithmetic cost is $\mathcal{O}(dn^2 r_A^2 r_{\mathcal{X}}^2)$.

3.3 Existence and quality of a TT-Decomposition

For a statement about the existence of a TT-Decomposition we need the following definition:

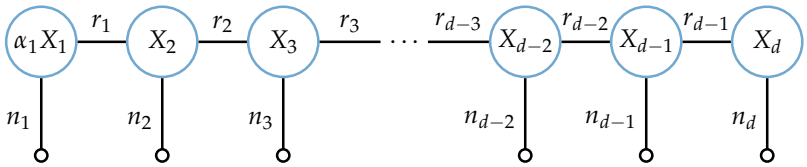


Figure 3.4: Scalar multiplication in TT-format.

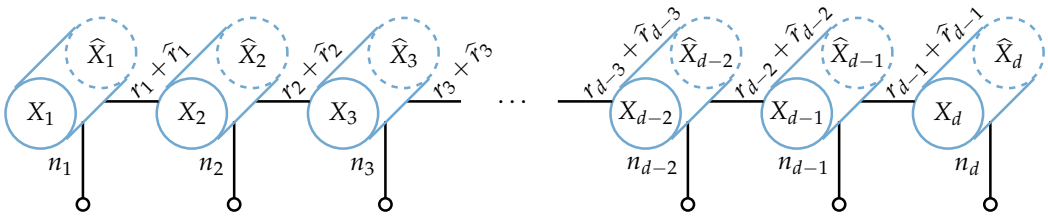


Figure 3.5: Addition in TT-format.

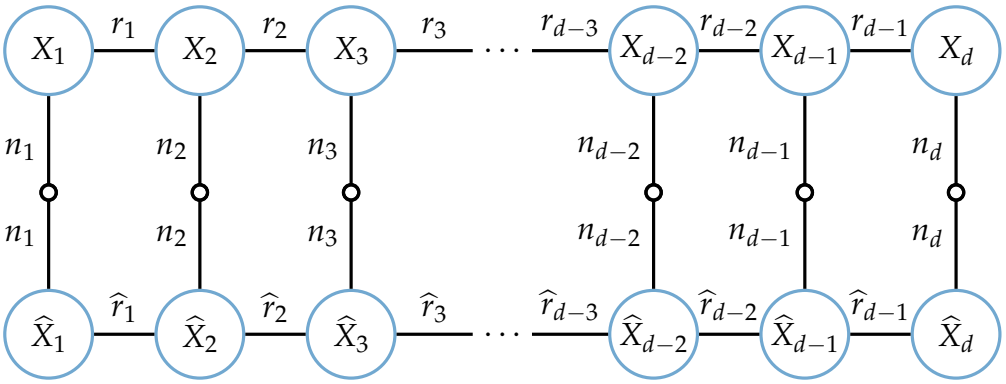


Figure 3.6: Inner product in TT-format.

Definition 3.4. The *unfolding* of a tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ with respect to the dimension indices $t \subseteq \{1, \ldots, d\}$ is a matrix $X_{(t)} \in \mathbb{R}^{\widetilde{n}_t \times \widetilde{n}_s}$ with $\widetilde{n}_t = \prod_{k \in t} n_k$ and $\widetilde{n}_s = \prod_{k \notin t} n_k$. The rows of $X_{(t)}$ correspond to the indices from t and the columns to the indices from $\{1, \ldots, d\} \setminus t$.

For illustration we consider the important special case that $t = \{1, \ldots, \ell\}$: In this case we have

$$\mathcal{X}(i_1, \ldots, i_d) = X_{(t)}(i, j)$$

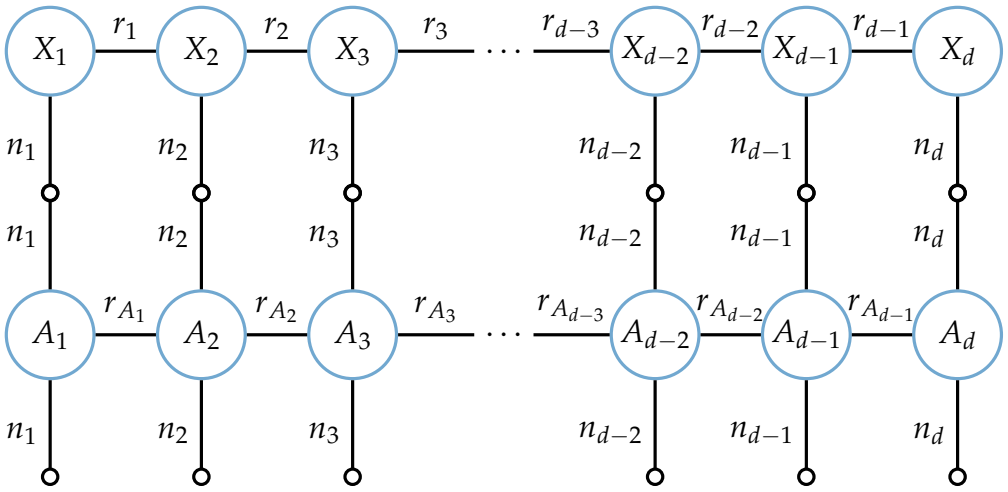


Figure 3.7: *TT*-Matrix-Vector-Multiplication.

with

$$i = i_1 + (i_2 - 1)n_1 + (i_3 - 1)n_1n_2 + \cdots + (i_\ell - 1)n_1n_2 \cdot \ldots \cdot n_{\ell-1})$$

and

$$j = i_{\ell+1} + (i_{\ell+2} - 1)n_{\ell+1} + \cdots + (i_d - 1)n_{\ell+1}n_{\ell+2} \cdot \ldots \cdot n_{d-1})$$

where $1 \leq i_k \leq n_k, k = 1, \ldots, d$. In MATLAB notation we have

```
X_t=reshape(X,prod(n(1:l)), prod(n(l+1:d)));
```

Theorem 3.5. *Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ and $r_k = \operatorname{rank}(X_{(\{1,\ldots,k\})})$ for $k = 1, \ldots, d-1$. Then there exists a TT-Decomposition of $\mathcal{X}$ with TT-ranks r_k.*

Proof. See [57, Theorem 2.1]. □

In most cases there is the need of an approximation $\widetilde{\mathcal{X}}$ of $\mathcal{X}$ with smaller ranks, because the ranks from Theorem 3.5 can be very large, as the unfolding matrices are huge. Besides, when using the *TT*-format in the presence of round-off error, it is not necessary or reasonable to work with the "exact" rank. Instead we use the numerical rank, i.e., the number of singular values above some specified tolerance. The following lemma shows that the error of this approximation is bounded by a term which includes only the "chopped part" of $\mathcal{X}$.

Lemma 3.6. *Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ be a tensor. Then there exists an approximation $\widetilde{\mathcal{X}}$ of $\mathcal{X}$ with TT-ranks r_k so that*

$$\|\mathcal{X} - \widetilde{\mathcal{X}}\|_F \leq \sqrt{\sum_{k=1}^{d-1} \sum_{\ell=r_k+1}^{N_k} \left(\sigma_\ell^{\{1,\dots,k\}}\right)^2},$$

where $\sigma_\ell^{\{1,\dots,k\}}$ *is the* ℓ*th singular value of* $X_{(\{1,\dots,k\})}$ *and*

$$N_k = \min\{n_1 \cdot \ldots \cdot n_k, n_{k+1} \cdot \ldots \cdot n_d\}.$$

Proof. See [57, Theorem 2.2] together with the fact that the error of the best rank-r_k approximation of $X_{(\{1,\dots,k\})}$ in the Frobenius norm is

$$\sqrt{\sum_{\ell=r_k+1}^{N_k} \left(\sigma_\ell^{\{1,\dots,k\}}\right)^2}.$$

□

Remark 3.7. Assume that one wants to approximate a TT-Tensor with an error smaller than *tol* and let $\delta = \frac{tol}{\sqrt{d-1}}$. If one defines

$$r_k = \max\left\{r^* : \sqrt{\sum_{\ell=r^*}^{N_k} \left(\sigma_\ell^{\{1,\dots,k\}}\right)^2} > \delta\right\}, \tag{3.6}$$

then Lemma 3.6 guarantees that there exists a TT-Tensor with ranks r_k which approximates $\mathcal{X}$ with accuracy at least *tol*.

The next section will give a method by which an approximation of a given TT-Tensor $\mathcal{X}$ with prescribed ranks or accuracy can be obtained. The proof of Lemma 3.6 in [57] is constructive, i.e., the tensor $\widetilde{\mathcal{X}}$ which yields the given error bound is explicitly specified. This construction is termed TT-SVD. It does not tell whether there exists a better approximation than $\widetilde{\mathcal{X}}$ with the same TT-rank constraints. By now we also do not know whether for prescribed ranks r_k there exists a best approximation at all. The following corollary answers this:

Corollary 3.8. *Given a tensor* $\mathcal{X}$ *and prescribed ranks* r_k*, there exists a best approximation* $\mathcal{X}_{best}$ *to* $\mathcal{X}$ *in the Frobenius norm with TT-ranks bounded by* r_k*. In addition, TT-SVD computes a quasi-optimal TT-Tensor* $\mathcal{X}_{approx}$ *in the sense that*

$$\|\mathcal{X} - \mathcal{X}_{approx}\|_F \leq \sqrt{d-1}\|\mathcal{X} - \mathcal{X}_{best}\|_F. \tag{3.7}$$

Proof. See [57, Corollary 2.4]. □

3.4 Truncation

As discussed in section 3.2 and 3.3, it is sometimes necessary to adapt the rank of a given TT-Tensor. This may happen either because linear algebra operations unnecessarily increased the rank, because the rank is higher than required for obtaining the desired accuracy or for bounding complexity.

Let $\mathcal{X}$ be a TT-Tensor with cores $X_k(i_k) \in \mathbb{R}^{r_{k-1} \times r_k}$. We want to replace these cores by cores with size $\widetilde{r}_k < r_k$, such that the approximation error is smaller than some tolerance *tol*. This process is called *truncation* (or *rounding*).

For sake of simplicity let us assume that we want to reduce r_1, i.e., we want to replace the first core $X_1 \in \mathbb{R}^{1 \times n_1 \times r_1}$, and the second core $X_2 \in \mathbb{R}^{r_1 \times n_2 \times r_2}$. Because of Lemma 3.6 it is useful to replace X_1, X_2 using singular vectors corresponding to large singular values of the first unfolding $X_{(\{1\})}$. For high-dimensional problems we are not able to form the matrix $X_{(\{1\})}$, let alone compute a (reduced) SVD of it.

From section 3.1 we know that the cores of $\mathcal{X}$ implicitly define a decomposition

$$X_{(\{1\})} = UV^T$$

where U corresponds to X_1 and V to the rest of the cores. With the help of the following lemma we can describe a reduced SVD of $X_{(\{1\})}$ without forming $X_{(\{1\})}$ explicitly. Instead we will use $\mathcal{X}$.

Lemma 3.9. *If $\mathcal{Z}$ is a tensor where each entry is a vector with running index α_1, i.e.,*

$$\mathcal{Z}(\alpha_1, i_2, \dots, i_d) = Q_2(i_2) \dots Q_d(i_d)$$

with $Q_k(i_k) \in \mathbb{R}^{r_{k-1} \times r_k}$ and the matrices $Q_k(i_k)$ satisfy

$$\sum_{i_k} Q_k(i_k) Q_k(i_k)^T = I_{r_{k-1}}, \tag{3.8}$$

then the rows of $\mathcal{Z}$, reshaped into a matrix of size $r_1 \times \prod_{k=2}^{d} n_k$ (with row indices corresponding to α_1), are orthonormal.

Proof. See [57, Lemma 3.1]. □

Lemma 3.9 states that we can obtain an orthonormal matrix by orthogonalizing the individual cores. This allows to compute a QR decomposition of V through a series of small QR decompositions of $X_d, \dots, X_2$, see [57, Section 3] for details. Given the QR decompositions $V = Q_v R_v$ and $U = Q_u R_u$ (which is easily

computable), we can then compute the leading $\widehat{r}_1 < r_1$ left and right singular vectors of $\widetilde{X}_{(\{1\})}$ as $Q_u\widehat{U}$, $Q_v\widehat{V}$, where $\widehat{U}$, $\widehat{V}$ are the left and right singular vectors of $R_u R_v^T$. These matrices now give us a lower-rank approximation of $X_{(\{1\})}$ as

$$X_{(\{1\})} \approx (Q_u\widehat{U})(Q_v\widehat{V})^T. \tag{3.9}$$

This can be translated into TT-cores as follows:

- X_1 is replaced by $Q_u\widehat{U}$,
- X_2 is replaced by $\widehat{V}^T X_2$,
- $X_3, \ldots, X_d$ are unchanged.

The next step is to reduce r_2. This can be done in the same way as it was done for r_1. We only have to orthogonalize the first and the second core before because they were changed in the first step of the method, the other cores are unchanged and still fulfil the orthogonality condition (3.8). The overall costs are $\mathcal{O}(dnr^3)$, where $n = \max n_k$ and $r = \max r_k$.

3.5 TT-representation of stationary distributions

In the following we want to show which TT-ranks are necessary for accurately representing the solutions of the models from section 2.2, and how these solutions and their ranks differ between the models. For this purpose we consider the models *overflow*, *simplified_kanban* and *directed_metab*, i.e., one of each category. For a better understanding we consider $d = 3$ automata with mode sizes $n = 33$ and different choices of the parameters, and show the components of the stationary distribution for the corresponding parameter choice on each automaton. In addition we investigate fixed rank approximations of the solution by showing the value of $\|A\mathcal{X}_R\|_2 / \|A\mathbf{1}\|_2$, where $\mathcal{X}_R$ is an approximation of the steady state distribution $\mathcal{X}$ with all TT-ranks bounded by R and $\mathbf{1}$ is the normalized vector of all ones (represented as a TT-Tensor). We use this value because this will be one of the stopping criteria for all methods later on, see chapters 4 and 7–9. The results can be seen in Figures 3.8–3.10. The solution is plotted on the right-hand side. We have three axes (one for each automaton) and each axis describes the capacity. The value of the solution is given by the colouring of its corresponding state. Note that only values which are at least ten percent as large as the maximal value of the steady state solution are shown, and that we scaled the solution such that the largest entry is equal to one. On the left-hand side the

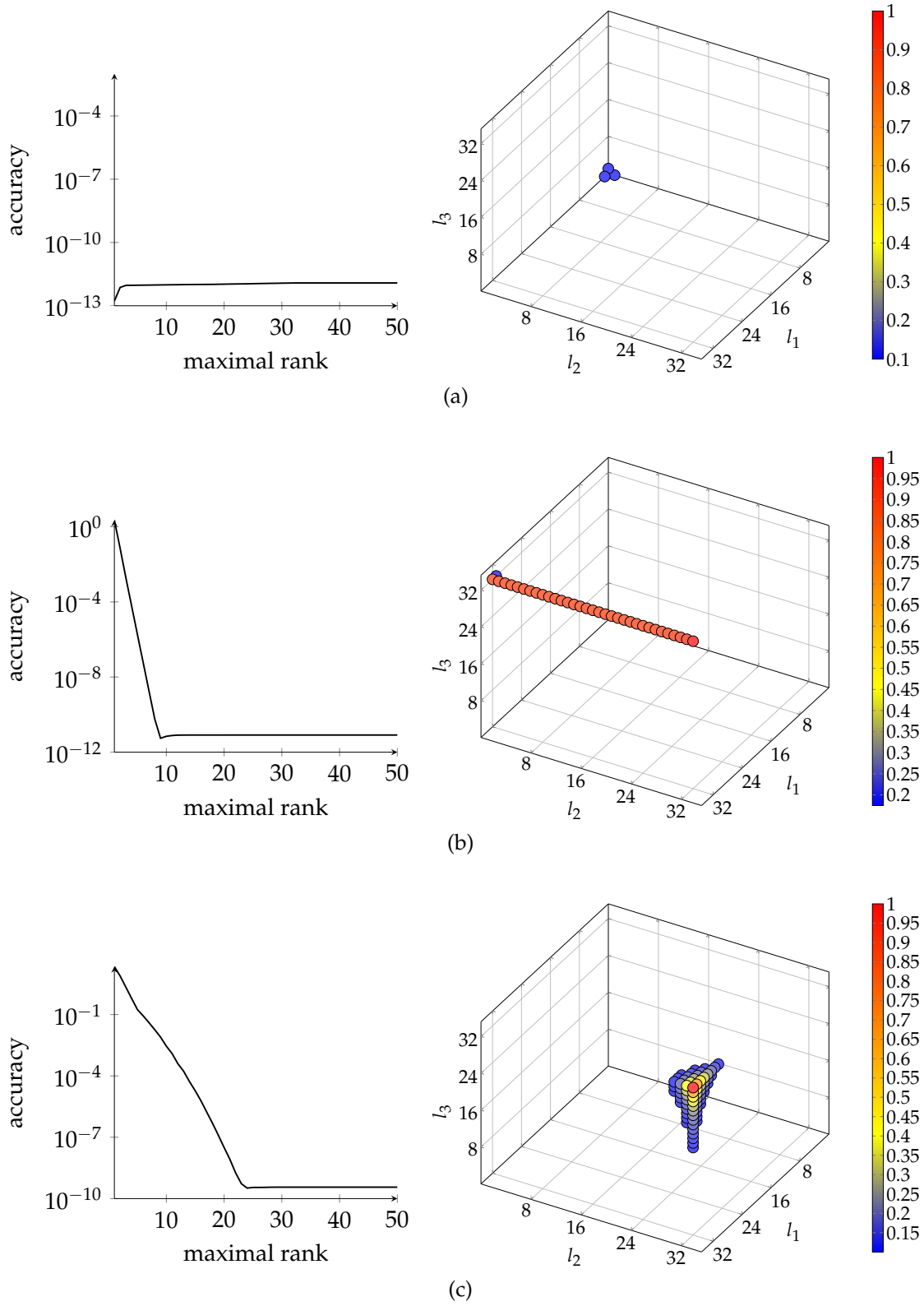


Figure 3.8: Accuracy $\|A\mathcal{X}_R\|_2/\|A\mathbf{1}\|_2$ of rank-R approximation (left) and solution (right) for model *overflow* for parameters (a) $\lambda = [0.1, 0.1, 0.1]$ and $\mu = [1, 1, 1]$, (b) $\lambda = [1, 0.1, 1]$ and $\mu = [0.1, 1, 0.1]$, (c) $\lambda = [1.2, 1.1, 1]$ and $\mu = [1, 1, 1]$.

values $\|A\mathcal{X}_R\|_2/\|A\mathbf{1}\|_2$ for different rank-R approximations are shown. Note that in some of the cases a slight decrease of accuracy can be observed for increasing rank. While this is most likely a numerical effect due to finite precision arithmetic, we stress that a monotonic decrease cannot be guaranteed because we measure the value $\|A\mathcal{X}_R\|_2/\|A\mathbf{1}\|_2$ and not $\|\mathcal{X}-\mathcal{X}_R\|_2$. Let us first consider the results for the model *overflow* shown in Figure 3.8. The first case, shown in Figure 3.8(a), is that the customers arrive slowly and are served rapidly. As an obvious consequence, these queues are empty and we need only a maximal TT-rank of 1 to reach a small value $\|A\mathcal{X}_R\|_2/\|A\mathbf{1}\|_2$. The second case, see Figure 3.8(b), is chosen such that the first and the third queue serve slowly and the second one rapidly. The arrival of customers to each queue is chosen reciprocally. The slow serving of the first and third queue causes their capacities to be exhausted. The distribution of the second queue shows that each scenario is possible. Because of the slow serving of the first queue, customers who arrive to the first queue are passed on to the second one. This results in a situation that customers arrive at the second queue just as they will be served by it. In contrast to the first example, the needed maximal rank is larger, namely to get a factor of 10^{-12} we need a maximal rank of 10 instead of 1. In the third example, depicted in Figure 3.8(c), we have the case that the service rate is equal in all queues and the arrival and service rates do not differ much. Customers will arrive more frequently than they are served. Therefore it is more likely that the capacities of all queues are exhausted. However, the solution shows that there is also a tendency that the capacities are not reached. This tendency is stronger in the third queue than in the other ones. The maximal TT-rank to get the value $\|A\mathcal{X}_R\|_2/\|A\mathbf{1}\|_2$ small is larger than in the first two examples. These three examples show that the more states have a probability close to the maximum one in the solution, the higher a maximal rank is needed to arrive at a small value of $\|A\mathcal{X}_R\|_2/\|A\mathbf{1}\|_2$. Note that $\|A\mathbf{1}\|_2$ is constant in the sense that it is independent of the rank R, so we need a small value of $\|A\mathcal{X}_R\|_2$ to arrive at a small value of $\|A\mathcal{X}_R\|_2/\|A\mathbf{1}\|_2$.

Let us now consider the results for *simplified_kanban* presented in Figure 3.9. The first case shown in Figure 3.9(a) is analogous to the first case in the model *overflow*, see Figure 3.8(a). In the second case we have the situation that λ_1 and the service rate in each queue are chosen to be equal. This results in two most likely scenarios, namely that the capacity of the first and the second queue are both exhausted or that only the capacity of the first one is exhausted. It is surprising that the possibility that all queues are full does not belong to the most likely scenarios. This is due to the fact that the customers have to pass the first queue to get to the second one and the second one to get to the third one. Notice that just as they arrive, they will be served. The more customers are in the second queue, the less customers will be in the first one. In the same way, the more customers are in the third queue, the less customers are in the first and second queue. The steady state solution has a complex structure with a lot more states which have

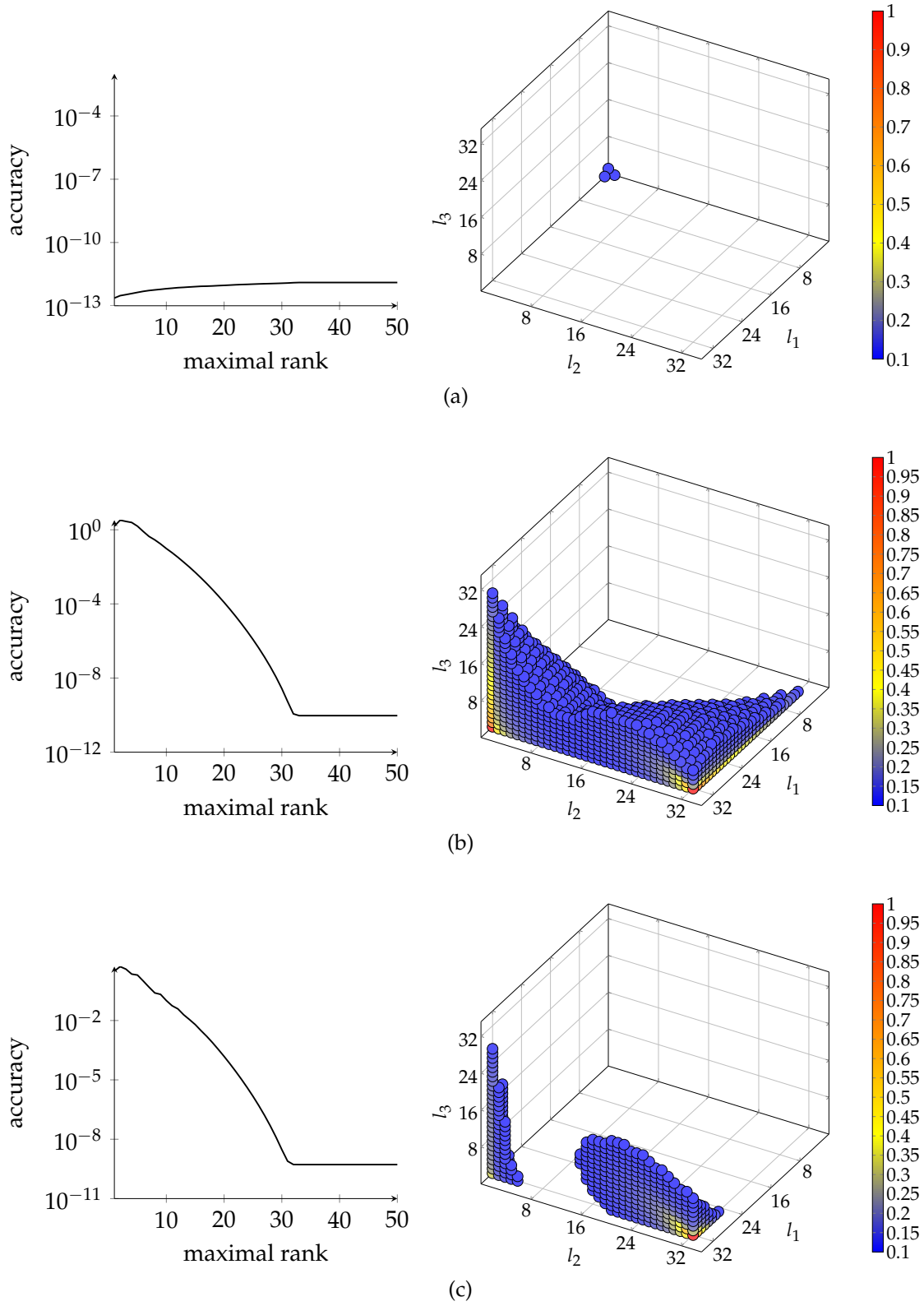


Figure 3.9: Accuracy $\|A\mathcal{X}_R\|_2/\|A\mathbf{1}\|_2$ of rank-R approximation (left) and solution (right) for model *simplified_kanban* for parameters (a) $\lambda_1 = 0.1$ and $\mu = [1,1,1]$, (b) $\lambda_1 = 0.9$ and $\mu = [0.9,0.9,0.9]$, (c) $\lambda_1 = 1.2$ and $\mu = [1,1,1]$.

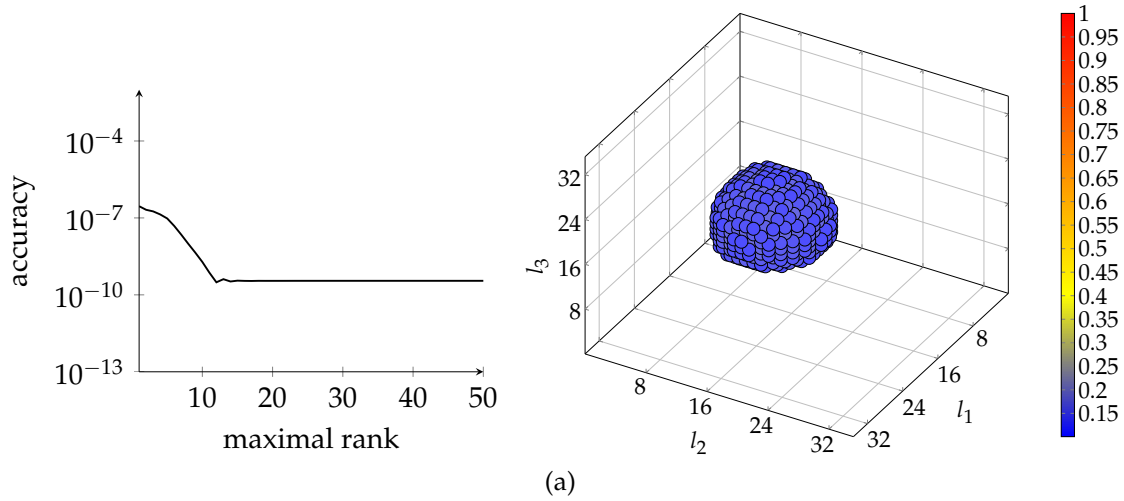
model	parameters
overflow	$\lambda_i = 1.2 - 0.1(i-1), \quad \mu_i = 1, \quad i = 1, \ldots, d$
overflow_cycling	$\lambda_i = 1.2 - 0.1(i-1), \quad \mu_i = 1, \quad i = 1, \ldots, d$
overflow_long	$\lambda_i = 1.2 - 0.1(i-1), \quad \mu_i = 1, \quad i = 1, \ldots, d$
simplified_kanban	$\lambda_1 = 1.2, \quad \mu_i = 1, \quad i = 1, \ldots, d$
kanban_control	$\lambda_i = 1.2, \quad \mu_i = 1, \quad i = 1, \ldots, d$
directed_metab	$\lambda_1 = 0.01, \quad v_i = 0.1, \quad K_i = 1000, \quad i = 1, \ldots, d$
diverging_metab	$\lambda_1 = 0.01, \quad v_i = 0.1, \quad K_i = 1000, \quad i = 1, \ldots, d$

Table 3.1: Parameters used in the numerical tests.

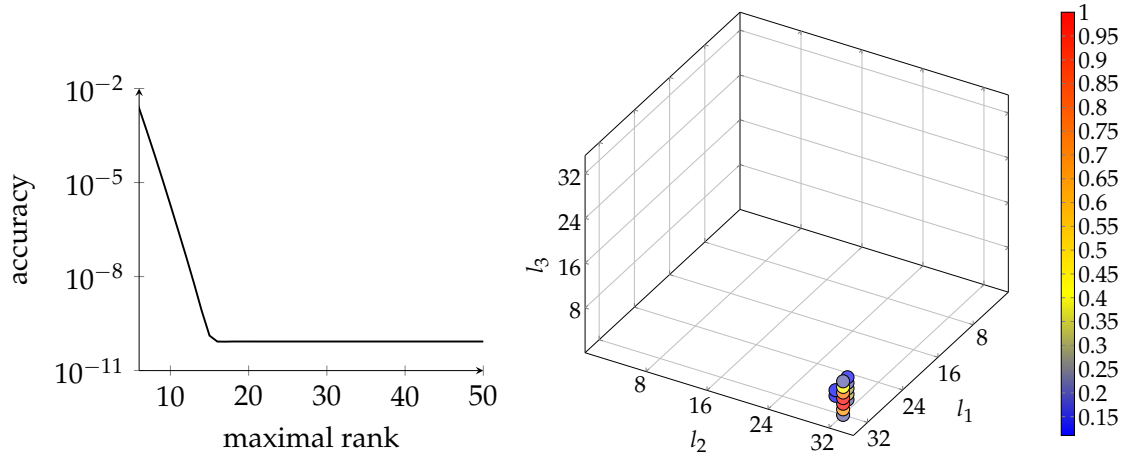
a non-negligible probability. For the third example, shown in Figure 3.9(c), the arrival rate is larger than the service rate. Therefore, in contrast to the previous example, the first queue has a larger tendency to be full, and the steady state solution is more concentrated at states corresponding to more customers. The required rank however is the same as in the example shown in Figure 3.9(b).

The results for the model *directed_metab* are depicted in Figure 3.10. We again consider three different cases, in which we vary the constants v_i which control the reaction rate, while using the same influx rate λ_1 in all cases. In the first example, see Figure 3.10(a), all v_i are equal to one, which means that all substrates are transformed rapidly. Therefore, in the steady state distribution, the states corresponding to (almost) full automata have a very low probability. All substrates behave very similarly and the resulting distribution is representable with rather low rank. The second case (see Figure 3.10(b)) corresponds to a situation where the second substrate is converted very infrequently. So the steady state solution is concentrated at the states corresponding to the first and second automaton being full, while the third one is almost empty. The needed maximal rank is not as low as one would expect. This is due to the fact, that the values of the stationary distribution are of highly different magnitude in a very small neighbourhood. In the third example the constants v_i are changed so that all reactions happen infrequently. Therefore the capacity of all substrates is almost exhausted. The maximal rank needed to get the value $\|A\mathcal{X}_R\|_2 / \|A\mathbf{1}\|_2$ small is about 15.

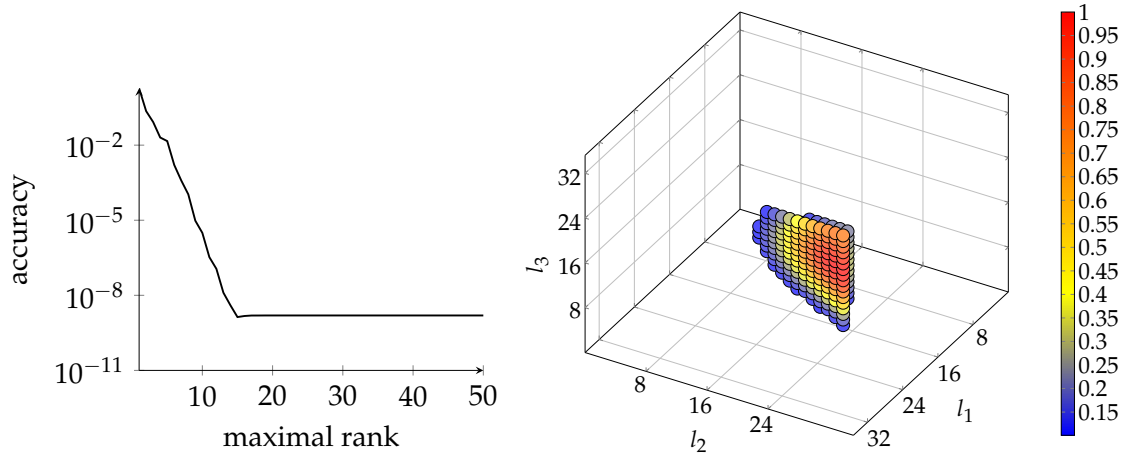
So, all the examples show that the more complex the structure of the solution is (this can, e.g., mean that the behaviour of each automaton is different from the others or that neighbouring states have highly different probabilities) the higher the needed maximal TT-rank to arrive at a small value of $\|A\mathcal{X}_R\|_2 / \|A\mathbf{1}\|_2$ and therefore to a small value of $\|A\mathcal{X}_R\|_2$. Besides this, these examples show that different parameters can lead to highly different solutions. It is also worth mentioning that from a certain rank on, a further rank increase does not improve



(a)



(b)



(c)

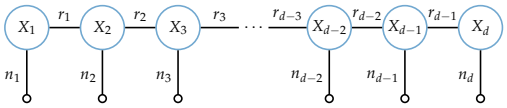
Figure 3.10: Accuracy $\|A\mathcal{X}_R\|_2\|A\mathbf{1}\|_2$ of rank-R approximation (left) and solution (right) for model *directed_metab* for parameters (a) $v = [1,1,1], K = [1000,1000,1000]$ and $\lambda_1 = 0.01$, (b) $v = [0.9,0.1,0.9], K = [1000,1000,1000]$ and $\lambda_1 = 0.01$, (c) $v = [0.1,0.1,0.1], K = [1000,1000,1000]$ and $\lambda_1 = 0.01$.

the result any further. This is due to the fact, that better results are not possible using the MATLAB machine precision as the entries of $\mathcal{X}$ become very small. This effect gets worse the larger the problem size is. For our numerical tests in chapters 4 and 7–9 we choose the parameters as they are recommended in the benchmark collection [53]. These parameters are presented in Table 3.1.

Summary of Chapter 3:

- A *TT-Tensor* $\mathcal{X}$ with TT-ranks $r_k, k = 1, \ldots, d$ is given by

$$\mathcal{X}(i_1, \ldots, i_d) = X_1(i_1) \cdot X_2(i_2) \cdot \ldots \cdot X_d(i_d), \text{ with } X_k(i_k) \in \mathbb{R}^{r_{k-1} \times r_k}$$



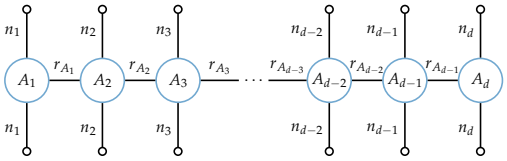
Storage complexity:

$\mathcal{O}((d-2)nr^2 + 2nr)$

- A *TT-Matrix* with TT-ranks $r_{A_k}, k = 1, \ldots, d$ is given by

$$\mathcal{A}(i_1, i_2, \ldots, i_d; j_1, \ldots, j_d) = A_1(i_1, j_1) \cdot A_2(i_2, j_2) \cdot \ldots \cdot A_d(i_d, j_d),$$

where $A_k(i_k, j_k) \in \mathbb{R}^{r_{A_{k-1}} \times r_{A_k}}$ is dependent on two indices.



Storage complexity:

$\mathcal{O}((d-2)n^2 r_A^2 + 2n^2 r_A)$

- *Complexities of operations in TT-format:*

Operation	Cost	Resulting tensor ranks
Summing two TT-tensors, $\mathcal{X} + \mathcal{Y}$	$\mathcal{O}(1)$	$r_\mathcal{X} + r_\mathcal{Y}$
TT-tensor times a scalar, $\alpha\mathcal{X}$	$\mathcal{O}(n_1 r_1)$	$r_\mathcal{X}$
Euclidean inner product, $\langle \mathcal{X}, \mathcal{Y} \rangle$	$\mathcal{O}(dnr^3)$	—
TT-matrix-vector product, $A\mathcal{X}$	$\mathcal{O}(dn^2 r^4)$	$r_A \cdot r_\mathcal{X}$
Truncation	$\mathcal{O}(dnr^3)$	can be prescribed

- *Existence of TT-Decomposition*:

Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ and $r_k = \operatorname{rank}(X_{(\{1,\ldots,k\})})$ for $k = 1, \ldots, d-1$. Then there exists a TT-Decomposition of $\mathcal{X}$ with TT-ranks r_k.

Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ be a TT-Tensor. Then there exists an approximation $\widetilde{\mathcal{X}}$ with TT-ranks r_k so that

$$\|\mathcal{X} - \widetilde{\mathcal{X}}\|_F \leq \sqrt{\sum_{k=1}^{d-1} \sum_{\ell=r_k+1}^{N_k} \left(\sigma_\ell^{\{1,\ldots,k\}}\right)^2},$$

where $\sigma_\ell^{\{1,\ldots,k\}}$ is the ℓth singular value of $X_{(\{1,\ldots,k\})}$ and $N_k = \min\{n_1 \cdot \ldots \cdot n_k, n_{k+1} \cdot \ldots \cdot n_d\}$.

- Given a tensor $\mathcal{X}$ and prescribed ranks r_k, there exists a best approximation $\mathcal{X}_{best}$ to $\mathcal{X}$ in the Frobenius norm with TT-ranks bounded by r_k and TT-SVD computes a quasi-optimal TT-Tensor $\mathcal{X}_{approx}$:

$$\|\mathcal{X} - \mathcal{X}_{approx}\|_F \leq \sqrt{d-1}\|\mathcal{X} - \mathcal{X}_{best}\|_F.$$

CHAPTER 4

From GMRES to multigrid

In this chapter we want to use the iterative method GMRES for computing the stationary distribution of a CTMC. Here the focus is not on the tensor structure but on the general feasibility of the method for this task.

4.1 Computing the stationary distribution of a CTMC via GMRES

In this chapter we assume that the reader is familiar with the Krylov subspace method GMRES [64] and give the algorithm only for the sake of completeness as Algorithm 4.1. We will motivate why it is advisable to use GMRES for solving the linear system (2.9) and will provide convergence analysis. Note that the transition rate matrix is singular and the right-hand side of our linear system (2.9) is zero. Therefore we clearly need a starting vector $x^{(0)} \neq \mathbf{0}$ for computing our Krylov subspace and we thus have to solve the residual equation

$$Ae^{(0)} = r^{(0)}, \tag{4.1}$$

where $e^{(0)} = x - x^{(0)}$ is the error and $r^{(0)} = b - Ax^{(0)}$ is the residual of the initial guess.

In [42] it is explained what kind of problems can occur when trying to solve a singular system by a Krylov subspace method, namely that the solution may not lie in the Krylov subspace. We say that a linear system $Ax = b$ has a *Krylov solution* x^* if $Ax^* = b$ and there exists some m such that $x^* \in \mathcal{K}_m(A, r^{(0)})$, i.e., if a solution to the linear system lies in the mth Krylov subspace corresponding to A and $r^{(0)}$. To guarantee the existence of a Krylov solution, some assumptions are necessary. Before handing this result out, we need the following definition:

Algorithm 4.1: GMRES method

$x^{(m)} = \text{GMRES}(A, b, x^{(0)}, m)$
$r^{(0)} \leftarrow b - Ax^{(0)}$
$v^{(1)} \leftarrow \frac{1}{\|r^{(0)}\|_2} r^{(0)}$
for $j = 1, \ldots, m$ **do**
  $w^{(j)} \leftarrow Av^{(j)}$
  for $i = 1, \ldots, j$ **do**
    $h_{i,j} \leftarrow (v^{(i)})^T w^{(j)}$
    $w^{(j)} \leftarrow w^{(j)} - h_{i,j} v^{(i)}$
  end
  $h_{j+1,j} \leftarrow \|w^{(j)}\|_2$
  if $h_{j+1,j} = 0$ **then**
    Stop.
  end
  $v^{(j+1)} \leftarrow \frac{1}{h_{j+1,j}} w^{(j)}$
end
Solve $\left\| \|r^{(0)}\|_2 e_1 - H_m y \right\|_2 \rightarrow \min$
Compute $x^{(m)} \leftarrow x^{(0)} + V_m y$

Definition 4.1. Let λ be an eigenvalue of A. The *index* of λ is the size of the largest Jordan block of A corresponding to λ.

Theorem 4.2. *A linear system $Ax = b$ has a Krylov solution if and only if $b \in range(A^i)$, where i is the index of the zero eigenvalue of A.*

Proof. See [42, Theorem 2]. □

We will show that for our system matrix A the index of the zero eigenvalue is one.

Lemma 4.3. *Let $A \in \mathbb{R}^{N \times N}$ be the generator matrix of a CTMC with unique stationary distribution x. Then the index of the zero eigenvalue of A is one.*

Proof. First note that the condition

$$range(A) \oplus N(A) = \mathbb{R}^N$$

is equivalent to the zero eigenvalue of A having index 1, see [42, Section 6]. Because of the uniqueness of the steady state distribution we have $N(A) = span(x)$ and thus $dim(N(A)) = 1$. Let $y \in range(A)$, i.e., $y = Az$. Then

$$\mathbf{1}^T y = \mathbf{1}^T Az = \mathbf{0}^T z = 0,$$

so that we have $range(A) \subseteq \mathbf{1}^\perp$. This even implies $range(A) = \mathbf{1}^\perp$, because both subspaces are of dimension $N - 1$. It remains to show that

$$range(A) \cap N(A) = \{\mathbf{0}\}.$$

Let $\mathbf{0} \neq y \in N(A)$, i.e., $y = \alpha x, \alpha \neq 0$. Then

$$\mathbf{1}^T y = \mathbf{1}^T \alpha x = \alpha \mathbf{1}^T x = \alpha \neq 0,$$

so that $y \notin \mathbf{1}^\perp = range(A)$. □

Due to Theorem 4.2 and Lemma 4.3 it follows that we need $b \in range(A)$ to guarantee the existence of a Krylov solution.

Starting with an initial guess $x^{(0)} \neq \mathbf{0}$, for our model problems the residual fulfils

$$r^{(0)} = \mathbf{0} - Ax^{(0)} = -Ax^{(0)} \in range(A).$$

Therefore the assumption of Theorem 4.2 is always fulfilled in our setting and it is thus reasonable to solve the system (2.9) with a Krylov subspace method. We are able to further characterize the Krylov solution from Theorem 4.2 and by doing so also briefly review why Krylov subspaces are good approximation spaces for singular systems.

First we define a particular pseudo inverse of A:

Definition 4.4. Let A have a zero eigenvalue with index i. The *Drazin inverse* A^D of A is the unique matrix which satisfies:

$$A^D A A^D = A^D, \quad A^D A = A A^D, \quad A^{i+1} A^D = A^i.$$

Note that if A is non-singular then $i = 0$ and $A^D = A^{-1}$. In [18, Corollary 7.2.1] it is shown that AA^D is a projector onto $range(A^i)$ along $N(A^i)$, the nullspace of A^i. Using this argument, one can prove that $AA^D b = b$ if and only if $b \in range(A^i)$, i.e., that $A^D b$ is a solution of $Ax = b$ in this case.

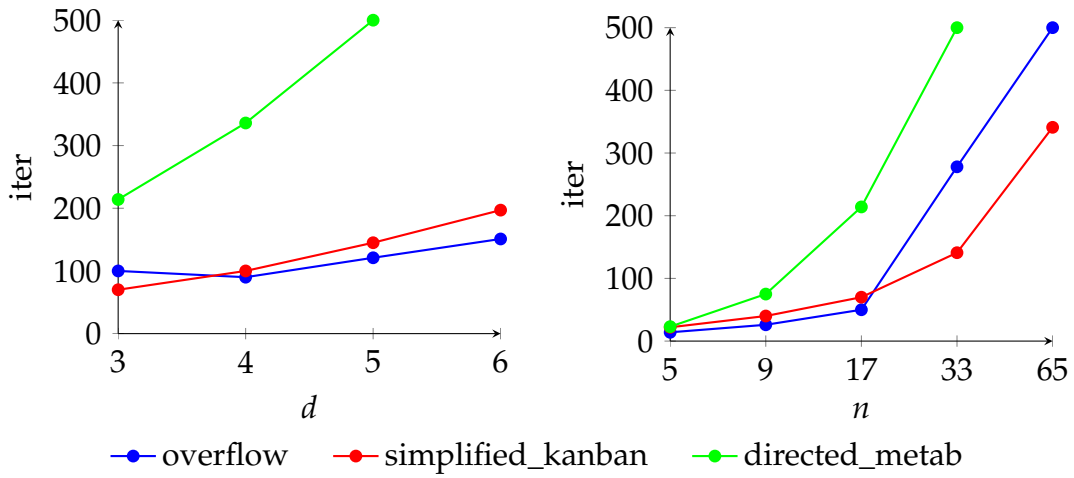


Figure 4.1: Number of GMRES(50) iterations for reducing the residual norm by a factor of 10^{-2} for three different model problems. Left: Mode size $n = 17$ and varying dimension d. Right: Dimension $d = 4$ and varying mode size n.

Missing for the classification of the Krylov solution from Theorem 4.2 is now only the correlation between this Krylov solution and the Drazin inverse. As shown in [42, Section 7], $A^D b$ can be expressed as $p(A)b$, where p is a polynomial of degree $m - i$ in A and m is the degree of the minimal polynomial of A, so that $A^D b \in \mathcal{K}_{m-i+1}(A, b)$. So summing everything up gives the following theorem [42, Theorem 3]:

Theorem 4.5. *Let m be the degree of the minimal polynomial of A and i the index of the zero eigenvalue of A. If $b \in range(A^i)$ then the linear system $Ax = b$ has a unique solution in $\mathcal{K}_{m-i+1}(A, b)$ given as $x = A^D b \in \mathcal{K}_{m-i+1}(A, b)$.*

Typically, GMRES is the method of choice for non-symmetric (and possibly singular) linear systems. Especially in our case, where we want to use tensor techniques (introduced in chapter 3), we benefit from the use of GMRES. Why this is the case can easily be seen from Algorithm 4.1, which depicts the GMRES method. The algorithm includes only basic operations, which can be realized with the TT-Format with cost linear in d. Note that if a method needs to access specific entries of a vector, a realization of this method in a TT format is not possible with costs linear in d. A tensorized version of GMRES is straightforwardly given, the only aspect which should be taken care of are the TT-ranks. Therefore, truncation should be done after every basic operation. In [28] a tensorized GMRES is considered and discussed. But the first step is now to look how GMRES behaves for our models described in section 2.2 without any tensor techniques.

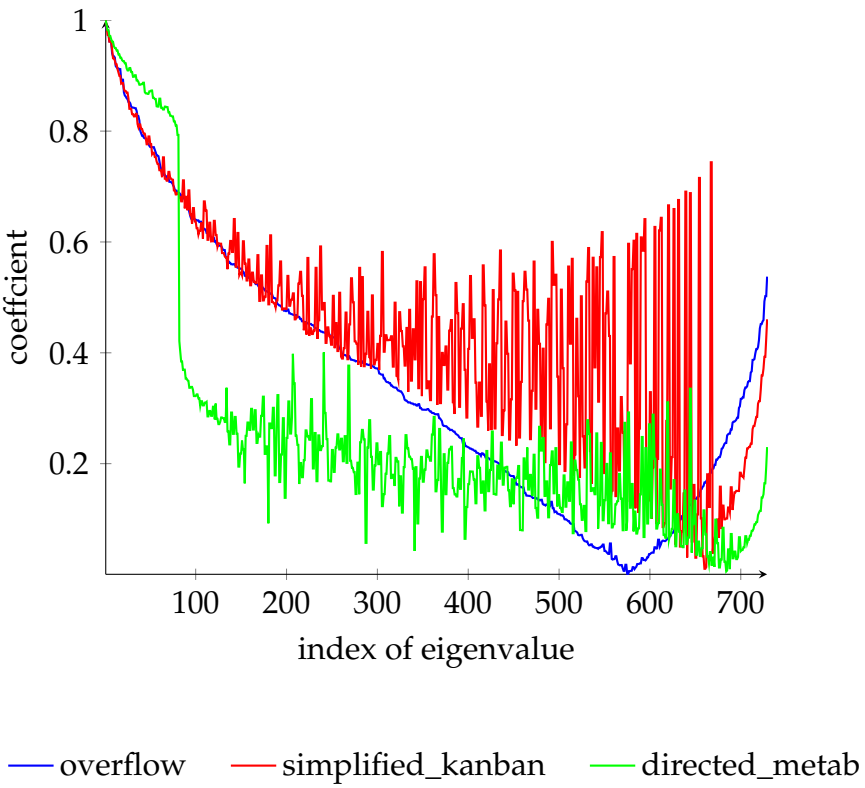


Figure 4.2: Reduction of error components belonging to different eigenvalues after one GMRES iteration.

To get a first impression we consider only three models, one of each category. We use restarted GMRES with restart length 50 (GMRES(50)) and stop after the residual of our initial guess—the normalized vector of all ones—is reduced by a factor of 10^{-2}. In Figure 4.1 we see the number of GMRES(50)-iterations which was needed for different problem sizes. The allowed maximum number of iterations is 500 (i.e., a maximum of ten restart cycles is performed), and we use the built-in MATLAB function `gmres`.

It is conspicuous that the iteration number increases superlinearly with growing problem size. This behaviour shows that the models are not easy to solve and that a new strategy should be considered for solving these kinds of problems. To find such a strategy, we first take a closer look at the reason for the slow convergence of GMRES. To do so, we apply a few GMRES steps to an initial guess which is chosen such that its error has equal contributions from all eigenvectors of A. In Figures 4.2, 4.3 and 4.4 we see the magnitude of the coefficients from the linear combination of the eigenvectors of the error of the GMRES iterate after one, three and five steps, respectively. The eigenvalues are ordered ascendingly with respect to their magnitude, i.e., $0 = |\lambda_1| < |\lambda_2| \leq \cdots \leq |\lambda_N|$. Note that

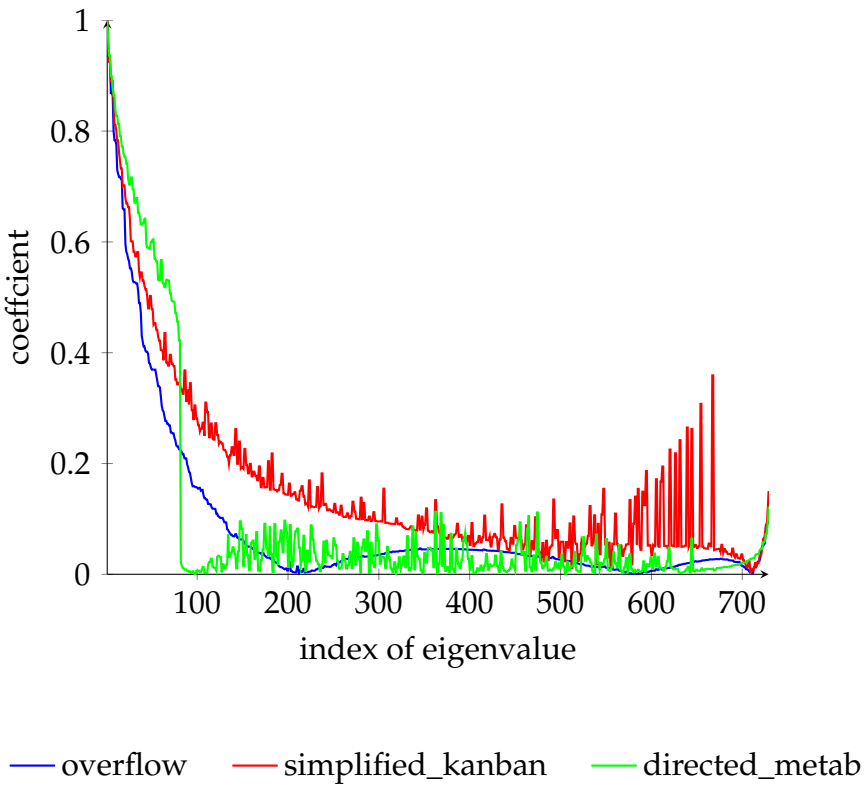


Figure 4.3: Reduction of error components belonging to different eigenvalues after three GMRES iterations.

before applying the GMRES steps the contributions from all eigenvalues were equal to one. So these figures show that only the parts corresponding to larger eigenvalues shrink substantially. It seems that the parts to smaller eigenvalues are "not touched" by GMRES after a few steps. Therefore the main remaining task after a few GMRES steps is to reduce the error parts corresponding to smaller eigenvalues.

In Figures 4.5–4.7 the value of each entry of the remaining error after three GMRES steps is indicated by the colouring of its corresponding state. Observe that for each state there are neighbouring states with similar values (independent of the connections in the model), so that we are able to approximate the error accurately with fewer degrees of freedom (and therefore also the system matrix). The hope is then that GMRES can handle the parts to smaller eigenvalues more efficiently. This idea is similar to the idea of multigrid methods, although with a slightly different motivation than usually.

Applying ν GMRES steps for the linear system (2.9) with initial guess $x^{(0)}$ delivers a vector $x^{(\nu)}$. Given $x^{(\nu)}$, to obtain the solution of the linear system (2.9), we

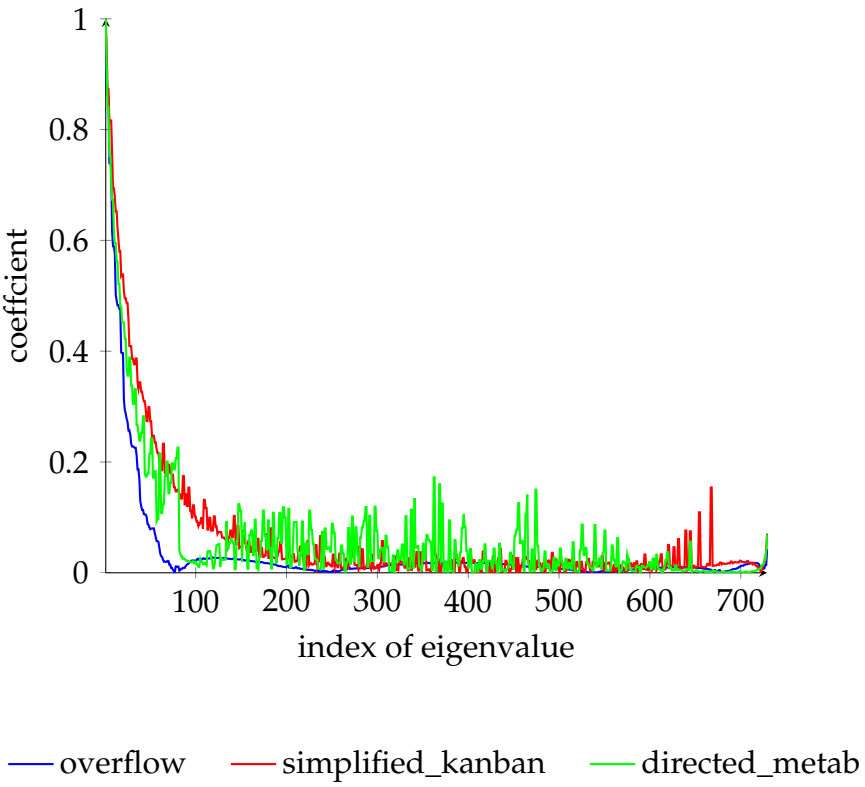


Figure 4.4: Reduction of error components belonging to different eigenvalues after five GMRES iterations.

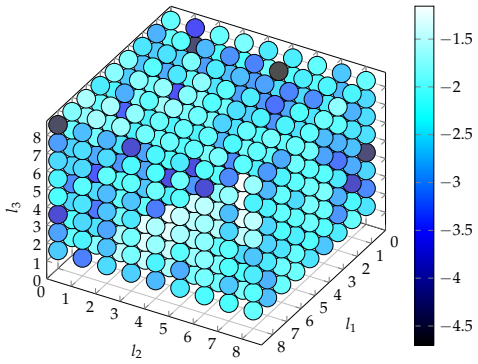


Figure 4.5: Error in the model *overflow* after three GMRES steps.

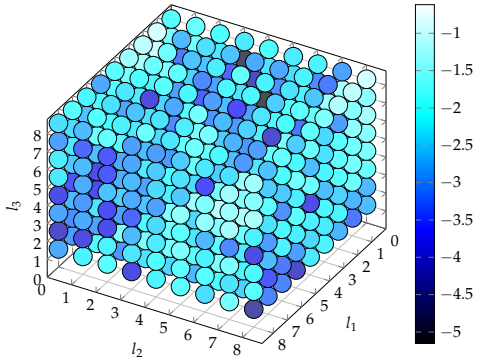


Figure 4.6: Error in the model *simplified_kanban* after three GMRES steps.

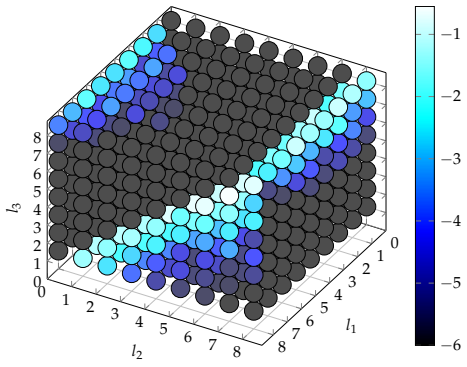


Figure 4.7: Error in the model *directed_metab* after three GMRES steps.

now consider the residual equation $Ae = r$ with $e = x^* - x^{(\nu)}$ and $r = -Ax^{(\nu)}$. According to Figures 4.2–4.4 this error e can be approximated accurately within a space $\mathcal{V}$ spanned by eigenvectors to small eigenvalues of A, i.e., with fewer degrees of freedom. If the columns of $V \in \mathbb{R}^{N \times N_c}$, where N_c is the new number of degrees of freedom, contain a basis of $\mathcal{V}$ we thus have $Ve_c \approx e$ for some $e_c \in \mathbb{R}^{N_c}$. This delivers an overdetermined system $AVe_c = r$.

To avoid the overdetermination we also require the residual of the error approximation to be orthogonal to $\mathcal{V}$, which gives $V^T(AVe_c - r) = 0$. This approach makes sense because we assume that the parts corresponding to smaller eigenvalues cause slower convergence of GMRES which we want to eliminate. We arrive at a system $V^TAVe_c = V^Tr$ of smaller dimension. The solution of this system then allows us to update our iterate as $\tilde{x}^{(\nu)} = x^{(\nu)} + Ve_c$. Ideally, the error of this iterate only contains components which can be efficiently handled by GMRES so that a few more iterations yield a good overall approximation.

In practice this procedure is not realizable, because of two reasons:

- The eigenvector information needed to build V is not available.

- V^TAV is in general a full matrix, even when A is sparse, so that the compressed system cannot be solved efficiently.

Therefore we need an easily computable and sparse matrix P, such that $span(P)$ approximates $span(V)$. Figures 4.5 and 4.6 show that locally the error changes only slowly, and this is to some extent also true for Figure 4.7. As the error mainly consists of components from $\mathcal{V}$ we can assume that the vectors in $\mathcal{V}$ also have this property. So a reasonable choice of P would be to have non-zero entries only where the entries belong to neighboring states. This delivers a sparse structure. This idea should motivate that a multigrid approach with GMRES as smoother for the models from chapter 2.2 is reasonable.

Next, we will give the basics of a multigrid method, to make the reader familiar with each important ingredient and our notation.

4.2 Algebraic multigrid basics

A multigrid method has the following building blocks:

- smoothing scheme,
- set of coarse variables,
- transfer operators,
- coarse grid operator.

In the remainder of this section, we briefly sketch how these building blocks are connected to each other and which role they take within a multigrid method. Thus, methods which we use to compute these building blocks in this thesis will only be listed here, their explanation and a reasoning for the specific choices we make will be given later in chapters 6 and 8.

For a thorough introduction to algebraic multigrid methods we refer the reader to [38, 62, 68] and the references therein.

The smoothing scheme is applied by doing a few iterations of a simple iterative method like Jacobi, Gauss-Seidel, Richardson [63], or a Krylov subspace method like GMRES [63, 64]. Note when using Jacobi, Gauss-Seidel or Richardson we have a so called *stationary smoother*. Stationary smoothers for a linear system

$Ax = b$ are characterized as follows: We split our system matrix $A = M - N$, M non-singular which defines an iteration of the form

$$Mx^{(k+1)} = Nx^{(k)} + b. \tag{4.2}$$

By elementary transformations we get an error propagation of the form

$$e^{(k+1)} = M^{-1}Ne^{(k)} =: Se^{(k)} \tag{4.3}$$

where $e^{(k)} = x - x^{(k)}$ and S is called iteration matrix of the method. The advantage of stationary smoothers concerning convergence analysis is that the error after ν smoothing steps can be written as

$$e^{(\nu)} = S^{\nu}e^{(0)}$$

which is not possible for non-stationary methods like GMRES. This allows to obtain convergence results more easily, see also section 5.2. The formula of the iteration matrices for each of the three methods will not be given here, but where the method is first used. For Richardson the iteration matrix is described in (5.16), for Gauss-Seidel and Jacobi in (6.2) and (6.3), respectively.

The smoothing process delivers an iterate $x^{(\nu)}$ with residual $r = b - Ax^{(\nu)}$, so that we can compute the error e by approximately solving the system $Ae = r$. For this we restrict the residual r by a matrix-vector multiplication with the restriction matrix $R \in R^{N_c \times N}$, and the operator $A^{(1)} := A$ is restricted via Petrov-Galerkin projection, yielding the coarse grid operator $A^{(2)} = RA^{(1)}P$, where $P \in \mathbb{R}^{N \times N_c}$ is the interpolation operator. If $A^{(2)}$ is small enough, i.e., computing the exact solution of the restricted system does not increase the computation time significantly, we have already arrived at the *coarsest grid*. Otherwise the process of smoothing and restriction is repeated until a coarsest grid is reached. After the coarsest system is solved and therefore a calculated error exists, this error approximation is interpolated to the next finer grid by a matrix-vector multiplication with P. The interpolated error is added to the current iterate and again a few smoothing steps are applied. This process is repeated until the finest grid is reached. Algorithm 4.2 reflects the described procedure and Figure 4.8 illustrates it. Figure 4.8 also shows that the recursive calls can be visualized by the letter V, therefore the procedure described in Algorithm 4.2 is also called V-cycle. Note that V-cycles can be performed repeatedly until a certain number of V-cycles has been performed or the residual has reached a certain accuracy. Therefore, whenever we mention an iteration of a multigrid method, we mean a complete V-cycle by this. Other cycle strategies, for example W- or F-cycles (see, e.g., [68]) can also be used, but we will only focus on V-cycles in this thesis.

For sake of notational simplicity let us now assume that we have a two level approach. By now it is not clear how to choose N_c and how to compute the entries

Algorithm 4.2: Multigrid V-cycle

$v^{(\ell)} = \mathrm{MG}(b^{(\ell)}, v^{(\ell)})$
if *coarsest grid is reached* **then**
  Solve coarse grid equation $A^{(\ell)}v^{(\ell)} = b^{(\ell)}$
else
  Perform ν_{pre} smoothing steps for $A^{(\ell)}v^{(\ell)} = b^{(\ell)}$ with initial guess $v^{(\ell)}$
  Compute the residual $r^{(\ell)} \leftarrow b^{(\ell)} - A^{(\ell)}v^{(\ell)}$
  Restrict $b^{(\ell+1)} \leftarrow R^{(\ell)}r^{(\ell)}$
  $v^{(\ell+1)} \leftarrow \mathbf{0}$
  $e^{(\ell+1)} = \mathrm{MG}(b^{(\ell+1)}, v^{(\ell+1)})$
  Interpolate $e^{(\ell)} \leftarrow P^{(\ell)}e^{(\ell+1)}$
  $v^{(\ell)} \leftarrow v^{(\ell)} + e^{(\ell)}$
  Perform ν_{post} smoothing steps for $A^{(\ell)}v^{(\ell)} = b^{(\ell)}$ with initial guess $v^{(\ell)}$
end

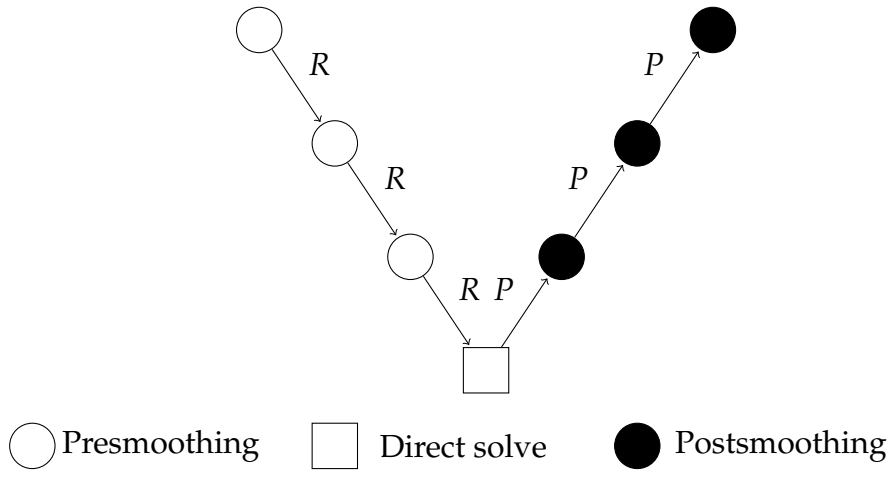


Figure 4.8: Multigrid V-cycle: On each level, a presmoothing iteration is performed before the problem is restricted to the next coarser grid. On the smallest grid, the problem is typically solved exactly by a direct solver. When interpolating back to the finer grids, postsmoothing iterations are applied on each level.

of the interpolation and restriction operator P and R to arrive at the coarse grid space. We use geometric coarsening to split the given N variables into fine and coarse variables, denoted with $\mathcal{F}$ and $\mathcal{C}$, respectively, i.e., $\mathcal{C} \cup \mathcal{F} = \{1, \dots, N\}$ and $\mathcal{C} \cap \mathcal{F} = \emptyset$, and set $N_c = |\mathcal{C}|$. If such a $\mathcal{C}/\mathcal{F}$-splitting is given, the transfer operators are defined as

$$R : \mathbb{R}^{|\mathcal{C}\cup\mathcal{F}|} \to \mathbb{R}^{|\mathcal{C}|}, \quad P : \mathbb{R}^{|\mathcal{C}|} \to \mathbb{R}^{|\mathcal{C}\cup\mathcal{F}|}.$$

It is still open how to obtain the entries of these operators. In this thesis linear interpolation, direct interpolation [62] and a bootstrap approach [6, 11] will be the methods of choice.

At the beginning of this chapter, the goal was to use GMRES for solving the models from section 2.2. Smaller problem sizes already show that GMRES as a stand-alone method does not solve this task satisfactorily. We illustrated why a multigrid method with GMRES as smoothing scheme can be expected to work efficiently. In the next chapter, we review some results from multigrid theory for non-symmetric and for singular linear systems, in order to use these results as guidance for how to choose the ingredients of our tensorized multigrid method for Markov chains.

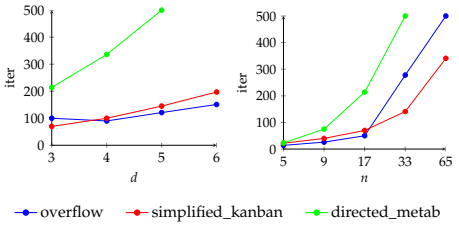
Summary of Chapter 4:

- For non-symmetric systems, the Krylov subspace method GMRES is typically the method of choice.
- *Existence of Krylov solution for singular systems:*

  A singular linear system $Ax = b$ has a Krylov solution if and only if $b \in range(A^i)$, where i is the index of the zero eigenvalue of A.
- The zero eigenvalue of all models from section 2.2 has index 1.
- Given an initial guess $x^{(0)}$ we apply GMRES to $Ae = r^{(0)}$ with

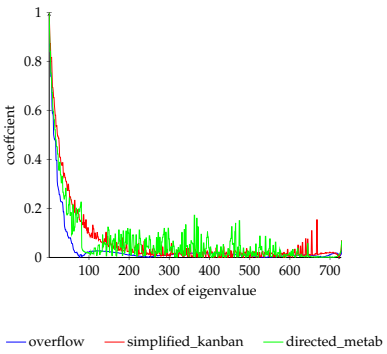
$$r^{(0)} = b - Ax^{(0)} = -Ax^{(0)} \in range(A).$$

  $\Rightarrow$ Krylov solution always exists.
- *Restarted GMRES as standalone solver?*

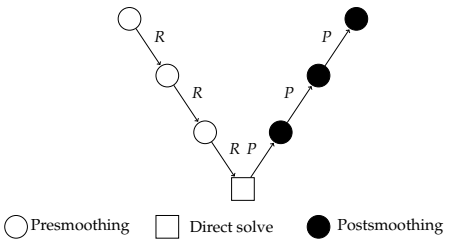


  $\oplus$ only basic operations $\Rightarrow$ can be implemented in TT-format

  $\ominus$ number of iterations increases superlinearly
- *Effect of GMRES on error components*
  - error components belonging to large eigenvalues are reduced efficiently.
  - remaining components can be handled on smaller subspace $\Rightarrow$ multigrid approach



- *Building blocks of a multigrid V-cycle*



  - smoothing scheme
  - set of coarse variables
  - transfer operators
  - coarse grid operator

CHAPTER 5

Elements of a convergence analysis of multigrid for non-symmetric and singular systems

The models introduced in section 2.2 are non-symmetric and singular. A convergence analysis of multigrid for these systems is not known and not easy to develop, while for *symmetric positive definite (spd)* matrices a convergence analysis for multigrid (MG) exists [54, 62, 68]. In this chapter we will consider two approaches for convergence analysis, where both of them only deal with one issue, either non-symmetric or singular systems. Besides, we will discuss at which point the transfer to the non-symmetric and singular case fails and which results we can transfer. We will start with the case that our system matrix is symmetric and singular.

5.1 The symmetric and singular case

The idea for the convergence analysis for symmetric and singular matrices is based on the following observation: If the choice of the ingredients for MG is such that the method can be described as a stationary iterative method with an error propagation operator $H = I - \widetilde{M}A$ with $\widetilde{M} \in \mathbb{R}^{N\times N}$ as

$$x^{(k+1)} = Hx^{(k)} + \widetilde{M}b, \quad k = 0, 1, \ldots \tag{5.1}$$

where A is the symmetric, singular system matrix and I the identity matrix, then each fixed point of (5.1) is a solution of $Ax = b$ if $\widetilde{M}$ is injective on the range of

A. To have a criterion under which conditions convergence to such a fixed point can be guaranteed, we need the following definition:

Definition 5.1. H is called *semiconvergent* if $\rho(H) = 1$, the eigenvalue $\lambda = 1$ is the only eigenvalue of modulus 1 and $\lambda = 1$ is a semisimple eigenvalue of H, i.e., its geometric multiplicity is equal to its algebraic multiplicity.

Then one can show

Lemma 5.2. *Iteration* (5.1) *converges to a fixed point x^* for any starting vector $x^{(0)}$ if and only if H is semiconvergent and $\widetilde{M}$ is injective on $range(A)$.*

Proof. See [4, Lemma 6.13]. □

As motivated in [7] with the help of the bilinear form

$$\langle .,. \rangle_A : \mathbb{R}^N \times \mathbb{R}^N, (x, y) \mapsto \langle x, y \rangle_A := \langle Ax, y \rangle$$

and the induced seminorm $\|x\|_A = \langle x, x \rangle_A^{1/2}$ we get a practical fundamental result for analysing the convergence of the iteration (5.1), namely:

Theorem 5.3. *Let H be the iteration operator of* (5.1) *and assume that there exists a constant $\gamma \in [0, 1)$ such that*

$$\|Hx\|_A \leq \gamma \cdot \|x\|_A \text{ for all } x \in \mathbb{R}^N. \tag{5.2}$$

Then

(i) $\widetilde{M}$ is injective on range(A) and

(ii) H is semiconvergent.

Besides, for $b \in range(A)$ iteration (5.1) *converges to a solution of $Ax = b$ for any starting vector.*

Proof. [7, Theorem 1]. □

We want to transfer this theorem to the non-symmetric and singular case. It is obvious that A does not induce a seminorm. But we can look at the seminorm induced by $\sqrt{A^TA}$, as is done in the analysis of multigrid for non-symmetric, non-singular systems [13, 65], and we get

$$\begin{aligned} \langle x, x\rangle_{\sqrt{A^TA}} &\geq 0 \text{ for all } x \in \mathbb{R}^N, && (5.3)\\ \|x\|_{\sqrt{A^TA}} &= 0 \text{ if and only if } x \in N(A), && (5.4)\\ \langle x, y\rangle_{\sqrt{A^TA}} &= 0 \text{ for } x \in N(A) \text{ or } y \in N(A). && (5.5) \end{aligned}$$

The second and third property follow from the fact that $N(\sqrt{A^TA}) = N(A)$ which can be seen using the SVD and that $\sqrt{A^TA}$ is symmetric. So we can adapt Theorem 5.3 in the following way:

Theorem 5.4. *Let H be the iteration operator of* (5.1) *and assume that there exists a constant $\gamma \in [0, 1)$ such that*

$$\|Hx\|_{\sqrt{A^TA}} \leq \gamma \cdot \|x\|_{\sqrt{A^TA}} \text{ for all } x \in \mathbb{R}^N. \tag{5.6}$$

Then

(i) $\widetilde{M}$ is injective on range(A) and

(ii) H is semiconvergent.

Besides, for $b \in$ range(A) iteration (5.1) *converges to a solution of $Ax = b$ for any starting vector.*

Proof. For the first assertion, we show that $N(\widetilde{M}A) = N(A)$. Note that $N(\widetilde{M}A) = N(I - H)$. First, let $y \in N(A)$. Then $\widetilde{M}Ay = \widetilde{M}\mathbf{0} = \mathbf{0}$, so that $y \in N(\widetilde{M}A)$.

Now, let $y \notin N(A)$. Then (5.6) implies

$$\|Hy\|_{\sqrt{A^TA}} \leq \gamma \cdot \|y\|_{\sqrt{A^TA}} \text{ with } \|y\|_{\sqrt{A^TA}} \neq 0,$$

from which it follows that $Hy \neq y$. Therefore,

$$(I - H)y = y - Hy \neq 0,$$

i.e., $y \notin N(I - H) = N(\widetilde{M}A)$. This proves the first assertion.

For the second part of the theorem, let x be an eigenvector of H, i.e., $Hx = \lambda x$. If $x \notin N(A)$ then $\|x\|_{\sqrt{A^TA}} > 0$ and (5.6) implies

$$|\lambda| \cdot \|x\|_{\sqrt{A^TA}} \leq \gamma \cdot \|x\|_{\sqrt{A^TA}},$$

so that $|\lambda| \leq \gamma < 1$.

If $x \in N(A)$ then $Hx = x$, i.e., $\lambda = 1$ which implies $\rho(H) = 1$, and $\lambda = 1$ is the only eigenvalue of modulus one. Assume that $\lambda = 1$ is not semisimple. In this case, there exists $v \neq 0$ such that $Hv = v + u$, where $Hu = u, u \neq 0$. Then v is not an eigenvector of H, and thus $v \notin N(A)$. Further, we have $u \in N(A)$, because $Hu = u$ and $\widetilde{M}$ is injective on $range(A)$. Using these facts together with the properties of the $\sqrt{A^TA}$-seminorm, we find

$$\begin{aligned} \|Hv\|^2_{\sqrt{A^TA}} &= \langle Hv, Hv\rangle_{\sqrt{A^TA}} \\ &= \langle v+u, v+u\rangle_{\sqrt{A^TA}} \\ &= \langle v, v\rangle_{\sqrt{A^TA}} + \langle v, u\rangle_{\sqrt{A^TA}} + \langle u, v\rangle_{\sqrt{A^TA}} + \langle u, u\rangle_{\sqrt{A^TA}} \\ &= \langle v, v\rangle_{\sqrt{A^TA}} = \|v\|^2_{\sqrt{A^TA}} \neq 0. \end{aligned}$$

This is a contradiction to the assumption (5.6), and $\lambda = 1$ must therefore be a semisimple eigenvalue. □

We want to find conditions under which the assumption (5.2) of Theorem 5.3 is fulfilled for the iteration operator H of a two grid method. From section 4.2 we are familiar with the constitutive elements of a MG method, so therefore it is clear that the iteration operator of a two grid method with interpolation operator P and a stationary iterative method with iteration matrix S from (4.3) as smoother is given by $H = S^{\nu_{post}} \cdot K \cdot S^{\nu_{pre}}$, where K is the coarse grid correction operator given via the formula $K = I - PA_c^{\dagger}P^TA$ and ν_{pre}, ν_{post} are the number of pre- and postsmoothing steps, respectively. The following theorem is the central result for the two grid convergence analysis in the symmetric singular case.

Theorem 5.5. *Let $A \in \mathbb{R}^{N\times N}$ be symmetric and positive semidefinite. Let $P \in \mathbb{R}^{N\times N_c}$ have full rank, let $K = I - PA_c^{\dagger}P^TA$ and $\nu_{pre} = 0, \nu_{post} = 1$, i.e., $H = S \cdot K$. Besides, suppose that there exists a number $\delta \in (0, 1]$ such that*

$$\|Sx\|_A^2 \leq \|x\|_A^2 - \delta \cdot \|Kx\|_A^2 \text{ for } x \in \mathbb{R}^N. \tag{5.7}$$

Then

(i) K is an A-orthogonal projector, i.e., $\langle Kx, (I-K)y\rangle_A = 0$ for all $x, y \in \mathbb{R}^N$, and

(ii) $\|Hx\|_A^2 \leq (1-\delta) \cdot \|x\|_A^2$ for all $x \in \mathbb{R}^N$.

Proof. For (i), see [7, Theorem 2].

For (ii), note that $K^2 = K$. With (5.7) and replacing x by Kx we get

$$\|Hx\|_A^2 \leq (1-\delta)\|Kx\|_A^2. \tag{5.8}$$

Because K is an A-orthogonal projection and therefore $\|Kx\|_A \leq \|x\|_A$ holds for all $x \in \mathbb{R}^N$, we arrive at (ii). □

Again we would like to adapt Theorem 5.5 to the seminorm $\|\cdot\|_{\sqrt{A^TA}}$. Consider the proof of statement (ii) in Theorem 5.5. It is obvious that equation (5.8) also holds for the seminorm induced by $\sqrt{A^TA}$, but the necessary inequality $\|Kx\|_{\sqrt{A^TA}} \leq \|x\|_{\sqrt{A^TA}}$ for concluding the proof does not hold, because albeit K is a projection, it is not $\sqrt{A^TA}$-orthogonal.

5.2 The non-symmetric and non-singular case

Now we consider the case that our system matrix is non-symmetric, but non-singular. In [13,65] a convergence analysis for this case is given, which we recapitulate in this section. The analysis is based on looking at two different coarse grid projections of a two grid method and their relation to each other. The first projection we are looking at is given via the formula

$$I - \Pi_{QA} \quad \text{with} \quad \Pi_{QA} = P(P^TQAP)^{-1}P^TQA \tag{5.9}$$

where P is the interpolation matrix. Here the matrix $Q = VU^T$, where U and V are the matrices containing the left and right singular vectors of $A = U\Sigma V^T$, plays an important role, because it allows to work with an spd matrix: We have $QA = V\Sigma V^T = \sqrt{A^TA}$, and thus QA is an spd matrix with the same singular values as A. We can prove that $I - \Pi_{QA}$ is a QA-orthogonal projection, see [13, Section 2]. Recall that the lack of an orthogonal projection was the main reason why the first convergence analysis presented in this chapter fails in our setting. $I - \Pi_{QA}$ is a full matrix (because QA is a full matrix), so it is only of theoretical interest, but not practical. For that reason we are looking at the next projection

$$I - \Pi_A \quad \text{with} \quad \Pi_A = P(RAP)^{-1}RA \tag{5.10}$$

where R is the restriction matrix. In contrast to $I - \Pi_{QA}$, this projection is not orthogonal with respect to any problem-related inner product, it is an oblique projection. The next lemma shows the relation between $I - \Pi_{QA}$ and $I - \Pi_A$:

Lemma 5.6. *The following projection identities for Π_{QA} and Π_A hold:*

$$\begin{aligned} \Pi_{QA}\Pi_A &= \Pi_A, \\ \Pi_A\Pi_{QA} &= \Pi_{QA}. \end{aligned}$$

Proof. By a direct calculation, we verify

$$\begin{aligned}\Pi_{QA}\Pi_A &= P(P^TQAP)^{-1}P^TQAP(RAP)^{-1}RA \\ &= P(RAP)^{-1}RA = \Pi_A, \\ \Pi_A\Pi_{QA} &= P(RAP)^{-1}RAP(P^TQAP)^{-1}P^TQA \\ &= P(P^TQAP)^{-1}P^TQA = \Pi_{QA}.\end{aligned}$$

□

With Lemma 5.6 and the following assumption we arrive at an estimate which will be useful for the convergence analysis.

Definition & Lemma 5.7. If for any $e \in \mathbb{R}^N$ there exists an $e_c \in \mathbb{R}^{N_c}$ such that

$$\|e - Pe_c\|_{QA}^2 \leq \frac{K_s}{\|QA\|_2}\langle QAe, QAe\rangle. \tag{5.11}$$

where K_s is a constant, we say that P fulfils the *strong approximation property* with constant K_s.

Besides, (5.11) implies

$$\|(I - \Pi_{QA})e\|_{QA}^2 \leq \frac{K_s}{\|A\|_2}\langle Ae, Ae\rangle = \frac{K_s}{\|A\|_2}\|e\|_{A^TA}^2 \tag{5.12}$$

for all $e \in \mathbb{R}^N$.

Proof. For given $e \in \mathbb{R}^N$ we have

$$\|e - \Pi_{QA}e\|_{QA} = \min_{v \in range(P)} \|e - v\|_{QA},$$

because Π_{QA} is the QA-orthogonal projection onto $range(P)$. In particular

$$\|e - \Pi_{QA}e\|_{QA}^2 \leq \|e - Pe_c\|_{QA}^2$$

for the vector e_c from (5.11). The result then follows by noting that $\|QA\|_2 = \|A\|_2$ and $\langle QAe, QAe\rangle = \langle Ae, Ae\rangle$, because Q is unitary. □

Lemma 5.8. *If* (5.11) *holds, we have*

$$\|(I - \Pi_A)e\|_{QA}^2 \leq \frac{\|\Pi_A\|_{QA}^2 K_s}{\|A\|_2}\|e\|_{A^TA}^2. \tag{5.13}$$

Proof. Because of the projection identities shown in Lemma 5.6 we have

$$(I - \Pi_A)(I - \Pi_{QA}) = I - \Pi_A.$$

Due to this identity and (5.12) we have:

$$\begin{aligned} \|(I - \Pi_A)e\|_{QA}^2 &= \|(I - \Pi_A)(I - \Pi_{QA})e\|_{QA}^2 \\ &\leq \|(I - \Pi_A)\|_{QA}^2 \|(I - \Pi_{QA})e\|_{QA}^2 \\ &\leq \|(I - \Pi_A)\|_{QA}^2 \frac{K_s}{\|A\|_2} \|e\|_{A^TA}^2. \end{aligned}$$

The result then follows because a projection has the same norm as its complement. This proof can be found in [13, Lemma 2.2]. □

Lemma 5.8 delivers an estimate for the effect of the coarse grid correction on the error. Next we need to incorporate the effect of a smoothing iteration to arrive at a two grid convergence result. For sake of simplicity we consider the weighted Richardson iteration applied to the normal equations, for which the error propagation operator S is given as

$$S = \left(I - \frac{1}{\|A\|_2^2} A^T A \right). \tag{5.14}$$

Theorem 5.9. *Under the assumptions of Lemma 5.8, it holds*

$$\|(I - \Pi_A)S^\nu e\|_{QA} \leq \frac{16\|\Pi_A\|_{QA}^2 K_s}{25\sqrt{4\nu + 1}} \|e\|_{QA}^2. \tag{5.15}$$

Proof. It holds

$$\begin{aligned} \|(I - \Pi_A)S^\nu e\|_{QA} &\leq \frac{\|\Pi_A\|_{QA}^2 K_s}{\|A\|_2} \|S^\nu e\|_{A^TA}^2 \\ &= \frac{\|\Pi_A\|_{QA}^2 K_s}{\|A\|_2} \langle QAS^\nu e, QAS^\nu e \rangle \\ &= \frac{\|\Pi_A\|_{QA}^2 K_s}{\|A\|_2} \|(QA)^{\frac{1}{2}} S^\nu e\|_{QA}^2. \end{aligned}$$

Decomposing e into the eigenvectors of QA (which form a basis of $\mathbb{R}^N$ because QA is symmetric), $e = \sum_{j=1}^N \beta_j v_j$ with $QAv_j = \sigma_j v_j$, we obtain

$$\begin{aligned} \|(QA)^{\frac{1}{2}} S^\nu e\|_{QA}^2 &= \| \sum_{j=1}^N \sigma_j^{\frac{1}{2}} \left(1 - \frac{\sigma_j^2}{\|A\|_2^2} \right)^\nu \beta_j v_j \|_{QA}^2 \\ &\leq s\|e\|_{QA}^2, \end{aligned} \tag{5.16}$$

where

$$s = \max_{\sigma \in [0, \|A\|_2^2]} \sigma \left(1 - \frac{\sigma^2}{\|A\|_2^2}\right)^{2\nu}$$

This maximum occurs at $\widetilde{\sigma} = \frac{\|A\|_2}{\sqrt{4\nu+1}}$ (see [13, Theorem 2.3]) and therefore

$$s = \frac{\|A\|_2}{\sqrt{4\nu+1}} \left(\frac{4\nu}{4\nu+1}\right)^{2\nu} \leq \frac{16\|A\|_2}{25\sqrt{4\nu+1}}$$

for $\nu \geq 1$, because $\left(\frac{4\nu}{4\nu+1}\right)^{2\nu}$ is monotonically decreasing for $\nu \geq 1$. This proof can be found in [13, Theorem 2.3]. □

Remark 5.10. Note that Theorem 5.9 implies that two grid convergence can always be achieved under our assumptions, but not necessarily for any number of smoothing steps, as the constant on the right-hand side of (5.15) may be larger than one for small values of ν. The necessary number of smoothing steps depends quadratically on the constant K_s from the approximation property.

Consider now the case that the system matrix A is singular. The first step is to replace the inverse matrices in projections Π_{QA} and Π_A by the Moore-Penrose pseudo inverse, therefore we have

$$I - \Pi_{QA} = I - P(P^T QAP)^\dagger P^T QA \quad \text{and} \tag{5.17}$$

$$I - \Pi_A = I - P(RAP)^\dagger RA. \tag{5.18}$$

We have to investigate whether $I - \Pi_{QA}$ is still a QA-orthogonal projection and whether the projection identities still hold. If this were the case then all other arguments for the proofs of Lemma 5.8 and of Theorem 5.9 would hold in exactly the same way. To make this directly evident to the reader was the reason why we reproduced the proofs of Lemma 5.8 and of Theorem 5.9 here instead of just citing the results.

Lemma 5.11. *$I - \Pi_{QA}$ given by* (5.17) *is a QA-orthogonal projection if A is singular.*

Proof. By direct calculation and with help of the properties of the Moore-Penrose pseudo inverse, we get $(I - \Pi_{QA})^2 = I - \Pi_{QA}$. So we only have to show that

$$\langle \Pi_{QA} x, (I - \Pi_{QA}) y \rangle_{QA} = 0 \text{ for all } x, y \in \mathbb{R}^N.$$

For this, consider

$$\begin{aligned}
& \langle \Pi_{QA} x, (I - \Pi_{QA}) y \rangle_{QA} \\
= \; & \langle x, \Pi_{QA}^T QA(I - \Pi_{QA}) y \rangle \\
= \; & \langle x, QAP(P^T QAP)^\dagger P^T QA(I - P(P^T QAP)^\dagger P^T QA) y \rangle \\
= \; & \langle x, QAP(P^T QAP)^\dagger P^T QAy - QAP(P^T QAP)^\dagger P^T QAy \rangle \\
= \; & \langle x, 0 \rangle \\
= \; & 0.
\end{aligned}$$

□

Remark 5.12. Note that if we use the projection $I - \Pi_{QA}$ for the operator K in Theorem 5.5, we have a convergence analysis for the non-symmetric singular case, but with a non-practical operator.

Besides, it is obvious that the proof for the projection identities can not be adapted for the singular case, because $B^\dagger B \neq I$ if B is singular. It is known that $B^\dagger B$ acts as the identity on vectors from $range(B^\dagger) = N(B)^\perp$, so for the proof of the crucial identity $\Pi_A \Pi_{QA} = \Pi_{QA}$ from Lemma 5.6 to hold, we would need to show that $(P^T QAP)^\dagger P^T QA$ maps onto the subspace $range(B^\dagger)$ with $B = RAP$. Unfortunately, it does not seem possible to find practical conditions under which this can be guaranteed.

Summary of Chapter 5:

Convergence analysis for MG for the symmetric and singular case

- Convergence if the error propagation operator H is semiconvergent.
- $\|Hx\|_A \leq \gamma \cdot \|x\|_A$ for all $x \in \mathbb{R}^N, \gamma \in [0,1) \Rightarrow H$ is semiconvergent.
- Choose the coarse grid correction as $K = I - PA_c^\dagger P^T A$
  $\Rightarrow K$ is an A-orthogonal projector and under suitable assumptions:
  $$\|Sx\|_A \leq \|x\|_A^2 - \delta \cdot \|Kx\|_A^2 \text{ for } x \in \mathbb{R}^N$$
  implies
  $$\|Hx\|_A^2 \leq (1-\delta) \cdot \|x\|_A^2 \text{ for all } x \in \mathbb{R}^N.$$

Convergence analysis for MG for the non-symmetric and non-singular case

Consider projections (also coarse grid corrections) and $QA = \sqrt{A^T A}$:

- $\Pi_{QA} = P(P^T QAP)^{-1} P^T QA \quad$ and $\quad \Pi_A = P(RAP)^{-1} RA$
- $I - \Pi_{QA}$ is a QA-orthogonal projector, but not a practical one.
- It holds $\Pi_A \Pi_{QA} = \Pi_{QA}$ and under suitable assumptions
  $$\|(I - \Pi_{QA})e\|_{QA}^2 \leq \frac{K_s}{\|A\|_2} \langle Ae, Ae \rangle = \frac{K_s}{\|A\|_2} \|e\|_{QA}^2$$
  implies
  $$\|(I - \Pi_A) S^\nu e\|_{QA} \leq \frac{16 \|\Pi_A\|_{QA}^2 K_s}{25\sqrt{4\nu + 1}} \|e\|_{QA}^2.$$

$\Rightarrow$ Convergence is guaranteed for sufficiently many smoothing steps.

Transfer to the non-symmetric and singular case

- $\|Hx\|_{QA} \leq \gamma \cdot \|x\|_{QA}$ for all $x \in \mathbb{R}^N, \gamma \in [0,1) \Rightarrow H$ is semiconvergent.
- When A is non-symmetric $K = I - PA_c^\dagger P^T A$ is not orthogonal with respect to the QA inner product and therefore it does not hold
  $$\|Hx\|_{QA}^2 \leq (1-\delta) \cdot \|x\|_{QA}^2 \text{ for all } x \in \mathbb{R}^N,$$
  but for $K = I - \Pi_{QA} = I - P(P^T QAP)^\dagger P^T QA$ (which is not practical).
- When A is singular $\Pi_A \Pi_{QA} \neq \Pi_{QA}$ with $\Pi_A = P(RAP)^\dagger RA$.
- The strong approximation property only allows a conclusion about Π_{QA}. Without the projection identity we cannot relate it to Π_A.

CHAPTER 6

Multigrid for tensor-structured problems

The convergence analysis in chapter 5 gives us a reference point how each ingredient for our method should be chosen. Note that we also have the demand that the choice of each ingredient should be compatible with the tensor structure of our system matrix A and with the TT-techniques from chapter 3. In the following we will discuss the selection of each ingredient with the focus on these two demands, whereby the focus on the compatibility with the TT-techniques is weighted more heavily.

6.1 Smoother

The proof of Theorem 5.9 motivates that we want to get the seminorm $\|(QA)^{\frac{1}{2}}S^\nu e\|_{QA}$ small. Because of the following relation

$$\begin{aligned}\|(QA)^{\frac{1}{2}}S^\nu e\|_{QA} &= \langle QA(QA)^{\frac{1}{2}}S^\nu e, (QA)^{\frac{1}{2}}S^\nu e\rangle \\ &= (S^\nu e)^T(QA)^2 S^\nu e \\ &= (S^\nu e)^T A^T A S^\nu e \\ &= \langle S^\nu e, S^\nu e\rangle_{A^TA} = \|S^\nu e\|^2_{A^TA}\end{aligned}$$

we demand that our smoother should reduce the A^TA-seminorm of our error. Recall that GMRES minimizes the two-norm of the residual over the Krylov subspace and that

$$\|r\|_2 = \|Ae\|_2 = \|e\|_{A^TA}.$$

In contrast to the Richardson iteration (5.14), GMRES is not a stationary iteration and cannot be described by an iteration matrix S and a reduction of the error

after a few smoothing steps cannot be guaranteed by GMRES (GMRES can, e.g., stagnate for $N-1$ steps and arrive at the solution at the Nth step, see [2,37]), so that an estimate of the form (5.7) or (5.16) cannot be given.

This all shows that Richardson is valuable for the theory, but for the practical implementation $\|A\|_2$ has to be known for choosing the optimal damping parameter. For our kind of model problems this computation is not possible because of the curse of dimensionality. In practice we can observe that a few GMRES steps reduce the error significantly for most considered models. Other smoothers which are nice for the analysis and are very popular choices for multigrid methods are Jacobi and Gauss-Seidel [36, 63, 68]. In the following we give the iteration matrices for both methods:

Definition 6.1. Let $A \in \mathbb{R}^{N\times N}$ be decomposed in the following way:

$$A = L + D + U, \tag{6.1}$$

where D is a diagonal matrix, L is a strictly lower and U a strictly upper triangular matrix. The *Gauss-Seidel* method is given via the iteration

$$x^{(k+1)} = (D+L)^{-1}(b - Ux^{(k)})$$

and the *weighted Jacobi* method via

$$x^{(k+1)} = \omega D^{-1}(b - (L+U)x^{(k)}) + (1-\omega)x^{(k)}$$

where ω is the relaxation parameter. Therefore the iteration matrices are

$$S_{GS} = (D+L)^{-1}U = I - (D+L)^{-1}A \quad \text{and} \tag{6.2}$$
$$S_J = I - \omega D^{-1}A, \tag{6.3}$$

respectively.

Eye-catching is that the iteration matrices (6.2) and (6.3) include inverse matrices. Naturally, a diagonal or triangular linear system is not difficult to solve, the bigger problem in our high-dimensional setting is that this implies access to each entry of the operator D or $D+L$. Recall that we want to use tensor techniques to avoid the handling of each entry for reasons of storage and computational complexity. The curse of dimensionality should also again be mentioned here. So in the tensor setting these smoothers are problematic at first sight.

A necessary requirement for being able to use them at all is that we can represent L, U and D in a (storage-efficient) tensor format. In the following, we explain that this is indeed possible in our case. For this recall that the models from section 2.2 are of the form $A = A_L + A_S + A_D$, see (2.16).

Lemma 6.2. *If $A = A_L + A_S + A_D$ is the generator matrix of a SAN, then D, L and U from* (6.1) *can be computed in the following way.*

- *For the diagonal matrix D:*

$$D = D_L + D_S + A_D$$

with

$$D_L = \sum_{i=1}^{d} I_1 \otimes \cdots \otimes I_{i-1} \otimes diag(E_i^{(i,i)}) \otimes I_{i+1} \otimes \cdots \otimes I_d$$

and

$$D_S = \sum_{(s,t)} \bigotimes_{i=1}^{d} diag(E_i^{(s,t)}).$$

- *For the strictly lower triangular matrix L:*

$$L = L_L + L_S$$

with

$$L_L = \sum_{i=1}^{d} I_1 \otimes \cdots \otimes I_{i-1} \otimes tril(E_i^{(i,i)}) \otimes I_{i+1} \otimes \cdots \otimes I_d$$

and

$$L_S = \sum_{(s,t)} \sum_{k=1}^{d} \left(\bigotimes_{i=1}^{k-1} diag(E_i^{(s,t)}) \right) \otimes tril(E_k^{(s,t)}) \otimes \left(\bigotimes_{i=k+1}^{d} E_i^{(s,t)} \right).$$

- *For the strictly upper triangular matrix U:*

$$U = U_L + U_S$$

with

$$U_L = \sum_{i=1}^{d} I_1 \otimes \cdots \otimes I_{i-1} \otimes triu(E_i^{(i,i)}) \otimes I_{i+1} \otimes \cdots \otimes I_d$$

and

$$U_S = \sum_{(s,t)} \sum_{k=1}^{d} \left(\bigotimes_{i=1}^{k-1} diag(E_i^{(s,t)}) \right) \otimes triu(E_k^{(s,t)}) \otimes \left(\bigotimes_{i=k+1}^{d} E_i^{(s,t)} \right).$$

Here, $tril(B)$ and $triu(B)$ denote the strict lower and strict upper triangular part of the matrix B, respectively.

Proof. See [69, Theorem 2.1, Lemma A.6–A.8]. □

Remark 6.3. According to Lemma 6.2, L, D and U from (6.1) can be represented by sums of tensor products and thus be stored efficiently. Note that the results of Lemma 6.2 also hold for general Kronecker structured matrices and are not restricted to generator matrices of SANs. In this case the "local part" given by the matrix A_L is omitted.

With Lemma 6.2 we can avoid the explicit representation of the iteration matrices and can compute the products Dx and $(D + L)x$ within the tensor format. But this does not allow to compute the action of the inverses of these matrices, so the next step is to compute the action of the inverse of $(D + L)$ or D approximately by an iterative method which uses matrix-vector products, e.g., GMRES. So in our tensor setting, one of the main advantages of the simple, splitting-based methods—namely that their iterations can be implemented easily and performed efficiently—is unfortunately lost. Still, it is interesting to investigate these methods in order to find out if the higher cost pays off due to better smoothing properties (e.g., in comparison to GMRES).

As in section 4.1 we again look at the magnitude of the coefficients from the linear combination of the eigenvectors of the error after three Gauss-Seidel and three weighted Jacobi steps, see Figures 6.1 and 6.2, respectively. Recall that the magnitudes of these coefficients were all one at the beginning and that all computations are done without using tensor techniques. Figure 6.1 shows that Gauss-Seidel seems to have problems with the model *overflow*. Instead of getting smaller, some of the coefficients increase. Besides we have peaks at different positions. Recall Figure 4.3, where GMRES only has problems with the coefficients corresponding to low magnitude eigenvalues. This also implies that the idea to approximate the error within a space $\mathcal{V}$ spanned by small eigenvectors for better handling the problematic parts is not so obvious here as it was in the case of GMRES. But for *simplified_kanban* and *directed_metab* we have the same behaviour as in Figure 4.3. It should be emphasized that for the model *simplified_kanban* the coefficients corresponding to the lower parts of the spectrum show an increase (they are larger than one). For a stationary method, this means that we cannot get the error arbitrarily small. But because of the nice splitting of the problematic parts and not so problematic parts, it can be motivated that this can be handled by the coarse grid correction within a multigrid method, if we approximate the error within a space $\mathcal{V}$ spanned by smaller eigenvectors. Figure 6.2 shows that Jacobi seems to have problems for all kinds of models. So therefore it is not sensible to use weighted Jacobi as smoother. We choose

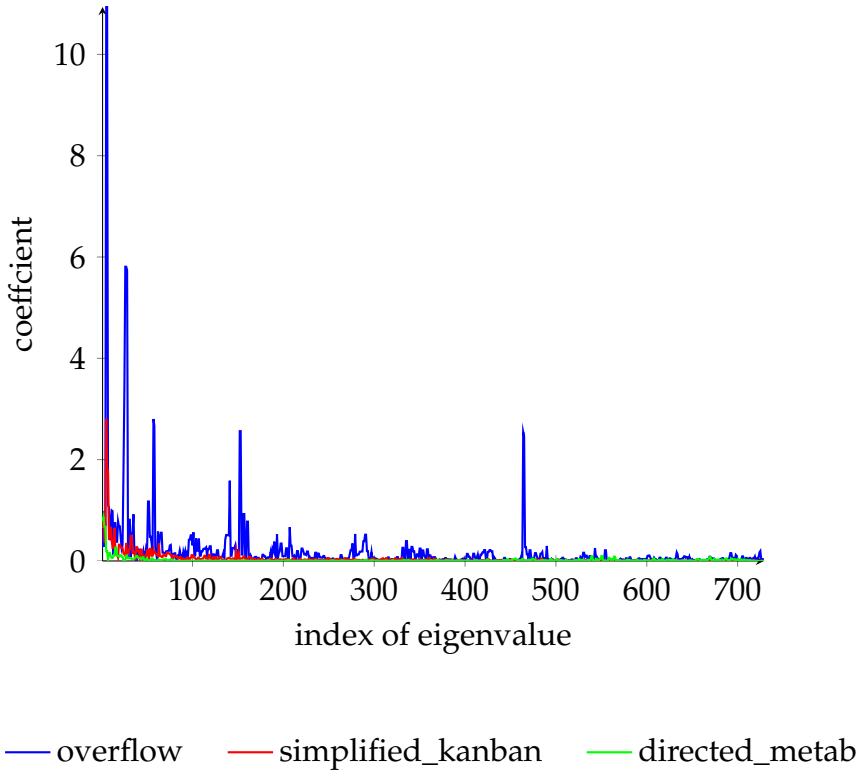


Figure 6.1: Reduction of error components belonging to different eigenvalues after three Gauss-Seidel iterations.

the relaxation parameter $\omega = \frac{2}{3}$, like it is recommended in [63] in the case of spd matrices. To be free from the argument that this behaviour is dependent on the relaxation parameter, we also used the relaxation parameters $\omega_1 = 1$ and $\omega_2 = 2/(|\lambda_{min}(D^{-1}A)| + |\lambda_{max}(D^{-1}A)|) = 2/|\lambda_{max}(D^{-1}A)|$ and observed the same behaviour, in experiments not reported here. The second relaxation parameter ω_2 is an adapted version from the spd case, where the optimal relaxation parameter is

$$\omega_{opt} = \frac{2}{\lambda_{min}(D^{-1}A) + \lambda_{max}(D^{-1}A)},$$

see [40, Section 5.2.2]. Of course, there could in principle be another relaxation parameter which is optimal for these kinds of models, but the three choices we used are the most natural, and it can be expected that other choices would not remedy the severe problems which the Jacobi method shows for these models. Due to this observation we will use GMRES and approximated Gauss-Seidel as smoothers, even though it is strongly expected that approximated Gauss-Seidel will need more computational time and cost when used as smoother than a tensorized GMRES. As mentioned, we will use an iterative method for approximately applying the inverse of $(D + L)$. To stay in the tensor setting, the al-

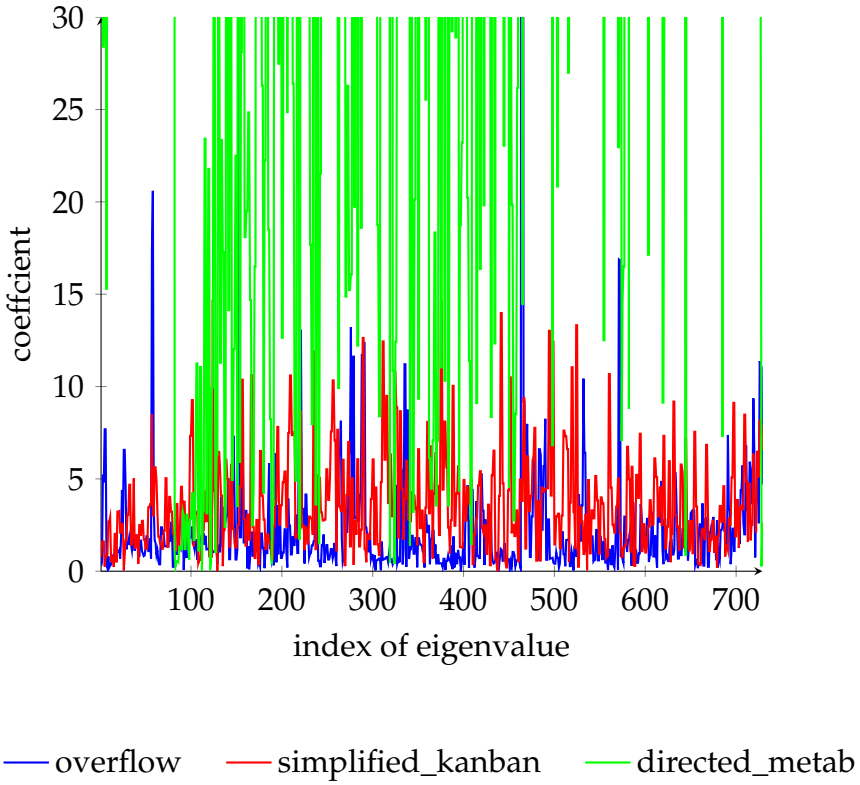


Figure 6.2: Reduction of error components belonging to different eigenvalues after three Jacobi iterations.

ternating minimal energy (AMEn) method, see [29, 30, 47, 48] , seems to be a more sensible choice than a tensorized version of GMRES. As discussed in [28], the main problem of tensorized GMRES is that accurately representing the orthogonal basis appearing in GMRES implies a high TT-rank and therefore an increase of computational time, cost and storage, in contrast to AMEn, which is an optimization-based low-rank tensor solver which will be discussed in detail in chapter 9. So for approximating the inverse of $(D + L)$, tensorized GMRES seems not very sensible. As smoother, however, this is not the case because we only use a few tensorized GMRES steps, so if we have problems with the TT-ranks, we can be sure that this is not caused by the representation of the orthogonal basis.

6.2 Set of coarse variables

For determining the set of coarse variables we do not have a direct demand from the convergence analysis. So on the one hand, our motivation will based

on Figures 4.5–4.7, where we see that we are able to approximate the error accurately with fewer degrees of freedom, because GMRES as smoother seems to produce errors that locally change slowly. On the other hand it will be based on the strong approximation property from Definition and Lemma 5.7, which gives us implicit evidence for the set of coarse variables.

First one should keep in mind that we want to minimize $\|e - Pe_c\|_{QA}$. For the relevant error e we can assume, because of Figures 4.5–4.7, that locally it only changes slowly, so the coarse set should be chosen in the following way: Recall that each index in our models from chapter 2.2 corresponds to a state. So each state $1, \dots, N$ should have a neighbour in the coarse set. The second demand is that we need enough degrees of freedom, i.e., enough coarse variables, to be able to fulfil (5.11) with a small constant K_s.

We have, however, also demands that mean that we should not choose too many coarse variables: The larger the coarse grid system, the higher the computational cost for solving it, which can make the method practically infeasible even if it has a fast convergence rate, see, e.g., [67].

The choice of coarse sets therefore has to be balanced between these concurrent demands, and neither too many nor too few coarse variables will lead to an efficient solver.

Summarizing, it is sensible to choose the coarse variables based on the geometry of the models (to ensure that all states have coarse neighbours). Here, neighbouring can either be based on the connections of the model or on the difference of the states of the submodels (so for example if we have states corresponding to substates $(1, 1, 1)$ and $(1, 2, 1)$ in a three dimensional model we consider them to be near to each other because only the second substate differs by one). The fact that the generator matrices of all considered models have a nice geometric structure, as Figures 6.3 and 6.4 show, also supports that it is reasonable to use geometric considerations when choosing the coarse set.

By now we did not take into account the tensor structure in the demands for the coarse set. Recall that the index set of our tensor $\mathcal{X}$ is of the form

$$\{1, \dots, n_1\} \times \dots \times \{1, \dots, n_d\}.$$

For keeping the tensor structure on a coarser grid, the index set of the reduced tensor $\mathcal{X}_c$ should be of the form

$$\{1, \dots, n_1^c\} \times \dots \times \{1, \dots, n_d^c\}.$$

To achieve this we demand to have a compatible coarse set in the following sense:

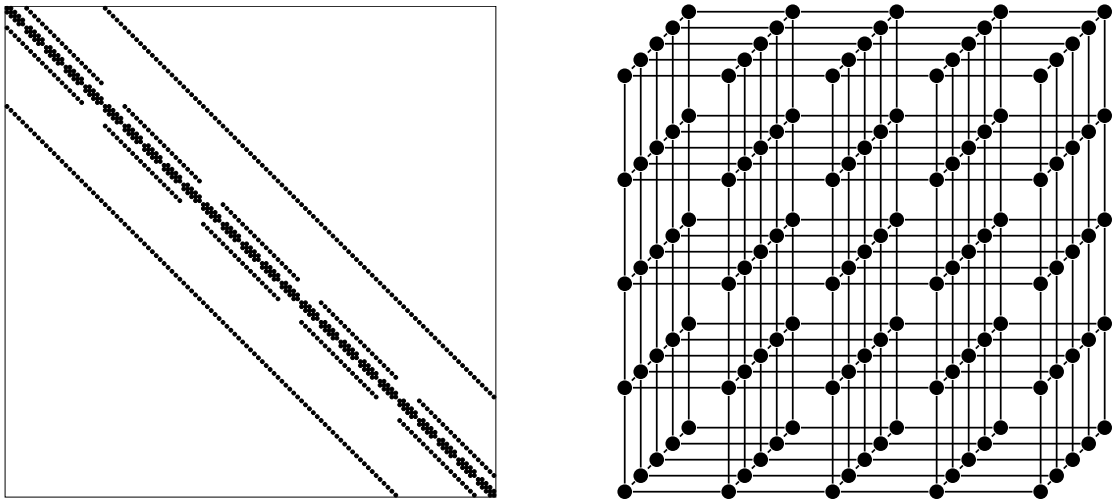


Figure 6.3: Sparsity structure (left) and corresponding geometric structure (right) for *overflow* ($d = 3, n = 5$)

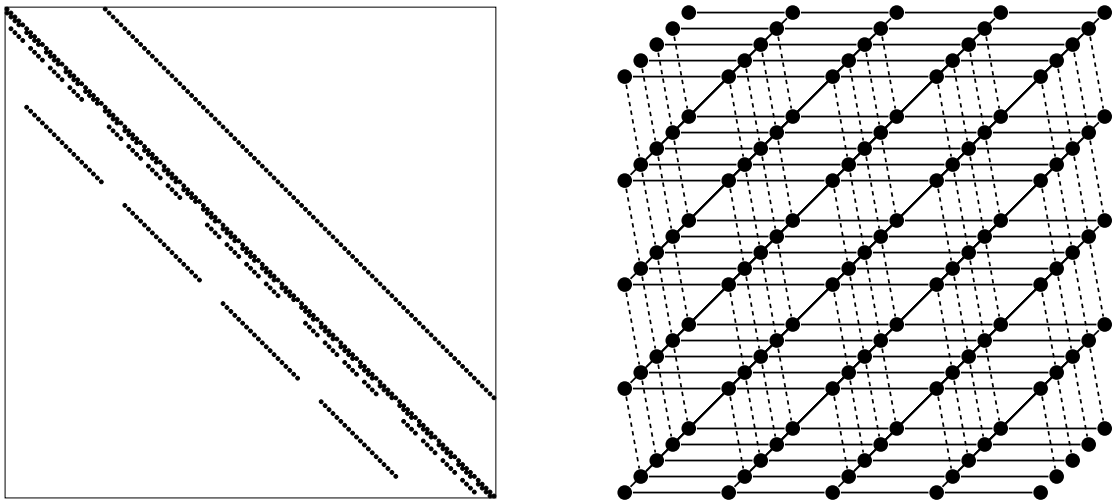


Figure 6.4: Sparsity structure (left) and corresponding geometric structure (right) for *simplified_kanban* and *directed_metab* ($d = 3, n = 5$)

Definition 6.4. A coarse set $\mathcal{C}$ is *compatible* with the structure of a tensor $\mathcal{X}$ if

$$\mathcal{C} = C_1 \times \ldots \times C_d \quad \text{and} \quad |C_i| \leq n_i. \tag{6.4}$$

We will consider three types of coarsening: full coarsening, semi-coarsening and aggregation.

6.2.1 Full coarsening

There is no coherent definition of the precise meaning of full coarsening. The main idea is to coarse in every dimension of the problem. So in this thesis we will use the following definition and meaning of full coarsening, which also satisfies the compatibility with the tensor structure in the sense of Definition 6.4. Before handing out the definition, note that for the ease of presentation, we assume in the following that all mode sizes satisfy $n_i = 2^{k_i} + 1, k_i \in \mathbb{N}$. In particular, all mode sizes are odd in this case.

Definition 6.5. *Full coarsening* for a d-dimensional problem is given via

$$\mathcal{C} = \{1, 3, \ldots, n_1\} \times \ldots \times \{1, 3, \ldots, n_d\}.$$

For illustration we show the coarse sets $\mathcal{C}$ corresponding to full coarsening for the model *overflow* with $d = 1, 2, 3$ and with mode size $n_i = 5$ in Figures 6.5–6.7, respectively. For this, we arranged the graph of the generator matrix as a regular d-dimensional grid like in the right-hand side of Figure 6.3. For the one-dimensional case each node either belongs to the coarse set or has a direct neighbour which belongs to the coarse set. For $d > 1$ this is not true any more and the distance to a coarse node can be up to d (where distance means the number of edges to arrive to a coarse node). Still, each fine node has coarse nodes which are "spatially close" to it, in the sense that the size of the whole graph grows exponentially with d while the distance to the next coarse node only grows linearly. Under the assumption that $n_i \equiv n = 2^k + 1$, full coarsening reduces the model size from $n^d = (2^k + 1)^d$ to $n_c^d = (2^{k-1} + 1)^d$, i.e., by a factor of roughly $\frac{1}{2^d}$.

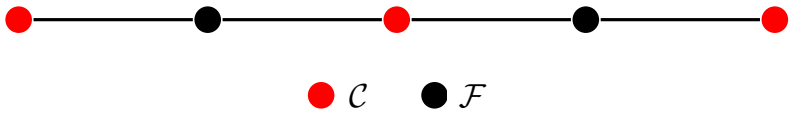


Figure 6.5: Full coarsening for *overflow* ($d = 1, n = 5$).

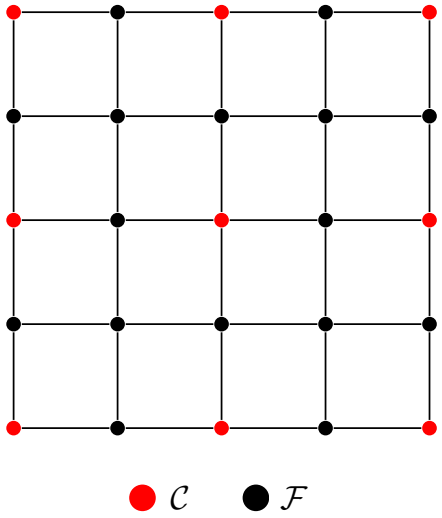


Figure 6.6: Full coarsening for *overflow* ($d = 2, n = 5$).

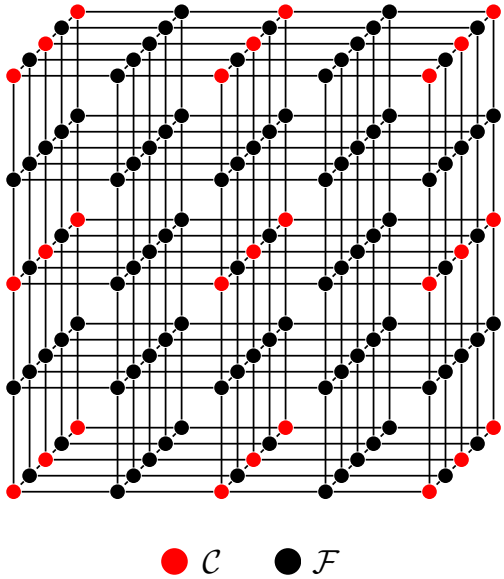


Figure 6.7: Full coarsening for *overflow* ($d = 3, n = 5$).

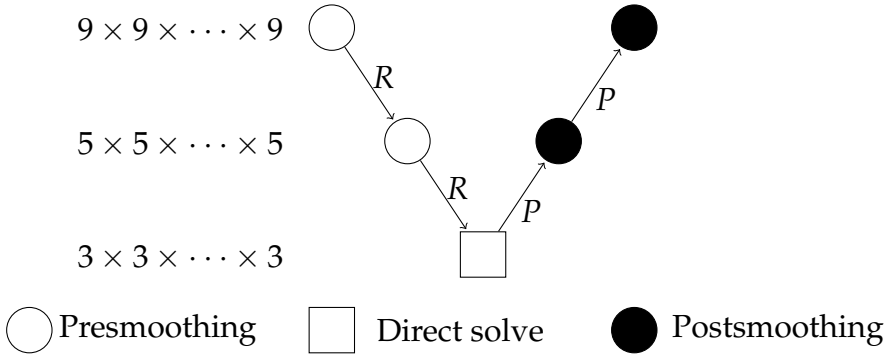


Figure 6.8: Full coarsening process for a problem with mode sizes $n = 9$.

When applying full coarsening recursively until the mode size is 3, the coarsening process looks as depicted in Figure 6.8 and corresponds to a $\log_2(n-1)$-level method. Note that the coarsest problem size is still exponential in d, unless coarsening is done until a mode size of one is reached, which does not lead to a sensible coarse grid equation. In this case the coarse grid matrix would be of size 1×1 and the individual subsystems get lost. For large values of d, this issue has to be taken into account when choosing the coarse grid solver.

6.2.2 Semi-coarsening

In contrast to full coarsening, semi-coarsening does not coarse in every direction. Its origin is from application of multigrid to two-dimensional anisotropic PDEs, i.e., PDEs where the coupling between grid points in one direction is much stronger than in the other direction. In this situation, it turns out that either coarsening in only one direction or using so-called *line smoothers* yields a much more efficient multigrid method than full coarsening and a standard smoother, see, e.g. [68, Section 5].

But for our models the motivation for semi-coarsening is a different one. With full coarsening we have observed that the coarsest system size is still exponential in d. With full coarsening we can only get rid of the exponential growth by reducing every subsystem to size one simultaneously. But this seems not sensible, because it results in a coarse grid system of size 1×1, which does not carry meaningful information any more. With semi-coarsening we get rid of the simultaneous reduction and can end up with a coarse system which only includes a fixed number d_c (instead of d) of subsystems (i.e., the other $d - d_c$ subsystems are removed from the model by reducing their size to 1).

There is of course not a single generalization of semi-coarsening to our high-dimensional tensor setting. While for $d = 2$, the states are reduced in one direc-

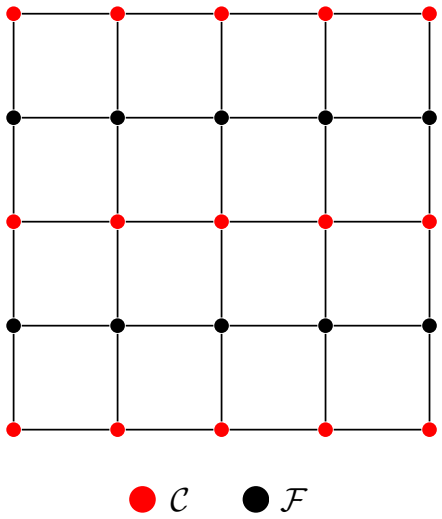


Figure 6.9: Semi-coarsening corresponding to the dimension $\{2\}$ for *overflow* ($d = 2, n = 5$).

tion and not reduced in the other direction, for $d > 2$ there are more choices, how many (and which) dimensions to reduce and how many (and which) to keep intact. Therefore, we give the following general definition of semi-coarsening here. Which choices are appropriate then depends on the model, mode sizes, dimensions etc. We again assume that the mode sizes satisfy $n_i = 2^{k_i} + 1, k_i \in \mathbb{N}$.

Definition 6.6. *Semi-coarsening* for a d-dimensional problem with respect to the dimensions $\{s_1, \ldots, s_k\}$ is given via

$$\mathcal{C} = C_1 \times \ldots \times C_d,$$

where

$$C_j = \begin{cases} \{1, 3, \ldots, n_j\} & \text{if } j \in \{s_1, \ldots, s_k\} \\ \{1, 2, \ldots, n_j\} & \text{if } j \notin \{s_1, \ldots, s_k\} \end{cases}$$

Obviously, this form of semi-coarsening is compatible with the tensor structure in the sense of Definition 6.4.

Figures 6.9 and 6.10 illustrate possible semi-coarsenings for the overflow model with $d = 2$ and $d = 3$.

Assume for example that we want to keep $d_c = 3$ subsystems intact and reduce all others to size 1. A sample coarsening process for this case is shown in Fig-

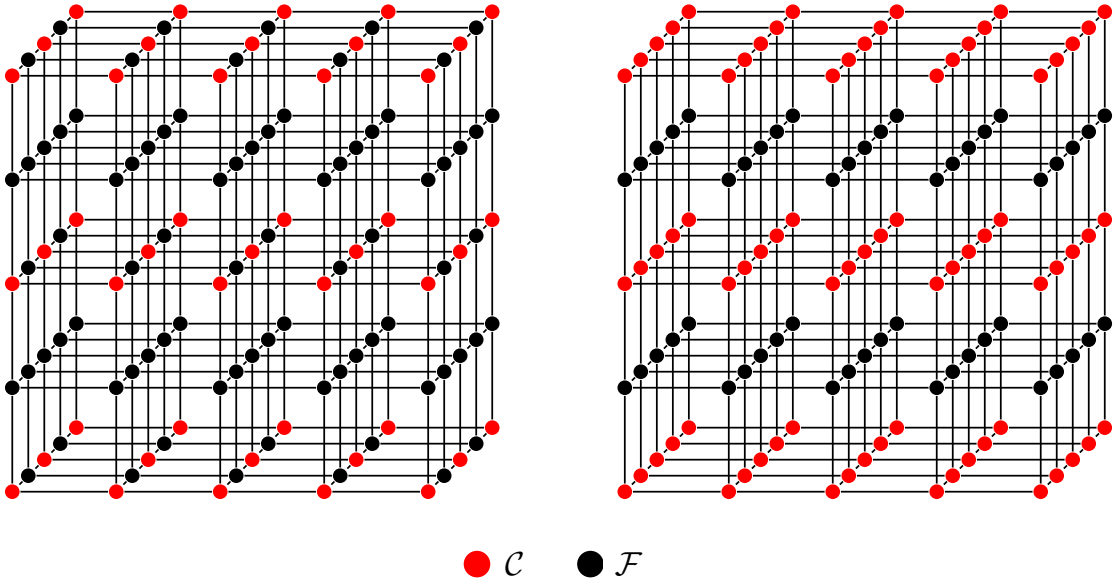


Figure 6.10: Semi-coarsening corresponding to the dimensions $\{2,3\}$ (left) and to the dimension $\{2\}$ (right) for *overflow* ($d = 3, n = 5$).

ure 6.11, where $d = 6$, $n_i \equiv n = 9$ for $i = 1, \ldots, d$ and semi-coarsening is done with respect to the dimension $\{1,3,5\}$, resulting in a coarse-grid system of size $n^{d_c} = 9^3$. In general, under the assumption that $n_i \equiv n = 2^k + 1$, the size of the system is reduced by a factor of roughly $\frac{1}{2^{d-d_c}}$ in one coarsening step, and overall, $\log_2(n-1)+2$ levels are needed to reach the coarse grid of size $(2^k+1)^{d_c}$. If the mode sizes were very large, e.g., $n = 1025$, the above approach again leads to a coarse grid system which may be too big to solve efficiently (note that d_c should not be chosen too small, so that, e.g., $d_c = 1$ is not feasible).

Summarizing, full coarsening has the disadvantage that the coarsest system size is still exponential in d and semi-coarsening—when it is applied as described above—has the disadvantage that some of the mode sizes still have their original value on the coarse grid. Therefore, full coarsening leads to a big coarse grid matrix when d is large, and semi-coarsening leads to a big coarse grid matrix when n is large. This motivates to combine the two approaches: First, reduce *all* mode sizes to a manageable value n_c by full coarsening, and then, in a second phase, eliminate all but d_c of the submodels by semi-coarsening, resulting in a coarse grid size of $n_c^{d_c}$. Note that this approach is in fact also a special case of our very general definition of semi-coarsening, where the first phase corresponds to choosing $\{s_1, \ldots, s_k\} = \{1, \ldots, d\}$. We illustrate this "combined" approach in Figure 6.12 for $n = 9, d = 6$ and a reduction to $d_c = 3, n_c = 3$ and $\{s_1, \ldots, s_k\} = \{1,3,5\}$ in the second phase of coarsening.

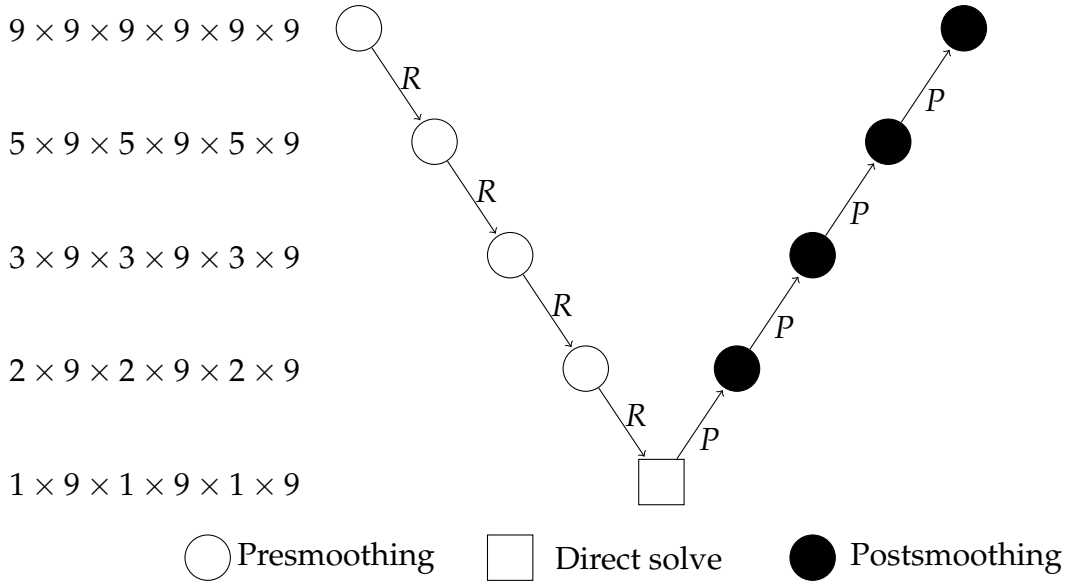


Figure 6.11: Semi-coarsening process for a problem with mode sizes $n = 9$ and $d = 6$.

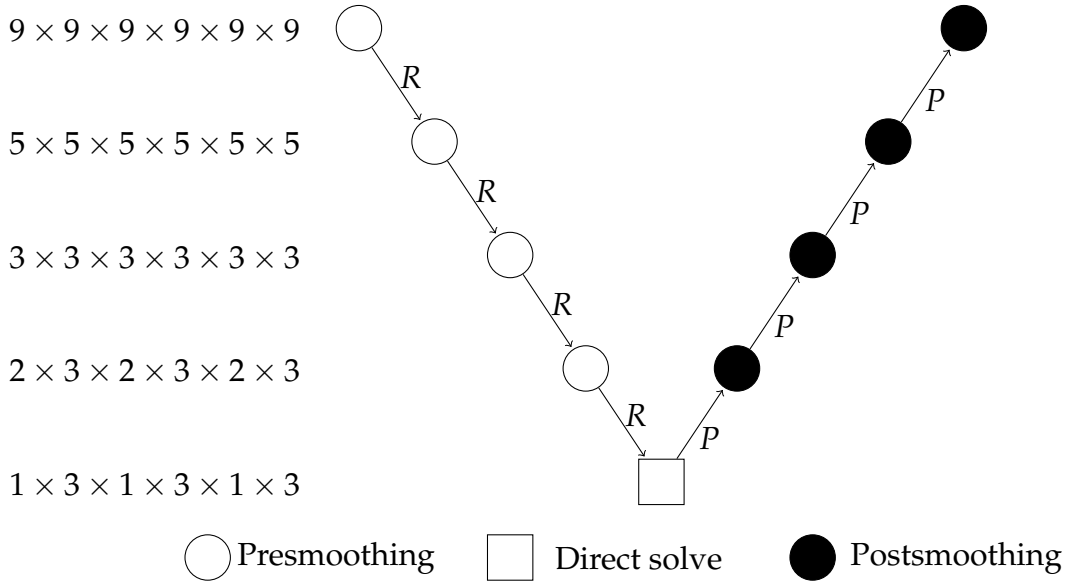


Figure 6.12: Full coarsening combined with semi-coarsening process for a problem with mode sizes $n = 9$ and $d = 6$.

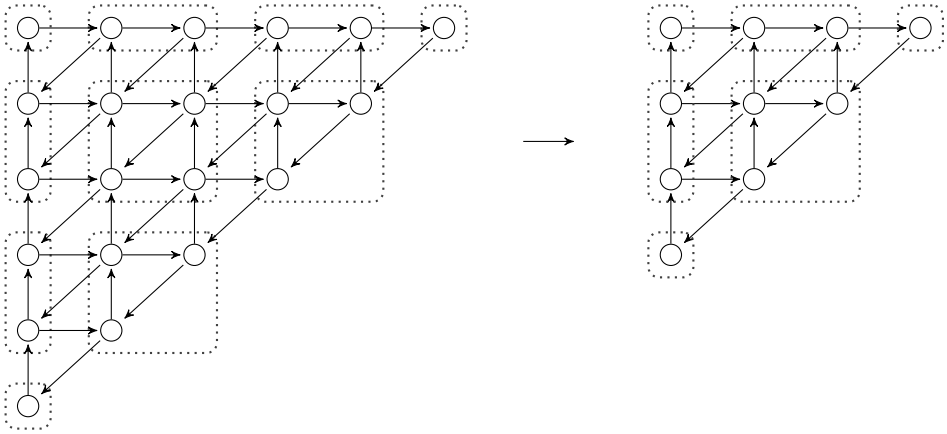


Figure 6.13: Choice of aggregates on all levels but the last for a queue with five waiting seats.

6.2.3 Aggregation

The full-coarsening and semi-coarsening approaches described in Section 6.2.1 and 6.2.2 can successfully be applied to all model problems from Section 2.2, except the model *kanban_control*. For all other models, the states of each subsystem can be arranged as a one-dimensional chain. As Figure 2.3 shows, this is clearly not possible for *kanban_control*. While it would theoretically be possible to view the graph shown in Figure 2.3 as an incomplete regular two-dimensional grid and apply full or semi-coarsening as shown in Figures 6.6 and 6.9 to it, this leads to coarse grids with a different structure, making a recursive application difficult. We therefore consider a different approach in this section, which allows to maintain the structure of the model *kanban_control* across all grids.

The approach we will consider is an aggregation approach [10, 70–72]. Aggregation means that our coarse set is not a subset of the original states. Instead, one coarse variable is an accumulation of states of the original system (an *aggregate*), and two aggregates i and j are connected to each other if any one state in i was connected to any one state in j in the graph of the original system. This approach helps us to keep the incomplete two grid structure of each automaton on the coarser grid, when the aggregates are chosen in an appropriate way. Figure 6.13 illustrates, for a subsystem with $k = 5$ waiting seats, the choice of aggregates and the resulting coarse grid (as well as the choice of aggregates on this grid for a further coarsening). The grid resulting from the aggregation shown on the right of Figure 6.13 corresponds to the coarsest grid possible, a further aggregation by the same approach is not possible. To be able to coarsen up to this point, we always assume that the number of waiting seats in each subsystem (which is *not* the mode size of the system) is given as $k_i \equiv k = 2^\ell + 1$, resulting

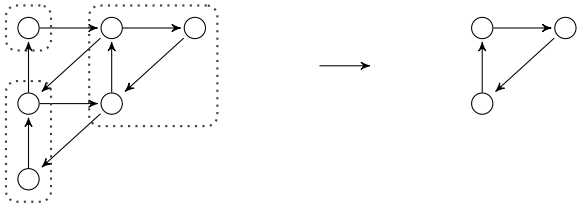


Figure 6.14: Choice of aggregates on the last level.

in a mode size of $n_i = \frac{(k+1)(k+2)}{2}$ for $i = 2, \ldots, d-1$ and $n_1 = n_d = k+1$.

This approach is also compatible with the tensor structure because for each automaton we obtain a coarse set C_i and therefore the whole coarse set $\mathcal{C}$ is given as $\mathcal{C} = C_1 \times \ldots \times C_d$.

Thus, this approach allows to reduce the size of the overall system from $N = (k+1)^2 \left(\frac{(k+1)(k+2)}{2}\right)^{d-2}$ to $3^2 \cdot 6^{d-2}$. A further reduction to $2^2 \cdot 3^{d-2}$ is possible by considering a modified aggregation on the second to last level, as shown in Figure 6.14.

6.3 Transfer operators

First we are looking for an appropriate choice for the transfer operators for all models but *kanban_control*, because there the set of the coarse variables comes from an aggregation approach and therefore the construction of the transfer operators should be treated separately. For the transfer operators we have the following crucial demands:

(i) the interpolation operator P should minimize $\|e - Pe_c\|_{QA}$ for "relevant" error vectors e,

(ii) the restriction operator R should fulfil $\mathbf{1}^T R = \mathbf{1}$,

(iii) R, P should be compatible with the tensor structure of A.

Note that the demand (ii) comes for the property (2.5) of the generator matrix A from our models, namely that $\mathbf{1}^T A = \mathbf{0}$, and this property should be kept for the coarse grid matrix A_c (recall that $A_c = RAP$.) Besides, the smallest left singular vector of A is a constant vector (because $A^T \mathbf{1} = \mathbf{0}$), so if $\mathbf{1}^T R = \mathbf{1}$ holds the constant vector is represented exactly on the coarser grid. In [13, Example 1.1] it is motivated to preserve the smallest left singular vector exactly in

a non-symmetric multigrid setting. To fulfil (iii) it is necessary that the transfer operators P, R are given via the formula

$$P = \sum_k \bigotimes_i P_i^{(k)} \quad \text{and} \quad R = \sum_k \bigotimes_i R_i^{(k)}, \quad i = 1, \ldots, d \tag{6.5}$$

with $P_i^{(k)} \in \mathbb{R}^{n_i \times n_i^c}$ and $R_i^{(k)} \in \mathbb{R}^{n_i^c \times n_i}$, otherwise the multiplication to obtain the coarse grid operator $A_c = RAP$ is not possible within the tensor format. For $d = 2$, one summand and $P_1^{(1)} = P_2^{(1)}$, transfer operators of this form have been investigated in [35] for solving Sylvester equations.

It is obvious that the more summands there are in P from (6.5), the smaller the value of $\|e - Pe_c\|_{QA}$ that can be achieved. But more summands in P mean higher cost when doing a matrix-vector multiplication with P, and in addition the coarse grid operator $A_c = RAP$ then has more summands in its representation than A, so the storage complexity of the coarse grid operator also increases. Consider the case that we demand only one summand, then the following proposition holds:

Proposition 6.7. *Let A of the form* (2.18) *be given, with $E_i^{(s,t)} \in \mathbb{R}^{n_i \times n_i}$. Let $P = \bigotimes_{i=1}^d P_i$ and $R = \bigotimes_{i=1}^d R_i$ with $P_i \in \mathbb{R}^{n_i \times n_i^c}$ and $R_i \in \mathbb{R}^{n_i^c \times n_i}$ where $n_i^c \leq n_i$. Then the corresponding Petrov-Galerkin operator satisfies*

$$RAP = \sum_{(s,t)} \bigotimes_{i=1}^d R_i E_i^{(s,t)} P_i - \sum_{(s,t) \neq (i,i)} \bigotimes_{i=1}^d R_i D_i^{(s,t)} P_i.$$

Proof. The result follows directly by repeated application of the linearity of the Kronecker product. □

So Proposition 6.7 yields that the task of constructing interpolation and restriction operators becomes a "local" task, i.e., part P_i of the interpolation matrix P corresponds the ith subsystem. Recall the construction of the coarse set (discussed in section 6.2) and Definition 6.4 in particular, where we also motivated to construct the set of coarse variables "locally". Therefore the construction of the set of coarse variables matches with the demand that the transfer operators should be given as

$$P = \bigotimes_{i=1}^d P_i, \quad R = \bigotimes_{i=1}^d R_i.$$

Thus we only have to consider how the values of each entry of each P_i, R_i can be constructed.

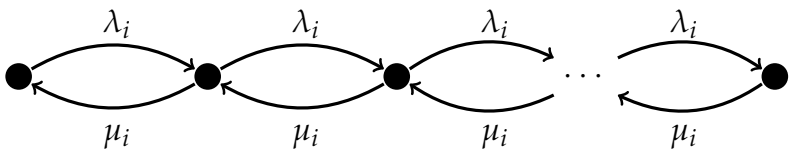


Figure 6.15: Structure and transition probabilities of the one-dimensonal *overflow* model.

6.3.1 Transfer operators in case of full coarsening

We begin by describing the construction for the case that the coarse variable are chosen by full coarsening, see section 6.2.1. Let us first concentrate on the construction of P. For this we consider first the model *overflow* and recall that each automaton $\mathcal{A}_i$ can be described by a one-dimensional Markov chain as Figure 6.15 illustrates. The first approach is *linear interpolation*. For each P_i we therefore look at the directed graph corresponding to the ith subsystem, see Figure 6.15.

To obtain an indexing of the columns of P_i we define the mapping

$$\pi_i : C_i \longrightarrow \{1, \ldots, n_i^c\} \tag{6.6}$$

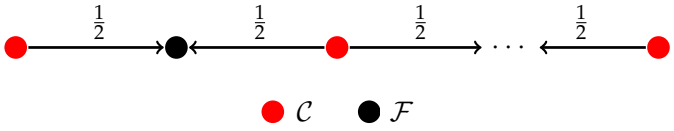
which numbers the coarse variables consecutively. Using this notation we choose each entry $p_{k,\ell}^{(i)}$ of P_i as

$$p_{k,\ell}^{(i)} = \begin{cases} 1/deg_{C_i}(k) & k \text{ connects to } \pi_i^{-1}(\ell) \\ 0 & \text{otherwise} \end{cases}$$

where $k = 1, \ldots, n_i$, $\ell = 1, \ldots, |C_i|$ and $deg_{C_i}(k)$ is the number of coarse variable which are incident with k (including k itself, if it is chosen as a coarse variable). The resulting P_i and the *interpolation graph*, i.e., the graph corresponding to matrix P_i, are shown in Figure 6.16.

For a two-dimensional system we get the interpolation matrix P as $P = P_1 \otimes P_2$, the resulting interpolation graph is given in Figure 6.17 for $n_1 = n_2 = 5$. Observe that the resulting interpolation structure is a generalization of the one-dimensional linear interpolation, i.e., each variable interpolates in equal parts from all surrounding coarse variables. This is a consequence of the definition of the Kronecker product and it is obvious that this also holds for $d > 2$.

Another choice for computing the interpolation matrix P_i is *direct interpolation* [62, 67] . In contrast to linear interpolation, the entries of the submatrix $E_i^{(i,i)}$ are involved here and the direction of the edges in the graph of each subsystem $\mathcal{A}_i$



$$P_i = \begin{pmatrix} 1 & & & & & & \\ \frac{1}{2} & \frac{1}{2} & & & & & \\ & 1 & & & & & \\ & \frac{1}{2} & \frac{1}{2} & & & & \\ & & \cdots & & & & \\ & & & 1 & & & \\ & & & \frac{1}{2} & \frac{1}{2} & & \\ & & & & & 1 \end{pmatrix}$$

Figure 6.16: Interpolation structure (left) and corresponding interpolation matrix P (right) for one-dimensional *overflow* model.

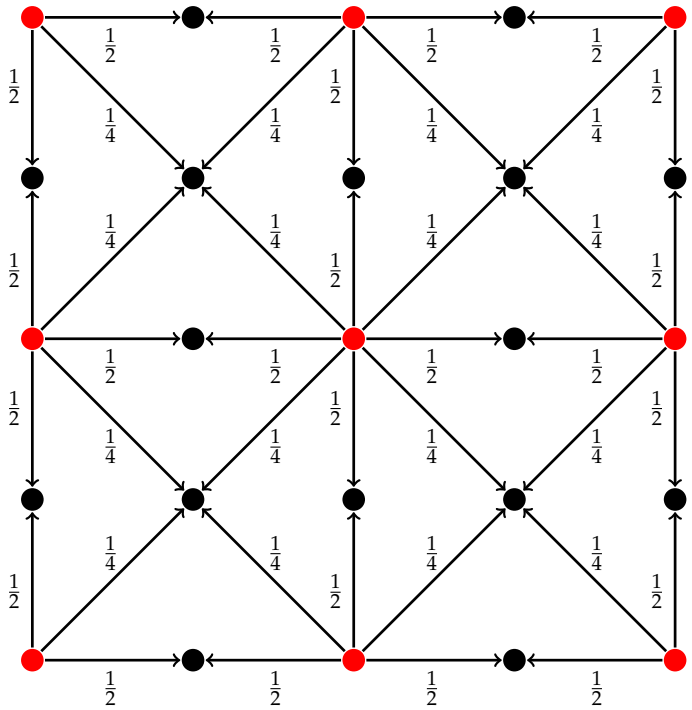


Figure 6.17: Interpolation structure for a two-dimensional *overflow* model in case of full coarsening ($n_1 = n_2 = 5$).

is important. Each entry $p_{k,\ell}^{(i)}$ of P_i is given via the formula

$$p_{k,\ell}^{(i)} = \frac{E_i^{(i,i)}(k, \pi_i^{-1}(\ell))}{E_i^{(i,i)}(k,k)}, \quad k = 1, \ldots, n_i \quad \text{and} \quad \ell = 1, \ldots, |C_i|. \tag{6.7}$$

Note that from (6.7) it follows that $p_{k,\ell}^{(i)} = 0$ if there is no edge from $\pi_i^{-1}(\ell))$ to k. For the model *overflow* the structure of the interpolation graph for directed interpolation is the same as for linear interpolation, because of the symmetry of the structure of the one-dimensional Markov chain of each subsystem $\mathcal{A}_i$.

For our realization of direct interpolation it is necessary that the matrix $E_i^{(i,i)}$ exists for each i, which means that there exist local transitions in the model which are independent from the other automata. This is not fulfilled for each model from section 2.2. So we could use direct interpolation for the subsystems which have local transitions, and linear interpolation for the others. E.g., for *direct_metab* we could do direct interpolation for the first and last automaton (i.e., P_1 is given by direct interpolation with respect to $E_1^{(1,1)}$ and P_d is given by direct interpolation with respect to $E_d^{(d,d)}$), and linear interpolation for all other automata. In numerical experiments, it turned out, however, that in this case it is better to only use direct interpolation for P_1. A possible explanation for this is that due to the directed nature of the considered problems, events in the first automaton influence all subsequent automata, while events in the last automaton do not have an influence on any other automaton. Therefore, it might not be reasonable to introduce information into the interpolation operator which includes weights based on the last automaton, which then would influence all automata due to the Kronecker structure of P.

By now we only discussed the computation of P. The crucial demand for R is $\mathbf{1}^T R = \mathbf{1}^T$, which is obviously fulfilled for the transpose of linear interpolation, but is in general not fulfilled by direct interpolation. Therefore, we construct R_i as the transpose of linear interpolation for each subsystem for each model. The property that all columns have sum one is then inherited by the matrix $R = \bigotimes_{i=1}^{d} R_i$ because

$$(\mathbf{1} \otimes \cdots \otimes \mathbf{1})^T \bigotimes_{i=1}^{d} R_i = \mathbf{1}^T R_1 \otimes \cdots \otimes \mathbf{1}^T R_d = (\mathbf{1} \otimes \cdots \otimes \mathbf{1})^T.$$

6.3.2 Transfer operators in case of semi-coarsening

When using semi-coarsening with respect to some dimensions $\{s_1, \ldots, s_k\}$, the mode sizes n_j with $j \notin \{s_1, \ldots, s_k\}$ are not reduced. To reflect this in the in-

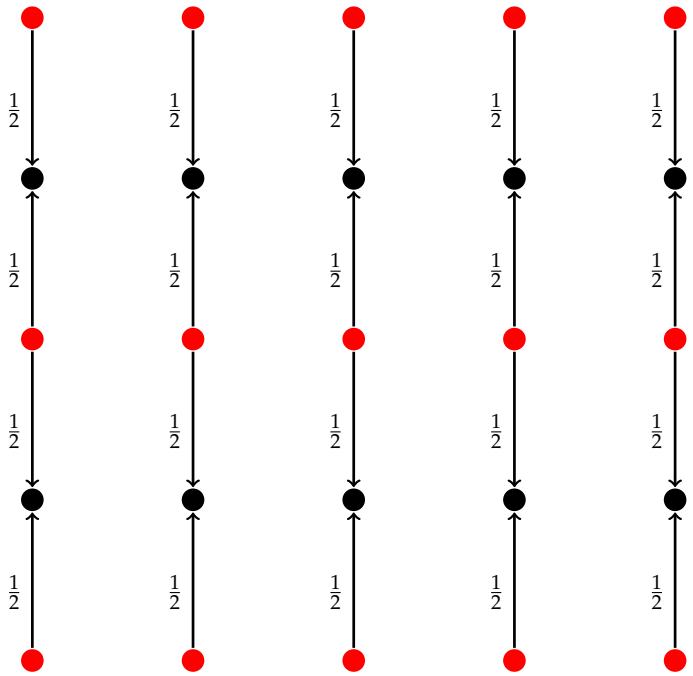


Figure 6.18: Interpolation structure for a two-dimensional *overflow* model in case of semi-coarsening with respect to dimension $\{2\}$ ($n_1 = n_2 = 5$).

terpolation and restriction operator formula (6.5), we choose the corresponding matrices P_j, R_j as identity matrices of size $n_j \times n_j$. The other matrices P_i, R_i for $i \in \{s_1, \ldots, s_k\}$ are chosen via linear or direct interpolation as described in section 6.3.1. We again illustrate the resulting interpolation structure for a two-dimensional problem with $n_1 = n_2 = 5$ with semi-coarsening with respect to dimension $\{2\}$. This illustration is shown in Figure 6.18. Note that the crucial property $\boldsymbol{1}^T R = \boldsymbol{1}^T$ for the restriction operator still holds when using this approach.

6.3.3 Transfer operators in case of coarsening by aggregation

As discussed in section 6.2.3, for the model *kanban_control* our coarse set is an accumulation of aggregates instead of coarse variables, which we denote by $\widehat{C}_\ell^{(i)}$. So each C_i includes a finite number m_i of aggregates $\widehat{C}_\ell^{(i)}, \ell = 1, \ldots, m_i$ and they satisfy the following two conditions:

i) $\widehat{C}_k^{(i)} \cap \widehat{C}_\ell^{(i)} = \emptyset, \quad k \neq \ell,$

ii) $\bigcup_{\ell=1}^{m_i} \widehat{C}_\ell^{(i)} = \{1, \ldots, n_i\}.$

For P_i this means that each column corresponds to one aggregate and can be represented by the entries

$$p_{k,\ell}^{(i)} = \begin{cases} 1 & k \in \widehat{C}_\ell^{(i)} \\ 0 & \text{else} \end{cases}, \quad k = 1, \ldots, n_i, \quad \ell = 1, \ldots, m_i.$$

This corresponds to interpolation with constant basis functions, i.e., interpolation assigns the value of a variable corresponding to the aggregate $\widehat{C}_\ell^{(i)}$ on the coarse grid to all variables of the fine grid contained in $\widehat{C}_\ell^{(i)}$. For the restriction matrix R we choose $R_i = P_i^T$ and therefore each column of R_i belongs to one state. Because of the fact that each state belongs to exactly one aggregate it holds $\boldsymbol{1}^T R = \boldsymbol{1}^T$.

6.4 Coarse grid operator

The coarse grid operator is chosen as a Petrov-Galerkin operator, i.e, $A_c = RAP$. Finally, on the coarsest grid the residual equation has to be solved. The described full coarsening process is limited to modest values of d, because of the need for solving the problem on the coarsest grid, so we have to distinguish two cases:

(i) the coarsening results in a problem of a size for which direct solving without exploiting tensor structure is possible,

(ii) the tensor structure has also to be kept on the coarsest level, because the problem size is too big for direct solving.

In case (i), direct solving means that we use the Moore-Penrose pseudo inverse for solving the coarse grid equation in a least squares sense. In this case, we have to transform the solution of the singular coarse grid system into a TT-Tensor again afterwards.

Case (ii) appears for larger values of d, because, as mentioned before, the size of the coarse grid operator still grows exponentially in d when using full coarsening, see Figure 6.8. A suggestion to avoid this problem was to combine full coarsening with semi-coarsening. But we could also use an iterative method instead which maintains the tensor structure. Recall from the description of the approximated Gauss-Seidel smoother in section 6.1 that the low-rank tensor method AMEn can be used for this task. Therefore, in chapter 9 of this thesis, we will discuss and test AMEn as a coarse grid solver in our multigrid method.

There are several choices for each ingredient of our multigrid method, so it is not clear at first sight which choices are the best or most appropriate for solving our task. Therefore, the next chapter will include a first series of numerical experiments. There we will vary the different ingredients for the tensorized multigrid method in order to compare them and also to illustrate the efficiency of each ingredient and therefore of our method.

Summary of Chapter 6:

- Choose ingredients based on theory from chapter 5 and compatibility with tensor structure.

Smoother

- GMRES is easily implementable in TT-format.
- Representation of splitting $A = L + D + U$ is possible in TT-format.

  But: Splitting methods require the action of inverses.

  $\Rightarrow$ approximate the inverses via AMEn.

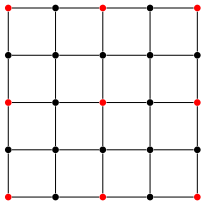
$\Rightarrow$ GMRES and Gauss-Seidel will be used.

Set of coarse variables

- Coarse set $\mathcal{C}$ should be compatible with the tensor structure of $\mathcal{X}$, i.e.,

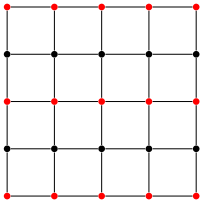
$$\mathcal{C} = C_1 \times \ldots \times C_d \quad \text{and} \quad |C_i| \leq n_i.$$

- Full coarsening and semi-coarsening lead to compatible coarse sets:



$\oplus$ deals with large n

$\ominus$ coarsest grid size n_c^d



$\oplus$ can reduce d

$\ominus$ problems with large n

Transfer operators

- Transfer operators are chosen of the form

$$P = \bigotimes_i P_i \quad \text{and} \quad R = \bigotimes_i R_i, \quad i = 1, \ldots, d.$$

- Petrov-Galerkin operator RAP has the same tensor structure as A:

$$RAP = \sum_{(s,t)} \bigotimes_i R_i E_i^{(s,t)} P_i - \sum_{(s,t)\neq(i,i)} \bigotimes_i R_i D_i^{(s,t)} P_i.$$

- Linear restriction for R_i to achieve $\boldsymbol{1}^T R = \boldsymbol{1}^T$.
- Linear or direct interpolation corresponding to ith subsystem for P_i.

Coarse grid operator

- If the coarsest system is small enough solve it directly.
- Otherwise use low-rank methods, e.g., AMEn.

CHAPTER 7

Experimental comparison of multigrid ingredients

In this chapter, we perform some first numerical tests in order to decide which of the different ingredients presented in chapter 6 yield the best results for some of the model problems from section 2.2. The results of these experiments are then used as guidance to build an optimized multigrid method for further numerical experiments in chapters 8 and 9, where all model problems (but *kanban_control*) are considered and the method is compared to other techniques for computing the stationary distribution of tensor structured Markov chains.

All tests are performed in MATLAB 2015a using the *TT*-Toolbox 2.2 [58] on a Linux PC with 32 GB RAM and an Intel Core i7-4770 processor with four cores and a clock frequency of 3.4 GHz.

7.1 Implementation details

In chapter 3 we discussed the effect of basic linear algebra operations on the *TT*-ranks and motivated that truncation after these operations is necessary. In our *V*-cycle, Algorithm 4.2, we therefore need to truncate after line 5, 6, 11 and 12. Recall that multiplying with P or R does not increase the *TT*-ranks as both operators have *TT*-ranks 1. The resulting *V*-cycle in our tensorized multigrid method is summarized in Algorithm 7.1.

The truncation function from the *TT*-Toolbox allows to input the desired truncation accuracy `tt_tol` as well as the maximum allowed rank `tt_maxr` . In the following we want to discuss how these two input parameters should be cho-

Algorithm 7.1: Tensorized multigrid V-cycle

$\mathcal{X}^{(\ell)} = \text{TensorMG}(\mathcal{F}^{(\ell)}, \mathcal{X}^{(\ell)}, \texttt{max\_rank})$
if *coarsest grid is reached* **then**
  Solve the coarse grid system $A^{(\ell)}\mathcal{X}^{(\ell)} = \mathcal{F}^{(\ell)}$ in a least squares sense
else
  Perform ν_{pre} sm. steps for $A^{(\ell)}\mathcal{X}^{(\ell)} = \mathcal{F}^{(\ell)}$ with initial guess $\mathcal{X}^{(\ell)}$
  Truncate $\mathcal{X}^{(\ell)}$ to maximal TT-rank `max_rank`
  Compute the residual $\mathcal{R}^{(\ell)} \leftarrow \mathcal{F}^{(\ell)} - A^{(\ell)}\mathcal{F}^{(\ell)}$
  Truncate $\mathcal{R}^{(\ell)}$ to maximal TT-rank `max_rank`
  Restrict $\mathcal{F}^{(\ell+1)} \leftarrow R^{(\ell)}\mathcal{R}^{(\ell)}$
  $\mathcal{X}^{(\ell+1)} \leftarrow \mathbf{0}$
  $\mathcal{E}^{(\ell+1)} = \text{TensorMG}(\mathcal{F}^{(\ell+1)}, \mathcal{X}^{(\ell+1)}, \texttt{max\_rank})$
  Interpolate $\mathcal{E}^{(\ell)} \leftarrow P^{(\ell)}\mathcal{E}^{(\ell+1)}$
  $\mathcal{X}^{(\ell)} \leftarrow \mathcal{X}^{(\ell)} + \mathcal{E}^{(\ell)}$
  Truncate $\mathcal{X}^{(\ell)}$ to maximal TT-rank `max_rank`
  Perform ν_{post} sm. steps for $A^{(\ell)}\mathcal{X}^{(\ell)} = \mathcal{F}^{(\ell)}$ with initial guess $\mathcal{X}^{(\ell)}$
  Truncate $\mathcal{X}^{(\ell)}$ to maximal TT-rank `max_rank`
end

sen. To keep the complexity low, the TT-ranks should be as small as possible while at the same time the desired accuracy should be obtainable. Of course we know neither the maximal rank of the unknown solution nor the maximal rank which our method needs to compute this solution. Note that these ranks can differ and clearly the first one is a lower bound for the second one. But the following observation from [35, Section 6.4] allows to deal with the problem easily: For a fixed maximum rank `tt_maxr` the method behaves as if it would perform its operations with a lower machine precision. This tells us that after a certain accuracy is reached the method would stagnate, indicating that the currently used rank is too small if the desired accuracy is not reached. We will use this occurring stagnation to increase `tt_maxr`. Therefore we start the iteration with `tt_maxr`$= 15$ and increase this value by a factor of $\sqrt{2}$ as soon as we observe stagnation. To characterize stagnation we compare the residual norms after two consecutive V-cycles. If they differ by less than ten percent, we interpret this as a signal of stagnation.

The `tt_tol` for truncating a tensor $\mathcal{X}$ on the fine grid in the kth V-cycle is chosen via the formula

$$\texttt{tt\_tol} = \min\{10^{-3}, \frac{10 \cdot \|\mathcal{X}\|_2 \cdot \|\mathcal{R}_{k-1}\|_2}{\|\mathcal{X}_{k-1}\|_2}\}, \tag{7.1}$$

where $\mathcal{X}_{k-1}$ is the iterate from the $k-1$st V-cycle and $\mathcal{R}_{k-1}$ is its residual. The reasoning behind this is as follows: The factor $\|\mathcal{X}\|_2/\|\mathcal{X}_{k-1}\|_2$ is used to relate the truncation accuracy to the norm of the iterate from the last V-cycle in order to take into account that the vectors occurring in the method have largely different norms. The factor $\|\mathcal{R}_{k-1}\|_2$ adapts the truncation accuracy to the approximation quality reached so far, i.e., the closer the current iterate lies to the solution, the stricter the accuracy requirement for the truncation is chosen, see [47].

For the initial guess for our experiments we use the nice advantage of the multigrid setting that we can cheaply compute the right singular vector corresponding to the smallest singular value of the coarsest grid matrix. This vector can then be interpolated to the finest grid and can be used as starting vector. Simple experiments, not reported here, show that this leads to much better results than using, e.g., a random initial guess or the scaled vector of all ones.

We apply ten pre- and post smoothing steps on each level and for solving the coarsest grid system we use the Moore-Penrose pseudo inverse which is computed once by the built-in Matlab function `pinv` and then applied to the coarse grid right-hand side in each iteration. The reasoning for using the Moore-Penrose pseudo inverse on the coarsest system is that we cannot guarantee that the coarsest system is consistent, i.e., that the right-hand side lies in the range of the coarse grid operator. In this case there is no vector which solves the linear system exactly and we instead aim for solving it in a least squares sense, which can be achieved by using the Moore-Penrose pseudo inverse, see [3].

We stop the iteration when the residual norm is smaller by a factor of 10^{-2} than the residual norm of the tensor of all ones (scaled so that the sum of its entries is one). The value of 10^{-2} might appear to be large at first sight. Note however that the iterates are scaled such that their entries sum to 1 after each iteration which means that the 2-norm of the iterates is very small for large problem sizes. So this value corresponds to a strict absolute accuracy. In addition this value leads to about the same accuracy as what is demanded for the considered model problems in [14, 15]. Besides, we stop when our maximal rank is larger than 120, a computation time of 3600 seconds is exceeded or 30 V-cycles have been performed. In these cases the tests are considered failed.

7.2 Numerical tests for different smoothers and interpolation operators

To get a first impression of our method we again take the models *overflow*, *simplified_kanban* and *directed_metab* from chapter 2.2 like for testing GMRES in

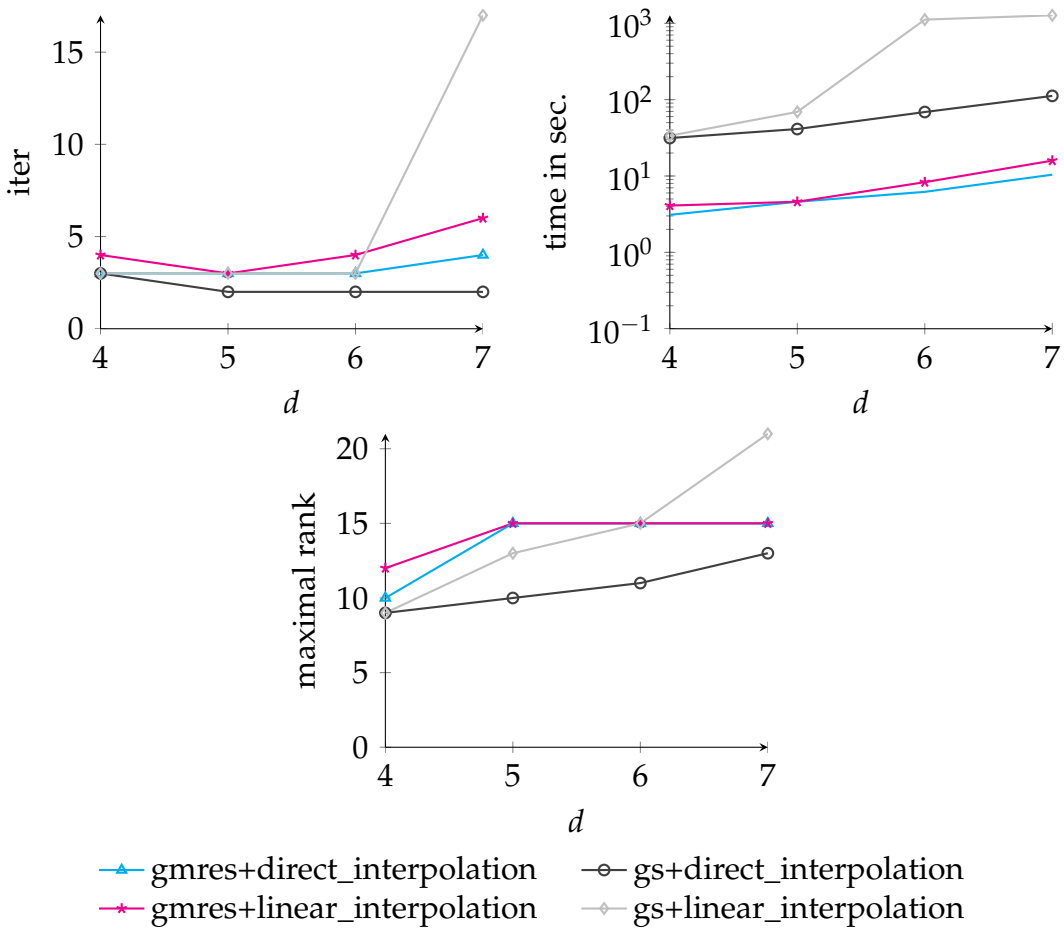


Figure 7.1: Number of iterations, running time and maximal rank for model *overflow* with mode size $n = 17$ and varying dimension d.

chapter 4. The focus of the first experiments is on the choice of the smoothing schemes and interpolation operators. First the coarse set is given by using *full coarsening* (described in chapter 6.2.1) and we repeat the coarsening process until every mode size is $n_c = 3$. In our first tests the biggest coarsest system is of size 3^7, so the pseudo inverse for solving the least squares problem on the coarsest grid can still be computed in a reasonable amount of time and therefore no other coarsening strategy which reduces the dimension is necessary here. Later on we will look at the different coarsening procedures—which we discussed in section 6.2—in section 7.3.

Recall that we motivated to use linear interpolation for the restriction operator R (see chapter 6.3). Therefore on the basis of chapter 6, what remains is to test the two smoothers (GMRES and approximated Gauss-Seidel (GS)) and the linear and direct interpolation approach for P. This results in the following four

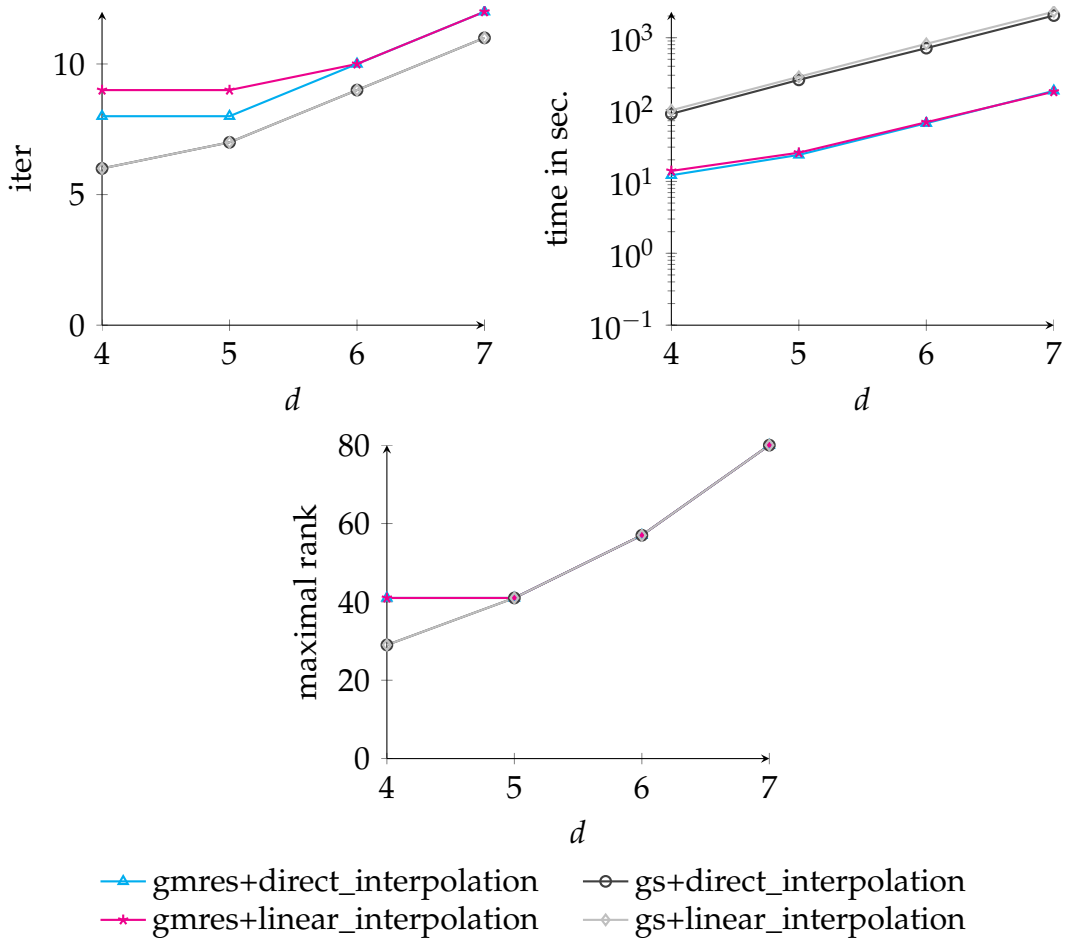


Figure 7.2: Number of iterations, running time and maximal rank for model *simplified_kanban* with mode size $n = 17$ and varying dimension d.

combinations:

- GMRES as smoother and P given via direct interpolation,
- GMRES as smoother and P given via linear interpolation,
- GS as smoother and P given via direct interpolation,
- GS as smoother and P given via linear interpolation.

For approximated Gauss-Seidel we use the AMEn code from [29, 30] which is implemented in the TT-Toolbox [58] for approximating the action of the inverse of the lower triangular matrix $D + L$. We use a maximum number of ten sweeps

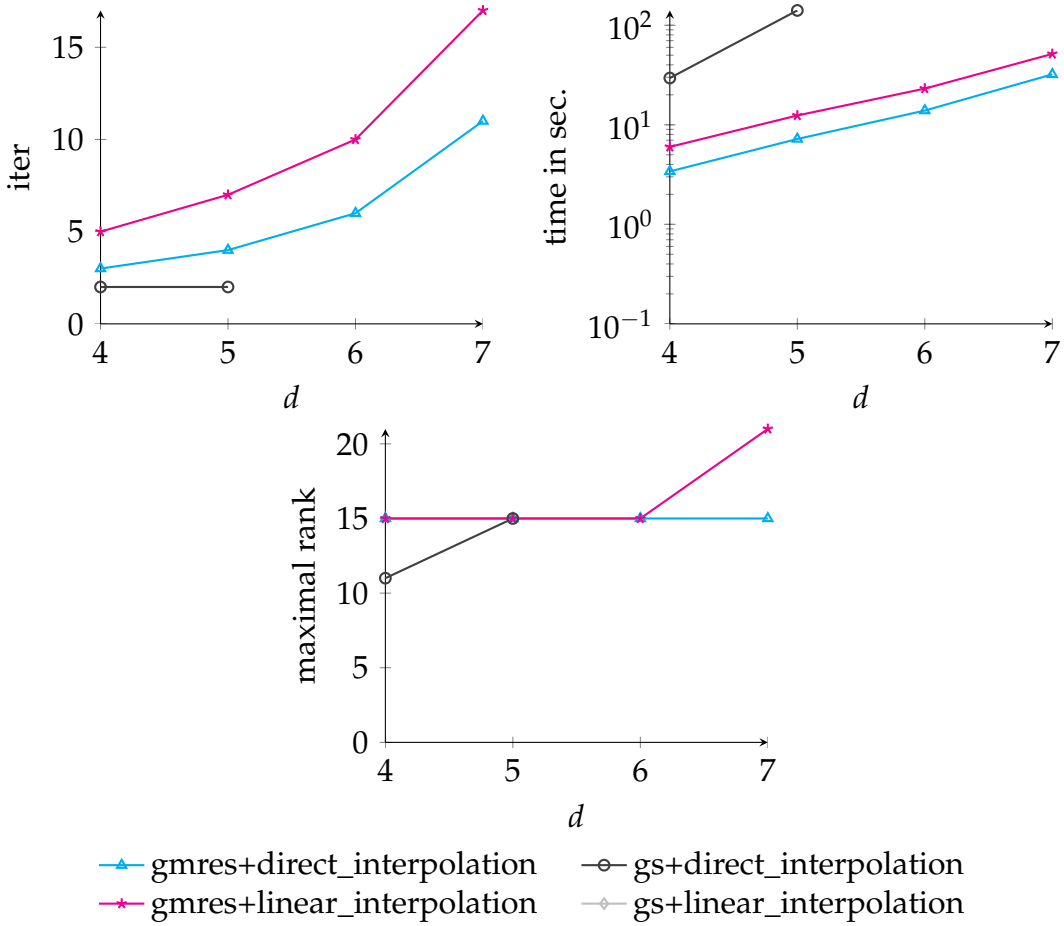


Figure 7.3: Number of iterations, running time and maximal rank for model *directed_metab* with mode size $n = 17$ and varying dimension d.

and terminate the iteration when an accuracy equal to the currently used truncation accuracy is reached. We stress again that AMEn will be discussed in detail in chapter 9. In order to illustrate the scaling behaviour of the four combinations, we first choose a capacity of 16 in each subsystem (i.e., mode sizes $n = 17$) in all models and vary the number of subsystems, i.e., the dimension d. The Figures 7.1–7.3 depict the number of iterations, the computation time and the maximal used rank of all four combinations for the three considered model problems. Based on these results, the best combination with respect to computation time seems to be GMRES with direct interpolation for all problems, even though the number of V-cycles is less when using GS as smoother with direct interpolation. The reason for this time increase is that applying $(D + L)^{-1}$ via AMEn is too expensive. The problem *directed_metab* with $d = 6$ and 7 could not be solved by both methods which use GS as smoother within one hour. It is in some cases possible to slightly improve the performance of the methods

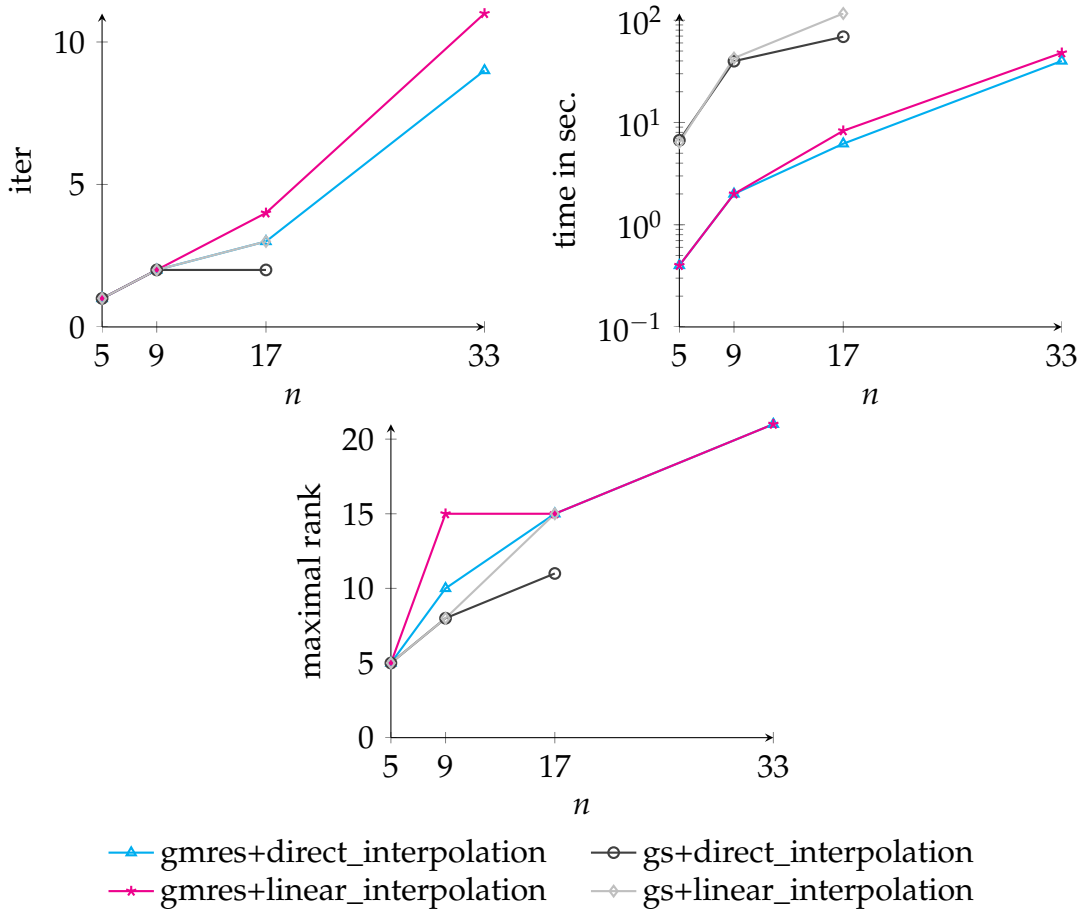


Figure 7.4: Number of iterations, running time and maximal rank for model *overflow* with dimension $d = 6$ and varying mode sizes n.

using GS by tuning the parameters of AMEn (e.g., the number of sweeps), but due to the large difference in performance in comparison to GMRES (a factor of more than 10 in terms of running time) this would not change the ranking of the methods.

Independently of the smoother, direct interpolation performs at least as good as or better than linear interpolation in all cases. For *overflow* and *simplified_kanban* the difference between both approaches is marginal (except for $d = 7$ in the *overflow* model in case of GS) but for *directed_metab* the gain in performance is clearly visible.

The maximal rank used by the different methods does not differ much in all cases, and although it is not always exactly equal, one cannot observe any particular pattern in the differences. Therefore one can assume that there is no un-

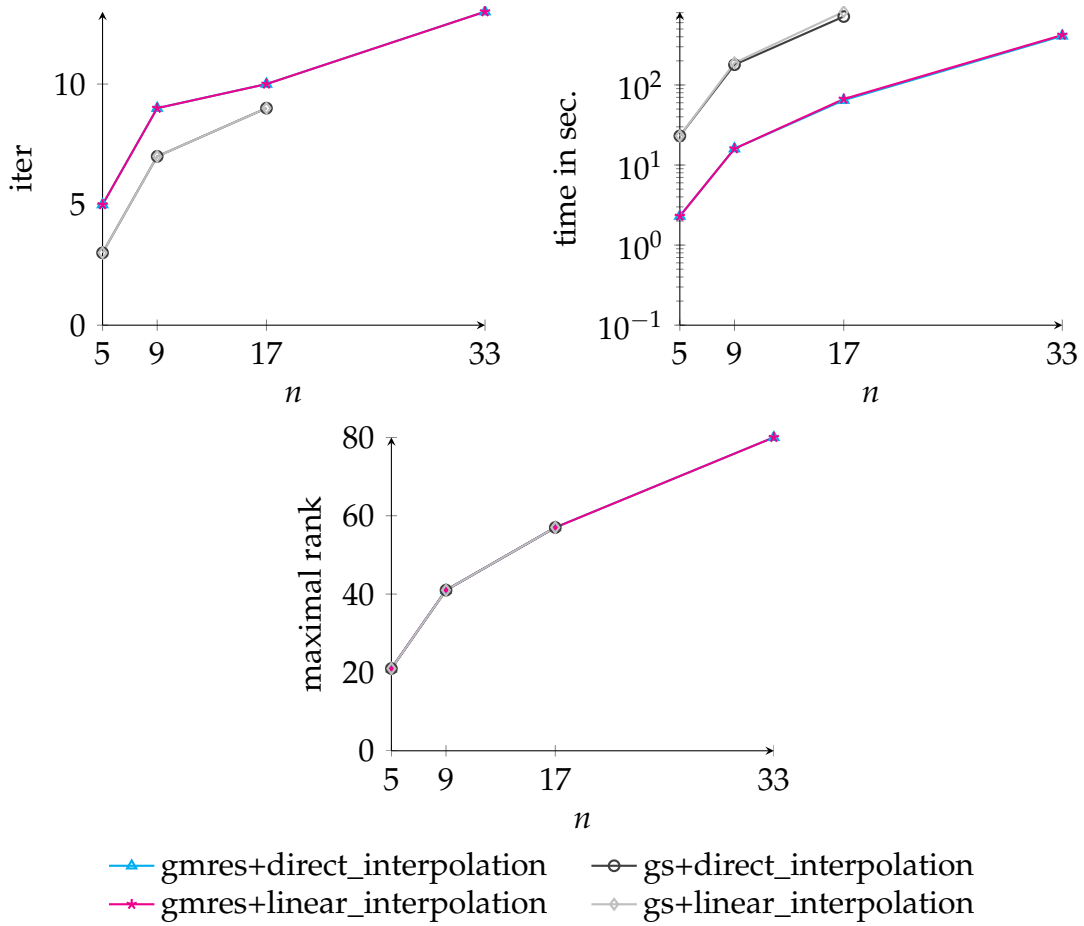


Figure 7.5: Number of iterations, running time and maximal rank for model *simplified_kanban* with dimension $d = 6$ and varying mode sizes n.

derlying reason based on the smoother or interpolation, but that the differences are mostly incidental.

Although the experiments so far show a clear tendency towards direct interpolation and GMRES as a smoother, we want to confirm them by a second series of experiments in which we vary the mode sizes. Figures 7.4–7.6 show the results of these experiments. The main differences to the previous tests is that GS fails for a larger number of test cases, especially when the mode sizes are $n = 33$. That AMEn has problems with larger mode sizes is known and was observed in [47], see also chapter 9 of this thesis. These results also agree with the previous experiments, in that direct interpolation with GMRES seems to be a good choice in all cases. Therefore we use this combination in all experiments from here on (except for the tests involving *kanban_control* in Section 7.4).

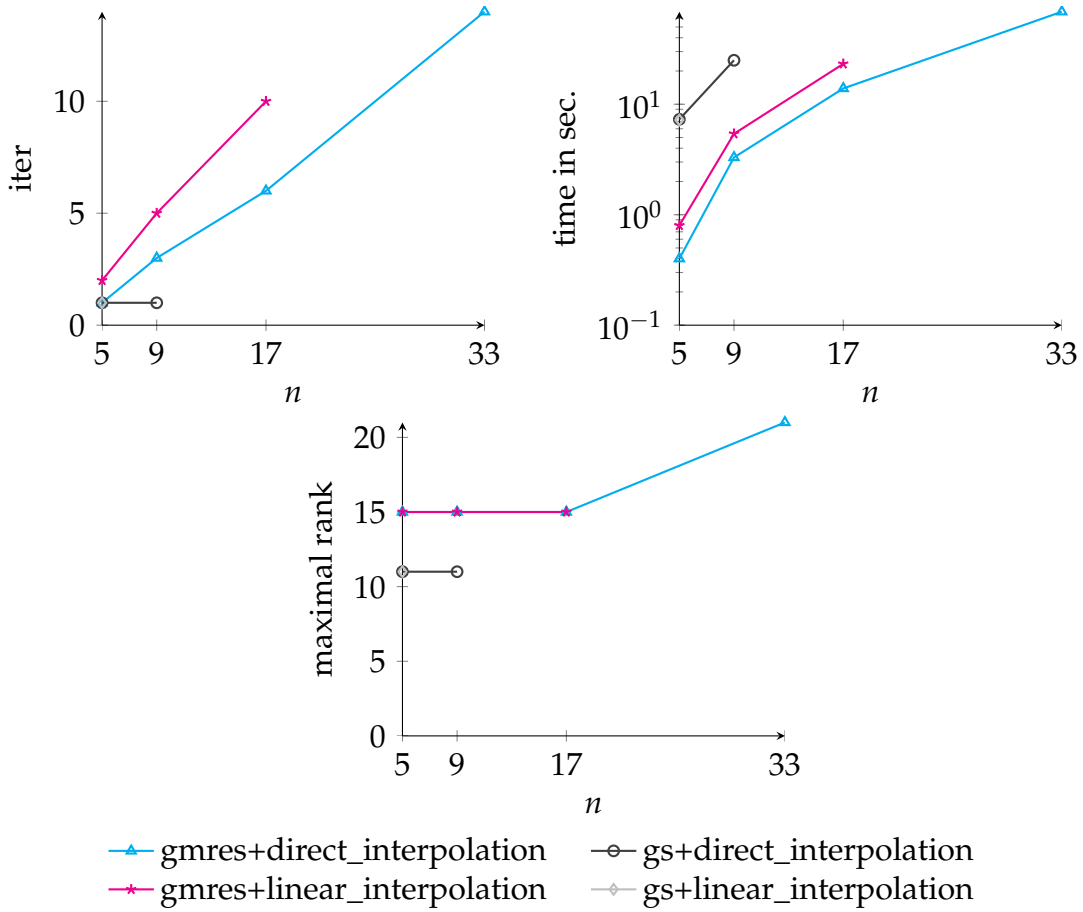


Figure 7.6: Number of iterations, running time and maximal rank for model *directed_metab* with dimension $d = 6$ and varying mode sizes n.

7.3 Numerical tests for different coarsening strategies

The next series of experiments depicts different coarsening strategies. Because of the experiments from section 7.2, we choose GMRES as smoother and compute P via direct interpolation. Besides, from the experiments from chapter 7.2, we observe that the full coarsening strategy and GMRES as smoother seem to fit nicely together. The main focus for the choice of the coarsening strategies should therefore be to get the curse of dimensionality on the coarsest grid under control. For this, a combination with semi-coarsening seems a sensible choice (see section 6.2). As mentioned in section 6.2.2, when using semi-coarsening there exist different choices for the direction in which the coarsening process can be applied. According to this, there are different ways to combine semi- and full

coarsening. Besides, the depth where the combination should be applied also has to be clarified, for this, one should aim at keeping the number of levels as low as possible while getting rid of the dimension. There are six different coarsening strategies which we will test in our next series of experiments. They are illustrated in detail in Table 7.1. Because of the experiments in section 7.2, we observed that a reduction of the mode sizes to a value of 3 via full coarsening seems to be a good choice. To better get the curse of dimensionality under control, we also want to test how the method behaves if we reduce every mode size to 2 via full coarsening. This method is labelled *full_coarsening*, $n_c = 2$. So if we assume that reducing the mode sizes to 2 delivers a coarsest system which does not include much information for the finer grids, then there is the opportunity to reduce only some of the mode sizes to 2 while the others are still reduced to a size of 3. This delivers a smaller coarsest system than if every mode size is reduced to 3. Because there is no reference point which and how many of the mode sizes should be reduced, we choose every other subsystem, starting from the first. This method is labelled *semi_coarsening 2*. The coarsening strategies labelled *semi_coarsening 3* and *semi_coarsening 4* are more aggressive. They simultaneously reduce every mode size to 1, except the first and the last one. The difference between *semi_coarsening 3* and *semi_coarsening 4* is that the first and the last subsystem are preserved through all levels in *semi_coarsening 3*, while in *semi_coarsening 4* they are reduced to a size of 3 instead. It is also interesting to consider a method where first only a subset of the subsystems is reduced in size and afterwards the remaining systems. This is done in the approach labelled *semi_coarsening 1*. Thereby we can split the method into two parts, where in the first part every other subsystem is reduced to size 2 and in the second part the remaining systems are reduced to size 5. This allows to also test whether it is reasonable to leave some systems at a larger size than the others.

Figures 7.7 and 7.8 illustrate these experiments for the model *overflow*. For the models *simplified_kanban* and *directed_metab* we get similar conclusions as for the model *overflow*, so that we do not list them here. Besides, we do not show the needed TT-ranks in our methods for these tests, because they do not differ noticeably between each other.

Figure 7.7 shows that most of the methods behave in the same way, that means the number of iterations and the needed time do not differ by much. Especially the number of iterations lie closely together for all methods, so that the focus is more on the timings. There the outliers are *full_coarsening*, $n_c = 3$ and *semi_coarsening 1* for $d = 8$. For these two methods the sizes of the coarsest systems are the largest ones compared to the other methods. This size difference is the reason for the time increase as the computation and application of the pseudo inverse for solving the least squares problem on the coarsest grid is much more expensive. If d is even larger than in the tests presented here,

Coarsening strategies:

full_coarsening, $n_c = 2$	full_coarsening, $n_c = 3$
$n \times n \times \cdots \times n$ $\downarrow$ $(\lceil \frac{n}{2} \rceil) \times (\lceil \frac{n}{2} \rceil) \times \cdots \times (\lceil \frac{n}{2} \rceil)$ $\downarrow$ $\vdots$ $\downarrow$ $2 \times 2 \times \cdots \times 2$ $\Rightarrow$ no. of levels $= \log_2(n-1) + 1$	$n \times n \times \cdots \times n$ $\downarrow$ $(\lceil \frac{n}{2} \rceil) \times (\lceil \frac{n}{2} \rceil) \times \cdots \times (\lceil \frac{n}{2} \rceil)$ $\downarrow$ $\vdots$ $\downarrow$ $3 \times 3 \times \cdots \times 3$ $\Rightarrow$ no. of levels $= \log_2(n-1)$
semi_coarsening 1 (WLOG d is odd)	semi_coarsening 2 (WLOG d is odd)
$n \times n \times \cdots \times n \times n$ $\downarrow$ $(\lceil \frac{n}{2} \rceil) \times n \times (\lceil \frac{n}{2} \rceil) \times \cdots \times n \times (\lceil \frac{n}{2} \rceil)$ $\downarrow$ $\vdots$ $\downarrow$ $2 \times n \times 2 \times \cdots \times n \times 2$ $\downarrow$ $2 \times (\lceil \frac{n}{2} \rceil) \times 2 \times \cdots \times (\lceil \frac{n}{2} \rceil) \times 2$ $\downarrow$ $\vdots$ $\downarrow$ $2 \times 5 \times 2 \times \cdots \times 5 \times 2$ $\Rightarrow$ no. of levels $=$ $(\log_2(n-1) - 1)$ $+(\log_2(n-1) + 1)$	$n \times n \times \cdots \times n \times n$ $\downarrow$ $(\lceil \frac{n}{2} \rceil) \times (\lceil \frac{n}{2} \rceil) \times \cdots \times (\lceil \frac{n}{2} \rceil)$ $\downarrow$ $\vdots$ $\downarrow$ $3 \times 3 \times \cdots \times 3 \times 3 \times 3$ $\downarrow$ $2 \times 3 \times \cdots \times 2 \times 3 \times 2$ $\Rightarrow$ no. of levels $= \log_2(n-1) + 1$
semi_coarsening 3	semi_coarsening 4
$n \times n \times \cdots \times n \times n$ $\downarrow$ $n \times (\lceil \frac{n}{2} \rceil) \times \cdots \times (\lceil \frac{n}{2} \rceil) \times n$ $\downarrow$ $\vdots$ $\downarrow$ $n \times 1 \times \cdots \times 1 \times 1 \times n$ $\Rightarrow$ no. of levels $= \log_2(n-1) + 2$	$n \times n \times \cdots \times n \times n$ $\downarrow$ $(\lceil \frac{n}{2} \rceil) \times (\lceil \frac{n}{2} \rceil) \times \cdots \times (\lceil \frac{n}{2} \rceil)$ $\downarrow$ $\vdots$ $\downarrow$ $3 \times 3 \times \cdots \times 3 \times 3$ $\downarrow$ $3 \times 2 \times \cdots \times 2 \times 3$ $\downarrow$ $3 \times 1 \times \cdots \times 1 \times 3$ $\Rightarrow$ no. of levels $= \log_2(n-1) + 2$

Table 7.1: Different coarsening strategies.

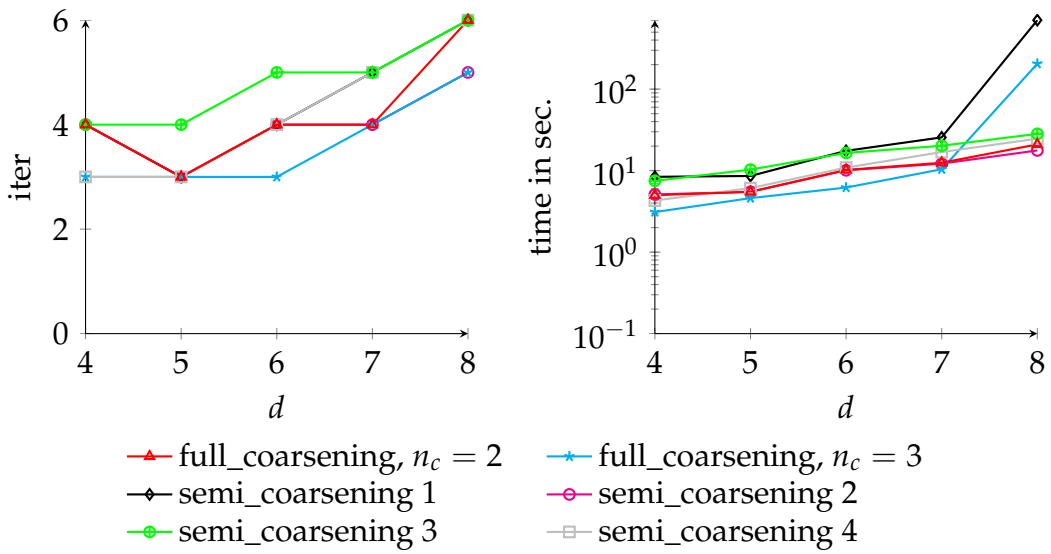


Figure 7.7: Number of iterations and running time for model *overflow* with mode size $n = 17$ and varying dimensions d for different coarsening strategies.

we would also expect a time increase for the methods *full_coarsening*, $n_c = 2$ and *semi_coarsening 2*. At first sight, it thus seems that the methods which get rid of the curse of dimensionality, namely the methods *semi_coarsening 3* and *semi_coarsening 4* work the most efficiently in general (as the performance of these is only slightly worse than that of the others here and they are not influenced as much by a further increase of d).

However, if we test the methods with a small dimension and ranging mode sizes, we get a different behaviour, as shown in Figure 7.8. Here we have a noticeable difference in the number of iterations and a correlation between the needed time and the number of iterations (so methods which need more iterations also need more computation time). The best behaviour is exhibited by the method *full_coarsening*, $n_c = 3$ whereas, as was discussed previously in connection with Figure 7.7, this method has problems when d is large. Here, the size of the coarsest system is not problematic. So these tests show that if d is under control, i.e., computing or applying the pseudo inverse on the coarsest grid can be done in an acceptable time, methods which want to get rid of the curse of dimensionality are not necessary.

Altogether, the method *semi_coarsening 4* seems to be a good method of choice. Note that the methods with semi-coarsening strategies do not give a gain in the iteration number, but only deliver a better time performance. Therefore it is difficult to give a general suggestion for the best method. The performance of course depends on the implementation and the machine and thus, depending

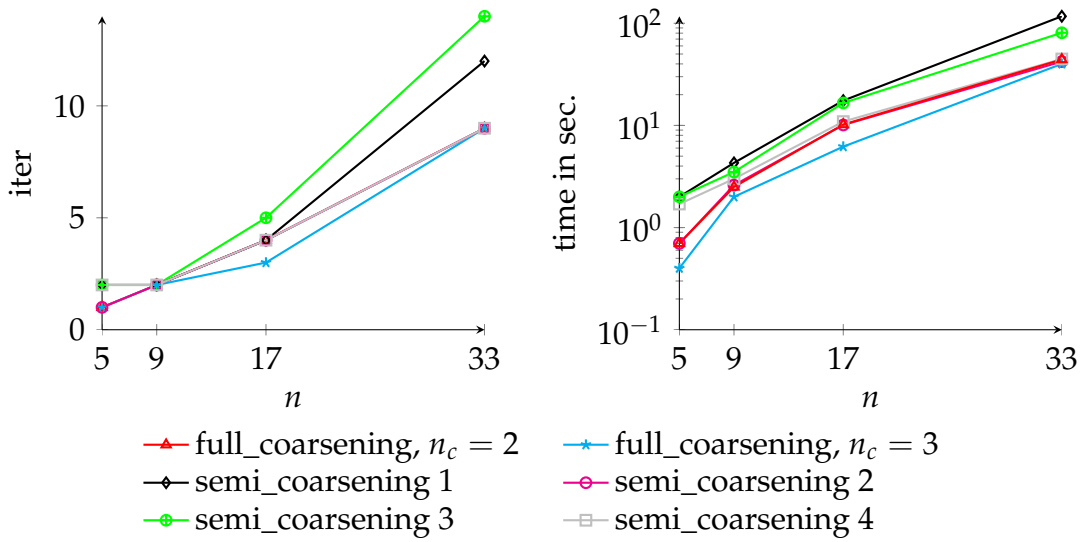


Figure 7.8: Number of iterations and running time for model *overflow* with dimension $d = 6$ and varying mode sizes n for different coarsening strategies.

on the setting, one method may be favored over the other (depending, e.g., on how efficient dense matrix operations can be performed versus sparse matrix or tensor operations).

So in general if d is small, we will prefer the method *full_coarsening*, $n_c = 3$. From the tests presented here we would say that d can be considered small in this sense when $d \leq 7$. For the other case we differ between an interval for d where the method *full_coarsening*, $n_c = 2$ gives a nice performance and outside the interval the method *semi_coarsening 4* will be the method of choice. In chapter 8 there will be more tests where we also consider models with larger values of d.

7.4 Numerical tests for *kanban_control*

In the experiments from section 7.2 the approximate Gauss-Seidel smoother did not perform well. The next series of numerical tests will show different results. For this we consider the model *kanban_control*. Recall Figure 2.3 and the fact that the states of each subsystem in the *kanban_control* model cannot be arranged into a one-dimensional graph, see section 2.2.3. Therefore the aggregation approach described in section 6.2.3 will be used. In the experiments in section 7.2 we observed that using Gauss-Seidel as smoother is too expensive, where the main cost comes from computing the action of $(D + L)^{-1}$ via AMEn. To alleviate the time increase due to Gauss-Seidel, we will use approximated Gauss-Seidel

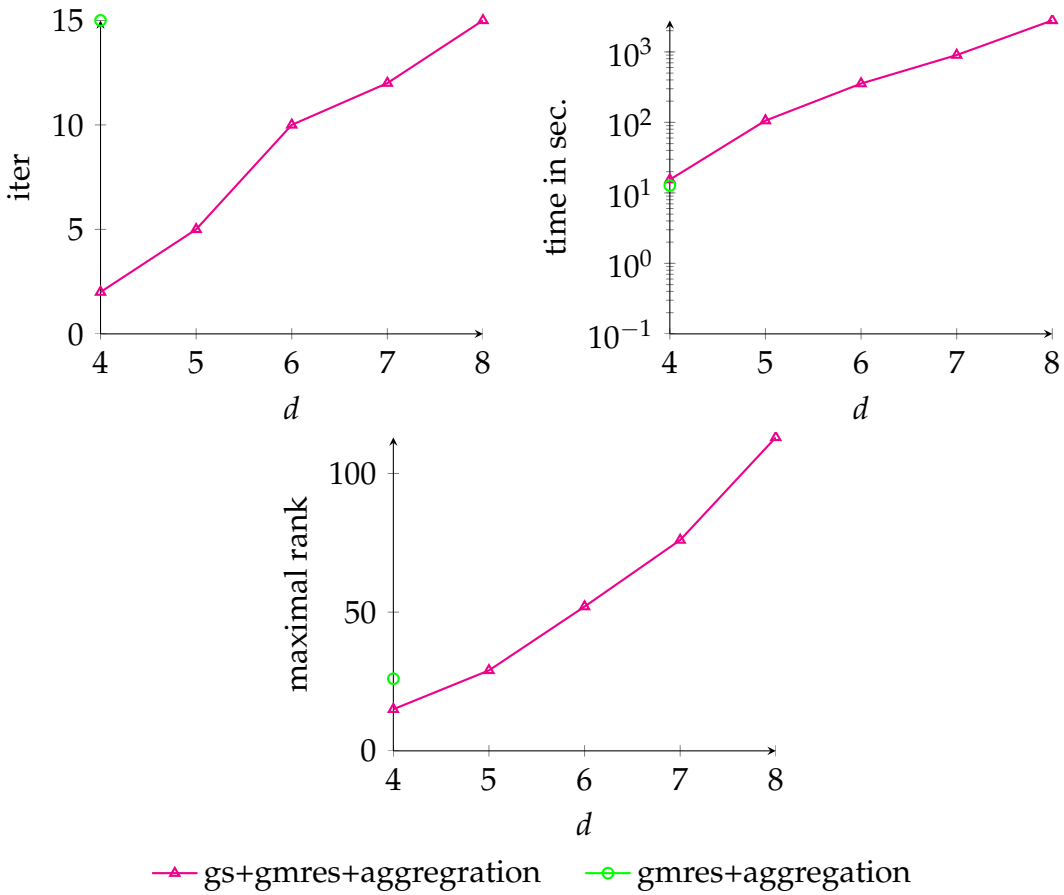


Figure 7.9: Number of iterations, running time and maximal rank for model *kanban_control* with number of waiting seats $k = 5$ and varying dimension d.

as smoother only on the first level of our multigrid hierarchy in our next series of experiments, and GMRES on the other levels. We denote this method as *gs+gmres+aggregation* in the following experiments. We compare this hybrid method with a method where we use GMRES as smoother on each level of our multigrid hierarchy. This method is labelled *gmres+aggregation*. Note again that we have to use the aggregation based interpolation, so the tests will focus more on the efficiency of the smoothers. Like in sections 7.2 and 7.3 we have two kinds of test series. In the first one we fix the number of waiting seats and vary the dimension, choosing $k = 5$ and $d = 5, 6, 7, 8$. In the second one, we vary the number of waiting seats (note that this implies that we vary the mode sizes) and fix the dimension. Here we choose the dimension $d = 6$ and, $k = 3, 5, 9, 17$ waiting seats in each queue.

Recall that the size of the generator matrix of *kanban_control* is given via the

d	5	6	7	8
N	333,396	7,001,316	147,027,636	3,087,580,356

Table 7.2: Size of the generator matrix of *kanban_control* for varying dimensions d and fixed number $k = 5$ of waiting seats.

k	3	5	9	17
N	160,000	7,001,316	915,062,500	277,031,690,244

Table 7.3: Size of the generator matrix of *kanban_control* for varying number k of waiting seats and fixed dimension $d = 6$.

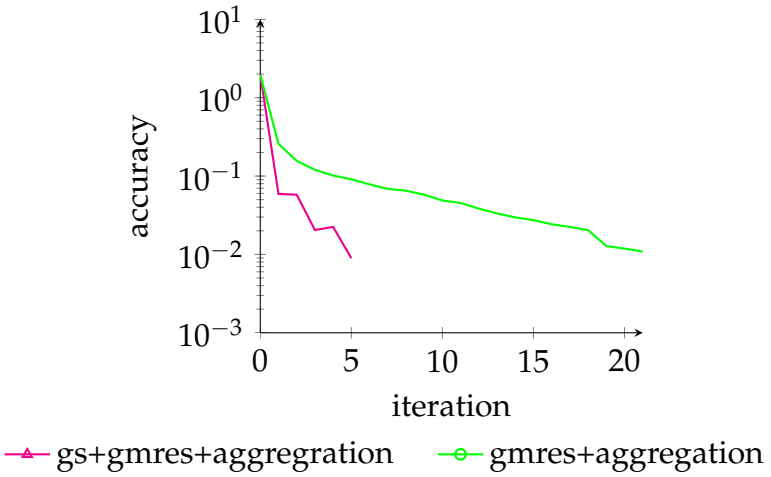


Figure 7.10: Convergence behaviour of *gs+gmres+aggregation* and *gmres+aggregation* for model *kanban_control* with $d = 5$ and $k = 5$.

formula

$$(k+1)^2 \left(\frac{(k+1)(k+2)}{2} \right)^{d-2}.$$

For illustration, in Tables 7.2 and 7.3 the corresponding problem sizes for the test cases we consider in this section are shown.

In Figures 7.9 and 7.11 we investigate these experiments by showing the number of V-cycles, the needed computation time in seconds and the maximal needed rank. Note that all other implementation details, like for example the stopping criteria, are chosen as described in section 7.1. Eye-catching are the results where we vary the dimension d, see Figure 7.2. Here the method *gmres+aggregation* fails for all $d > 4$. For $d = 4$ the method *gmres+aggregation* needs 15 iterations, in con-

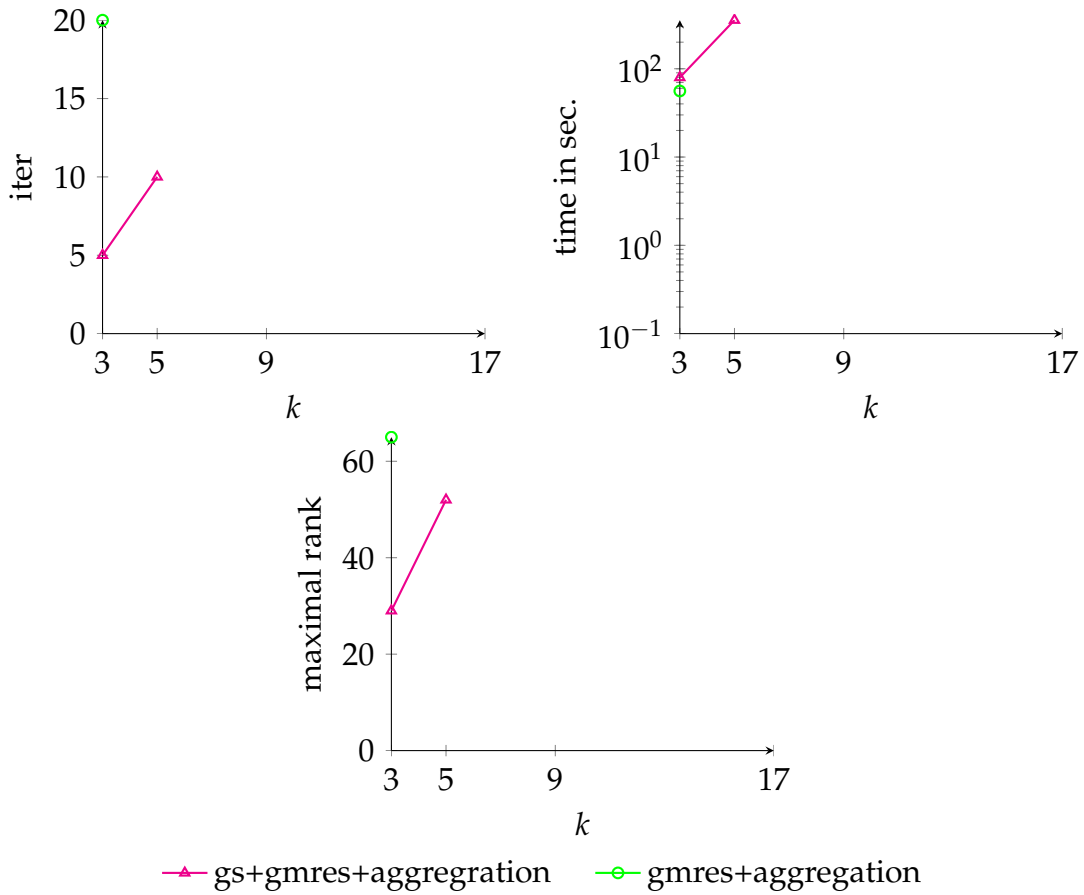


Figure 7.11: Number of iterations, running time and maximal rank for model *kanban_control* with dimension $d = 6$ and varying number k of waiting seats.

trast to two iterations for the method *gs+gmres+aggregation*. Still they both need about the same computation time, which shows that the Gauss-Seidel smoother is much more expensive, but also has much better smoothing properties.

The reason for the failure for $d \geq 5$ is that the allowed maximal rank is exceeded. Figure 7.10 shows the convergence plot of both methods for $d = 5$ and $k = 5$. The low convergence rate of *gmres+aggregation* leads to an unnecessary rank increase, recall that we increase the rank if two consecutive residual norms differ by less than 10 percent, which turned out to be a good choice for the other models. In chapter 9 we will also see that with this value the methods typically find almost exactly the rank that is necessary for representing the solution for the desired accuracy. Note that even when allowing both more iterations and a larger maximum rank, the method using GMRES will still fail for larger problems as the many unnecessary rank increases will lead to exploding computation time. This leads to changing parameters even further to make the method

work for this model. Note that this was not and will not be necessary for any of the other models, which stresses that the method really struggles with this model. The method *gs+gmres+aggregation* however manages to solve all considered problem instances within the given restrictions, which also shows that the aggregation based interpolation works well.

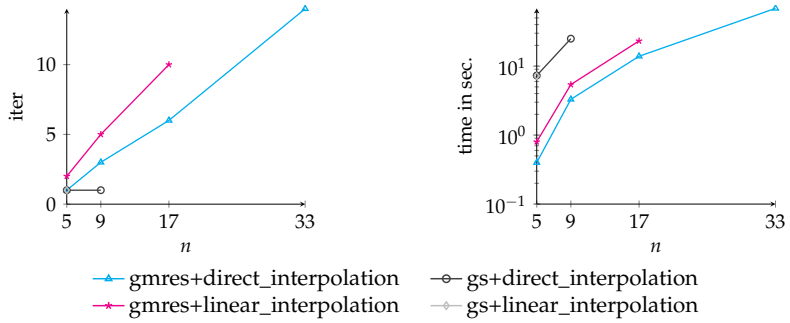
In chapter 9 we will argue that AMEn does not work well for larger mode sizes. So the results for varying mode sizes in Figure 7.11 do not surprise us, namely that the method *gs+gmres+aggregation* fails for $k \geq 9$, which means a mode size $n \geq 55$ for the inner subsystems. If AMEn fails, then it is clear that the approximated Gauss-Seidel cannot work as well and subsequently the overall multigrid method cannot be expected to perform well. Nevertheless the method *gmres+aggregation* fails for even smaller problems. Concerning the comparison of running time and needed rank between both methods, we again observe the same behaviour as when varying d.

At first glance the results of this section do not look as promising as the results of the previous sections 7.2 and 7.3. But they should be seen in the context that the *kanban_control* model is known to be extremely difficult to solve. In [47] only capacities $k = 1$ were considered and already this problem was reckoned to be more difficult to solve than, e.g., overflow queueing models.

Summary of Chapter 7:

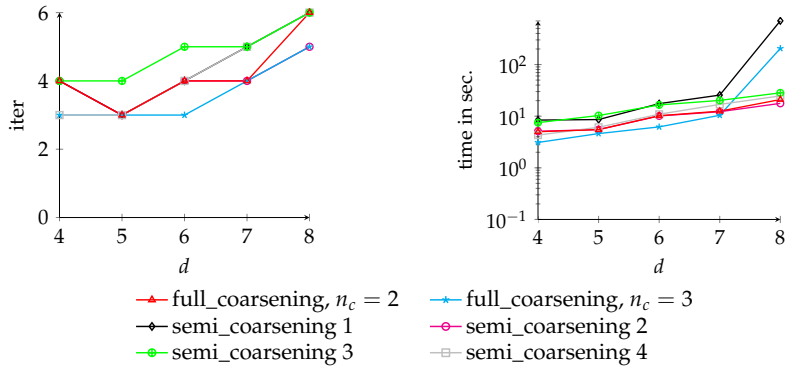
Numerical tests for different smoothers and interpolation operators:

- Direct interpolation is better than linear interpolation.
- GMRES is a better smoother than GS in terms of computation time.



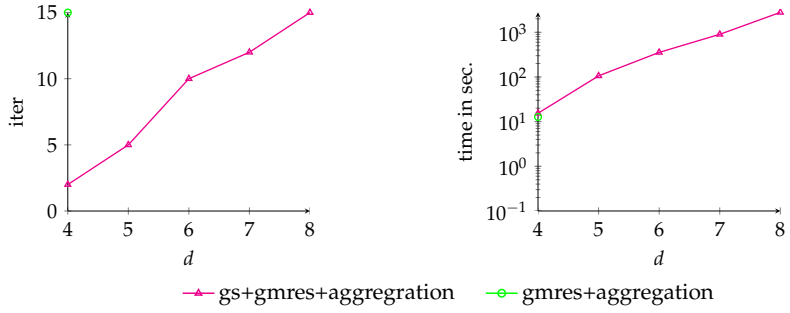
Numerical tests for different coarsening strategies:

- Suggestion: for $d \leq 7$, *full_coarsening*, $n_c = 3$
  for $d \geq 8$, *full_coarsening*, $n_c = 2$ or *semi_coarsening 4*



Numerical tests for kanban_control

- Aggregation based multigrid approach is reasonable.
- Hybrid GS+GMRES smoother works better than only GMRES.



CHAPTER 8

Bootstrap AMG

In the previous chapters, the focus was on the computation of stationary distributions of continuous time Markov chains (CTMCs) (see chapter 2). There, we have to deal with the curse of dimensionality, which explains the need for using the low-rank format Tensor Train (see chapter 3). Based on this format, a tensorized algebraic multigrid method was built. For constructing the ingredients of the multigrid method, the geometric structure of the models played an important role.

An alternative is to find a way to get rid of using the geometric structure. This means that we want to find out whether there is a way to build these ingredients adaptively, i.e., by the method itself, without additional information provided from the outside. Here, we will only focus on the entries of the interpolation operator. In [6, 9] a bootstrap algebraic multilevel approach for discrete time Markov chains (without tensor structure) was considered and shown to work efficiently as a preconditioner. One nice advantage of this method, amongst others, is that the entries of the interpolation operator were built adaptively via a least squares approach. We will discuss this technique in the following and develop a specialized bootstrap approach for the tensorized case.

8.1 Least squares based interpolation

Recall that the statement of the *strong approximation property* (see (5.11)) given via the condition

$$\|e - Pe_c\|_{QA}^2 \leq \frac{K_s}{\|QA\|_2} \langle QAe, QAe \rangle.$$

for each e on the fine grid and for a small constant $K_s \geq 0$, includes the demand that the vector e needs to be approximated in the range of P with an accuracy that is inversely proportional to $\|QAe\|_2$. Besides, we discussed that the range of P should lie in a space $\mathcal{V}$ spanned by eigenvectors to small eigenvalues of A (recall that these were the parts which cause slower convergence of the smoother).

To satisfy these two demands we use the common idea for adaptively constructing suitable transfer operators which is described in [6,9,11,43]. This idea is the following: Before starting the actual solution process, i.e., iteratively applying V-cycles until the approximation is good enough, a first so-called *setup phase* is performed in which the interpolation operator P is constructed and refined.

After applying a few smoothing steps on a set of random vectors, the parts of the error belonging to eigenvectors corresponding to smaller eigenvalues dominate (see chapter 4). These smoothed random vectors will then be our first test vectors. Then we try to compute P so that these test vectors lie (approximately) in $range(P)$ via a least squares approach, i.e., given a set of normalized test vectors $\mathcal{V} = \{v^{(k)} : \|v^{(k)}\|_2 = 1; k = 1, \dots, r\}$ we solve a weighted least squares problem of the form

$$\min \mathcal{L}(p_{i,\bullet}) = \sum_{k=1}^{r} \omega_k \Big(v^{(k)} - \sum_{j \in \mathcal{J}_i} (p_{i,\bullet})_j (\widetilde{R} v^{(k)})_j \Big)^2 \tag{8.1}$$

to determine the non-zero entries $p_{i,j}, j \in \mathcal{J}_i$ of each row $p_{i,\bullet}$ of P, where $\mathcal{J}_i \subset \mathcal{C}$ is the index set of the coarse variables from which the ith variable interpolates. $\widetilde{R}$ is a matrix of size $n_c \times n$ which describes the canonical injection of a vector onto its coarse grid components, i.e.

$$\widetilde{r}_{k,\ell} = \begin{cases} 1 & k = \pi^{-1}(\ell) \\ 0 & \text{else} \end{cases}$$

with π^{-1} from (6.6). This optimization procedure only determines the entries of P and we assume that the sets $\mathcal{J}_i$ are given beforehand. In [6,11] there is also an optimization strategy for finding suitable coarse interpolation variables for solving (8.1). Because of the tensor structure of our models, a general, purely algebraic approach to compute $\mathcal{J}_i$ (a set of coarse variables in the neighbourhood of i) is not sensible, either it would mean to lose the tensor structure or we arrive at the same result as for full coarsening. Because we determine $\mathcal{J}_i$ via the geometry of our models, an optimization procedure over this set is not necessary afterwards. The weights ω_k are chosen like it was recommended in [6,9,11,43] as

$$\omega_k = \frac{1}{\|Ae\|_2^2}.$$

The idea comes from the spd case. There it is shown that the error components which cause slower convergence satisfy the inequality $\|Ae\|_2 \ll \|e\|_2$, so these vectors should have the largest weights in the minimization process. In our case these weights are heuristic, but sensible because our error components have a similar behaviour, see chapter 4. Another way to motivate these weights is that, because the matrix Q is unitary, they can equivalently be written as

$$\omega_k = \frac{1}{\|QAe\|_2^2},$$

which is a natural choice in light of the strong approximation property.

The approach described above delivers an interpolation operator which fulfils $v^{(k)} \approx P(\widetilde{R}v^{(k)})$ for the test vectors. Whether this means that P leads to a fast converging multigrid method of course depends on the choice of those test vectors. In particular, recalling again the strong approximation property, it is desirable that small eigenvectors of A lie close to the range of P. This should be reflected in the choice of the test vectors, i.e., a test vector should mostly consist of small eigenvectors of A. On the finest level we are not able to compute an eigenbasis of A, not even a single eigenvector (note that the smallest eigenvector of A is the stationary distribution vector we are looking for). Therefore we can only compute a certain number of small eigenvectors on the coarsest level which can then be interpolated to the finest grid and hopefully be good approximations to the small eigenvectors of A.

To be able to use this approach we of course need to have a coarse grid operator available already. Therefore, a first multigrid hierarchy is built using random, smoothed test vectors. Recalling Figures 4.2–4.4, after just a few smoothing steps, the remaining error will consist mostly of small eigenvectors, so that these test vectors give a reasonable starting point for forming a good interpolation operator. After arriving at the coarsest grid we are able to compute a certain number of eigenvectors to small eigenvalues of the coarse grid operator cheaply (details on what one has to take into account for this are given in the next paragraph). These eigenvectors can be interpolated and used as test vectors alongside the random, smoothed test vectors in the next setup cycle for computing a new interpolation operator. In this manner we adaptively obtain a better interpolation operator, i.e., an interpolation operator which interpolates the eigenvectors to small eigenvalues more accurately. Thus, iteratively applying this approach should ideally lead to a constantly improving multigrid hierarchy and thus a faster converging method. In addition, note again that the eigenvector to the smallest eigenvalue $\lambda = 0$ is the solution we are seeking, so as a further advantage, we should also get a better approximation of our solution this way, which can then be used as a starting guess for the multigrid method.

A crucial point which still needs to be clarified is how to extract the "correct"

Algorithm 8.1: BAMG setup, first cycle

Input: A_ℓ $(A_1 = A)$, T_ℓ $(T_1 = I)$, test vectors $\mathcal{V}_\ell$
$(\mathcal{V}_1 = \{v_1^{(k)}, k = 1, \ldots, r_{pre})\}$
Output: The set $(\Lambda_\ell, \mathcal{V}_\ell)$ of approximate eigenvectors and values of A

if coarsest grid is reached **then**
  Determine the r smallest generalized eigenvectors and eigenvalues of $A_\ell v_\ell^{(k)} = \lambda_\ell T_\ell v_\ell^{(k)}, k = 1, \ldots, r$
else
  for $k = 1, \ldots, r_{pre}$
    Apply smoothing to $A_\ell v_\ell^{(k)} = \mathbf{0}$ with initial guess $v_\ell^{(k)}$
  end
  Construct P_ℓ via least squares interpolation (8.1)
  Construct R_ℓ via linear interpolation
  Compute $A_{\ell+1} \leftarrow R_\ell A_\ell P_\ell$
  Compute $T_{\ell+1} \leftarrow R_\ell T_\ell P_\ell$
  Set $\mathcal{V}_{\ell+1} \leftarrow \{\widetilde{R}_\ell v_\ell^{(k)} : k = 1, \ldots, r_{pre}\}$
  $(\Lambda_{\ell+1}, \mathcal{V}_{\ell+1})$ = bamg_mle$(A_{\ell+1}, T_{\ell+1}, \mathcal{V}_{\ell+1})$
  Set $(\Lambda_{\ell+1}, \mathcal{V}_{\ell+1}) \leftarrow \{(\lambda_{\ell+1}^{(k)}, P_\ell v_{\ell+1}^{(k)}) : k = 1, \ldots, r\}$
  for $k = 1, \ldots, r$
    Apply smoothing to $A_\ell v_\ell^{(k)} = \mathbf{0}$ with initial guess $v_\ell^{(k)}$
  end
end

eigenvector information from the coarse grid matrix $A_c = RAP$. Our goal is to find vectors v_c such that Pv_c is a good approximation to a small eigenvector of A, i.e., such that

$$APv_c - \lambda Pv_c \approx 0, \quad \text{for a small } \lambda.$$

Similar to the construction presented in chapter 4, we can do so by using a Galerkin ansatz, demanding that the eigenvector residual is orthogonal to the range of R^T, i.e.,

$$APv_c - \lambda Pv_c \perp range(R^T) \Leftrightarrow A_c v_c - \lambda RPv_c = 0.$$

So, this approach leads to a generalized eigenvalue problem on the coarsest level. Note that the value of λ is not affected by this construction, so we have to compute the generalized eigenvectors to the smallest generalized eigenvalues of A_c.

So as mentioned before, these generalized eigenvectors computed on the coarsest grid will be interpolated up to the finer grids and some smoothing steps will

be applied to them, so that they can be used as test vectors alongside the random, smoothed test vectors in the next setup cycle. Algorithm 8.1 summarizes this whole process for the first setup cycle. The subsequent setup cycles differ from the first one only in that the approximate eigenvectors from the first setup cycle are used as test vectors from the beginning on.

8.2 Tensorized bootstrap AMG

We would like to adapt algorithm 8.1 to the TT-format now. The main difficulty is the part for constructing the interpolation operator via least squares (see line 7 of Algorithm 8.1). Recall that we are not able to compute the interpolation operator row-wise because of the nature of the model. Besides we also do not want to compute it this way, because the interpolation operator should fulfil

$$P = \bigotimes_{i=1}^{d} P_i,$$

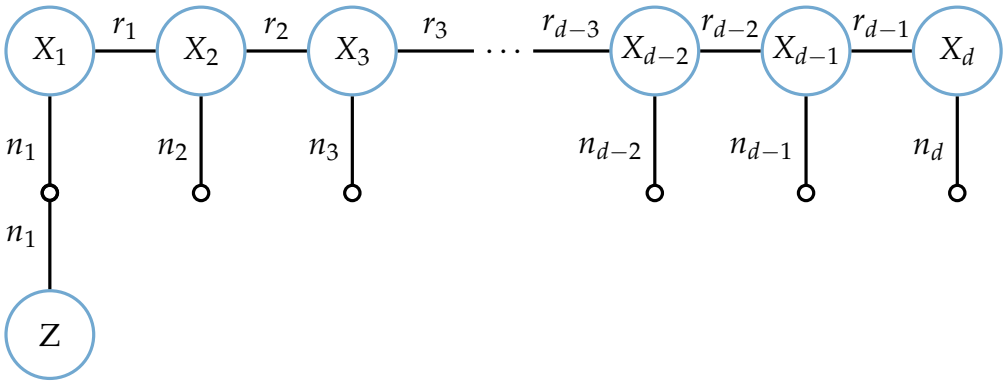
in order to maintain the tensor structure on the next grid (see section 6.3). Therefore the adaptive construction of P is reduced to the adaptive construction of the small matrices P_i. Recall that we have two kinds of test vectors,

- the random ones, to which a few smoothing steps have been applied,
- the interpolated approximate eigenvectors from the coarsest grid.

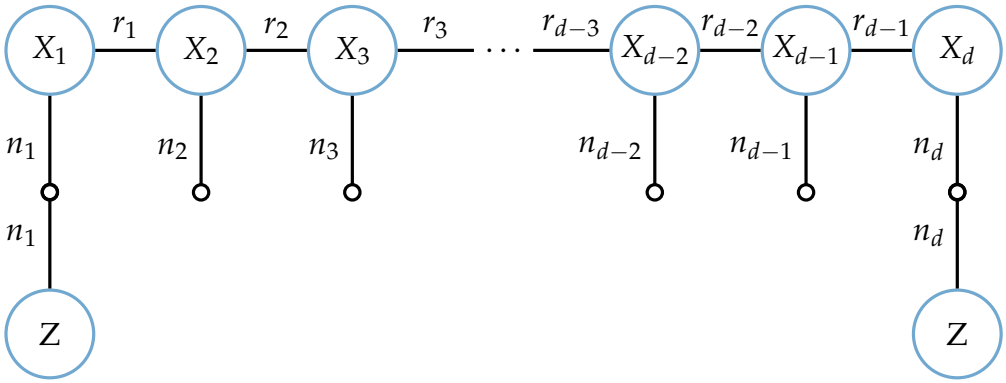
This characterization of the test vectors should be kept in the tensorized case. However, this is not compatible with the fact that we have to construct the small matrices P_i, because for generating P_i via the least squares problem (8.1) we need test vectors of size n_i, but we only have test vectors of size $n_1 \times n_2 \times \cdots \times n_d$. We thus need a method to generate vectors of size n_i from a TT-Tensor of size $n_1 \times n_2 \times \cdots \times n_d$. In the case that we have a TT-Tensor where all TT-ranks are $r_i = 1, i = 1, \ldots, d$, it seems sensible to take the core X_i, which in this case is also a vector, as test vector for constructing P_i. Unfortunately, our test vectors will never be of rank 1 because of the smoothing steps which are applied to them and increase their rank, see chapter 3. A generalization of the rank-1 approach is to construct the test vector as a linear combination of the columns of the ith core X_i. In order to also incorporate information from the other cores, we generate the coefficients of this linear combination from them as follows:

We successively reduce the other cores $X_k, k \neq i$ to scalars by left or right multiplication with a vector of all ones. Figure 8.1 illustrates this approach. Note that

Contract n_1 (and therefore also r_1):



Contract n_d (and therefore also r_d):



Continue until a vector of size n_2 is left:

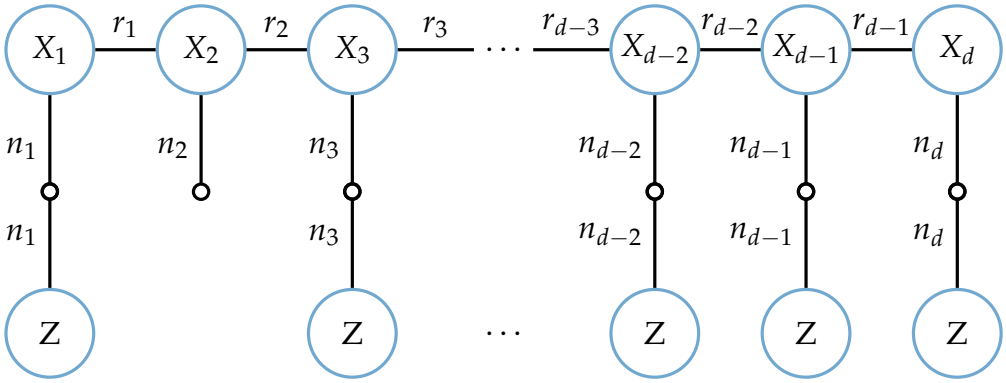


Figure 8.1: Contraction of a $n_1 \times n_2 \times \cdots \times n_d$ TT-Tensor to a vector of size n_2

in the rank-1 case this contraction delivers a multiple of the vector X_i, so it can be seen as a generalization of what we sketched for the rank-1 case. Still, the approach is clearly heuristic and it has to be verified in numerical experiments that the test vectors constructed this way allow to find a good interpolation operator.

All other operations in Algorithm 8.1 can be performed in the tensor format in the same way as described in the tensorized multigrid method developed in chapter 6.

8.3 Numerical tests

8.3.1 Implementation details

In the following we want to test the tensorized bootstrap approach described in section 8.2. In our experiments we name this method *Boot*. For comparison, we also test the tensorized multigrid method with direct interpolation (see chapter 6). This method is now labelled *Multi* instead of *gmres + direct_interpolation* (like in chapter 7.2). Until now, the largest dimension we tested was $d = 8$, while fixing the subsystem sizes to $n = 17$, and the largest size of the subsystems was $n = 33$, while fixing the dimension to $d = 6$ (see section 7.3). In this chapter, we vary the dimension up to $d = 11$, while fixing $n = 17$, and the subsystem size up to $n = 129$, while keeping $d = 6$. Because of the curse of dimensionality on the coarsest grid for both methods, we use the following coarsening strategy for the tests, based on the experiments from section 7.3: If $d \leq 7$, we coarse until each subsystem is of size 3, otherwise we coarse until each subsystem is of size 2. This way it is guaranteed that the least squares problem on the coarsest grid can be solved in a reasonable amount of time for all test cases. Recall that the method *semi_coarsening 4* from section 7.3 also appeared to be a good method of choice when dealing with high dimension. We now label this method *Multisemi* in this chapter. Obviously this strategy can also be adapted to the bootstrap approach in a straightforward way. This adapted method is named *Bootsemi*. So, we have four methods to compare. In all methods GMRES is used as smoother. To focus on the quality of the interpolation operator (recall that this is the main reason for using bootstrap) we use five pre- and postsmoothing steps instead of ten as it was the case in the previous experiments (see chapter 7). The starting rank in the solve process is now increased to 20 instead of 15, because we test more problems of larger size and can expect a rank increase for these problems. Apart from that, the parameters are the same as those given in section 7.1.

In the methods *Boot* and *Bootsemi*, we perform four setup cycles before starting

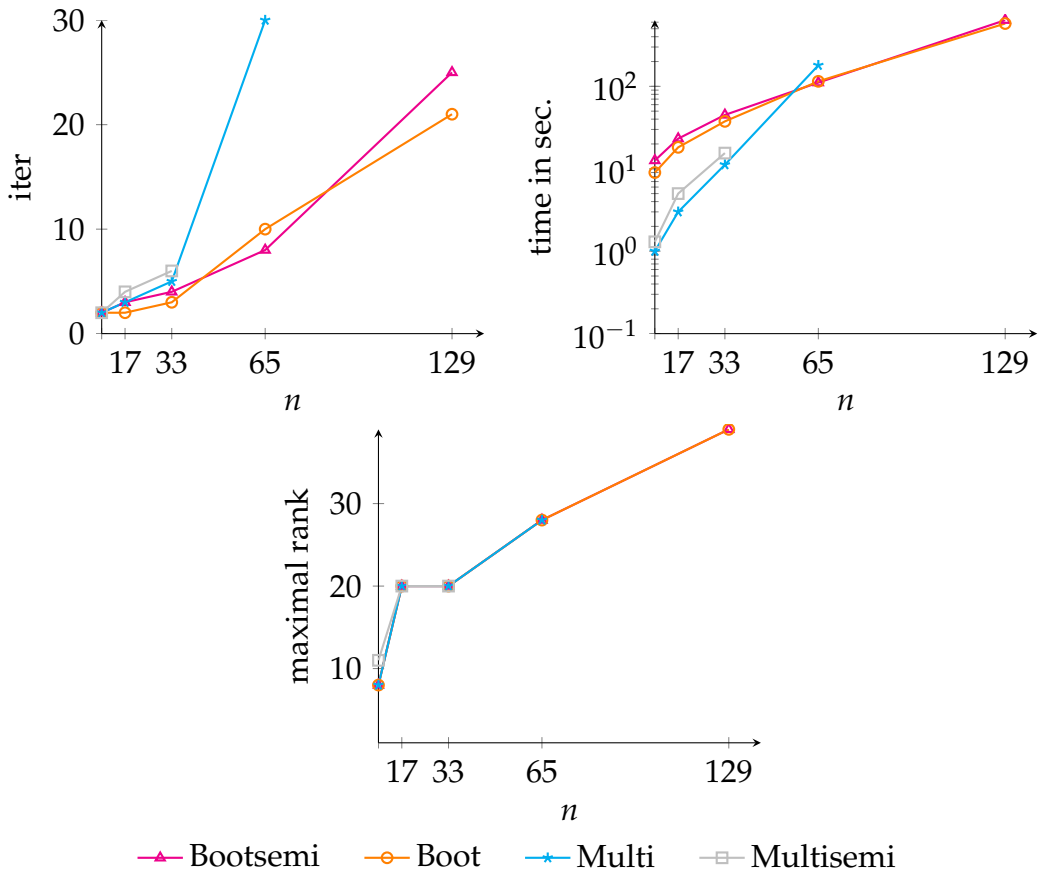


Figure 8.2: Number of iterations, running time and maximal rank for model *overflow* with dimension $d = 6$ and varying mode sizes n.

the solve part. In the setup procedure, we apply seven pre- and post smoothing steps and also use a truncation rank of 20. The choice of seven smoothing steps is based on the observation (in preliminary numerical tests), that this additional work invested in the setup leads to an overall more efficient method, because the quality of the test vectors is increased. Besides, it is possible to perform additional setup cycles after each V-cycle, in case that divergence of the method is observed. If the residual norm after two consecutive V-cycles increased by more than ten percent, we interpret this as divergence and perform an additional setup cycle in order to increase the quality of the multigrid hierarchy and hopefully overcome the divergence this way.

For solving the least squares problem (8.1) we use the Matlab built-in function `lsqlin`, which applies the trust-region-reflective algorithm from [22].

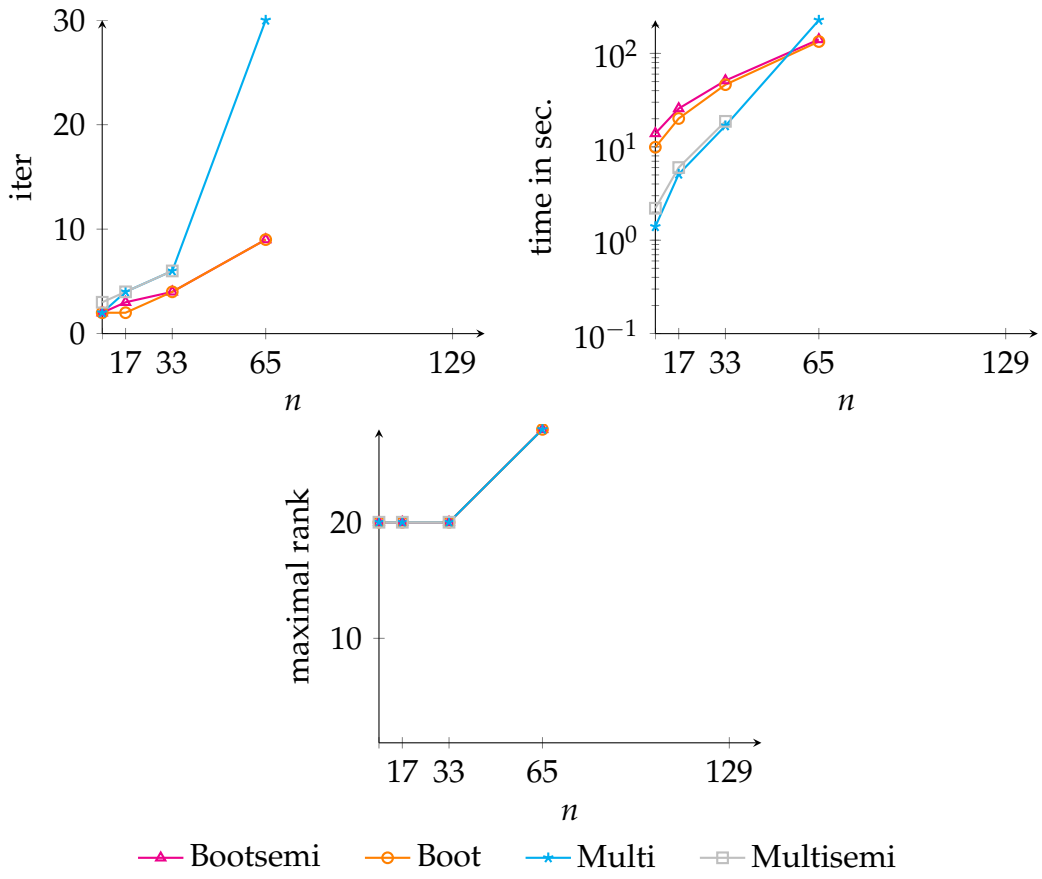


Figure 8.3: Number of iterations, running time and maximal rank for model *overflow_cycling* with dimension $d = 6$ and varying mode sizes n.

8.3.2 Numerical tests for tensorized multigrid and bootstrap with different coarsening strategies

First, we take a look at the results of the experiments where we vary the mode sizes n. We illustrate these experiments by presenting the needed number of V-cycles, computation time and the maximal rank. These results are shown in Figures 8.2–8.7. Note that the computation time of each method includes the computation time of its setup procedure as well, which obviously only has a noticeable effect on the computation time of the bootstrap methods *Boot* and *Bootsemi*. This effect is confirmed by the experiments and will be discussed first by considering and comparing *Boot* and *Multi*. We observe that the iteration number of *Multi* and *Boot* lie closely together for smaller mode sizes ($n \leq 17$ or $n \leq 33$, depending on the problem), and that for larger mode sizes *Boot* often needs much fewer iterations than *Multi*. To be more precise, for all models

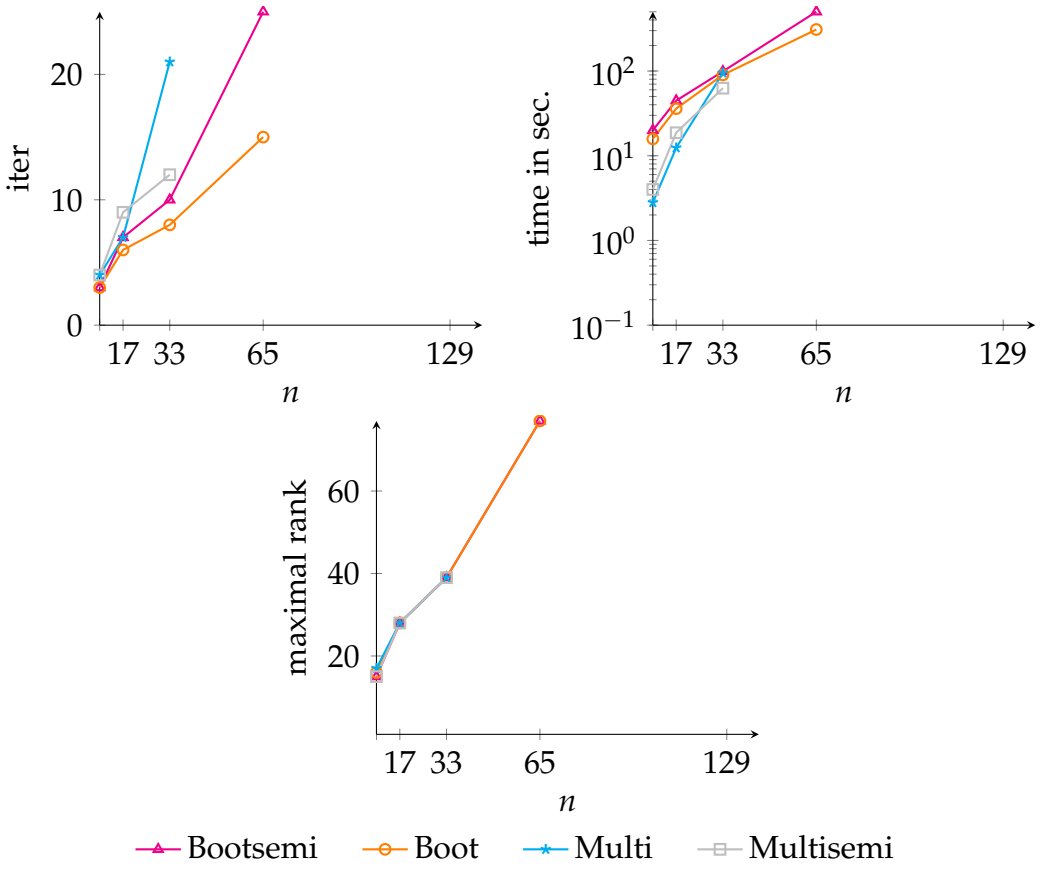


Figure 8.4: Number of iterations, running time and maximal rank for model *overflow_long* with dimension $d = 6$ and varying mode sizes n.

but *diverging_metab* the number of V-cycles in the *Boot* method is less than in *Multi*. Although the number of iterations gives evidence of a good performance of the *Boot* approach, the computation time of *Boot* is beaten by *Multi* for all experiments with mode sizes $n \leq 33$ (the other experiments with $n > 33$ are considered later). So the lower number of V-cycles is not enough to compensate for the computation time of the setup procedure.

Still, the number of V-cycles gives rise to optimism because it shows that the *Boot* approach works and delivers a nice convergence. And it should not be despised that a few problems are solved by the *Boot* method, but not by *Multi* under the requested stopping criteria. This is the case for the model *overflow* with mode size $n = 129$ and for the model *overflow_long* with mode size $n = 65$. Besides, we observe that with larger mode sizes the computation time of *Multi* and *Boot* lie closer together for all models but *diverging_metab*. For the models *overflow_long* and *simplified_kanban*, the computation times are nearly the same for the experiments with mode size $n = 33$, for *overflow* and *overflow_cycling*

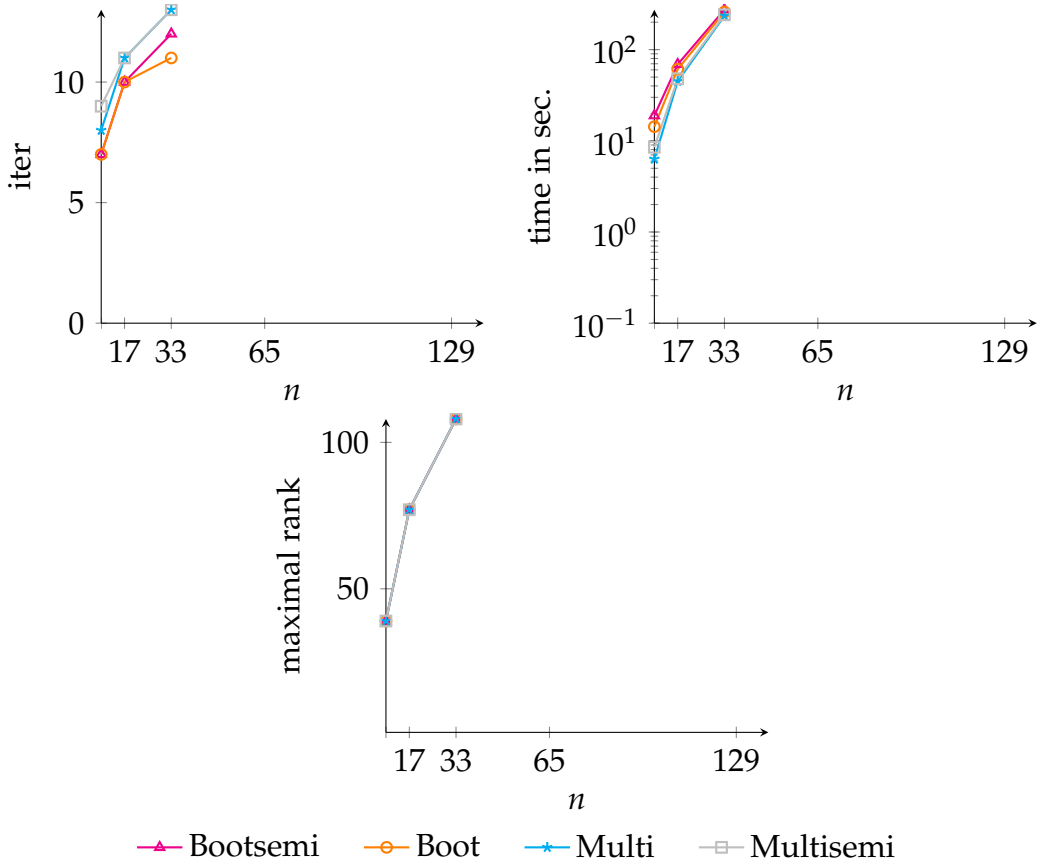


Figure 8.5: Number of iterations, running time and maximal rank for model *simplified_kanban* with dimension $d = 6$ and varying mode sizes n.

this happens for $n = 65$. The reason for this is that these are cases where the difference of the needed V-cycles in the *Multi* method is much larger than for *Boot* (for example the method *Boot* needs 8 iterations and *Multi* needs 21 iterations for the model *overflow_long* with $n = 33$). For the model *directed_metab* the time performance of *Boot* is better than that of *Multi* for the experiment with mode size $n = 65$. The reason for this is not a large difference in the number of the needed iterations, because both methods need 22 iterations, here the adaptive rank increase during the method is the cause. The increase occurs earlier in *Multi* than in *Boot*, which also shows that the convergence rate in *Boot* is better than in *Multi*.

Now we also consider the semi-coarsening approaches *Multisemi* and *Bootsemi*. The performance of *Multisemi* and *Bootsemi* is the same as that of *Multi* and *Boot*, respectively, in many cases. For the models of the category overflow, *Boot* and *Bootsemi* behave very similarly. For the models *directed_metab* and *diverging_metab* the method *Bootsemi* fails for all mode sizes $n \geq 17$.

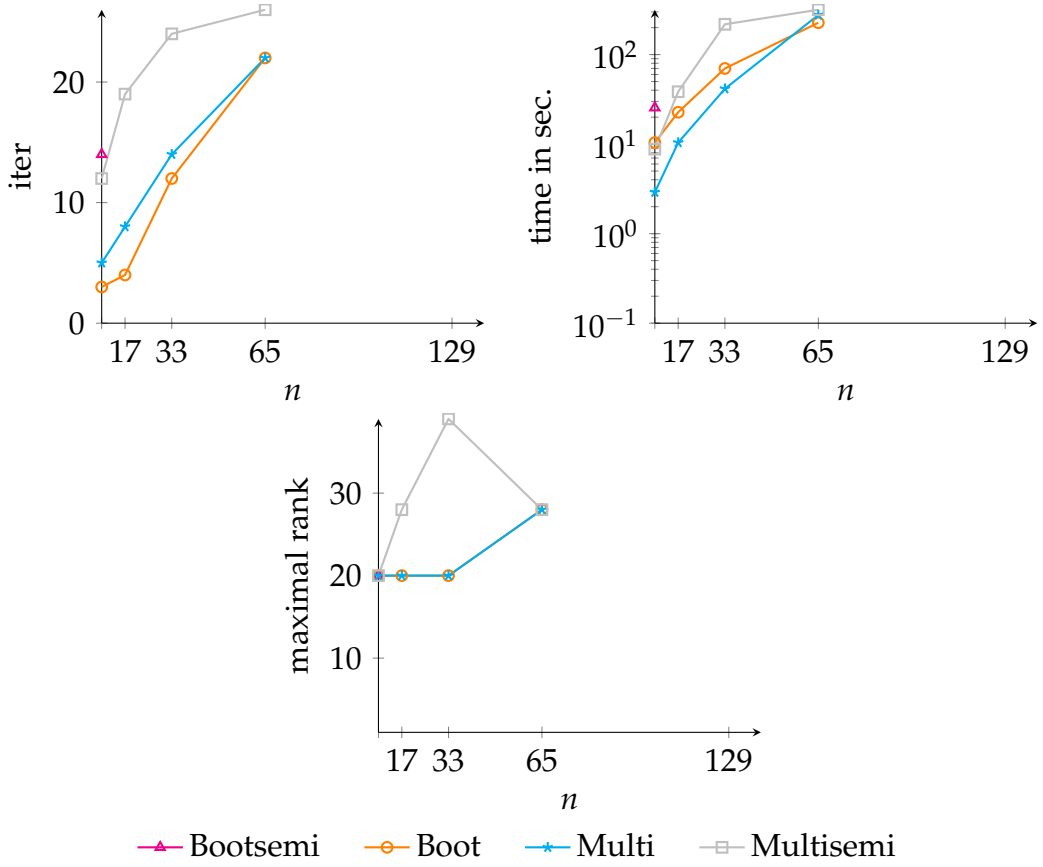


Figure 8.6: Number of iterations, running time and maximal rank for model *directed_metab* with dimension $d = 6$ and varying mode sizes n.

The comparison between *Multisemi* and *Multi* is very similar to that between *Bootsemi* and *Boot*. One can again observe that the semi-coarsening approach works quite well (but not better) for the overflow models and *simplified_kanban*, and shows clearly worse performance for the metabolite models (although it works better than *Bootsemi* for *directed_metab*). We stress that the semi-coarsening strategies (which in our setting were mainly motivated by their potential to cure the curse of dimensionality) can be expected to perform worse than the other approaches for $d = 6$, as the time needed for solving the coarsest grid system is very small anyway. Therefore, these approaches unnecessarily increase the number of levels, which typically worsens the convergence rate, without any real gain.

Concerning the maximal rank that is needed, all four methods behave almost identically whenever they work well and converge nicely to the solution. The only (few) major differences that can be observed are in cases where a method does not work well and unnecessary rank increases occur due to a low conver-

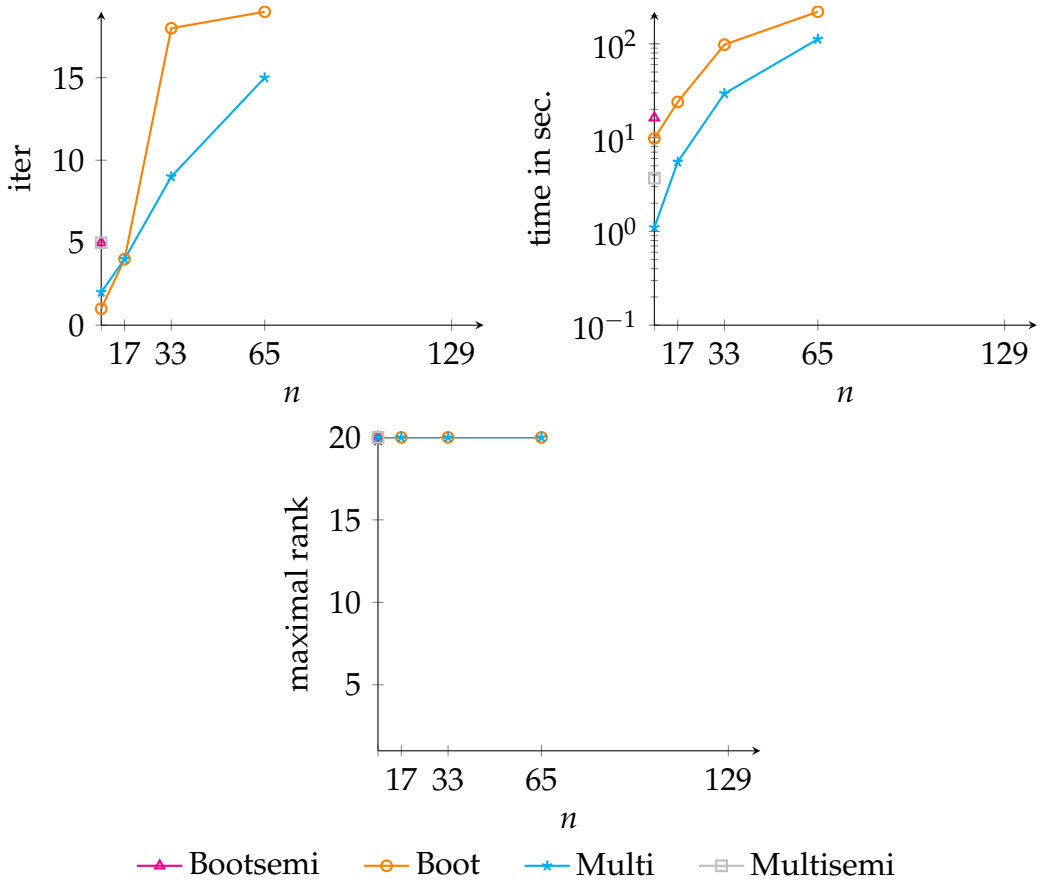


Figure 8.7: Number of iterations, running time and maximal rank for model *diverging_metab* with dimension $d = 6$ and varying mode sizes n.

gence rate.

Now we are looking how the four methods behave when varying the dimension d. This series of experiments is shown in Figures 8.8–8.13. The general trend we can observe from these experiments is again that *Boot* outperforms *Multi* in terms of iteration numbers for almost all problems but *diverging_metab*. However this is again not reflected in the time performance, because the setup procedure for the bootstrap approaches is more expensive than the iterations which are saved in the solve process.

One notable exception from this observation is the model *directed_metab* where *Boot* outperforms *Multi* also in terms of computation time for $d \geq 8$. One reason for the time performance is the difference in the ranks, which the methods need. For $d = 9$, e.g., the rank needed by *Boot* is by a factor of two smaller than the rank needed by *Multi*. Additionally, it manages to solve the problem for $d =$

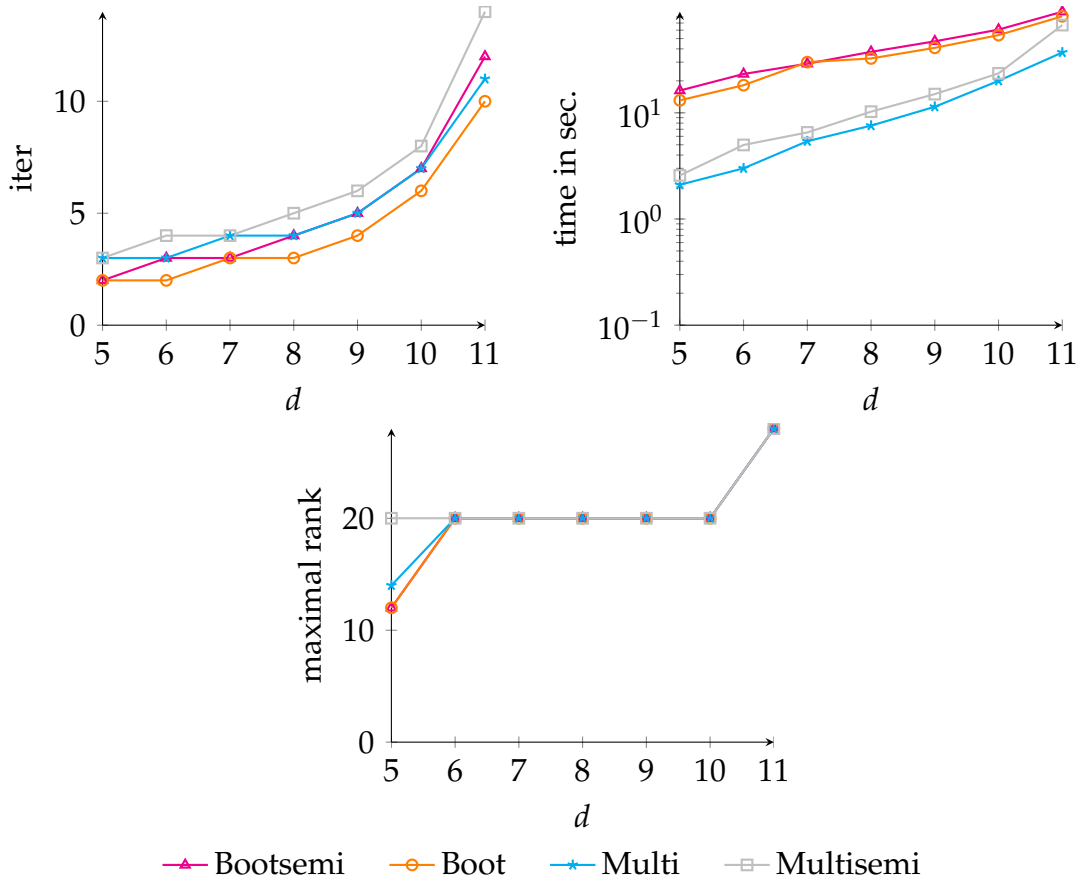


Figure 8.8: Number of iterations, running time and maximal rank for model *overflow* with mode size $n = 17$ and varying dimension d.

10 for which *Multi* fails. Recall that the model *directed_metab* does not have d local interactions (but only two instead), so these are the models where direct interpolation cannot be used to construct all small interpolation matrices $P_i, i = 1, \ldots, d$, so that we had to use linear interpolation in all dimensions but the first one (see section 6.3). Due to this, one could expect that the adaptive bootstrap approach can be advantageous, since it does not rely on local structure at all. The results for the model *directed_metab* show that this is indeed the case and one can obtain a significantly better interpolation operator (and thus performance). Unfortunately, this is not true for *diverging_metab*. Here all approaches fail at the latest for $d \geq 8$, as Figure 8.13 shows. A possible reason for this may lie in the structure of the model. Recall that we have two reaction paths, which are completely independent of each other from one point on. This structure cannot be represented in a rank-1 interpolation operator.

In order to verify that the bootstrap approach does indeed give a better multi-

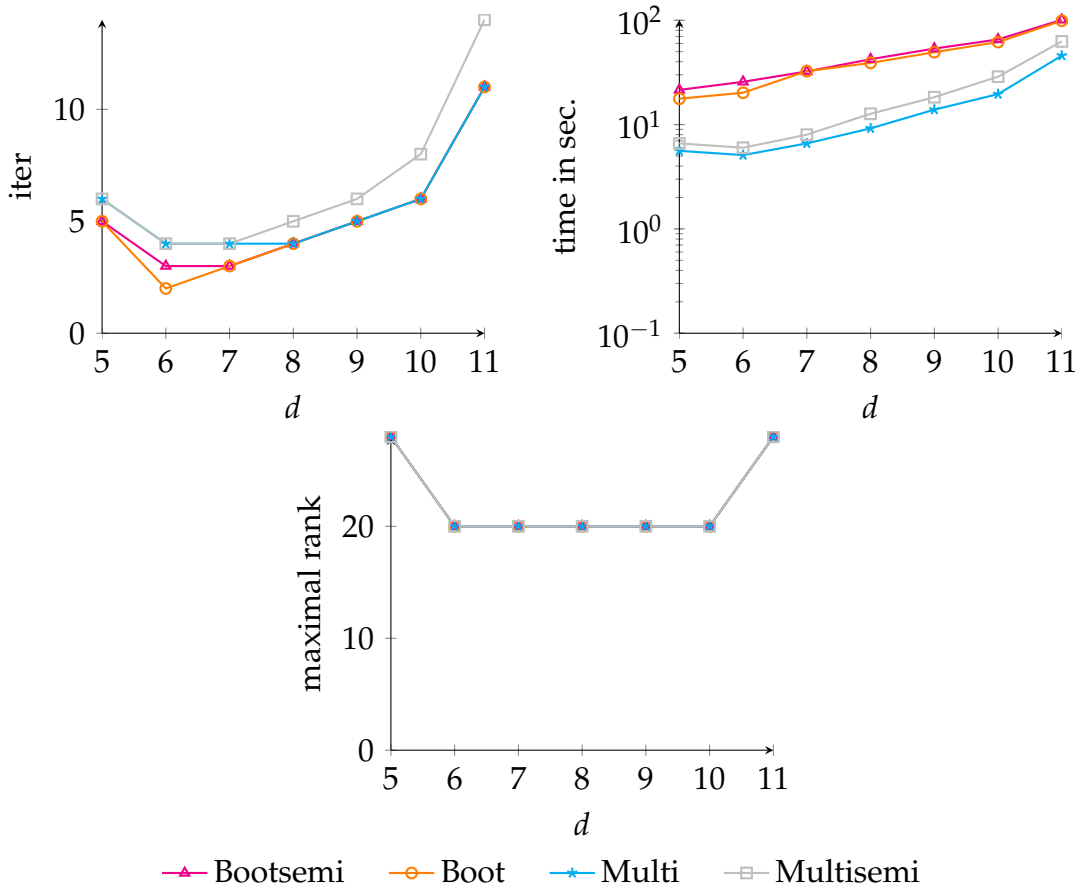


Figure 8.9: Number of iterations, running time and maximal rank for model *overflow_cycling* with mode size $n = 17$ and varying dimension d.

grid hierarchy (i.e., one which gives a better convergence factor) and not just a better starting guess, we take a closer look at the convergence history of *Boot* and *Multi* for *directed_metab* with $d = 10$ in Figure 8.14. One clearly sees that not only the residual norm at the beginning is smaller for *Boot*, but also the reduction of the residual norm from one iteration to the next. Averaged over all V-cycles, the convergence factor of *Multi* is about 0.8256 and the one of *Boot* is 0.7816.

So summarizing these experiments, we can say that the bootstrap approach works and delivers a better interpolation and starting guess than *Multi* in some cases. In general *Multi* and *Boot* are both good methods of choice. Semi-coarsening strategies should only be used if the full coarsening strategies fail, because the coarsest system is too large for solving it exactly. The bootstrap approach is particularly attractive in situations where direct interpolation cannot be applied to all subsystems and one therefore would use linear interpolation in *Multi*.

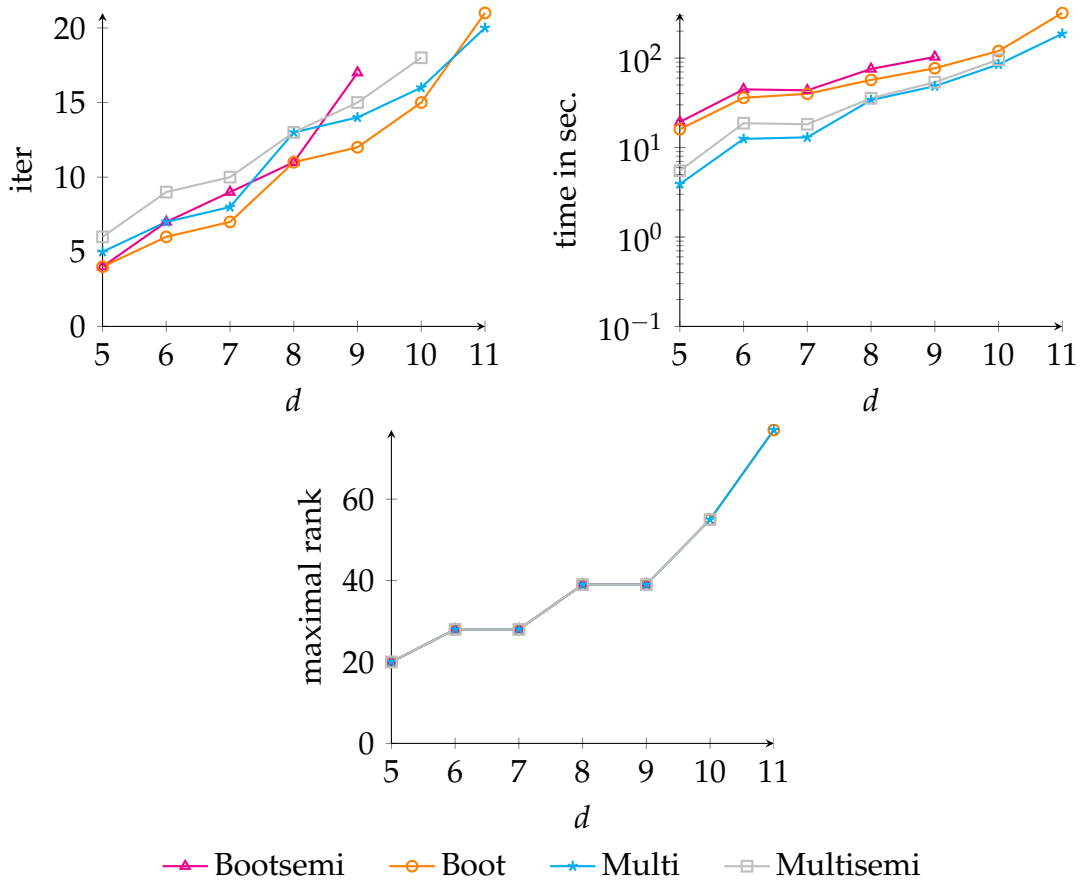


Figure 8.10: Number of iterations, running time and maximal rank for model *overflow_long* with mode size $n = 17$ and varying dimension d.

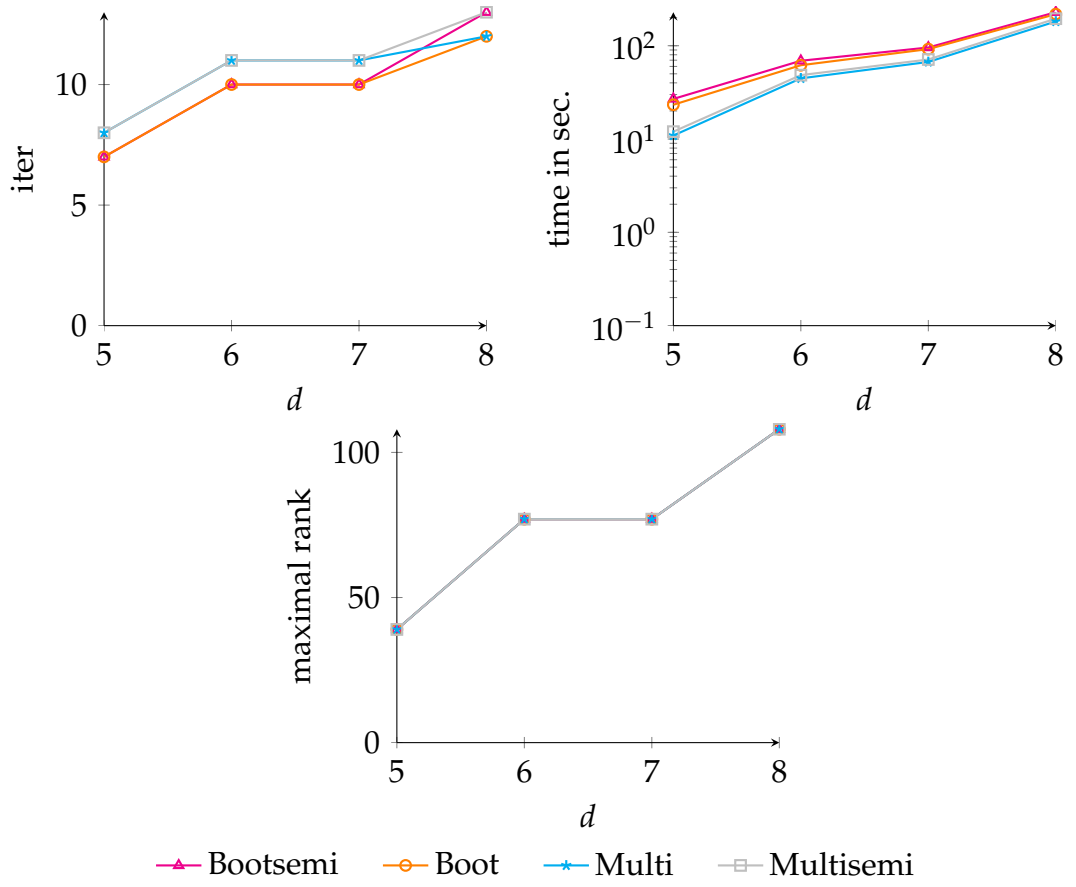


Figure 8.11: Number of iterations, running time and maximal rank for model *simplified_kanban* with mode size $n = 17$ and varying dimension d.

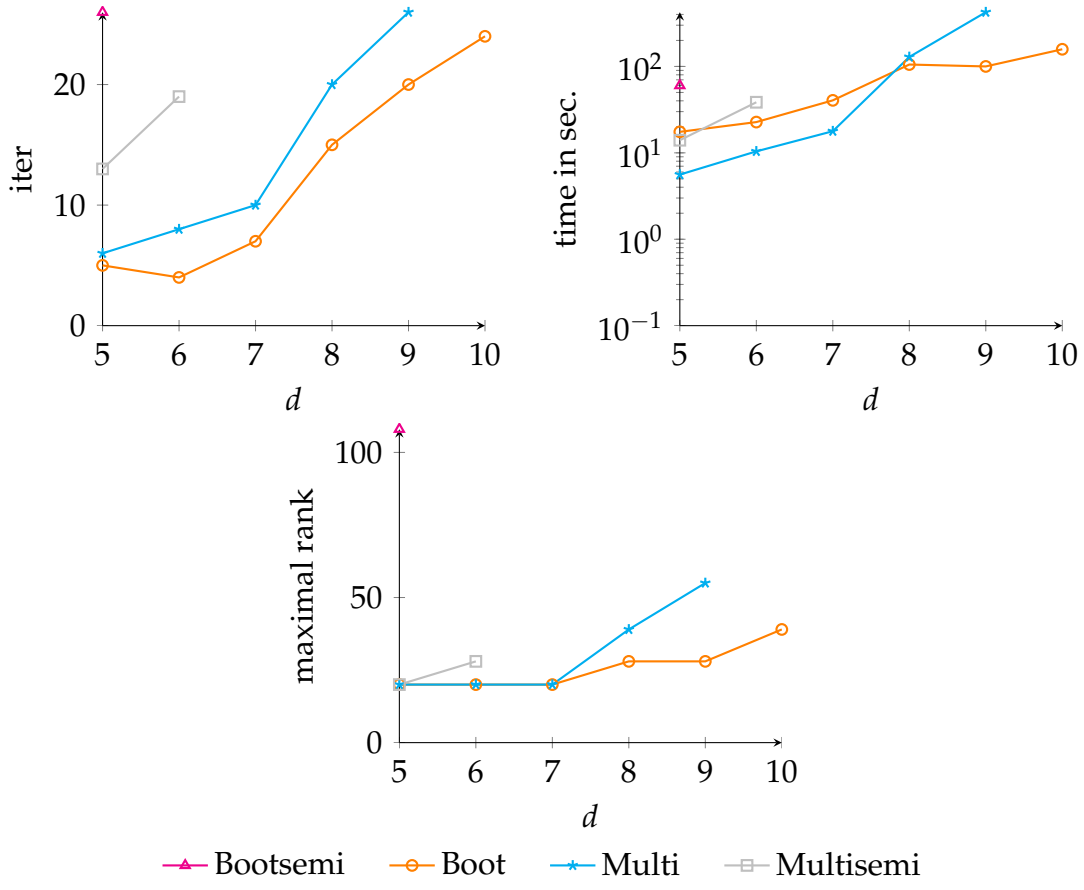


Figure 8.12: Number of iterations, running time and maximal rank for model *directed_metab* with mode size $n = 17$ and varying dimension d.

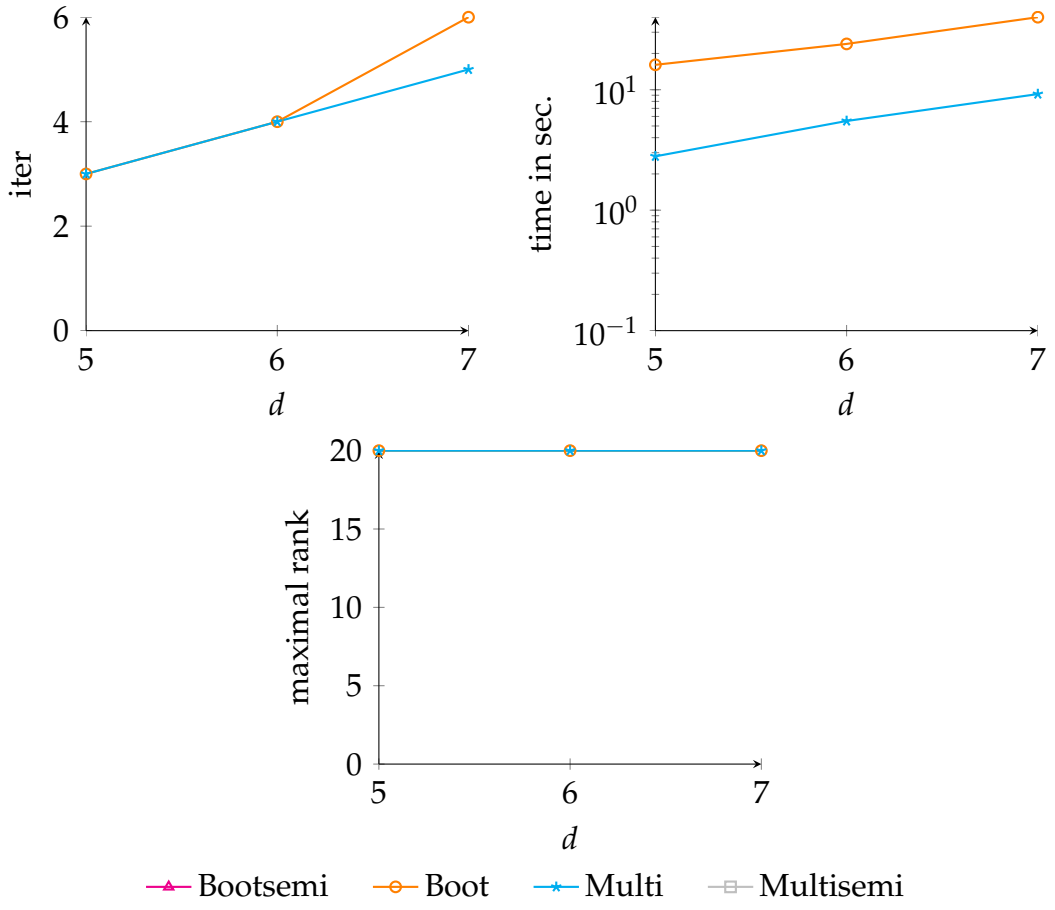


Figure 8.13: Number of iterations, running time and maximal rank for model *diverging_metab* with mode size $n = 17$ and varying dimension d.

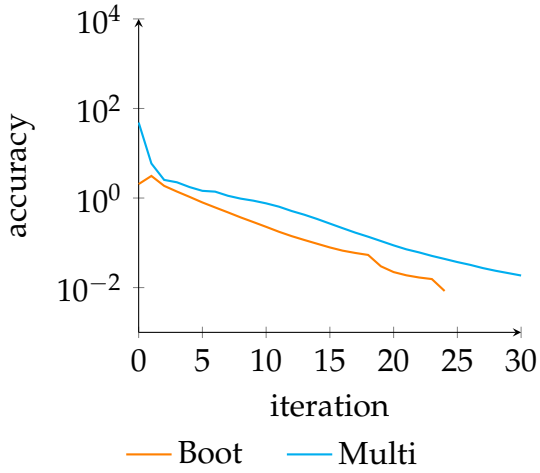


Figure 8.14: Convergence behaviour of *Boot* and *Multi* for *directed_metab* with $d = 10$ and $n = 17$.

Summary of Chapter 8:

Least squares based interpolation:

- Goal: adaptive computation of interpolation weights.
- Given test vectors $v^{(k)}$, compute P row-wise by least squares approach:

$$\min \mathcal{L}(p_{i,\bullet}) = \sum_{k=1}^{r} \omega_k \Big(v^{(k)} - \sum_{j \in \mathcal{J}_i} (p_{i,\bullet})_j (\widetilde{R} v^{(k)})_j \Big)^2 .$$

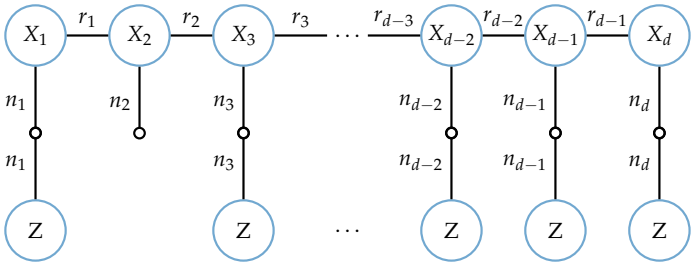
- Two kinds of test vectors: (i) smoothed random vectors,
  (ii) gen. eigenvectors of coarse grid matrix.

Tensorized Bootstrap AMG:

- Goal: Adaption of least squares approach to the case $P = \bigotimes_{i=1}^{d} P_i$.

$\Rightarrow$ Compute each row of each P_i via least squares.

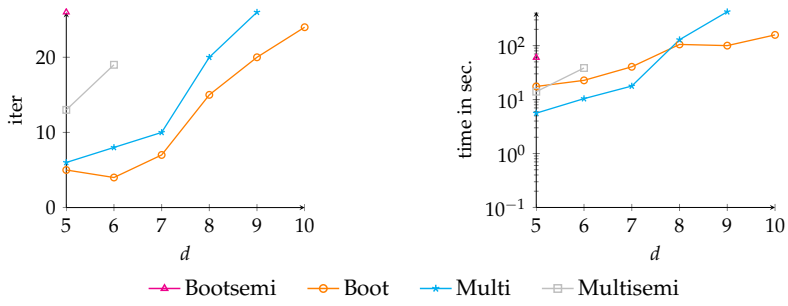
$\Rightarrow$ Test vectors need to be contracted to size n_i from $n_1 \times n_2 \times \cdots \times n_d$:



Numerical Experiments:

$\oplus$ Bootstrap improves convergence behaviour in terms of V-cycles.

$\ominus$ But this does not always compensate for the cost of the setup.



- Semi-coarsening approaches are not suitable for all models.

CHAPTER 9

Optimization based low-rank tensor methods

In chapters 7 and 8 we got a feeling how our tensorized multigrid method behaves, but by now there is no reference point how the performance of our method is to be judged in comparison to other methods which also deal with tensorized problems. This is what we now want to look into. Alternating optimization techniques are frequently used to compute approximate solutions within a low-rank tensor format. We want to motivate this for tensor structured Markov chain models. First note that the linear system (2.9) can be reformulated as

$$\min \|Ax\|_2^2 \quad \text{subject to} \quad \mathbf{1}^T x = 1. \tag{9.1}$$

Interpreting the generator matrix A as a linear operator on $\mathbb{R}^{n_1 \times \cdots \times n_d}$ and replacing x by a TT-Tensor $\mathcal{X}$, we get

$$\min \|A\mathcal{X}\|_2^2 \text{ subject to } \langle \mathcal{X}, \mathbf{1} \rangle = 1, \ \mathcal{X} \text{ is in } TT\text{-format (3.1)}, \tag{9.2}$$

where $\mathbf{1}$ now refers to the $n_1 \times \cdots \times n_d$ tensor of all ones. Using the fact that the TT-Format is linear in each of the TT-cores, an alternating least square (ALS) approach is given by successively optimizing one core at a time, while keeping all other cores fixed.

This chapter will summarize how the optimization of one core can be done in a TT-representation and will also show the limitations of ALS. These limitations will be the motivation for the alternating minimal energy (AMEn) method, a procedure which will also be discussed here. These two procedures are well discussed in [29,30], and applied to tensor structured Markov chains in [47].

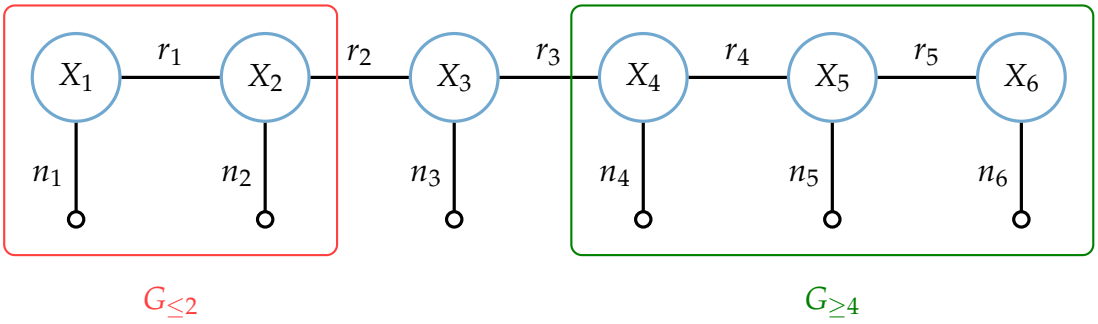


Figure 9.1: Illustration of interface matrices for $d = 6$.

9.1 Alternating least squares

For optimizing one core with respect to (9.2) we need to formulate a subproblem which performs this task. In [29,30,47,48] the *interface matrices*

$$\begin{aligned} G_{\leq k-1} &= \left[X_1(i_1)\cdots X_{k-1}(i_{k-1})\right] \in \mathbb{R}^{(n_1\cdot\ldots\cdot n_k)\times r_{k-1}}, \\ G_{\geq k+1} &= \left[X_{k+1}(i_{k+1})\cdots X_d(i_d)\right]^T \in \mathbb{R}^{(n_{k+1}\cdot\ldots\cdot n_d)\times r_k} \end{aligned}$$

are defined. This notation should be understood as follows: In $G_{\leq k-1}$, the first $k-1$ cores of the TT-representation of $\mathcal{X}$ are grouped together, and all other cores are left out, and this group (which depends on $k-1$ indices) is reshaped into a matrix. Explicit formulas for efficiently computing these matrices via Kronecker products are given in [48, Section 2.1.1]. The corresponding grouping of cores is illustrated in Figure 9.1.

With help of the interface matrices, the frame matrix

$$G_{\neq k} = G_{\leq k-1} \otimes I_{n_{k-1}} \otimes G_{\geq k+1} \tag{9.3}$$

is then constructed. The idea is now to vectorize the kth core, denoted by g_k (so $g_k \in \mathbb{R}^{r_{k-1}n_k r_k}$), to achieve

$$\operatorname{vec}(\mathcal{X}) = G_{\neq k} g_k.$$

We now insert this relation into (9.1) and obtain

$$\min \|AG_{\neq k}g_k\|_2^2 \quad \text{subject to} \quad \langle G_{\neq k}g_k, \boldsymbol{1}\rangle = 1.$$

This problem can be solved by introducing the Lagrange multiplier λ, which leads to the linear system

$$\begin{bmatrix} G_{\neq k}^T A^T A G_{\neq k} & \widetilde{e} \\ \widetilde{e}^T & 0 \end{bmatrix} \begin{bmatrix} g_k \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{9.4}$$

Algorithm 9.1: Alternating least squares method

```
Choose initial guess 𝒳 with TT-ranks r_0, ..., r_d
while stopping criterion not satisfied do
    for k = 1, 2, ..., d do
        Build G_{≠k} via (9.3) and ẽ = G_{≠k}^T 1_{g_k}
        Solve the linear system (9.4)
        Reshape g_k into a r_{k-1} × n_k × r_k tensor X̃_k
        Replace X_k by X̃_k
        Update X_k and X_{k+1} to restore orthonormality
    end
    for k = d − 1, d − 2, ..., 1 do
        Build G_{≠k} via (9.3) and ẽ = G_{≠k}^T 1_{g_k}
        Solve the linear system (9.4)
        Reshape g_k into a r_{k-1} × n_k × r_k tensor X̃_k
        Replace X_k by X̃_k
        Update X_k and X_{k-1} to restore orthonormality
    end
end
```

with $\widetilde{e} = G_{\neq k}^T \mathbf{1}_{g_k}$ and $\mathbf{1}_{g_k}$ a vector of ones of size $r_{k-1} n_k r_k$. To obtain this system, we used the fact that

$$\|Ax\|_2^2 = \|A(G_{\neq k} g_k)\|_2^2 = g_k^T (G_{\neq k}^T A^T A G_{\neq k}) g_k.$$

After solving (9.4), the new iterate $\mathcal{X}$ is constructed by reshaping g_k into its kth TT-core. An ALS procedure is now given by solving these subsystems for all different cores. Note that in [47, 48] it is recommended for a full ALS step, also called *ALS sweep*, to apply a forward sweep over the TT-cores $1, 2, \ldots, d$ and afterwards a backward sweep over the TT-cores $d-1, \ldots, 1$. In Algorithm 9.1 the whole ALS procedure is summarized. Recall that in chapter 3 we motivated that it is advantageous to keep the orthonormality condition of the TT-cores intact, so after updating a core in the ALS procedure, an orthogonalization procedure is applied, see, e.g., [47, 48]. Note that in this way the orthonormality of the interface matrices is also ensured in the subsequent optimization step. The ALS procedure described by Algorithm 9.1 does not include any rank adaptivity. The needed rank has to been known a priori, because each iterate has again the same rank as the starting guess. This can be a disadvantage in practice.

AMEn is an extension of ALS, which includes rank adaptivity. Besides, it reaches faster convergence than ALS by enriching the cores with gradient information, which we will discuss in the following.

Algorithm 9.2: Alternating minimal energy method (AMEn)

Choose initial guess $\mathcal{X}$ with TT-ranks $r_0, \ldots, r_d$
while *stopping criterion not satisfied* **do**
  for $k = 1, 2, \ldots, d$ **do**
    Build $G_{\neq k}$ via (9.3) and $\widetilde{e} = G_{\neq k}^T \mathbf{1}_{g_k}$
    Solve the linear system (9.4)
    Reshape g_k into a $r_{k-1} \times n_k \times r_k$ tensor $\widetilde{X}_k$
    Compute residual R with TT-cores $R_i, i = 1, \ldots, d$
    Truncate the residual to low (user-specified) rank
    Replace X_k by $(\widetilde{X}_k, R_k)$
    Update X_k and X_{k+1} to restore orthonormality
  end
  for $k = d-1, d-2, \ldots, 1$ **do**
    Build $G_{\neq k}$ via (9.3) and $\widetilde{e} = G_{\neq k}^T \mathbf{1}_{g_k}$
    Solve the linear system (9.4)
    Reshape g_k into a $r_{k-1} \times n_k \times r_k$ tensor $\widetilde{X}_k$
    Compute residual R with TT-cores $R_i, i = 1, \ldots, d$
    Truncate the residual to low (user-specified) rank
    Replace X_k by $(\widetilde{X}_k, R_k)$
    Update X_k and X_{k-1} to restore orthonormality
  end
end

9.2 Alternating Minimal Energy method (AMEn)

In [29,30], the spd case was considered. In the spd case one exploits the fact that solving a linear system $A\mathcal{X} = \mathcal{F}$ (with non-zero right-hand side $\mathcal{F}$) is equivalent to the minimization of the convex quadratic functional

$$\min_{\mathcal{X}} J(\mathcal{X}) = \frac{1}{2}\langle \mathcal{X}, A\mathcal{X}\rangle - \langle \mathcal{F}, \mathcal{X}\rangle. \tag{9.5}$$

This formulation has the advantage that the residual $\mathcal{R} = \mathcal{F} - A\mathcal{X}$ is a descent direction for $J(\mathcal{X})$. Thus, the idea is to perform a steepest descent step after solving the subproblem (which in the spd case has a simpler form than in (9.4)). So AMEn enriches the TT-core locally by gradient information and through this gradient information an extension of the cores is obtained. For illustrating this extension in more detail, we consider the case $d = 2$. The general case $d > 2$ then follows analogously by applying the case $d = 2$ to neighbouring cores. The AMEn method starts with an initial guess of the form $\mathcal{X} = X_1 X_2^T$ with $X_1 \in \mathbb{R}^{n_1 \times r_1}$ and $X_2 \in \mathbb{R}^{n_2 \times r_1}$. Then we optimize the first core X_1 by applying the

first step of the ALS procedure (lines 4–8 for $k = 1$ in Algorithm 9.1), giving the new core $\widetilde{X}_1$. The residual, which is, as mentioned before, equal to the negative gradient of the objective function, is then given by $\mathcal{R} = \mathcal{F} - A\mathcal{X} \approx R_1 R_2^T$. Then a steepest descent step is applied to minimize $J(\mathcal{X})$ from (9.5) which delivers the new iterate

$$\mathcal{X} + \alpha\mathcal{R} \approx \begin{pmatrix} \widetilde{X}_1 & R_1 \end{pmatrix} \begin{pmatrix} X_2 & \alpha R_2 \end{pmatrix}^T \tag{9.6}$$

where the step size is

$$\alpha = \frac{\langle \mathcal{R}, \mathcal{R} \rangle}{\langle \mathcal{R}, A\mathcal{R} \rangle},$$

see [29, Section 4.1]. So we expand the TT-cores, and the next step would be to orthogonalize the first core of the new iterate $\begin{pmatrix} \widetilde{X}_1 & R_1 \end{pmatrix}$ and to repeat the whole process for the second core, where, however, for the ALS part we only take the part X_2 of the second core. Thereby we get an approximation $\mathcal{X}$ that is at least as good as the one obtained from one forward sweep of ALS and at least as good as one step of steepest descent. The first statement is obvious, the second one comes from the fact that the part αR_2 of the second core would be overwritten anyway when solving the subproblem for $k = 2$, so that it can be left out without sacrificing accuracy, and it is enough to start from X_2. Another advantage of this observation is that it is not necessary to compute the step size α. In [30] a convergence analysis for the spd case is given and it is concluded that the practical convergence of AMEn is usually better than that of ALS.

Note that in our case the AMEn procedure is heuristic, simply for the reason that the residual $R = -A\mathcal{X}$ does not have to be a descent direction for the objective function $\|A\mathcal{X}\|_2^2$. But because of the fact that $\alpha = 0$ in (9.6) is also a possible choice for the step size (and therefore the residual information cannot worsen the approximation quality) and because of the rank adaptivity, the AMEn ansatz is still preferable over ALS for our models and for comparing to our tensorized multigrid method. This way, one can start with a very low rank initial guess (e.g., a tensor of rank one) and let the method find the rank which is needed for accurately representing the solution. The overall method for arbitrary values of d is summarized in Algorithm 9.2.

9.3 Combination of multigrid and AMEn

In the following we will investigate the computational costs of AMEn in more detail, which will then show the limitations of the method. If we afterwards consider the limitations of our tensorized multigrid method with direct interpolation and full coarsening strategy from chapter 6 again, we will see that there

are advantages if we combine these two methods. So we will describe this combination and will later be able to confirm that this combination may overcome the limitations of both methods in some cases in the numerical tests.

9.3.1 Limitations of AMEn and multigrid

In AMEn, a solution of the linear system (9.4) has to be computed in each step. The size of this system is $r_{k-1}r_k n_k + 1$ (recall the $+1$ comes from the additional restriction to the solution). If we use a direct solver, the cost would be $\mathcal{O}(\widetilde{r}^6\widetilde{n}^3))$ with $\widetilde{n} = \max_{k=1}^{d} n_k$ and $\widetilde{r} = \max_{k=1}^{d} r_k$. So if the rank and the mode sizes are large, the cost would explode. In [8] and [47], it was recommended to use an iterative solver, for example MINRES [36]. The idea behind this is that even though the matrix $G_{\neq k}^T A^T A G_{\neq k}$ is not sparse, an efficient matrix-vector multiplication can be applied because of the Kronecker structure of $G_{\neq k}^T A^T A G_{\neq k}$. However, it was observed in [8] and [47] that with increasing mode size the condition number of the matrix $G_{\neq k}^T A^T A G_{\neq k}$ also increases. As a consequence, MINRES stagnates or converges very slowly.

Recall that our tensorized multigrid method with full coarsening strategy is limited to modest values of d, see sections 6.2.1 and 7.3 and especially Figure 7.7. By now we looked at strategies to get rid of d on the coarsest grid, for example the combination with semi-coarsening, but it would also be possible to use a low-rank tensor method like AMEn for solving the coarsest system. The advantage of using AMEn on the coarsest grid is that the mode sizes of the coarsest grid operator are low, so it can be expected that the subproblems appearing in AMEn are less expensive to solve in this case.

9.3.2 MultiAMEn

So, we will test how AMEn works when using it for solving the coarsest grid system. It should be mentioned that we have a residual equation on the coarsest grid. So the right-hand side is non-zero and there is no apparent reason to demand that the solution of the coarsest system should have sum 1. So we adapted AMEn to this situation by ignoring the linear constraint and applying it to the normal equations

$$A^T A e = A^T r.$$

A schematic version of the resulting method is given as Algorithm 9.3. The presentation of the algorithm is simplified in some points for improved readability. For example the truncation tolerance is not explicitly specified.

Algorithm 9.3: MultiAMEn V-cycle

$\mathcal{X}^{(\ell)} = \text{MGAMEn}(\mathcal{F}^{(\ell)}, \mathcal{X}^{(\ell)}, \texttt{max\_rank})$
if *coarsest grid is reached* **then**
  Apply AMEn to the normal equations $(A^{(\ell)})^T A^{(\ell)} \mathcal{X}^{(\ell)} = (A^{(\ell)})^T \mathcal{F}^{(\ell)}$
else
  Perform ν_{pre} GMRES steps for $A^{(\ell)} \mathcal{X}^{(\ell)} = \mathcal{F}^{(\ell)}$ with initial guess $\mathcal{X}^{(\ell)}$
  Truncate $\mathcal{X}^{(\ell)}$ to maximal TT-rank `max_rank`
  Compute the residual $\mathcal{R}^{(\ell)} \leftarrow \mathcal{F}^{(\ell)} - A^{(\ell)} \mathcal{F}^{(\ell)}$
  Truncate $\mathcal{R}^{(\ell)}$ to maximal TT-rank `max_rank`
  Restrict $\mathcal{F}^{(\ell+1)} \leftarrow R^{(\ell)} \mathcal{R}^{(\ell)}$
  $\mathcal{X}^{(\ell+1)} \leftarrow \mathbf{0}$
  $\mathcal{E}^{(\ell+1)} = \text{MGAMEn}(\mathcal{F}^{(\ell+1)}, \mathcal{X}^{(\ell+1)}, \texttt{max\_rank})$
  Interpolate $\mathcal{E}^{(\ell)} \leftarrow P^{(\ell)} \mathcal{E}^{(\ell+1)}$
  $\mathcal{X}^{(\ell)} \leftarrow \mathcal{X}^{(\ell)} + \mathcal{E}^{(\ell)}$
  Truncate $\mathcal{X}^{(\ell)}$ to maximal TT-rank `max_rank`
  Perform ν_{post} GMRES steps for $A^{(\ell)} \mathcal{X}^{(\ell)} = \mathcal{F}^{(\ell)}$ with initial guess $\mathcal{X}^{(\ell)}$
  Truncate $\mathcal{X}^{(\ell)}$ to maximal TT-rank `max_rank`
end

But also recall the nice advantage of our tensorized multigrid method that during the setup process an initial guess can be constructed cheaply, see section 7.1, by computing the eigenvector of the coarsest grid matrix corresponding to the eigenvalue zero. This advantage should be kept also in MultiAMEn. One way is to use AMEn as it was described in Algorithm 9.2 for the coarsest system. However, it is in theory not guaranteed that this approach works well, because of the constraint $\langle \mathcal{X}, \boldsymbol{1} \rangle = 1$ for the smallest eigenvector of the coarse grid matrix. In rare cases where the eigenvector is (almost) orthogonal to the vector of all ones this constraint may thus lead to problems, because scaling the eigenvector to fulfil this constraint then means to multiply it by a very large factor. To avoid this problem we use a different strategy in our experiments in section 9.4, namely we use the same initial guess as in our standard multigrid method *Multi*. Of course this is not possible if d is very large, because it requires the explicit computation of the eigenvector corresponding to the smallest eigenvalue of the coarse grid matrix. If this is not possible one has to use AMEn on the coarsest grid or another coarsening strategy for the setup, see section 6.2.2.

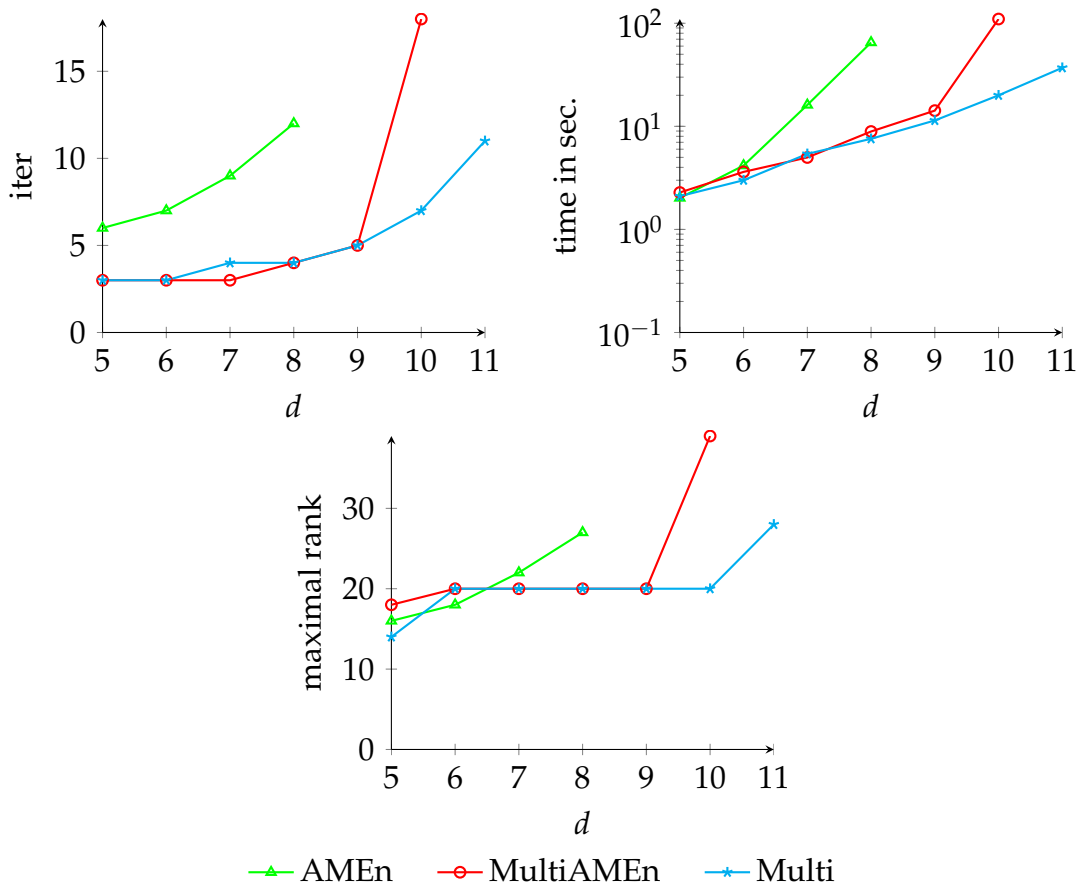


Figure 9.2: Number of iterations, running time and maximal rank for model *overflow* with mode size $n = 17$ and varying dimension d.

9.4 Numerical tests

9.4.1 Implementation details

In our next series of experiments we want to test *AMEn* and *MultiAMEn* and compare their performance to that of *Multi*. Therefore we do the same test as in section 8.3 and repeat the *Multi* results for easier reference.

For *AMEn*, we use the MATLAB implementation of the method described in [47][1]. As suggested in [47] we increase the rank by at most 3 when adding residual information to each core. The starting guess for AMEn is a tensor of all ones, scaled to have sum 1. We allow a maximum of 30 iterations, but this value

[1]The corresponding code was provided to us by Francisco Macedo.

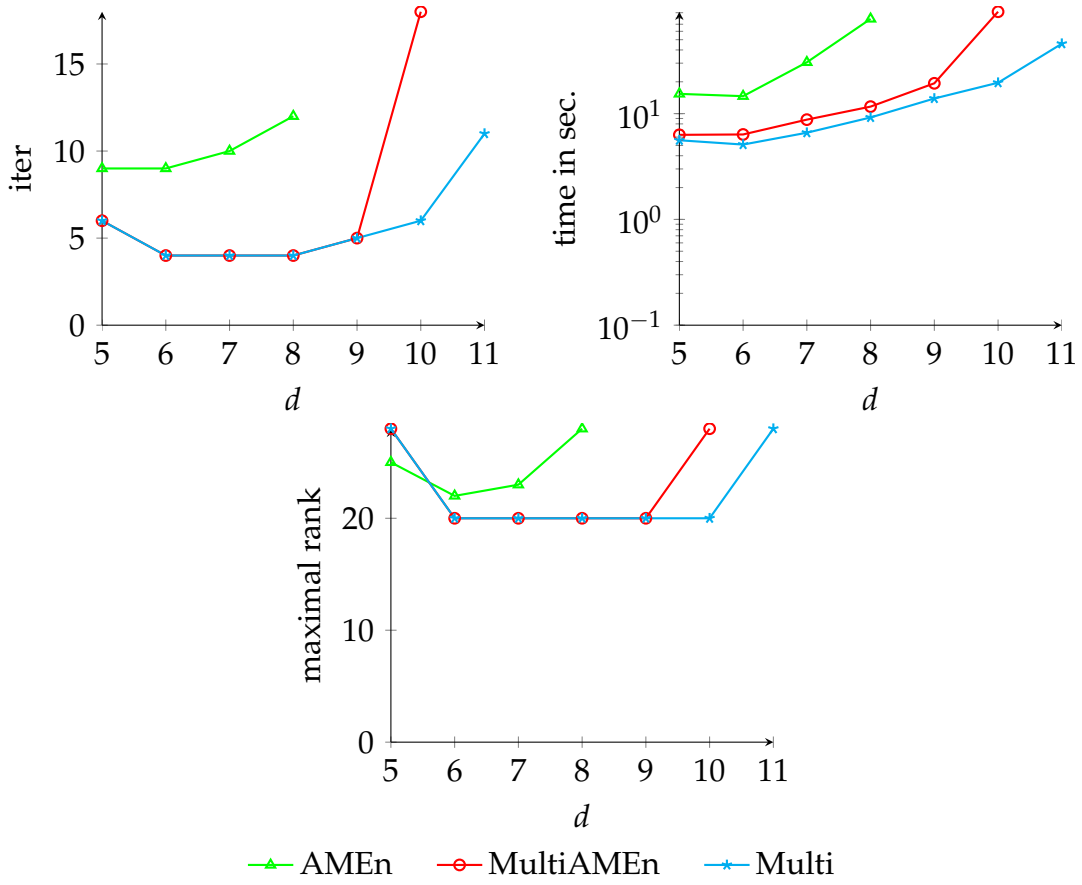


Figure 9.3: Number of iterations, running time and maximal rank for model *overflow_cycling* with mode size $n = 17$ and varying dimension d.

was never reached in the experiments here, for details see section 9.4.2.

As coarse grid solver in *MultiAMEn*, we use the MATLAB function `amen_solve2` from the *TT*-Toolbox [58]. We use the default parameters except for the enrichment rank, where we also use 3.

Because *MultiAMEn* is meant to alleviate the curse of dimensionality on the coarsest grid we coarse each subsystem to a size of 5 when varying the mode sizes. In this series of experiments the coarsest system is thus of size $5^6 = 15{,}625$. Therefore we get the opportunity to potentially achieve a better convergence behaviour compared to the method *Multi* from chapter 6, because we have one level less in the multigrid hierarchy. Note that the multigrid hierarchy in *MultiAMEn* is the same as in *Multi* except for the missing last level. With the same number of levels one could of course not expect *MultiAMEn* to perform better than the standard multigrid method, as the time for solving the least squares

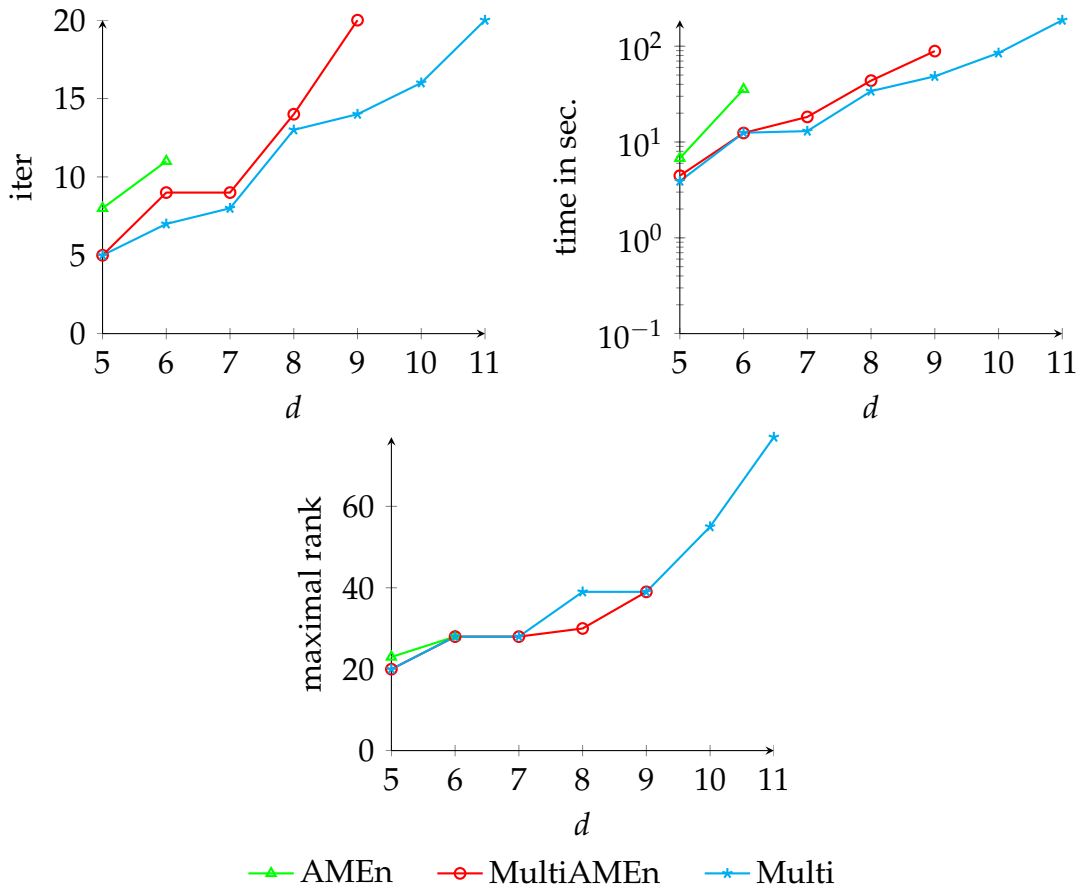


Figure 9.4: Number of iterations, running time and maximal rank for model *overflow_long* with mode size $n = 17$ and varying dimension d.

problem on the coarsest system of size 3^6 is negligible, and so using AMEn as a coarse solver has no benefit. When varying d we coarse each subsystem to 3. Therefore the largest coarse system is of size $3^{11} = 177{,}147$.

In [8] we already presented results of similar experiments involving the three methods. These experiments differ in some points from the ones presented here, because some of the parameters which are used were changed. So we use another initial guess for *MultiAMEn*, the coarsening strategy for *Multi* and *MultiAMEn* is different and the computations were done on another machine. Nevertheless, the scaling and the qualitative behaviour are similar, although some iteration numbers and running times differ.

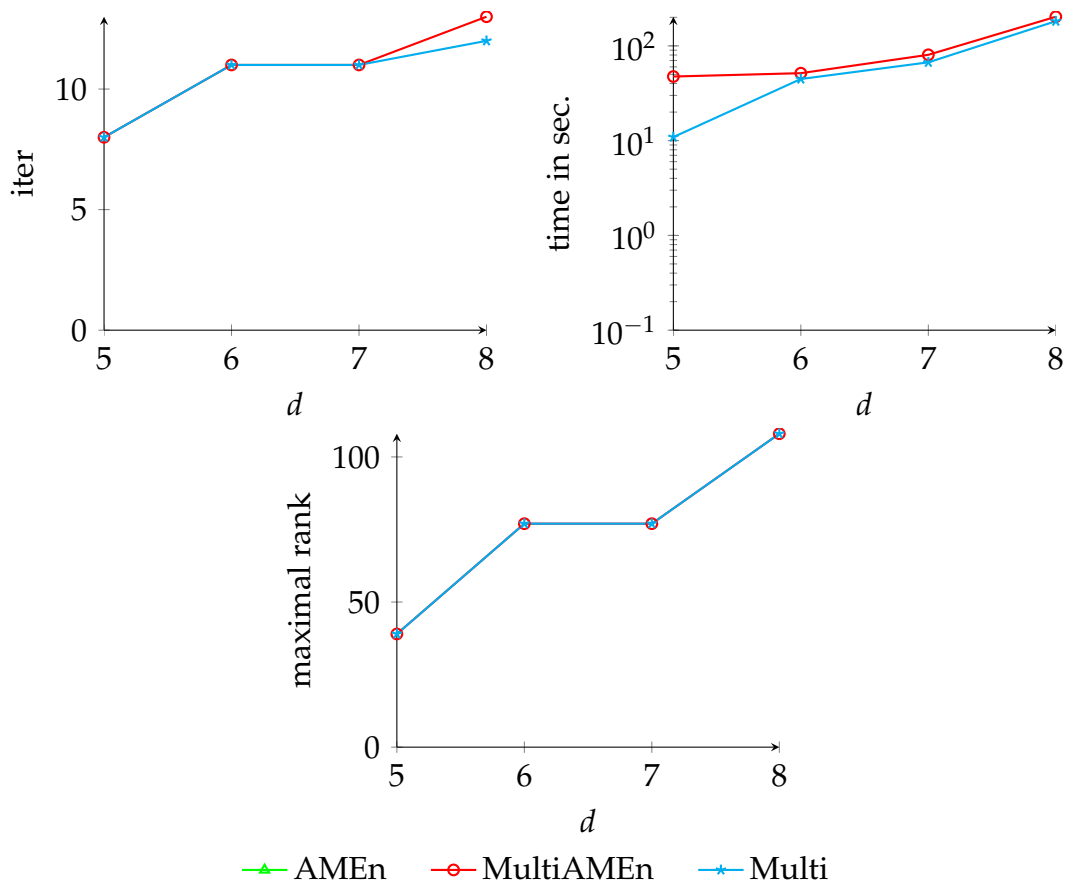


Figure 9.5: Number of iterations, running time and maximal rank for model *simplified_kanban* with mode size $n = 17$ and varying dimension d.

9.4.2 Numerical tests for AMEn and MultiAMEn

Figures 9.2–9.7 illustrate the behaviour of the methods for varying values of d, while the mode sizes are fixed to $n = 17$. We again depict the number of iterations, the running time and the needed maximal rank. Note that the number of AMEn iterations is mainly given for sake of completeness, but cannot be compared to the number of multigrid iterations as completely different operations are necessary within these iterations. So the focus of these experiments is on the running time.

Because of the large number of results we begin by first giving an overview of the behaviour of each of the methods and then compare the methods and summarize our findings afterwards as in section 8.3.

We observe that AMEn fails for all models when $d \geq 9$ (because of memory

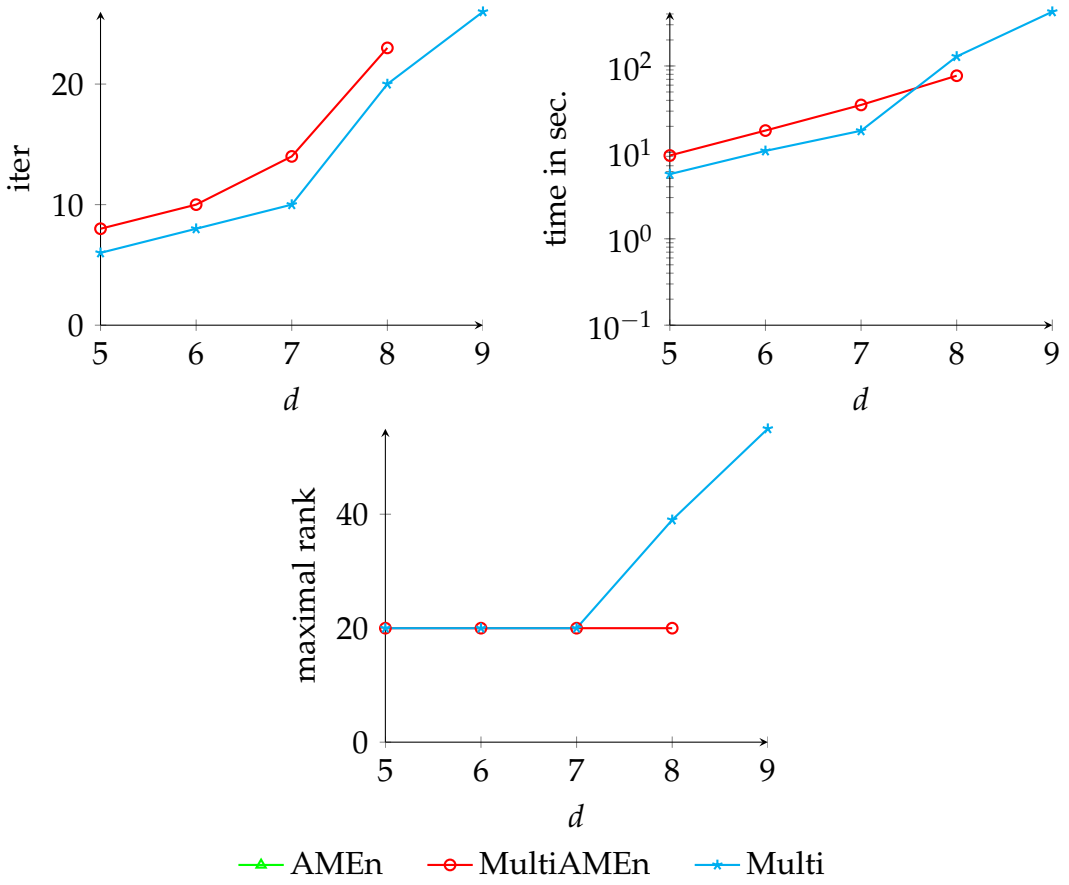


Figure 9.6: Number of iterations, running time and maximal rank for model *directed_metab* with mode size $n = 17$ and varying dimension d.

constraints). It is also apparent that AMEn has a superlinear time increase for growing d in the experiments, for example in Figure 9.2. The reason for this mainly is that already $n = 17$ is a mode size which is often too large for AMEn. Recall that solving the subproblems (9.4) in AMEn consumes most of the running time and that the size of these subproblems depends on the mode sizes of the model. So these results are not very surprising and are in line with the experiments in [8].

The two models for which AMEn worked particularly badly are *simplified_kanban* and *directed_metab*, see Figures 9.5 and 9.6. For *simplified_kanban* the reason for this are the very high ranks which are needed for accurately representing the solution. Because the subproblem size depends quadratically on the ranks, these sizes get too big even for $d = 5$. The reason for *directed_metab* is the low convergence rate of AMEn for this model. Thus, many iterations were performed and the rank was unnecessarily increased by the enrichment steps. For *overflow* and

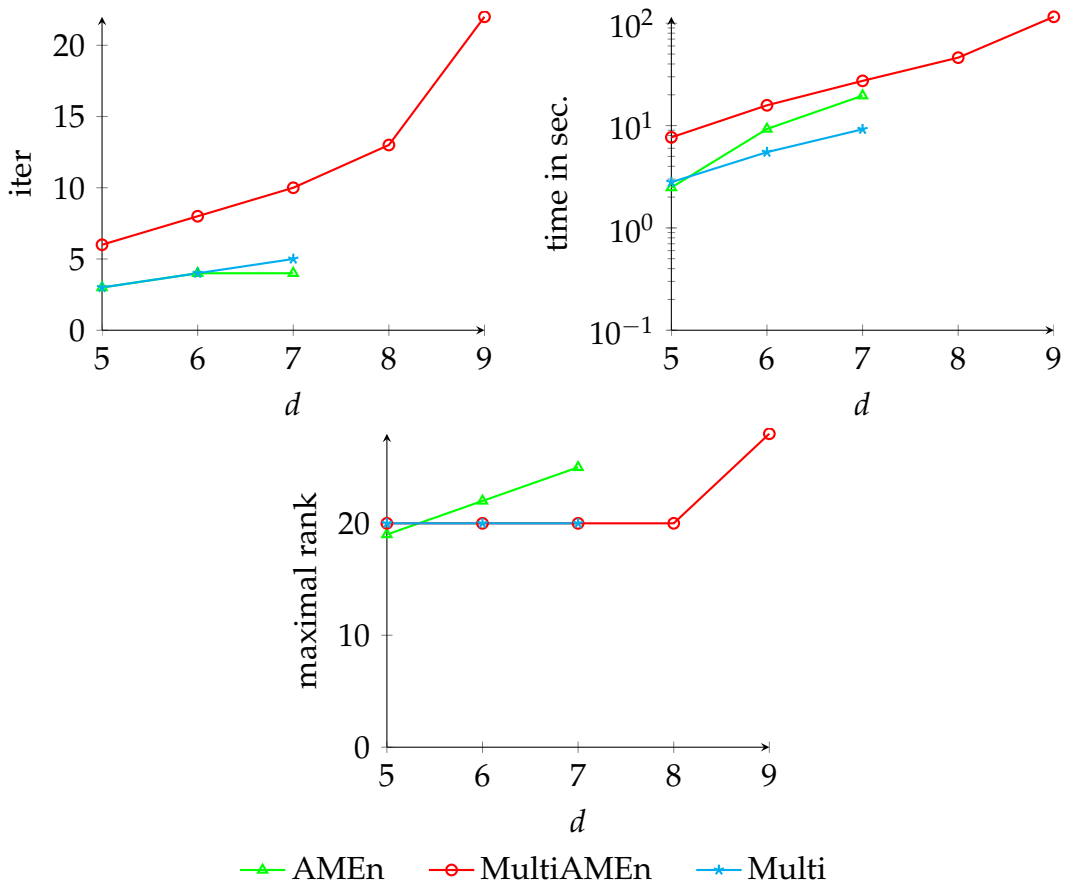


Figure 9.7: Number of iterations, running time and maximal rank for model *diverging_metab* with mode size $n = 17$ and varying dimension d.

overflow_cycling the needed ranks are smaller and so AMEn behaves better for these models than for the others, as can be seen from Figures 9.2 and 9.3.

The targeted better convergence of *MultiAMEn*, due to the lower number of levels in the multigrid hierarchy (for $d \geq 8$) can unfortunately not be observed for most of the models. Instead, we observe the following: In many cases almost the same results as for *Multi* are obtained. In some experiments, however, it can be observed that *MultiAMEn* needs significantly more iterations (and therefore also more computation time) than *Multi*, see for example Figure 9.3, $d = 10$. The reason for this is that the convergence rate degrades, because the coarsest system is not solved accurately enough by AMEn. On the other hand for the model *diverging_metab* (see Figure 9.7) *MultiAMEn* clearly outperforms the other methods. Here the lower number of levels in comparison to *Multi* seems to play an important role for the convergence rate of the method *MultiAMEn*.

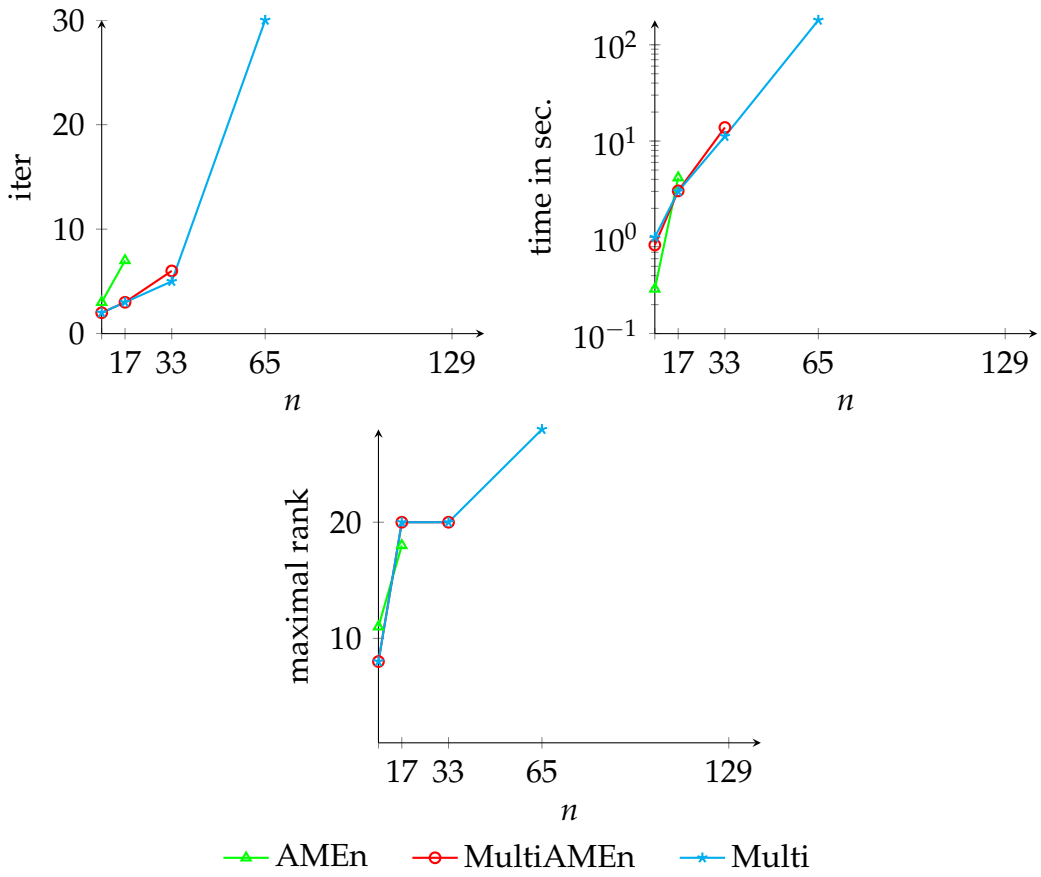


Figure 9.8: Number of iterations, running time and maximal rank for model *overflow* with dimension $d = 6$ and varying mode sizes n.

Now we look at the experiments where we vary the mode size while keeping $d = 6$ fixed. Figures 9.8–9.13 illustrate the results of these tests. Again the iteration number, the computation time and the needed maximal rank are presented.

We first describe the behaviour of the method *AMEn* and observe that it fails for $n \geq 33$ for all models. This is again based on the fact that larger subproblems cause running times which exceed 3600 seconds or memory demands beyond the size of main memory. As in the previous series of experiments, the performance is especially bad for the models *simplified_kanban* and *directed_metab*, see Figures 9.11 and 9.12.

Although we have a lower number of levels (recall for these experiments we coarse every subproblem to 5 instead of 3) *MultiAMEn* does not perform better than *Multi*. For the three overflow models and *simplified_kanban*, both multigrid methods behave virtually identically, with the exception that *MultiAMEn* fails

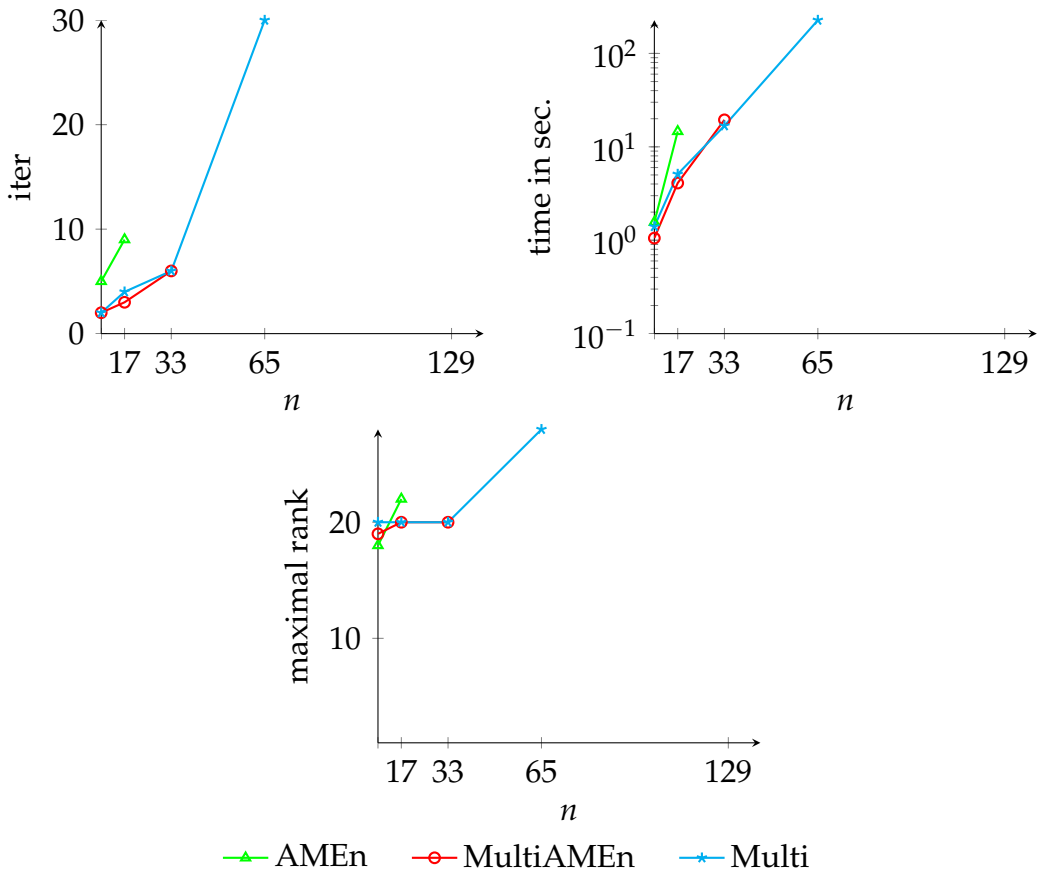


Figure 9.9: Number of iterations, running time and maximal rank for model *overflow_cycling* with dimension $d = 6$ and varying mode sizes n.

to solve *overflow* and *overflow_cycling* for $n = 65$ within the maximum allowed number of iterations because its speed of convergence was slightly worse than that of *Multi*, which solved those problems after exactly 30 iterations. For *directed_metab* and *diverging_metab* (see Figures 9.12 and 9.13) *MultiAMEn* fails for $n \geq 33$. For *directed_metab* this can be explained by the poor performance of *AMEn* for this model even for small mode sizes. For *diverging_metab* this behaviour is surprising and cannot fully be explained. In [8] this problem is not reported, and in further experiments not reported here we could also achieve convergence by altering some of the parameters of the coarse grid solver (e.g., increasing the enrichment rank to 5, lowering the tolerance by a factor of 100 and doubling the number of allowed sweeps led to convergence for all $n \leq 65$ and $d = 6$). These altered parameters however led to worse convergence behaviour for other models. So as mentioned in [8] the problem-dependent tuning of the parameters in AMEn is also an important issue.

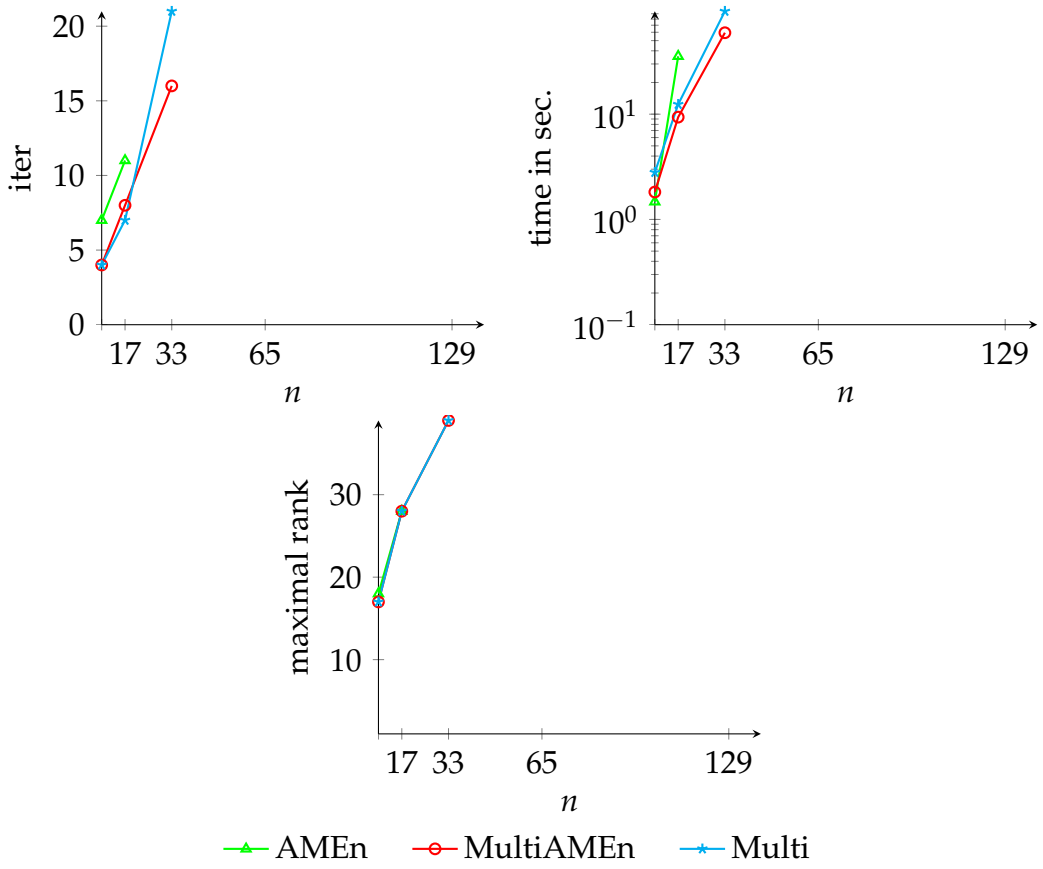


Figure 9.10: Number of iterations, running time and maximal rank for model *overflow_long* with dimension $d = 6$ and varying mode sizes n.

Summarizing, we have the following observation: AMEn is not suitable for models with larger mode sizes, but it is the method of choice for smaller ones. For example, in [47] it is shown that AMEn converges rapidly for the model *overflow_long* with $n = 2$ and $d = 24$. In this situation a multigrid method cannot be used because the mode sizes cannot be reduced further. So both approaches are complementary in the sense that their application areas are very different.

Concerning the comparison of *Multi* and *MultiAMEn*, we cannot give a universal advice on which method to prefer. For most of the test cases, the performance of both methods was very similar, and for the few cases in which the performance differed, sometimes *Multi* showed better results, and sometimes *MultiAMEn* did. One big difference between both methods is that the performance of *MultiAMEn* depends on a lot of parameters, and the best values are very problem-dependent. A suboptimal choice of these parameters may even lead to divergence of the method. Such problems do not occur for *Multi*, which is thus preferable if the user has no detailed knowledge of the model to be solved.

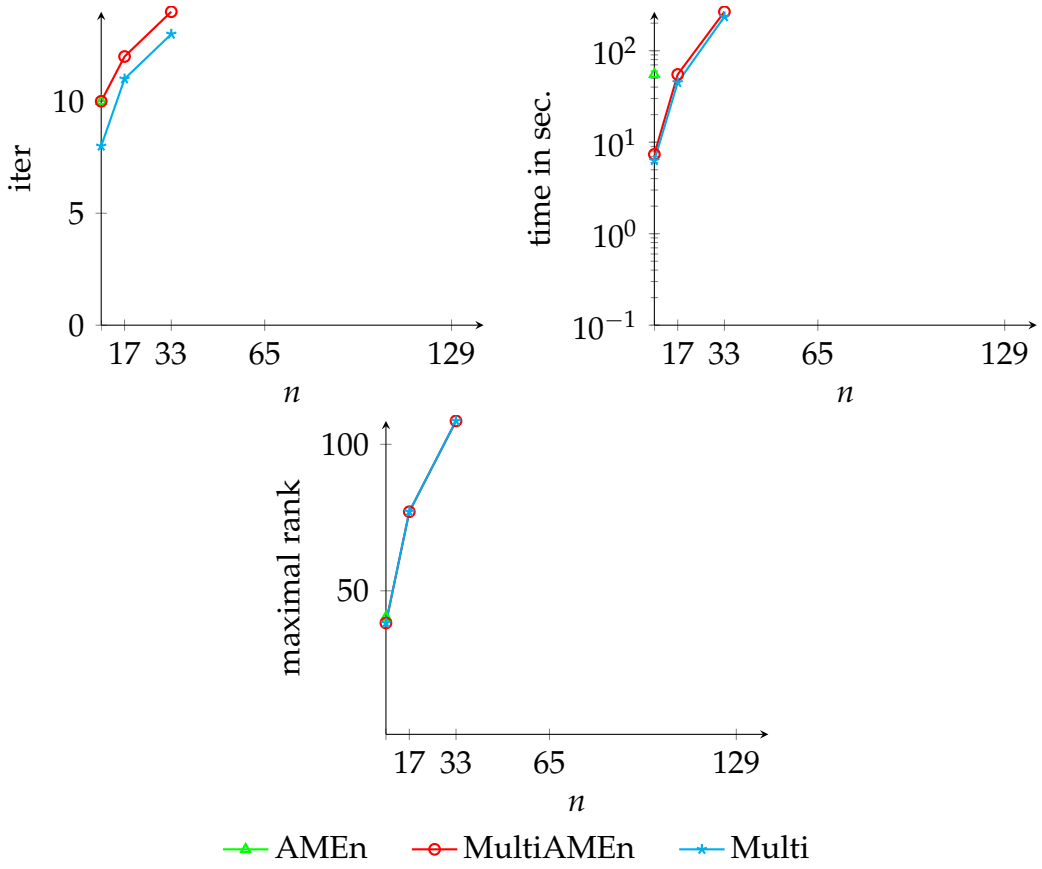


Figure 9.11: Number of iterations, running time and maximal rank for model *simplified_kanban* with dimension $d = 6$ and varying mode sizes n.

We did not discuss the needed ranks of the methods so far, because there is almost no difference between them. This can be seen as a sign that the ranks are really needed for representing the solution and not only when performing the methods. This is an indicator that conceptually all three methods are well suited for tensor structured Markov chains.

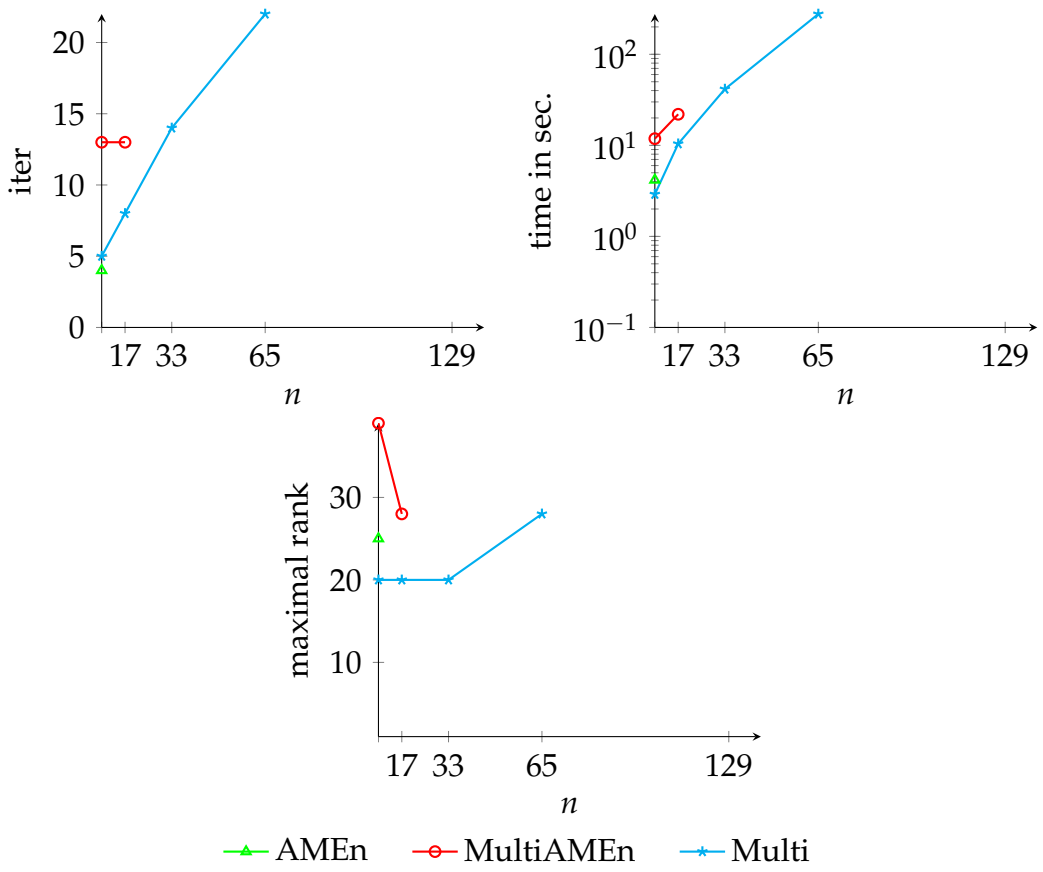


Figure 9.12: Number of iterations, running time and maximal rank for model *directed_metab* with dimesnion size $d = 6$ and varying mode sizes n.

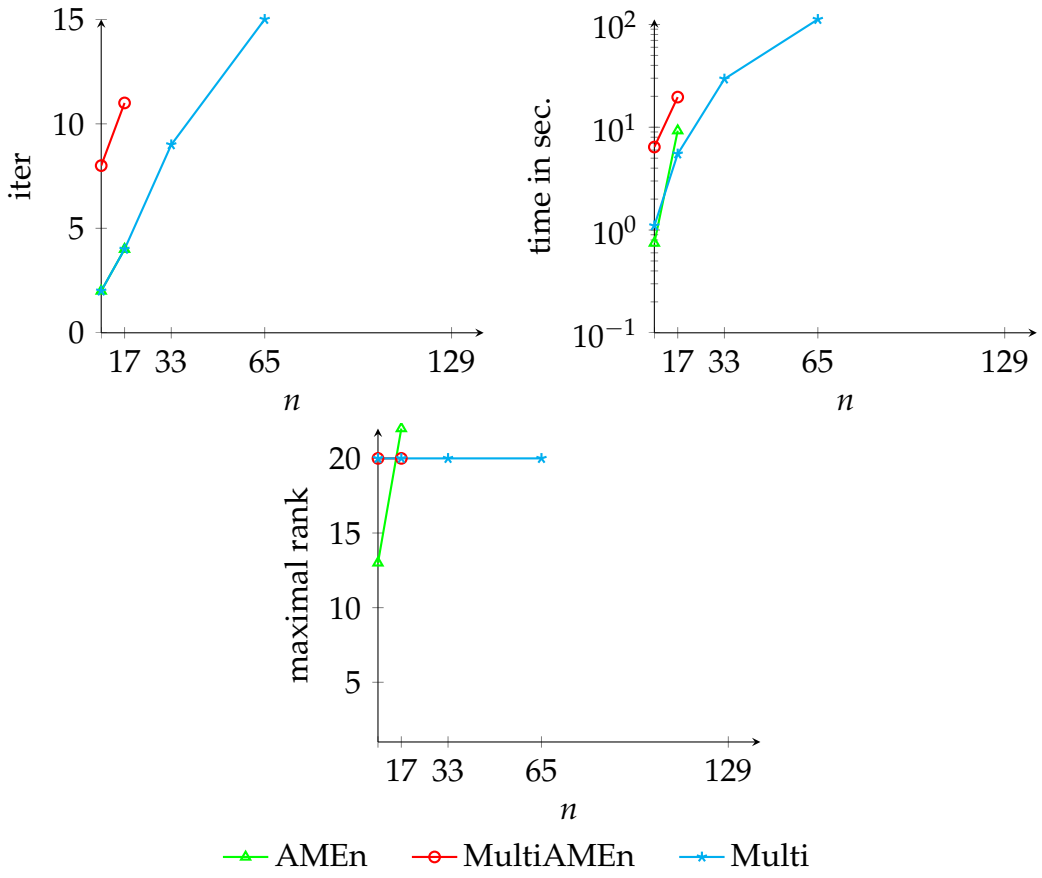


Figure 9.13: Number of iterations, running time and maximal rank for model *diverging_metab* with dimesnion size $d = 6$ and varying mode sizes n.

Summary of Chapter 9:

AMEn:

- *AMEn* is an extended alternating least squares approach.

$\Rightarrow$ kth subproblem: solve the linear system

$$\begin{bmatrix} G_{\neq k}^T A^T A G_{\neq k} & \widetilde{e} \\ \widetilde{e}^T & 0 \end{bmatrix} \begin{bmatrix} g_k \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

$\oplus$ Residual information is added to the cores for rank adaptivity.

$\ominus$ Cost of solving one subproblem: $\mathcal{O}(n^3 r^6)$.

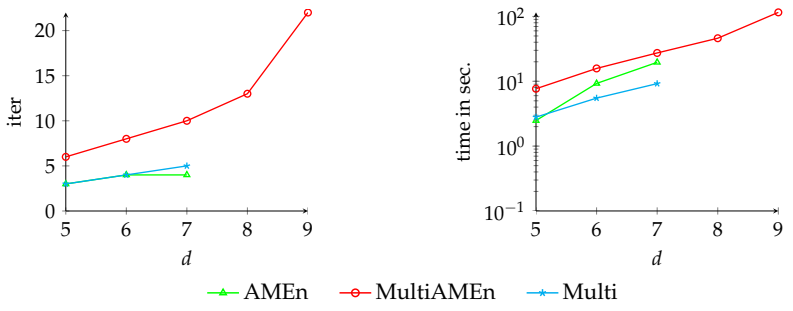
MultiAMEn:

- *AMEn* struggles with large n, but can deal with large d.
- *Multi* struggles with large d, but can deal with large n.

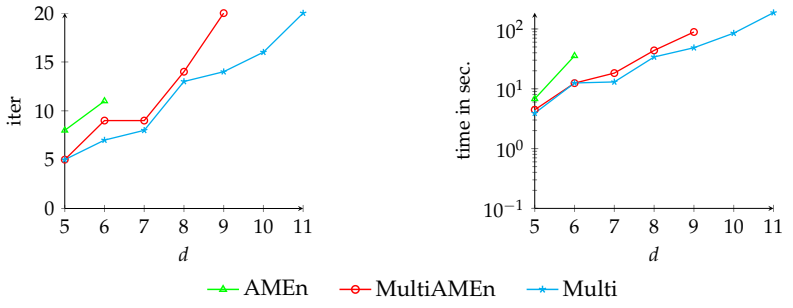
$\Rightarrow$ Combination of both methods could overcome their limitations.

Numerical Experiments:

- As was expected, *AMEn* reaches its limitations quite fast.
- At times, *MultiAMEn* solves problems where *Multi* and *AMEn* fail.



- In other cases, coarse solution is too inexact and *Multi* works better.



Chapter 10

Concluding remarks

In this thesis we developed a tensorized multigrid method for high-dimensional tensor structured continuous time Markov chains. The challenge was keeping the tensor structure intact on each level of the multigrid hierarchy. Thereby we looked at different smoothing schemes, transfer operators and coarsening strategies to fulfil these tasks efficiently. First the geometric structure of the considered models was decisive for building the multigrid hierarchy. In a first series of numerical tests we could show which of a variety of smoothers, interpolation operators and coarsening strategies are promising.

We then extended our method in several ways: We developed an adaptive method for constructing the interpolation operators by generalizing the bootstrap AMG approach to tensor structured problems. The difficulty in this was to find appropriate test vectors for the least squares interpolation within the tensor structure. We therefore introduced a scheme which allows to contract tensor structured test vectors in a suitable way. In a subsequent series of numerical tests we could show that this tensorized bootstrap approach is also promising and was in almost all cases at least as good (in terms of iteration numbers) as the multigrid method using hand-tailored interpolation operators and even clearly better in some cases.

For putting the behaviour of our multigrid method into context we compared it to another state of the art method for solving tensor structured problems, namely the low-rank method AMEn (alternating minimal energy). Thereby we observed that the limitations of AMEn and our method motivate to combine both methods into one method. In a last series of experiments we tested whether the combined method overcomes the limitations of the individual methods. At the same time we also compared the behaviour of AMEn to that of our multigrid method. The most important conclusion from these experiments is that the methods developed in this thesis are able to solve high-dimensional problems

which are completely out of reach for methods like AMEn. However we have also seen that our approach has problems with models which have a more complicated interaction between the submodels (in particular when the submodels cannot be arranged into a one-dimensional chain).

Further research topics can be seen in an extension for the tensorized bootstrap method by generalizing compatible relaxation, a method which generates the set of coarse variables adaptively, to tensor structured matrices.

Another extension would be to look at rank-k interpolation operators with $k > 1$, while in this thesis we always used a rank-1 interpolation. This extension may be useful for problems like *diverging_metab*, where there are groups of subsystems which do not interact with each other. Obviously, rank-k interpolation would have the disadvantage that the coarse grid matrix would have higher TT-ranks than the original generator matrix. In addition, more truncation would need to be done in the solution process. But the higher number of degrees of freedom may be useful in special cases.

In this thesis we only looked at continuous time Markov chains, but there also exist tensor structured problems in other fields, for example high-dimensional partial differential equations which are discretized on a regular grid. An application of our method to these models would also be interesting.

Danksagung

Natürlich haben zahlreiche Personen zum Gelingen dieser Arbeit beigetragen, bei denen ich mich hier bedanken möchte:

Beginnend bei Prof. Dr. Andreas Frommer, der mir überhaupt die Möglichkeit zu promovieren verschafft hat und mit zahlreichen Vorschlägen und Ideen zur Seite stand, und auch für die Möglichkeit an vielen Konferenzen teilnehmen zu dürfen. Prof. Dr. Matthias Bolten und Dr. Karsten Kahl für viele hilfreiche Stunden in denen ich auch viel Neues gelernt habe, sei es wegen Fragen zur Programmierung oder dem Verständnis, oder wegen des Lesens und Korrigierens von von mir geschriebenen Artikeln.

Des Weiteren auch ein Dankeschön an Prof. Dr. Daniel Kressner, da durch ein Gespräch mit ihm die Idee zum Thema meiner Dissertation entstanden ist, und an Francisco Macedo, der mir seinen Matlabcode für AMEn zur Verfügung gestellt hat.

Auch wenn nicht direkt an der Dissertation beteiligt, gilt mein Dank auch Dr. Holger Arndt und Prof. Dr. Bruno Lang mit denen ich immer sehr gerne in der Lehre zusammengearbeitet habe, und der ganzen Arbeitsgruppe "Angewandte Informatik" mit der nicht nur die Arbeit, sondern auch die Freizeit immer ein Vergnügen war und hoffentlich weiterhin sein wird. Natürlich auch ein besonderes Dankeschön an meine Familie und meine Freunde, die ich hier nicht alle namentlich erwähnen kann. Ihr wisst hoffentlich wie wichtig ihr mir alle seid.

Und zuletzt einen besonderen Dank an Marcel & Maja, mit euch bin ich gewachsen, in jeglicher Hinsicht.

Menschen mit einer neuen Idee gelten solange als Spinner, bis sich die Sache durchgesetzt hat. -Mark Twain-

List of Figures

List of Tables

List of Algorithms

List of Notations

$\mathbb{R}$	the field of real numbers
$\mathbb{R}^n$	n-dimensional Euclidean vector space over $\mathbb{R}$
$\mathbb{R}^{m\times n}$	the space of real-valued $m \times n$ matrices
$P[X(t) = i]$	probability that the random variable X has the value i at time t
$P[X(t) = i \mid X(s) = j]$	conditional probability that the random variable X has the value i at time t under the condition that it had value j at time s
$E[X]$	expectation value of the random variable X
A^T	the transpose of the matrix A
$A^\dagger$	the Moore-Penrose pseudo inverse of the matrix A
A^D	the Drazin inverse of the matrix A
$range(A)$	the range of the matrix A
$N(A)$	the nullspace of the matrix A
$\|\cdot\|_2$	the Euclidean vector or matrix norm
$\|\cdot\|_F$	the Frobenius matrix norm
$\boldsymbol{1}$	the vector of all ones
$\mathbf{0}$	the vector of all zeros
I	the identity matrix
$A_1 \otimes A_2$	the Kronecker product of the matrices A_1 and A_2
$\bigotimes_{i=1}^{d} A_1$	the Kronecker product $A_1 \otimes A_2 \otimes \cdots \otimes A_d$
$S_1 \oplus S_2$	the direct sum of the subspaces S_1 and S_2
$S^\perp$	the orthogonal complement of the subspace S
$K_m(A, b)$	the mth Krylov subspace corresponding to the matrix A and the vector b
$C_1 \times C_2$	the Cartesian product of the sets C_1 and C_2
$o(\Delta t)$	Landau-o, a quantity that goes to zero faster than Δt

Bibliography

[1] D. F. ANDERSON, G. CRACIUN, AND T. G. KURTZ, *Product-form stationary distributions for deficiency zero chemical reaction networks*, Bull. Math. Biol., 72 (2010), pp. 1947–1970.

[2] M. ARIOLI, V. PTÁK, AND Z. STRAKOŠ, *Krylov sequences of maximal length and convergence of GMRES*, BIT, 38 (1998), pp. 636–643.

[3] A. BEN-ISRAEL AND T. N. E. GREVILLE, *Generalized Inverses: Theory and Applications*, John Wiley & Sons, Inc., 1974.

[4] A. BERMAN AND R. J. PLEMMONS, *Nonnegative Matrices in the Mathematical Sciences*, SIAM, 1994.

[5] D. A. BINI, G. LATOUCHE, AND B. MEINI, *Numerical Methods for Structured Markov Chains*, Oxford Science Publications, 2005.

[6] M. BOLTEN, A. BRANDT, J. BRANNICK, A. FROMMER, K. KAHL, AND I. LIVSHITS, *A bootstrap algebraic multilevel method for Markov chains*, SIAM J. Sci. Comput., 33 (2011), pp. 3425–3446.

[7] M. BOLTEN, S. FRIEDHOFF, A. FROMMER, M. HEMING, AND K. KAHL, *Algebraic multigrid methods for Laplacians of graphs*, Linear Algebra Appl., 434 (2011), pp. 2225–2243.

[8] M. BOLTEN, K. KAHL, D. KRESSNER, F. MACEDO, AND S. SOKOLOVIĆ, *Multigrid methods combined with low-rank approximation for tensor structured Markov chains*, arxiv e-print 1605.06246, 2016. (Submitted to Electron. Trans. Numer. Anal.).

[9] M. BOLTEN, K. KAHL, AND S. SOKOLOVIĆ, *Multigrid methods for tensor structured Markov chains with low rank approximation*, SIAM J. Sci. Comput., 38 (2016), pp. A649–A667.

[10] D. BRAESS, *Towards algebraic multigrid for elliptic problems of second order*, Computing, 55 (1995), pp. 379–393.

[11] A. BRANDT, J. BRANNICK, K. KAHL, AND I. LIVSHITS, *Bootstrap AMG*, SIAM J. Sci. Comput., 33 (2011), pp. 623–632.

[12] J. BRANNICK, K. KAHL, AND S. SOKOLOVIĆ, *An adaptively constructed algebraic multigrid preconditioner for irreducible Markov chains*, arxiv e-print 1402.4005, 2014.

[13] M. BREZINA, T. A. MANTEUFFEL, S. F. MCCORMICK, J. RUGE, AND G. SANDERS, *Towards adaptive smoothed aggregation (αSA) for nonsymmetric problems*, SIAM J. Sci. Comput., 32 (2010), pp. 14–39.

[14] P. BUCHHOLZ, *Structured analysis approaches for large Markov chains*, Appl. Numer. Math., 31 (1999), pp. 375–404.

[15] ——, *Multilevel solutions for structured Markov chains*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 342–357.

[16] ——, *Product form approximations for communicating Markov processes*, Perform. Eval., 67 (2010), pp. 135–144.

[17] P. BUCHHOLZ AND T. DAYAR, *Comparison of multilevel methods for Kronecker-based Markovian representations*, Computing, 73 (2004), pp. 349–371.

[18] S. L. CAMPBELL AND C. D. MEYER, *Generalized Inverses of Linear Transformations*, Pitman London San Francisco Melbourne, 1979.

[19] R. CHAN, *Iterative methods for overflow queueing networks I*, Numer. Math., 51 (1987), pp. 143–180.

[20] ——, *Iterative methods for overflow queueing networks II*, Numer. Math., 54 (1988), pp. 57–78.

[21] K. L. CHUNG, *Markov Chains with Stationary Transition Probabilites*, Springer-Verlag Berlin Heidelberg New York, 1967.

[22] T. F. COLEMAN AND Y. LI, *A reflective newton method for minimizing a quadratic function subject to bounds on some of the variables*, SIAM J. Optim., 6 (1996), pp. 1040–1058.

[23] H. DE STERCK, *A self-learning algebraic multigrid method for extremal singular triplets and eigenpairs*, SIAM J. Sci. Comput., 34 (2012), pp. A2092–A2117.

[24] H. DE STERCK, T. A. MANTEUFFEL, S. F. MCCORMICK, K. MILLER, J. RUGE, AND G. SANDERS, *Algebraic multigrid for Markov chains*, SIAM J. Sci. Comput., 32 (2010), pp. 544–562.

[25] H. DE STERCK, T. A. MANTEUFFEL, S. F. MCCORMICK, Q. NGUYEN, AND J. RUGE, *Multilevel adaptive aggregation for Markov chains, with application to web ranking*, SIAM J. Sci. Comput., 30 (2008), pp. 2235–2262.

[26] H. DE STERCK, T. A. MANTEUFFEL, S. F. MCCORMICK, J. RUGE, K. MILLER, J. PEARSON, AND G. SANDERS, *Smoothed aggregation multigrid for Markov chains*, SIAM J. Sci. Comput., 32 (2010), pp. 40–61.

[27] H. DE STERCK AND K. MILLER, *An adaptive algebraic multigrid algorithm for low-rank canonical tensor decomposition*, SIAM J. Sci. Comput., 35 (2013), pp. B1–B24.

[28] S. V. DOLGOV, *TT-GMRES: solution to a linear system in the structured tensor format*, Russ. J. Numer. Anal. Math. Model., 28 (2013), pp. 149–172.

[29] S. V. DOLGOV AND D. V. SAVOSTYANOV, *Alternating minimal energy methods for linear systems in higher dimensions. part I: SPD systems*, arxiv e-print 1301.6068, 2013.

[30] ——, *Alternating minimal energy methods for linear systems in higher dimensions. part II: Faster algorithm and application to nonsymmetric systems*, arxiv e-print 1304.1222, 2013.

[31] C. ECKART AND G.YOUNG, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), p. 211–218.

[32] J. S. EDWARDS, M. COVERT, AND B. PALSSON, *Metabolic modelling of microbes: the flux-balance approach*, Environ. Microbiol., 4 (2002), pp. 133–140.

[33] G. H. GOLUB AND C. F. V. LOAN, *Matrix Computations, 3rd edition*, Johns Hopkins University Press, 1996.

[34] L. GRASEDYCK, *Hierarchical singular value decomposition of tensors*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2029–2054.

[35] L. GRASEDYCK AND W. HACKBUSCH, *A multigrid method to solve large scale Sylvester equations*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 870–894.

[36] A. GREENBAUM, *Iterative Methods for Solving Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[37] A. GREENBAUM, V. PTÁK, AND Z. STRAKOŠ, *Any nonincreasing convergence curve is possible for GMRES*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 465–469.

[38] W. HACKBUSCH, *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin, Heidelberg, 1985.

[39] ——, *Tensor Spaces and Numerical Tensor Calculus*, Springer-Verlag, Berlin, Heidelberg, 2012.

[40] ——, *Iterative Solution of Large Sparse Systems of Equations*, Springer-Verlag, New York, NY, USA, 2016.

[41] W. HACKBUSCH AND S. KÜHN, *A new scheme for the tensor representation*, J. Fourier Anal. Appl., 15 (2009), pp. 706–722.

[42] I. C. F. IPSEN AND C. D. MEYER, *The idea behind Krylov methods*, Amer. Math. Monthly, 105 (1998), pp. 889–899.

[43] K. KAHL, *Adaptive Algebraic Multigrid for Lattice QCD Computations*, PhD thesis, Bergische Universität Wuppertal, Fachbereich Mathematik und Naturwissenschaften, Wuppertal, 2009.

[44] L. KAUFMAN, *Matrix methods for queuing problems*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 525–552.

[45] H. A. L. KIERS, *Towards a standardized notation and terminology in multiway analysis*, J. Chemometrics, 14 (2000), pp. 105–122.

[46] T. G. KOLDA AND B. W. BADER, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.

[47] D. KRESSNER AND F. MACEDO, *Low-rank tensor methods for communicating Markov processes*, in Quantitative Evaluation of Systems, G. Norman and W. Sanders, eds., vol. 8657 of Lecture Notes in Computer Science, Springer, 2014, pp. 25–40.

[48] D. KRESSNER, M. STEINLECHNER, AND A. USCHMAJEW, *Low-rank tensor methods with subspace correction for symmetric eigenvalue problems*, SIAM J. Sci. Comput., 36 (2014), pp. A2346–A2368.

[49] D. KRESSNER AND C. TOBLER, *Algorithm 941: Htucker—a Matlab toolbox for tensors in hierarchical Tucker format*, ACM Trans. Math. Softw., 40 (2014), pp. 22:1–22:22.

[50] S. P. LALLEY, *Continuous-time Markov chains*, 2012. lecture notes, available at galton.uchicago.edu/~lalley/Courses/313/ContinuousTime.pdf.

[51] A. N. LANGVILLE AND W. J. STEWART, *The Kronecker product and stochastic automata networks*, J. Comput. Appl. Math., 167 (2004), pp. 429–447.

[52] E. LEVINE AND T. HWA, *Stochastic fluctuations in metabolic pathways*, Proc. Natl. Acad. Sci. U.S.A., 104 (2007), pp. 9224–9229.

[53] F. MACEDO, *Benchmark problems on stochastic automata networks in tensor train format*, tech. rep., MATHICSE, EPF Lausanne, Switzerland, 2015. Available from `http://anchp.epfl.ch/SAN_TT`.

[54] S. F. MCCORMICK AND J. W. RUGE, *Multigrid methods for variational problems*, SIAM J. Numer. Anal., 19 (1982), pp. 924–929.

[55] D. MITRA AND I. MITRANI, *Analysis of a kanban discipline for cell coordination in production lines, II: Stochastic demands*, Oper. Res., 39 (1991), p. 807–823.

[56] I. V. OSELEDETS, *Approximation of $2^d \times 2^d$ matrices using tensor decomposition*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2130–2145.

[57] ——, *Tensor-Train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317.

[58] I. V. OSELEDETS AND S. DOLGOV, *MATLAB TT-Toolbox Version 2.2*, 2011. Available at `http://spring.inm.ras.ru/osel/?page\_id=24`.

[59] I. V. OSELEDETS AND S. V. DOLGOV, *Solution of linear systems and matrix inversion in the TT-format*, SIAM J. Sci. Comput., 34 (2012), pp. A2718–A2739.

[60] B. PLATEAU AND J.-M. FOURNEAU, *A methodology for solving markov models of parallel systems*, J. Parallel Distrib. Comput., 12 (1991), pp. 370–387.

[61] B. PLATEAU AND W. J. STEWART, *Stochastic automata networks*, in Computational Probability, W. K. Grassmann, ed., Kluwer Academic Press, 2000, pp. 113–152.

[62] J. RUGE AND K. STÜBEN, *Algebraic multigrid*, in Multigrid Methods, S. F. McCormick, ed., SIAM, Philadelphia, 1986, pp. 73–130.

[63] Y. SAAD, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelphia, 2003.

[64] Y. SAAD AND M. H. SCHULTZ, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Comput., 7 (1986), pp. 856–869.

[65] G. SANDERS, *Extensions to Adaptive Smoothed Aggregation Multigrid (αSA): Eigensolver Initialization and Nonsymmetric Problems*, PhD thesis, University of Colorado, Boulder, CO, 2008.

[66] W. J. STEWART, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, 1994.

[67] K. STÜBEN, *A review of algebraic multigrid*, J. Comput. Appl. Math., 128 (2001), p. 281—309.

[68] U. TROTTENBERG, C. OSTERLEE, AND A. SCHÜLLER, *Multigrid*, Academic Press, 2001.

[69] E. UYSAL AND T. DAYAR, *Iterative methods based on splittings for stochastic automata networks*, Eur. J. Oper. Res., 110 (1998), pp. 166–186.

[70] P. VANEK, J. MANDEL, AND M. BREZINA, *Algebraic multigrid on unstructured meshes*, UCD/CCM Report, 34 (1994).

[71] P. VANĚK, J. MANDEL, AND M. BREZINA, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), p. 179–196.

[72] P. VANĚK, *Fast multigrid solver*, Appl. Math., 40 (1995), pp. 1–20.

[73] E. VIRNIK, *An algebraic multigrid preconditioner for a class of singular M-matrices*, SIAM J. Sci. Comput., 29 (2007), pp. 1982–1991.