Computing High-Dimensional Value Functions of Optimal Feedback Control Problems using the Tensor-Train Format

vorgelegt von

M. Sc.
Leon Jasper Sallandt

an der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
- Dr. rer. nat. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Peter Bank
Gutachter:	Prof. Dr. Reinhold Schneider
Gutachter:	Prof. Dr. Karl Kunisch
Gutachter:	Prof. Dr. Tobias Breiten

Tag der wissenschaftlichen Aussprache: 6. Dezember 2021

Berlin 2022

Acknowledgements

First of all, I want to thank my advisor Reinhold Schneider for your supervision, numerous advises and interest in my work. Sharing your drive to explore new application fields of our methods has greatly helped me developing this thesis. I also want to thank Karl Kunisch and Tobias Breiten for taking your time to review my thesis.

I am thankful for my three closest colleagues Mathias Oster, who, especially in the beginning of my three years, worked together with me every day, Michael Götte, who introduced the infamous Michagsseminar which greatly helped the team spirit and communication of our work group - especially when the home-office season started, and Philipp Trunschke, whose advises and ideas were always helpful. I would like to thank Alexandra Schulte for keeping the group organized and helping me wherever it was possible.

I would like to thank Konstantin Fackeldey, Lorenz Richter, Nikolas Nüsken, Martin Eigel, Christian Bayer and Philipp Krah for the collaborations and discussions. Additionally, I want to thank Henrik Eisenmann discussing and proofreading my dissertation and but also for the great time we had while studying. I would like to thank the DFG and the graduate school DAEDALUS (GRK 2433) for providing the funding for my employment.

I want to thank my parents Gisela and Peter, and my brothers Oliver and Henry for your interest in my research and your support not only during the making of this thesis but during my whole life. Last but not least, I want to thank Laura for being there for me and your unconditional support.

Abstract

We consider high-dimensional, non-linear functional equations. These functional equations are mostly the Bellman equation known from optimal control or related fields. Within this framework we deal with the occurring non-linearity using fixed-point iterations, for the most part the Policy Iteration algorithm, reducing them to a series of linear problems. These linear problems suffer from the so-called curse of dimensionality. We apply hierarchical tensor formats, in particular tensor-trains, to represent the sought function. Here, we also make use of an extension of the tensor-train format, where single functions can be added into the function space. The linear problems are approximated by regression and minimal residual formulations, which means that high-dimensional integrals appear. We apply Monte Carlo methods to estimate these integrals. Applying this framework, we compute feedback controllers of infinite and finite horizon optimal control problems. For the finite horizon case we also consider an algorithm based on open-loop control and provide a novel error propagation bound. We also consider the case of stochastic exit-time control problems. Finally, we consider a regression approach in the context of parabolic partial differential equations, which can be reformulated to backward stochastic differential equations. In this context, we apply the tensor-train model and compare to state-of-the-art neural network methods with respect to run-time and accuracy. We numerically observe that for many problems, low-rank approximation of the sought functions can be found, yielding close to optimal feedback controllers.

Deutsche Kurzzusammenfassung

Wir betrachten hochdimensionale, nicht-lineare Funktionengleichungen, wie zum Beispiel die Bellmangleichung, bekannt aus dem Gebiet der optimalen Steuerung. Die auftretende nicht-linearität behandeln wir mit Fixpunktiterationen, insbesondere der Policy Iteration, und erhalten damit eine Folge von linearen Problemen. Diese Probleme leiden in hohen Dimensionen unter dem sogenannten Fluch der Dimensionalität (curse of dimensionality), was wir mit der Verwendung von hierarchischen Tensorformaten, insbesondere Tensor-Trains, behandeln. Wir stellen damit die gesuchten Funktionen dar und verwenden auch eine Erweiterung des Konzepts, bei der einzelne Funktionen in den Funktionenraum hinzugefügt werden. Die auftretenden linearen Probleme werden dann mithilfe von Regression und ähnlichen Methoden gelöst. Die daher auftretenden hoch-dimensionalen Integrale werden mithilfe von Monte Carlo Methoden approximiert.

Mithilfe dieses Ansatzes werden optimale Feedbacksteuerungen von verschiedenen Optimalsteuerungsproblemen berechnet - von (deterministischen) Problemen mit endlichem und unendlichem Zeithorizont zu stochastischen Problemen mit Exit-Bedingung. Schlussendlich werden noch allgemeine semi-lineare parabolische Differentialgleichungen mithilfe von backward stochastic differential equations gelöst, wobei wir die Ergebnisse mit state-of-the-art neuronalen Netz Methoden vergleichen. Hier achten wir auf die Genauigkeit der Ergebnisse und auf die Laufzeit des Algorithmus. Wir beobachten numerisch, dass für viele Probleme gute Approximationen der gesuchten Funktionen, und damit auch des optimalen Feedbackgesetzes, mithilfe des Tensor-Train Ansatzes gefunden werden können.

Contents

Chapter 1

Introduction and Outline

Controlling systems constrained by *ordinary* and *partial differential equations* (ODE/PDE) is a subject of major importance for engineering and science. Originating from the calculus of variations, optimal control problems are optimization problems, where a cost functional shall be minimized and a governing differential equation is given as a constraint. This ODE describes the behavior of an initial state in dependence of a given control. In that sense, an optimal trajectory over time is sought. Several variations of optimal control problems are known and some of them are covered in this thesis. For example, the constraining ODE can be replaced by a *stochastic differential equation* (SDE), the *time-horizon* can be either finite, infinite, or depending on the trajectory.

In the 1950s and 1960s the foundations of modern optimal control were set by the groups of Pontryagin, who introduced *Pontryagin's Maximum Principle* (PMP) [Pon+62] and Bellman, who introduced *Dynamic Programming* [Bel57]. While Pontryagin's approach leads to so-called *open-loop* optimal control problems, Bellman's approach leads to *closed-loop* optimal control problems, which are also called *feedback control* problems in the literature. In applications and also in the mathematical theory, the difference of these two approaches is fundamental.

In the open-loop approach, the initial state is measured and using knowledge of the system and the cost functional, an optimal control is computed. For finding a numerical approximation of the optimal control, standard gradient descent methods can be used, which makes it an appealing tool for usage. The drawback of this method is its inflexibility in the presence of uncertainties and errors in the model. To give an example: Using numerical methods, it is possible to compute the optimal trajectory of an airplane even before it takes off. However, if conditions such as the wind change once the airplane is in the air, the target might be missed, leading to devastating consequences. These conditions are encoded in the ODE. Open-loop control seeks a mapping from the initial state to the optimal controlled trajectory and is oftentimes used only for a single initial value.

In contrast to that, closed-loop control remeasures the state while the process takes place and is thus able to account for uncertainties within the model. Using the airplane example, a closed-loop controller measures the current time and the current state and computes the optimal control for exactly this time-point. In order to obtain an optimal trajectory, it is of course necessary to recompute the control for different time-points. As this should happen while the state is moving, the evaluation of the feedback law has to be fast, which is a

draw-back when compared to the open-loop approach, where a long optimization can be done before the airplane takes off. However, while the closed-loop controller cannot directly account for the changed dynamics, the controller measures that its position is not where it expected to be and reacts by changing the control, making it more robust with respect to (w.r.t.) perturbations. In this thesis we mainly consider solution methods of the closed-loop problem.

In order to solve the closed-loop problem, the so-called *value function* plays a central role. The value function obeys governing equations such as the *Bellman equation* and the *Hamilton-Jacobi-Bellman* (HJB) equation and the theoretical and numerical treatment these equations received and still receive is immense. One of the main theoretical breakthroughs is the discovery of a new solution concept of a PDE - the *viscosity solution*. This concept allows for irregular solutions and the terminology was introduced by P.-L. Lions and Crandall in the 1980s [CL83]. In this thesis the (possibly) irregularity of the value function is not a main concern. This issue mainly occurs when constraints on the set of states or controls are imposed. Instead, we cover problems where the value function is (assumed to be) smooth and use methods for smooth approximation. These problems are all unconstrained problems with smooth cost functional and state dynamics.

However, another difficulty of the equations occurs - the non-linearity of the Bellman and HJB equation. To alleviate this problem, different specialized methods have been proposed in the literature. In this thesis we make use of the *Policy Iteration* algorithm which is also known as *Howard's algorithm*. This algorithm reduces the non-linear equation to a sequence of linear equations and is able to solve the Bellman equation and the HJB equation. For certain optimal control problems we link the PMP to the value function, which allows us to compute feedback laws using the PMP. Moreover, we also apply a non-specialized method which can be solved to solve HJB equations with a stochastic dynamic, but also general (semi-linear) parabolic PDEs. All our approaches have in common, that a PDE or functional equation has to be solved iteratively, either by iterating backwards in time, or by iteratively solving linear equations to approximate a non-linear equation.

Another difficulty appears when the governing control problems, and thus the state space, become high-dimensional. The value function has to be encoded as a function from the state space to the real numbers and if the spatial dimension becomes large, even representing such a function becomes challenging. Traditional methods to represent multi-dimensional functions rely on tensorization of one-dimensional ansatz functions. Most prominent are probably multi-dimensional polynomials, where a set of n one-dimensional polynomials and a d-dimensional state space yields a n^d-dimensional function space. We call this model, as it forms a linear function space, a linear model. For $\mathbf{5}$ one-dimensional ansatz functions and a $\mathbf{32}$-dimensional space, this yields a $\mathbf{5^{32} \approx 10^{22}}$ dimensional function space, which means that for storing the function, more than one zettabyte disc space has to be allocated. This is more than $\mathbf{10^{12}}$ gigabyte. Function spaces or more generally data of this form are said to suffer from the *curse of dimensionality*, which describes exactly the exponential growth of the dimension of the function space with increasing spatial dimension.

For certain optimal control problems, the dimension of the function space scales moderately with the dimension of the state space. This is the case of so-called *linear quadratic problems*. This class of problems has a linear state dynamics and a quadratic cost functional. For problems of this type the structure of the value function is known exactly - it is a quadratic

function. Moreover, the HJB equation can in this case be reformulated to a matrix equation - the *Riccati equation*. This reformulation gives rise to efficient numerical algorithms from linear algebra to solve the Riccati equation. Provided that the cost functional stays quadratic, feedback controllers for non-linear systems can be obtained by linearizing the dynamical system and then solving the corresponding Riccati equation. This procedure is done in engineering applications and a simple example of such a controller is obtained by linearizing the system around zero. This controller is called the *linear quadratic regulator* (LQR) and in many examples we use this controller as a benchmark. Using the LQR, encoded as a symmetric matrix, the parameter space is only $\frac{d(d-1)}{2}$ dimensional. The simplicity of this controller raises the question of proportionality - is it worth finding the value function in a high-dimensional or non-linear function space, when a simple quadratic function can perform reasonably well? In this thesis we encounter problems where the LQR controller fails and thus more sophisticated model spaces are required.

To tame the curse of dimensionality we use tensor compression techniques allowing for approximations of the value function of higher polynomial degree. More exactly, we use hierarchical tensor formats, in particular tensor-trains, to obtain a multi-linear parameterization of a low-dimensional manifold within the high-dimensional linear space. This allows us to reduce the number of parameters from the above example from $\mathbf{10^{22}}$ to $\approx \mathbf{5000}$, which is less than a megabyte, making computations on a laptop or a PC feasible.

Historically, the first low-rank tensor format was the *CP-format*, introduced in 1927 by Hitchcock [Hit27]. However, computational and theoretical issues limit the applicability of this format for our purpose. In 1966, Tucker [Tuc66] published another low-rank tensor format basing on subspaces. For the Tucker decomposition, it was later identified that a generalization of the *singular value decomposition (SVD)* [DDV00] can be used to compute such a low-rank representation. Due to its close relation to the SVD, favorable robustness and theoretical properties are present in this decomposition. However, this format does still have exponential increase of degrees of freedom w.r.t. the spatial dimension d. This problem was alleviated by the tensor-train model and hierarchical tensor formats. These models have been discovered in the mathematical community by Hackbusch [HK09] and Oseledets [Ose11]. However, in the physics community these objects are known since the **90**s under the name *matrix product states* (MPS). Steven White [Whi92] introduced the most famous algorithm for optimization on the set of MPS called *density matrix renormalization group* (DMRG), which is a variant of the algorithm used in this thesis as well, known in the mathematical community as *Alternating Linear Scheme* or *Alternating Least Squares* (ALS). Favorable mathematical properties such as a smooth manifold structure have lead to extensive research on these objects from a geometrical perspective [HRS12b; UV13] and new, mathematically more pleasing algorithms such as *Riemannian optimization* approaches have been developed [Ste16]. However, even with these improved algorithms, beating the performance of the ALS algorithm is difficult, which is the reason why in the survey [KB09] the ALS is deemed the "workhorse" algorithm. Nowadays, tensor compression techniques are used in various fields of science, ranging from quantum physics [Whi92; Sch11] and chemistry [KKS11; KKF11] over machine learning [Sid+17; Leb+14], to neuroscience [Miw+04; Con+15] and other fields.

In the context of high-dimensional data or functions, other approaches have seen great successes in the recent past. Most notably, *deep neural networks* are widely used to handle high-dimensional data. Groundbreaking successes in the field reinforcement learning [Mni+15]

and image recognition have lead to extensive usage of deep neural networks in different scientific fields ranging from solving various PDEs [Kut+19] to signal processing [YD10] and finding ground states of the Schrödinger equation [MST17]. All these successes combined with accessible software packages such as `TensorFlow` [Aba+16] and `PyTorch` [Pas+17] make deep neural networks a - and maybe even the - fastest growing research field in computer science. Other methods such as *support vector machines* or *sparse polynomials* are promising methods as well. In this thesis we have one numerical example where the performance of tensor-trains is directly compared to neural networks.

We perform numerical studies for different high-dimensional problems. These problems are mostly of Bellman or HJB type with the exception of certain parabolic PDEs that we use with a *backwards stochastic differential equation* (BSDE) approach. However, all these problems share a similar backwards or fixed-point iteration - the result from the previous step affects the next step. Driven by the successes of model-free deep reinforcement learning, a current trend is model-free controlling and computing. To this end, we identify the Bellman equation as a bridge between model-based controlling using the HJB equation and model-free controlling. However, we also construct algorithms based on the PMP where a lot of information about the system is encoded.

Outline

The thesis is structured in the following way.

Chapter 1 introduces concepts of (optimal) control theory. The difference between open-loop and closed-loop control and other basic notions is covered. The value function and governing Bellman and Hamilton-Jacobi-Bellman equations are introduced for a model problem. We consider the Policy Iteration and discuss convergence. We also cover Pontryagin's maximum principle and briefly cover methods for sub-optimal control such as Model Predictive Control and the Linear Quadratic Regulator.

Chapter 2 considers regression problems on a non-linear ansatz space. A theorem bounding the minimal number of samples needed for recovery of a function is presented.

Chapter 3 introduces our model set, the set of tensor-trains. We first define simple operations on tensor spaces and then cover different tensor networks. The set of tensor-trains, which is our main concern, is introduced and operations on the set are considered. We cover the representation of high-dimensional functions using tensor-trains and finding optimal coefficients using the regression approach. Important concepts such as regularization and rank-adaptive solvers are covered.

Chapter 4 is the main part of this thesis. In this chapter we combine the content of the previous chapters and consider several different numerical problems, ranging from finite horizon deterministic problems over stochastic exit-time control problems to general parabolic partial differential equations. For the infinite horizon case we identify long trajectories as a preconditioner for the linearized Bellman equation. For the finite horizon case we introduce an algorithm based on the PMP, which replaces the Policy Iteration approach. We also state a linear error propagation w.r.t. the number of discretization points within the time-interval. For the BSDE problem we compare the performance of our tensor-train model to neural network approaches. In the end of all these section, we give a brief conclusion and outlook for

possible future research. After this chapter, we finalize the thesis with a general conclusion.

Chapter 2

Introduction to Optimal Control

This chapter is devoted to introducing basic concepts of optimal feedback control. We first introduce a model problem, from where we consider Pontryagin's maximum principle, the value function and the corresponding Bellman - and Hamilton-Jacobi-Bellman (HJB) equation. These concepts do not only exist for our model problem, but also for optimal control problems with different structure. While the equations are not exactly the same, the basic definitions and ideas are similar. Note that the content of this chapter, at least for our model problem, is covered in several textbooks. In our presentation, we follow the description given in [BC97].

Optimal control is a field of mathematics, where a dynamical system, oftentimes described by an ordinary differential equation (ODE) is modified via a control, such that certain objectives, encoded in a cost functional, are fulfilled. There is a vast amount of possible cost functional designs and as an example, we consider a discounted infinite horizon deterministic optimal control problem of the following form.

Optimal Control Problem 1. Set $\gamma \geq 0$. Assume that $\ell : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$, $\ell \geq 0$, and $f : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$ are continuously differentiable. For $x \in \mathbb{R}^d$ minimize w.r.t. $u \in L^2(0, \infty; \mathbb{R}^m)$ the cost functional $\mathcal{J} : \mathbb{R}^d \times L^2(0, \infty; \mathbb{R}^m) \to \mathbb{R} \cup \infty$, defined as,

$$\mathcal{J}(x, u(\cdot)) := \int_0^\infty e^{-\gamma t} \ell(y(t), u(t)) \, \mathrm{d}t,$$

where

$$\begin{aligned} \partial_t y(t) &= f(y(t), u(t)) \\ y(0) &= x. \end{aligned} \tag{2.1}$$

If either the trajectory $y(t)$ does not exist for every $t \in [0, \infty)$ or the integral within the cost functional does not converge, we set $\mathcal{J}(x, u) = \infty$.

Remark 1. Note that in the literature, the arguments of the cost functional can differ. We consider a dependence on the initial state x and the control $u(\cdot)$ to stress the dependence of the cost functional on the initial state. In the following we sometimes further stress this dependence by denoting the trajectory $y(\cdot)$ starting in x by $y_x(\cdot)$. Another common notation of the cost functional, which stresses the dependence of the cost functional on the whole

trajectory, exchanges the initial state x by the whole trajectory $y(\cdot)$ in the argument of the cost functional. Our notation is mostly used in the feedback control community and the other notation is used in the open-loop control community.

We do not consider existence theory of the flow or of optimal controls. Instead, we focus on presenting the key concepts of (optimal) feedback control. Thus, we rely on the following assumptions throughout this chapter.

Assumption 2.0.1. The differential equation (2.1) is given in a way such that for every $x \in \mathbb{R}^d$ and $u \in L^2(0,\infty;\mathbb{R}^m)$ there exists $\tau_{x,u} > 0$ such that the differential equation has a unique solution y defined on $[0, \tau_{x,u})$.

Assumption 2.0.2. The cost functional $\mathcal{J}$ and the differential equation is given in a way such that for every initial state $x \in \mathbb{R}^d$ there exists an optimal control $u^* \in L^2(0,\infty;\mathbb{R}^m)$ with finite cost, which means that $\mathcal{J}(x,u^*) = \inf_{u \in L^2(0,\infty;\mathbb{R}^m)} \mathcal{J}(x,u) < \infty$.

Remark 2. Note that Assumption 2.0.1 does not guarantee the existence of the trajectory on $[0,\infty)$, but instead on a small time-frame. Thus, the existence of blow-ups, where the state steers to infinity, is possible. Assumption 2.0.2 guarantees the existence of an optimal control where the corresponding cost is finite.

2.1 Open-Loop and Closed-Loop Control

The theory of open-loop optimal control seeks conditions under which optimal controls exist and are unique. Moreover, numerical techniques for finding an optimal control for a given initial state x are derived. In contrast to that, closed-loop control is trying to compute and encode an optimal control for every initial value. In this way, more flexibility w.r.t. model errors and uncertainties can be achieved. More exactly, open-loop control seeks for an initial value x an optimal control $u^*(\cdot) : [0,\infty) \to \mathbb{R}^m$, while feedback-control seeks a mapping $\alpha^*(\cdot) : \mathbb{R}^d \to \mathbb{R}^m$, such that the optimal controls for every state is encoded in the mapping α. Note that from here on, we always denote open-loop controls by u and closed-loop controls or feedback laws by α.

The open loop system (2.1) takes the form

$$\partial_t y(t) = f(y(t), u(t))), \quad y(0) = x$$

and the corresponding closed-loop system is

$$\partial_t y(t) = f(y(t), \alpha(y(t)), \quad y(0) = x. \tag{2.2}$$

Obviously, the closed-loop system might not be well-posed for certain mappings $\alpha : \mathbb{R}^d \to \mathbb{R}^m$. Thus, we define admissible feedback controls in the following way.

Definition 3 ([BC97])**.** A feedback law α on a set Ω is *admissible* if for all $x \in \Omega$ there exists a unique solution to the closed-loop system (2.2), $t \mapsto \alpha(y_x^\alpha(t))$ is measurable on $[0,\infty)$ and $y_x^\alpha \in \Omega$ on $[0,\infty)$.

Denoting the trajectory of the closed-loop system starting in x by $y = y_x^\alpha$, we can reformulate the open-loop cost functional

$$\mathcal{J}(x, u(\cdot)) := \int_0^\infty e^{-\gamma t} \ell(y(t), u(t)) \, \mathrm{d}t$$

for feedback laws as

$$v^\alpha(x) := \int_0^\infty e^{-\gamma t} \ell(y(t), \alpha(y(t))) \, \mathrm{d}t. \tag{2.3}$$

As before, if the trajectory does not exist on $[0, \infty)$, we set $v^\alpha(x) = \infty$. Using v^α we can give notion to optimal feedback controls.

Definition 4 ([BC97])**.** An admissible feedback law $\alpha^* : \mathbb{R}^d \to \mathbb{R}^m$ is *optimal* if

$$v^{\alpha^*}(x) = \min_{u \in L^2(0,\infty;\mathbb{R}^m)} \mathcal{J}(x, u)$$

for all $x \in \mathbb{R}^d$.

Formally, if a feedback law $\alpha : \mathbb{R}^d \to \mathbb{R}^m$ is available, we can obtain for every initial state $x \in \mathbb{R}^d$ an open loop control $u_x^\alpha : [0, \infty) \to \mathbb{R}^m$ by setting $u_x^\alpha(t) = \alpha(y_x^\alpha(t))$. Thus, if an optimal feedback law is available, an optimal open-loop control can be synthesised implicitly.

We define the notion of exponential stability.

Definition 5 ([Son90])**.** Let $\alpha : \mathbb{R}^m \to \mathbb{R}^d$ be an admissible feedback law on $\Omega \subset \mathbb{R}^d$ and consider the feedback-controlled system (2.2). We say that α stabilizes (2.2) exponentially (around 0), if for every $x \in \Omega$ we have

$$|y_x^\alpha(t)| \leq M e^{-\alpha t} |x|.$$

For an admissible feedback law we define the flow of a system.

Definition 6 (Flow)**.** Consider an ODE of the form

$$\partial_t y(t) = g(y(t)), \tag{2.4}$$

where the solution for every initial value $x \in \mathbb{R}^d$ exists on the interval $[0, \tau]$. Then, we define the flow

$$\mathbf{\Phi}_\tau : \mathbb{R}^d \to \mathbb{R}^d, x \mapsto y_x(\tau),$$

where $y_x(\tau)$ is the trajectory of (2.4) w.r.t. the initial value $y(0) = x$.

For an exponentially stable (feedback-) system, for every open set $\tilde{\Omega}$ containing 0, we can find a $\tau > 0$, such that the flow $\mathbf{\Phi}_\tau$ maps $\tilde{\Omega}$ to $\tilde{\Omega}$. If the flow w.r.t. the closed-loop system (2.2) exists, we also denote it by $\mathbf{\Phi}_\tau^\alpha$.

Before we consider an ansatz for solving a closed-loop optimal control problem, we first give an overview of open-loop optimal control theory, namely adjoint methods and Pontryagin's maximum principle.

2.2 Adjoint Methods and Pontryagin's Maximum Principle

In this section, we replace the infinite horizon control problem by a finite horizon problem. In a finite horizon control problem the ∞ in the cost functional is replaced by a finite time T. Further, we allow time dependencies within the ODE and the cost functional.

Optimal Control Problem 2. Assume that $\ell : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$, $\ell \geq 0$, $f : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$, and $c_T : \mathbb{R}^d \to \mathbb{R}_+$ are continuously differentiable. For $x \in \mathbb{R}^d$ minimize w.r.t. $u \in L^2(0, T; \mathbb{R}^m)$ the cost functional $\mathcal{J} : \mathbb{R}^d \times L^2(0, T; \mathbb{R}^m) \to \mathbb{R} \cup \infty$, defined as,

$$\mathcal{J}(x, u(\cdot)) := \int_0^T \ell(t, y(t), u(t)) \, \mathrm{d}t + c_T(y(T)), \tag{2.5}$$

where

$$\begin{aligned} \partial_t y(t) &= f(t, y(t), u(t)) \\ y(0) &= x. \end{aligned}$$

If either the trajectory $y(t)$ does not exist for every $t \in [0, T]$ or the integral within the cost functional does not converge, we set $\mathcal{J}(x, u) = \infty$.

Note that while there exist approaches for the PMP for infinite horizon problems, see e.g. [AK07] and references therein, we only cover the finite horizon case. This is done because the PMP for finite horizon problems can be used to obtain a gradient descent scheme, allowing for numerical approximation of optimal controls. Note that these controls are of open-loop type. However, we will also use this gradient scheme in Section 5.2 to obtain a method for computing optimal feedback laws for finite horizon problems. We also state that an optimal control for an infinite horizon problem can be approximated by simply setting the final time T large enough such that the remainder of the running cost can be considered negligible.

If for every u the ODE has a unique solution, it is possible to reduce the problem to an unconstrained optimization problem on $L^2(0, T; \mathbb{R}^m)$. This allows for usage of classical optimization techniques. Pontryagin and other researchers derived necessary conditions for such optimal controls. The Pontryagin Maximum Principle (PMP) is a necessary condition for a minimal control. It is given by the following Theorem.

Theorem 2.2.1 ([Pon+62; Don+95])**.** *Let y^*, u^* be an optimal state-control pair for Optimal Control Problem 2. Then, there exists a costate $p^*(\cdot)$ satisfying the following ODE*

$$\partial_t p^*(t) = -\partial_y H(t, y^*, u^*(t), p^*(t)), \quad p^*(T) = \partial_y c_T(y^*(T)),$$

where $H(t, y, u, p) = \ell(t, y, u) + p^T f(t, y, u)$ and $u^*(t) \in \operatorname{arg\,min}_u H(t, y^*(t), u, p^*(t))$.

After introducing a time discretization, this system is solved by standard gradient decent methods. This is done by identifying the PMP as a representation of the gradient of the cost functional w.r.t. the control, see e.g. [HK10; Trö10] for more algorithms. The optimization is done by computing the forward and then the backward equation, using these to compute the gradient of the cost functional and then updating the control. We summarize this in

Algorithm 1.

Algorithm 1: Simple gradient descent

input : An initial control u_0, defined on $[0, T]$.
output : An optimal control, defined on $[0, T]$.
Set $k = 0$
do
  Compute y_k by solving
  $$\partial_t y_k = f(t, y_k, u_k).$$
  Compute p_k by solving
  $$\partial_t p_k = \partial_y H(t, y_k, u_k, p_k).$$
  Compute $d_k = D_u \mathcal{J}(t, y_k, u_k) = \partial_u H(t, y_k, u_k, p_k)$ and update the control with a step-size $\tau_k > 0$
  $$u_{k+1} = u_k + \tau_k d_k$$
  and set $k := k + 1$.
while $\|d_k\| > tol$;

2.3 The Value Function and Implications for Closed-Loop Control

After finding a way to compute optimal open-loop controls, we now turn to the problem of feedback control. We do this by introducing the value function. For an optimal control problem it is defined in the following way.

Definition 7. Consider Optimal Control Problem 1. Then, the *value function* is defined as

$$v^*(x) = \inf_u \mathcal{J}(x, u).$$

Under certain assumption to be specified later, the value function yields a handy way to encode optimal controls for every initial value. In fact, in these cases the optimal control is proportional to the gradient of the value function, cf. Theorem 2.5.4 and Remark 11. At this point we make a short consideration w.r.t. the dimensional complexity that is needed for encoding a feedback law and the value function.

Remark 8. Consider a d dimensional state space and an m dimensional control space. In order to encode the feedback law directly, we have to find the optimal feedback law $\alpha^* : \mathbb{R}^d \to \mathbb{R}^m$. In contrast to that the value function is a mapping $v^* : \mathbb{R}^d \to \mathbb{R}$, which means that the output dimension is 1. Thus, for high-dimensional controls, the structure of the value function appears to be easier to encode. However, its main advantage is the existence of governing equations that the value function obeys, which gives rise to possible numerical treatment.

In fact, in the context of Machine Learning, in particular Reinforcement Learning, there exist many different algorithms for encoding an optimal controller directly. These approaches are covered in [SB18] and include TD-learning, Q-learning and actor-critic models. In these approaches functions similar to the value functions have to be learned. In particular within

the actor-critic model, the value function can be seen as the critic and the gradient of the value function is proportional to the actor, see also [ZHL21] for a Least-Squares method closely related to our method in the context of Reinforcement Learning.

Up until now it is unclear how to obtain the value function or the optimal feedback law. Note that by computing an optimal control and the corresponding cost, the value function can be computed point-wise, which might be a starting point for numerical treatment and is done e.g. in [AKK21]. However, there are fundamental equations like the Bellman equation or the Hamilton-Jacobi-Bellman equation which the value function obeys. We now elaborate on these fundamental equations.

2.4 The Bellman Equation

The Bellman equation or Bellman's Principle of Optimality or Dynamic Programming Principle is a fundamental principle which is not only used in optimal control theory. Other fields of application are economics [LS12] or graph theory [Bel58]. In words of Richard Bellman, who is generally considered to be the inventor of Dynamic Programming, it is summarized as

"Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." [Bel57]

For the infinite horizon optimal control problem, it reads as follows.

Proposition 2.4.1 (Bellman equation, [BC97])**.** *For all $x \in \mathbb{R}^d$ and $\tau \geq 0$ we have*

$$v^*(x) = \inf_{u \in L^2(0,\tau;\mathbb{R}^m)} \big[\int_0^\tau e^{-\gamma t} \ell(y_x(t), u(t)) \, \mathrm{d}t + e^{-\gamma \tau} v^*(y_x(\tau)) \big].$$

Remark 9. Assume that there exists an optimal control u^* for every initial value $x \in \mathbb{R}^d$. Then, we have

$$v^*(x) = \int_0^\tau e^{-\gamma t} \ell(y_x(t), u^*(t)) \, \mathrm{d}t + e^{-\gamma \tau} v^*(y_x(\tau)),$$

If we assume that an optimal feedback law α^* exists, we can write down a similar Bellman principle, using feedback laws.

$$v^{\alpha^*}(x) = \int_0^\tau e^{-\gamma t} \ell(y_x(t), \alpha^*(y_x(t))) \, \mathrm{d}t + e^{-\gamma \tau} v^{\alpha^*}(y_x(\tau)), \tag{2.6}$$

In addition to the Bellman equality for the optimal feedback law (2.6), a similar equality holds for any feedback law with finite cost. The proof is a simpler version of the Bellman equation.

Theorem 2.4.2. *Let α be an admissible feedback law with finite cost. Then, for all $x \in \mathbb{R}^d$ and all $0 \leq \tau < \infty$, we have*

$$v^\alpha(x) = \int_0^\tau e^{-\gamma t} \ell(y_x(t), \alpha(y_x(t))) \, \mathrm{d}t + e^{-\gamma \tau} v^\alpha(y_x(\tau)), \tag{2.7}$$

Proof. We have

$$\begin{aligned} v^\alpha(x) &= \int_0^\infty e^{-\gamma t}\ell(y_x(t),\alpha(y_x(t)))\,\mathrm{d}t \\ &= \int_0^\tau e^{-\gamma t}\ell(y_x(t),\alpha(y_x(t)))\,\mathrm{d}t + \int_\tau^\infty e^{-\gamma t}\ell(y_x(t),\alpha(y_x(t)))\,\mathrm{d}t \\ &= \int_0^\tau e^{-\gamma t}\ell(y_x(t),\alpha(y_x(t)))\,\mathrm{d}t + \int_0^\infty e^{-\gamma(t+\tau)}\ell(y_x(t+\tau),\alpha(y_x(t+\tau)))\,\mathrm{d}t \\ &= \int_0^\tau e^{-\gamma t}\ell(y_x(t),\alpha(y_x(t)))\,\mathrm{d}t + e^{-\gamma\tau}v^\alpha(y_x(\tau)). \end{aligned}$$

□

In order to understand the significance of the Bellman equation, we recapitulate that the value function can be evaluated (or approximated) point-wise. This is done by solving the optimal control problem for a single initial value x using a numerical method of open-loop optimal control theory, as described in Section 2.2. In order to obtain this value one has to compute several long trajectories with the corresponding controls. Within the Bellman equation, only short trajectories of length $\tau > 0$ have to be computed. However, a difficult function equation is remaining and it is on first sight unclear how to solve it. This topic will be a main concern in the remainder of this thesis. In particular, it is covered in Section 2.6.

2.5 The Hamilton-Jacobi-Bellman Equation

We next consider an infinitesimal version of the Bellman equation, the Hamilton-Jacobi-Bellman (HJB) equation. In contrary to the Bellman equation, where trajectories of length τ have to be computed, the HJB equation only uses infinitesimal information of the cost functional and the dynamics. This means, that only the integrand of the cost functional and the r.h.s. of the ODE has to be evaluated. The HJB equation has been broadly studied within the last 40 years and a central aspect to the theory of the existence theory of solutions. In this context, the notion of viscosity solutions was introduced in [CL83]. Viscosity solutions are a generalization of classical solutions to PDEs. While the exact definition is not necessary to understand the rest of this thesis, we still give it for the sake of completeness. In this context, we consider a continuous Hamiltonian defined on an open set $\Omega \subset \mathbb{R}^d$

$$F : \Omega \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$$

and a corresponding Hamilton-Jacobi equation

$$F(x, v(x), Dv(x)) = 0, \quad x \in \Omega. \tag{2.8}$$

We will now define viscosity solutions of such a differential equation. These solutions are not necessarily differentiable but only continuous. Thus, the notation of the differential operator D above is formal.

Definition 10. A function $v \in C(\Omega)$ is a *viscosity subsolution* of (2.8) if for any $\varphi \in C^1(\Omega)$

$$F(x_0, v(x_0), D\varphi(x_0)) \leq 0$$

holds for any local maximum point $x_0 \in \Omega$ of $v - \varphi$. Similarly, $v \in C(\Omega)$ is a *viscosity supersolution* of (2.8) if

$$F(x_0, v(x_0), D\varphi(x_0)) \geq 0$$

holds for any local minimum point $x_0 \in \Omega$ of $v - \varphi$.

Finally, v is a *viscosity solution* of (2.8) if it is both a viscosity sub- and supersolution.

We stress that by our definition a viscosity solution is at least continuous. There is also a notion of discontinuous viscosity solutions which will not be discussed here. The interested reader is referred to [BC97, Chapter 5]. We are particularly interested in regular value functions. The following proposition ensures that every classical solution is a viscosity solution as well.

Proposition 2.5.1 ([BC97])**.**

(I) If $v \in C(\Omega)$ is a viscosity solution of (2.8) *in Ω, then v is a viscosity solution of* (2.8) *on any open set $\tilde{\Omega} \subset \Omega$.*

(II) If $v \in C(\Omega)$ is a classical solution of (2.8)*, then v is a viscosity solution of* (2.8)*.*

(III) If $v \in C^1(\Omega)$ is a viscosity solution of (2.8)*, then v is a classical solution of* (2.8)*.*

Viscosity solutions allow us to prove that the value function is a viscosity solution to a PDE, namely the Hamilton-Jacobi-Bellman equation.

Theorem 2.5.2 ([BC97; Da 00])**.** *Assume that there exists $L > 0$ such that*

$$(f(x,u) - f(y,u), x - y) \leq L|x - y|^2$$

for all $x, y \in \mathbb{R}^d$ and $u \in \mathbb{R}^m$ and that v^* is continuous. Then, the value function $v^*(x)$ is a viscosity solution of

$$\gamma v^*(x) + \sup_{u \in \mathbb{R}^m} \Big[- Dv^*(x) \cdot f(x,u) - \ell(x,u) \Big] = 0 \tag{2.9}$$

with $v^*(0) = 0$.

We now give an example which is a prototype of a control problem where the solution is not differentiable, but only continuous. In this sense, we consider a HJB equation and then identify the solution as the value function of an optimal exit-time control problem.

Example 2.5.3 (Eikonal equation)**.** Consider the one dimensional HJB equation defined on $\Omega = (-1, 1)$

$$\sup_{u \in \mathbb{R}} [-uDv - 1 - \frac{1}{2}|u|^2] = 0, \quad v(-1) = v(1) = 0. \tag{2.10}$$

Then, the maximizer can be computed explicitly, i.e. $u = -Dv$. Thus, (2.10) can be rewritten as

$$|Dv|^2 - 1 - \frac{1}{2}|Dv|^2 = 0$$

which means that

$$|Dv|^2 - 2 = 0$$

This equation does not have any classical solution, but it has infinitely many weak solutions. Namely, every piece wise linear function with $v(-1) = v(1) = 0$ and $|Dv| = \sqrt{2}$ almost everywhere (a.e.).

We will now first prove that a viscosity solution is

$$v^*(x) = \sqrt{2}(1 - |x|).$$

Then we prove that no other function of the type $v(-1) = v(1) = 0$ and $|Dv| = \sqrt{2}$ a.e. is a viscosity solution.

Step 1: $\mathbf{v^*(x) = \sqrt{2}(1 - |x|)}$ is a viscosity solution. We first show that v^* is a viscosity subsolution. Let $\varphi \in C^1(\mathbb{R})$ and let $0 \neq x \in (-1, 1)$ be a local maximum point of $v^* - \varphi$. Then, $v^* - \varphi$ is differentiable at x and its derivative is 0. Thus, $\varphi'(x) = \sqrt{2}$ and $|\varphi'(x)|^2 - 2 = 0$. Now let $x = 0$ be a local maximum of $v^* - \varphi$. As the partial derivatives of v^* at $x = 0$ exist, it follows that $-\sqrt{2} \leq \varphi'(0) \leq \sqrt{2}$ and thus

$$|\varphi'(0)|^2 - 2 \leq 0,$$

which proves that v^* is a viscosity subsolution.

Proving that v^* is a viscosity supersolution is analogous, for the case $x \neq 0$. For $x = 0$ we notice that in order to be a local minimum of $v^* - \varphi$, again due to the partial derivatives of v^* at 0, it follows that $\sqrt{2} \leq \varphi'(0) \leq -\sqrt{2}$, which is a contradiction. Thus, $x = 0$ cannot be a local minimum of $v^* - \varphi$, which proves that v^* is a viscosity supersolution. This finishes the proof that v^* is a viscosity solution of (2.10).

Uniqueness requires a so-called comparison theorem, which needs more in-depth arguments. We refer the reader to [BC97, Chapter 2, Theorem 5.9] and the following remark. However, it is easy to verify that no other function of the type $v(-1) = v(1) = 0$ and $|Dv| = \sqrt{2}$ a.e. is a viscosity solution.

Step 2: No other function of the above type is a viscosity solution. Indeed, for any such function $\tilde{v}$, there must exist a neighborhood U of $y \in (-1, 1)$, such that locally $\tilde{v}$ can be written as $\tilde{v}(x) = a + \sqrt{2}(|x - y|)$ with $a \in \mathbb{R}$ and $x \in U$. We now verify that v is not a viscosity supersolution. Let $\varphi \equiv 0 \in C^1(\mathbb{R})$. Then x is a local minimum point of $\tilde{v} - \varphi$. However, $|\varphi'|^2 - 2 = -2 < 0$, which proves that $\tilde{v}$ is not a viscosity supersolution.

Setting $\ell(x, u) = 1 + \frac{1}{2}|u|^2$ and $f(x, u) = u$, we identify the Eikonal equation as a HJB corresponding to an optimal control problem with dynamical system f and running cost ℓ. The structure of the cost functional however, is not the infinite horizon control problem. Instead we identify it as an optimal exit-time problem with cost functional

$$\mathcal{J}(x, u) = \int_0^\theta 1 + \frac{1}{2}|u(t)|^2 \, \mathrm{d}t,$$

where $\theta = \inf\{t > 0 | y(t) > 1 \text{ or } y(t) < -1\}$, i.e. the integration is stopped when a certain exit set, in this case $E = \{x \in \mathbb{R} : |x| \geq 1\}$, is reached. In this case the value function is not differentiable on $[-1, 1]$. However, one can prove that for convex exit sets the value function takes the form

$$v^*(x) = d(x, E)$$

and is indeed differentiable on the complement $\Omega \setminus \bar{E}$. This observation also holds for $d > 1$-dimensional problems, if the exit set has a smooth boundary and is convex [BC97, Remark 2.15].

In the following theorem, it is shown that the optimal control can be characterized as the maximum of the Hamiltonian.

Theorem 2.5.4 ([BC97])**.** *Assume that the value function v^* is differentiable. Then, an optimal feedback law is given by*

$$\alpha^*(x) \in \operatorname*{arg\,max}_{u \in \mathbb{R}^m} \{-f(x,u) \cdot \nabla v^*(x) - \ell(x,u)\}.$$

Remark 11. Using further restrictions for the state dynamics and for the cost functional, we obtain an explicit form of the optimal feedback control. More exactly, assume that the state dynamics are of the form

$$f(x,u) = g(x) + h(x)u$$

with $h : \mathbb{R}^d \to \mathbb{R}^{d,m}$ and the running cost is of the form

$$\ell(x,u) = c(x) + u'Ru,$$

where R is positive definite. Then, the maximization in (2.6) can be computed explicitly and is given in the form

$$\alpha^*(x) = -\frac{1}{2} R^{-1} h(x)' \nabla v^*(x).$$

under the assumption that $\nabla v^*(x)$ exists for all x.

As in the Bellman case, there also exists a linearized HJB equation for a fixed feedback law.

Theorem 2.5.5. *Let α be given such that $v^\alpha(x) < \infty$ for all $x \in \mathbb{R}^d$ and v^α continuous. Then, it holds that v^α is a viscosity solution of*

$$\gamma v^\alpha + H^\alpha(x, Dv^\alpha) = 0,$$

where $H^\alpha(x,p) := -p \cdot (f(x,\alpha(x)) - \ell(x,\alpha(x))$.

Proof. Again, the proof is a simplification of the proof for the value function being a viscosity solution of (2.9).

Let $\varphi \in C^1(\mathbb{R}^d)$ and x be a local maximum point of $v^\alpha - \varphi$. Then, we have that there exists $s > 0$ such that $\varphi(x) - \varphi(z) \le v^\alpha(x) - v^\alpha(z)$ for all $z \in B_s(x)$. Thus, as the trajectory $y_x^\alpha(t) = y_x(t)$ is continuous, there also exists $\tau^* > 0$ such that for all $0 \le \tau \le \tau^*$ it holds

$$\varphi(x) - \varphi(y_x(\tau)) \le v^\alpha(x) - v^\alpha(y_x(\tau)).$$

Using Theorem 2.4.2 we obtain

$$\varphi(x) - \varphi(y_x(\tau)) \le \int_0^\tau e^{-\gamma t} \ell(y_x(t), \alpha(y_x(t)))\,\mathrm{d}t + (e^{-\gamma\tau} - 1) v^\alpha(y_x(\tau)).$$

By the differentiability of φ, and the continuity of v^α, y_x, and ℓ, we obtain, dividing by τ and letting $\tau \to 0$,

$$-D\varphi(x) \cdot y_x'(0) = -D\varphi(x) \cdot f(x,\alpha(x)) \le \ell(x,\alpha(x)) - \gamma v^\alpha(x),$$

which proves that $\gamma v^\alpha + H^\alpha(x, D\varphi) \le 0$, that is v^α is a viscosity subsolution.

Proving that v^α is a viscosity supersolution is analogous to the previous case. □

2.6 Approximating the Value Function by Policy Iteration

Policy Iteration is an algorithm that can be used to improve an initial sub-optimal policy in an iterative process. It was originally introduced by Richard Bellman [Bel57] and Ronald Howard [How60], for discrete time and space Markovian processes. This algorithm can be applied in a very general context and we use this algorithm for every control problem that we consider. It basically consists of iterating two steps, namely computing v^α for a given policy α and by updating the policy.

Algorithm 2: Policy Iteration

input : A Policy α_0 such that $v^{\alpha_0}(x) < \infty$ for all $x \in \mathbb{R}^d$.
output: An approximation of v^* and α^*.
Set $k = 0$.
do
  Find

$$v_k = v^{\alpha_k}, \tag{2.11}$$

  defined in (2.3), then update the policy according to

$$\alpha_{k+1}(x) \in \operatorname*{arg\,max}_{u \in \mathbb{R}^m} H^u(x, Dv),$$

  where $H^u(x,p) := -p \cdot (f(x,u) - \ell(x,u)$,
  and set $k := k + 1$.
while $\|\alpha_k - \alpha_{k+1}\| > tol$;

Note that in our numerical examples we only consider problems where the minimizer in the policy update can be computed as in Remark 11. Thus, for our application, finding v^α is the computational bottleneck and will be studied in the following chapters. We now proceed by giving a simple proof for convergence, given exact arithmetic.

Theorem 2.6.1 (Convergence of Policy Iteration)**.** *Let α_0 be an admissible policy on $\mathbb{R}^d$ such that $v^{\alpha_0} < \infty$. Further, assume that for all k the Policy Iteration produces feedback controls such that the corresponding flow exists and that $v_k \in C^1(\mathbb{R}^d)$. Then, the Policy Iteration produces a point-wise monotonically decreasing sequence v_k. Consequently, the Policy Iteration/Algorithm 2 converges point-wise.*

There are different proofs for the convergence of the Policy Iteration, see [PB79; SG77]. We adapt the proof from [SG77] to the infinite horizon problem. [1]

Proof. Let α_0 be an admissible policy on $\mathbb{R}^d$ such that $v^{\alpha_0} < \infty$. Further, write $v_k = v^{\alpha_k}$ and $H_k(x,p) = -p \cdot f(x, \alpha_k(x)) - \ell(x, \alpha_k(x))$, which means that $H_k = H^{\alpha_k}$.

By assumption we have $0 \leq v^{\alpha_0} < \infty$ and by Theorem 2.5.5 we have

$$\gamma v_0 + H_0(x, Dv_0) = 0.$$

Also, because of the policy update criterion, we have

$$\gamma v_0 + H_1(x, Dv_0) \geq \gamma v_0 + H_0(x, Dv_0) = 0,$$

[1] The adaption of the proof was done in conjunction with Mathias Oster.

which means that there exists a non-negative function $r_0 : \mathbb{R} \to \mathbb{R}$ such that

$$\gamma v_0(x) + H_1(x, Dv_0(x)) = r_0(x)$$

Plugging in the definition of H_1 and reordering yields

$$\gamma v_0(x) - Dv_0 \cdot f(x, \alpha_1(x)) = r_0(x) + \ell(x, \alpha_1(x))$$

In this formulation, we see that the PDE can be reformulated using the characteristic curves of the underlying controlled ODE system with an adapted running-cost

$$\begin{aligned} \partial_t y(t) &= -f(y(t), \alpha_1(y(t))) \\ \partial_t v_0(y(t)) &= -\gamma v_0 + r_0(y(t)) + \ell(y(t)) \\ y(0) &= x. \end{aligned}$$

Integrating along the trajectories yields

$$\begin{aligned} v_0(x) - \underbrace{e^{-\gamma t} v_0(y(t))}_{\geq 0} &= \int_0^t e^{-\gamma s}((r_0(y(s)) + \ell(y(s), \alpha_1(y(s)))\,\mathrm{d}s \\ &\geq \int_0^t e^{-\gamma s} \ell(y(s), \alpha_1(y(s)))\,\mathrm{d}s \end{aligned}$$

for all $t > 0$, which proves that $v_0(x) \geq \int_0^t e^{-\gamma s} \ell(y(s), \alpha_1(y(s)))\,\mathrm{d}s$ for any $t > 0$. Letting $t \to \infty$ yields $v_0(x) \geq v_1(x)$.

□

Remark 12. This theorem implies that if the flow remains well-defined within the Policy Iteration, convergence can be shown. However, the existence of the flow after the policy update is still work in progress. This problem can be circumvented by considering the linearized HJB equation to compute v^α. In this case, only the r.h.s. of the ODE and the integrand of the cost functional have to be evaluated, which means that this formulation does not need a flow and integrable cost.

Moreover, the assumption that v_k is $C^1(\Omega)$ is only needed for the existence of the characteristic curves. If the existence can be guaranteed in another way, this assumption can be dropped.

Note that v^α can be computed by numerical integration along the trajectories. Moreover, using the linearized Bellman equation as in Theorem 2.4.2 the length of the trajectories can be reduced arbitrarily. An infinitesimal version is the linearized HJB equation, as in Theorem 2.5.5. Finally, in [PB79], the Policy Iteration is characterized as a Newton-Kantorovich iteration [Kan48]. In particular, under additional assumptions, a quadratic convergence of Newton's method is established.

2.7 Approximative Policy Iteration

In the application, computing Algorithm 2 is numerically infeasible. In particular, (2.11), i.e. for given policy α, finding $v^\alpha(x)$ is not possible for every x. While evaluating v^α at points

x is possible by numerically integrating along the trajectory, we cannot expect to find a closed-form solution v^α. Similarly, using the linearized Bellman equation, cf. Theorem 2.4.2, or the linearized HJB equation, cf. Theorem 2.5.5, no closed form solution can be found. Thus, in order to obtain a computable algorithm, we have to replace the Policy Iteration by an approximative Policy Iteration, where (2.11) is replaced by a computable equation. We introduce an abstract loss functional, defined on a compact set $\Omega \subset \mathbb{R}^d$

$$\mathcal{R}^\alpha : L^2(\Omega) \to \mathbb{R}_+, \quad v \mapsto \mathcal{R}^\alpha(v),$$

where we assume that $\mathcal{R}^\alpha(v^\alpha) = 0$, i.e. v^α is a minimizer of $\mathcal{R}^\alpha$. Then, we replace (2.11) by a minimization of $\mathcal{R}^\alpha$ over a computable ansatz set $\mathcal{M} \subset L^2(\Omega)$ and obtain the approximative Policy Iteration, which is stated in Algorithm 3.

Algorithm 3: approximative Policy Iteration

input : A policy α_0 such that $v^{\alpha_0}(x) < \infty$ for all $x \in \Omega$.
output : An approximation of v^* and α^*.
Set $k = 0$.
do
Find

$$v_k = \operatorname*{arg\,min}_{v \in \mathcal{M}} \mathcal{R}^{\alpha_k}(v), \tag{2.12}$$

then update the policy according to

$$\alpha_{k+1}(x) \in \operatorname*{arg\,max}_{u \in \mathbb{R}^m} H^u(x, Dv_k)$$

and set $k := k + 1$.
while $\|\alpha_k - \alpha_{k+1}\| > tol$;

Note that in general, the policy update is not computable as well. In this case we have to approximate this maximization process. However, in the special case as in Theorem 2.5.4 an explicit solution is known. As we only consider this special case within this thesis we do not cover approximation ideas here.

Remark 13. Replacing the exact solution to (2.11) by the approximation (2.12) breaks the arguments of the convergence proof in Theorem 2.6.1. Thus, we cannot guarantee convergence for the approximative Policy Iteration. Note that within the policy update it is possible to replace the maximization by an approximative maximization, as long as one can ensure that $H^{\alpha_{k+1}}(x, Dv_k)$ is non-negative for all x.

Designing explicit forms of the loss functional $\mathcal{R}^\alpha$ will be done in Chapter 3 and in Chapter 5.

2.8 Digression: Model Predictive Control and the Importance of Fast Evaluation of the Feedback Law

In this section, we consider an approach to finding sub-optimal feedback laws, namely *Model Predictive Control* (MPC). We later use MPC as a motivation for one algorithm that we

introduce in Section 5.2. Note that MPC is an example of a sub-optimal controller and that is used in various (industrial) applications, see e.g. [QB03].

MPC control originates for simple applications in the 1970s for stabilizing linear systems, see e.g. [Kle70], and later reformulated for non-linear systems, see e.g. [GP17]. For a survey on MPC w.r.t. theoretical results we refer to [GPM89]. In the MPC approach an infinite or finite horizon control problem is considered. We introduce the MPC approach for the infinite horizon case - but only on an intuitive level. This means that we do not state exact theorems but only explain the ideas. For a rigorous introduction we refer to [GP17].

Consider Optimal Control Problem 1 and in particular the cost functional

$$\mathcal{J}(x, u(\cdot)) := \int_0^\infty e^{-\gamma t}\ell(y(t), u(t))\,\mathrm{d}t.$$

Intuitively, it is easy to see that this functional can be approximated by replacing the ∞ in the integral by a large final time $T \gg 0$, i.e.

$$\mathcal{J}_T(x, u(\cdot)) := \int_0^T e^{-\gamma t}\ell(y(t), u(t))\,\mathrm{d}t. \tag{2.13}$$

Similarly, the control that is optimal for $\mathcal{J}_T$ will approach the optimal control for $\mathcal{J}$ for $T \to \infty$. Noticing that the PMP methods from Section 2.2 can be used to find an optimal control for $\mathcal{J}_T$, a feedback law for the infinite horizon problem can be defined as

$$\alpha(x) = u_T^*(0),$$

where $u_T^* : [0, T] \to \mathbb{R}^m$ is the optimal control for the cost functional $\mathcal{J}_T$ with initial condition x. The definition of a feedback law in this sense does not only raise the question of feasibility, which we do not cover here, but also of computability. For applications the feedback law has to be computed in real time. It is clear that the complexity of the feedback law increases with longer time-horizons and that for real-time evaluation of this functional T has to be set as small as possible. For small T, however, the optimal control u_T^* does not approximate the optimal control of the infinite horizon control problem. The balance of computability and approximation quality has to be done in MPC applications and the frequency of industrial application shows that for a large set of problems a balance can be found. Note that the approach can be improved by adding a final condition to $\mathcal{J}_T$ as done in (2.5). However, designing such a final condition is a challenging task and it is unclear how to proceed. In Section 5.2 we observe that in the MPC approach the Bellman equation can be recovered by adding the final condition $v^*(y(T))$ to (2.13). This observation leads to the approach being viable even for small final time T.

2.9 Digression: The Linear Quadratic Case and its Advantages

The case of a linear ODE and a quadratic cost functional is one of the few cases where the structure of the value function is known. In fact it is a quadratic function. The HJB equation can then be rewritten as a Riccati (differential-) equation. In this case, even for high

dimensional problems, the value function can easily be computed. Because of the success of quadratic value functions, in many non-linear applications the value function is approximated by a quadratic function. In this section, we revise the main steps that have to be performed to obtain such a controller. Here, we are guided by [Son98].

2.9.1 The Riccati Equation

In this section, we first consider a linear quadratic infinite horizon optimal control problem without discount.

Optimal Control Problem 3. For $x \in \Omega \subset \mathbb{R}^d$ minimize w.r.t. $u \in L^2(0,\infty;\mathbb{R}^m)$ the cost functional $\mathcal{J} : \Omega \times L^2(0,\infty;\mathbb{R}^m) \to \mathbb{R} \cup \infty$, defined as,

$$\mathcal{J}(x,u) = \int_0^\infty x(t)^T Q x(t) + u(t)^T R u(t)\,\mathrm{d}t$$

where

$$\begin{aligned} \partial_t y(t) &= Ay(t) + Bu(t) \\ y(0) &= x \end{aligned}$$

for $Q \in \mathbb{R}^{n,n}$ positive semi-definite, $R \in \mathbb{R}^{m,m}$ positive definite, $A \in \mathbb{R}^{n,n}$ and $B \in \mathbb{R}^{n,m}$. If either the trajectory $y(t)$ does not exist for every $t \to \infty$ or the integral within the cost functional does not converge we set $\mathcal{J}(x,u) = \infty$.

We are aiming to find conditions where we can find an optimal feedback law on $\Omega = \mathbb{R}^d$. A central assumption is the controllability of the pair (A,B), which is defined as follows.

Definition 14 ([Son98])**.** The pair (A,B) is *controllable* if the matrix

$$R(A,B) := [B, AB, A^2B, \ldots, A^{n-1}B] \in \mathbb{R}^{n,nm}$$

has full rank.

Note that the controllability of non-linear systems is more in-depth. Here, one has to resort to local controllability and it is possible to introduce a linear test to verify this condition. For further information we refer the reader to [Cor07]. In the linear case controllability ensures that there exists an optimal control for every initial value.

Theorem 2.9.1 ([Son98])**.** *Assume that the pair (A,B) is controllable. Then, there exists an optimal feedback law solving Optimal Control Problem 3. The value function is quadratic, i.e. $v^*(x) = x^T \Pi x$ with $\Pi \in \mathbb{R}^{n,n}$ symmetric and Π solves the algebraic Riccati equation*

$$\Pi A + A^T \Pi - \Pi B R^{-1} B^T \Pi + Q = 0.$$

The unique optimal feedback law is then given by

$$\alpha^*(x) = -R^{-1} B^T \Pi x.$$

From Theorem 2.9.1 we deduce that in linear quadratic control theory the Riccati equation plays a central role. In fact, the analytical problem of the HJB equation can be reduced to an algebraic matrix equation. Therein, efficient algorithms for solving the Riccati equation exist and are widely used [Lau79; Doo81]. Several solvers for large scale Riccati equations exist and are surveyed in [Ben+20]. This makes the Riccati equation the state-of-the-art tool for solving linear quadratic control problems.

2.9.2 The Linear Quadratic Regulator

The success of the Riccati equation has lead to applying it for non-linear problems as well. The resulting controller is called *Linear Quadratic Regulator* (LQR). This approach has lead to success especially when the cost functional is quadratic and the dynamics are close to linear. We will investigate the success of the LQR numerically in Chapter 5 and see cases where the control is close to optimal, but also cases where the controller fails. An intuitive reason for the success in this case is that the value function is still close to quadratic. We will now briefly present the approach to finding the LQR. For this we assume that an infinite horizon optimal control problem is given. The cost functional shall be quadratic and the dynamics shall be given as a sum of a linear part $L \in \mathbb{R}^{n,n}$ and a non-linear part $N(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$. Furthermore, the ODE shall be linear in the control, i.e.

$$\begin{aligned} \mathcal{J}(x,u) &= \int_0^\infty y(t)^T Q y(t) + u(t)^T R u(t)\,\mathrm{d}t \\ \partial_t y &= Lx + N(x) + Bu \\ y(0) &= x. \end{aligned}$$

The LQR is obtained in two steps. First we compute the linearization of the ODE around the origin, i.e.

$$\partial_t y = Lx + [DN(0)]x + Bu.$$

This is of course only valid if N is smooth around the origin. In the second step the corresponding Riccati equation is solved. Note that this is the most simple approach to finding a linear fit to the non-linear system. More involved strategies such as a power series approach, see e.g. [Gar72], are available and might perform even better, see e.g. [KK18] for a numerical study. Of course, the success of this strategy is limited by the relation of the linear part L and the non-linear part N. One can safely assume that this strategy performs well if L dominates N.

Chapter 3

Finding High-Dimensional Functions using Regression

We now elaborate on how we approximately solve functional equations defined on high-dimensional state spaces. Here, we take inspiration from statistical learning, in particular empirical risk minimization. The method applied here is described in [EST20; Eig+19]. We also refer to the recent application to tensor networks in [Tru21]. To this end, we formulate the problem in the sense of a regression problem and later specialize it to our needs. We consider a compact Lipschitz domain $\Omega \subset \mathbb{R}^d$ and a finite dimensional, not necessarily linear, model set $\mathcal{M} \subset L^2(\Omega)$ feasible for numerical optimization. During this process, we have a high-dimensional regression problem in mind, i.e. for $r \in L^2(\Omega)$, with the additional assumption that r can be evaluated point-wise, we want to find

$$v^{\mathcal{M}} = \operatorname*{arg\,min}_{v \in \mathcal{M}} \|v - r\|^2_{L_2(\Omega)}. \tag{3.1}$$

As the domain is high-dimensional, exact integration using quadrature rules suffers from the curse of dimensionality. Thus, we use Monte Carlo quadrature instead, which means that we draw $N \in \mathbb{N}$ samples x_i, $1 \leq i \leq N$ from a (uniform) distribution on Ω. We replace the norm in (3.1) by an empirical semi-norm

$$v_M^{\mathcal{M}} = \operatorname*{arg\,min}_{v \in \mathcal{M}} \frac{1}{N} \sum_{i=1}^{N} w_i |(v(x_i) - r(x_i))|^2 \tag{3.2}$$

with weights w_i. In our numerical tests we sample with uniform distributions and choose constant weightings $w_i \equiv w = \frac{1}{|\Omega|}$ such that the empirical semi-norm approximates the $L^2(\Omega)$-norm. As the minimizer $v_M^{\mathcal{M}}$ is also a minimizer of the loss functional with weightings $w_i \equiv 1$, we drop the weightings w_i from now on.

The central question is under which conditions the distance of the two solutions $\|v^{\mathcal{M}} - v_M^{\mathcal{M}}\|_{L^2(\Omega)}$ is low. The first step is noticing that as we draw the samples at random, every bound can only be given in probability. Using existing theory from [EST20] we obtain a quasi-best approximation in probability in two steps. First, we establish that if a *Restricted Isometry Property* (RIP) holds, the approximation error can be bounded. In the next step,

we establish, that the probability of not fulfilling the RIP decreases exponentially with the number of samples.

In [Tru21] the theory is derived in greater generality using a weighting function and more general function spaces. We omit this generality here. We denote the empirical semi-norm $\|v\|_N = \frac{1}{N}\sum_{i=1}^{N}|v(x_i)|^2$ and define the RIP in the following way.

Definition 15. Let $\mathcal{M} \subset L^2(\Omega)$. We say that the Restricted Isometry Property with constant $\delta > 0$ is fulfilled on $\mathcal{M}$ if for all $v \in \mathcal{M}$ we have

$$(1-\delta)\|v\|^2_{L^2(\Omega)} \leq \|v\|^2_N \leq (1+\delta)\|v\|^2_{L^2(\Omega)}$$

Fulfilling the RIP on certain sets ensures a quasi-best approximation using the regression approach on our ansatz set $\mathcal{M}$.

Theorem 3.0.1 ([EST20])**.** *Let $\delta > 0$. If $\mathbf{RIP}_{\{v_{\mathcal{M}}\}-(\mathcal{M}\cup\{r\})}(\delta)$ holds then*

$$\|v^{\mathcal{M}} - r\|^2 \leq \|v_N^{\mathcal{M}} - r\|^2 \leq \left(1 + 2\frac{\sqrt{1+\delta}}{\sqrt{1-\delta}}\right)\|v^{\mathcal{M}} - r\|^2.$$

Theorem 3.0.2 ([EST20])**.** *For any model class $\mathcal{M} \subset L^2(\Omega)$ with finite dimensional span and $\delta > 0$ there exists C such that*

$$\mathbb{P}[\mathbf{RIP}_{\mathcal{M}}(\delta)] \geq 1 - C\exp\left(-\frac{N}{2}\left(\frac{\delta}{k}\right)^2\right).$$

The constant C is independent of d and depends only polynomially on δ and k^{-1}. We describe the constant $k^{-1} = K(\mathcal{U}(\mathcal{M}))$ in detail in the following two definitions.

This theorem ensures that the probability of the above inequality not holding decreases exponentially with increasing number of samples N. The constant $k = K(\mathcal{U}(\mathcal{M}))$ is introduced in [EST20] to measure how much the values of the functions in $\mathcal{M}$ can vary.

Definition 16. Let $A \subset L^2(\Omega)$. Then, the *variation constant* of the set A is defined as

$$K(A) = \sup_{v\in A}\|v\|_{L^\infty(\Omega)}.$$

Definition 17. Let $A \subset L^2(\Omega)$. The *normalization operator* U acts on A by

$$U(A) = \left\{\frac{v}{\|v\|_{L^2(\Omega)}} | v \in A \setminus \{0\}\right\}.$$

Putting the definitions together we can bound the probability of $\mathbf{RIP}_{\{v_{\mathcal{M}}\}-(\mathcal{M}\cup\{r\})}(\delta)$ using Theorem 3.0.2 and $k = \max\{k_1, k_2\}$, where $k_1 = K(U(\mathcal{M} - v^{\mathcal{M}}))$ and $k_2 = K(U(\{r - v^{\mathcal{M}}\}))$, where $\mathcal{M} - v^{\mathcal{M}} = \{v - v^{\mathcal{M}} | v \in \mathcal{M}\}$. Note that k_1 measures the variation constant on the set $\mathcal{M}$ with respect to the best-approximation within the set. In contrast to that, k_2 measures the variation constant of the singleton $r - v^{\mathcal{M}}$, which means that it measures the variation constant of the best-approximation minus the exact solution r. In particular, if $r \in \mathcal{M}$ we have $K_2 = -\infty$. This means, that in this case we have to consider the variation constant of the normalized ansatz set, shifted by the solution r.

Note that the variation constant has a monotone nature in the sense that for two sets $B \subset A$ it holds $K(B) \leq K(A)$. This observation goes hand in hand with the intuitive notion that restricting the model set, if possible, is favorable. Later in this thesis, the set of tensor-trains is considered. As shown in [EST20], the scaling of the variation constant of this set is unfavorable. In [GST21] a subset of the tensor-trains based on sparsity is considered, to reduce the variation constant. While this is also possible, and we expect improved results w.r.t. the number of samples needed, for our numerical approaches, we use a simpler method of restricting the model set. We make use of a regularization term, by replacing the empirical regression problem (3.2) with a regularized version

$$v_M^{\mathcal{M}} = \operatorname*{arg\,min}_{v \in \mathcal{M}} \frac{1}{M} \sum_{i=1}^{M} |v(x_i) - r(x_i)|^2 + \kappa \|v\|,$$

where the norm $\|\cdot\|$ will be specified later. We cover this topic in more detail in Section 4.8. Note that in [Tru21] the Alternating Least Squares algorithm, which we use in later parts of the thesis is modified to optimize for a smaller variation constant. Applying this adaption to our numerical examples is an interesting topic for future research.

Remark 18. In the following chapters we will encounter theorems and considerations where an error measured in the $L^\infty(\Omega)$ norm is needed. Using a regression approach, however does only yield bounds in the $L^2(\Omega)$ norm. Here, we make use of our finite-dimensional model set. As the model set $\mathcal{M}$ is continuously embedded into $\operatorname{span}(\mathcal{M})$, this space is a finite-dimensional, and thus closed, subspace of $L^2(\Omega)$. Due to the boundedness of Ω and the regularity of the ansatz functions this subspace can also be equipped with the $L^\infty(\Omega)$ norm and as a finite dimensional space these norms are equivalent, which means that a bound in the $L^2(\Omega)$ norm is in fact also a bound in the $L^\infty(\Omega)$ norm. Note that the same argument holds if we add $\operatorname{span} r$ to the finite dimensional space.

3.1 Reformulation of the Linearized Bellman Equation as a Fixed-Point Equation

We recall the linearized Bellman equation (2.7)

$$v^\alpha(x) = \int_0^\tau e^{-\gamma t} \ell^\alpha(y_x^\alpha(t), \alpha(y_x^\alpha(t)))\, \mathrm{d}t + e^{-\gamma\tau} v^\alpha(y_x^\alpha(\tau))$$

and notice that the equation cannot be written as a regression problem. Instead, we can formulate this equation as an operator equation using the Koopman or composition operator.

Definition 19. Let $\Phi_\tau(x)$ be a continuous flow, cf. Definition 6, from $\Omega \to \Omega$ and let $w : \Omega \to \mathbb{R}$. Then the *Koopman operator* is defined as

$$K_\tau^\Phi w(x) = w(\Phi_\tau(x)).$$

As we have not specified function spaces on which the Koopman operator acts, this definition is only informal. We interpret K_τ^Φ as an operator mapping $L^\infty(\Omega)$ to $L^\infty(\Omega)$. Then

K_τ^Φ is a linear operator with operator norm $\|K_\tau^\Phi\|_{\mathcal{L}(L^\infty(\Omega))} = 1$, where $\mathcal{L}(L^\infty(\Omega))$ is the space of bounded linear operators mapping from $L^\infty(\Omega)$ to $L^\infty(\Omega)$.

If the flow is given in a feedback formulation as in (2.2) with a feedback law α, we abbreviate $K_\tau^\alpha = K_\tau^{\Phi^\alpha}$. Further, we abbreviate $\ell^\alpha(x) = \ell(x, \alpha(x))$. This allows us to reformulate the linearized Bellman equation as an operator equation

$$Iv^\alpha(x) = \int_0^\tau e^{-\gamma t} \ell^\alpha(y_x^\alpha(t)) \,\mathrm{d}t + e^{-\gamma\tau} K_\tau^\alpha v^\alpha(x) \tag{3.3}$$

and we observe that, if $\gamma > 0$, $e^{-\gamma\tau} K_\tau^\alpha$ is a contraction on $L^\infty(\Omega)$. Using the Banach fixed-point theorem, the fixed-point iteration

$$v_{k+1}^\alpha(x) = \int_0^\tau e^{-\gamma t} \ell^\alpha(y_x^\alpha(t)) \,\mathrm{d}t + e^{-\gamma\tau} K_\tau^\alpha v_k^\alpha(x)$$

converges in $L^\infty(\Omega)$. Observing, that this is a regression problem, we can fit this equation in the rest of this chapter. In the case $\gamma = 0$ we cannot prove convergence of the fixed-point iteration. Moreover, the existence of a (bounded) inverse of $I - K_\tau^\alpha$ is unclear. If the inverse can be bounded, (3.3) can be approached directly in a minimal residual way, which is exactly what we do in Section 5.1. In this case, a similar result to Theorem 3.0.2 can be found in [Eig+19].

Chapter 4

Representation of High-Dimensional Functions using Tensor Formats

In the previous chapters we saw that finding the value function is a way to obtain an optimal feedback law. While one inherent difficulty is finding the value function via functional equations, another difficulty is encoding it. While encoding the value function

$$v^* : \mathbb{R}^d \supset \Omega \to \mathbb{R}$$

in low dimensions is easily possible, it becomes a huge problem in higher dimensions. We now recall some traditional representation of functions and consider their scaling with the spatial dimension d. Here, we distinguish between grid-based and global approaches. Examples of traditional grid-based approaches are finite differences or finite elements. In the finite elements approach the set Ω is subdivided into small Ω_k with $\bigcup_k \Omega_k = \Omega$. These sets Ω_k oftentimes have simplex structure. In the case of a two dimensional set Ω this is simply a triangle. In the next step, ansatz functions φ_i are defined, whose support is only on a small number of Ω_k. In order to see the exponential increase of Ω_k with increasing spatial dimension, we consider $\Omega = [0,1]^d$ with d-dimensional volume 1. In order to cover the d dimensional cube with cubes of side length $\frac{1}{m}$, we need m^d small cubes Ω_k. The d being in the exponent of this number is referred to as *curse of dimensionality*. The main goal of fighting the curse of dimensionality is finding another ansatz that can reduce the exponential dependence on d to a polynomial dependence. While the naive attempt of taking uniformly sized finite elements can be improved by using *adaptive mesh refinement*, *sparse grid* or other methods, the success of these methods is still limited in dimensions larger than 10.

The global approach is similar in the sense that φ_i are defined. However, their support is not limited to a small subset of Ω, but instead the whole set Ω is taken. One prominent ansatz space is (multi-dimensional) polynomials. A representation of multi-dimensional polynomials is taking for every spatial dimension $1 \leq i \leq d$ a space of one dimensional polynomials of degree smaller than p, i.e. $\Pi_p := \mathbf{span}\{\phi_i | 0 \leq i \leq p\}$ and representing the multi-variate

polynomial v as

$$v(x_1, \dots, x_d) = \sum_{i_1,\dots,i_d} c_{i_1,\dots,i_d} \phi_{i_1}(x_1) \cdot \dots \cdot \phi_{i_d}(x_d). \tag{4.1}$$

It can be immediately seen that the coefficients $c_{i_1,\dots,i_n}$ can be stored in a d dimensional array $\mathbf{c}$ of size $p+1$. Thus, we still have exponential increase of the coefficients with respect to the spatial dimensions d. This can be reduced by truncating the polynomials of (multi-dimensional) degree larger than p to obtain a space that can be represented by a symmetric array of the same size. Note that in [KK18] this approach was used to solve the HJB via the Policy Iteration in up to 14 spatial dimensions.

4.1 Tensor Spaces

We now proceed by identifying the multi-dimensional arrays we considered previously as tensors in a structured way. We will later perform tensor compression techniques on these arrays. While there are very general definitions of tensor products for Hilbert spaces and Banach spaces [Hac12], using the universal property, we do not need such generality. In this section, we generally use the presentation in [Wol19], where a perspective from the numerical treatment of these tensor spaces and formats is taken. In this sense, our presentation is for the most part a subset of the presentation in [Wol19]. We use the most simple definition as a generalization of vectors and matrices, as multi-dimensional arrays. In the following we denote $\mathbb{N}_p = \{1, 2, \dots, p\}$.

Definition 20 (Tensor)**.** A *tensor* $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$ of order d is a mapping

$$\begin{aligned} \mathbf{A} : \mathbb{N}_{n_1} \times \dots \times \mathbb{N}_{n_d} &\to \mathbb{R} \\ i = (i_1, \dots, i_d) &\mapsto \mathbf{A}[i_1, \dots, i_d]. \end{aligned}$$

The number n_μ is the μ-th dimension of $\mathbf{A}$ and d is the order of the tensor. It is clear that order 2 tensors are indeed matrices and order 1 tensors are vectors. In the following, we denote the entries of the tensors by $\mathbf{A}[i_1, \dots, i_d]$ and oftentimes abbreviate $\mathbf{i} = i_1, \dots, i_d$.

Moreover, by defining a canonical addition and multiplication, the set of tensors forms a vector space.

Proposition 4.1.1. *Consider a fixed index set $\mathbb{N}_{n_1} \times \dots \times \mathbb{N}_{n_d}$. Let*

$$\mathbb{R}^{n_1,\dots,n_d} := \{\mathbf{A} : \mathbb{N}_{n_1} \times \dots \times \mathbb{N}_{n_d} \to \mathbb{R}\}.$$

Define an addition $+$ on $\mathbb{R}^{n_1,\dots,n_d}$ as

$$(\mathbf{A} + \mathbf{B})[\mathbf{i}] = \mathbf{A}[\mathbf{i}] + \mathbf{B}[\mathbf{i}].$$

Then, the tensor $0 \in \mathbb{R}^{n_1,\dots,n_d}$ with

$$0[\mathbf{i}] = 0$$

for all $\mathbf{i} \in \mathbb{N}_{n_1} \times \dots \times \mathbb{N}_{n_d}$ is the neutral element of the addition. This, together with the scalar multiplication $\cdot$

$$(\alpha \cdot \mathbf{A})[\mathbf{i}] = \alpha \mathbf{A}[\mathbf{i}]$$

makes the space $\mathbb{R}^{n_1,\dots,n_d}$ a vector space.

We next define the Frobenius inner product, which makes $\mathbb{R}^{n_1,\dots,n_d}$ a Hilbert space

Proposition 4.1.2 (Frobenius inner product)**.** *Let* $\mathbf{A},\mathbf{B} \in \mathbb{R}^{n_1,\dots,n_d}$*. The Frobenius inner product* $(\cdot,\cdot)_F$ *is defined as*

$$(\cdot,\cdot)_F : \mathbb{R}^{n_1,\dots,n_d} \times \mathbb{R}^{n_1,\dots,n_d} \to \mathbb{R}, \quad (\mathbf{A},\mathbf{B}) \mapsto \sum_{\mathbf{i}\in\mathbb{N}_{n_1}\times\cdots\times\mathbb{N}_{n_d}} A[\mathbf{i}]B[\mathbf{i}].$$

Definition 21 (Tensor product)**.** Consider for $0 < l < d$, $l,d \in \mathbb{N}$ two fixed index sets $\mathbb{N}_{n_1} \times \cdots \times \mathbb{N}_l$ and $\mathbb{N}_{l+1} \times \cdots \times \mathbb{N}_d$. The *tensor product* of the two spaces of two tensors $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_l}$ and $\mathbf{B} \in \mathbb{R}^{n_{l+1},\dots,n_d}$, denoted by $\mathbf{A}\otimes\mathbf{B}$, is defined by

$$(\mathbf{A}\otimes\mathbf{B})[i_1,\dots,i_l,i_{l+1}\dots i_d] = \mathbf{A}[i_1,\dots,i_l]\cdot\mathbf{B}[i_{l+1},\dots,i_d].$$

In particular, we have $\mathbf{A}\otimes\mathbf{B} \in \mathbb{R}^{n_1,\dots,n_l,n_{l+1},\dots,n_d}$. In this definition, it becomes clear that the tensor product is associative, i.e. $(A\otimes B)\otimes C = A\otimes(B\otimes C) = A\otimes B\otimes C$. Moreover, we have $\mathbb{R}^{n_1,\dots,n_d} = \bigotimes_{1\le i\le n}\mathbb{R}^{n_i}$. Finally, the Frobenius inner product is (uniquely) induced by the Euclidean scalar product in $\mathbb{R}^d$. We can also generalize the matrix-matrix and matrix-vector multiplication to higher order tensors in the following way.

Definition 22 (Tensor contraction)**.** Let $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$ and $\mathbf{B} \in \mathbb{R}^{m_1,\dots,m_e}$ be two tensors of order d and e, respectively. The *contraction* $\mathbf{A} *_{\mu,\nu} \mathbf{B}$ of the μ-th mode of $\mathbf{A}$ and the ν-th mode of $\mathbf{B}$ is defined entry-wise as

$$\begin{aligned}(\mathbf{A} *_{\mu,\nu} \mathbf{B})[i_1,\dots,i_{\mu-1},i_{\mu+1},\dots,i_d,j_1,\dots,j_{\nu-1},j_{\nu+1},\dots,j_e]&\\ := \sum_{p=1}^{n_\mu} \mathbf{A}[i_1,\dots,i_{\mu-1},p,i_{\mu+1},\dots,i_d]\mathbf{B}[j_1,\dots,j_{\nu-1},p,j_{\nu+1},\dots,j_e],&\end{aligned}$$

under the assumption that $n_\mu = m_\nu$. The resulting tensor

$$(\mathbf{A} *_{\mu,\nu} \mathbf{B}) \in \mathbb{R}^{n_1,\dots,n_{\mu-1},n_{\mu+1},\dots,n_d,m_1,\dots,m_{\nu-1},m_{\nu+1},\dots,m_e}$$

is of order $d+e-2$. Finally, we define the contraction on an empty set of mode pairs $\mathbf{A}*\mathbf{B}$ as the dyadic product, i.e. entry-wise we have

$$(\mathbf{A}*\mathbf{B})[i_1,\dots,i_d,j_1,\dots,j_e] = \mathbf{A}[i_1,\dots,i_d]\mathbf{B}[j_1,\dots,j_e].$$

Note that using the $*_{\mu,\nu}$ notation, associativity from the matrix multiplication is not preserved, because the indices get reordered. This can be seen by the simple example below.

Example 4.1.3. Let $\mathbf{A} \in \mathbb{R}^{n_1,n_2,n_3}$, $\mathbf{B} \in \mathbb{R}^{n_3,n_4}$, $\mathbf{C} \in \mathbb{R}^{n_4,n_5}$. Then, we have

$$\begin{aligned}\left((\mathbf{A} *_{(3,1)} \mathbf{B}) *_{(3,1)} \mathbf{C}\right)[i_1,i_2,i_5] &= \sum_{j_4=1}^{n_4}\left(\sum_{j_3=1}^{n_3}\mathbf{A}[i_1,i_2,j_3]\mathbf{B}[j_3,j_4]\right)\mathbf{C}[j_4,i_5]\\ &= \sum_{j_3=1}^{n_3}\mathbf{A}[i_1,i_2,j_3]\left(\sum_{j_4=1}^{n_4}\mathbf{B}[j_3,j_4]\mathbf{C}[j_4,i_5]\right)\\ &= \left(\mathbf{A} *_{(3,1)} (\mathbf{B} *_{(2,1)} \mathbf{C})\right)[i_1,i_2,i_5],\end{aligned}$$

which means that $(\mathbf{A} *_{(3,1)} \mathbf{B}) *_{(3,1)} \mathbf{C} = \mathbf{A} *_{(3,1)} \big((\mathbf{B} *_{(2,1)} \mathbf{C})\big)$. However, as $\mathbf{B}$ is a mode 2 tensor, i.e. a matrix, the product $\mathbf{B} *_{(3,1)} \mathbf{C}$ is not well-defined. Thus, the $*_{(\mu,\nu)}$ product is not associative. However, in combination with a reordering of the indices, every operation of the form $(\mathbf{A} *_{(\mu,\nu)} \mathbf{B}) *_{(\tilde{\mu},\tilde{\nu})} \mathbf{C}$ can be represented as $\mathbf{A} *_{(\eta,\theta)} (\mathbf{B} *_{(\tilde{\eta},\tilde{\theta})} \mathbf{C})$.

Another important operation of the set of tensors is the *reshape* operator, which essentially is a reordering of the indices. It is given as follows.

Definition 23 (Reshape)**.** For a given order $d \in \mathbb{N}$ and a permutation $(\mu_1, \dots, \mu_d)$ of $(1, \dots, d)$, a *reshape* is an isomorphism

$$\mathbf{reshape}_{(\mu_1,\dots,\mu_d)} : \mathbb{R}^{n_1,\dots,n_d} \to \mathbb{R}^{n_{\mu_1},\dots,n_{\mu_d}}$$

defined entry-wise as

$$\mathbf{reshape}_{(\mu_1,\dots,\mu_d)}(\mathbf{A})[i_{\mu_1}, \dots, i_{\mu_d}] := \mathbf{A}[i_1, \dots, i_d].$$

It is clear that for order two tensors, i.e. matrices, the operator $\mathbf{reshape}_{(2,1)}$ is the matrix transpose. Finally, we define the useful μ-th mode product, which is a multiplication of a tensor with a matrix, where only the index that is being contracted is changed.

Definition 24 (μ-th Mode Product)**.** Given a tensor $\mathbf{A} \in \mathbb{R}^{n_1,\cdots,n_d}$ and a matrix $B \in \mathbb{R}^{n_\mu,m}$, the μ-th product between $\mathbf{A}$ and B is defined as

$$(\mathbf{A} \times_\mu B)[i_1, \dots, i_d] := \sum_{j=1}^{n_\mu} \mathbf{A}[i_1, \dots, i_{\mu-1}, j, i_{\mu+1}, \dots, i_d] B[j, i_\mu]$$

Note that this can be equivalently defined as

$$\mathbf{A} \times_\mu B = \mathbf{reshape}_{(1,\dots,\mu-1,d,\mu+1,\dots,d-1)}(\mathbf{A} *_{(\mu,1)} B).$$

Definition 25. Let $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$ and $\mathbf{B} \in \mathbb{R}^{n_d,m_2,\dots,m_e}$ be order d and e tensors. By $\mathbf{A} \circ \mathbf{B}$ we denote the contraction of the last with the first index, i.e.

$$A \circ B = A *_{d,1} B.$$

This notation will be useful to represent a tensor-train in a simple way. We introduce homeomorphisms which enable us to represent tensors as matrices and vectors.

Definition 26 (Vectorization)**.** Given a bijection $\varphi : \mathbb{N}_{n_1} \times \cdots \times \mathbb{N}_{n_d} \to \mathbb{N}_{m_{\mathbf{vec}}}$ with $m_{\mathbf{vec}} = \prod_{i=1}^{d} n_i$, the mapping

$$\begin{aligned} \mathbf{Vec} : \mathbb{R}^{n_1,\dots,n_d} &\to \mathbb{R}^{m_{\mathbf{vec}}} \\ \mathbf{Vec}(\mathbf{A})[i] &= \mathbf{A}[\varphi^{-1}(i)] \end{aligned}$$

is called a vectorization. The inverse operation $\mathbf{Vec}^{-1}$ is called de-vectorization or tensorization. The choice of the bijective map does not matter, provided that the same rule is chosen in all vectorizations (and matricizations). As a convention, in this thesis, a lexicographical ordering of the indices is used, i.e.

$$\varphi(i_1, \dots, i_d) = 1 + \sum_{\mu=1}^{d} (i_\mu - 1) \prod_{\nu<\mu} n_\nu.$$

Definition 27 (Matricization)**.** Given a tensor space $\mathbb{R}^{n_1,\dots,n_d}$, let $\Gamma \subset \{1,\dots,d\}$ denote a subset of the modes and let $\Gamma^C = \{1,\dots,d\} \setminus \Gamma$ be its complement. Define $m_1 = \prod_{\mu\in\Gamma} n_\mu$ and $m_1 = \prod_{\nu\in\Gamma^C} n_\nu$. Given a bijection

$$\begin{aligned}\phi : \mathbb{N}_{n_1} \times \cdots \times \mathbb{N}_{n_d} &\to \mathbb{N}_{m_1} \times \mathbb{N}_{m_2} \\ (i_1,\dots,i_d) &\mapsto \left(\varphi_1(i_\mu|\mu\in\Gamma), \varphi_2(i_\nu|\nu\in\Gamma^C)\right),\end{aligned}$$

the mapping

$$\begin{aligned}\mathbf{Mat}_\Gamma : \mathbb{R}^{n_1,\dots,n_d} &\to \mathbb{R}^{m_1,m_2} \\ \mathbf{Mat}_\Gamma(\mathbf{A})[i,j] &= \mathbf{A}[\phi^{-1}(i,j)]\end{aligned}$$

is called the Γ-matricization. The inverse $\mathbf{Mat}_\Gamma^{-1}$ is called the de-matricization or tensorization. As with the vectorization, the choice of the bijection ϕ does not matter, provided that the same rule is chosen in every instance. We use the same lexicographical order as in the vectorization case.

$$\phi(i_1,\dots,i_d) = \left(1 + \sum_{\mu=1}^{d}(i_\mu - 1)\prod_{\substack{\nu<\mu \\ \nu\in\Gamma}} n_\nu, 1 + \sum_{\mu=1}^{d}(i_\mu - 1)\prod_{\substack{\nu<\mu \\ \nu\in\Gamma^C}} n_\nu\right).$$

Note that the notation $\mathbf{Mat}_\Gamma^{-1}$ does not contain the entirety of information. In particular the dimension of the modes, i.e. n_i has to be clear in order to use this notation. Thus, it is only possible to use this operator when the dimensions are clear from the context.

The operators we introduced are important for defining and representing tensor networks. In the context of stability of the representations of these networks, orthogonality plays an important role. In particular for the (multi-dimensional) Singular Value Decomposition (SVD) the concept of left- and right-orthogonal matrices are of central importance. Left- and right-orthogonal matrices are not only defined for square matrices, but also for rectangular matrices as follows.

Definition 28. Let $A \in \mathbb{R}^{n,m}$. Then, A is left-/right-orthogonal if its column/rows are orthogonal unit vectors. That is, A is left-orthogonal if and only if $A^TA = I_n$ and right-orthogonal if and only if $AA^T = I_m$. We say that $B \in \mathbb{R}^{n,n}$ is orthogonal, if it is both left- and right-orthogonal.

Using the matricization we can give rise to left- and right-orthogonal tensors.

Definition 29. Let $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$. Then, $\mathbf{A}$ is left-orthogonal, if $\mathbf{Mat}_{(1,\dots,d-1)}\mathbf{A}$ is a left-orthogonal matrix. Analogously, $\mathbf{A}$ is right-orthogonal, if and only if $\mathbf{Mat}_{(1)}\mathbf{A}$ is a right-orthogonal matrix.

4.2 Tensor Networks

In this section, we consider more complex networks of tensors using the operations we previously defined. In order to reduce the number of indices represented in the equations, we introduce

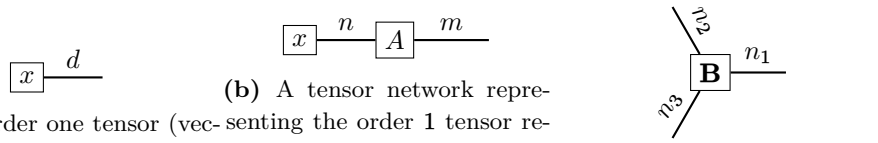
a graphical notation of tensor networks and equations. These graphical notations are widespread in the literature and while they are very helpful for compact representation of networks, certain information is lost. Thus, caution is advised when using such a representation.

We begin by defining the concept of a tensor network

Definition 30 (Tensor Network)**.** A *tensor network* is defined by a finite set of tensors T and a set of contractions C acting on (unique) pairs of tensors. The structure of a tensor network can be visualized by a graph, with nodes corresponding to T and edges corresponding to C, i.e. two nodes are connected if and only if there is a contraction performed between them. The tensor that is obtained by performing all contractions C is said to be the tensor represented by the tensor network. In case the corresponding graph is not connected, dyadic contractions, i.e. contractions with an empty set of contracted nodes, are performed between all connected components. As shown in the previous section, the order of the contractions does not matter.

We give some examples of simple tensor networks, known from linear algebra in Figure 4.1.

Note that in Figure 4.1b, it is not evident which mode of A is contracted, as we can either



(a) An order one tensor (vector) $x \in \mathbb{R}^n$

(b) A tensor network representing the order **1** tensor resulting from the matrix-vector multiplication $(Ax) \in \mathbb{R}^m$

(c) An order **3** tensor $\mathbf{B} \in \mathbb{R}^{n_1,n_2,n_3}$.

Figure 4.1: Graphical notation of simple tensors and tensor networks.

have $A \in \mathbb{R}^{n,m}$ or $A \in \mathbb{R}^{m,n}$. Thus, writing down the exact contractions is more precise. However, in most cases it is evident which contraction is to be performed and in these cases we can use the graphical notation. Another tensor contraction that will be very important in this chapter is the Singular Value Decomposition (SVD). For $A \in \mathbb{R}^{n,m}$ there exists a decomposition $U \in \mathbb{R}^{n,r}$ left-orthogonal, $\Sigma \in \mathbb{R}^{r,r}$ diagonal and $V \in \mathbb{R}^{r,m}$ right-orthogonal. Its diagrammic notation is given below. As the indices n and m appear on the l.h.s. in the full

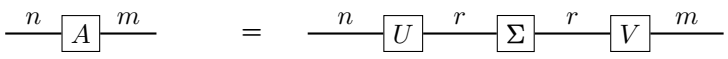


Figure 4.2: Graphical representation of the SVD

tensor A, we call these indices physical indices. The remaining indices (r) are called virtual indices as they only appear within the tensor network.

4.3 Low Rank Matrices

Low-rank matrices are a classical way of compressing data. Not only is the theory behind it understood thoroughly, but there also exists a numerical stable and efficient algorithm for

computing an optimal approximation of higher rank matrices: the *Singular Value Decomposition*(SVD). The SVD has a long history, being discovered in 1873/74 [Bel73; Jor74], see also [Ste93] for historical remarks on the discovery.

When considering a low-rank format, mathematical questions that arise are the following.

(I) Does every matrix (or tensor) posses a representation with minimal rank?

(II) If the representation exists, is it unique?

(III) Is it computable efficiently and in a numerical stable way?

The first sense of compression for a matrix we give is its minimal rank representation. Note that there exist several equivalent definitions for the matrix (row and column) rank. We use the following definition because it prepares us for the notions of ranks for higher order tensors.

Definition 31. Let $A \in \mathbb{R}^{n,m}$. The *matrix rank* of A is the smallest $r \in \mathbb{N}$ such that there exist $L \in \mathbb{R}^{n,r}$ and $R \in \mathbb{R}^{r,m}$ such that

$$A = LR.$$

A matrix $A \in \mathbb{R}^{n,n}$ has low rank, if its matrix rank r is considerably smaller than n, i.e. $r \ll n$. Storing L and R instead of A reduces the complexity of storing from $\mathcal{O}(n^2)$ entries to $\mathcal{O}(nr)$. Thus, assuming that r is small and constant, the complexity of storage does not increase quadratically with the dimension n, but only linearly. In this way, the dimension of data that can be numerically handled increases tremendously. In the matrix case, all three answers are given by the SVD and the corresponding stable algorithm. Every matrix has a unique rank and the rank is the minimal r that can be used. Moreover, using the SVD we do not obtain a unique representation. However, the significant part of the SVD, the singular values of the matrix, is unique. This is summarized precisely in the following proposition.

Proposition 4.3.1 ([GV13])**.** *Let $A \in \mathbb{R}^{n,m}$. Then, there exist left-orthogonal matrices $U \in \mathbb{R}^{n,r}$ and $V \in \mathbb{R}^{m,r}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{r,r}$, where r is the rank for the matrix A, and the diagonal elements σ_i of Σ are non-decreasing and positive, i.e. $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, such that $A = U\Sigma V^T$. The diagonal elements of Σ are called singular values and they are unique.*

This is the rank revealing or compact formulation of the SVD. There exist other formulations, where singular values can be 0. However, for our purpose this rank revealing formulation is the most helpful. We further consider a truncated SVD, which produces a best approximation of lower rank. While this property is usually attributed to [EY36], see in particular historical remarks in [Ste93] for an earlier discovery.

Theorem 4.3.2 (Eckart-Young)**.** *Let $A \in \mathbb{R}^{n,m}$ be rank r and $A = U\Sigma V^T$ be its compact SVD. Further denote by $\sigma_i = \Sigma_{ii}$ the i-th largest singular value of A. Set $r' < r$ and define Σ' by truncating the lowest $r - r'$ columns and rows of Σ, such that $\Sigma' \in \mathbb{R}^{r',r'}$. Further define $U' \in \mathbb{R}^{n,r'}$, $V' \in \mathbb{R}^{m,r'}$ by truncating the last $r - r'$ rows of U, V respectively. Then*

$$A' = U'\Sigma' V'^T$$

is the best rank r' approximation of A, i.e. it holds

$$\|A - A'\| = \min\{\|A - \hat{A}\| | \hat{A} \in \mathbb{R}^{n,m}, \mathrm{rank}(\hat{A}) \leq r'\},\}$$

where $\|\cdot\|$ is either the Frobenius norm $\|\cdot\|_F$, or the spectral norm $\|\cdot\|_2$. The errors are given by

$$\|A - A'\|_F = \left(\sum_{k>r'} \sigma_k(A)^2\right)^{1/2},$$
$$\|A - A'\|_2 = \sigma_{r'+1}(A).$$

The final question of numerical stability was answered in 1965 in [GK65].

Remark 32. Using the μ-th mode product, cf. Definition 24, we can reformulate the matrix product in the SVD.

$$A = \Sigma \times_1 U \times_2 V.$$

Our aim is to generalize the results for higher order tensors. However, the generalization is not straight-forward and a numerical stable extension was only discovered **35** years later in [DDV00]. A major difficulty of the generalization is that many equivalent formulations of the unique rank r of a matrix do not lead to equivalent generalizations in higher order tensors. These equivalences are given in the following lemma.

Proposition 4.3.3. *Let $A \in \mathbb{R}^{n,n}$ and $r \in \mathbb{N}$, $r \leq n$. Then, the following are equivalent*

(I) A has matrix rank r.

(II) There exist $b_i, b_j \in \mathbb{R}^n$, $1 \leq i, j \leq n$ such that

$$A = \sum_{i=1}^{r} b_i \otimes b_j. \tag{4.2}$$

(III) There exists an SVD with $U \in \mathbb{R}^{n,r}$ left-orthogonal, $\Sigma \in \mathbb{R}^{r,r}$ diagonal, $V \in \mathbb{R}^{n,r}$ right-orthogonal such that

$$A = U\Sigma V^T.$$

(IV) r is the smallest number, such that there exist r-dimensional subspaces $U \subset \mathbb{R}^n$ and $V \in \mathbb{R}^m$, such that A is an element of the induced tensor space $U \otimes V \subset \mathbb{R}^{n,m}$.

(V) We have that $dim(Im(A)) = r$.

4.4 Tensor Low-Rank Formats

The (historically) first generalization of the matrix rank for higher order tensors is the *canonical format*, published in 1927 [Hit27]. Other names of the decomposition in the literature are *PARAFAC* (Parallel Factor analysis) or *Candecomp* (Canonical Decomposition). It relies on (4.2) in Proposition 4.3.3 and reads

Proposition 4.4.1. *Let $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$. Then, there exists a unique minimal r and vectors $b_{il} \in \mathbb{R}^{n_l}$, $1 \leq i \leq r$, $1 \leq l \leq d$, such that*

$$\mathbf{A} = \sum_{i=1}^{r} \bigotimes_{l=1}^{d} b_{il}.$$

This proposition answers the first two questions. However, there does not exist a numerical stable way of computing such a representation. In fact, finding the lowest r such that this is possible is *NP-hard* [Hås90] and low-rank approximation using the canonical format can be ill-posed [DL08]. Thus, the canonical format is not optimal for the approximation problems we consider later. A major breakthrough was the *Tucker format*. Originally introduced in 1966 [Tuc66] for the special case $d = 3$, it became numerically feasible by reinterpreting it as a generalized SVD in [DDV00]. The idea is finding minimal subspaces, such that the underlying tensor can be represented. This corresponds to a generalization of Property III in Proposition 4.3.3 in the following sense.

Consider $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$ and subspaces $V_\mu \subset \mathbb{R}^{n_\mu}$ such that $\mathbf{A} \in \bigotimes_{i=1}^{d} V_\mu$. Choosing an orthonormal basis of V_μ, with basis change matrix $U_\mu \in \mathbb{R}^{r_\mu,n_\mu}$, we can represent $\mathbf{A}$ by

$$\mathbf{A}[i_1,\dots,i_d] = \sum_{j_1=1}^{r_1} \sum_{j_2=1}^{r_2} \cdots \sum_{j_d=1}^{r_d} \mathbf{C}[j_1,\dots,j_d] U_1[j_1,i_1] U_2[j_2,i_2] \dots U_d[j_d,i_d].$$

We say that the tensor $\mathbf{C} \in \mathbb{R}^{r_1,\dots,r_d}$ is the core tensor corresponding to the tensor $\mathbf{A}$. In the following we formalize this subspace approach and give notion to a rank of a tensor in this context.

Proposition 4.4.2. *Let $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$. Then, there exits $(r_1,\dots,r_d) \in \mathbb{N}^d$, matrices $U_\mu \in \mathbb{R}^{r_\mu,n_\mu}$ right-orthogonal (i.e. $U_\mu U_\mu^T = I_{r_\mu}$) and a core-tensor $\mathbf{C} \in \mathbb{R}^{r_1,\dots,r_d}$, such that*

$$\mathbf{A} = \mathbf{C} \times_1 U_1 \times_2 U_2 \cdots \times_d U_d.$$

We call this representation a Tucker representation*. The d-tuple $\mathbf{r} = (r_1,\dots,r_d)$ is called the representation rank and is associated with the particular representation. In contrast, the* Tucker-rank *of $\mathbf{A}$ is defined as the minimal d-tuple, such that there exists a Tucker representation of $\mathbf{A}$ with rank $\mathbf{r}$.*

Remark 33. This proposition is not directly taken from [DDV00], but instead from [Wol19]. In [DDV00] more emphasis is given on the properties of the core tensors, which we do not need. Moreover, note that the well-definedness of the Tucker-rank is not obvious. This will be treated below.

Definition 34 (Partial Ordering)**.** Let $\mathbf{s} = (s_1,\dots,s_d) \in \mathbb{N}^d$ and $\mathbf{t} = (t_1,\dots,t_d) \in \mathbb{N}^d$. Then, we define a partial order $\preceq$ on $\mathbb{N}^d$ as follows

$$\mathbf{s} \preceq \mathbf{t} \iff s_i \leq t_i \text{ for all } 1 \leq i \leq d.$$

We next consider the well-definedness of the Tucker-rank. The first step is considering the Higher-Order-SVD (HOSVD) algorithm, stated in Algorithm 4, where a Tucker representation is computed. We will then show that this representation has minimal rank. We finish our considerations by proving that the Tucker-rank is well-defined.

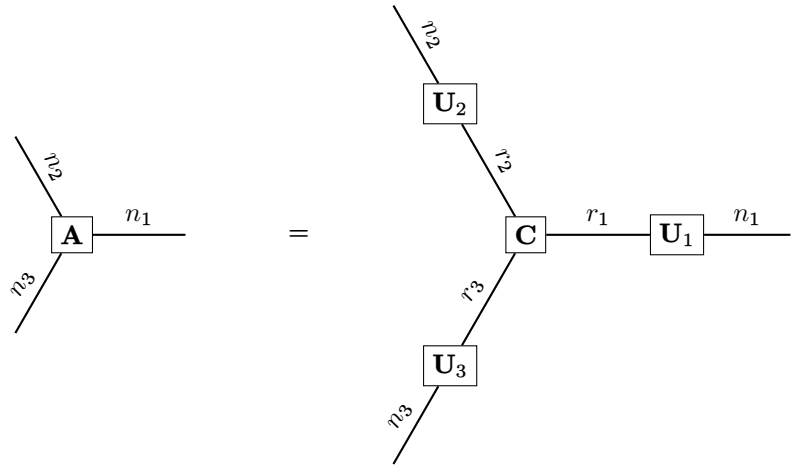


Figure 4.3: An order **3** tensor and a Tucker representation.

Algorithm 4: Higher-Order Singular Value Decomposition (HOSVD)

input : An order d tensor $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$
output : Core tensor $\mathbf{C} \in \mathbb{R}^{r_1,\dots,r_d}$ and basis matrices $U_1 \in \mathbb{R}^{r_1,n_1}, \dots, U_d \in \mathbb{R}^{r_d,n_d}$.
for $\mu = 1, \dots, d$ **do**
  Calculate $\tilde{U}_\mu S_\mu V_\mu^T = \mathbf{SVD}(\mathbf{Mat}_{(\mu)}\mathbf{A})$
  Set $U_\mu = \tilde{U}_\mu^T$
end
Set $\mathbf{C} = \mathbf{A} \times_1 U_1^T \times_2 U_2^T \cdots \times_d U_d^T$

Theorem 4.4.3 ([DDV00; Wol19])**.** *Given a tensor $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$, let $\mathbf{C}$ be the core tensor and $U_1, \dots, U_d$ the basis matrices obtained by the HOSVD in Algorithm 4. Then,*

$$\mathbf{A} = \mathbf{C} \times_1 U_1 \times_2 U_2 \cdots \times_d U_d \tag{4.3}$$

is a Tucker representation of $\mathbf{A}$ of minimal rank. The obtained representation rank, which thereby is also the Tucker-rank of $\mathbf{A}$, is related to the matrix rank of matricizations of the $\mathbf{A}$ via

$$\textit{Tucker-rank}\ (\mathbf{A}) = \Big(\mathbf{rank}\big(\mathbf{Mat}_{(1)}(\mathbf{A})\big), \dots, \mathbf{rank}\big(\mathbf{Mat}_{(d)}(\mathbf{A})\big)\Big). \tag{4.4}$$

Proof. We first show that (4.3) holds. From the SVD-based definition of U_μ, it is clear that for all μ we have $U_\mu^T U_\mu \mathbf{Mat}_{(\mu)}(\mathbf{A}) = \mathbf{Mat}_{(\mu)}(\mathbf{A})$. This is equivalent to $\mathbf{A} \times_\mu U_\mu^T U_\mu = \mathbf{A}$. Therefore, it follows that

$$\begin{aligned}
\mathbf{C} \times_1 U_1 \cdots \times_d U_d &= \big(\mathbf{A} \times_1 U_1^T \cdots \times_d U_d^T\big) \times_1 U_1 \cdots \times_d U_d \\
&= \mathbf{A} \times_1 U_1^T U_1 \cdots \times_d U_d^T U_d \\
&= \mathbf{A},
\end{aligned}$$

as claimed. From the SVD-based definition of U_μ, it furthermore follows that $U_\mu \in \mathbb{R}^{r_\mu,n_\mu}$, where $r_\mu = \mathbf{rank}(\mathbf{Mat}_{(\mu)}(\mathbf{A}))$ is the rank of the μ-th mode matricization of $\mathbf{A}$, which shows

that for the representation rank (4.4) holds. It remains to show that there cannot be a Tucker decomposition with smaller representation rank. Assume towards a contradiction that

$$\mathbf{A} = \mathbf{C}' \times_1 U_1' \times_2 U_2' \cdots \times_d U_d'$$

is a Tucker representation of rank $(r_1', \ldots, r_d')$, where $r_\mu' < \mathbf{rank}(\mathbf{Mat}_{(\mu)}\mathbf{A})$. With

$$\mathbf{R} := \mathbf{C}' \times_1 U_1' \cdots \times_{\mu-1} U_{\mu-1}' \times_{\mu+1} U_{\mu+1}' \cdots \times_d U_d',$$

we have

$$\mathbf{A} = \mathbf{C}' \times_1 U_1' \times_2 U_2' \cdots \times_d U_d' = \mathbf{R} \times_\mu U_\mu'.$$

By assumption, $U_\mu' \in \mathbb{R}^{r_\mu', n_\mu}$ and therefore,

$$\mathbf{Mat}_{(\mu)}(\mathbf{A}) = U_\mu' \, \mathbf{Mat}_{(\mu)}(\mathbf{R})$$

is a rank r_μ' factorization of $\mathbf{Mat}_{(\mu)}(\mathbf{A})$ in contradiction to the claim that $\mathbf{rank}\big(\mathbf{Mat}_{(\mu)}(\mathbf{A})\big) > r_\mu'$. □

We have now established, that the three questions regarding the low-rank format are answered positively. The relationship of the HOSVD with the SVD allows us to consider a truncated HOSVD using Theorem 4.3.2. In that sense, we simply replace the SVD step in Algorithm 4 by the truncated SVD to obtain a matrix, and thus also a core tensor, of lower rank. Here, the optimality of the approximation in the matrix case is replaced by a quasi-optimality.

Theorem 4.4.4 ([DDV00; Wol19])**.** *Given a tensor $\mathbf{A} \in \mathbb{R}^{n_1,\ldots,n_d}$, let $\mathbf{C}$ be the core tensor and $U_1, \ldots, U_d$ the basis obtained by the rank $\mathbf{r}$ truncated HOSVD. These define a rank $\mathbf{r}$ Tucker representation*

$$\mathbf{A} = \mathbf{C} \times_1 U_1 \cdots \times_d U_d$$

of a tensor $\mathbf{A}'$, which is a quasi-best rank $\mathbf{r}$ approximation of $\mathbf{A}$, in the sense that

$$\|\mathbf{A} - \mathbf{A}'\|_F \leq \sqrt{d} \cdot \min\{\|\mathbf{A} - \mathbf{B}\|_F \mid \mathbf{B} \in \mathbb{R}^{n_1,\ldots,n_d}, \ \textit{Tucker-rank } (\mathbf{B}) \preceq \mathbf{r}\}.$$

The format does fight the curse of dimensionality by reducing the dimension of the tensor from n^d to r^d in the core tensor. However, the order d of the tensor is still in the exponent, making this format infeasible for higher order tensors, i.e. $d > 10$. As we do not use this representation for our numerical part, we do not dive further into the Tucker format. For further considerations we refer the reader to [Wol19].

4.5 Tensor-Train Decomposition

We now come to the tensor format that is used within this thesis. In the mathematical community, the *Tensor-Train* (TT) format was introduced in 2011 [Ose11]. However, the format was known in physics under the name *Matrix Product States* (MPS) before. This format has been used successfully in a broad variety of scientific fields, including quantum physics and chemistry [Whi92; Sch11; Sza+15] and machine learning [YKT17; SS16; KG19;

Gel+19; Goe+20]. Note that this decomposition fits as a special case into a more general framework, the hierarchical tensor formats [HK09], which will not be covered here.

Similar to the Tucker format, the TT format is a subspace approach. However, instead of taking the tensor product of these subspaces, they are nested in a hierarchical way. The aim is then finding minimal subspaces V_i such that for $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$,

$$\begin{aligned}
&V_1 \subset \mathbb{R}^{n_1} && \text{with } \mathbf{A} \in V_1 \otimes \mathbb{R}^{n_2,\dots,n_d}\\
&V_2 \subset V_1 \otimes \mathbb{R}^{n_2} \subset \mathbb{R}^{n_1,n_2} && \text{with } \mathbf{A} \in V_2 \otimes \mathbb{R}^{n_3,\dots,n_d}\\
&V_3 \subset V_2 \otimes \mathbb{R}^{n_3} \subset \mathbb{R}^{n_1,n_2,n_3} && \text{with } \mathbf{A} \in V_3 \otimes \mathbb{R}^{n_4,\dots,n_d}\\
&\quad\vdots\\
&V_{d-1} \subset \mathbb{R}^{n_{d-1}} \otimes V_{d-2} \subset \mathbb{R}^{n_1,\dots,n_{d-1}} && \text{with } \mathbf{A} \in V_{d-1} \otimes \mathbb{R}^{n_d}.
\end{aligned}$$

In the following we find a representation of the above spaces as a sequence of order **3** tensors. The idea is similar to the Tucker format construction. However, due to the nestedness it is more technical.

For $\mu \in \mathbb{N}_\mu$ choose orthogonal bases $\{\mathbf{b}_{\mu,1},\dots,\mathbf{b}_{\mu,r_\mu}\}$ of V_μ. Note that $\mathbf{b}_{\mu,j}$ is an order μ tensor. By construction there exist $u_{\mu,j,k} \in \mathbb{R}^{n_\mu}$ such that

$$\mathbf{b}_{\mu,j} = \sum_{k=1}^{r_\mu} \mathbf{b}_{\mu-1,k} \otimes u_{\mu,j,k} \quad \text{for all } k \le r_\mu. \tag{4.5}$$

Representing this basis as a tensor

$$\mathbf{W}_\mu[i_1,\dots,i_\mu,j] = \mathbf{b}_{\mu,j}[i_1,\dots,i_\mu]$$

and setting

$$\mathbf{U}_\mu[i,j,k] = u_{\mu,i,k}[j]$$

we can write (4.5), using the contraction $\circ$ defined in Definition 25, in tensor form as

$$\begin{aligned}
\mathbf{W}_\mu[i_1,\dots,i_\mu,k] &= \sum_{i=1}^{r_{\mu-1}} \mathbf{W}_{\mu-1}[i_1,\dots,i_{\mu-1},i] u_{\mu,i,k}[i_\mu]\\
&= \sum_{i=1}^{r_{\mu-1}} \mathbf{W}_{\mu-1}[i_1,\dots,i_{\mu-1},i] \mathbf{U}_\mu[i,i_\mu,k]\\
&= (\mathbf{W}_{\mu-1} \circ \mathbf{U}_\mu)[i_1,\dots,i_\mu,k].
\end{aligned}$$

This yields a recursive formulation of $\mathbf{W}_\mu$. As $\mathbf{A} \in V_{d-1} \otimes \mathbb{R}^{n_d}$, there exist $u_{d,j} \in \mathbb{R}^{n_d}$, such that

$$\mathbf{A} = \sum_{j=1}^{r_{d-1}} \mathbf{b}_{d-1,j} \otimes u_{d,j}.$$

Writing $\mathbf{U}_d[i,j] = u_{d,i}[j]$, we obtain

$$\begin{aligned}\mathbf{A} &= \mathbf{W}_{d-1} \circ \mathbf{U}_d \\ &= (\mathbf{W}_{d-2} \circ \mathbf{U}_{d-1}) \circ \mathbf{U}_d \\ &= ((\mathbf{W}_{d-3} \circ \mathbf{U}_{d-2}) \circ \mathbf{U}_{d-1}) \circ \mathbf{U}_d \\ &\vdots \\ &= \mathbf{W}_1 \circ \mathbf{U}_2 \circ \mathbf{U}_3 \circ \mathbf{U}_4 \cdots \circ \mathbf{U}_{d-1} \circ \mathbf{U}_d \\ &= \mathbf{U}_1 \circ \mathbf{U}_2 \circ \mathbf{U}_3 \circ \mathbf{U}_4 \cdots \circ \mathbf{U}_{d-1} \circ \mathbf{U}_d,\end{aligned}$$

where we defined $\mathbf{U}_1 = \mathbf{W}_1$. This tensor network can be represented as a graph as in Figure 4.4.

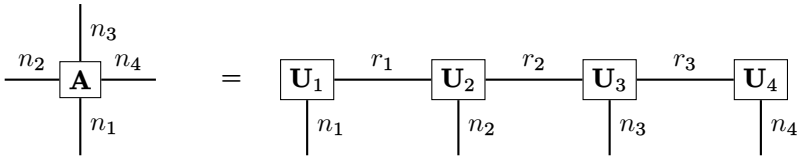


Figure 4.4: An order 4 tensor and a tensor-train representation.

We define the set of tensor-trains in $\mathbb{R}^{n_1,\dots,n_d}$ in the following way.

Definition 35 ([Ose11; Wol19])**.** Let $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$ be an order d tensor. A factorization

$$\mathbf{A} = \mathbf{U}_1 \circ \mathbf{U}_2 \circ \cdots \circ \mathbf{U}_d,$$

where $\circ$ is defined as in Definition 25 and $\mathbf{U}_1 \in \mathbb{R}^{n_1,r_1}$, $\mathbf{U}_i \in \mathbb{R}^{r_{i-1},n_i,r_i}$, $\mathbf{2} \leq i \leq d-1$, $\mathbf{U}_d \in \mathbb{R}^{r_{d-1},n_d}$, is called *tensor-train representation* of $\mathbf{A}$. We say that $\mathbf{U}_i$ are *component tensors*. The tuple of the dimensions $(r_1,\dots,r_{d-1})$ is called the representation rank of $\mathbf{A}$ and is associated with the specific representation. In contrast to that, the *tensor-train rank* (TT-rank) is defined as the minimal rank tuple $\mathbf{r} = (r_1,\dots,r_{d-1})$, such that there exists a TT representation of $\mathbf{A}$ with representation rank equal to $\mathbf{r}$. The representation is said to be orthogonalized if for some μ, the component tensors $\mathbf{U}_1,\dots,\mathbf{U}_{\mu-1}$ are left-orthogonal and $\mathbf{U}_{\mu+1},\dots,\mathbf{U}_d$ are right-orthogonal. The index μ of the possibly non-orthogonal component tensor is called the core position.

As in the case of the Tucker decomposition, the existence of a TT decomposition is clear, because it is possible to set each subspace V_i to the whole space $\mathbb{R}^{n_i}$. However, the existence of a minimal rank w.r.t. to the partial ordering as in Definition 34 is again not obvious. The definition via subspaces yields a close relation to the SVD. Thus, similar to the Tucker format, we first state the generalization of the SVD and then give a constructive proof.
In order to prove that the TTSVD calculates a valid TT representation, we need the following lemma, where we omit the proof.

Lemma 4.5.1 ([Wol19])**.** *Given left-orthogonal tensors* $\mathbf{V} \in \mathbb{R}^{n_1,\dots,n_{\mu-1},r_{\mu-1}}$ *and* $\mathbf{W} \in \mathbb{R}^{r_{\mu-1},n_\mu,r_\mu}$ *the contraction* $\mathbf{V} \circ \mathbf{W}$ *is also left-orthogonal. Analogously, given right-orthogonal tensors* $\mathbf{W} \in \mathbb{R}^{r_{\mu-1},n_{\mu-1},r_\mu}$ *and* $\mathbf{V} \in \mathbb{R}^{r_\mu,n_\mu,\dots,n_d}$ *the contraction* $\mathbf{W} \circ \mathbf{V}$ *is right-orthogonal.*

Algorithm 5: Tensor-Train Singular Value Decomposition (TTSVD)

input : An order d tensor $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$

output : Component tensors $\mathbf{U}_1 \in \mathbb{R}^{n_1,r_1}$, $\mathbf{U}_i \in \mathbb{R}^{r_{i-1},n_i,r_i}$, $2 \leq i \leq d-1$, $\mathbf{U}_d \in \mathbb{R}^{r_{d-1},n_d}$.

Set $\mathbf{A}_1 = \mathbf{A}$.

Calculate $V_1\Sigma_1 W_1 = \mathrm{SVD}(\mathrm{Mat}_{(1)}(\mathbf{A}_1))$

Set $\mathbf{U}_1 = V_1$ and $\mathbf{A}_2 = \mathrm{Mat}_{(1)}^{-1}(\Sigma_1 W_1) \in \mathbb{R}^{r_1,n_2,\dots n_d}$.

for $\mu = 1 \dots d-1$ **do**

  Calculate $V_\mu \Sigma_\mu W_\mu = \mathrm{SVD}(\mathrm{Mat}_{(1,2)}(\mathbf{A}_\mu))$,

  where $V_\mu \in \mathbb{R}^{r_{\mu-1}\cdot n_\mu, r_\mu}, \Sigma_\mu \in \mathbb{R}^{r_\mu,r_\mu}, W_\mu \in \mathbb{R}^{r_\mu,n_{\mu+1},\dots,n_d}$

  Set $\mathbf{U}_\mu = \mathrm{Mat}_{(1,2)}^{-1} V_\mu$ and $\mathbf{A}_{\mu+1} = \mathrm{Mat}_{(1)}^{-1}(\Sigma_\mu W_\mu) \in \mathbb{R}^{r_\mu,n_{\mu+1},\dots,n_d}$

end

Set $\mathbf{U}_d = \mathbf{A}_d$.

This allows us to prove that the TTSVD obtains a TT-representation with minimal rank.

Theorem 4.5.2 ([HRS12b; Wol19])**.** *For every given tensor $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$, the TTSVD obtains a valid tensor-train representation of $\mathbf{A}$. The representation rank is related to the rank of matricizations of $\mathbf{A}$ via*

$$r_\mu = \mathrm{rank}(\mathrm{Mat}_{(1,\dots,\mu)}(\mathbf{A})). \tag{4.6}$$

Additionally, the representation obtained by the TTSVD is right-orthogonal, which means that the core position is d.

Proof. Let $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$. We compute $V_1\Sigma_1 W_1 = \mathrm{SVD}(\mathrm{Mat}_{(1)}(\mathbf{A}))$ and set $\mathbf{U}_1 = V_1 \in \mathbb{R}^{n_1,r_1}$ and $\mathbf{A}_2 = \mathrm{Mat}_{(1)}^{-1}(\Sigma_1 W_1) \in \mathbb{R}^{r_1,n_2,\dots,n_d}$. Note that in particular we have $\mathbf{A} = \mathbf{U}_1 \circ \mathbf{A}_2$, that $\mathbf{U}_1$ is right-orthogonal and that (4.6) is fulfilled for this core.

In the next step, we compute

$$V_2\Sigma_2 W_2 = \mathrm{SVD}(\mathrm{Mat}_{(1,2)}\mathbf{A}_2)$$

and set $\mathbf{U}_2 = \mathrm{Mat}_{(1,2)}^{-1}(V_2) \in \mathbb{R}^{r_1,n_2,r_2}$ and $\mathbf{A}_3 = \mathrm{Mat}_{(1)}^{-1}(\Sigma_2 W_2) \in \mathbb{R}^{r_2,n_3,\dots,n_d}$. Note that again we have $\mathbf{A}_2 = \mathbf{U}_2 \circ \mathbf{A}_3$ and thus $\mathbf{A} = \mathbf{U}_1 \circ \mathbf{U}_2 \circ \mathbf{A}_3$. Moreover, $\mathbf{U}_2$ is right-orthogonal. In order to prove that (4.6) is fulfilled, we observe that Lemma 4.5.1 ensures right-orthogonality of $\mathbf{V} = \mathbf{U}_1 \circ \mathbf{U}_2$. Thus, $\mathrm{Mat}_{(1,2)}(\mathbf{V})\Sigma_2 W_2$ is a valid SVD of $\mathrm{Mat}_{(1,2)}(\mathbf{A})$. As the SVD is rank-revealing we obtain that (4.6) is fulfilled. We can inductively show that (4.6), the right-orthogonality and

$$\mathbf{U}_1 \circ \dots \circ \mathbf{U}_\mu \circ \mathbf{A}_{\mu+1} = \mathbf{A} \tag{4.7}$$

holds for any $\mu < d-1$.

Finally, in the final SVD we obtain $\mathbf{A}_d \in \mathbb{R}^{r_{d-1},n_d}$. By (4.7) we have

$$\mathbf{U}_1 \circ \dots \circ \mathbf{U}_d = \mathbf{A}$$

and by the same argumentation as in the previous case we also have that (4.6) holds. This concludes the proof. □

The following corollary shows that the rank is indeed minimal.

Corollary 4.5.3. *For every tensor $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$, the tensor-train rank $\mathbf{r} = (r_1, \dots, r_{d-1})$ is given by*

$$r_\mu = \mathbf{rank}(\mathbf{Mat}_{(1,\dots,\mu)}\mathbf{A}). \tag{4.8}$$

Proof. From Theorem 4.5.2 we see that for every tensor there exists a TT representation with ranks as in (4.8). Towards a contradiction assume that there exists a TT representation with $r_\mu < \mathbf{rank}(\mathbf{Mat}_{(1,\dots,\mu)}\mathbf{A})$ for a tensor $\mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$. Then, there exist $\mathbf{U}_i$, $1 \leq i \leq d$ such that

$$\mathbf{A} = (\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_\mu) \circ (\mathbf{U}_{\mu+1} \circ \cdots \circ \mathbf{U}_d) =: \mathbf{V} \circ \mathbf{W}$$

with $\mathbf{V} \in \mathbb{R}^{n_1,\dots,n_\mu,r_\mu}$ and $\mathbf{W} \in \mathbb{R}^{r_\mu,n_{\mu+1},\dots,n_d}$. The identity

$$\mathbf{Mat}_{(1,\dots,\mu)}(\mathbf{A}) = \mathbf{Mat}_{(1,\dots,\mu)}(\mathbf{V})\,\mathbf{Mat}_{(1)}(\mathbf{W})$$

reveals a rank r_μ factorization of $\mathbf{Mat}_{(1,\dots,\mu)}(\mathbf{A})$, which contradicts our assumption that $r_\mu < \mathbf{rank}(\mathbf{Mat}_{(1,\dots,\mu)}\mathbf{A})$. This concludes the proof. □

We now have a way to represent higher order tensors in $\mathcal{O}(d)$, given that the ranks of certain matricizations are low. Another property of the SVD in the matrix case is its optimal approximation property, cf. Theorem 4.3.2. This property generalizes to the TTSVD as a quasi-optimality.

Theorem 4.5.4 ([HRS12b; Wol19])**.** *For a given rank tuple $\mathbf{r}^* = (r_1^*, \dots, r_d^*)$, for every tensor $\mathbf{A} \in \mathbb{R}^{n_1,\dots n_d}$, the tensor $\mathbf{A}^*$ obtained by truncating the SVDs in the TTSVD to rank r_μ^* fulfills*

$$\|\mathbf{A} - \mathbf{A}^*\|_F \leq \sqrt{d-1} \min_{\mathbf{B} \in \mathcal{M}_{\preceq \mathbf{r}^*}(\mathbb{R}^{n_1,\dots,n_d})} (\|\mathbf{A} - \mathbf{B}\|_F),$$

where

$$\mathcal{M}_{\preceq \mathbf{r}^*}(\mathbb{R}^{n_1,\dots,n_d}) = \{\mathbf{B} \in \mathbb{R}^{n_1,\dots,n_d} \,|\, \textit{TT-rank}(\mathbf{B} \preceq \mathbf{r}^*)\}.$$

We have now found a set of ansatz functions which are capable of beating the curse of dimensionality, given that the ranks stay constant or moderate with increasing order.

While the set of tensor-trains of fixed ranks has some interesting properties, in particular, it forms a smooth manifold, we do not cover these properties but instead refer to [Wol19], where the manifold structure and many computational aspects are covered in detail. We finalize this section by showing how to move the core position within a tensor-train representation. This will be essential in the following section.

Let $\mathbf{U}_1 \circ \dots \mathbf{U}_d = \mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$ be a tensor-train representation, where the core position is μ. Then, we can move the core to position $\mu + 1$ by computing a QR decomposition of $\mathbf{U}_\mu$ as follows. Compute

$$\mathrm{QR}(\mathbf{Mat}_{(1,2)}(\mathbf{U}_\mu)) = QR, \quad Q \in \mathbb{R}^{r_{\mu-1}\cdot n_\mu, r_\mu}, R \in \mathbb{R}^{r_\mu, r_\mu}$$

and set $\mathbf{U}_\mu = \mathbf{Mat}_{(1,2)}^{-1}(Q) \in \mathbb{R}^{r_{\mu-1},n_\mu,r_\mu}$ and update $\mathbf{U}_{\mu+1} = R \circ \mathbf{U}_{\mu+1} \in \mathbb{R}^{r_\mu,n_{\mu+1},r_{\mu+1}}$. This yields a right-orthogonal core $\mathbf{U}_\mu$ and now the core $\mathbf{U}_{\mu+1}$ is not necessarily orthogonal. Similarly, the core can be moved to the left by computing a RQ decomposition. Note that for the same task the SVD can be used as well.

4.6 Tensor-Train Representation of High-Dimensional Functions

In this section, we return to the problem of representing high-dimensional functions and apply the TT-format to the coefficient tensors.

Recall (4.1), i.e.

$$v(x_1, \dots, x_d) = \sum_{i_1,\dots,i_d} c_{i_1,\dots,i_d} \phi_{i_1}(x_1) \cdot \dots \cdot \phi_{i_d}(x_d),$$

where $\phi_{i_j}(\cdot), 1 \le i \le d, 1 \le j \le n_i$ are one-dimensional ansatz-functions. We replace the coefficient tensor $\mathbf{C}[i_1, \dots, i_d] = c_{i_1,\dots,i_d}$ with a TT-representation

$$\mathbf{C} = \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d$$

and denote by $\mathbf{\Phi}_i : \mathbb{R} \to \mathbb{R}^{n_i}$, $x \mapsto (\phi_{i_1}(x), \phi_{i_2}(x), \dots, \phi_{n_i}(x))$ and the above equation can be written as

$$\begin{aligned} v(x_1, \dots, x_d) &= (((\mathbf{C} \circ \mathbf{\Phi}_d(x_d)) \circ \mathbf{\Phi}_{d-1}(x_{d-1})) \circ \cdots \circ \mathbf{\Phi}_1(x_1)) && (4.9) \\ &= ((\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d \circ \mathbf{\Phi}_d(x_d)) \circ \cdots \circ \mathbf{\Phi}_1(x_1)). \\ &= (\mathbf{U}_1 \circ \cdots \circ (\mathbf{U}_d \circ \mathbf{\Phi}_d(x_d)) \circ \cdots \circ \mathbf{\Phi}_1(x_1)). && (4.10) \end{aligned}$$

Note that because $\mathbf{\Phi}_i$ maps a real number to an order $\mathbf{1}$ tensor, every $\circ$ operation with $\mathbf{\Phi}_i$ involved reduces the order of the resulting tensor by one. Thus, the resulting tensor is of order $\mathbf{0}$, i.e. a number in $\mathbb{R}$. While the representation using explicit formulas is not easy to follow, the tensor network becomes more clear with the graphical notation as below.

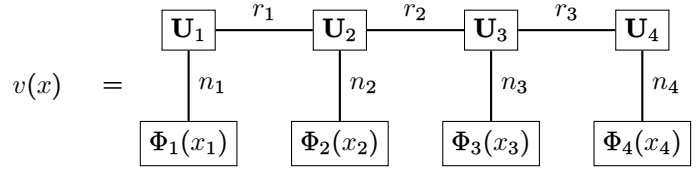


Figure 4.5: A 4-dimensional function and a TT-representation of its coefficient tensor.

In the following, we compare the complexity of evaluating functions in the TT-format as in (4.10) and the complexity when the full tensor as in (4.9) is considered. We also consider other common operations.

4.6.1 Evaluation of the Function in TT Representation

We assume that $n_i = n$ and that $r_i = r$. In order to evaluate v at a certain point $x \in \mathbb{R}^d$, we have to compute $\mathbf{\Phi}_i(x_i)$ for all i. This corresponds to $n \cdot d$ evaluations of one-dimensional functions, where we assume that these evaluations are $\mathcal{O}(\mathbf{1})$. The contraction is performed from left-to-right (or from right-to-left) and is of complexity $\mathcal{O}(dnr^2)$, because for every component tensor of dimension $r \times n \times r$ one vector of size r and one vector of size n has to be contracted, leaving a vector of size r. This compares to the exponential scaling of the full-tensor formulation, where the function evaluation is $\mathcal{O}(n^d + dn)$.

4.6.2 Computing the Gradient of the Function in TT Representation

We assume that $n_i = n$ and that $r_i = r$. In order to evaluate ∇v at a certain point $x \in \mathbb{R}^d$, we have to compute $\mathbf{\Phi}_i(x_i)$ and $\mathbf{\Phi}_i'(x_i)$ for all i, where $\mathbf{\Phi}_i'(x_i)$ is the derivative of the one-dimensional functions ϕ_i' stacked into a vector. Next we observe that because of the tensor structure of our basis, the partial derivative $\frac{\partial v}{\partial x_i}$ is given by

$$\frac{\partial}{\partial x_i} v(x_1, \dots, x_d) = \mathbf{C} \circ \mathbf{\Phi}_d(x_d) \circ \cdots \circ \mathbf{\Phi}_{i+1}(x_{i+1}) \circ \mathbf{\Phi}_i'(x_i) \circ \mathbf{\Phi}_{i-1}(x_{i-1}) \circ \cdots \circ \mathbf{\Phi}_1(x_1)$$

and thus

$$\frac{\partial}{\partial x_i} v(x_1, \dots, x_d) = \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d \circ \mathbf{\Phi}_d(x_d) \circ \cdots \circ \mathbf{\Phi}_{i+1}(x_{i+1}) \circ \mathbf{\Phi}_i'(x_i) \circ \mathbf{\Phi}_{i-1}(x_{i-1}) \circ \cdots \circ \mathbf{\Phi}_1(x_1).$$

Noticing the similarity of the above formulas with (4.9) and (4.10) we can again estimate the complexity of computing a partial derivative as $\mathcal{O}(dnr^2)$ in the TT-format and $\mathcal{O}(n^d + dn)$ in the uncompressed format. A naive implementation of the gradient then yields another factor d in the complexity estimation. However, in the TT-format the naive implementation computes many redundant contractions, which means that we can save some complexity here. For this, we define the contractions

$$\begin{aligned} \mathbf{\Psi}_i^+(x_{i+1}, \dots, x_d) &= \mathbf{U}_{i+1} \circ \cdots \circ \mathbf{U}_d \circ \mathbf{\Phi}_d(x_d) \cdots \circ \mathbf{\Phi}_{i+1}(x_{i+1}) \\ \mathbf{\Psi}_i^-(x_1, \dots, x_{i-1}) &= \mathbf{U}_1 \circ \cdots \circ \mathbf{U}_{i-1} \circ \mathbf{\Phi}_{i-1}(x_{i-1}) \cdots \circ \mathbf{\Phi}_1(x_1). \end{aligned}$$

Note that $\mathbf{\Psi}_i^+$ is the contraction of every core with larger index than i, while $\mathbf{\Psi}_i^-$ is the contraction of every core with smaller index than i. We observe that $\mathbf{\Psi}_i^+(x_{i+1} \dots, x_d) \in \mathbb{R}^r$, $\mathbf{\Psi}_i^-(x_1 \dots, x_{i-1}) \in \mathbb{R}^r$ and that

$$\frac{\partial}{\partial x_i} v(x_1, \dots, x_d) = \mathbf{\Psi}_i^-(x_1, \dots, x_{i-1}) \circ \mathbf{U}_i \circ \mathbf{\Psi}_i^+(x_{i+1}, \dots, x_d) \circ \mathbf{\Phi}_i'(x_i). \tag{4.11}$$

Finally, omitting the arguments of the functions, the recursive properties

$$\mathbf{\Psi}_i^+ = \mathbf{U}_{i+1} \circ \mathbf{\Psi}_{i+1}^+ \circ \mathbf{\Phi}_{i+1}, \quad \mathbf{\Psi}_i^- = \mathbf{\Phi}_{i-1} \circ (\mathbf{\Psi}_{i-1}^- \circ \mathbf{U}_{i-1}) \tag{4.12}$$

yields Algorithm 6.
Note that the computational complexity of (4.12) and (4.11) is $\mathcal{O}(nr^2)$. Thus, Algorithm 6 has computational complexity of $\mathcal{O}(dnr^2)$. This compares to the complexity of the naive implementation $\mathcal{O}(d^2nr^2)$ and to the complexity of the full tensor formulation $\mathcal{O}(dn^d)$.

4.6.3 Computing the Frobenius norm and Related Norms

The Frobenius norm of a tensor can be computed using the component tensors. Assuming the core position $1 \leq l \leq d$, the Frobenius norm of the tensor $\mathbf{C}$ is equal the Frobenius norm of the component tensor $\mathbf{U}_l$, yielding an efficient norm computation cost of $\mathcal{O}(r^2n)$. This observation can also be used to compute the norm of the represented function in the function space via Parseval's identity. Assuming that the one-dimensional functions $\{\phi_1, \dots, \phi_n\}$,

Algorithm 6: Computing the gradient of a function v in TT-format.

input : A function v in TT-format, core tensors $\mathbf{U}_1, \dots, \mathbf{U}_d$, one-dimensional basis functions $\phi_1, \dots, \phi_n$, point $x = (x_1, \dots, x_d) \in \mathbb{R}^d$.

output : The gradient $\nabla v(x)$

for $\mu = 1, \dots, d-1$ **do**
  Calculate $\mathbf{\Psi}_i^-$ using the recursive formula (4.12).
end
for $\mu = d, \dots, 1$ **do**
  Calculate $\mathbf{\Psi}_i^+$ using the recursive formula (4.12).
end
for $\mu = d, \dots, 1$ **do**
  Calculate $\nabla v(x)[i] = \frac{\partial v}{\partial x_i}(x)$ by using (4.11).
end

form an orthonormal basis, the Frobenius norm of the coefficient tensor is exactly the norm of the represented function in the corresponding tensor product space. In particular for one-dimensional $H^k([a,b])$-orthonormal, $a < b$, the tensor-product space is given by so-called Sobolev spaces with dominating mixed smoothness $H^k_{\text{mix}}([a,b]^d)$. These spaces are equipped with the $\|\cdot\|_{H^k_{\text{mix}}([a,b]^d)}$-norm, which is given as follows

$$\|v\|_{H^k_{\text{mix}}([a,b]^d)} = \left(\sum_{i_1=0}^{k} \sum_{i_2=0}^{k} \cdots \sum_{i_d=0}^{k} \| \frac{\partial^{i_1}}{\partial_{x_1}} \frac{\partial^{i_2}}{\partial_{x_2}} \cdots \frac{\partial^{i_d}}{\partial_{x_d}} v(x_1, \dots, x_d) \|^2_{L^2([a,b]^d)} \right)^{\frac{1}{2}}. \tag{4.13}$$

Note that the mixed derivatives are of much larger degree for this function space. As we are considering a finite dimensional subspace of H^k_{mix}, it is a simple calculation using Fubini's theorem to validate the tensor-product structure of $\bigotimes_{k=1}^d \mathbf{span}\{\phi_1, \dots, \phi_n\}$ equipped with the $H^k_{\text{mix}}([a,b]^d)$-norm and that the orthonormal basis $\{\phi_1, \dots, \phi_n\}$ induces an orthonormal basis on the tensor-product space. For the special case $[a,b] = \mathbb{R}$ there is the theory of Sobolev spaces with dominating mixed smoothness, see e.g. the survey [Sch07] and references therein. In this case, the embedding $H^k_{\text{mix}}([a,b]^d) \hookrightarrow W^{k-1,\infty}([a,b]^d)$ holds for any $d > 0$ [SU09].

Remark 36. An orthonormal basis of one-dimensional functions in the $H^k([a,b])$ norm can be computed using the Gram-Schmidt process. In this thesis, we exclusively use one-dimensional polynomials obtained by the Gram-Schmidt process starting with monomials.

4.6.4 Handling Time-Dependent Functions

Up until now we have only considered functions $v : \mathbb{R}^d \to \mathbb{R}$. For several problems that we consider later, the sought functions are time-dependent, which means that $v : [0,T] \times \mathbb{R}^d \to \mathbb{R}$. We handle these occurrences by a time-discretization and linear interpolation. More precisely, we consider a time-discretization $0 = t_1 < \cdots < t_L = T$ with time-step $\tau > 0$ and then use linear interpolation between these time-points to obtain an approximation of the function values at any time-point.

$$v(t,x) \approx \frac{t_{l+1} - t}{\tau} v(t_l, x) + \frac{t - t_l}{\tau} v(t_{l+1}, x) \text{ for } t \in [t_l, t_{l+1}), x \in \Omega.$$

Note that this representation works nicely with the backwards iteration that is used in all our algorithms. One idea holding the promise of further compression would be treating the time-dependency as an extra dimension, which means that in this case essentially $v : \mathbb{R}^{d+1} \to \mathbb{R}$. However, this representation does not work nicely with the algorithms we use later, which is why we omit it. For a quadratic function, this representation was used in [BDS19] in a slightly different context.

4.7 Optimization on the Set of Tensor-Trains

We now come to the algorithm, which first made optimization on the set of tensor-trains possible, the *Alternating Linear Scheme* or *Alternating Least Squares* (ALS) algorithm. While there are algorithms using the manifold structure like Riemannian optimization [Ste16] which are mathematical more pleasing, the ALS impresses through simplicity and empirical success.

The underlying idea is exploiting the multi-linearity of the tensor-train representation. That is, while optimizing every core U_i at once yields a multi-linear problem, fixing every component but one yields a linear problem, that is also low-dimensional. In the ALS algorithm the cores are updated individually. By ordering the updates in an increasing or decreasing manner, this can be done efficiently. We give an informal algorithm stating the basic idea in Algorithm 7.

Algorithm 7: Alternating Least Squares (ALS)

input : An order d tensor $\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d = \mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$
output : Optimized component tensors $\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d = \mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$.
while *not converged* **do**
  for $i = 1$ *to* d **do**
    Move core position to i.
    Optimize $\mathbf{U}_i$.
  end
end

In the following we call a single iteration over the `for`-loop within the ALS algorithm a sweep and a single step within the `for`-loop a micro-step.

We continue with a detailed consideration of regression-type optimizations on the set of tensor-trains. To this end, we consider a loss functional of the type $\mathcal{R}_N(v) = \sum_{i=1}^N |v(x_i) - r_i|^2$, which is a discrete version of (3.1). We first consider the problem on a linear ansatz space and then come to the multi-linear ansatz space of tensor-trains. Assume that $\mathcal{U} \subset L^2(\Omega)$ linear and we want to find $\operatorname{arg\,min}_{v \in \mathcal{U}} \mathcal{R}_N(v)$. Given a basis $\{b_1, \dots, b_M\}$ of $\mathcal{U}$ and representing $v(x) = \sum_{j=1}^M c_j b_j(x)$ the optimization problem can be transformed onto the coefficient space. Deriving this problem and using the convexity of the loss functional we can validate that the optimal coefficients are given as the solution to the linear problem

$$A^T A c = A^T r, \tag{4.14}$$

where $A = [a_{ij}] \in \mathbb{R}^{N,M}$, $a_{ij} = b_i(x_j)$ and $r \in \mathbb{R}^N$ with entries r_i. Note that this basis can suffer from the curse of dimensionality and thus become extremely large. We now go to the space of tensor-trains and identify local basis functions of low-dimensional sub-problems to solve the regression problem. To this end, consider the set of tensor-trains of rank $\mathbf{r}$, denoted by $\mathcal{M}_\mathbf{r}$ and the optimization problem $\operatorname{arg\,min}_{v \in \mathcal{M}_\mathbf{r}} \mathcal{R}_N(v)$. Due to the multi-linearity of $\mathcal{M}_\mathbf{r}$

no linear problem can be identified and also no global basis. However, when fixing every component tensor except for one, we can identify a local basis. We first explicitly write down the basis using formulas and then give an intuition using the graphical notation.

Reordering the contractions within (4.10) and abbreviating $\mathbf{\Phi}_i(x_i) = \mathbf{\Phi}_i \in \mathbb{R}^{n_i}$, we obtain

$$v(x_1, \dots, x_d) = \sum_{i_l}^{n_l} \sum_{j_{l-1}, j_l}^{r_{l-1}, r_l} \mathbf{U}_j[j_{l-1}, i_l, j_l] \mathbf{\Phi}_l[i_l] \cdot$$

$$\Big(\sum_{i_1, \dots i_{l-1}}^{n_1, \dots, n_{l-1}} \sum_{j_1, \dots, j_{l-2}}^{r_1, \dots, r_{l-2}} \mathbf{U}_1[i_1, j_1] \dots \mathbf{U}_{l-1}[j_{l-2}, i_{l-1}, j_{l-1}] \mathbf{\Phi}_1[i_1] \dots \mathbf{\Phi}_{l-1}[i_{l-1}] \tag{4.15}$$

$$\sum_{i_{l+1}, \dots i_d}^{n_{l+1}, \dots n_d} \sum_{j_{l+1}, \dots, j_{d-1}}^{r_{l+1}, \dots, r_{d-1}} \mathbf{U}_{l+1}[j_l, i_{l+1}, j_{l+1}] \dots \mathbf{U}_d[j_{d-1}, i_d] \mathbf{\Phi}_{l+1}[i_{l+1}] \dots \mathbf{\Phi}_d[i_d] \Big). \tag{4.16}$$

For fixed component tensors $\mathbf{U}_1, \dots, \mathbf{U}_{l-1}$ we observe that (4.15) encodes a set of functions depending on $x_{i<l} := (x_1, \dots, x_{l-1})$, which we denote by $\tilde{b}^L(x_{i<l})$, i.e. using the abbreviation $\mathbf{\Phi}_i(x_i) = \mathbf{\Phi}_i \in \mathbb{R}^{n_i}$

$$\tilde{b}_l^L(x_{i<l}) = \sum_{i_1, \dots i_{l-1}}^{n_1, \dots, n_{l-1}} \sum_{j_1, \dots, j_{l-2}}^{r_1, \dots, r_{l-2}} \mathbf{U}_1[i_1, j_1] \dots \mathbf{U}_{l-1}[j_{l-2}, i_{l-1}, j_{l-1}] \mathbf{\Phi}_1[i_1] \dots \mathbf{\Phi}_{l-1}[i_{l-1}] \in \mathbb{R}^{r_{l-1}}.$$

Similarly, we identify (4.16) by a set of functions in $d - l - 1$ variables, denoted by $\tilde{b}_l^R(x_{i>l})$, where again $x_{i>l} = [x_{l+1}, \dots, x_d] \in \mathbb{R}^{d-l-1}$. This allows us to write

$$v(x_1, \dots, x_d) = \sum_{i_l}^{n_l} \sum_{j_{l-1}, j_l}^{r_{l-1}, r_l} \mathbf{U}_l[j_{l-1}, i_l, j_l] \big(\mathbf{\Phi}_l(x_l)\big)[i_l] \big(\tilde{b}_l^L(x_{i<l})\big)[j_{l-1}] \big(\tilde{b}_l^R(x_{i>l})\big)[j_l]$$

and to observe that the indices of $\mathbf{U}_l$ correspond to a linear parameterization of a linear function space. We see that $\tilde{b}_l^L$ maps from $\mathbb{R}^{l-1} \to \mathbb{R}^{r_{l-1}}$ and spans a function space of dimension r_{l-1}, where every entry $\tilde{b}_l^L(\cdot)_k$, $1 \leq k \leq r_{l-1}$ is an $l-1$-dimensional function. In the same sense $\mathbf{\Phi}_l$ maps from $\mathbb{R} \to \mathbb{R}^{n_l}$ and $\tilde{b}_l^R(x_{i>l})$ maps from $\mathbb{R}^{d-l} \to \mathbb{R}^{r_l}$. Taking the tensor product of these function spaces, yields basis functions from $\mathbb{R}^d \to \mathbb{R}$. This tensor-product space is of dimension $M_l := r_{l-1} \cdot n_l \cdot r_l$ and is exactly the low-dimensional, linear space that we seek. Finding optimal coefficients w.r.t. this function space yields an optimal coefficient tensor.

Assuming a four-dimensional space, i.e. $d = 4$, the corresponding pictorial diagram is given in Figure 4.6.

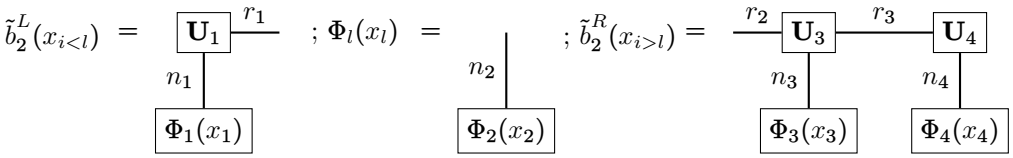


Figure 4.6: Graphical representation of the local basis functions for the case $l = 2$.

We formalize this procedure in the following lemma.

Lemma 4.7.1. *Let $\mathbf{U}_1 \circ \mathbf{U}_2 \circ \cdots \circ \mathbf{U}_d$ be a tensor-train representation of $\mathbf{C} \in \mathbb{R}^{n_1,\ldots,n_d}$ with minimal rank and core position $1 \leq l \leq d$. Then, the local basis functions $\tilde{b}_l^L(\cdot)_j$ and $\tilde{b}_l^R(\cdot)_j$, if defined, form an orthogonal set. Solving the regression problem over the tensor product of these bases yields optimal coefficients for the coefficient tensor $\mathbf{U}_l$.*

Proof. We begin with the orthogonality of $\{\tilde{b}_l^L(\cdot)_j, \mathbf{1} \leq j \leq r_l\}$ and proceed by induction. For $l = \mathbf{1}$, $\tilde{b}_l^L$ is not defined. We thus start with $l = \mathbf{2}$. As $\mathbf{U}_1$ is a (left-orthogonal) matrix, the orthogonality of the basis $\{\phi_1, \ldots, \phi_{n_1}\}$ is inherited to $\tilde{b}_2^L$. For the case $l > \mathbf{2}$, the tensor-product basis $\tilde{b}_{l-1}^L \otimes \mathbf{\Phi}_l$ is orthogonal as well. This, combined with the left-orthogonality of the component tensor $\mathbf{U}_l$ again proves that $\tilde{b}_l^L$ is orthogonal. The proof that $\tilde{b}_l^R$ is orthogonal is analogous.

Finally, regarding the statement of (locally) optimal coefficients, we observe that under the condition that $\mathbf{U}_k$, $k \neq l$ are fixed, the mno-th entry of $\mathbf{U}_l$ exactly encodes the product of the m-th entry of $\hat{b}_l^L$ and ϕ_n and the o-th entry of $\hat{b}_l^R$. □

Remark 37. In the above lemma the restriction 'if defined' is necessary, because for $l = \mathbf{1}$, we cannot define b_1^L. In this case, the local basis functions only consist of $\mathbf{\Phi}_1 \otimes b_1^R$. As the first component tensor is only of order two, this does not pose a problem. We treat the case $l = d$, where b_d^R cannot be defined, analogously.

Note that the orthogonality of the local basis functions is only fulfilled if the core position of the TT is exactly the core that is being optimized. Thus, moving the core position within the alternating optimization is a standard procedure in the literature. Moreover, due to the recursive nature of the local basis functions, i.e.

$$\tilde{b}_l^L(x_{i<l}) = \phi(x_{l-1}) \circ (\tilde{b}_{l-1}^L(x_{i<l-1}) \circ \mathbf{U}_{l-1}) \tag{4.17}$$

$$\tilde{b}_l^R(x_{i>l}) = \phi(x_{l+1}) \circ (\tilde{b}_{l+1}^R(x_{i>l+1}) \circ \mathbf{U}_{l+1}) \tag{4.18}$$

many contractions can be saved by computing so-called stacks and then optimizing the component tensors from left-to-right. Due to the nature of the regression problem, it is not necessary to encode the local basis functions directly. Instead, we need the local basis functions evaluated at the sample points x^i, where we can again use the recursive property (4.17). Finally, noticing that the core position of the TT can be moved by a simple QR-decomposition, we can give a more detailed ALS algorithm in Algorithm 8.

Note that the perspective with the local basis functions for a regression problem is somewhat non-standard in the literature. Usually, the ALS algorithm is considered for a linear equation of the type

$$\mathcal{R} : \mathcal{M}_{\leq \mathbf{r}} \to \mathbb{R}, \quad \mathbf{A} \mapsto \|\mathcal{L}\mathbf{A} - \mathbf{B}\|_F^2,$$

where $\mathcal{L}$ is a linear operator and $\mathbf{A}, \mathbf{B}$ are tensors. Note that our regression problem can be reformulated as the operator equation above. For a presentation of the ALS algorithm with this perspective we refer the reader to [HRS12a; Wol19].

Algorithm 8: detailed Alternating Least Squares (ALS)

input : An order d tensor $\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d = \mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$, sample points $x^i \in \mathbb{R}^d$ and data $r^i \in \mathbb{R}$

output : Optimized component tensors $\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d = \mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$.

while *not converged* **do**

  for $l = d-1$ *to* 1 **do**

    // build stacks

    Move core position to l.

    Compute (and store) $\tilde{b}_l^R(x^i)$ for all i using (4.18).

  end

  for $l = 1$ *to* d **do**

    // optimize cores

    Move core position to l.

    Compute (and store) $\tilde{b}_l^L(x^i)$ for all i using (4.17).

    Solve the local regression problem for the local basis $\tilde{b}_l^L \otimes \phi_l \otimes \tilde{b}_l^R$. The solution $\mathbf{c} \in \mathbb{R}^{M_l}$ is the component tensor $\mathbf{U}_l \in \mathbb{R}^{r_{l-1},n_l,r_l}$ unfolded into a vector.

  end

end

4.7.1 Adding Single Functions to the Ansatz Space

In some cases it is advantageous to include other functions into the ansatz space as well. This can be either functions that do not have tensor product structure, or simple functions that can be expected to have large impact in the representation of a function. In Section 5.4 we represent the solution to parabolic PDEs as a tensor-train. For this case we have a final condition, which means that $v(T,\cdot) = g(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is known. As v at time point T can be represented using the function g, it can be in some cases advantageous to enrich the ansatz space by exactly this function. The representation of the function is then of the form as in Figure 4.7.

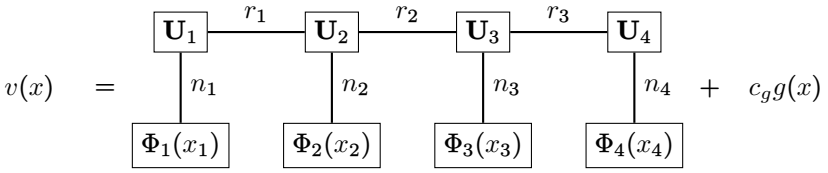


Figure 4.7: A 4-dimensional function and a TT-representation of its coefficient tensor.

Finding optimal coefficients c_g and optimal coefficient tensors does not get noticeably more challenging. Within the micro-steps in Algorithm 8 we simply find the optimal c_g as well, which means that the set of local basis functions gets enlarged by a single function. We use this extension not only in Section 5.4 but also in [Bay+21] for pricing of American options.

4.8 Regularization of Regression-Type Problems

In this section, we combine the results from Section 4.6.3 and Section 4.7 to obtain an efficient regularization. To this end, we consider a (discrete) loss functional $\mathcal{R}_N : \mathcal{M} \to \mathbb{R}_+$ as before. We modify the loss functional by adding a regularization term $r : \mathcal{M} \to \mathbb{R}_+$

$$\mathcal{R}_N(v) + r(v).$$

In our numerical examples this regularization term is the H^k_{mix} norm, cf. (4.13), which means that $r(v) = c_r \|v\|^2_{H^k_{\text{mix}}}$ with $c_r > 0$. If c_r is a constant, we call the strategy *constant regularization*. If c_r is chosen adaptively, we call the strategy *adaptive regularization*. We realize this adaptive regularization strategy within the ALS by measuring the residual $\text{res} = R_N(v) + r(v)$ at the beginning of one (ALS-) sweep and then setting $c_r = c \cdot \text{res}$ for this sweep. After completing the sweep we update the regularization parameter again. Using the adaptive regularization strategy, the effect of the regularization gets reduced within the ALS iteration. Note that again, using the appropriate basis of one-dimensional polynomials yields an isometry between the H^k_{mix} norm and the Frobenius norm of the coefficient tensor. Assuming the core position l of the tensor-train, the isometry between the tensor and the component tensor, cf. Section 4.6.3 allows for efficient implementation within the ALS algorithm.

It is possible to incorporate additional regularization or penalty terms. In Section 5.1 we enforce $v(\mathbf{0}) = 0$. This is done by first recalling the linear problem $\|Ac - r\|_F^2$, where $A = [a_{ij}] \in \mathbb{R}^{N,M}$, $a_{ij} = b_i(x_j)$ and $r \in \mathbb{R}^N$ with entries r_i, see (4.14) for its solution, and then replacing it by

$$\|Ac - r\|^2 + \delta \sum_{j=1}^{M} (c_j b_j(\mathbf{0}))^2$$

with $\delta > 0$. In a similar fashion we can enforce $\nabla v(\mathbf{0}) = 0$. In Section 5.4 we use a regularizer to enable rank-adaptive solution of the underlying regression problem. This regularizer as well as the rank-adaptive approach is covered in detail in Section 4.8.1.

The direct correspondence of norms in function spaces with the Frobenius norm of the coefficient tensor and thus also of the component tensor yields a way of regularizing with spectacular efficiency. We again state that in [Tru21] advanced ways of regularizing yield sample efficient algorithms. Note that using a way to regularize the coefficients in the context of deep neural networks hold the promise of similar increases in performance and finding a way to efficiently regularize (in function space) is of interest for future research.

4.8.1 Rank-adaptive Regression Solvers and the SALSA Algorithm

The relationship of the TT representation with the SVD allows for rank-adaptive strategies when solving the equations. This is particularly of interest because while for many problems a low-rank structure can be imposed, the exact rank is not known beforehand.

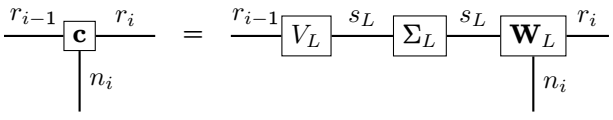
A rank-adaptive strategy has to answer how to adapt the rank and when to adapt the rank. The first rank-adaptive algorithm (in the context of TTs) was the *DMRG* algorithm introduced in [Whi92]. It modifies the ALS algorithm and relies on combining two component tensors, finding the optimal component, and then splitting it again via a SVD. The presence of the SVD answers both questions at once. By finding two component tensors at once the

ranks are adapted whenever large singular values are found via the SVD of the solution tensor. This method proves to be successful and is considered the state of the art algorithm for a long time. However, the ranks may increase rapidly, which might be unfavorable, especially in a regression setting where the problem of overfitting is present.

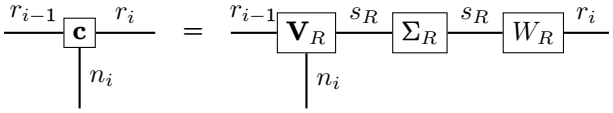
Recently, another rank-adaptive strategy, relying on a regularization technique was introduced in [GK19]. This algorithm is called *stable ALS approximation (SALSA)* and we use it in Section 5.4. In the following we give an overview of the main ideas behind the algorithm.

Since adding rank-1 tensors to a TT increases the rank by one (almost surely), a candidate for a rank-adaptive algorithm is finding a criterion of when to increase the ranks and then adding a rank-1 tensor to increase the ranks. In [GK19] it was shown that for the simple ALS algorithm this strategy can lead to instabilities: adding such small perturbations can lead to large deviations in the solution. This behavior is formalized in a concept of stability, which we do not explain in detail and instead refer to [GK19, Definition 1]. To circumvent such instabilities, the local regression problem, e.g. the micro-step within in the ALS (Algorithm 8), is exchanged by a regularized one.

We now cover the regularizer in detail. We assume that the core position of the tensor train is i and denote the corresponding component tensor as $\mathbf{c} \in \mathbb{R}^{r_{i-1},n_i,r_i}$. While the corresponding linear regression problem can be solved as in (4.14) with the coefficient tensor interpreted as a vector, the SALSA algorithm keeps the interpretation of the tensor $\mathbf{c}$ and regularizes certain singular values. These are the singular values of matricizations of the coefficient tensor. We represent them in Figure 4.8 where the matrices $\mathbf{\Sigma}_L$ and $\mathbf{\Sigma}_R$ are introduced.



(a) Left singular values.



(b) Right singular values.

Figure 4.8: Graphical notation if simple tensors and tensor networks.

For a given TT-representation, the matrices $\mathbf{\Sigma}_L$ and $\mathbf{\Sigma}_R$ are exactly the singular values that are also obtained by the TT SVD (Algorithm 5). The orthogonal matrices V_L and U_R are contracted onto the corresponding component tensors left or right from the current component tensor. Note that this does not alter their orthogonality property. We add a regularizer that is proportional to the inverse of the singular values. For singular values $\to 0$ this becomes ill-posed, which is why we cap the regularizer with a threshold $\sigma_{\min} > 0$. This is realized by defining the diagonal matrices with an entry-wise maximum

$$\tilde{\Sigma}_L = \mathbf{diag}(\mathbf{max}\{\Sigma_L[1,1], \sigma_{\min}\}, \ldots, \mathbf{max}\{\Sigma_L[s_R, s_R], \sigma_{\min}\},$$

i.e. singular values smaller than $\sigma_{\min}$ are replaced by $\sigma_{\min}$. The loss functional is then modified

by adding

$$\tilde{\mathcal{R}}_N(v) := \mathcal{R}_N(v) + r(v) + \lambda(\|\tilde{\Sigma}_L^{-1} \circ \mathbf{c}\|_F^2 + \|\mathbf{c} \circ \tilde{\Sigma}_R^{-1}\|_F^2), \tag{4.19}$$

where v is the function representation with corresponding component tensor $\mathbf{c}$. Note that in the micro-step every other component tensor is fixed and a linear problem is remaining, which means that there is a one-to-one correspondence from v to $\mathbf{c}$ and the formulation above is well-defined. As proven in [GK19] this adaption of the micro-step yields a method where the ranks can safely be increased by adding a rank-$\mathbf{1}$ TT. In order to make the method rank-adaptive we need a strategy to increase the ranks. This is done by separating the singular values σ_L and σ_R into two parts. Those larger than $\sigma_{\min}$ are called stabilized. The others are called minor. During the alternating optimization we keep a constant number $r_{\mathbf{min}} \in \mathbb{N}$ of minor singular values and if a singular value becomes larger than the barrier $\sigma_{\min}$, we increase the rank by one. During the iteration, the threshold $\sigma_{\min}$ is decreased adaptively. In our implementation we use the parameters proposed in [GK19, Section 9.4]. We finish this chapter by stating the SALSA algorithm (Algorithm 9).

Algorithm 9: stable Alternating Least Squares algorithm(SALSA)

input : An initial order d tensor $\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d = \mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$ encoding a function $v : \mathbb{R}^d \to \mathbb{R}$, sample points $x^i \in \mathbb{R}^d$, data $r^i \in \mathbb{R}$, positive parameters $\varepsilon_{\text{start}}$, $s_{\min}$, λ_{start}, and $\lambda_{\min}$, and $r_{\min} \in \mathbb{N}$.

output : Optimized component tensors $\mathbf{U}_1 \circ \cdots \circ \mathbf{U}_d = \mathbf{A} \in \mathbb{R}^{n_1,\dots,n_d}$.

Set $\lambda = \lambda_{\text{start}}$, $\sigma_{\min} = \sigma_{\text{start}}$

while *not converged* **do**

  for $l = d-1$ *to* 1 **do**

    `// build stacks`

    Move core position to l.

    Compute (and store) $\tilde{b}_l^R(x^i)$ for all i using (4.18).

  end

  for $l = 1$ *to* d **do**

    `// optimize cores`

    Move core position to l.

    Compute (and store) $\tilde{b}_l^L(x^i)$ for all i using (4.17).

    Solve the local regularized regression problem (4.19) for the local basis $\tilde{b}_l^L \otimes \phi_l \otimes \tilde{b}_l^R$.

    Compute the singular vector matrices Σ_L and Σ_R, cf. Figure 4.8, and denote by σ_L the $r_{\min}$-th smallest singular value in Σ_l and by σ_R the $r_{\min}$-th smallest singular value in Σ_R.

    `// If largest minor singular value is larger than barrier adapt ranks`

    if $\sigma_L > \sigma_{\min}$ **then**

      increase $r_{\mu-1}$

    end

    if $\sigma_R > \sigma_{\min}$ **then**

      increase r_μ

    end

  end

  `// update SALSA parameters`

  Set res $= \tilde{\mathcal{R}}_N(v)$, where v is the function representative of the TT.

  Update $\lambda = \min\{\text{res}, \lambda/\lambda_{\min}\}$ and $\sigma_{\min} = s_{\min} \cdot \text{res}$

end

Chapter 5

Numerical Experiments

In this chapter, we present the heart of our work, various numerical tests. For different control and control related problems we discuss the governing equations, the adaption of our base algorithm and difficulties for the theoretical part. The content of every section is covered in different publications and preprints, which we state in the beginning of every section.

In the implementation we used several libraries, where we list the most important below. For the implementation of the tensor networks we use the library `xerus` [HW17]. We also make use of the python packages `Numpy` [VCV11; Oli06], `SciPy` [Vir+20] for various linear algebra operations and algorithms, and `Matplotlib` [Hun07] for visualization of our results.

We start with the infinite horizon case, because it is already covered in Chapter 2. In the subsequent section we consider the finite horizon case. For this case the additional difficulty of time-dependence of the value function appears. We are able to prove a theorem on the error propagation of the value function for this case. The third control problem we consider is a stochastic exit-time control problem where the obvious additional difficulty is the stochastic nature of the differential equation. Finally, we consider a solution method for parabolic PDEs with a non-standard solution approach - backward stochastic differential equations. Note that in this chapter we have a slight abuse of notation. In the following sections, c is a function within the cost functional. However, c is sometimes a coefficient tensor in accordance to the notation from previous chapters. Distinguishing both is possible from the context.

In every section we give a short conclusion with statements on possible extensions of the approach. Note that in the end of the thesis there is another conclusion and outlook, where information valid for every approach is gathered. In order to make the individual sections as self-contained as possible, we keep some duplicate remarks.

5.1 The Infinite Horizon Case

Most of the content of this section is previously published in [OSS21b]. In the infinite horizon case we consider an autonomous system of ODEs on an infinite time horizon as in Chapter 2.

Optimal Control Problem 4. Set $\gamma \geq 0$. For $x \in \mathbb{R}^d$ minimize w.r.t. $u \in L^2(0, \infty; \mathbb{R}^m)$

the cost functional $\mathcal{J} : \mathbb{R}^d \times L^2(0,\infty;\mathbb{R}^m) \to \mathbb{R} \cup \infty$, defined as,

$$\mathcal{J}(x, u(\cdot)) := \int_0^\infty e^{-\gamma t}(c(y(t)) + u(t)'Ru(t))\,\mathrm{d}t,$$

where

$$\begin{aligned} \dot{y}(t) &= f(y(t)) + g(y(t))u(t) \\ y(0) &= x. \end{aligned} \tag{5.1}$$

If either the trajectory $y(t)$ does not exist for every $t \in [0,T]$ or the integral within the cost functional does not converge, we set $\mathcal{J}(x,u) = \infty$.

We assume that $c : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ is non-negative, coercive, $C^1(\mathbb{R}^d)$ with $0_{\mathbb{R}^d}$ being the only root of c. Further, let $R \in \mathbb{R}^{m,m}$ positive definite. For initial data $x \in \mathbb{R}^d$ and fixed control $u(\cdot) \in \mathcal{A}$, we denote by $y_x(t,u) \in \mathbb{R}^d$ the evaluation of the trajectory at time t. If the context is clear we just write $y(t)$. We further assume that $f : \mathbb{R}^d \to \mathbb{R}^d$ and $g : \mathbb{R}^d \to \mathbb{R}^{d,m}$ are (non-linear) $C^1(\mathbb{R}^d)$, Lipschitz functions and that $f(0) = 0$. We write $\ell(x,u) = c(x) + u'Ru$. We define the value function as

$$v^* : \mathbb{R}^d \to \mathbb{R}, \quad (t,x) \mapsto \inf_{u \in L^2(0,\infty;\mathbb{R}^m)} \mathcal{J}(x,u)$$

and the corresponding Bellman equation takes the following form.

Theorem 5.1.1 ([BC97])**.** *For all $x \in \mathbb{R}^d$ and $\tau > 0$ we have*

$$v^*(x) = \inf_{u \in L^2(0,\tau;\mathbb{R}^m)} \Big[\int_0^\tau \ell(y(t),u(t))\,\mathrm{d}t + e^{-\gamma\tau} v^*(y(\tau)) \Big],$$

where $y(\cdot)$ is the trajectory corresponding to (5.1) *with control u and initial value $y(0) = x$.*

We further have the following HJB equation.

Theorem 5.1.2 ([BC97; Da 00])**.** *Write $h(x,u) = f(x) + g(x)u$. Assume that there exists $L > 0$ such that*

$$(h(x,u) - h(y,u), x - y) \leq L|x-y|^2$$

for all $x, y \in \mathbb{R}^d$ and $u \in \mathbb{R}^m$ and that v^* is continuous. Then, the value function $v^*(x)$ is a viscosity solution of

$$\gamma v^*(x) + \sup_{u \in \mathbb{R}^m} \Big[- Dv^*(x) \cdot (f(x) + g(x)u) - \ell(x,u) \Big] = 0$$

with $v(0) = 0$.

Theorem 5.1.3. *For Optimal Control Problem 4 an optimal feedback control is given by*

$$\alpha^*(x) = -\frac{1}{2} R^{-1} g(x)' \nabla v^*(x),$$

if ∇v^* exists.

We denote the closed-loop system as

$$\dot{y} = f(y) + g(y)\alpha(y), \quad y(0) = x \tag{5.2}$$

for any admissible policy $\alpha : \mathbb{R}^d \to \mathbb{R}^m$.

Proposition 5.1.4. *Let $\alpha : \mathbb{R}^d \to \mathbb{R}^m$ be Lipschitz and g be constant. Then the flow Φ_τ^α,*

$$\Phi_\tau^\alpha(x) = y_x^\alpha(\tau),$$

of the dynamical system (5.2) *is well defined and Lipschitz continuous.*

The proof is a simple application of the Picard–Lindelöf theorem. We note that any other condition for ensuring that the flow exists is sufficient for the rest of the section.

5.1.1 Setup and Adaption of the Algorithm

We assume that there exists a compact $\Omega \subset \mathbb{R}^d$ that we want to approximate the value function in. Let $\alpha : \Omega \to \mathbb{R}^m$ be a feasible feedback control such that the flow does not leave Ω with finite cost. Then, the flow $\Phi_\tau^\alpha : \Omega \to \Omega$ is well-defined. We abbreviate $\ell^\alpha(x) = \ell(x, \alpha(x))$. We then also have that the Koopman operator, cf. Definition 19, is well-defined as a linear operator $K_\tau^\alpha : L^\infty(\Omega) \to L^\infty(\Omega)$ with norm $\|K_\tau^\alpha\|_{\mathcal{L}(L^\infty(\Omega))} = 1$. Thus, if $\gamma > 0$, the operator equation

$$[(I - e^{-\gamma\tau} K_\tau^\alpha)v](x) = \int_0^\tau \ell^\alpha(\Phi_t^\alpha(x))\,\mathrm{d}t, \tag{5.3}$$

has a unique solution in $L^\infty(\Omega)$. Note that this equation is exactly the linearized Bellman equation from Theorem 2.4.2 and that the r.h.s. is indeed in $L^\infty(\Omega)$ because α has finite cost. Hence, we make use of the Policy Iteration algorithm, cf. Algorithm 2 and compute v^α by solving the linearized Bellman equation (5.3). Note that in the case $\gamma = 0$, we cannot prove that the linearized Bellman equation has a unique solution. However, it is still possible to verify that v^α is a solution to the linearized Bellman equation. In order to apply the approximative Policy Iteration, cf. Algorithm 3, we define the loss functional

$$\mathcal{R}^\alpha(v) = \|(I - e^{-\gamma\tau} K_\tau^\alpha)v - \int_0^\tau \ell^\alpha(\Phi_t^\alpha(\cdot))\,\mathrm{d}t\|_{L^2(\Omega)}^2 \tag{5.4}$$

and its discrete version

$$\mathcal{R}_N^\alpha(v) = \frac{1}{N}\sum_{i=1}^N |(I - e^{-\gamma\tau} K_\tau^\alpha)v(x_i) - \int_0^\tau \ell^\alpha(\Phi_t^\alpha(x_i))\,\mathrm{d}t|^2, \tag{5.5}$$

where $x_i \sim \mathcal{U}(\Omega)$ are uniformly distributed.

The structure of $\mathcal{R}_N^\alpha$ is in this case not of regression-type. Instead, it is a minimal residual formulation. However, this problem can be solved in a similar way. We briefly explain the case of a linear ansatz space and the generalization to the set of tensor-trains is then as in Section 4.7.

Given a basis $\{b_1, \dots, b_M\}$ of $\mathcal{U}$ and representing $v(x) = \sum_{j=1}^M c_j b_j(x)$ the optimization problem can be transformed onto the coefficient space. Deriving this problem and using the

convexity of the loss functional we can validate that the optimal coefficients are given as the solution to the linear problem

$$A^T A c = A^T r,$$

where $A = [a_{ij}] \in \mathbb{R}^{N,M}$, $a_{ij} = b_i(x_j) - e^{-\gamma\tau} b_i(\Phi_\tau^\alpha(x_j))$ and $r \in \mathbb{R}^N$ with entries $r_i = \int_0^\tau \ell^\alpha(\Phi_t^\alpha(x_i))$.

We finish this section be repeating the approximative Policy Iteration algorithm adapted to the present problem in Algorithm 10.

Algorithm 10: approximative Policy Iteration

input : A Policy α_0 such that $v^{\alpha_0}(x) < \infty$ for all $x \in \Omega$.

output : An approximation of v^* and α^*.

Set $k = 0$.

do

Find

$$v_k = \operatorname*{arg\,min}_{v \in \mathcal{M}} \mathcal{R}_N^\alpha(v)$$

then update the policy according to

$$\alpha_{k+1}(x) = -\frac{1}{2} R^{-1} g(x)' \nabla v_k(x),$$

and set $k := k + 1$.

while $\|\alpha_k - \alpha_{k+1}\| > tol$;

5.1.2 A Preconditioner of the Linearized Bellman Equation

We will now cover the choice of the length of the trajectories τ in the (linearized) Bellman equation. From the perspective of computational time it is of course helpful to have τ as small as possible and in theory v^α is a minimizer of the loss functional for every τ. However, calculating longer trajectories can be seen as a preconditioner of the linearized Bellman equation (5.3), as we show now.

To this end we consider the linear equation 5.3. Recall that $\|K_\tau^\alpha\|_{\mathcal{L}(L^\infty(\Omega))} = 1$, which means that for any discount factor $\gamma > 0$ the operator is a contraction and thus the inverse is given by the Neumann series

$$(\mathrm{Id} - e^{-\gamma\tau} K_\tau^\alpha)^{-1} = \sum_{i=0}^{\infty} (e^{-\gamma\tau} K_\tau^\alpha)^i.$$

Observing that for small discount factor this operator becomes arbitrarily bad conditioned, a good candidate for a preconditioner of (5.3) is $\sum_{i=0}^N (e^{-\gamma\tau} K_\tau^\alpha)^i$. Due to the semi-group property of the Koopman operator, we have

$$\begin{aligned}(\sum_{i=0}^{N} (e^{-\gamma\tau} K_\tau^\alpha)^i (\mathrm{Id} - K_\tau^\alpha v))(x) &= (\mathrm{Id} - e^{-\gamma\tau(N+1)} K_{(N+1)\tau}^\alpha) v(x) \\ &= v(x) - e^{-\gamma\tau(N+1)} v(\Phi_{(N+1)\tau}(x)).\end{aligned}$$

In the same manner we obtain

$$\sum_{i=0}^{N}((e^{-\gamma\tau}K_\tau^\alpha)^i\int_0^\tau e^{-\gamma t}r_t^\alpha(x)\,\mathrm{d}t=\sum_{i=0}^{N}e^{-\gamma\tau i}\int_{i\tau}^{(i+1)\tau}e^{-\gamma t}r_t^\alpha(x)\,\mathrm{d}t=\int_0^{(N+1)\tau}e^{-\gamma t}r_t^\alpha(x)\,\mathrm{d}t$$

From this equation we see that calculating longer trajectories can be seen as a preconditioner of the Bellman equation.

Remark 38. Note that due to the Neumann series, we can solve (5.3) point-wise by calculating trajectories with $\tau\to\infty$, which recovers the definition of v^α. In this case, the linearized Bellman equation reduces to a regression problem, which is of course perfectly conditioned. The numerics of this approach will, however, not be presented here.

5.1.3 Variations of the VMC equation

The formulation with a loss functional (5.4) allows for easy incorporation of additional loss terms. We introduce a variant of the loss functional by incorporating information about $g(x)'\nabla v$ into the loss functional, as described in Section 4.8. The idea behind this approach is that for computing the control we do not only need a good approximation on v but also of ∇v. In particular the directions $g(x)'\nabla v$ are of interest because of the policy update in the Policy Iteration, cf. (10) and Theorem 5.1.3. Thus, by incorporating information about this derivative we hope to obtain an improved controller. This will of course be tested in the experiments, where we have the special case $g(x)=g\in\mathbb{R}^{d,1}$, i.e. we only have one control parameter.

Denoting for a sample x_i by $\tilde{x}_i=x_i+\varepsilon g$, we modify $\mathcal{R}_N(v)$, cf. (5.5), by adding a discrete derivative.

$$\mathcal{R}_N^{H^1}(v)=\sum_{i=1}^{N}|\mathrm{Id}-K_\tau^\alpha v(x_i)-R(x_i)|^2+|\frac{(\mathrm{Id}-K_\tau^\alpha)(v(\tilde{x}_i)-v(x_i))-(R(\tilde{x}_i)-R(x_i))}{\varepsilon}|^2.$$

In our tests we use $\varepsilon=10^{-3}$. We add additional information into the loss functional by incorporating that $v(0)=0$ and $\nabla v(0)=0$. Moreover, we add the adaptive regularizer as described in Section 4.8 to the loss functionals

$$\begin{aligned}\tilde{\mathcal{R}}_N(v)&=\mathcal{R}_N(v)+\delta_1(v(0)^2+|\nabla v(0)|^2)+\delta_2\|v\|_{H^1_{\mathrm{mix}}},\\ \tilde{\mathcal{R}}_N^{H^1}(v)&=\mathcal{R}_N^{H^1}(v)+\delta_1(v(0)^2+|\nabla v(0)|^2)+\delta_2\|v\|_{H^1_{\mathrm{mix}}}.\end{aligned}\tag{5.6}$$

We specify δ_1 and δ_2 in the numerical tests. As polynomial ansatz spaces we use the tensor product of one-dimensional H^1-orthogonal [1] polynomials, such that the tensor-product space is H^1_{mix}, as described in Section 4.6.3 and 4.8. The performance of the two loss functional is compared in the following tests.

[1] In this thesis, we mostly use H^2 orthonormal polynomials. The usage of H^1 orthonormal polynomials for this example has historical reasons. By the time the numerical results were produced we still used H^1 orthonormal polynomials. Afterwards we noticed that H^2 orthonormal polynomials yield slight improvements in performance. We state that the difference between these sets is marginal.

5.1.4 Results

We present results of numerical tests for different optimal control problems. For the implementation of the tensor networks we use the open source `c++` library `xerus` [HW17]. The calculations were performed on a AMD Phenom II 6x 3.20GHz, 8 GB RAM openSUSE Leap 15.0 Linux distribution. In every test we consider a cost functional of the form

$$\underset{u\in L^2((0,\infty);\mathbb{R}^m)}{\arg\min} \mathcal{J}(x,u) = \int_0^\infty \|y(t)\|^2 + 0.1|u(t)|^2 \ \mathrm{d}t \tag{5.7}$$

and a PDE, which we denote here as

$$\dot{y} = f(y) + g(y)u, \quad y \in \Theta \subset L^2(-1,1).$$

As the first step we discretize the PDE in space, such that we obtain a finite dimensional system of ODEs, which we also denote as

$$\dot{y} = f(y) + g(y)u, \quad y \in \mathbb{R}^d.$$

The degree of the one-dimensional polynomials 4. We implement Algorithm 10 for the spatially discretized PDE using the loss functionals (5.6) with adaptive regularization. For the loss functional $\tilde{\mathcal{R}}_N$ we denote the corresponding value function approximation by v_{L^2} and the corresponding controller by α_{L^2}, and for the loss functional $\tilde{\mathcal{R}}_N^{H^1}$ we denote the value function approximation by v_{H^1} and the controller by α_{H^1}. For the the L^2 controller we use 32768 Sobol [Sob67] quasi Monte Carlo samples and for the H^1 controller we use 16384 Sobol quasi Monte Carlo samples.

We compare our controllers to the linear quadratic regulator (LQR), which is a closed-loop controller that is obtained by linearizing the systems around 0 and then solving the Riccati equation, cf. Section 2.9 for more information. We denote this controller by α_{LQR}. We also compute the optimal open loop control via a classical gradient descent method as described in Section 2.2. We denote this control by u_{opt} to stress its open loop nature. Here, we have to restrict ourselves to a finite time frame and choose $T = 5$ for computing the open loop control, because at this point the norm of the state was negligible. The run-time of our (non-optimized) code for the offline computations, i.e. the Policy Iteration, was approximately 10 hours.

Remark 39. We distinguish between the policy α, the corresponding cost estimator v and the real generated cost $\mathcal{J}(\cdot, \alpha(\cdot))$. For fixed x, we obtain $v(x)$ by simply evaluating v. Here, no trajectory has to be computed. We obtain $\mathcal{J}(x, \alpha(x))$ by numerically integrating along the trajectory with initial condition x. Note that $\mathcal{J}(x, \alpha(x))$ is basically the numerical approximation of the cost functional with respect to a feedback law, defined in (2.3).

5.1.4.1 Test 1: Diffusion with Unstable Reaction Term

We consider a diffusion equation with unstable reaction term and homogeneous Neumann boundary condition, cf. [KK18, Test 2]. Solve (5.7) for $y \in \Theta = L^2(-1,1)$ subject to

$$\begin{aligned} \dot{y} &= \sigma\Delta y + y^3 + \chi_\omega u \\ y(0) &= x \end{aligned}$$

with homogeneous Neumann boundary condition and the characteristic function χ_ω w.r.t. $\omega = [-0.4, 0.4] \subset [-1, 1]$. We choose $\sigma = 1$ and use a finite differences grid with $d \in \mathbb{N}$ grid points to discretize the spatial domain. We denote by A this finite difference discretization of the Laplace operator and by $G \in \{0, 1\}^d$ the discretization of the characteristic function χ_ω. Then, we obtain a system of d ordinary differential equations.

$$\begin{aligned} \dot{y} &= \sigma A y + y^3 + Gu \\ y(0) &= x \end{aligned}$$

Using the step-size $h = \frac{2}{d+1}$ we get a finite dimensional approximation of the term $\|y(t)\|_H^2$ in the cost functional. For this test we choose a spatial dimension of $d = 32$. As the underlying equation is non-linear, our ansatz for the value function is the tensor product of polynomials up to degree 4. The internal TT-rank chosen is

$$[3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 4, 3].$$

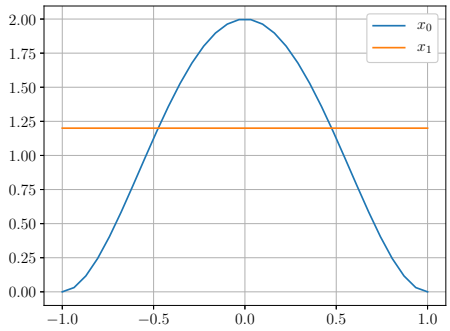
We state that while decreasing the ranks yields a performance loss, an increase did not alter the performance.

Remark 40. We stress that the number of quasi Monte Carlo samples is extremely small when comparing it to the dimension of the ambient space $32768 \ll 5^{32} \approx 10^{22}$ and only when comparing it to the degrees of freedom in the TT representation the numbers become comparable, $32768 \approx 6 \cdot 5395$. This further indicates, that the curse of dimensionality is broken by our ansatz.

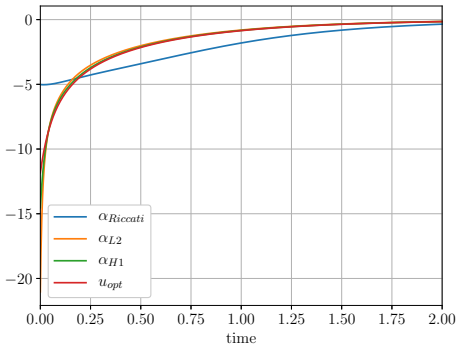
For every sample we calculate a trajectory with 1000 steps with step-size $\tau_{\text{ODE}} = 0.001$ using the classical Runge-Kutta 4 method. This yields $\tau = 1000 \cdot \tau_{\text{ODE}} = 1$ in the notation of the previous sections. For the adaptive regularization we use $\delta_1 = 100$ and $\delta_2 = 10^{-3}$, cf. (5.6). We report that when we ran the same algorithms with only 100 steps, the resulting controls were not better than the LQR controller. This shows the effect of the preconditioner, cf. Section 5.1.2. In both cases we stop after 100 policy updates, where the relative difference between value functions was 10^{-3}. Note that so many iterations were necessary, because we stopped the ALS algorithm after 20 ALS sweeps, where the solution was not exact.

We first test the feedback controllers for certain initial values, visualized in Figure 5.1. For both the α_{L^2} and the α_{H^1} controller, significant improvement in cost is noticeable, with the greatest being approximately 50% of the cost saved compared to the LQR controller. Moreover, we see that the computed feedback laws generate costs close to optimal costs for the tested initial values.

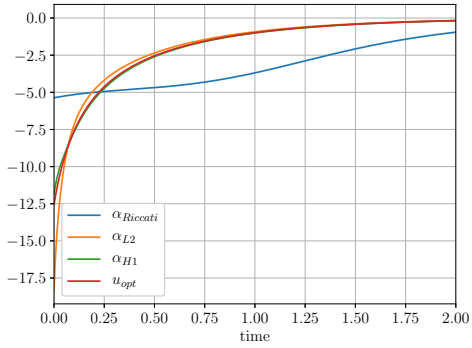
Next we test the feedback law for random initial values. Note that because of the diffusion, equally distributed samples and normally distributed samples yield low cost on average, even with the LQR controller, and in this case no improvements of the cost is to be expected. Thus, we use a special distribution of initial values that we specify now. For every initial value we choose an equally distributed integer between 2 and 20. This number is the degree of a random polynomial. Next we choose a polynomial with normal distributed coefficients of the degree we chose. As this polynomial $\tilde{p}$ has its maximum in the interval $[-1, 1]$ on the boundary with high probability, we modify the polynomial in the following way $p(x) := \tilde{p}(x)(x-1)(x+1)$,



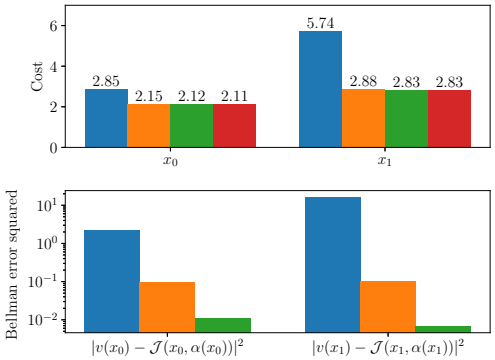
(a) Initial values.



(b) Generated controls, initial value x_0.



(c) Generated controls, initial value x_1.



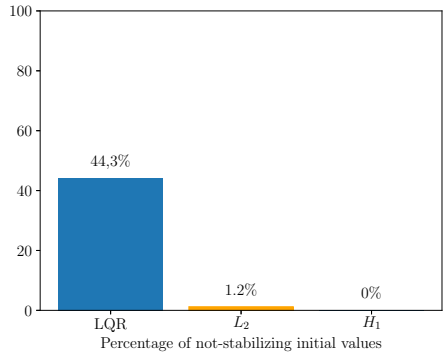
(d) Generated cost and Least-Squares error. Blue is v_{LQR}, orange is V_{L^2}, green is V_{H^1}, and red is the open-loop optimal control. $v(x_i)$ is the approximated value function evaluated at x_i and $\mathcal{J}(x_i, \alpha(x_i))$ is the actual cost, cf. Remark 39.

Figure 5.1: The generated controls for different initial values for the diffusion with unstable reaction term.

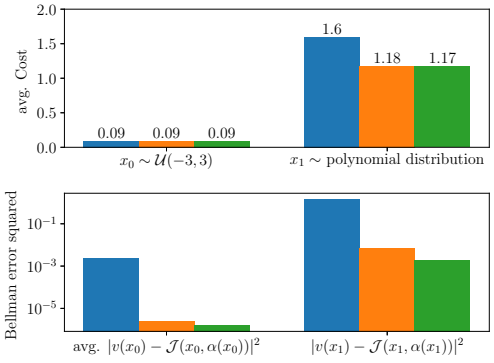
such that we have $p(-1) = p(1) = 0$. Note that these initial values do not obey the Neumann boundary condition. However, due to our discretization this does not pose a problem. Finally, we rescale p such that its maximum in $[-1, 1]$ is 1.75. In order to have an idea how these initial values look, we plotted 10 initial values in Figure 5.4a. We report that for these initial values the LQR controller was not stabilizing in 443 out of 1000 initial values, while the α_{L^2} controller was not stabilizing in 12 cases. The α_{H^1} controller was stabilizing for every initial value.

Remark 41. Note that from these initial values we can deduce that the area of attraction was increased by computing the α_{L^2} and α_{H^1} controller. This further underlines the importance of computing controllers for non-linear systems. However, other methods for non-linear systems, like Model Predictive Control, might also increase the area of attraction.

We compare the average cost only for the initial values where LQR was stabilizing and we observe that even for these initial values, we obtain an average cost reduction of more than 25%. This is visualized in Figure 5.2.



(a) Percentage of unstable initial values for different controllers, 1000 initial values $\sim$ polynomial distribution.



(b) Average cost for 1000 initial values. Blue is v_{LQR}, orange is v_{L_2} and green is v_{H_1}. Left $x \sim \mathcal{U}(-3, 3)$. Right:$x_1 \sim$ polynomial distribution, only the 557 values where the LQR controller was stabilizing.

Figure 5.2: The generated cost for random initial values for the diffusion with unstable reaction term.

5.1.4.2 Test 2: Viscous Burgers'-like Equation

As underlying equation we use a one-dimensional viscous Burgers'-like equation similar to [KK18, Test 1] with a destabilizing exponential term. Solve (5.7) for $y \in \Theta = L^2(-1, 1)$ subject to

$$\dot{y} = \sigma \Delta y + \nabla(\frac{y^2}{2}) + 1.5 y e^{-0.1y} + \chi_\omega u$$
$$y(0) = x$$

with homogeneous Dirichlet boundary condition. Here, we use the same discretization as in the previous section. The constants are the same except for $\sigma = 0.2,\ \omega = [-0.5, 0.2]$. We choose the same ranks as in the last test

$$[3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 6, 6, 6, 6, 6, 6, 6, 5, 5, 5, 4, 3]$$

and solve the HJB on $[-3, 3]^{32}$. We report that the choice of ranks is corresponds to the ranks of the LQR controller in TT-format, rounded with threshold 10^{-6}. We calculate v_{L^2} and v_{H^1} as before only change the step-size $\tau_{\text{ODE}} = 0.01$ and $\delta_2 = 10^{-6}$.

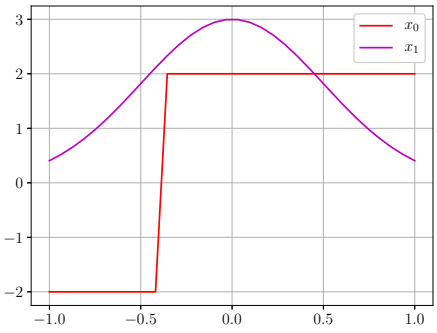
We report that we also attempted computing an optimal open-loop control for this test, but did not succeed with both constant step-size and with Armijos step-size control. Instead, we added a control computed by the Nelder-Mead method provided by the python package `SciPy` [Vir+20]. We also report that while this method did not converge as well, improvement w.r.t. the cost was achieved and thus we included the result.

From Figure 5.3 we deduce that for certain initial values, significant improvement of cost is possible for both loss functionals with the greatest improvement being 9.2% of the cost. In both cases the H^1 loss functional gave small performance improvements, while the resulting cost is close to the approximated optimal cost. We again notice that for these initial values our calculated value functions are more accurate than the Riccati value function.

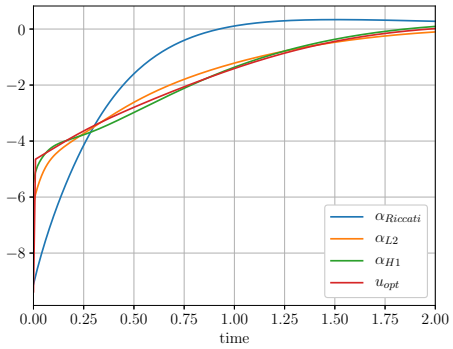
Next we again test random initial values. We are using the same setup as in the previous test. The normalization of the polynomial random initial values is changed to 2.75 instead of 1.75 because the integration area was changed. In Figure 5.4 we compare the performance of the controllers for 1000 random initial values. For $x \sim \mathcal{U}(-3, 3)$, no significant improvements in cost are visible, while the cost prediction for our value functions is more exact. The improvements are, however, not significant for these initial values. For x distributed in the way described above there is a visible difference. On average, 6% of the cost is saved by v_{L^2} and v_{H^1}. Moreover, the computed value functions predict their corresponding cost functional more exactly than the LQR value function.

5.1.5 Short Conclusion and Outlook

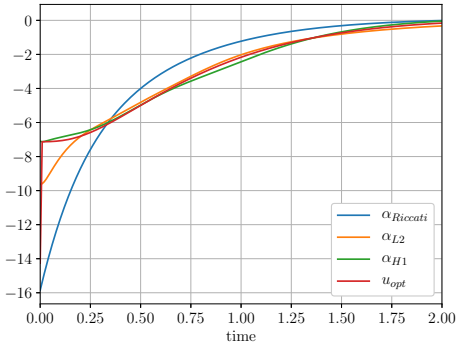
We have demonstrated the applicability of our tensor-train approach for state spaces in 32 dimensions. While the controllers are close to optimal, the length of the trajectories is concerning. In order to reduce the length of the trajectories an approach will be presented in the next section and discussed in the conclusion of the next section. Another potential approach would be using an iterative procedure to solve the linearized Bellman equation. This approach is used and presented in Section 5.3. Finally, an simple approach is computing trajectories until the norm of the state is negligibly small. This approach is shortly presented in Remark 38. Here, an evaluation of the value function of the corresponding LQR controller can be added as a final condition to further enhance the accuracy of the computations. This is motivated by the fact that the value function of the LQR controller is locally accurate around the steady state. Note that all these adaptions reduce the minimal residual problem to a regression problem.



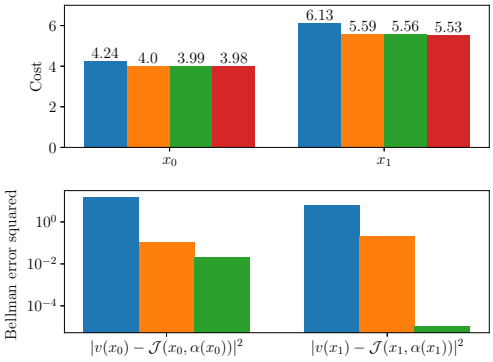
(a) Initial values.



(b) Generated controls, initial value x_0

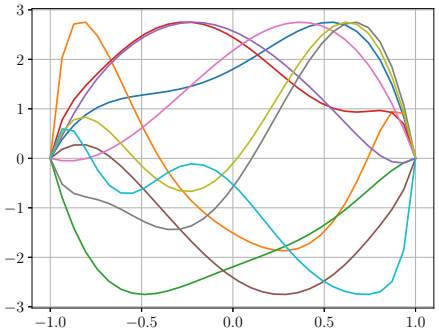


(c) Generated controls, initial value x_1.

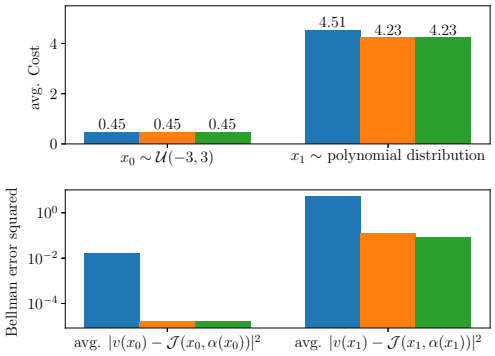


(d) Generated cost and Least-Squares error. Blue is Riccati, orange is V_{L^2}, green is V_{H^1} and red is the optimal open-loop control. $v(x_i)$ is the approximated value function evaluated at x_i and $\mathcal{J}(x_i, \alpha(x_i))$ is the actual cost, cf. Remark 39.

Figure 5.3: The generated controls for different initial values for the viscous Burgers'-like equation.



(a) Examples of 10 random initial values, drawn as described above, normalized such that the maximum is 2.75.



(b) Average cost for 1000 initial values. Blue is Riccati, orange is V_{L^2} and green is V_{H^1}. Left $x \sim \mathcal{U}(-3,3)$, right $x \sim$ polynomial distribution. $v(x_i)$ is the approximated value function evaluated at x_i and $\mathcal{J}(x_i, \alpha(x_i))$ is the actual cost, cf. Remark 39.

Figure 5.4: The generated cost for random initial values for the viscous Burgers'-like equation.

5.2 The Finite Horizon Case

The content of this section is previously published in [OSS21a]. We consider a finite horizon, deterministic problem. Note that this problem already appeared within the thesis in Section 2.2, where Pontryagin's Maximum Principle was considered.

We first introduce the basics such as the (feedback-) control problem, Bellman equation and the HJB equation. Next, we design a computable algorithm using PMP and the Bellman equation and we state the Policy Iteration algorithm adapted to the finite horizon problem. After that we briefly describe the application of our framework from the previous chapters and finally give numerical evidence.

We consider an optimal control problem with finite time horizon.

Optimal Control Problem 5. For $x \in \mathbb{R}^d$ minimize w.r.t. $u \in L^2(0,T;\mathbb{R}^m)$ the cost functional $\mathcal{J} : [0,T] \times \mathbb{R}^d \times L^2(0,T;\mathbb{R}^m) \to \mathbb{R} \cup \infty$, defined as,

$$\mathcal{J}(t_0, x, u(\cdot)) := \int_{t_0}^{T} c(y(t)) + u(t)' R u(t) \,\mathrm{d}t + c_T(y(T)),$$

where

$$\begin{aligned} \dot{y}(t) &= f(y(t)) + g(y(t))u(t) \\ y(0) &= x. \end{aligned} \tag{5.8}$$

If either the trajectory $y(t)$ does not exist for every $t \in [t_0, T]$ or the integral within the cost functional does not converge, we set $\mathcal{J}(t_0, x, u) = \infty$.

We assume $c : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ and $c_T : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ are non-negative, coercive, and $C^1(\mathbb{R}^d)$. Further, let $R \in \mathbb{R}^{m,m}$ be symmetric, positive definite. For initial data $x \in \mathbb{R}^d$ and fixed control $u(\cdot) \in L^2(0,T;\mathbb{R}^m)$, we denote by $y_x(t,u) \in \mathbb{R}^d$ the evaluation of the trajectory at time t. If the context is clear we just write $y(t)$. We further assume that $f : \mathbb{R}^d \to \mathbb{R}^d$ and $g : \mathbb{R}^d \to \mathbb{R}^{d,m}$ are $C^1(\mathbb{R}^d)$ functions. We write $\ell(x,u) = c(x) + u'Ru$. Note that time-dependent cost functionals and state dynamics can be covered by an autonomization as described in [BC97].

We define the value function as

$$v^* : [0,T] \times \mathbb{R}^d \to \mathbb{R}, \quad (t,x) \mapsto \inf_{u \in L^2(0,T;\mathbb{R}^m)} \mathcal{J}(t,x,u)$$

and the corresponding Bellman equation takes the following form.

Theorem 5.2.1 ([BC97])**.** *For all $x \in \mathbb{R}^d$ and and $0 \leq t_0 < t_1 \leq T$ we have*

$$v^*(t_0,x) = \inf_{u \in L^2(t_0,t_1;\mathbb{R}^m)} \Big[\int_{t_0}^{t_1} \ell(y(t),u(t))\,\mathrm{d}t + v^*(t_1,y(t_1)) \Big], \tag{5.9}$$

where $y(\cdot)$ is the trajectory corresponding to (5.8) with control u and initial value $y(t_0) = x$. Additionally, we have $v^*(T,\cdot) = c_T(\cdot)$.

We further have the following HJB equation.

Theorem 5.2.2 ([BC97; BD97])**.** *Write $h(x,u) = f(x) + g(x)u$. Assume that there exists $L > 0$ such that*

$$(h(x,u) - h(y,u), x - y) \leq L|x-y|^2$$

for all $x, y \in \mathbb{R}^d$ and $u \in \mathbb{R}^m$ and that v^* is continuous. Then, the value function $v^*(t,x)$ is a viscosity solution of

$$\partial_t v^*(t,x) + \sup_{u \in \mathbb{R}^m} \Big[- D_x v^*(t,x) \cdot (f(x) + g(x)u) - \ell(x,u) \Big] = 0$$

with final condition $v(T,\cdot) = c_T(\cdot)$.

Theorem 5.2.3. *For Optimal Control Problem 5 an optimal feedback control is given by*

$$\alpha^*(t,x) = -\frac{1}{2} R^{-1} g(x)' \nabla v^*(t,x),$$

if ∇v^* exists.

Note that in contrast to the content of Chapter 2, i.e. the infinite horizon case, the value function is time-dependent in the finite horizon case. The same holds for the corresponding Bellman and HJB equation.

We denote the closed-loop system as

$$\dot{y} = f(y) + g(y)\alpha(t,y), \quad y(t_0) = x \tag{5.10}$$

for any policy $\alpha : [0,T] \times \mathbb{R}^d \to \mathbb{R}^m$ continuous on $[0,T]$ and Lipschitz in $\mathbb{R}^d$. For $\tau \geq 0$ we define the flow $\mathbf{\Phi}_\tau^\alpha : [0, T-\tau] \times \mathbb{R}^d \to [\tau, T] \times \mathbb{R}^d$ such that $\mathbf{\Phi}_\tau^\alpha(t_0,x) = (t_0 + \tau, y(t_0+\tau))$,

where $y(t_0+\tau)$ is the evaluation of the trajectory at time $\tau+t_0$ with initial condition $y(t_0)=x$ w.r.t (5.10). In analogy to (2.3) we write

$$v^\alpha(t,x)=\int_t^T \ell(\Phi^\alpha_{s-t}(t,x),\alpha(\Phi^\alpha_{s-t}(t,x)))\,\mathrm{d}s+c_T(\Phi^\alpha_{T-t}(t,x)). \tag{5.11}$$

We finish this section by stating the Policy Iteration algorithm for the finite horizon case.

Algorithm 11: Policy Iteration time-continuous

input : A Policy $\alpha\in F$.
output: An approximation of v^* and α^*.
Set $k=0$.
while *not converged* **do**
For all $t\in[0,T)$ solve

$$v_{k+1}(t,x)=v^{\alpha_k}(t,x), \tag{5.12}$$

defined in (5.11).
Then update the policy according to

$$\alpha_{k+1}(t,x)=-\frac{1}{2}R^{-1}g(x)'\nabla v_{k+1}(t,x).$$

$k:=k+1$.
end

In the above formulation we observe that we cannot solve (5.12) for all t. Thus, we reinterpret the Bellman equation to obtain a computable algorithm in the next section.

5.2.1 Reformulation as a Series of Open-Loop Control Problems

In this section, we reinterpret the Bellman equation as a series of open-loop control problems. These open-loop control problems are defined on a small subset of $[t_l,t_{l+1})\subset[0,T]$, which is why we refer to them as local optimal control problems. To this end, we consider a discrete set of time points $0=t_1<\cdots<t_L=T$. We notice that the Bellman equation has essentially the same structure as the finite horizon control problem and that we can interpret it as such. For convenience we repeat the Bellman equation below

$$v^*(t_l,x)=\inf_{u\in L^2(t_l,t_{l+1};\mathbb{R}^m)}\Big[\int_{t_l}^{t_{l+1}}\ell(y(t),u(t))\,\mathrm{d}t+v^*(t_{l+1},y(t_{l+1}))\Big], \tag{5.13}$$

and notice that $v^*(t_{l+1},y(t_{l+1}))$ can be interpreted as a final condition and $\ell(y(\cdot),u(\cdot))$ as the running cost with governing ODE system (5.8). Due to its restriction to the time-interval $[t_l,t_{l+1}]$, we call this problem the local optimal control problem. Note that this local optimal control problem can be interpreted as a Model Predictive Control (MPC) cost functional as in Section 2.8 with the value function as final condition.

The local optimal control problem can be solved by adjoint methods, which is already covered in Section 2.2. Recall that the Pontryagin Maximum Principle (PMP) gives as necessary conditions for a minimal control

$$\partial_t p^*(t)=-\partial_y H(t,y^*,u^*(t),p^*(t)),\quad p^*(T)=\partial_y c_T(y^*(T)),$$

where $H(t, y, u, p) = \ell(t, y, u) + p^T f(t, y, u)$ and $u^*(t) \in \arg\min_u H(t, y^*(t), u, p^*(t))$ [Pon+62; Don+95], cf. Section 2.2. After introducing a time discretization, this system is solved by standard gradient decent methods, see e.g. [HK10; Trö10] and Algorithm 1. Note that in this formulation the gradient of the final condition appears, which means that by setting the value function to be the final condition we might run into regularity issues. However, we ignore this problem for now and assume that the value function is differentiable. We alleviate this problem by replacing the exact final condition with a (smooth) approximation.

Traditionally, open-loop control methods suffer from long time horizons. However, due to our time-discretization, we can assume that the time-horizon is small. This results in the following algorithm for the continuous case, where we denote by $\hat{v}$ an approximation of the value function v^*. Note that a similar algorithm appears in [Atk+94; Zho+13] for time-discrete control problems.

Algorithm 12: Backwards solution to the Bellman equation - continuous case

input : Time points $0 = t_0 < \cdots < t_L < T$.
output : An approximation $\hat{v}$ of the value function v^* at the time points
Set $\hat{v}(t_L, \cdot) = c_T(\cdot)$
for $l = L - 1$ *to* 0 **do**
  Calculate an approximation $\hat{v}(t_l, \cdot)$ of the value function by solving the local optimal control problem (5.13) with final condition $\hat{v}(t_{l+1}, \cdot)$.
end

The approach for the approximation step within this algorithm can of course be chosen freely. We observe that within this formulation we have not done any discretization other than setting time points where we want to compute the value function. The next step is usually a time-discretization of the underlying ODE (5.8), such that the optimal control problem can be solved. This time-discretization does not have to be on the same time-grid as the time-points where the value function shall be evaluated. In fact, it can be finer. In this case, we use linear interpolation of the value function to obtain the value function at the intermediate points as described in Section 4.6.4 and repeated below

$$v^*(t, x) \approx \frac{t_{l+1} - t}{\tau} v^*(t_l, x) + \frac{t - t_l}{\tau} v^*(t_{l+1}, x) \text{ for } t \in [t_l, t_{l+1}),\ x \in \mathbb{R}^d. \tag{5.14}$$

Note that this interpolation does not affect Algorithm 12 as in this algorithm the value function is only evaluated at the time-points t_l. This is only relevant when the value function is used to compute a feedback-law via Theorem 5.2.3. We also make use of this interpolation for the Policy Iteration approach in Section 5.2.2.

Assuming an equidistant discretization of the ODE, the dimension of the local optimal control problems increases only linearly in time and does not increase with the spatial dimension. Note that solving the local optimal control problems for every initial value is computationally not feasible. We instead draw samples $x_i \in \Omega \subset \mathbb{R}^d$, Ω compact, and use a regression type approach as described in Chapter 3.

5.2.2 Policy Iteration Approach

The second approach we consider is the Policy Iteration algorithm.

As in the previous chapter, we use the same time subdivision $0 = t_0 < \cdots < t_L = T$. We use the Policy Iteration to approximate the value function by solving the local optimal control problems on our time-grid in the following way. For a fixed policy α defined on $[t_l, t_{l+1})$, the corresponding linearized Bellman equation takes the linearized form

$$v^\alpha(t_l, \cdot) = \int_{t_l}^{t_{l+1}} \ell^\alpha_{t-t_l}(t_l, \cdot)\,\mathrm{d}t + v^\alpha(t_{l+1}, \Phi^\alpha_\tau(t_{l+1}, \cdot)).$$

Note that plugging in the optimal policy α^* and the optimal final condition v^* recovers the Bellman equation (5.9). In this formulation, the policies are depending on the time t. However, if we only compute $v^\alpha(t_l, x)$ for every l, the policy using the optimality condition in Theorem 5.2.3 cannot be evaluated at times other than t_l. Thus, we again make use of interpolation between the time-points

$$v^\alpha(t, \cdot) = \frac{t_{l+1} - t}{\tau} v^\alpha(t_l, \cdot) + \frac{t - t_l}{\tau} v^\alpha(t_{l+1}, \cdot) \text{ for } t \in [t_l, t_{l+1}). \tag{5.15}$$

Note that because of the backwards iteration, $v^\alpha(t_{l+1}, \cdot) \approx v^*(t_{l+1}, \cdot)$ is already known (or approximated) and thus fixed within the Policy Iteration. The Policy Iteration algorithm is then given by Algorithm 13, where we abbreviate $\ell^\alpha(t, x) = \ell(x, \alpha(t, x))$.

Algorithm 13: Policy Iteration for approximating the local optimal control problem.

input : A Policy α_0, $0 \le t_l < t_{l+1} \le T$ and an approximation of $v^*(t_{l+1}, \cdot)$, denoted by $\hat{v}(t_{l+1}, \cdot)$

output : An approximation of $v^*(t_l, \cdot)$ and $\alpha^*(t_l, \cdot)$, denoted by $\hat{v}(t_{l+1}, \cdot)$ and $\hat{\alpha}(t_{l+1}, \cdot)$.

Set $k = 0$.

while *not converged* **do**

Solve the linear equation

$$v_{k+1}(t_l, x) = \int_{t_l}^{t_{l+1}} \ell^{\alpha_k}(t_l, \Phi^{\alpha_k}_{t-t_l}(t_l, x))\,\mathrm{d}t + \hat{v}(t_{l+1}, \Phi^{\alpha_k}_\tau(t_l, x)) \tag{5.16}$$

and use linear interpolation between v_{k+1} and $\hat{v}(t_{l+1}, \cdot)$ as in (5.15) to obtain $v_{k+1}(t, \cdot)$ for $t \in [t_l, t_{l+1})$. Then update the policy according to

$$\alpha_{k+1}(t, x) = -\frac{1}{2} R^{-1} g(x)' \nabla v_{k+1}(t, x), \quad t \in [t_l, t_{l+1}).$$

$k := k + 1$.

end

Set $\hat{v}(t, \cdot) = v_k(t, \cdot)$ and $\hat{\alpha}(t, \cdot) = \alpha_k$ for $t \in [t_l, t_{l+1})$.

Again, convergence of this algorithm for the inexact case, especially with the linear interpolation, is still an open problem. Due to the policy update where the gradient of v appears within the fixed-point iteration, this method can be prone to overfitting. We address this issue by adding the adaptive regularization term, as described in Section 4.8.

Note that the right-hand-side of (5.16) can be computed point-wise without knowledge of the underlying dynamical system. Only a black-box solver for the flow $\Phi^\alpha_{t_l}$ and a function for evaluation of the running cost $\ell^\alpha_{t_l}$ is needed. We also observe, that the time-discretization of the value function does not necessarily have to be the same as the time discretization of the

ODE. The above algorithm is used to solve the Bellman equation on the complete time frame in a backwards iteration as in Algorithm 12.

5.2.3 Error Propagation

The structure of the optimization problem allows for proving an error propagation with respect to time, provided that the value function is found with a fixed error in the $L^\infty(\Omega)$ norm, where Ω is again a compact subset of $\mathbb{R}^d$. We first prove that for optimal control problems of similar form, the value functions are similar.

Lemma 5.2.4. *Let v_1, v_2 be value functions to optimal control problems of the form*

$$v_1(x) = \inf_u \int_{t_0}^{t_1} \ell(y,u)\,\mathrm{d}t + c_1(y(t_1)), \quad v_2(x) = \inf_u \int_{t_0}^{t_1} \ell(y,u)\,\mathrm{d}t + c_2(y(t_1)),$$

with the same underlying ODEs, $y(\cdot)$ be the solution to the ODEs with initial condition x and $|c_1(x) - c_2(x)| \leq \delta$ for all x. Then

$$|v_1(x) - v_2(x)| \leq \delta$$

as well.

Proof. Let $x \in \Omega$ and w.l.o.g. assume that $v_1(x) \geq v_2(x)$. Due to the definition of v_2, for every $\varepsilon > 0$ there is a control u_2 such that

$$\int_{t_0}^{t_1} \ell(y_2,u_2)\,\mathrm{d}t + c_2(y_2(t_1)) - \varepsilon \leq v_2(x),$$

where y_2 is the trajectory corresponding to u_2. Plugging u_2 into the first cost functional yields

$$v_1(x) \leq \int_{t_0}^{t_1} \ell(y,u_2)\,\mathrm{d}t + c_1(y(t_1)) := \tilde{v}_1(x).$$

Now it follows that

$$\begin{aligned} v_1(x) - v_2(x) &\leq \tilde{v}_1(x) - v_2(x) \\ &\leq \int_{t_0}^{t_1} \ell(y_2,u_2)\,\mathrm{d}t + c_1(y_2(t_1)) - \Big(\int_{t_0}^{t_1} \ell(y_2,u_2)\,\mathrm{d}t + c_2(y_2(t_1))\Big) + \varepsilon \\ &= c_1(y_2(t_1)) - c_2(y_2(t_1)) + \varepsilon \leq \delta + \varepsilon. \end{aligned}$$

Since this holds for every $\varepsilon > 0$, the claim follows. □

This Lemma allows us to prove an error propagation within our discrete algorithm. The main idea is that the solution $\tilde{v}_l$ to the local optimal control problem (5.13) can be seen as an intermediate between the exact value function $v^*(t_l, \cdot)$ and the computable approximation $\hat{v}_l$.

Theorem 5.2.5. *By $\tilde{v}_l$ denote the solution to the local optimal control problem* (5.13) *with terminal condition $\hat{v}_{l+1}$. Assume that $|\tilde{v}_l(x) - \hat{v}_l(x)| \leq \delta$ for all $x \in \Omega$ and all $1 \leq l \leq L$ and that the trajectories of the ODE system stay within Ω. Then*

$$\|\hat{v}_{L-l}(\cdot) - v^*(T - l\tau, \cdot)\|_{L^\infty(\Omega)} \leq l\delta$$

for all $0 \leq l \leq L$.

Proof. We prove the theorem inductively. First note that we have $v^*(T,\cdot) = \hat{v}_L(\cdot)$. Thus, $\|\hat{v}_{L-1}(\cdot) - v^*(T-\tau,\cdot)\|_{L^\infty(\Omega)} \leq \delta$ by assumption.

Now assume that $\|\hat{v}_{L-l}(\cdot) - v^*(T-l\tau,\cdot)\|_{L^\infty(\Omega)} < l\delta$. Finally, we have

$$\begin{aligned}
&\|\hat{v}_{L-(l+1)}(\cdot) - v^*(T-(l+1)\tau,\cdot)\|_{L^\infty(\Omega)} \\
&\quad\leq \underbrace{\|\hat{v}_{L-(l+1)}(\cdot) - \tilde{v}_{L-(l+1)}(\cdot)\|_{L^\infty(\Omega)}}_{\leq\delta \text{ by assumption}} + \underbrace{\|\tilde{v}_{L-(l+1)}(\cdot) - v^*(T-(l+1)\tau,\cdot)\|_{L^\infty(\Omega)}}_{\leq l\delta \text{ by assumption and Lemma 5.2.4}} \\
&\quad\leq (l+1)\delta.
\end{aligned}$$

This finishes the proof. □

This theorem yields an error bound in the $L^\infty(\Omega)$ norm. Using a regression approach, however does only yield bounds in the $L^2(\Omega)$ norm. Here, we make use of our finite-dimensional model set, cf. Remark 18.

5.2.4 Comparing both Approaches and Possible Extensions

In this section, we compare the open-loop and Policy Iteration approaches and give rise to possible improvements/adaptions of the algorithms.

Comparison

We notice that both algorithms share the same backwards iteration to approximate $v^*(t_l,\cdot)$ for all t_l. Thus, we have to compare how the local optimal control problems are solved.

Within the Policy Iteration approach the value function is approximated in an iterative scheme, where for every iteration step trajectories of length τ_k have to be computed and then a linear equation has to be solved. In contrast to that, for the open-loop approach, an optimal control problem has to be solved for every sample point. After that a single linear equation has to be solved. Consequently, generating the samples for the open-loop approach is more expensive, provided that the Policy Iteration does not need too many iterations, while solving the linear equation is more expensive in the Policy Iteration case due to the iteration scheme.

For both approaches, we draw samples $x_i \in \Omega$ and keep the same samples during the complete iteration. Within the backwards iteration we have to choose initial data for both approaches. In the Policy Iteration approach, an initial policy has to be chosen. Due to the short time-horizon of the local optimization problems it is in most cases possible to choose the 0 policy. However, in many cases choosing the policy from the previous step is valid as well, i.e. setting $v_0(t,\cdot) = \hat{v}(t_{l+1},\cdot)$, $t \in [t_l, t_{l+1})$ and obtaining α_0 using the optimality condition from Theorem 5.2.3. In our numerical tests we use the latter approach.

For the open-loop approach we do not need an initial policy. Instead, we have to choose initial controls for the open-loop solver. Here, it is again possible to choose the 0 control. However, using the data that was generated in the previous time step can increase the convergence rate tremendously. More exactly, we denote by $u_{i,l}$ the initial guess, to emphasize its dependency on the sample x_i and on the initial time t_l. We use the control that was computed in the previous time step, i.e. $u_{i,l} = u^*_{i,l+1}$, which is close to optimal if the difference between the final conditions is small. After optimizing $u_{i,l}$ we denote the optimal control by $u^*_{i,l}$.

For both approaches we notice that generating the samples can be parallelized perfectly. In the case of the Policy Iteration approach we have to solve an ODE on a short time frame, whereas in the open-loop approach an optimal control problem has to be solved, which involves a gradient descent scheme where several forward and backward ODEs have to be solved.

Possible Improvements

The first improvement we propose is based on the error propagation from Theorem 5.2.5 and the Bellman equation. We first observe that the Bellman equation does not only hold for initial time t_l and final time t_{l+1}, but instead for every final time larger than t_l. Assuming that we obtain the same error δ for the regression w.r.t. every end-point larger than t_l, larger time-horizon decreases the error propagation. By setting the end-point to be t_{l+2}, the error bound from Theorem 5.2.5 is halved. Setting the final time to be T for every initial time t_l prevents any error propagation. Of course, solving the open-loop control problems with longer time-horizon becomes increasingly difficult. Here, a balance between time-horizon and error propagation has to be found.

In a similar way it is possible to increase the time horizon for the Policy Iteration approach. This is done by integrating along longer trajectories as indicated in Figure 5.5.

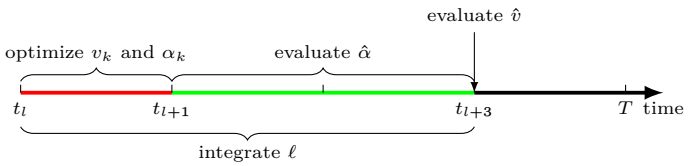


Figure 5.5: Visualization of increased integration horizons. The red part is optimized via the Policy Iteration, the green part is only evaluated and the evaluate or black part is estimated by evaluating $\hat{v}$ at time t_{l+3} . Note that $\hat{\alpha}$ and $\hat{v}$ are already computed and thus fixed.

The second proposed improvement can only be used by the open-loop approach. The idea is using the optimal controls $u^*_{i,l}$ that we compute via the gradient descent method. Due to Theorem 5.2.3, we know that

$$u^*_{i,l}(t_l) = -\frac{1}{2}R^{-1}g(x_i)'\nabla v^*(t_l, x_i).$$

This additional information can be incorporated to a modified regression problem, where we add a quadratic loss term containing the control

$$\hat{v}_l = \operatorname*{arg\,min}_{v\in\mathcal{M}} \frac{1}{N}\sum_{i=1}^{N} |v(x_i) - \tilde{v}(x_i)|^2 + \eta|u^*_{i,l}(t_l) + \frac{1}{2}R^{-1}g(x_i)'\nabla v(x_i)|^2,$$

where $\eta \geq 0$. Adding the loss w.r.t. to the gradient of v is closely related to the so-called *physical informed neural network* (PINN) approach known in deep learning [RPK19]. We also want to highlight that in [AKK21] a similar adaption of the loss functional is done, where improved performance is reported.

Finally, another improvement is again motivated by MPC, cf. Section 2.8. Instead of computing the feedback law by using the optimality condition in Theorem 5.2.3 we can

compute it by an MPC approach using the value function as final condition. Depending on the length of the time horizon this can be done in real-time, yielding an optimal feedback law. Due to the error propagation (backwards in time) it can be expected that the error in the value function is smaller at later time steps. Moreover, the gradient of the value function does not appear directly in the feedback law, but only indirect in the adjoint method within the MPC method.

A possible adaption of the open-loop algorithm is to decouple the value function and the controller. In the open-loop approach it is possible to solve separate regression problems for approximating $\tilde{v}$ and u^*. As the information on u^* is a byproduct of the open-loop ansatz this does not increase the complexity of generating the data. However, instead of one regression problem for the value function, two regression problems have to be solved with one being the value function and the other being the controller. Decoupling the value function and the controller is an approach also used in the actor-critic approach known from reinforcement learning [SB18].

5.2.5 Results

We present results of numerical tests for different optimal control problems. The calculations were performed on a AMD Ryzen 5 PRO 3500U 8x 2.60GHz, 16 GB RAM Fedora 33 Linux distribution and the code is available on `https://github.com/lsallandt/finitehorizon_bellman`. In every test we consider a cost functional of the form

$$\underset{u\in L^2((0,\infty);\mathbb{R}^m)}{\arg\min} \mathcal{J}(x,u) = \int_0^T \|y(t)\|^2 + 0.1|u(t)|^2 \ \mathrm{d}t + c\|y(T)\|^2, \tag{5.17}$$

where $c \geq 0$ and a PDE

$$\dot{y} = f(y) + g(y)u, \quad y \in L^2(-1,1).$$

As the first step we discretize the PDE in space, such that we obtain a finite dimensional system of ODEs, which we also denote as

$$\dot{y} = f(y) + g(y)u, \quad y \in \mathbb{R}^d.$$

For this discretization we use simple finite differences methods. We implement our algorithms for the spatially discretized PDE. As polynomial ansatz spaces we use the tensor product of one-dimensional H^2-orthogonal polynomials of degree smaller than 4 and we use the loss functional with adaptive regularization strategy, cf. Section 4.8. We benchmark the controllers obtained by our optimization by comparing it to the optimal open-loop control over the whole time-horizon $[0,T]$. We compute this optimal control by using the same gradient descent method used for the local optimal control problems. In fact, we use a simple fixed step-size and for the update direction we use the current gradient and the gradient of the previous step. In particular for Test 1 finding the optimal control over the whole time horizon without a good initial control is non-trivial, and computing the first gradient fails, due to the instability of the system. Thus, we use the control generated by our feedback controllers as initial controls. This problem did not occur for the local optimal control problems due to their short time horizon. We further compare our controllers to the linear quadratic regulator (LQR), a closed-loop controller that is obtained by linearizing the systems around 0 and then

solving the Riccati equation. We denote this controller by α_{LQR}. In the numerical tests we discretize the time-dependency of the value function using $\tau = \tau_l = t_{l+1} - t_l = 0.01$. The ODE is discretized using a different step-size, namely 0.001 and the explicit Runge-Kutta 4 method, which means that for every step in the value function, 10 steps of the ODE are computed. Here, we use linear interpolation of the value function, cf. (5.14), to obtain the value function at the steps between our discretization points t_l. We state that for this example the uncoupling of the time discretization is integral for the numerical success of our method. Setting $\tau = \tau_l = 0.001$ we obtain worse results. This effect can be attributed to the error propagation from Theorem 5.2.5.

Remark 42. In the following tests, we distinguish between the policy α, the corresponding cost estimator v and the real generated cost $\mathcal{J}(\cdot, \alpha(\cdot))$. For fixed x, we obtain $v(x)$ by simply evaluating v. Here, no trajectory has to be computed. We obtain $\mathcal{J}(x, \alpha(x))$ by numerically integrating along the trajectory with initial condition x. Note that $\mathcal{J}(x, \alpha(x))$ is basically the numerical approximation of the cost functional with respect to a feedback law, defined in (2.4.2).

5.2.5.1 Test 1: Diffusion with Unstable Reaction Term

We consider a diffusion equation with unstable reaction term and Neumann boundary condition, cf. [KK18, Test 2]. Solve (5.17) with $c = 1$ and $T = 0.3$ for $y \in L^2(-1, 1)$ subject to

$$\begin{aligned} \dot{y} &= \sigma \Delta y + y^3 + \chi_\omega u \\ y(0) &= x \end{aligned}$$

with Neumann boundary condition and χ_ω is the characteristic function w.r.t. $\omega = [-0.4, 0.4]$. We choose $\sigma = 1$ and use a finite differences grid with $d \in \mathbb{N}$ grid points to discretize the spatial domain. We denote by A this finite difference discretization of the Laplace operator and by $G \in \{0, 1\}^d$ the discretization of the characteristic function χ_ω. Then, we obtain a system of d ordinary differential equations

$$\begin{aligned} \dot{y} &= \sigma A y + y^3 + G u \\ y(0) &= x. \end{aligned}$$

Using the step-size $h = \frac{2}{d+1}$ we get a finite dimensional approximation of the term $\|y(t)\|_H^2$ in the cost functional. For this test we choose a spatial dimension of $d = 32$. As the underlying equation is non-linear, our ansatz for the value function is the tensor product of polynomials up to degree 4. The internal ranks chosen are

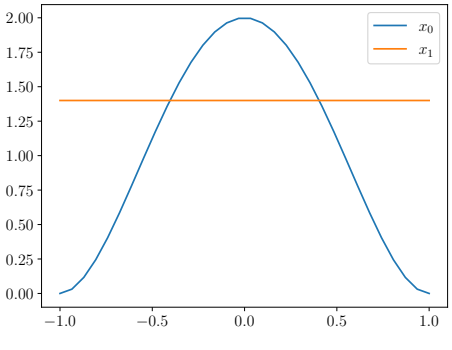
$$[3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 6, 6, 6, 6, 6, 6, 5, 5, 5, 4, 3].$$

We solve the HJB equation in 32 dimensions on the set $[-2, 2]^d$. While the full ansatz space has dimension 5^{32}, the TT has 5395 degrees of freedom. For the calculations we use 32370 uniformly distributed Monte Carlo samples.

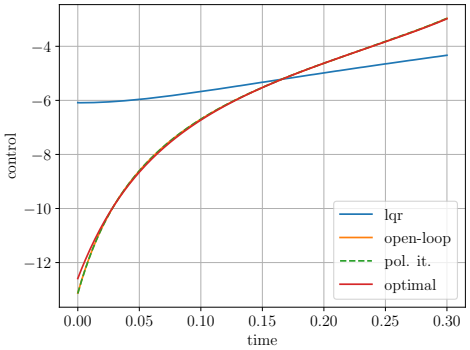
Remark 43. We stress that the number of Monte Carlo samples is extremely small when comparing it to the dimension of the ambient space $32370 \ll 5^{32} \approx 10^{22}$ and only when

comparing it to the degrees of freedom in the TT representation the numbers become comparable, $\mathbf{32370 = 6 \cdot 5395}$. This further indicates, that the curse of dimensionality is broken by our ansatz. Additional studies, where the minimal number of samples is compared to the dimensions of the underlying systems have to be done.

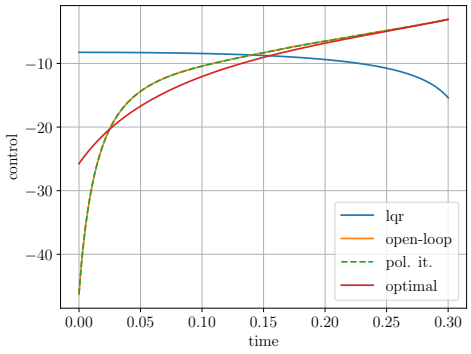
We first test the feedback controllers for certain initial values, visualized in Figure 5.6. For both controllers, significant improvement in cost is noticeable, with the greatest being approximately $\mathbf{38\%}$ of the cost saved compared to the LQR controller. Moreover, we see that the computed feedback laws generate close to optimal costs for the tested initial values. We



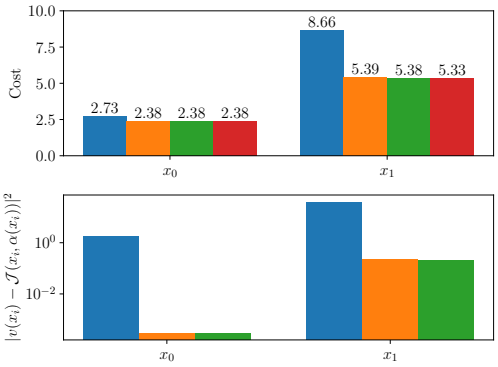
(a) Initial values x_0 and x_1.



(b) Generated controls, initial value x_0.



(c) Generated controls, initial value x_1.



(d) Generated cost and accuracy of the value function. Blue is the LQR controller, orange is the open-loop ansatz, green is the Policy Iteration ansatz, red is the optimal control.

Figure 5.6: The generated controls and cost for different initial values for the diffusion with unstable reaction term.

further investigate the performance of the controller by choosing initial values of the type $[x, x, \ldots, x]$, where $x \in [\mathbf{0}, \mathbf{2}]$. In Figure 5.7 we see that for small x every controller is close to optimal. For x larger than $\mathbf{1}$ the LQR controller performs significantly worse than the other controllers. By increasing the initial values beyond $\mathbf{1.5}$, the LQR controller fails to

stabilize the system, while our controllers still stabilize the system. However, for such values the computed feedback laws differs evidently from the optimal control. For such large initial values the present setting was too coarse to achieve better accuracy. Note that for computing the optimal control for such extreme initial values a good initial guess is needed. In particular, due to the blow-up, it is not possible to use the control generated by the LQR controller as initial guess. We were only able to find the optimal control by using the controls generated by our feedback controllers as initial guesses.

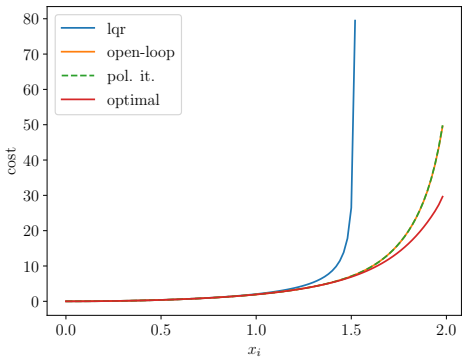


Figure 5.7: Cost of initial values of the type $[x, x, \ldots, x]$ for the diffusion with unstable reaction term.

Next we test the feedback law for random initial values. Note that as in the previous section, because of the diffusion, equally distributed samples and normally distributed samples yield low cost on average, even with the LQR controller, and in this case no improvements of the cost is to be expected. Thus, we use a special distribution of initial values that we specify here again. For every initial value we choose an equally distributed integer between 2 and 20. This number is the degree of a random polynomial. Next we choose a polynomial with normal distributed coefficients of the degree we chose. We further modify the polynomial in the following way $p(x) := \tilde{p}(x)(x-1)(x+1)$, such that we have $p(-1) = p(1) = 0$. Finally, we rescale p such that its maximum in $[-1, 1]$ is 1.9. In order to have an idea how these initial values look, we plotted 10 initial values in Figure 5.4a and report the results in Table 5.1. We observe that for these initial values the LQR controller failed to obtain cost smaller than 100 in 65 out of 1000 initial values, while our controllers were not only stabilizing for every initial value, but also close to optimal. On the set of initial values that the LQR controller did not fail we see an average improvement of 32% while being close to optimal. Finally, we report that on the whole set of initial values, which means that these where the LQR controller failed are included, the average difference to the optimal control was 1%. We note that out of all 1000 samples, the largest relative difference between the optimal control and our feedback controllers is $\approx 34\%$, which was also observed in Figure 5.7 for values close to the boundary of our integration area.

controller	% cost < 100	avg. cost	max. rel. diff. to opt.	avg. Bellman error
LQR	93.5	4.453	nan	39.67
open-loop	100	2.61	0.3419	0.048
pol. it.	100	2.61	0.3381	0.047
optimal	100	2.60	0	

Table 5.1: Performance of the different controllers for 1000 samples drawn from the polynomial distribution for the diffusion with unstable reaction term. The averaged values are only taken from the subset of initial values that the LQR succeeded in stabilizing.

5.2.5.2 Test 2: Allen-Cahn Equation

As underlying equation we use a one-dimensional Allen-Cahn equation similar to [KK18, Test 3]. Solve (5.17) for $y \in L^2(-1,1)$ subject to

$$\begin{aligned} \dot{y} &= \sigma \Delta y + y - y^3 + \chi_\omega u \\ y(0) &= x \end{aligned}$$

with Neumann boundary condition. Here, we use the same discretization as in the previous section. The constants are the same except for $\sigma = 0.2,\ \omega = [-0.5, 0.2]$. We choose the same ranks as in the last test

$$[3,4,5,5,5,6,6,6,6,7,7,7,7,7,7,7,7,7,6,6,6,6,6,6,5,5,5,4,3]$$

and solve the HJB on $[-2,2]^{32}$.

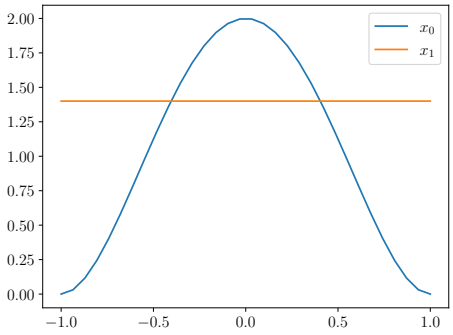
From Figure 5.8 we deduce that for certain initial values, significant improvement of cost is possible for both controllers with the greatest improvement being 25% of the cost. Again, the open-loop and the Policy Iteration approach yield similar performance. We again notice that for these initial values our calculated value functions are more accurate than the predictions from the LQR-based value function.

We again further investigate the performance of the controller by choosing initial values of the type $[x, x, \ldots, x]$, where $x \in [0,2]$. In Figure 5.9 we see that for small x every controller is close to optimal. For x larger than 0.5 the LQR controller yields significantly worse performance than the other controllers. However, in this case the LQR controller stabilizes the system for every initial value.

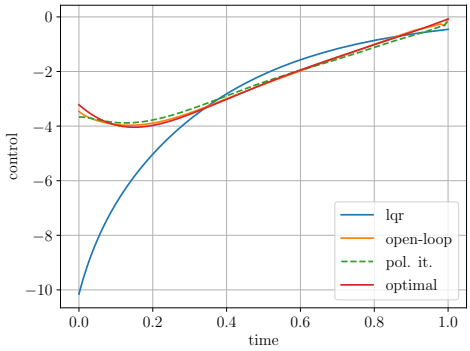
Next we again test random initial values using the same setup as in the previous test. In Table 5.2 we compare the performance of the controllers for 1000 random initial values drawn from the polynomial distribution. Again, the open-loop and the Policy Iteration approach yield close to optimal performance, while the LQR generates 22% more cost. Moreover, the the open-loop and the Policy Iteration approaches predict the generated cost accurately.

5.2.6 Short Conclusion and Outlook

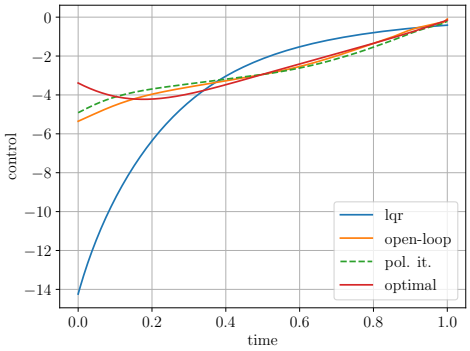
We have demonstrated the applicability of our tensor-train approach for state spaces in 32 dimensions. For both approaches we have obtained near-optimal controllers. Several extensions of the present approaches are already discussed in Section 5.2.4. An interesting



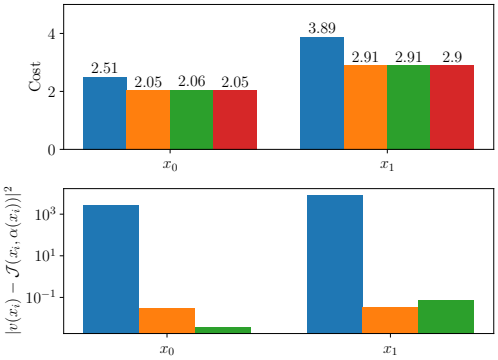
(a) Initial values x_0 and x_1.



(b) Generated controls, initial value x_0.



(c) Generated controls, initial value x_1.



(d) Generated cost and accuracy of the value function. Blue ist the LQR controller, orange is the open-loop ansatz, green is the Policy Iteration ansatz, red is the optimal control.

Figure 5.8: The generated controls and cost for different initial values for the Allen-Cahn equation.

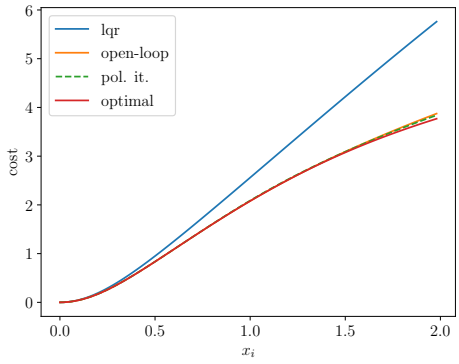


Figure 5.9: Cost of initial values of the type $[x, x, \ldots, x]$ for the Allen-Cahn equation.

controller	% cost < 100	avg. cost	max. rel. diff. to opt.	avg. Bellman error
LQR	100	2.434	0.4234	24172
open-loop	100	1.985	0.0146	0.03267
pol. it.	100	1.988	0.0152	0.030593
optimal	100	1.981	0	

Table 5.2: Performance of the different controllers for 1000 samples drawn from the polynomial distribution. The averaged values are only taken from the subset of initial values that the LQR succeeded in stabilizing.

application of the open-loop approach to the infinite horizon case is by solving the local open-loop problems in an iterative fashion, which approximates the infinite horizon case. Using this approach, promising first results were achieved.

5.3 The Stochastic Exit-Time Case

The content of this section is previously published in [Fac+20]. In this section, we consider the exit-time problem for stochastic systems. The exit-time problem was already considered in Example 2.5.3 and the aim of this problem is steering the state out of (or into) a set. Further, we replace the ODE by a stochastic ODE - an SDE.

We assume, that the SDE in $\Xi \subset \mathbb{R}^d$ open and bounded, given by

$$\begin{aligned} \mathrm{d}X_t &= b(X_t)\,\mathrm{d}t + \sigma(X_t)\,\mathrm{d}W_t + g(X_t)u_t\,\mathrm{d}t \\ X_0 &= x, \end{aligned} \tag{5.18}$$

describes the state of a system at time t, where $X_t \in \mathbb{R}^d$, $b : \mathbb{R}^d \to \mathbb{R}^d$ is smooth with bounded derivatives, $g : \Xi \to \mathbb{R}^{d,m}$, $\sigma : \mathbb{R}^d \to \mathbb{R}^{d,d}$ are smooth with bounded derivatives and W_t is a Wiener process. Additionally, $u_t \in \mathbb{R}^m$ is a progressively measurable [FS06] control parameter adapted to the process X_t. For each control u we define the exit-time η

$$\eta = \inf\{t > 0 | X_t \notin \Xi\}$$

and the discounted cost function

$$\mathcal{J}(x,u) = \mathbb{E}\Big[\int_0^\eta e^{-\gamma t}(c(X_t) + u_t' R u_t)\,\mathrm{d}t + e^{-\gamma\eta} w(X_\eta) | X_0 = x\Big],$$

where c and $w : \mathbb{R}^d \to \mathbb{R}$ are some given continuous and positive functions and $R \in \mathbb{R}^{m,m}$ is positive definite. It holds that $\eta < \infty$ a.s., see e.g. [Kle12].

Our goal is it to find the control u with minimal cost, i.e.

$$\min_u \mathcal{J}(x,u).$$

Here, we do not specify the space that we minimize over. Formally speaking, we consider the space of controls mapping to $\mathbb{R}^m$ that are measurable and adapted to a filtration induced by the Brownian motion. As we are later considering feedback controls, we omit the technical

details and instead refer to [Bis78; HS06] and references therein. Due to the lack of precise formulation of the open-loop problem, we can only formally represent the optimal control problem in our usual framework by using the notation $\mathbf{CS}$ as the (informal) control space.

Optimal Control Problem 6. For $x \in \Xi \subset \mathbb{R}^d$ minimize w.r.t. $u \in \mathbf{CS}$ the cost functional $\mathcal{J} : \Xi \times \mathbf{CS} \to \mathbb{R} \cup \infty$, defined as,

$$\mathcal{J}(x,u) = \mathbb{E}\Big[\int_0^\eta e^{-\gamma t}(c(X_t) + u_t' R u_t)\,\mathrm{d}t + e^{-\gamma\eta} w(X_\eta) | X_0 = x\Big],$$

where

$$\begin{aligned}\mathrm{d}X_t &= b(X_t)\,\mathrm{d}t + \sigma(X_t)\,\mathrm{d}W_t + g(X_t)u_t\,\mathrm{d}t\\ X_0 &= x,\end{aligned}$$

and

$$\eta = \inf\{t > 0 | X_t \notin \Xi\}.$$

If either the trajectory X_t does not exist for every $t \in \mathbb{R}_+$ or the integral within the cost functional or the expectation value does not converge, we set $\mathcal{J}(x,u) = \infty$.

We define the value function $v^* : \mathbb{R}^d \to \mathbb{R}$ as infimum of the cost functional over all controls, i.e.

$$v^*(x) := \inf_u \mathcal{J}(x,u) \text{ for all } x \in \Xi.$$

The control u^* is optimal if $v^*(x) = \mathcal{J}(x,u^*)$ holds. Under the assumption that there exists a Lipschitz continuous feedback control $\alpha : \Xi \to \mathbb{R}^m$ with finite cost for any initial state $x \in \Xi$ we replace (5.18) by the closed loop system

$$dX_t = b(X_t) + \sigma(X_t)dW_t + g(X_t)\alpha(X_t) \tag{5.19}$$

and denote the corresponding state by X_t^α to stress the dependence on the feedback. It has been shown under suitable regularity assumptions on the right hand side of the SDE, that (5.19) is well defined and is differentiable with respect to the initial values [Arn98, chapter 2],[AS95]. Since we are considering time-homogeneous Itô diffusions, our processes fulfill the (strong) Markov property [Oks13].

In the following we only consider feedback laws that give us finite costs. In analogy to (2.3), we define v^α as

$$v^\alpha(x) := \mathbb{E}\Big[\int_0^\eta e^{-\gamma t}(c(X_t^\alpha) + \alpha(X_t^\alpha)' R \alpha(X_t^\alpha))\,\mathrm{d}t + e^{-\gamma\eta} w(X_\eta) | X_0 = x\Big] \tag{5.20}$$

and assume that there exists an optimal, Lipschitz continuous feedback law such that $v^{\alpha^*} = v^*$. If the value function v^* is known and differentiable, the optimal control is given explicitly by [BC97]

$$\alpha^*(x) = -\frac{1}{2} R^{-1} g(x)' \nabla v^*(x).$$

Moreover, abbreviating $r^\alpha(x) = c(x) + \alpha(x)' R\alpha(x)$ and $\tau \wedge \eta = \min\{\tau, \eta\}$, a stochastic Bellman equation is obeyed for every $\tau > 0$ [BN16]

$$v^*(x) = \mathbb{E}\Big[\int_0^{\eta\wedge\tau} e^{-\gamma t} r^{\alpha^*}(X_t^{\alpha^*})\,\mathrm{d}t + e^{-\gamma\eta\wedge\tau} v^*(X_{\eta\wedge\tau}^{\alpha^*})|X_0 = x\Big] \tag{5.21}$$

$$\alpha^*(x) = -\frac{1}{2} R^{-1} g(x)' \nabla v^*(x) \tag{5.22}$$

with Dirichlet boundary condition

$$v^*(x) = w(x) \text{ on } \partial\Xi.$$

We again employ the Policy Iteration algorithm to solve this coupled equation by alternating between the value updates given by (5.21) and the policy updates given by (5.22). To this end we again notice that by fixing a policy α this coupled equation becomes uncoupled and a linear function equation is remaining

$$\begin{aligned} v^\alpha(x) &= \mathbb{E}\Big[\int_0^{\eta\wedge\tau} e^{-\gamma t} r^\alpha(X_t^\alpha)\,\mathrm{d}t + e^{-\gamma\eta\wedge\tau} v^\alpha(X_{\eta\wedge\tau}^\alpha)\Big] \\ &:= \mathbb{E}\Big[\int_0^{\eta\wedge\tau} e^{-\gamma t} r^\alpha(X_t^\alpha)\,\mathrm{d}t + e^{-\gamma\eta\wedge\tau} v^\alpha(X_{\eta\wedge\tau}^\alpha)|X_0^\alpha = x\Big]. \end{aligned} \tag{5.23}$$

with boundary condition $v^\alpha = w$ on $\partial\Xi$. Note, that the expectation value is a conditional expectation value with respect to the initial value $X_0 = x$. For the ease of notation, in the sequel we sometimes drop the condition $X_0 = x$ when the context is clear.

As in the previous section, the value function obeys a HJB equation under additional assumptions. This equation is given as [Pen92; BN16]

$$0 = \frac{1}{2}\sum_{i,j=1}^{d} (\sigma\sigma^\top)_{ij}(x)\partial_{x_i}\partial_{x_j} v^*(x) + \inf_{u\in\mathbb{R}^m}\{\nabla v^*(x)\cdot(b(x) + g(x)u) + c(x) + u'Ru\}$$

with Dirichlet boundary condition

$$v^* = w \text{ on } \partial\Xi.$$

Again, for a fixed policy the HJB equation reduces to a linear equation. The HJB formulation has a critical advantage in this case. Namely, the expectation value in the Bellman equation gets replaced by a diffusive term. While this is advantageous, the implementation of the boundary condition poses difficulties in this case. Note that in the Bellman equation, the boundary condition is (approximately) fulfilled by approximating the expectation value.

5.3.1 Policy Iteration

We can again formulate the governing linearized Bellman equation via the Koopman operator [Koo31]. To do so we rewrite (5.23) by using the Koopman operator [KKS16; Las94; BMM12; ČMM19] with a slight modification to incorporate the exit-time

$$K_\tau^\alpha[v] = \mathbb{E}[v(X_{\eta\wedge\tau}^\alpha)|X_0^\alpha = x].$$

This allows us, to reformulate equation (5.23) as operator equation

$$(I - e^{-\gamma\eta\wedge\tau}K_\tau^\alpha)v(x) = \mathbb{E}[\int_0^{\eta\wedge\tau} e^{-\gamma t}r^\alpha(X_t^\alpha)\,\mathrm{d}t|X_0^\alpha = x]. \tag{5.24}$$

with boundary condition $v^\alpha = w$ on $\partial\Xi$. We later comment on how to enforce the boundary condition and how to detect the exit-time. With the operator equation (5.24) we can now give the Policy Iteration algorithm.

Algorithm 14: Policy Iteration for solving (5.21)

input : A Policy $\alpha_0 \in F$.
output : An approximation of v^* and α^*.
Set $k = 0$.
while *not converged* **do**
Solve the linear equation

$$(I - e^{-\gamma\eta\wedge\tau}K_\tau^{\alpha_k})v_k(x) = \mathbb{E}\big[\int_0^{\eta\wedge\tau} e^{-\gamma t}r^{\alpha_k}(X_t^{\alpha_k})\,\mathrm{d}t|X_0^{\alpha_k} = x\big], \tag{5.25}$$

with boundary condition $v_k = w$ on $\partial\Xi$.
Then update the policy according to

$$\alpha_{k+1}(x) = -\frac{1}{2}R^{-1}g(x)'\nabla v_k(x).$$

$k := k + 1$.
end

Note that in the algorithm we have to choose an initial policy $\alpha_0 \in F$. In some cases this is a particular hard challenge, as the policy has to lead to finite cost for every initial state x. However, the Wiener process ensures that the uncontrolled dynamics driven by a potential arrives at the exit set in finite time almost surely. Thus, we initialize the Policy Iteration with the zero control $\alpha \equiv 0$. For solving (5.25) in the above algorithms we again combine the regression approach with the tensor-train approach.

In order to apply the regression approach, we interpret the above equation as a fixed-point equation, as done in Section 3.1, and then we use Monte Carlo quadrature to integrate within the state and probability space. For convenience, we repeat the procedure below. For the sake of readability we set the discount $\gamma = 0$.

To this end, consider a domain $\Omega \subset \Xi$ suitable for Monte Carlo integration. The policy $\alpha := \alpha_k$ is given and we assume that $\hat{v} : \mathbb{R}^d \to \mathbb{R}$ on the r.h.s. of the equation below is given as well. The aim is finding $\tilde{v}$, where

$$\tilde{v}(x) := \mathbb{E}\big[\int_0^{\eta\wedge\tau} r(X_s^\alpha)\,\mathrm{d}s + \hat{v}(X_{\eta\wedge\tau}^\alpha)|X_0^\alpha = x\big]\ ,\ x \in \Omega.$$

Note that if $\tilde{v} = \hat{v}$ we have found a solution to (5.25) and that the above equation is of regression-type.

We again introduce a loss functional defined on $V := L^2(\Omega)$, (or more generally $V := L^2(\Omega, \rho)$ with some probability density ρ) such that $\hat{v}, \tilde{v} \in V$. Then, we have

$$\tilde{v} = \operatorname*{arg\,min}_{v\in V} \mathcal{R}^\alpha(v, \hat{v}), \quad \mathcal{R}^\alpha(v, \hat{v}) = \|v(\cdot) - \mathbb{E}\big[\int_0^{\eta\wedge\tau} r^\alpha(X_s^\alpha)\,\mathrm{d}s + \hat{v}(X_{\eta\wedge\tau}^\alpha)|X_0^\alpha = \cdot\big]\|_V^2$$

and since $\tilde{v} \in V$, we have $\mathcal{R}^\alpha(\tilde{v}, \hat{v}) = 0$. We are seeking a solution $\hat{v} = \tilde{v}$ which constitutes a fixed point problem

$$\tilde{v} = \operatorname*{arg\,min}_{v \in V} \mathcal{R}^\alpha(v, \tilde{v}), \quad \mathcal{R}^\alpha(v, \tilde{v}) = \|v - \mathbb{E}\big[\int_0^{\eta \wedge \tau} r^\alpha(X_s^\alpha)\,\mathrm{d}s + \tilde{v}(X_{\eta\wedge\tau}^\alpha) | X_0^\alpha = \cdot\big]\|_V^2.$$

Again, finding the exact minimizer $\tilde{v} \in V$ is infeasible and we restrict to the set of TTs with fixed ranks. Replacing both $\|\cdot\|_V$, and the expectation value by a Monte Carlo approximation yields the following loss functional

$$\begin{aligned} v_{(U,N,M)}^\alpha &= \operatorname*{arg\,min}_{v \in U} \mathcal{R}_{N,M}^\alpha(v, v), \\ \mathcal{R}_{N,M}^\alpha(v, \hat{v}) &= \frac{1}{N}\sum_{i=1}^N \Big| v(x_i) - \frac{1}{M}\sum_{j=1}^M \int_0^{\eta\wedge\tau} r(x_i^{j,\alpha}(t))\,\mathrm{d}t + \hat{v}(x_i^{j,\alpha}(\eta \wedge \tau))\Big|^2, \end{aligned}$$

where $x_i^{j,\alpha}(0) = x_i$ for $i = 1, \dots, N$. We again stress, that we have two Monte Carlo approximations: The first for integrating the stochastic differential equation (5.19) with different paths $t \mapsto x^{j,\alpha}(t)$, $j = 1, \dots, M$. And the second Monte Carlo integration for different initial values $x = x_i$, $i = 1, \dots N$.

Remark 44. We compute the minimizer $v_{(U,N,M)}^\alpha$ using an iterative scheme, i.e. by iterating a regression problem of the form

$$v_{k+1} = \operatorname*{arg\,min}_{v \in U} \mathcal{R}_{N,M}^\alpha(v, v_k).$$

We now comment on how we compute the terms within $\mathcal{R}_{N,M}^\alpha(v, v_k)$. For our forward simulations $x_i^{j,\alpha}$, computed using the Euler-Mayurama scheme [KP92], we approximate the integral $\int_0^{\eta\wedge\tau} r(x_i^{j,\alpha}(t))dt$ using the trapezoidal rule. For every time point we evaluate whether the set Ξ^C is hit. If it is exited, we stop the integration and add the boundary condition w evaluated at the end-point. If Ξ is not exited within the time τ, we add v_k evaluated at the end-point. Note that in this way the boundary condition is (approximately) enforced. It is possible to include additional penalty terms for points on the boundary of Ξ, which we have done in the first example in our numerical tests. Note that we can proceed similarly if the process is discounted, i.e. $\gamma > 0$. Finally, we modify the loss functional by adding the adaptive regularization strategy, as described in Section 4.8

We finish this section by combining the content of the previous sections into Algorithm 15.

Algorithm 15: Approximative Policy Iteration with Tensor-Trains

input : A Policy $\alpha_0 \in F$, samples x_i, $1 \leq i \leq N$, number of repetitions M, length of trajectory τ, discount γ

output : An approximation of v^* and α^*.

Set $k = 0$.

while *not converged* **do**

Compute trajectories $x_i^{j,\alpha_k}(t)$, $i \leq N$, $j \leq M$ and corresponding running cost $c_i^j = \int_0^{\eta \wedge \tau} e^{-\gamma t} r(x_i^{j,\alpha}(t))dt$.

Set $i = 0$ and $v_{k,0} = v_k$.

while *not converged* **do**

Set $w_i^j = e^{-\gamma \eta} w(x_i^{j,\alpha_k}(\eta))$.

If $\eta < \tau$ (for this trajectory), else set $w_i^j = e^{-\gamma \tau} v_{k,i}(x_i^{j,\alpha_k}(\tau))$.

Find

$$v_{k,i+1} = \operatorname*{arg\,min}_{v \in U} \frac{1}{N} \sum_{i=1}^{N} \left| v(x_i) - \frac{1}{M} \sum_{j=1}^{M} c_i^j + w_i^j \right|^2 + \delta \|v\|_{H^2_{\text{mix}}}^2$$

using the ALS algorithm.

$i = i + 1$.

end

$v_k := v_{k,i}$.

Update the policy according to

$$\alpha_{k+1}(x) = -\frac{1}{2} B^{-1} g(x)^T \nabla v_k(x).$$

$k := k + 1$.

end

5.3.2 Numerical Results

We present results of numerical tests for different optimal control problems. The calculations were performed on a AMD Phenom II 4x 3.20GHz, 16 GB RAM Fedora 31 Linux distribution. In our tests we consider a set $E = \Xi^C$ that we want to steer the state to and a cost functional of the form

$$\mathcal{J}(x_0, u) = \mathbb{E} \int_0^{\eta} 1 + \frac{1}{2} |u_t|^2 \, \mathrm{d}t,$$

if not specified otherwise. We also denote the integration domain $\Omega \subset \Xi$. The first test is a one-dimensional problem, where the exact solution is known numerically. The second test has a two-dimensional state space and finally we test the algorithm on a 6 dimensional state space, where metastabilities are present, and a 40-dimensional example.

Remark 45. In the following, we distinguish between the policy α, the corresponding policy estimation function v and the policy evaluation function $\mathcal{J}(\cdot, \alpha(\cdot))$. For fixed x, we obtain $v(x)$ by simply evaluating v. Here, no trajectory has to be computed. We obtain $\mathcal{J}(x, \alpha(x))$ by numerically integrating along the trajectory with initial condition x. Note that $\mathcal{J}(x, \alpha(x))$ is basically the numerical approximation of the cost functional with respect to a policy, defined in (5.20).

Remark 46. Within the test cases we specify the constants that we chose. Namely, the length of the trajectory τ, the number of spatial samples N, the number of repetitions for every sample M and, as we set N proportional to the degrees of freedom of the tensor-train representation of v, we also state the number of degrees of freedom. Note that for the numerical tests the length of the trajectory does not necessarily have to be the step-size of the Euler-Mayurama scheme for solving the SDE. In fact, for the tests we use a step size of $\tau_{\text{SDE}} = 0.001$ for the Euler-Mayurama scheme, if not specified otherwise. The length of the trajectory is mostly set to 0.1, which means that 100 steps within the SDE solver are used.

For every test we set the regularization to be adaptive as described in Section 4.8. In the beginning of every 'left-to-right sweep' within the ALS algorithm, we set the constant to be the current residuum $\mathcal{R}^{\alpha}_{N,M}$.

5.3.2.1 Test 1: One-Dimensional Double Well Potential

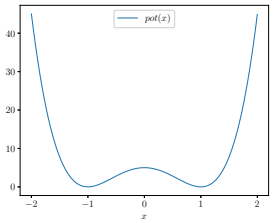


Figure 5.10: The double well potential pot.

We consider the double well potential on $\Omega = [-2, 2]$ with $\Xi = [-2, 1)$, E = [1, 2].

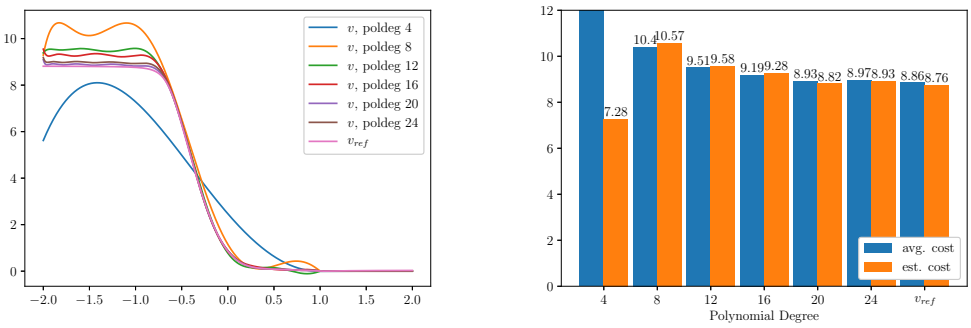
$$\text{pot}(x) = 5(x^2 - 1)^2,$$

visualized in Figure 5.10. The corresponding SDE is

$$dX_t = \nabla \text{pot}(X_t)\,\mathrm{d}t + \sigma(X_t)\,\mathrm{d}W_t + g(X_t)u_t(X_t)\,\mathrm{d}t,$$

where $g(x) = g \in \mathbb{R}^{d,d}$ maps to the identity matrix. Here, we cannot expect the value function to be included in our ansatz space. Thus, we experiment with different polynomial degrees, visualized in Figures 5.11a and 5.11b. For the computation of the controllers we set the number of samples to $N = 10 \cdot dof$, where dof is the polynomial degree increased by 1. We set the length of the trajectory to $\tau = 0.1$ and compute for every sample $M = 1000$ trajectories. Note that the length of the trajectory consists of 100 individual steps in the Euler-Mayurama scheme, i.e. $0.1 = 100 \cdot 0.001$. For this example we alter the loss functional $\mathcal{R}_{N,M}$ by adding a penalty for deviations of v^{α} on the boundary 1, as described in Section 4.8. We compare the results to a reference solution that is obtained by solving the HJB equation with a finite differences scheme with 3000 grid points.

We observe, that higher polynomial degrees yield a better approximation of the reference solution, with polynomial degree of 20 yielding the best results. We also deduce from Figure 5.11a that we do not exactly reproduce v_{ref}. From Figure 5.11b we deduce that the performance of the controller of polynomial 20 is less than 1% higher than the performance of the reference solution. The average exit-time for polynomial degree 20 is 1.24.



(a) The approximated value functions and the reference solution for the double well potential.

(b) Estimated cost and average cost by averaging over 10000 trajectories with initial value $x_0 = -1$. Avg. cost for polynomial degree 4 is 34.03.

Figure 5.11: One dimensional double well potential

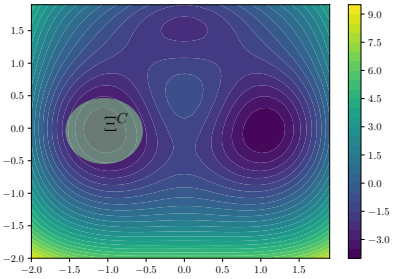
5.3.2.2 Test 2: Two Dimensional Three-Hole Potential

We consider a two dimensional three-hole potential with one being less significant than the others. Note that this potential has already been used in different contexts and is sometimes referred to as Müller-Brown potential, see i.e. [HS97; Par+03]. In particular we have $\Omega = [-3, 3]^2$ and

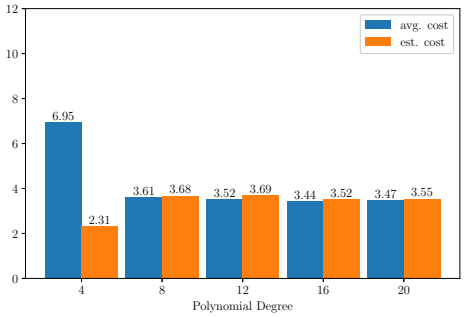
$$\begin{aligned}\text{pot}(x_1, x_2) = &\, 3e^{-x_1^2-(x_2-1/3)^2} - 3e^{-x_1^2-(x_2-5/3)^2} - 5e^{-(x_1-1)^2-x_2^2} - 5e^{-(x_1+1)^2-x_2^2} \\ &+ 0.2x_1^4 + 0.2(x_2 - 1/3)^4.\end{aligned}$$

The dynamics are then given as in the previous example. We choose a ball of radius 0.5 around $x_{target} \approx [-1.048, -0.042]$ as target set E and set $\Xi = \Omega \setminus E$. Note that x_{target} contains a local minimum of pot. Both sets are visualized in Figure 5.12a. Again, we compare the performance of controllers with different polynomial degree and see that higher order polynomials substantially increase the performance of the controllers. The best performance is achieved by a controller of polynomial degree 16. However, lower polynomial degree yields 'good' results as well. For the computation of the controllers we set the number of samples to $N = 10 \cdot dof$, where dof is again the number of degrees of freedom of the tensor-train. In this two-dimensional example, we set the rank of the tensor-train to be maximal for every polynomial degree. Further, we set $M = 100$ and $\tau = 0.1$. In Figure 5.13, we plot different trajectories of the dynamical system for both, the uncontrolled and controlled system. Note that the controlled trajectories get steered into the set E within the given time frame of 10, while the uncontrolled dynamics stay in the minimum on the right. Here, we visually see the effect of controlling this dynamical system. Figure 5.12b again visualizes the performance of controllers for different polynomial degrees.

As the polynomial degree of 16 had the best performance, we add contour plots of the value function in Figure 5.14. In particular compare this Figure to those in [Har+13], where a similar Figure appears in a different context. The average exit-time for polynomial degree 16 is 1.44.

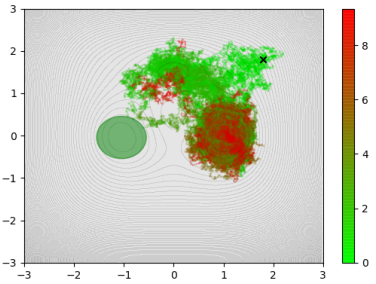


(a) The three-hole potential pot and the exit set Ξ^C

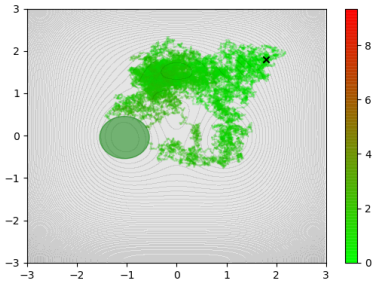


(b) Estimated cost and average cost by averaging over 10000 trajectories with initial value $x_0 = [1.8, 1.8]$.

Figure 5.12: Left: The potential pot. Right: Performance of controllers of different polynomial degree.



(a) Visualization of 10 trajectories, uncontrolled



(b) Visualization of 10 trajectories, controlled

Figure 5.13: Top: Plots of 10 trajectories, starting at $x_0 = [1.8, 1.8]$. The colors indicate the time within the trajectory, starting with green dots and ending with red dots. We stop the simulation at $T = 10$ or when the exit set is reached. As no red dots appear for the controlled dynamics, we see that in this case the exit set is reached in a fast manner. The black cross is the initial value.

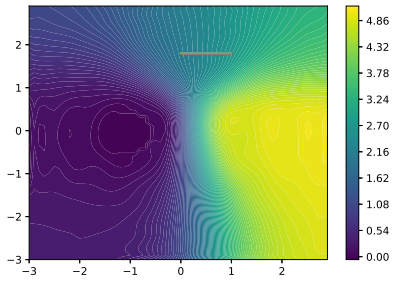


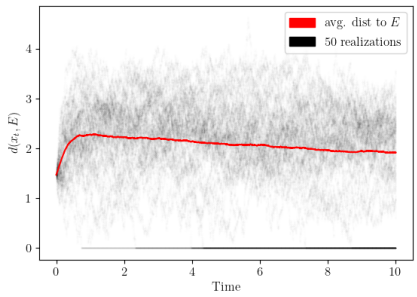
Figure 5.14: Contour plot of the approximation of the value function. Polynomial degree is 16.

5.3.2.3 Test 3: Higher Dimensional Problem

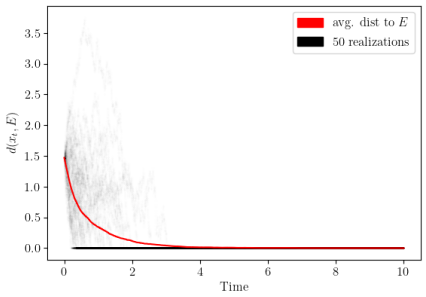
We consider the multi-dimensional Double Well potential

$$\text{pot}(x) = \sum_{i=1}^{d} \kappa_i (x_i^2 - 1)^2.$$

Note that this potential has 2^d local minima and the choice of κ determines their metastability. In our test we use $\kappa_i = 5$ for all i. The exit set E is $B_{0.5\sqrt{d}}(1)$, the ball of radius $0.5\sqrt{d}$ around $[1, \dots, 1] \in \mathbb{R}^d$. We further choose $\Omega = [-\frac{\pi}{2}, \frac{\pi}{2}]^d$. Note that while the radius of the exit set seems large, for the case $d = 6$, its volume is only 1.8% of the volume of Ω. We further stress, that while in the dynamics the dimensions are independent from each other, we cannot expect the value function and thus also the policy to have a strict separation in the dimensions. Here, the curse of dimensions comes into play. Choosing a polynomial degree of 6 and a tensor-train rank of $[5, 5, 5, 5, 5]$ allows us to reduce the ansatz space from 46656 degrees of freedom to 770. Again, setting $N = 10 \cdot dof$, $M = 100$ and $\tau = 0.1$ we visualize the resulting controller in Figure 5.15. We start the trajectory at $[-0.1, \cdots - 0.1] \in \mathbb{R}^d$. We see that the uncontrolled dynamics approach the exit set slowly, while the controlled dynamics most trajectories have reached the exit set at time 3. The resulting cost of the controlled dynamics is 4.44 and the predicted cost 4.74. Note that computing the average cost for the uncontrolled dynamics is not feasible because of the high metastability of the minima, as seen in Figure 5.15. The average exit-time in this test is 0.94. The metastability of this example limits our capabilities for solving higher-dimensional problems in this setting - the trajectories simply do not arrive at the exit set. As this issue is sampling-based, advanced sampling strategies can improve this result. Instead of diving into these strategies, we next demonstrate the capabilities of the method for a higher-dimensional problem, where the metastability is not an issue.



(a) Red: average distance of the exit set to 1000 realizations of the uncontrolled dynamics. Black (dots): scatter plot of the distance of 50 realizations of the uncontrolled dynamics to the exit set.



(b) Red: average distance of the exit set to 1000 realizations of the controlled dynamics. Black (dots): scatter plot of the distance of 50 realizations of the controlled dynamics to the exit set.

Figure 5.15: Visualization of the distance of trajectories to the exit set.

5.3.2.4 Test 4: High Dimensional Stochastic van der Pol Oscillator

We consider a stochastic Van der Pol oscillator as proposed in [ZHL21], see also [Xu+11]. The control problem is given as follows

$$\mathcal{J}(x,u) = \mathbb{E}\Big[\int_0^\eta e^{-\gamma t}(c(X_t) + |u_t|^2)dt + e^{-\gamma\eta}w(X_\eta)|X_0 = x\Big],$$

with dynamics

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t + g\alpha(X_t)dt.$$

We set $\gamma = 1$ and choose a spatial dimension of $d = 2m \in \mathbb{N}$ even. The governing functions are given as follows using the convention $x_0 = x_m$ and $x_{2m+1} = x_{m+1}$

$$w(x) = a|x|^2 - \varepsilon(\sum_{i=1}^{m} x_{i-1}x_i + \sum_{i=m+1}^{2m} x_i x_{i+1})$$

with $a, \varepsilon > 0$,

$$b_i(x) = \begin{cases} x_{i+m}, & 1 \le i \le m \\ (1 - x_{i-m}^2)x_i - x_{i-m}, & m+1 \le i \le 2m. \end{cases}$$

We further have $g \in \mathbb{R}^{d,m}$, $g_{ij} = 1$ if $i + m = j$ and 0 otherwise. We set the running cost

$$\begin{aligned} c(x) = \gamma a|x|^2 - \gamma[\sum_{i=1}^{m} \varepsilon x_i x_{i-1} + \sum_{i=m+1} \varepsilon x_i x_{i+1}] - 4ma \\ + \frac{1}{4q}[(2ax_{m+1} - \varepsilon(x_{2m} + x_{m+2}))^2 + \sum_{i=m+2}^{2m} (2ax_i - \varepsilon(x_{i-1} + x_{i+1}))^2] \\ - 2a\sum_{i=1}^{m} x_{m+ix_i} + \varepsilon\sum_{i=1}^{m} x_{m+i}x_{i-1} + \varepsilon\sum_{i=1}^{m-1} x_{m+i}x_{i+1} + \varepsilon x_{2m}x_1 \\ - (x_{m+1} - x_1 - x_1^2 x_{m+1})(2ax_{m+1} - \varepsilon(x_{2m} + x_{m+2})) \\ - \sum_{i=2}^{m}((x_{i+m} - x_i - x_i^2 x_{i+m})(2ax_{i+m} - \varepsilon(x_{i+m-1} + x_{i+m+1})). \end{aligned}$$

We choose $\Xi = B_1(0)$, i.e. an open ball of radius 1, and $E = \mathbb{R}^d \setminus \Xi$. This setting allows us to compute the exact value function as

$$v^*(x) = a|x|^2 - \varepsilon(\sum_{i=1}^{m} x_{i-1}x_i + \sum_{i=m+1}^{2m} x_i x_{i+1}).$$

We consider the case $d = 20$ and $d = 40$ and observe that the optimal ranks of the exact value function are given as

$$[3, 4, 4, 4, 4, 4, 4, 4, 3, 2, 3, 4, 4, 4, 4, 4, 4, 3]$$

for $d = 20$ and

$$[3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3]$$

for $d = 40$. We later test different ranks.

The presence of a reference solution allows us to compute reference losses on samples $x_i \in \Xi$ of the form

$$\mathrm{err}_v = \sum_{i=1}^{M_{\mathrm{err}}} |v_{TT}(x_i) - v^*(x_i)|^2 / \sum_{i=1}^{M} |v^*(x_i)|^2, \quad \mathrm{err}_\alpha = \sum_{i=1}^{M_{\mathrm{err}}} |\alpha_{TT}(x_i) - \alpha^*(x_i)|^2 / \sum_{i=1}^{M} |\alpha^*(x_i)|^2,$$

where v_{TT} and α_{TT} are our computed value function and corresponding control, and v^*, α^* are the optimal value function and control.

We set $N = 6 \cdot dof$, $M = 100$ and $\tau = 10 \cdot \tau_{\mathrm{SDE}}$. Using different polynomial degrees and ranks, we obtain accurate results as seen in Table 5.3. Note that for the four last rows we did reduce $M = 50$. The average exit-times for this problem are $2.55 \cdot 10^{-2}$ for the case $d = 20$ and $9.76 \cdot 10^{-3}$ for the case $d = 40$.

5.3.3 Short Conclusion and Outlook

We have demonstrated the applicability of our tensor-train approach for state spaces in up to 40 dimensions. Due to the stochastic nature of the trajectories, replacing the Bellman

d	pol.deg.	rank	τ_{SDE}	err_v	err_α
20	2	*opt*	$2 \cdot 10^{-3}$	$5.61 \cdot 10^{-3}$	$1.29 \cdot 10^{-2}$
20	2	$r_i = 3$	$2 \cdot 10^{-3}$	$5.61 \cdot 10^{-3}$	$1.34 \cdot 10^{-2}$
20	2	$r_i = 5$	$2 \cdot 10^{-3}$	$5.60 \cdot 10^{-3}$	$1.39 \cdot 10^{-2}$
20	4	*opt*	$2 \cdot 10^{-3}$	$5.58 \cdot 10^{-3}$	$1.56 \cdot 10^{-2}$
20	4	$r_i = 5$	$2 \cdot 10^{-3}$	$5.61 \cdot 10^{-3}$	$1.72 \cdot 10^{-2}$
20	2	*opt*	10^{-4}	$3.08 \cdot 10^{-4}$	$4.58 \cdot 10^{-4}$
20	2	$r_i = 3$	10^{-4}	$3.41 \cdot 10^{-4}$	$1.90 \cdot 10^{-4}$
20	2	$r_i = 5$	10^{-4}	$3.02 \cdot 10^{-4}$	$4.88 \cdot 10^{-4}$
20	4	*opt*	10^{-4}	$3.00 \cdot 10^{-4}$	$7.77 \cdot 10^{-4}$
20	4	$r_i = 5$	10^{-4}	$3.12 \cdot 10^{-4}$	$6.16 \cdot 10^{-4}$
40	2	*opt*	$2 \cdot 10^{-3}$	$6.86 \cdot 10^{-3}$	$1.47 \cdot 10^{-1}$
40	2	*opt*	10^{-4}	$2.55 \cdot 10^{-4}$	$1.04 \cdot 10^{-3}$
40	2	$r_i = 3$	10^{-4}	$2.68 \cdot 10^{-4}$	$1.29 \cdot 10^{-3}$
40	2	$r_i = 5$	10^{-4}	$2.66 \cdot 10^{-4}$	$8.15 \cdot 10^{-4}$
40	4	*opt*	10^{-4}	$2.68 \cdot 10^{-4}$	$9.67 \cdot 10^{-4}$
40	4	$r_i = 3$	10^{-4}	$2.72 \cdot 10^{-4}$	$2.34 \cdot 10^{-3}$
40	4	$r_i = 5$	10^{-4}	$2.71 \cdot 10^{-4}$	$9.67 \cdot 10^{-4}$

Table 5.3: Results for the stochastic Van der Pol oscillator.

equation by the HJB equation is particularly beneficial - in this approach no stochastic terms appear. Of course, instead of replacing the loss functionals, a combination of both losses can be beneficial, too. However, in the HJB approach the boundary condition is not fulfilled automatically.

In order to reduce the complexity of the computations several adaptions can be done. For computing the expected cost of the feedback law an adaptive number of trajectories can be used. Using this approach, areas where the variance is low can be represented with less complexity, whereas areas with high variance can be represented more accurately.

Due to the running-cost function c, which is $c \equiv 1$ for many examples, the choice of the initial control is important. For this case, if the zero control is used and no trajectory hits the exit-set, the running cost for every initial value is the same - which yields a constant function as the solution. In this case, the iterative scheme for solving the linearized Bellman equation does not converge. A non-naive way of choosing the initial control can help improving the results.

We finish this section by giving an informal interpretation of the exit-time problem as an infinite horizon problem.

5.3.4 Reinterpretation of the Finite Exit Time Problem as an Infinite Horizon Problem

We give an informal way to interpret the finite exit-time problem as an non-smooth infinite horizon optimal control problem. Taking this viewpoint has the advantage that standard HJB theory can be (formally) applied and that the usual Koopman operator without the stop time

η can be defined. Note that our implementation we use this viewpoint. To this end we restate the finite exit-time problem

$$\mathcal{J}(x,u) = \mathbb{E}\int_0^\eta c(X_t) + u_t' R u_t \,\mathrm{d}t,$$

subject to

$$\begin{aligned} dX_t &= b(X_t)\,\mathrm{d}t + \sigma(X_t)dW_t + g(X_t)u_t\,\mathrm{d}t \\ X_0 &= x \end{aligned}$$

and

$$\eta = \inf\{t > 0 | X_t \notin \Xi\}.$$

Denoting the characteristic function on Ξ by χ_Ξ we can represent this problem as an infinite horizon problem in the following way.

$$\mathcal{J}(x,u) = \mathbb{E}\int_0^\infty \chi_\Xi(X_t)(c(X_t) + u_t' R u_t)\,\mathrm{d}t,$$

subject to

$$\begin{aligned} dX_t &= \chi_\Xi(X_t)(b(X_t)\,\mathrm{d}t + \sigma(X_t)dW_t + g(X_t)u_t\,\mathrm{d}t) \\ X_0 &= x. \end{aligned}$$

Note that the characteristic function χ_Ξ basically sets the running cost and the dynamics to zero once the state is steered out of Ξ. Moreover, the dynamics and the cost functional is then non-smooth.

5.4 Solving High-Dimensional PDEs using Backward SDEs

The content of this section is previously published in [RSN21]. In this section, we consider a slightly different problem. However, the numerical treatment reveals many similarities. Instead of considering HJB equations, we consider a general semi-linear parabolic PDE. Note that HJB equations of stochastic finite horizon optimal control problems are of this type. In this section, we do not introduce fundamental terms such as stochastic processes, adapted to a filtration and Itô's lemma. For an in-depth introduction to this matter we refer to the textbooks [Oks13; KS98].

First we consider how backward stochastic differential equations (BSDEs) can be used to design iterative algorithms for approximating the solutions of high-dimensional PDEs. In this section, we consider parabolic PDEs of the form

$$(\partial_t + L)V(x,t) + h(x,t,V(x,t),(\sigma^\top \nabla V)(x,t)) = 0 \tag{5.26}$$

for $(x,t) \in \mathbb{R}^d \times [0,T]$, a non-linearity $h : \mathbb{R}^d \times [0,T] \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$, and a differential operator

$$L = \frac{1}{2}\sum_{i,j=1}^d (\sigma\sigma^\top)_{ij}(x,t)\partial_{x_i}\partial_{x_j} + \sum_{i=1}^d b_i(x,t)\partial_{x_i}, \tag{5.27}$$

with coefficient functions $b : \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$ and $\sigma : \mathbb{R}^d \times [0,T] \to \mathbb{R}^{d\times d}$. The terminal value is given by

$$V(x,T) = g(x), \tag{5.28}$$

for a specified function $g : \mathbb{R}^d \to \mathbb{R}$. Note that by using the time inversion $t \mapsto T - t$, the terminal value problem (5.26)-(5.28) can readily be transformed into an initial value problem.

BSDEs were first introduced in [Bis73] and their systematic study began with [PP90]. Loosely speaking, they can be understood as non-linear extensions of the celebrated Feynman-Kac formula [Par98], relating the PDE (5.26) to the stochastic process X_s defined by

$$\mathrm{d}X_s = b(X_s, s)\,\mathrm{d}s + \sigma(X_s, s)\,\mathrm{d}W_s, \quad X_0 = x_0, \tag{5.29}$$

where b and σ are as in (5.27) and W_s is a standard d-dimensional Brownian motion. The key idea is then to define the processes

$$Y_s = V(X_s, s), \qquad Z_s = (\sigma^\top \nabla V)(X_s, s) \tag{5.30}$$

as representations of the PDE solution and its gradient, and apply Itô's lemma to obtain

$$\mathrm{d}Y_s = -h(X_s, s, Y_s, Z_s)\,\mathrm{d}s + Z_s \cdot \mathrm{d}W_s, \tag{5.31}$$

with terminal condition $Y_T = g(X_T)$. Noting that the processes Y_s and Z_s are adapted to the filtration generated by the Brownian motion W_s, they should indeed be understood as backward processes and not be confused with time-reversed processes. A convenient interpretation of the relations in (5.30) is that solving for the processes Y_s and Z_s under the constraint (5.31) corresponds to determining the solution of the PDE (5.26) (and its gradient) along a random grid which is provided by the stochastic process X_s defined in (5.29).

5.4.1 Numerical Approximation of BSDEs

The BSDE formulation (5.31) opens the door for Monte Carlo algorithms aiming to numerically approximate Y_s and Z_s, and hence yielding approximations of solutions to the PDE (5.26) according to (5.30), see [BT04; GLW+05]. In this section, we discuss suitable discretizations of (5.31) and corresponding optimization problems.

To this end, let us define a discrete version of the process (5.29) on a time grid $0 = t_0 < t_1 < \cdots < t_L = T$ by

$$\widehat{X}_{l+1} = \widehat{X}_l + b(\widehat{X}_l, t_l)\Delta t + \sigma(\widehat{X}_l, t_l)\xi_{l+1}\sqrt{\Delta t}, \tag{5.32}$$

where $l \in \{0, \dots, L-1\}$ enumerates the steps, $\Delta t = t_{l+1} - t_l$ is the step-size, $\xi_{l+1} \sim \mathcal{N}(0, \mathrm{Id}_{d\times d})$ are normally distributed random variables and $\widehat{X}_0 = x_0$ provides the initial condition. Two discrete versions of the backward process (5.31) are given by

$$\widehat{Y}_{l+1} = \widehat{Y}_l - h_{l+1}\Delta t + \widehat{Z}_l \cdot \xi_{l+1}\sqrt{\Delta t}, \tag{5.33a}$$

$$\widehat{Y}_{l+1} = \widehat{Y}_l - h_l\Delta t + \widehat{Z}_l \cdot \xi_{l+1}\sqrt{\Delta t}, \tag{5.33b}$$

where we have introduced the shorthands

$$\begin{aligned} h_l &= h(\widehat{X}_l, t_l, \widehat{Y}_l, \widehat{Z}_l), \\ h_{l+1} &= h(\widehat{X}_{l+1}, t_{l+1}, \widehat{Y}_{l+1}, \widehat{Z}_{l+1}). \end{aligned} \tag{5.34a}$$

Finally, we complement (5.33a) and (5.33b) by specifying the terminal condition $\widehat{Y}_L = g(\widehat{X}_L)$. The reader is referred to [Zha17] for further details as well as existence and uniqueness results. It can be shown that both converge to the continuous-time process (5.31) as $\Delta t \to 0$, see [KP92].

As in the previous sections, our schemes solve the discrete processes (5.33a) and (5.33b) backwards in time. To wit, we start with the known terminal value $\widehat{Y}_L = g(\widehat{X}_L)$ and move backwards in iterative fashion until reaching $\widehat{Y}_0$. Throughout this procedure, we posit functional approximations $\widehat{V}_l(\widehat{X}_l) \approx \widehat{Y}_l \approx V(\widehat{X}_l, t_l)$ to be approximated in the update step $l+1 \to l$ which can either be based on (5.33a) or on (5.33b).

5.4.2 Setup of the Explicit Algorithm

Starting with the former, it can be shown that solving (5.33a) is equivalent to minimizing

$$\mathbb{E}\left[\left(\widehat{V}_l(\widehat{X}_l) - h_{l+1}\Delta t - \widehat{V}_{l+1}(\widehat{X}_{l+1})\right)^2\right] \tag{5.35}$$

with respect to $\widehat{V}_l$. This can be seen as follows. Consider the explicit discrete backward scheme

$$\widehat{Y}_{l+1} = \widehat{Y}_l - h(\widehat{X}_{l+1}, t_{l+1}, \widehat{Y}_{l+1}, \widehat{Z}_{l+1})\Delta t + \widehat{Z}_l \cdot \xi_{l+1}\sqrt{\Delta t}.$$

Taking conditional expectations w.r.t. to the σ-algebra generated by the discrete Brownian motion at time step l, denoted by $\mathcal{F}_l$, yields

$$\widehat{Y}_l = \mathbb{E}\left[\widehat{Y}_{l+1} + h(\widehat{X}_{l+1}, t_{l+1}, \widehat{Y}_{l+1}, \widehat{Z}_{l+1})\Delta t \middle| \mathcal{F}_l\right].$$

We can now recall that a conditional expectation can be characterized as a best approximation in L^2, namely

$$\mathbb{E}[B|\mathcal{F}_l] = \underset{\substack{Y \in L^2 \\ \mathcal{F}_l-\text{measurable}}}{\arg\min} \mathbb{E}\left[|Y - B|^2\right],$$

for any random variable $B \in L^2$, which brings

$$\widehat{Y}_l = \underset{\substack{Y \in L^2 \\ \mathcal{F}_l-\text{measurable}}}{\arg\min} \mathbb{E}\left[\left(Y - h_{l+1}\Delta t - \widehat{Y}_{l+1}\right)^2\right].$$

This then yields the explicit scheme depicted in (5.35). We refer once more to [GLW+05] for extensive numerical analysis, essentially showing that the proposed scheme is of order $\frac{1}{2}$ in the time step Δt.

Keeping in mind that $\widehat{V}_{l+1}$ is known from the previous step this results in an explicit scheme. Methods based on (5.35) have been extensively analyzed in the context of linear ansatz spaces for $\widehat{V}_l$ and we refer to [Zha04; GLW+05].

5.4.3 Handling Implicit Regression Problems

Moving on to (5.33b), we may as well penalize deviations in this relation by minimizing the alternative loss

$$\mathbb{E}[(\widehat{V}_l(\widehat{X}_l) - \widehat{h}_l\Delta t + \widehat{Z}_l \cdot \xi_{l+1}\sqrt{\Delta t} - \widehat{V}_{l+1}(\widehat{X}_{l+1}))^2], \tag{5.36}$$

with respect to $\widehat{V}_l$, see [HPW20]. In analogy to (5.34a) we use the shorthand notation

$$\widehat{h}_l = h(\widehat{X}_l, t_l, \widehat{V}_l(\widehat{X}_l), \sigma'(\widehat{X}_l, t_l)\nabla\widehat{V}_l(\widehat{X}_l)).$$

We note that since $\widehat{h}_l$ depends on $\widehat{V}_l$, approaches based on (5.36) will necessarily lead to implicit schemes. At the same time, we expect algorithms based on (5.36) to be more accurate in highly non-linear scenarios as the dependence in h is resolved to higher order.

In order to solve (5.36), we choose an initial guess $\widehat{V}_l^0$ and iterate the optimization of

$$\mathbb{E}[(\widehat{V}_l^{k+1}(\widehat{X}_l) - h(\widehat{X}_l, t_l, \widehat{Y}_l^k, \widehat{Z}_l^k)\Delta t + \widehat{Z}_l^k \cdot \xi_{l+1}\sqrt{\Delta t} - \widehat{V}_{l+1}(\widehat{X}_{l+1}))^2] \tag{5.37}$$

with respect to $\widehat{V}_l^{k+1}$ until convergence. In the above display, $\widehat{Y}_l^k = \widehat{V}_l^k(\widehat{X}_l)$ and $\widehat{Z}_l^k = (\sigma^\top\nabla\widehat{V}_l^k)(\widehat{X}_l)$ are computed according to (5.30). For theoretical foundation, we guarantee convergence of the proposed scheme when the step size Δt is small enough.

Theorem 5.4.1 ([RSN21])**.** *Assume that $\mathcal{U} \subset L^2(\Omega) \cap C_b^\infty(\Omega)$ is a finite dimensional linear subspace, that $\sigma(x,t)$ is non-degenerate for all $(x,t) \in \mathbb{R}^d \times [0,T]$, and that h is globally Lipschitz continuous in the last two arguments. Then, there exists $\delta > 0$ such that the iteration* (5.37) *converges for all $\Delta t \in (0, \delta)$.*

Remark 47. In order to ensure the boundedness assumption in Theorem 5.4.1 and to stabilize the computation we add a regularization term involving the Frobenius norm of the coefficient tensor to the objective in (5.37). This is done as described in Section 4.8 using adaptive regularization and the H^2_{mix}-norm. See Appendix A.1.2 for more details.

Remark 48 (Parameter initialization)**.** Since we expect $V(\cdot, t_l)$ to be close to $V(\cdot, t_{l+1})$ for any $l \in \{0, \dots, L-1\}$, we initialize the parameters of $\widehat{V}_l^0$ as those obtained for $\widehat{V}_{l+1}$ identified in the preceding time step.

Clearly, the iterative optimization using (5.37) is computationally more costly than the explicit scheme described in Section 5.4.2 that relies on a single optimization of the regression type per time step. However, implicit schemes typically ensure improved robustness [KP92] and therefore hold the promise of more accurate approximations (see Section 5.4.4 for experimental confirmation). In the following sections we compare the tensor-train performance to neural network (NN) implementations, which are nowadays considered state-of-the-art. In this approach, we perform (stochastic) gradient descent for both the explicit and implicit schemes and observe no significant differences in the corresponding run-times. However, for the TT approach we see significant differences in the run-time. For convenience, we summarize the developed methods in Algorithm 16.

Algorithm 16: PDE approximation

input : Initial parametric choice for the functions $\widehat{V}_l$ for $l \in \{0, \dots, L-1\}$.
output : Approximation of $V(\cdot, t_l) \approx \widehat{V}_l$ along the trajectories for $l \in \{0, \dots, L-1\}$.
Simulate K samples of the discretized SDE (5.32).
Choose $\widehat{V}_L = g$. **for** $l = L-1$ **to** 0 **do**
  Approximate either (5.35) or (5.36) (both depending on $\widehat{V}_{l+1}$) using Monte Carlo.
  Minimize this quantity (explicitly or by iterative schemes) and set $\widehat{V}_l$ to be the minimizer.
end

5.4.4 Numerical Examples

In this section, we consider some examples of high-dimensional PDEs that have been addressed in recent articles and treat them as benchmark problems in order to compare against our algorithms with respect to approximation accuracy and computation time. We refer to Appendix A.1 for implementation details and to Appendix A.2 for additional experiments. All our code is available under `https://github.com/lorenzrichter/PDE-backward-solver`. For the TT implementation we use the usual ansatz set of polynomials, enriched with the final condition, cf. Section 4.7.1.

5.4.4.1 Hamilton-Jacobi-Bellman Equation

The Hamilton-Jacobi-Bellman equation (HJB) is a PDE for the value function that represents the minimal cost-to-go in stochastic optimal control problems from which the optimal control policy can be deduced. As suggested in [EHJ17], we consider the HJB equation

$$\begin{aligned}(\partial_t + \Delta)\, V(x,t) - |\nabla V(x,t)|^2 &= 0,\\ V(x,T) &= g(x),\end{aligned}$$

with $g(x) = \log\left(\frac{1}{2} + \frac{1}{2}|x|^2\right)$, leading to

$$b = 0, \quad \sigma = \sqrt{2}\,\mathrm{Id}_{d\times d}, \quad h(x,s,y,z) = -\frac{1}{2}|z|^2$$

in terms of the notation established in Section 5.4. One appealing property of this equation is that (up to Monte Carlo approximation) a reference solution is available:

$$V(x,t) = -\log \mathbb{E}\left[e^{-g(x+\sqrt{T-t}\sigma\xi)}\right], \tag{5.38}$$

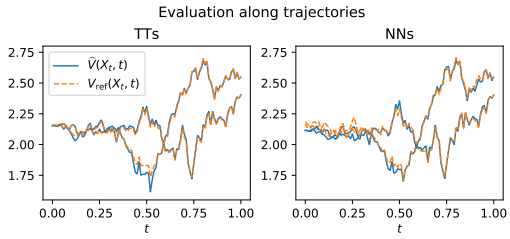
where $\xi \sim \mathcal{N}(0, \mathrm{Id}_{d\times d})$ is a normally distributed random variable (see Appendix A.2.1 for further details).

In our experiments we consider $d = 100$, $T = 1$, $\Delta t = 0.01$, $x_0 = (0,\dots,0)^\top$ and $K = 2000$ samples. In Table 5.4 we compare the explicit scheme stated in (5.35) with the implicit scheme from (5.36), once with TTs and once with NNs. For the tensor-trains we try different polynomial degrees, and it turns out that choosing constant ansatz functions is the best choice, while fixing the rank to be 1. For the NNs we use a DenseNet like architecture with 4 hidden layers (all the details can be found in Appendices A.1 and A.2).

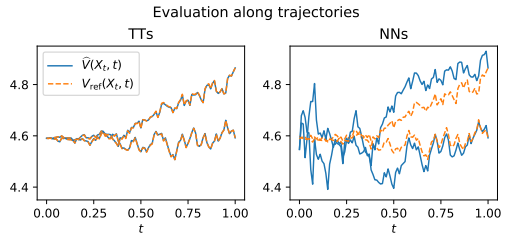
We display the approximated solutions at $(x_0, 0)$, the relative errors $\left|\frac{\widehat{V}_l(x_0) - V_{\mathrm{ref}}(x_0,0)}{V_{\mathrm{ref}}(x_0,0)}\right|$ with $V_{\mathrm{ref}}(x_0,0) = 4.589992$ being provided in [EHJ17], their computation times, as well as PDE and reference losses, which are specified in Appendix A.1. We can see that the TT approximation is both more accurate and much faster than the NN-based approaches, improving also on the results in [EHJ17; Bec+19]. As it turns out that the explicit scheme for NNs is worse in terms of accuracy than its implicit counterpart in all our experiments, but takes a very similar amount of computation time. Thus, we will omit reporting it for the remaining experiments. In Figures 5.16a and 5.16b we plot the reference solutions computed by (5.38) along two trajectories of the discrete forward process (5.32) in dimensions $d = 10$ and $d = 100$

	$\mathrm{TT}_{\mathrm{impl}}$	$\mathrm{TT}_{\mathrm{expl}}$	$\mathrm{NN}_{\mathrm{impl}}$	$\mathrm{NN}_{\mathrm{expl}}$
$\widehat{V}_0(x_0)$	4.5903	4.5909	4.5822	4.4961
relative error	$5.90\mathrm{e}^{-5}$	$3.17\mathrm{e}^{-4}$	$1.71\mathrm{e}^{-3}$	$2.05\mathrm{e}^{-2}$
reference loss	$3.55\mathrm{e}^{-4}$	$5.74\mathrm{e}^{-4}$	$4.23\mathrm{e}^{-3}$	$1.91\mathrm{e}^{-2}$
PDE loss	$1.99\mathrm{e}^{-3}$	$3.61\mathrm{e}^{-3}$	90.89	91.12
comp. time	41	25	44712	25178

Table 5.4: Comparison of approximation results for the HJB equation in $d = 100$.



(a) Reference solutions compared with implicit TT and NN approximations along two trajectories in $d = 10$ dimensions.



(b) Reference solutions compared with implicit TT and NN approximations along two trajectories in $d = 100$ dimensions.

and compare to the implicit TT and NN-based approximations. We can see that the TT approximations perform particularly well in the higher dimensional case $d = 100$.

In Figure 5.17 we plot the mean relative error over time, as defined in Appendix A.1, indicating that both schemes are stable and where again the implicit TT scheme yields better results than the NN scheme.

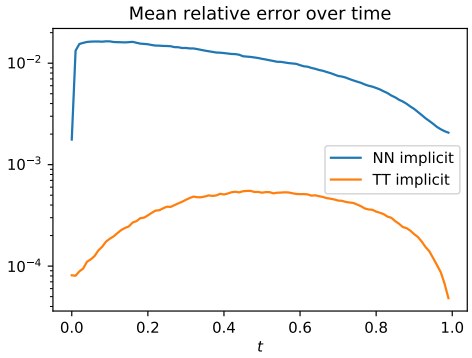


Figure 5.17: Mean relative error for TT and NN attempts.

The accuracy of the TT approximations is surprising given that the ansatz functions are constant in space. We further investigate this behavior in Table 5.5 and observe that the required polynomial degree decreases with increasing dimension. While similar "blessings of

dimensionality" have been reported and discussed (see, for instance, Figure 3 in [Bay+21] and Section 1.3 in [Kho12]), a thorough theoretical understanding is still lacking. To guide intuition, we would like to point out that the phenomenon that high-dimensional systems become in some sense simpler is well known from the theory of interacting particle systems ("propagation of chaos", see [Szn91]): In various scenarios, the joint distribution of a large number of particles tends to approximately factorize as the number of particles increases (that is, as the dimensionality of the joint state space grows large). It is plausible that similar approximate factorizations are relevant for high-dimensional PDEs and that tensor methods are useful (i) to detect this effect and (ii) to exploit it. In this experiment, the black-box nature of neural networks does not appear to reveal such properties.

d	Polynomial degree				
	0	1	2	3	4
1	$3.62\mathrm{e}^{-1}$	$3.60\mathrm{e}^{-1}$	$2.47\mathrm{e}^{-3}$	$3.86\mathrm{e}^{-4}$	$4.27\mathrm{e}^{-2}$
2	$1.03\mathrm{e}^{-1}$	$1.02\mathrm{e}^{-1}$	$1.87\mathrm{e}^{-2}$	$1.79\mathrm{e}^{-2}$	$1.79\mathrm{e}^{-2}$
5	$1.55\mathrm{e}^{-2}$	$1.54\mathrm{e}^{-2}$	$1.03\mathrm{e}^{-3}$	$9.52\mathrm{e}^{-4}$	$1.96\mathrm{e}^{-2}$
10	$2.84\mathrm{e}^{-3}$	$2.86\mathrm{e}^{-3}$	$1.37\mathrm{e}^{-3}$	$1.34\mathrm{e}^{-3}$	$1.10\mathrm{e}^{-1}$
50	$1.17\mathrm{e}^{-4}$	$1.29\mathrm{e}^{-4}$	$2.79\mathrm{e}^{-4}$	$3.35\mathrm{e}^{-4}$	$6.96\mathrm{e}^{-5}$
100	$5.90\mathrm{e}^{-5}$	$4.99\mathrm{e}^{-5}$	$8.65\mathrm{e}^{-5}$	$1.23\mathrm{e}^{-4}$	$3.62\mathrm{e}^{-5}$

Table 5.5: Relative errors of the TT approximations $\widehat{V}_n(x_0)$ for different dimensions and polynomial degrees.

5.4.4.2 HJB with Double Well Dynamics

In another example we consider again an HJB equation, however this time making the drift in the dynamics non-linear, as suggested in [NR20]. The PDE becomes

$$(\partial_t + L)\, V(x,t) - \frac{1}{2}|(\sigma^\top \nabla V)(x,t)|^2 = 0,$$
$$V(x,T) = g(x),$$

with L as in (5.27), where now the drift is given as the gradient of the double well potential

$$b = -\nabla \Psi, \qquad \Psi(x) = \sum_{i,j=1}^{d} C_{ij}(x_i^2 - 1)(x_j^2 - 1),$$

and the terminal condition is $g(x) = \sum_{i=1}^{d} \nu_i (x_i - 1)^2$ for $\nu_i > 0$. Similarly as before a reference solution is available,

$$V(x,t) = -\log \mathbb{E}\left[e^{-g(X_T)} \middle| X_t = x \right], \tag{5.39}$$

where X_t is the forward diffusion as specified in (5.29) (see again Appendix A.2.1 for details).

First, we consider diagonal matrices $C = 0.1\,\mathrm{Id}_{d\times d}$, $\sigma = \sqrt{2}\,\mathrm{Id}_{d\times d}$, implying that the dimensions do not interact, and take $T = 0.5$, $d = 50$, $\Delta t = 0.01$, $K = 2000$, $\nu_i = 0.05$. We set the TT-rank to 2, use polynomial degree 3 and refer to Appendix A.2 for further details

on the TT and NN configurations. Since in the solution of the PDE the dimensions do not interact either, we can compute a reference solution with finite differences. In Table 5.6 we see that the TT and NN approximations are compatible with TTs having an advantage in computational time. We assume that the TT result could possibly be improved by choosing a better fit of ansatz functions, as due to the local behavior of the Double Well potential non-global ansatz functions might be a better choice.

	$\mathrm{TT}_{\mathrm{impl}}$	$\mathrm{NN}_{\mathrm{impl}}$
$\widehat{V}_0(x_0)$	9.6876	9.6942
relative error	$1.41\mathrm{e}^{-3}$	$7.27\mathrm{e}^{-4}$
reference loss	$1.36\mathrm{e}^{-3}$	$4.25\mathrm{e}^{-3}$
PDE loss	$3.62\mathrm{e}^{-2}$	$2.66\mathrm{e}^{-1}$
computation time	95	1987

Table 5.6: Approximation results for the HJB equation with non-interacting double well potential in $d = 50$.

Let us now consider a non-diagonal matrix $C = \mathrm{Id}_{d\times d} + (\xi_{ij})$, where $\xi_{ij} \sim \mathcal{N}(0, 0.01)$ are sampled once at the beginning of the experiment and further choose $\sigma = \sqrt{2}$, $\mathrm{Id}_{d\times d}$, $\nu_i = 0.5$, $T = 0.3$. We aim at the solution at $x_0 = (-1, \dots, -1)^\top$ and compute a reference solution with (5.39) using 10^7 samples. We see in Table 5.7 that TTs are much faster than NNs, while yielding a similar performance. Note that due to the non-diagonality of C it is expected that the TTs are of rank larger than 2. For the explicit case we do not cap the ranks of the TT and the rank-adaptive solver SALSA, cf. Section 4.8.1, finds ranks of mostly 4 and never larger than 6. Motivated by these results we cap the ranks at $r_i \leq 6$ in the implicit case and indeed they are obtained for nearly every dimension, as seen from the ranks below

$$[5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5].$$

The results were obtained with polynomial degree 7.

	$\mathrm{TT}_{\mathrm{impl}}$	$\mathrm{TT}_{\mathrm{expl}}$	$\mathrm{NN}_{\mathrm{impl}}$
$\widehat{V}_0(x_0)$	35.015	34.756	34.917
relative error	$1.52\mathrm{e}^{-3}$	$2.82\mathrm{e}^{-3}$	$4.24\mathrm{e}^{-3}$
reference loss	$1.30\mathrm{e}^{-2}$	$1.59\mathrm{e}^{-2}$	$6.38\mathrm{e}^{-2}$
PDE loss	79.9	341	170.64
computation time	460	15	16991

Table 5.7: Approximation results for the HJB equation with interacting double well potential in $d = 20$.

5.4.4.3 Cox–Ingersoll–Ross Model

Our last example is taken from financial mathematics. As suggested in [JL21] we consider a bond price in a multidimensional Cox–Ingersoll–Ross (CIR) model, see also [Hyn07; Alf+15].

The underlying PDE is specified as

$$\partial_t V(x,t) + \frac{1}{2}\sum_{i,j=1}^{d} \sqrt{x_i x_j}\gamma_i\gamma_j \partial_{x_i}\partial_{x_j} V(x,t) + \sum_{i=1}^{d} a_i(b_i - x_i)\partial_{x_i} V(x,t) - \left(\max_{1\le i\le d} x_i\right) V(x,t) = 0.$$

Here, $a_i, b_i, \gamma_i \in [0,1]$ are uniformly sampled at the beginning of the experiment and we choose $V(T,x) = 1$. We set $d = 100$.

We aim to estimate the bond price at the initial condition $x_0 = (1,\dots,1)^\top$. As there is no reference solution known, we rely on the PDE loss to compare our results. Table 5.8 shows that all three approaches yield similar results, while having a rather small PDE loss. For this test it is again sufficient to set the TT-rank to 1 and polynomial degree of 3. The TT approximations seem to be slightly better and we note that the explicit TT scheme is again much faster.

	$\mathrm{TT}_{\mathrm{impl}}$	$\mathrm{TT}_{\mathrm{expl}}$	$\mathrm{NN}_{\mathrm{impl}}$
$\widehat{V}_0(x_0)$	0.312	0.306	0.31087
PDE loss	$5.06\mathrm{e}^{-4}$	$5.04\mathrm{e}^{-4}$	$7.57\mathrm{e}^{-3}$
computation time	5281	197	9573

Table 5.8: $K = 1000$, $d = 100$, $x_0 = [1,1,\dots,1]$

In Table 5.9 we compare the PDE loss using different polynomial degrees for the TT ansatz function and see that we do not get any improvements with polynomials of degree larger than 1.

	Polynom. degree			
	0	1	2	3
$\widehat{V}_0(x_0)$	0.294	0.312	0.312	0.312
PDE loss	$9.04\mathrm{e}^{-2}$	$7.80\mathrm{e}^{-4}$	$1.05\mathrm{e}^{-3}$	$5.06\mathrm{e}^{-4}$
comp. time	110	3609	4219	5281

Table 5.9: PDE loss and computation time for TTs with different polynomial degrees

Noticing the similarity between the results for polynomial degrees 1, 2, and 3, we further investigate by computing the value function along a sample trajectory in Figure 5.18, where we see that indeed the approximations with those polynomial degrees are indistinguishable.

5.4.5 Short Conclusion and Outlook

We have demonstrated the applicability of our tensor-train approach for state spaces in up to 100 dimensions. Further, we have demonstrated that tensor-train approaches can not only produce results with similar accuracy than state-of-the-art neural network approaches, but also produce them in a faster manner. A critical comparison - or combination - of both methods is important and necessary.

The proposed method can be used for solving open-loop stochastic optimal control problems using a methods similar to the Policy Iteration. Starting at the point x, we can compute for

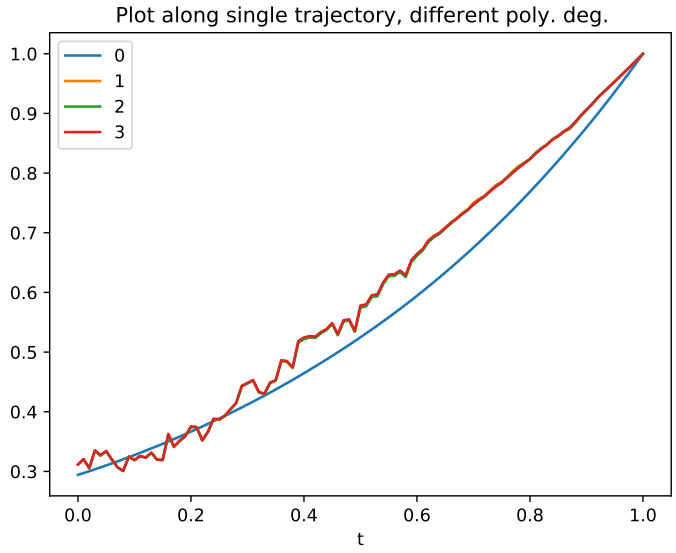


Figure 5.18: Reference trajectory for different polynomial degrees.

a feedback law $v^{\alpha}(t, x)$ along trajectories. This value function can then be used to compute new trajectories in an iterative fashion, which yields a non-global (because it is only accurate along the trajectories) controller.

Chapter 6

Conclusion and Outlook

We have considered the problem of finding and representing high-dimensional functions in different control-related contexts. The representation problem is solved by multi-polynomials combined with hierarchical tensor compression models. All problems have in common, that the underlying non-linear equations can be reduced to a sequence of linear integral equations with an iterative scheme. These equations are approximated using Monte Carlo quadrature. As model problems we considered examples where a smooth solution can be expected and we have experimentally shown that for these problems our approach is successful. We have compared our results to optimal reference values and observed that the approximations are accurate. Note that in the end of the individual sections of Chapter 5 we have already given short conclusions regarding the specialized methods that are described within the sections.

The success of multi-polynomials in combination with tensor-trains for regression problems opens the door to several optimization tasks of regression-type. We have shown that tensor-train approaches can challenge state-of-the-art neural network approaches for high-dimensional regression tasks. Further experimental comparison in different machine learning tasks are certainly of interest. An explanation for the performance of our tensor approach versus the neural network approach is that the ALS algorithm is specialized towards regression-type problems. However, for more challenging optimization tasks, the generality of gradient descents in combination with automatic differentiation as it is used in machine learning libraries can yield improved performance. Another explanation is that while the function spaces that our TT models can encode are very clear, the non-linearity of neural network approaches yields a richer function class. Finding the optimal representant within a richer function class is of course more challenging. However, if both function classes can approximate the solution, then having a richer function class is not beneficial. In the future, a critical comparison of both approaches not only for model problems, but also for large-scale problems is of interest.

For the application of the regression task in our context several extensions are possible. On the one hand recent refinements of the tensor-train model with block-sparse component tensors [GST21] can be utilized. On the other hand an adaption of the ALS algorithm utilizing a refined regularization method as proposed in [Tru21] can be beneficial.

As we considered semi model-free controlling, a natural next step is true model-free controlling. In that context the actor-critic approach known from reinforcement learning is a natural next step.

While the method proved to be successful for our model problems, scaling the size of the experiments to real-world applications is certainly of interest. Another challenging task is extending the current framework, to state-constrained and other restricted control problems. Here, the value function cannot be expected to be smooth and thus polynomials might not be an appropriate ansatz space.

Bibliography

[Aba+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al.
"Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[Abd+16] A. Abdelfattah, M. Baboulin, V. Dobrev, J. Dongarra, C. Earl, J. Falcou, A. Haidar, I. Karlin, T. Kolev, I. Masliah, et al.
"High-performance tensor contractions for GPUs".
In: *Procedia Computer Science* 80 (2016), pp. 108–118.

[AK07] S. M. Aseev and A. V. Kryazhimskii.
"The Pontryagin maximum principle and optimal economic growth problems".
In: *Proceedings of the Steklov institute of mathematics* 257.1 (2007), pp. 1–255.

[AKK21] B. Azmi, D. Kalise, and K. Kunisch. "Optimal Feedback Law Recovery by Gradient-Augmented Sparse Polynomial Regression."
In: *J. Mach. Learn. Res.* 22 (2021), pp. 48–1.

[Alf+15] A. Alfonsi et al.
Affine diffusions and related processes: simulation, theory and applications.
Vol. 6. Springer, 2015.

[Arn98] L. Arnold. *Random Dynamical Systems*. eng. 1998.

[AS95] L. Arnold and M. Scheutzow.
"Perfect cocycles through stochastic differential equations".
In: *Probability Theory and Related Fields* 101 (1995), pp. 65–88.

[Atk+94] C. G. Atkeson et al. "Using local trajectory optimizers to speed up global optimization in dynamic programming".
In: *Advances in neural information processing systems* (1994), pp. 663–663.

[Bay+21] C. Bayer, M. Eigel, L. Sallandt, and P. Trunschke.
"Pricing high-dimensional Bermudan options with hierarchical tensor formats".
In: *arXiv preprint arXiv:2103.01934* (2021).

[BC97] M. Bardi and I. Capuzzo-Dolcetta.
Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations.
Boston: Birkäuser, 01/1997.

[BD97] M. Bardi and F. Da Lio.
"On the Bellman equation for some unbounded control problems". In: *Nonlinear Differential Equations and Applications NoDEA* 4.4 (1997), pp. 491–510.

[BDS19] T. Breiten, S. Dolgov, and M. Stoll. "Solving differential Riccati equations: A nonlinear space-time method using tensor trains".
In: *arXiv preprint arXiv:1912.06944* (2019).

[Bec+19] C. Beck, S. Becker, P. Cheridito, A. Jentzen, and A. Neufeld.
"Deep splitting method for parabolic PDEs".
In: *arXiv preprint arXiv:1907.03452* (2019).

[Bel57] R. Bellman. *Dynamic programming*. Mineola, N.Y: Dover Publications, 1957.

[Bel58] R. Bellman. "On a routing problem".
In: *Quarterly of Applied Mathematics* 16.1 (04/1958), pp. 87–90.

[Bel73] E. Beltrami. "Sulle funzioni bilineari". In: *Giornale di Matematiche ad Uso degli Studenti Delle Universita* 11.2 (1873), pp. 98–106.

[Ben+20] P. Benner, Z. Bujanović, P. Kürschner, and J. Saak.
"A Numerical Comparison of Different Solvers for Large-Scale, Continuous-Time Algebraic Riccati Equations and LQR Problems".
In: *SIAM Journal on Scientific Computing* 42.2 (01/2020), A957–A996.

[Bis73] J.-M. Bismut. "Conjugate convex functions in optimal stochastic control".
In: *Journal of Mathematical Analysis and Applications* 44.2 (1973), pp. 384–404.

[Bis78] J.-M. Bismut.
"An Introductory Approach to Duality in Optimal Stochastic Control".
In: *SIAM Review* 20.1 (01/1978), pp. 62–78.

[BMM12] M. Budišić, R. Mohr, and I. Mezić. "Applied Koopmanism".
In: *Chaos: An Interdisciplinary J. of Nonlinear Science* 22.4 (2012), p. 047510.

[BN16] R. Buckdahn and T. Nie.
"Generalized Hamilton–Jacobi–Bellman Equations with Dirichlet Boundary Condition and Stochastic Exit Time Optimal Control Problem".
In: *SIAM Journal on Control and Optimization* 54.2 (01/2016), pp. 602–631.

[BT04] B. Bouchard and N. Touzi. "Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations".
In: *Stochastic Processes and their applications* 111.2 (2004), pp. 175–206.

[CL83] M. G. Crandall and P.-L. Lions.
"Viscosity solutions of Hamilton-Jacobi equations".
In: *Transactions of the American Mathematical Society* 277.1 (01/1983), pp. 1–1.

[ČMM19] N. Črnjarić-Žic, S. Maćešić, and I. Mezić.
"Koopman operator spectrum for random dynamical systems".
In: *Journal of Nonlinear Science* (2019), pp. 1–50.

[Con+15] F. Cong, Q.-H. Lin, L.-D. Kuang, X.-F. Gong, P. Astikainen, and T. Ristaniemi.
"Tensor decomposition of EEG signals: a brief review".
In: *Journal of neuroscience methods* 248 (2015), pp. 59–69.

[Cor07] J.-M. Coron. *Control and nonlinearity*. 136. American Mathematical Soc., 2007.

[Da 00] F. Da Lio. "On the Bellman Equation for Infinite Horizon Problems with Unbounded Cost Functional".
In: *Applied Mathematics and Optimization* 41.2 (04/2000), pp. 171–197.

[DDV00] L. De Lathauwer, B. De Moor, and J. Vandewalle.
"A multilinear singular value decomposition". In: *SIAM journal on Matrix Analysis and Applications* 21.4 (2000), pp. 1253–1278.

[DL08] V. De Silva and L.-H. Lim.
"Tensor rank and the ill-posedness of the best low-rank approximation problem".
In: *SIAM Journal on Matrix Analysis and Applications* 30.3 (2008),
pp. 1084–1127.

[Don+95] A. L. Dontchev, W. W. Hager, A. B. Poore, and B. Yang.
"Optimality, stability, and convergence in nonlinear control".
In: *Applied Mathematics and Optimization* 31.3 (05/1995), pp. 297–326.

[Doo81] P. V. Dooren.
"A Generalized Eigenvalue Approach for Solving Riccati Equations". In: *SIAM Journal on Scientific and Statistical Computing* 2.2 (06/1981), pp. 121–135.

[EHJ17] W. E, J. Han, and A. Jentzen.
"Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations".
In: *Communications in Mathematics and Statistics* 5.4 (2017), pp. 349–380.

[Eig+19] M. Eigel, R. Schneider, P. Trunschke, and S. Wolf.
"Variational Monte Carlo—bridging concepts of machine learning and high-dimensional partial differential equations".
In: *Advances in Computational Mathematics* (10/2019).

[EST20] M. Eigel, R. Schneider, and P. Trunschke.
"Convergence bounds for empirical nonlinear least-squares".
In: *arXiv preprint arXiv:2001.00639* (2020).

[EY18] W. E and B. Yu. "The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems".
In: *Communications in Mathematics and Statistics* 6.1 (2018), pp. 1–12.

[EY36] C. Eckart and G. Young.
"The approximation of one matrix by another of lower rank".
In: *Psychometrika* 1.3 (1936), pp. 211–218.

[Fac+20] K. Fackeldey, M. Oster, L. Sallandt, and R. Schneider.
"Approximative Policy Iteration for Exit Time Feedback Control Problems driven by Stochastic Differential Equations using Tensor Train format".
In: *arXiv preprint arXiv:2010.04465* (2020).

[FS06] W. H. Fleming and H. M. Soner.
Controlled Markov processes and viscosity solutions. Vol. 25.
Springer Science & Business Media, 2006.

[Gar72] W. L. Garrard. “Suboptimal feedback control for nonlinear systems”.
In: *Automatica* 8.2 (1972), pp. 219–221.

[Gel+19] P. Gelß, S. Klus, J. Eisert, and C. Schütte.
“Multidimensional approximation of nonlinear dynamical systems”.
In: *Journal of Computational and Nonlinear Dynamics* 14.6 (2019).

[GK19] L. Grasedyck and S. Krämer. “Stable ALS approximation in the TT-format for rank-adaptive tensor completion”.
In: *Numerische Mathematik* 143.4 (2019), pp. 855–904.

[GK65] G. Golub and W. Kahan.
“Calculating the singular values and pseudo-inverse of a matrix”.
In: *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2.2 (1965), pp. 205–224.

[GLW+05] E. Gobet, J.-P. Lemor, X. Warin, et al. “A regression-based Monte Carlo method to solve backward stochastic differential equations”.
In: *The Annals of Applied Probability* 15.3 (2005), pp. 2172–2202.

[Goe+20] A. Goeßmann, M. Götte, I. Roth, R. Sweke, G. Kutyniok, and J. Eisert.
“Tensor network approaches for learning non-linear dynamical laws”.
In: *arXiv preprint arXiv:2002.12388* (2020).

[GP17] L. Grüne and J. Pannek. “Nonlinear model predictive control”.
In: *Nonlinear Model Predictive Control*. Springer, 2017, pp. 45–69.

[GPM89] C. E. Garcia, D. M. Prett, and M. Morari.
“Model predictive control: Theory and practice—A survey”.
In: *Automatica* 25.3 (1989), pp. 335–348.

[GST21] M. Götte, R. Schneider, and P. Trunschke. “A Block-Sparse Tensor Train Format for Sample-Efficient High-Dimensional Polynomial Regression”.
In: *Frontiers in Applied Mathematics and Statistics* 7 (2021), p. 57.

[GV13] G. H. Golub and C. F. Van Loan. “Matrix computations, 4th”.
In: *Johns Hopkins* (2013).

[Hac12] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*. Vol. 42.
Springer, 01/2012.

[Har+13] C. Hartmann, R. Banisch, M. Sarich, T. Badowski, and C. Schütte.
“Characterization of Rare Events in Molecular Dynamics”.
In: *Entropy* 16.1 (12/2013), pp. 350–376.

[Har+17] C. Hartmann, L. Richter, C. Schütte, and W. Zhang.
“Variational characterization of free energy: Theory and algorithms”.
In: *Entropy* 19.11 (2017), p. 626.

[Hås90] J. Håstad. “Tensor rank is NP-complete”.
In: *Journal of algorithms (Print)* 11.4 (1990), pp. 644–654.

[Hit27] F. L. Hitchcock. “The expression of a tensor or a polyadic as a sum of products”.
In: *Journal of Mathematics and Physics* 6.1-4 (1927), pp. 164–189.

[HK09] W. Hackbusch and S. Kühn.
"A New Scheme for the Tensor Representation". English.
In: *Journal of Fourier Analysis and Applications* 15.5 (2009), pp. 706–722.

[HK10] R. Herzog and K. Kunisch. "Algorithms for PDE-constrained optimization".
In: *GAMM-Mitteilungen* 33.2 (2010), pp. 163–176.

[How60] R. A. Howard. *Dynamic Programming and Markov Processes.*
Cambridge, MA: MIT Press, 1960.

[HPW20] C. Huré, H. Pham, and X. Warin.
"Deep backward schemes for high-dimensional nonlinear PDEs".
In: *Mathematics of Computation* 89.324 (2020), pp. 1547–1579.

[HR21] C. Hartmann and L. Richter.
"Nonasymptotic bounds for suboptimal importance sampling".
In: *arXiv preprint arXiv:2102.09606* (2021).

[HRS12a] S. Holtz, T. Rohwedder, and R. Schneider. "The Alternating Linear Scheme for Tensor Optimization in the Tensor Train Format".
In: *SIAM J. Sci. Comput.* 34.2 (2012), A683–A713.
eprint: `https://doi.org/10.1137/100818893`.

[HRS12b] S. Holtz, T. Rohwedder, and R. Schneider.
"On manifolds of tensors of fixed TT-rank".
In: *Numerische Mathematik* 120.4 (2012), pp. 701–731.

[HS06] W. H. Fleming and H. Soner. *Control Markov Processes and Viscosity Solutions.*
2nd ed. Springer, 2006.

[HS97] S. Huo and J. E. Straub.
"The MaxFlux algorithm for calculating variationally optimized reaction paths for conformational transitions in many body systems at finite temperature".
In: *The Journal of Chemical Physics* 107.13 (10/1997), pp. 5000–5006.

[Hua+17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger.
"Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 4700–4708.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment".
In: *Computing in science & engineering* 9.03 (2007), pp. 90–95.

[HW17] B. Huber and S. Wolf. *Xerus - A General Purpose Tensor Library.*
https://libxerus.org/. 2014–2017.

[Hyn07] C. B. Hyndman. "Forward-backward SDEs and the CIR model".
In: *Statistics & probability letters* 77.17 (2007), pp. 1676–1682.

[JL21] Y. Jiang and J. Li. "Convergence of the Deep BSDE method for FBSDEs with non-Lipschitz coefficients". In: *arXiv preprint arXiv:2101.01869* (2021).

[Jor74] C. Jordan. "Mémoire sur les formes bilinéaires."
In: *Journal de mathématiques pures et appliquées* 19 (1874), pp. 35–54.

[Kan48] L. V. Kantorovich. "Functional analysis and applied mathematics".
In: *Uspekhi Matematicheskikh Nauk* 3.6 (1948), pp. 89–185.

[KB09] T. G. Kolda and B. W. Bader. “Tensor decompositions and applications”. In: *SIAM review* 51.3 (2009), pp. 455–500.

[KB14] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[KG19] S. Klus and P. Gelß. “Tensor-based algorithms for image classification”. In: *Algorithms* 12.11 (2019), p. 240.

[Kho12] B. N. Khoromskij. “Tensors-structured numerical methods in scientific computing: Survey on recent advances”. In: *Chemometrics and Intelligent Laboratory Systems* 110.1 (2012), pp. 1–19.

[KK18] D. Kalise and K. Kunisch. “Polynomial approximation of high-dimensional Hamilton-Jacobi-Bellman equations and applications to feedback control of semilinear parabolic PDEs”. In: *SIAM J. Sci. Comput.* 40.2 (2018), A629–A652.

[KKF11] B. N. Khoromskij, V. Khoromskaia, and H.-J. Flad. “Numerical solution of the Hartree–Fock equation in multilevel tensor-structured format”. In: *SIAM journal on scientific computing* 33.1 (2011), pp. 45–65.

[KKS11] V. Khoromskaia, B. Khoromskij, and R. Schneider. “QTT representation of the Hartree and exchange operators in electronic structure calculations”. In: *Computational methods in applied mathematics* 11.3 (2011), pp. 327–341.

[KKS16] S. Klus, P. Koltai, and C. Schütte. “On the numerical approximation of the Perron-Frobenius and Koopman operator”. In: *J. of Computational Dynamics* 3.2158-2491-2016-1-51 (2016), p. 51.

[Kle12] F. C. Klebaner. *Introduction to Stochastic Calculus with Applications*. 3rd. IMPERIAL COLLEGE PRESS, 2012. eprint: `https://www.worldscientific.com/doi/pdf/10.1142/p821`.

[Kle70] D. Kleinman. “An easy way to stabilize a linear constant system”. In: *IEEE Transactions on Automatic Control* 15.6 (1970), pp. 692–692.

[Koo31] B. O. Koopman. “Hamiltonian Systems and Transformation in Hilbert Space”. In: *Proc. of the National Academy of Sciences* 17.5 (1931), pp. 315–318. eprint: `https://www.pnas.org/content/17/5/315.full.pdf`.

[KP92] P. E. Kloeden and E. Platen. “Stochastic differential equations”. In: *Numerical Solution of Stochastic Differential Equations*. Springer, 1992, pp. 103–160.

[KS98] I. Karatzas and S. E. Shreve. *Brownian Motion and Stochastic Calculus*. Springer, 1998.

[Kut+19] G. Kutyniok, P. Petersen, M. Raslan, and R. Schneider. “A theoretical analysis of deep neural networks and parametric PDEs”. In: *arXiv preprint arXiv:1904.00377* (2019).

[Las94] A. Lasota. *Chaos, fractals, and noise : stochastic aspects of dynamics*. eng. 2. ed. Applied mathematical sciences BV000005274 97. Springer, 1994.

[Lau79] A. Laub. “A Schur method for solving algebraic Riccati equations”. In: *IEEE Transactions on Automatic Control* 24.6 (12/1979), pp. 913–921.

[Leb+14] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. “Speeding-up convolutional neural networks using fine-tuned cp-decomposition”. In: *arXiv preprint arXiv:1412.6553* (2014).

[LS12] L. Ljungqvist and T. J. Sargent. *Recursive Macroeconomic Theory, third edition* -. Cambridge: MIT Press, 2012.

[Miw+04] F. Miwakeichi, E. Martınez-Montes, P. A. Valdés-Sosa, N. Nishiyama, H. Mizuhara, and Y. Yamaguchi. “Decomposing EEG data into space–time–frequency components using parallel factor analysis”. In: *NeuroImage* 22.3 (2004), pp. 1035–1045.

[Mni+15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[MST17] K. Mills, M. Spanner, and I. Tamblyn. “Deep learning and the Schrödinger equation”. In: *Physical Review A* 96.4 (2017), p. 042113.

[NR20] N. Nüsken and L. Richter. “Solving high-dimensional Hamilton-Jacobi-Bellman PDEs using neural networks: perspectives from the theory of controlled diffusions and measures on path space”. In: *arXiv preprint arXiv:2005.05409* (2020).

[Oks13] B. Oksendal. *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[Oli06] T. E. Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[Ose11] I. Oseledets. “Tensor-Train Decomposition”. In: *SIAM J. Sci. Comput.* 33 (01/2011), pp. 2295–2317.

[OSS21a] M. Oster, L. Sallandt, and R. Schneider. “Approximating optimal feedback controllers of finite horizon control problems using hierarchical tensor formats”. In: *arXiv preprint arXiv:2104.06108* (2021).

[OSS21b] M. Oster, L. Sallandt, and R. Schneider. *Approximating the Stationary Bellman Equation by Hierarchical Tensor Products*. 2021. arXiv: `1911.00279 [math.OC]`.

[Par+03] S. Park, M. K. Sener, D. Lu, and K. Schulten. “Reaction paths based on mean first-passage times”. In: *The Journal of Chemical Physics* 119.3 (07/2003), pp. 1313–1319.

[Par98] É. Pardoux. “Backward stochastic differential equations and viscosity solutions of systems of semilinear parabolic and elliptic PDEs of second order”. In: *Stochastic Analysis and Related Topics VI*. Springer, 1998, pp. 79–127.

[Pas+17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. “Automatic differentiation in pytorch”. In: (2017).

[PB79] M. L. Puterman and S. L. Brumelle.
"On the Convergence of Policy Iteration in Stationary Dynamic Programming".
In: *Mathematics of Operations Research* 4.1 (1979), pp. 60–69.

[Pen92] S. Peng. "A Generalized dynamic programming principle and hamilton-jacobi-bellman equation".
In: *Stochastics and Stochastic Reports* 38.2 (1992), pp. 119–134.
eprint: `https://doi.org/10.1080/17442509208833749`.

[Pon+62] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The mathematical theory of optimal processes*. New York, NY: Translated from the Russian by K. N. Trirogoff; edited by L. W.Neustadt. Wiley, 1962.

[PP90] E. Pardoux and S. Peng.
"Adapted solution of a backward stochastic differential equation".
In: *Systems & Control Letters* 14.1 (1990), pp. 55–61.

[QB03] S. J. Qin and T. A. Badgwell.
"A survey of industrial model predictive control technology".
In: *Control engineering practice* 11.7 (2003), pp. 733–764.

[Ric21] L. Richter. "Solving high-dimensional PDEs, approximation of path space measures and importance sampling of diffusions".
dissertation. BTU Cottbus-Senftenberg, 2021.

[RPK19] M. Raissi, P. Perdikaris, and G. E. Karniadakis.
"Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations".
In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[RSN21] L. Richter, L. Sallandt, and N. Nüsken.
"Solving high-dimensional parabolic PDEs using the tensor train format".
In: *Proceedings of the 38th International Conference on Machine Learning*.
Ed. by M. Meila and T. Zhang. Vol. 139.
Proceedings of Machine Learning Research. PMLR, 18–24 Jul/2021,
pp. 8998–9009.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*.
MIT press, 2018.

[Sch07] H.-J. Schmeisser. "Recent developments in the theory of function spaces with dominating mixed smoothness".
In: *Nonlinear Analysis, Function Spaces and Applications* (2007), pp. 145–204.

[Sch11] U. Schollwöck.
"The density-matrix renormalization group in the age of matrix product states".
In: *Annals of physics* 326.1 (2011), pp. 96–192.

[SG77] G. N. Saridis and C. S. G. Lee.
"Optimal control approximations for trainable manipulators". In: *1977 IEEE Conference on Decision and Control including the 16th Symposium on Adaptive Processes and A Special Symposium on Fuzzy Set Theory and Applications*.
12/1977, pp. 749–754.

[Sid+17] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos.
"Tensor decomposition for signal processing and machine learning".
In: *IEEE Transactions on Signal Processing* 65.13 (2017), pp. 3551–3582.

[Sob67] I. M. Sobol'. "On the distribution of points in a cube and the approximate evaluation of integrals". In: *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* 7.4 (1967), pp. 784–802.

[Son90] E. D. Sontag. "Feedback stabilization of nonlinear systems".
In: *Robust control of linear systems and nonlinear control* (1990), pp. 61–81.

[Son98] E. D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems (2nd Ed.)* Berlin, Heidelberg: Springer-Verlag, 1998.

[SS16] E. M. Stoudenmire and D. J. Schwab.
"Supervised learning with quantum-inspired tensor networks".
In: *arXiv preprint arXiv:1605.05775* (2016).

[Ste16] M. Steinlechner.
"Riemannian optimization for high-dimensional tensor completion".
In: *SIAM Journal on Scientific Computing* 38.5 (2016), S461–S484.

[Ste93] G. W. Stewart. "On the early history of the singular value decomposition".
In: *SIAM review* 35.4 (1993), pp. 551–566.

[SU09] W. Sickel and T. Ullrich. "Tensor products of Sobolev-Besov spaces and applications to approximation from the hyperbolic cross".
In: *Journal of Approximation Theory* 161.2 (2009), pp. 748–786.

[Sza+15] S. Szalay, M. Pfeffer, V. Murg, G. Barcza, F. Verstraete, R. Schneider, and Ö. Legeza. "Tensor product methods and entanglement optimization for ab initio quantum chemistry".
In: *International j. of quantum chemistry* 115.19 (2015), pp. 1342–1391.

[Szn91] A.-S. Sznitman. "Topics in propagation of chaos".
In: *Ecole d'été de probabilités de Saint-Flour XIX—1989*. Springer, 1991, pp. 165–251.

[Trö10] F. Tröltzsch. *Optimal control of partial differential equations: theory, methods, and applications*. Vol. 112. American Mathematical Soc., 2010.

[Tru21] P. Trunschke. "Convergence bounds for nonlinear least squares and applications to tensor recovery". In: *arXiv preprint arXiv:2108.05237* (2021).

[Tuc66] L. R. Tucker. "Some mathematical notes on three-mode factor analysis".
In: *Psychometrika* 31.3 (1966), pp. 279–311.

[UV13] A. Uschmajew and B. Vandereycken.
"The geometry of algorithms using hierarchical tensors".
In: *Linear Algebra and its Applications* 439.1 (2013), pp. 133–166.

[VCV11] S. Van Der Walt, S. C. Colbert, and G. Varoquaux.
"The NumPy array: a structure for efficient numerical computation".
In: *Computing in Science & Engineering* 13.2 (2011), p. 22.

[Vir+20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. 1. 0. Contributors.
"SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python".
In: *Nature Methods* 17 (2020), pp. 261–272.

[Whi92] S. R. White. "Density matrix formulation for quantum renormalization groups".
In: *Physical review letters* 69.19 (1992), p. 2863.

[Wol19] A. S. J. W. Wolf. "Low rank tensor decompositions for high dimensional data approximation, recovery and prediction".
Doctoral Thesis. Berlin: Technische Universität Berlin, 2019.

[Xu+11] Y. Xu, R. Gu, H. Zhang, W. Xu, and J. Duan. "Stochastic bifurcations in a bistable Duffing–Van der Pol oscillator with colored noise".
In: *Physical Review E* 83.5 (2011), p. 056215.

[YD10] D. Yu and L. Deng. "Deep learning and its applications to signal and information processing [exploratory dsp]".
In: *IEEE Signal Processing Magazine* 28.1 (2010), pp. 145–154.

[YKT17] Y. Yang, D. Krompass, and V. Tresp.
"Tensor-train recurrent neural networks for video classification".
In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3891–3900.

[Zha04] J. Zhang. "A numerical scheme for BSDEs".
In: *The annals of applied probability* 14.1 (2004), pp. 459–488.

[Zha17] J. Zhang. *Backward stochastic differential equations*. Springer, 2017.

[ZHL21] M. Zhou, J. Han, and J. Lu.
"Actor-Critic Method for High Dimensional Static Hamilton–Jacobi–Bellman Partial Differential Equations based on Neural Networks".
In: *arXiv preprint arXiv:2102.11379* (2021).

[Zho+13] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov.
"Value function approximation and model predictive control".
In: *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE. 2013, pp. 100–107.

Appendix A

FBSDE additional information

A.1 Implementation Details

For the evaluation of our approximations we rely on reference values of $V(x_0, 0)$ and further define the following two loss metrics, which are zero if and only if the PDE is fulfilled along the samples generated by the discrete forward SDE (5.32). In the spirit of [RPK19], we define the *PDE loss* as

$$\mathcal{L}_{\text{PDE}} = \frac{1}{KN} \sum_{n=1}^{N} \sum_{k=1}^{K} \Big((\partial_t + L) V(\widehat{X}_n^{(k)}, t_n) + h(\widehat{X}_n^{(k)}, t_n, V(\widehat{X}_n^{(k)}, t_n), (\sigma^\top \nabla V)(\widehat{X}_n^{(k)}, t_n)) \Big)^2,$$

where $\widehat{X}_n^{(k)}$ are realizations of (5.32), the time derivative is approximated with finite differences and the space derivatives are computed analytically (or with automatic differentiation tools). We leave out the first time step $n = 0$ since the regression problem within the explicit and the implicit schemes for the tensor-trains are not well-defined due to the fact that $\widehat{X}_0^k = x_0$ has the same value for all k. We still obtain a good approximation since the added regularization term brings a minimum norm solution with the correct point value $V(x_0, 0)$. Still, this does not aim at the PDE being entirely fulfilled at this point in time.

Further, we define the *relative reference loss* as

$$\mathcal{L}_{\text{ref}} = \frac{1}{K(N+1)} \sum_{n=0}^{N} \sum_{k=1}^{K} \left| \frac{V(\widehat{X}_n^{(k)}, t_n) - V_{\text{ref}}(\widehat{X}_n^{(k)}, t_n)}{V_{\text{ref}}(\widehat{X}_n^{(k)}, t_n)} \right|,$$

whenever a reference solution for all x and t is available.

All computation times in the reported tables are measured in seconds.

Our experiments have been performed on a desktop computer containing an AMD Ryzen Threadripper **2990** WX **32**x **3.00** GHz mainboard and an NVIDIA Titan RTX GPU, where we note that only the NN optimizations were run on this GPU, since our TT framework does not include GPU support. It is expected that running the TT approximations on a GPU will improve time performances in the future [Abd+16].

A.1.1 Details on Neural Network Approximation

For the neural network architecture we rely on the *DenseNet*, which consists of fully-connected layers with additional skip connections as for instance suggested in [EY18] and being rooted in [Hua+17]. To be precise, we define a version of the *DenseNet* that includes the terminal condition of the PDE (5.26) as an additive extension by

$$\Phi_\varrho(x) = A_L x_L + b_L + \theta g(x),$$

where x_L is specified recursively as

$$y_{l+1} = \varrho(A_l x_l + b_l), \qquad x_{l+1} = (x_l, y_{l+1})^\top$$

for $1 \leq l \leq L-1$ with $A_l \in \mathbb{R}^{r_l \times \sum_{i=0}^{l-1} r_i}, b_l \in \mathbb{R}^l, \theta \in \mathbb{R}$ and $x_1 = x$. The collection of matrices A_l, vectors b_l and the coefficient θ comprises the learnable parameters, and we introduce the vector $r := (d_{\text{in}}, r_1, \ldots, r_{L-1}, d_{\text{out}})$ to represent a certain choice of a DenseNet architecture, where in our setting $d_{\text{in}} = d$ and $d_{\text{out}} = 1$. If not otherwise stated we fix the parameter θ to be 1. For the activation function $\varrho : \mathbb{R} \to \mathbb{R}$, that is to be applied componentwise, we choose $\tanh$. For more information and discussion on this setup we refer to [Ric21].

For the gradient descent optimization we choose the Adam optimizer with the default parameters $\beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$ [KB14]. In most of our experiments we chose a fixed learning rate η_{N-1} for the approximation of the first backward iteration step to approximate $\widehat{V}_{N-1}$ and another fixed learning rate η_n for all the other iteration steps to approximate $\widehat{V}_n$ for $0 \leq n \leq N-2$ (cf. Remark 48). Similarly, we denote with G_{N-1} and G_n the amount of gradient descent steps in the corresponding optimizations.

In Tables A.1 and A.2 we list our hyperparameter choices for the neural network experiments that we have conducted.

A.1.2 Details on Tensor-Train Approximation

Within the optimization we have to specify the regularization parameter for the adaptive regularization strategy as noted in Remark 4.8, which we denote here by $\eta > 0$. We adapt this parameter in dependence of the current residual in the regression problem (5.37), i.e. $\eta = cw$, where $c > 0$ and w is the residual from the previous sweep of SALSA. In all our experiments we set $c_\eta = 1$. Further, we have to specify the condition "*not converged* is *true*" within Algorithm 8. This is in particular important, because due to the addition of the final condition to the ansatz space, the orthonormality of the basis is violated. Thus, the usual link between the Frobenius norm of the coefficient tensor and the underlying function space using Parseval's identity does not hold. In order to estimate the error we introduce a test set with equal size as our training set. We measure the residual within a single run of SALSA on the test set and the training set. If the change of the residual on either of this sets is below $\delta = 0.0001$ we set *noChange* = *true*. For the fixed-point iteration we have a two-fold stopping condition. We stop the iteration if either the Frobenius norm of the coefficients has a smaller relative difference than $\gamma_1 < 0.0001$ or if the values $\widehat{V}_n^{k+1}$ and $\widehat{V}_n^k$ and their gradients, evaluated at the points of the test set, have a relative difference smaller than $\gamma_2 < 0.00001$. Note that the second condition is essentially a discrete H^1 norm.

HJB, $d = 10$, $\mathrm{NN}_{\mathrm{impl}}$ Figure 5.16a	HJB, $d = 100$, $\mathrm{NN}_{\mathrm{impl}}$ Table 5.4, Figures 5.16b, 5.17
$K = 2000, \Delta t = 0.01$	$K = 2000, \Delta t = 0.01$
$r = (100, 110, 110, 50, 50, 1)$	$r = (100, 130, 130, 70, 70, 1)$
$G_n = 8000, G_{N-1} = 40000$	$G_n = 5000, G_{N-1} = 40000$
$\eta_n = 0.0001, \eta_{N-1} = 0.0001$	$\eta_n = 0.0001, \eta_{N-1} = 0.0003$

HJB, $d = 100$, $\mathrm{NN}_{\mathrm{expl}}$ Table 5.4, Figures 5.16b, 5.17	HJB double well $d = 50$, $\mathrm{NN}_{\mathrm{impl}}$, Table 5.6
$K = 2000, \Delta t = 0.01$	$K = 2000, \Delta t = 0.01$
$r = (100, 110, 110, 50, 50, 1)$	$r = (50, 30, 30, 1)$
$G_n = 500, G_{N-1} = 7000$	$G_n = 2000, G_{N-1} = 25000$
$\eta_n = 0.00005, \eta_{N-1} = 0.0003$	$\eta_n = 0.0002, \eta_{N-1} = 0.0005$

HJB interacting double well $d = 20$, $\mathrm{NN}_{\mathrm{impl}}$, Table 5.7	CIR, $d = 100$, $\mathrm{NN}_{\mathrm{impl}}$ Table 5.8
$K = 2000, \Delta t = 0.01$	$K = 1000, \Delta t = 0.01$
$r = (50, 20, 20, 20, 20, 1)$	$r = (100, 110, 110, 50, 50, 1)$
$G_n = 3000, G_{N-1} = 30000$	$G_n = 2000$ for $0 \leq n \leq 15$
$\eta_n = 0.0007, \eta_{N-1} = 0.001$	$G_n = 300$ for $16 \leq n \leq N-2$
	$G_{N-1} = 10000$
	$\eta_n = 0.00005, \eta_{N-1} = 0.0001$

Table A.1: Neural network hyperparameters for the BSDE experiments in Section 5.4.4.

PDE with unbounded solution $d = 10$, $\mathrm{NN}_{\mathrm{impl}}$, Table A.3	Allen-Cahn $d = 100$, $\mathrm{NN}_{\mathrm{impl}}$, Table A.4
$K = 1000, \Delta t = 0.001$	$K = 8000, \Delta t = 0.01$
$r = (10, 30, 30, 1)$	$r = (10, 30, 30, 1)$
$G_n = 100, G_{N-1} = 10000$	$G_n = 10000$ for $0 \leq n \leq 5$
$\eta_n = 0.0001, \eta_{N-1} = 0.0001$	$G_n = 6000$ for $6 \leq n \leq N-2$
	$G_{N-1} = 15000$
	$\eta_n = 0.0002, \eta_{N-1} = 0.001$

Table A.2: Neural network hyperparameters for the additional experiments in Section A.2.1.

Finally, we comment on the area $[a, b]$ where the 1-dimensional polynomials are orthonormalized w.r.t. the $H^2([a, b])$ norm, cf. Remark 4.8. We obtain these polynomials by performing a Gram-Schmidt process starting with one-dimensional monomials, cf. Remark 36 Thus, we have to specify the integration area $[a, b]$ for the different tests. In Section 5.4.4.1 we set $a = -6$ and $b = 6$. In Section 5.4.4.2 we set $a = -3$ and $b = 3$ for the case C diagonal and for the interacting case, where C is non-diagonal, we set $a = -8$ and $b = 2$. In Section 5.4.4.3 we choose $a = -0.2$ and $b = 6$.

A.2 Further Numerical Examples

In this section we elaborate on some of the numerical examples from the paper and provide two additional problems.

A.2.1 Hamilton-Jacobi-Bellman Equation

Let us consider the HJB equation from Sections 5.4.4.1 and 5.4.4.2, which we can write as

$$\begin{aligned}(\partial_t + L)\, V(x,t) - \frac{1}{2}|(\sigma^\top \nabla V)(x,t)|^2 &= 0,\\ V(x,T) &= g(x),\end{aligned}$$

in a generic form with the differential operator L being defined in (5.27). We can introduce the exponential transformation $\psi := e^{-V}$ and with the chain rule find that the transformed function fulfills the linear PDE

$$\begin{aligned}(\partial_t + L)\, \psi(x,t) &= 0,\\ \psi(x,T) &= e^{-g(x)}.\end{aligned}$$

This is known as Hopf-Cole transformation, see also [FS06; Har+17]. It is known that via the Feynman-Kac theorem [KS98] the solution to this PDE has the stochastic representation

$$\psi(x,t) = \mathbb{E}\left[e^{-g(X_T)}\middle| X_t = x\right],$$

such that we readily get

$$V(x,t) = -\log \mathbb{E}\left[e^{-g(X_T)}\middle| X_t = x\right],$$

which we can use as a reference solution by approximating the expectation value via Monte Carlo simulation, however keeping in mind that in high dimensions corresponding estimators might have high variances [HR21].

Let us stress again that our algorithms only aim to provide a solution of the PDE along the trajectories of the forward process (5.29). Still, there is hope that our approximations generalize to regions "close" to where samples are available. To illustrate this, consider for instance the d-dimensional forward process

$$X_s = x_0 + \sigma W_s,$$

as for instance in Section 5.4.4.1, where now $\sigma > 0$ is one-dimensional for notational convenience. We know that $X_t \sim \mathcal{N}(x_0, \sigma^2 t\, \mathrm{Id}_{d\times d})$ and therefore note that for the expected distance to the origin it holds

$$\mathbb{E}\left[|X_t - x_0|\right] < \sqrt{\mathbb{E}\left[|X_t - x_0|^2\right]} = \sigma\sqrt{dt}.$$

This motivates evaluating the approximations along the curve

$$X_t = x_0 + \sigma\sqrt{t}\mathbf{1},$$

where $\mathbf{1} = (1,\dots,1)^\top$. Figure A.1 shows that in this case we indeed have good agreement of the approximation with the reference solution when using TTs and that for NNs the deep neural network that we have specified in Table A.1 generalizes worse than a shallower network with only two hidden layers consisting of 30 neurons each.

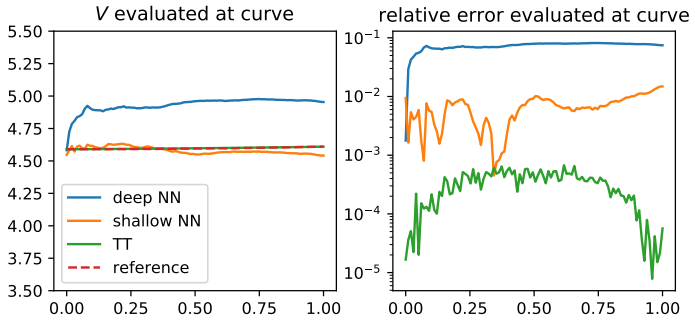


Figure A.1: Approximations of the HJB equation in $d = 100$ evaluated along a representative curve.

A.2.2 PDE with Unbounded Solution

As an additional problem, we choose an example from [HPW20] which offers an analytical reference solution. For the PDE as defined in (5.26) we consider the coefficients

$$b(x,t) = 0,\ \sigma(x,t) = \frac{\mathrm{Id}_{d\times d}}{\sqrt{d}},\ g(x) = \cos\left(\sum_{i=1}^{d} i x_i\right),$$
$$h(x,t,y,z) = k(x) + \frac{y}{2\sqrt{d}}\sum_{i=1}^{d} z_i + \frac{y^2}{2},$$

where, with an appropriately chosen k, a solution can shown to be

$$V(x,t) = \frac{T-t}{d}\sum_{i=1}^{d}\left(\sin(x_i)1_{x_i<0} + x_i 1_{x_i\geq 0}\right) + \cos\left(\sum_{i=1}^{d} i x_i\right).$$

In Table A.3 we compare the results for $d = 10$, $K = 1000$, $T = 1$, $\Delta t = 0.001$ and $x_0 = (0.5, \ldots, 0.5)^\top$. For the TT case it was sufficient to set the ranks to 1 and the polynomial degree to 6. We see that the results are improved significantly if we increase the sample size K from 1000 to 20000. Note that even when increasing the sample size by a factor 20, the computational time is still lower than the NN implementation. It should be highlighted that adding the function g to the neural network (as explained in Appendix A.1) is essential for its convergence in higher dimensions and thereby mitigates the observed difficulties in [HPW20]).

	$\mathrm{TT}_{\mathrm{impl}}$	$\mathrm{TT}^*_{\mathrm{impl}}$	$\mathrm{NN}_{\mathrm{impl}}$
$\widehat{V}_0(x_0)$	-0.1887	-0.2136	-0.2137
relative error	$1.22\mathrm{e}^{-1}$	$6.11\mathrm{e}^{-3}$	$5.50\mathrm{e}^{-3}$
ref loss	$2.47\mathrm{e}^{-1}$	$7.57\mathrm{e}^{-2}$	$3.05\mathrm{e}^{-1}$
abs. ref loss	$2.52\mathrm{e}^{-2}$	$9.29\mathrm{e}^{-3}$	$1.69\mathrm{e}^{-2}$
PDE loss	2.42	0.60	1.38
computation time	360	1778	4520

Table A.3: Approximation results for the PDE with an unbounded analytic solution. For $\mathrm{TT}^*_{\mathrm{impl}}$ we choose $K = 20000$, for the others we choose $K = 1000$.

A.2.3 Allen-Cahn Like Equation

Finally, let us consider the following Allen-Cahn like PDE with a cubic non-linearity in $d = 100$:

$$\begin{aligned}(\partial_t + \Delta)V(x,t) + V(x,t) - V^3(x,t) &= 0,\\ V(x,T) &= g(x),\end{aligned}$$

where we choose $g(x) = \left(2 + \frac{2}{5}|x|^2\right)^{-1}$, $T = \frac{3}{10}$ and are interested in an evaluation at $x_0 = (0,\dots,0)^\top$. This problem has been considered in [EHJ17], where a reference solution of $V(x_0, 0) = 0.052802$ calculated by means of the branching diffusion method is provided. We consider a sample size of $K = 1000$ and a step-size $\Delta t = 0.01$ and provide our approximation results in Table A.4. Note that for this example it is again sufficient to use a TT-rank of 1 and a polynomial degree of 0.

	$\mathrm{TT}_{\mathrm{impl}}$	$\mathrm{TT}_{\mathrm{expl}}$	$\mathrm{NN}_{\mathrm{impl}}$	$\mathrm{NN}^*_{\mathrm{impl}}$
$\widehat{V}_0(x_0)$	0.052800	0.05256	0.04678	0.05176
relative error	$4.75\mathrm{e}^{-5}$	$4.65\mathrm{e}^{-3}$	$1.14\mathrm{e}^{-1}$	$1.97\mathrm{e}^{-2}$
PDE loss	$2.40\mathrm{e}^{-4}$	$2.57\mathrm{e}^{-4}$	$9.08\mathrm{e}^{-1}$	$6.92\mathrm{e}^{-1}$
comp. time	24	10	23010	95278

Table A.4: Approximations for Allen-Cahn PDE, where $\mathrm{NN}^*_{\mathrm{impl}}$ uses $K = 8000$ and the others $K = 1000$ samples.