

Approximating optimal feedback controllers of finite horizon control problems using hierarchical tensor formats

Mathias Oster¹

Leon Sallandt¹

oster@math.tu-berlin.de sallandt@math.tu-berlin.de

Reinhold Schneider¹

schneider@math.tu-berlin.de

¹Technische Universität Berlin

April 14, 2021

Abstract

Controlling systems of ordinary differential equations (ODEs) is ubiquitous in science and engineering. For finding an optimal feedback controller, the value function and associated fundamental equations such as the Bellman equation and the Hamilton-Jacobi-Bellman (HJB) equation are essential. The numerical treatment of these equations poses formidable challenges due to their non-linearity and their (possibly) high-dimensionality.

In this paper we consider a finite horizon control system with associated Bellman equation. After a time-discretization, we obtain a sequence of short time horizon problems which we call local optimal control problems. For solving the local optimal control problems we apply two different methods, one being the well-known policy iteration, where a fixed-point iteration is required for every time step. The other algorithm borrows ideas from Model Predictive Control (MPC), by solving the local optimal control problem via open-loop control methods on a short time horizon, allowing us to replace the fixed-point iteration by an adjoint method.

For high-dimensional systems we apply low rank hierarchical tensor product approximation/tree-based tensor formats, in particular tensor trains (TT tensors) and multi-polynomials, together with high-dimensional quadrature, e.g. Monte-Carlo.

We prove a linear error propagation with respect to the time discretization and give numerical evidence by controlling a diffusion equation with unstable reaction term and an Allen-Kahn equation.

1 Introduction

We consider a finite horizon optimal control problem on a d -dimensional state space. In our applications the constraining dynamical systems are spatial discretizations of partial differential equations (PDE) and thus d can become large. Opposed to open-loop controls, feedback laws provide controls for all initial values simultaneously and in many cases produce stabilizing controls even under perturbations. For finding an optimal feedback law, a mapping from the state to the optimal control is needed. In order to allow online usage of the feedback law fast evaluation of this map is critical.

One popular approach for finding an optimal feedback law is to calculate the value function $v^* : \mathbb{R}^d \times [0, T] \rightarrow \mathbb{R}$, which is a mapping from the state space to the real numbers. This function obeys a fundamental equation, the Bellman equation [Bel57; BC97]. For given time-points the Bellman equation is solved recursively by a sequence of optimal control problems with short time horizon $\tau \ll T$, where τ is the step size in a time discretization of the value function.

We treat these time-local optimal control problems using two different methods, one being an open-loop/adjoint approach and the other being the policy iteration algorithm [How60].

The open-loop ansatz is motivated by the observation that the value function can be computed point-wise using traditional methods from optimal control such as open-loop control/adjoint methods [Pon+62].

In order to obtain such point-evaluations of the value function, the optimal control problem has to be solved for an initial value. The dimension of this optimal control problem scales with the time-horizon and for every gradient step the ODE has to be solved in a forward and a backward way. In that sense, computing an optimal control and thus the value function can become expensive for long time horizons. However, due to short time horizons τ of the local optimal control problems the open-loop approach is efficient for our approach.

Using these point measurements we perform an interpolation/regression to obtain the value function for every initial state. This yields a backwards iteration w.r.t. time. Note that in our numerical tests computing the local open-loop control problems is the numerical bottleneck and this step can be parallelized perfectly.

In the second approach we use the policy iteration algorithm combined with a regression to solve the local optimal control problems. This yields a similar backwards iteration with a nested fixed-point equation. In this approach several regression problems have to be solved for every time-point. However, generating samples is less expensive, shifting the numerical bottleneck from generating the samples to solving the equations.

As the dimension of the state space increases, traditional methods to represent the function, like Galerkin approximation by splines, finite elements or multi polynomials suffer from the so-called curse of dimensionality, which means that their complexity increases exponentially with the dimension of the state space. To alleviate this problem we use a non-linear model class, namely low

rank hierarchical tensor product approximation/tree-based tensor formats, in particular tensor trains (TT tensors) and multi-polynomials, together with high dimensional Monte-Carlo quadrature. By this approach we compute a low-fidelity/complexity representation of the required optimal feedback control such that it is easily applied in online computations. We prove a linear error propagation with respect to the time discretization in Theorem 6.

Finally, we provide numerical evidence for both approaches by testing a feedback law computed for a diffusion equation with unstable reaction term as well as a Allen-Kahn equation. Moreover, we propose possible extensions of the methods where more information about the system is used.

Previous Work

The Bellman equation, also known as dynamic programming, was introduced in the 1950s by R. Bellman [Bel66]. Note that in addition to the Bellman equation, there exists an infinitesimal version, the Hamilton-Jacobi-Bellman (HJB) equation, see e.g. [BC97; FF13]. Both equations can be approximated by the policy iteration [How60], which is a widely used tool in optimal feedback control, see e.g. [KK18; AFK15]. Other popular methods for solving the HJB equation are semi-Lagrangian methods [TAK17; DJ14; Fal87], Domain splitting algorithms [FLS94], variational iterative methods [KDK13], data based methods with Neural Networks [Luo+14], actor-critic methods [ZHL21], tree-based methods [AS20] and tropical algorithms [AGL09; AF18].

Simultaneously to the work of Bellman, Pontryagin developed a set of necessary conditions for optimal problems [Pon+62], the so-called Pontryagin maximum principle (PMP). This approach naturally leads to adjoint methods, from where feedback control is not immediately applicable.

Despite its open-loop nature, the PMP has already been used to find feedback controls, see e.g. [BTB00; KW17; NGK19; AKK20]. However, in contrast to our open-loop approach, these approaches do not use the Bellman equation and thus the PMP approach can suffer from long time-horizons.

Another approach, closely related to our open-loop ansatz, where the concept of adjoint methods and feedback control are combined is Model Predictive Control (MPC). Model predictive control originates for simple applications in the 1970s for stabilizing linear systems, see i.e. [Kle70], and later reformulated for non-linear systems, see e.g. [GP17]. For surveys on MPC w.r.t. theoretical results and industrial applications we refer to [GPM89; QB03]. The MPC approach replaces the optimal control problem with short and overlapping time horizon and obtains a feedback control by solving this short time horizon problem, to obtain a sub-optimal online controller. However, due to its short time horizon, this optimal control problem can be solved in real-time. Our open-loop method can be characterized as a MPC approach with optimal final condition, leading to an optimal controller. This combination of MPC and the Bellman equation has already been considered in the context of reinforcement learning [Atk+94; Zho+13]. In these applications, low-dimensional systems are considered.

The second approach we consider in this paper is an application of the policy iteration algorithm.

In order to control higher-dimensional systems, memory efficient methods have to be applied to circumvent the curse of dimensionality. Hierarchical tensor products such as tensor trains have also been recently used in this context. Rooted in quantum physics under the name *matrix product states*, tensor trains have been introduced to the mathematical community in [Ose11] to tackle the curse of dimensionality. Note that tensor trains are a special case of a more general framework - hierarchical tensor networks. These networks have been developed in [HK09] where a well-founded mathematical framework is introduced. For surveys and more details, see [Hac14; HS14; Sza+15; BSU16]. Tensor trains have already been applied to represent the value function of infinite horizon control problems [DKK19; OSS19] and stochastic control problems [HDB14; Fac+20; GKM18].

Instead of this ansatz space it is possible to use other methods from machine learning such as neural networks, see e.g., [DLM20; NR20; IRZ20], kernel methods or sparse polynomials, as it is done in [AKK20].

Finally, our regression/Least-Squares-based approach is closely related with empirical risk minimization [Vap92; SC08] and classical machine learning tasks with one fundamental twist. In classical tasks from statistical and machine learning the data is noisy and samples are prescribed and biased. In our approach, we are able to compute the data with close to arbitrary precision due to the adjoint method approach and advanced ODE solvers. Moreover, we are able to generate or data points arbitrarily. Due to the precision of the adjoint approach, computing the data can be understood as a so-called high-fidelity computation. In contrast to that, our approximation of the value function can be seen as a low-fidelity representation of the value function, enabling real-time evaluations of the gradient and thus of the (approximative) optimal control.

While the generalization to controlling stochastic systems is straight-forward in the case of the policy iteration, the generalization of the open-loop approach to stochastic control is more complex. Here, possible approaches are the solution of forward-backward stochastic differential equations (FBSDE) [BT04; GLW+05], for an optimal control context see e.g. [Pha+05]. We want to highlight recent groundbreaking successes in the treatment of FBSDEs using deep learning, see e.g. [EHJ17]. It was also shown that tensor trains are efficiently applicable in this context [RSN21].

Structure

The rest of the paper is organized as follows. In the preliminaries Section 2 we introduce the optimal control problem, and fundamental concepts such as the value function and the Bellman equation.

Section 3 is devoted to reinterpreting the Bellman equation as an optimal control problem and the corresponding open-loop ansatz. In the following sections we treat the Bellman equation via the policy iteration algorithm. Then, we clarify the numerical feasibility by introducing the regression method in com-

bination with our ansatz set, the tensor trains. In Section 7 we give several ideas on how to improve the approach and in the final section we give numerical evidence.

2 The Optimal Control Problem

In this section we formulate the optimal control problem in open-loop and closed-loop form. Furthermore, we define basic notions of control theory and formulate governing equations.

We consider a deterministic, finite time horizon optimal control problem of the following form. For $x \in \Omega \subset \mathbb{R}^d$ minimize w.r.t. $u \in L^2(0, T; \mathbb{R}^m)$ the cost functional $\mathcal{J} : [0, T] \times \Omega \times L^2(0, T; \mathbb{R}^m) \rightarrow \mathbb{R}$, defined as,

$$\mathcal{J}(t_0, x, u(\cdot)) := \int_{t_0}^T c(t, y(t)) + u(t)' R(t) u(t) dt + c_T(y(T)), \quad (1)$$

where

$$\dot{y}(t) = f(t, y(t)) + g(t, y(t))u(t) \quad (2)$$

$$y(0) = x. \quad (3)$$

We assume $c : [0, T] \times \Omega \rightarrow \mathbb{R}_{\geq 0}$ and $c_T : \Omega \rightarrow \mathbb{R}_{\geq 0}$ are non-negative, coercive and smooth. Further, let $R : [0, T] \rightarrow \mathbb{R}^{m, m}$ be a family of positive definite matrices continuous in time. Note that $u(t)'$ denotes the transpose of $u(t)$. For initial data $x \in \Omega$ and fixed control $u(\cdot) \in \mathcal{A}$, we denote by $y^x(t, u) \in \Omega$ the evaluation of the trajectory at time t . If the context is clear we just write $y(t)$. We further assume that $f : [0, T] \times \Omega \rightarrow \Omega$ and $g : [0, T] \times \Omega \rightarrow \mathbb{R}^{n, m}$ are (non linear) smooth functions. We define the value function as

$$v^* : [0, T] \times \Omega \rightarrow \mathbb{R}, \quad (t, x) \mapsto \inf_{u \in L^2(0, T; \mathbb{R}^m)} \mathcal{J}(t, x, u) \quad (4)$$

and the corresponding Bellman equation takes the following form.

Theorem 1. [BC97; BD97] *Set $\ell(t, x, u) = c(t, x) + u' R(t) u$. Then for all $x \in \Omega$ and $0 \leq t_0 < t_1 \leq T$ we have*

$$v^*(t_0, x) = \inf_{u \in L^2(t_0, t_1; \mathbb{R}^m)} \left[\int_{t_0}^{t_1} \ell(y(t), u(t)) dt + v^*(t_1, y(t_1)), \right] \quad (5)$$

where $y(\cdot)$ is the trajectory corresponding to (2) with control u and initial value $y(t_0) = x$.

We further have the following HJB equation

Theorem 2. [BC97; BD97] *Assume there are $\sigma, \delta \geq 1$ with $\sigma < \delta$, $\ell_0 > 0$ and for every compact $K \subset \mathbb{R}^d$ some $f_K > 0$ such that*

$$\begin{aligned} \|f(x, u)\| &\leq f_K(1 + \|x\|^\sigma) \quad \forall (x, u) \in K \times \mathbb{R}^m, \\ |\ell(x, u)| &\geq \ell_0 \|a\|^\delta \quad \forall (x, u) \in \mathbb{R}^d \times \mathbb{R}^m. \end{aligned}$$

Then the value function $v^*(t, x)$ is the unique viscosity solution of

$$\frac{\partial}{\partial t} v^*(t, x) + \sup_{u \in \mathbb{R}^m} \left[-\nabla v^*(t, x) \cdot (f(t, x) + g(t, x)u) - \ell(t, x, u) \right] = 0 \quad (6)$$

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

In the following, we refer to the collection (1) and (2) as the open-loop control problem. In contrast to that, in the closed-loop formulation, the parameter $u(t)$ is replaced by a mapping $\alpha : \Omega \rightarrow \mathbb{R}^m$, such that the closed-loop system takes the form.

$$\dot{y} = f(t, y) + g(t, y)\alpha(t, y), \quad y(t_0) = x \quad (7)$$

for any policy $\alpha : [0, T] \times \Omega \rightarrow \mathbb{R}^m$ continuous on $[0, T]$ and Lipschitz in Ω and denote the policy evaluation function

$$\mathcal{J}^\alpha(t_0, x) := \int_{t_0}^T c(t, y^\alpha(t)) + \alpha(x(t))' R(t) \alpha(x(t)) dt + c_T(y^\alpha(T)). \quad (8)$$

Here, $y^\alpha(t)$ is the trajectory of the closed-loop system (7) evaluated at time t . We refer to the collection (8), (7) as the closed-loop problem. If the value function is known and differentiable, we can use it to explicitly compute an optimal feedback law.

Theorem 3. [BC97] *An optimal feedback control is given by*

$$\alpha^*(t, x) = -\frac{1}{2} R^{-1} g(t, x)' \nabla v^*(t, x), \quad (9)$$

if ∇v^* exists.

For $\tau \geq 0$ we define the flow $\Phi_\tau^\alpha : [0, T] \times \Omega \rightarrow [\tau, T + \tau] \times \Omega$ such that $\Phi_\tau^\alpha(t_0, x) = (t_0 + \tau, y(t_0 + \tau))$, 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 (7).

3 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 open-loop control problems. To this end, we consider a discrete set of time points $0 = t_1 < \dots < t_L = T$. 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

$$v^*(t_l, x) = \inf_{u \in L^2(t_l, t_{l+1}; \mathbb{R}^m)} \left[\int_{t_l}^{t_{l+1}} \ell(y(t), u(t)) dt + v^*(t_{l+1}, y(t_{l+1})) \right], \quad (10)$$

and notice that $v^*(t_{l+1}, y(\cdot))$ can be interpreted as a final condition and $\ell(y(\cdot), u(\cdot))$ as the running cost with governing ODE system (2).

The local open-loop control problem can be solved by adjoint methods. Recall that the Pontryagin Maximum Principle (PMP) gives as necessary conditions for a minimal control

$$\begin{aligned} \dot{y}(t) &= f(t, y) + g(t, y)\lambda(t), & y(0) &= x \\ \dot{\lambda}(t) &= -\partial_y H(y, u, \lambda) & \lambda(T) &= \partial_y c_T(y(T)) \\ H(y, u, \lambda) &= \min_{\tilde{u}} H(y, \tilde{u}, \lambda) \end{aligned}$$

where $H(y, u, \lambda) = \ell(y, u) + \lambda^T(f(t, y) + g(t, y)u)$, [Pon+62; Don+95]. After introducing a time discretization, this system is solved by standard gradient decent methods, see e.g. [HK10]. 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 in Section 5.

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.

Algorithm 1: Backwards solution to the Bellman equation - continuous case

input : Time points $0 = t_0 < \dots < t_L < T$.
output: The value function v^* evaluated at the time points

- 1 Set $v^*(t_L, \cdot) = c_T(\cdot)$
- 2 **for** $l = L - 1$ **to** 0 **do**
- 3 | Calculate $v^*(t_l, \cdot)$ by solving the local open-loop control problem
 | (10) with final condition $v^*(t_{l+1}, \cdot)$.
- 4 **end**

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 (2), 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

$$v^*(t, x) \approx \frac{t_{l+1} - t}{\tau} v^*(t_l, x) + \frac{t - t_l}{\tau} v^*(t_{l+1}, x) \text{ for } t \in [t_l, t_{l+1}), x \in \Omega. \quad (11)$$

Note that this interpolation does not affect Algorithm 1 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 3. We also make use of this interpolation for the policy iteration approach in Section 4.

Assuming an equidistant discretization of the ODE, the dimension of the local open-loop control problems increases only linearly in time and does not increase with the spatial dimension. However, the dimension of the regression problem is problematic and is covered in Section 6. Note that solving the local optimal control problems for every initial value is computationally not feasible. We instead draw samples $x_i \in \Omega$ and use a regression type approach as described in Section 5.

4 Policy Iteration approach

Another popular approach is the Policy iteration algorithm. This Newton-type method is an iterative scheme that alternates between solving a linearized Bellman equation and improving a policy.

As in the previous chapter, we use the same time subdivision $0 = t_0 < \dots < t_L = T$. 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

$$v^\alpha(t_l, \cdot) = \int_{t_l}^{t_{l+1}} \ell_{t-t_l}^\alpha(t_l, \cdot) dt + v^*(t_{l+1}, \Phi_\tau^\alpha(t_{l+1}, \cdot)), \quad v^\alpha(t_{l+1}, \cdot) = v^*(t_{l+1}, \cdot). \quad (12)$$

Note that plugging in the optimal policy recovers the Bellman equation (5). 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 3 cannot be evaluated at times other than t_l . Thus, we again make use of interpolation between the time-points

$$v^\alpha(t, \cdot) = \frac{t_{l+1} - t}{\tau} v^\alpha(t_l, \cdot) + \frac{t - t_l}{\tau} v^*(t_{l+1}, \cdot) \text{ for } t \in [t_l, t_{l+1}). \quad (13)$$

The policy iteration algorithm is then given by Algorithm 2.

Algorithm 2: Policy Iteration for approximating the local optimal control problem.

input : A Policy α_0 , $0 \leq t_l < t_{l+1} \leq T$ 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 $\hat{\alpha}(t_{l+1}, \cdot)$.

- 1 Set $k = 0$.
- 2 **while** *not converged* **do**
- 3 Solve the linear equation

$$v_{k+1}(x) = \int_{t_l}^{t_{l+1}} \ell_{t-t_l}^{\alpha_k}(t_l, x) dt + \hat{v}(t_{l+1}, \Phi_\tau^{\alpha_k}(t_l, x)) \quad (14)$$

and use linear interpolation between v_{k+1} and $\hat{v}(t_{l+1}, \cdot)$ as in (13) to obtain $v_{k+1}(t, \cdot)$ for $t \in [t_l, t_{l+1}]$. Then update the policy according to

$$\alpha_{k+1}(t, x) = -\frac{1}{2}B(t)^{-1}g(t, x)^T \nabla v_{k+1}(t, x), \quad t \in [t_l, t_{l+1}].$$

- 4 $k = k + 1$.
 - 5 **end**
 - 6 Set $\hat{v}(t, \cdot) = v_k(t, \cdot)$ and $\hat{\alpha}(t, \cdot) = \alpha_k$ for $t \in [t_l, t_{l+1}]$.
-

In [PB79; SG77] the convergence of the exact Policy iteration is analyzed and in [LB98] extended to Galerkin methods. The convergence and error estimates are still open problems in the Least-Squares setting. 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 a regularization term, as described in (28).

Note that the right-hand-side of (14) can be computed point wise without knowledge of the underlying dynamical system. Only a black-box solver for $\Phi_{t_l}^\alpha$ and a function for evaluation of $\ell_{t-t_l}^\alpha$ 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.

5 Formulation of the Regression Problem

For solving (10, 14), computing the value function for every point x is not feasible. Moreover, we cannot expect to have access to the exact value function $v^*(t_{l+1}, \cdot)$ as final condition, but only an approximation. We compute the approximation of the value function by a regression ansatz, which is very similar for both approaches. We first describe the procedure for the local open-loop ansatz and thereafter comment on the policy iteration approach.

We replace the value function as a final condition with an approximation

$\hat{v}_{l+1}(\cdot)$ that we computed in the step before, i.e. we replace the local optimal control problem (10) by

$$\tilde{v}_l(x) = \inf_{u \in L^2(t_l, t_{l+1}; \mathbb{R}^m)} \int_{t_l}^{t_{l+1}} \ell(y(t), u(t)) dt + \hat{v}_{l+1}(y(t_{l+1})). \quad (15)$$

Note that we write \tilde{v}_l on the l.h.s. and \hat{v}_{l+1} on the r.h.s. of the equation.

We do not use the same notation because we are not able to find the exact minimizer \tilde{v}_l for every initial value x . Instead we find an approximation, which we again denote by \hat{v}_l . In order to find this approximation we observe that $\tilde{v}_l \in L^2(\Omega) =: V$. Choosing an ansatz space $\mathcal{M} \subset V$ and we seek an approximation in the sense of a Least-Squares method

$$\inf_{v \in \mathcal{M}} \|v - \tilde{v}_l\|_V^2. \quad (16)$$

As we cannot compute the norm, we replace this equation by a discrete version using empirical risk minimization [CS01]. To this end we consider a set of samples $\{x_1, \dots, x_J\} \subset \Omega$ and compute $\tilde{v}_l(x_i)$ for all $1 \leq j \leq J$, which is a regression problem. In particular we set

$$\hat{v}_l = \arg \min_{v \in \mathcal{M}} \frac{1}{J} \sum_{j=1}^J |v(x_j) - \tilde{v}_l(x_j)|^2. \quad (17)$$

In this paper we consider tensor networks as ansatz space. Error estimates w.r.t. the number of samples are given in [Eig+20] for our ansatz space. At this point we notice that the problem of a potentially irregular value function can be solved here. By replacing the final condition $v^*(t_l, \cdot) \in L^2(\Omega)$ by the approximation $\hat{v}_l \in \mathcal{M}$ we can enforce a smooth final condition by choosing an ansatz space \mathcal{M} consisting of smooth functions. This results in the following discrete version of Algorithm 1.

Algorithm 3: Backwards solution to the Bellman equation - discrete case

input : Time points $0 = t_0 < \dots < t_L < T$ and samples $x_1, \dots, x_J \in \Omega$
output: An approximation \hat{v}_l of the value function $v^*(t_l, \cdot)$ evaluated at the time points

- 1 Set $\hat{v}_L = c_T$
- 2 **for** $l = L - 1$ **to** 0 **do**
- 3 Calculate $\tilde{v}_l(x_j)$ by solving the local open-loop control problem (15) with initial values x_j and final condition \hat{v}_{l+1} .
- 4 Compute \hat{v}_l by solving the regression problem

$$\hat{v}_l = \arg \min_{v \in \mathcal{M}} \frac{1}{J} \sum_{j=1}^J |v(x_j) - \tilde{v}_l(x_j)|^2. \quad (18)$$

5 **end**

Remark 4. Up until now we have not specified the ansatz set \mathcal{M} . Note that here, any ansatz set where (18) is computable can be used. In low dimension, traditional ansatz sets such as finite elements, splines or polynomial ansatz spaces are feasible. In higher dimension, other ansatz sets such as neural networks or tensor networks, as considered here, are possible. Note that computing this algorithm consists of L equations that have to be solved where J samples have to be generated. This formulation is especially viable if solving the regression problems is particularly expensive and if the local open-loop control problems can be solved efficiently.

For the policy iteration approach the discrete version of the algorithm is very similar. Here, we simply replace (14) by a discrete version

$$v_{k+1} = \arg \min_{v \in \mathbb{F}} \frac{1}{J} \sum_{j=1}^J \int_{t_l}^{t_{l+1}} |v(x_j) - \ell_{t-t_l}^{\alpha_k}(t_l, x_j) dt - \hat{v}(t_{l+1}, \Phi_{\tau}^{\alpha_k}(t_l, x_j))|^2. \quad (19)$$

We finalize this section by giving an error propagation under the assumption that we can bound the difference between \tilde{v} and \hat{v} in the $L^\infty(\Omega)$ norm.

We first prove that for optimal control problems of similar form, the value functions are similar.

Lemma 5. Let v_1, v_2 be value functions to optimal control problems of the form

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

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.

The proof is given in the appendix. 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 (15) can be seen as an intermediate between the exact value function $v^*(t_l, \cdot)$ and the computable approximation \hat{v}_l .

Theorem 6. By \tilde{v} denote the solution to the local optimal control problem (15) 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 $1 \leq l \leq L$. 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

$$\|\hat{v}_{L-l}(\cdot) - v^*(T - l\tau, \cdot)\|_{L^\infty(\Omega)} < l\delta. \quad (20)$$

Finally, we have

$$\begin{aligned} & \|\hat{v}_{L-(l+1)}(\cdot) - v^*(T - (l+1)\tau, \cdot)\|_{L^\infty(\Omega)} \quad (21) \\ \leq & \underbrace{\|\hat{v}_{L-(l+1)}(\cdot) - \tilde{v}_{L-(l+1)}(\cdot)\|_{L^\infty(\Omega)}}_{\leq \delta \text{ by assumption}} + \underbrace{\|\tilde{v}_{L-(l+1)}(\cdot) - v^*(T - (l+1)\tau, \cdot)\|_{L^\infty(\Omega)}}_{\leq l\delta \text{ by (20) plugged into Lemma 5}} \end{aligned} \quad (22)$$

$$\leq (l+1)\delta. \quad (23)$$

This finishes the proof. \square

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. As the model set \mathcal{M} is continuously embedded into its span, i.e. $\mathcal{M} \hookrightarrow \text{span}(\mathcal{M})$, 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.

6 Representation of the Value Function

The question of how to represent the approximation of the value function $\hat{v}_l \in \mathcal{M}$ and how to find it within the ansatz space is remaining. For the sake of readability we drop the subscript l in this section.

Traditional methods such as finite elements, splines or multi-variate polynomials leads to a computational complexity that scales exponentially in the state space dimension d . However, interpreting the coefficients of such ansatz functions as entries in a high-dimensional tensor allows us to use tensor compression methods to reduce the number of parameters. To this end, we define a set of functions $\{\phi_1, \dots, \phi_m\}$ with $\phi_i : \mathbb{R} \rightarrow \mathbb{R}$, e.g. one-dimensional polynomials or finite elements. The approximation $\hat{v} : \mathbb{R}^d \rightarrow \mathbb{R}$ takes the form

$$\hat{v}(x_1, \dots, x_d) = \sum_{i_1=1}^m \cdots \sum_{i_d=1}^m c_{i_1, \dots, i_d} \phi_{i_1}(x_1) \cdots \phi_{i_d}(x_d). \quad (24)$$

Note that this is a standard formulation for finite elements or multivariate polynomial bases. For the sake of simplicity we choose the set of ansatz functions to be the same in every dimension. The coefficient tensor $c \in \mathbb{R}^{m \times m \times \cdots \times m} \equiv \mathbb{R}^{m^d}$ suffers from the curse of dimensionality since the number of entries increases

exponentially in the dimension d . In what follows, we review the tensor train format to compress the tensor c .

For the sake of readability we will henceforth write $c_{i_1, \dots, i_d} = c[i_1, \dots, i_d]$ and represent the contraction of the last index of a tensor $w_1 \in \mathbb{R}^{r_1 \times m \times r_2}$ with the first index of another tensor $w_2 \in \mathbb{R}^{r_2 \times m \times r_3}$ by

$$w = w_1 \circ w_2 \in \mathbb{R}^{r_1 \times m \times m \times r_3}, \quad (25a)$$

$$w[i_1, i_2, i_3, i_4] = \sum_{j=1}^{r_2} w_1[i_1, i_2, j] w_2[j, i_3, i_4]. \quad (25b)$$

In the literature on tensor methods, graphical representations of general tensor networks are widely used. In these pictorial descriptions, the contractions \circ of the component tensors are indicated as edges between vertices of a graph. As an illustration, we provide the graphical representation of an order-4 tensor and a tensor train representation (see Definition 7 below) in Figure 1.

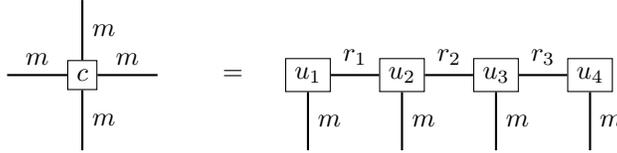


Figure 1: An order 4 tensor and a tensor train representation.

Tensor train representations of c can now be defined as follows [Ose11].

Definition 7 (Tensor Train). *Let $c \in \mathbb{R}^{m \times \dots \times m}$. A factorization*

$$c = u_1 \circ u_2 \circ \dots \circ u_d, \quad (26)$$

where $u_1 \in \mathbb{R}^{m \times r_1}$, $u_i \in \mathbb{R}^{r_{i-1} \times m \times r_i}$, $2 \leq i \leq d-1$, $u_d \in \mathbb{R}^{r_{d-1} \times m}$, is called tensor train representation of c . We say that u_i are component tensors. The tuple of the dimensions (r_1, \dots, r_{d-1}) is called the representation rank and is associated with the specific representation (26). In contrast to that, the tensor train rank (TT-rank) of c is defined as the minimal rank tuple $\mathbf{r} = (r_1, \dots, r_{d-1})$, such that there exists a TT representation of c with representation rank equal to \mathbf{r} . Here, minimality of the rank is defined in terms of the partial order relation on \mathbb{N}^d given by

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

for $\mathbf{r} = (r_1, \dots, r_d)$, $\mathbf{s} = (s_1, \dots, s_d) \in \mathbb{N}^d$.

It can be shown that every tensor has a TT-representation with minimal rank, implying that the TT-rank is well defined [HRS12b]. An efficient algorithm for computing a minimal TT-representation is given by the Tensor-Train-Singular-Value-Decomposition (TT-SVD) [OT09]. Additionally, the set of tensor trains with fixed TT-rank forms a smooth manifold, and if we include lower ranks, an algebraic variety is formed [Lan12].

The TT-representation of (24) is then given as

$$\hat{v}(x) = \sum_{i_1}^m \cdots \sum_{i_d}^m \sum_{j_1}^{r_1} \cdots \sum_{j_{d-1}}^{r_{d-1}} u_1[i_1, j_1] u_2[j_1, i_2, j_2] \cdots \cdots u_d[j_{d-1}, i_d] \phi_{i_1}(x_1) \cdots \phi_{i_d}(x_d). \quad (27)$$

Introducing the compact notation

$$\phi : \mathbb{R} \rightarrow \mathbb{R}^m, \quad \phi(x) = [\phi_1(x), \dots, \phi_m(x)],$$

the corresponding graphical TT-representation (with $d = 4$ for definiteness) is then given as in Figure 2.

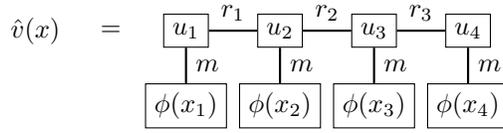


Figure 2: Graphical representation of $v_n : \mathbb{R}^4 \rightarrow \mathbb{R}$.

Within the TT format regression problems can be solved by using the *alternating least squares* (ALS) algorithm [HRS12a], where the regression problem, suffering from the curse of dimensionality is reduced to a sequence of linear sub-problems, whose dimensionality is given by the size of a single component tensor. The routine is based on the observation, that fixing all component tensors but one reduces the multilinear ansatz to a linear problem, that can be solved by standard linear regression models, c.f. Figure 3. Note that we replace the integral in Figure 3 by Monte-Carlo quadrature, c.f. Section 5. Iteratively updating the component tensors produces a monotone sequence, that converges at least locally. In our version of the ALS algorithm we update the component

$$\min_{\tilde{u} \in \mathbb{R}^{r_1 \times m \times r_2}} \int_{\Omega} \left| \begin{array}{c} \boxed{u_1} \\ | \\ \boxed{\phi(x_1)} \end{array} - \begin{array}{c} \circled{\tilde{u}} \\ | \\ \boxed{\phi(x_2)} \end{array} - \begin{array}{c} \boxed{u_3} \\ | \\ \boxed{\phi(x_3)} \end{array} - \begin{array}{c} \boxed{u_4} \\ | \\ \boxed{\phi(x_4)} \end{array} \right|^2 dx_1 \dots dx_4$$

Figure 3: Graphical representation of ALS algorithm. In this linear sub-problem the second component tensor is being optimized, while component tensors u_1, u_3, u_4 are fixed (from previous iterations).

tensors from left to right. We say that after every component tensor is updated once, one sweep is complete. In order to circumvent overfitting, we further add a penalization term to the loss functional (18), such that the regression problem becomes

$$\hat{v} = \arg \min_{v \in \mathcal{M}} \frac{1}{J} \sum_{j=1}^J |v(x_j) - \tilde{v}(x_j)|^2 + \delta \|v\|_{H_{\text{mix}}^2(\Omega)}, \quad (28)$$

where $H_{\text{mix}}^2(\Omega)$ is the tensor product of one-dimensional H^2 Sobolev spaces [SU09], assuming that Ω can be written as $\Omega = \bigotimes_{i=1}^d [a, b]$, where $a < b$. Choosing the one-dimensional ansatz functions to be orthonormal w.r.t. $H^2(a, b)$, this regularization term is realized via Parseval's identity, by penalizing the Frobenius norm of the component tensors. As $H_{\text{mix}}^2(\Omega)$ is continuously embedded into $W^{1,\infty}(\Omega)$ [SU09] we are penalizing the $L^\infty(\Omega)$ norm of the gradient of v , which prevents overfitting. Note that in order to reduce the impact of the regularizer we reduce δ within the iteration of the ALS. After every completion of a sweep, we set δ to be the 10^{-3} times the residual of the loss functional (28). In our numerical tests we have noticed that this regularizer is integral to the success of our method.

6.1 Complexity calculations for the evaluation and gradient of the function

In the following we cover the evaluation of v and the computation of the gradient of v within the TT-format. Note that for fast evaluation of the feedback law, efficient computation of the gradient is critical. In the following complexity considerations we assume that $r_i = r$ for all i .

Evaluating v in the TT format. First, we consider the mapping $x \mapsto v(x)$ under the assumption that v is in TT-format. This operation can be denoted using the \circ operation as

$$v(x_1, \dots, x_d) = u_1 \circ \dots \circ u_d \circ \phi(x_d) \circ \dots \circ \phi(x_1). \quad (29)$$

We notice that in order to evaluate v at a certain point $x \in \mathbb{R}^d$, we have to compute $\phi(x_i)$ for all i . This corresponds to $m \cdot d$ evaluations of one-dimensional functions, where we assume that these evaluations are $\mathcal{O}(1)$ each. The contraction is performed from left-to-right (or from right-to-left) and is of complexity $\mathcal{O}(dmr^2)$, because for every component tensor of dimension $r \times m \times r$ one vector of size r and one vector of size m has to be contracted, leaving a vector of size r .

Evaluating ∇v in the TT format. In order to evaluate ∇v at a certain point $x \in \mathbb{R}^d$, we have to compute $\phi(x_i)$ and $\phi'(x_i)$ for all i , where $\phi'(x_i) \in \mathbb{R}^m$ 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

$$\begin{aligned} \frac{\partial v}{\partial x_i}(x_1, \dots, x_d) = \\ u_1 \circ \dots \circ u_d \circ \phi(x_d) \circ \dots \circ \phi(x_{i+1}) \circ \phi'(x_i) \circ \phi(x_{i-1}) \circ \dots \circ \phi(x_1). \end{aligned} \quad (30)$$

Noticing the similarity of the above formulas with (29) we can again estimate the complexity of computing a partial derivative to be similar to the previous case. A naive implementation of the gradient then yields another factor d in

the complexity estimation, obtaining a total complexity of $\mathcal{O}(d^2mr^2)$. However, in the TT-format the naive implementation computes many redundant contractions, which means that we can save some complexity here. As shown in Appendix B it is possible to reduce the complexity to $\mathcal{O}(dmr^2)$ via a recurrent scheme.

7 Comparing both approaches and possible improvements

In this section we compare both 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. In [Fac+20] first ideas to the formal scaling w.r.t. the spatial dimension are described for the Policy Iteration combined with Least-Squares methods in the context of stochastic exit time problems. As indicated there within, the complexity of deterministic systems is in a similar but reduced fashion since the stochastic behaviour does not need to be resolved.

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 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 6 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 6 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 4

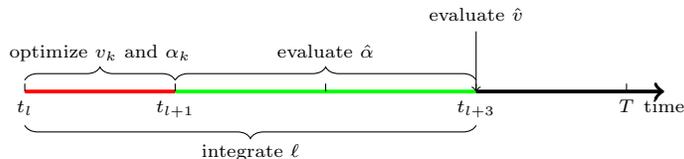


Figure 4: Visualization of increased integration horizons. The red part is optimized via the policy iteration, the green part is only evaluated and the evaluate or black part is estimated by evaluating \hat{v} at time t_{l+3} . Note that \hat{a} and \hat{v} are already computed and thus fixed.

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 3, we know that

$$u_{i,l}^*(t_l) = -\frac{1}{2}R^{-1}g(t, x_i)' \nabla v^*(t_l, x_i). \quad (31)$$

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

$$\hat{v}_l = \arg \min_{v \in \mathcal{M}} \frac{1}{L} \sum_{l=1}^L |v(x_l) - \tilde{v}(x_l)|^2 + \eta |u_{i,l}^*(t_l) + \frac{1}{2}R^{-1}g(t, x_i)' \nabla v(x_i)|^2 + \delta \|v\|_{H_{\text{mix}}^2(\Omega)}, \quad (32)$$

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] and we also want to highlight that in [AKK20] a similar adaption of the loss functional is done, where improved performance is reported.

Finally, another improvement is again motivated by MPC. Instead of computing the feedback law by using the optimality condition in Theorem 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 lower 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.

8 Numerical Results

We present results of numerical tests for different optimal control problems. For the implementation of the tensor networks we use the library `xerus` [HW17]. 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

$$\arg \min_{u \in L^2((0, \infty); \mathbb{R}^m)} \mathcal{J}(x, u) = \int_0^T \|y(t)\|^2 + 0.1\|u(t)\|^2 + c\|y(T)\|^2 dt, \quad (33)$$

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

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

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 regularizer (28). 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 to obtain the value function at the steps between our discretization points t_l . We state that this uncoupling 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 6.

Remark 8. *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 (8).*

8.1 Test 1: Diffusion with Unstable Reaction Term

We consider a diffusion equation with unstable reaction term and Neumann boundary condition, c.f. [KK18, Test 2]. Solve (33) 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. $\omega = [-0.4, 0.4] \subset [-1, 1]$. 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

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

Using the step-size $h = \frac{1}{d+1}$ 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

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

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 9. *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, $32370 = 6 \cdot 5395$. 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. For both controllers, significant improvement in cost is noticeable, with the greatest being approximately 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 further investigate the performance of the controller by choosing initial values of the type $[x, x, \dots, x]$, where $x \in [0, 2]$. In Figure 6 we see that for small x every controller is close to optimal. For x larger than 1 the LQR controller performs significantly worse than the other controllers. By increasing the initial values beyond 1.5, the LQR controller fails to stabilize the system, while the 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. 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 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. 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

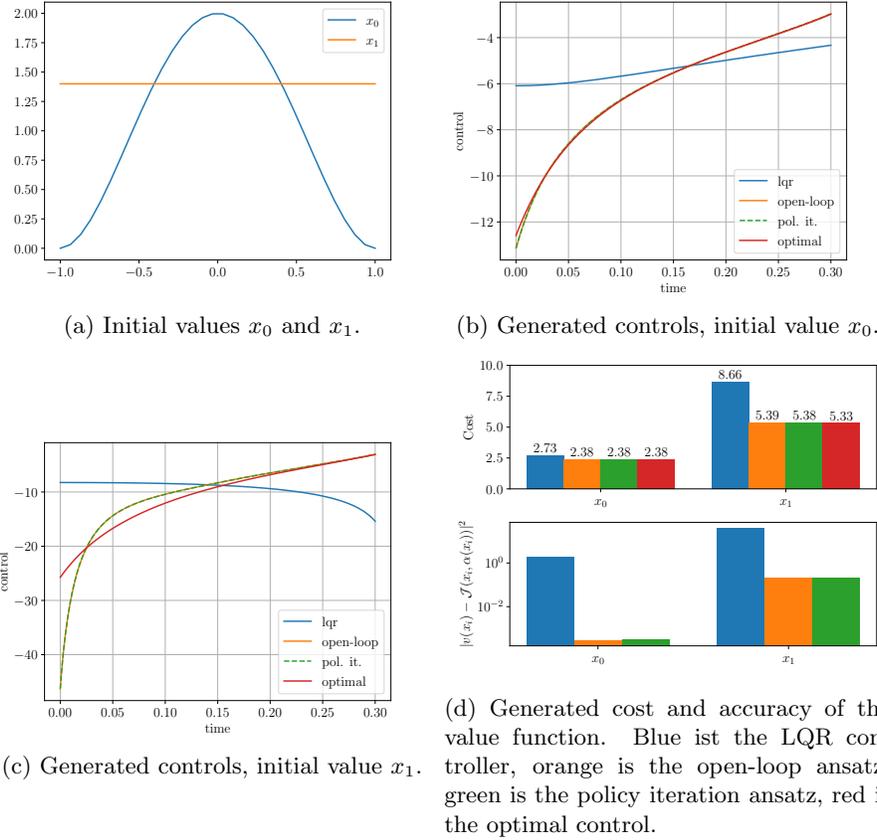


Figure 5: The generated controls and cost for different initial values.

values in Figure 7 and report the results in Table 1. We report 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 6 for values close to the boundary of our integration area.

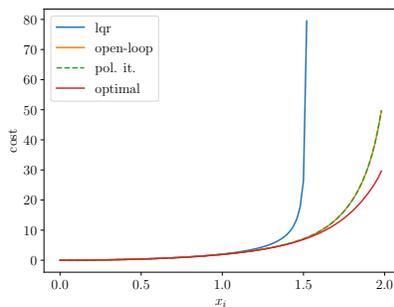


Figure 6: Cost of initial values of the type $[x, x, \dots, x]$.

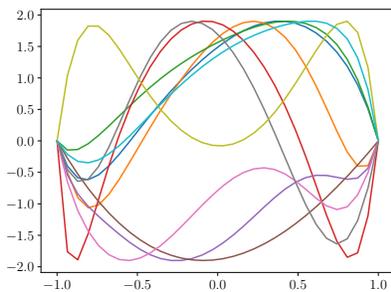


Figure 7: Examples of samples drawn from the polynomial distribution.

| 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 1: 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.

8.2 Test 2: Allen-Kahn equation

As underlying equation we use a one-dimensional Allen-Kahn equation similar to [KK18, Test 3]. Solve (33) 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 Section 8.2. The constants are the same except for $\sigma = 0.2$, $\omega = [-0.5, 0.2]$. Again, an ansatz of polynomials up to degree 4 is used. 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.

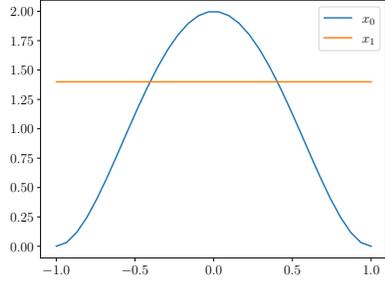
From Figure 8 we deduce that for certain initial values, significant improvement of cost is possible for both controllers with the greatest improvement being 25% of the cost. Again, the open-loop and the policy iteration approach yield similar performance. We again notice that for these initial values 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 $[x, x, \dots, x]$, where $x \in [0, 2]$. In Figure 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.

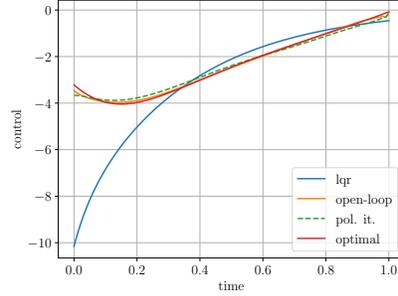
Next we again test random initial values using the same setup as in the previous test. In Table 2 we compare the performance of the controllers for 1000 random initial values drawn from the polynomial distribution. Again, the open-loop and the policy iteration approach yield close to optimal performance, while the LQR generates 22% more cost. Moreover, the the open-loop and the policy iteration approaches predict the generated cost accurately.

| 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 | |

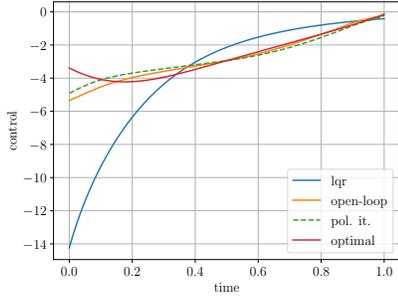
Table 2: Performance of the different controllers for 1000 samples drawn from the polynomial distribution. The averaged values are only taken from the subset of initial values that the LQR succeeded in stabilizing.



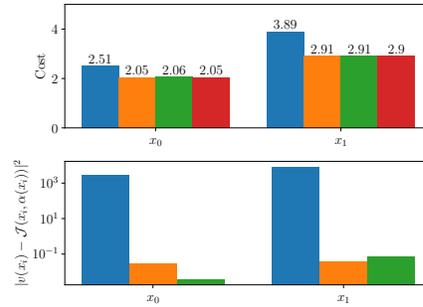
(a) Initial values x_0 and x_1 .



(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 8: The generated controls and cost for different initial values.

Conclusion

We have compared two methods for finding optimal controllers for finite horizon optimal control problems. In numerical tests we have observed similar performance for both methods. In contrast to the linear quadratic regulator we have obtained close to optimal costs for many initial states. Moreover, in none of the tests blow-ups occurred for our controllers. By encoding an approximation of the value function in a low-rank tensor model we have obtained a low-fidelity representation that allows fast evaluation of the feedback law. If even higher accuracy is needed, it is possible to take the control generated by our low-fidelity model as initial guess for further improvements via an open-loop optimization, where the control generated by the LQR controller fails.

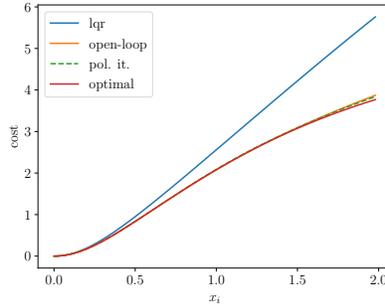


Figure 9: Cost of initial values of the type $[x, x, \dots, x]$.

Acknowledgements

Leon Sallandt and Mathias Oster acknowledge support from the Research Training Group "Differential Equation- and Data-driven Models in Life Sciences and Fluid Dynamics: An Interdisciplinary Research Training Group (DAEDALUS)" (GRK 2433) funded by the German Research Foundation (DFG).

References

- [AF18] Marianne Akian and Eric Fodjo. "Probabilistic Max-Plus Schemes for Solving Hamilton-Jacobi-Bellman Equations". In: Jan. 2018, pp. 183–209. ISBN: 978-3-030-01958-7. DOI: [10.1007/978-3-030-01959-4_9](https://doi.org/10.1007/978-3-030-01959-4_9).
- [AFK15] Alessandro Alla, Maurizio Falcone, and Dante Kalise. "An efficient policy iteration algorithm for dynamic programming equations". In: *SIAM Journal on Scientific Computing* 37.1 (2015), A181–A200.
- [AGL09] Marianne Akian, Stephane Gaubert, and Asma Lakhoua. "Convergence analysis of the Max-Plus Finite Element Method for Solving Deterministic Optimal Control Problems". In: Jan. 2009, pp. 927–934. DOI: [10.1109/CDC.2008.4739501](https://doi.org/10.1109/CDC.2008.4739501).
- [AKK20] Behzad Azmi, Dante Kalise, and Karl Kunisch. "Optimal Feedback Law Recovery by Gradient-Augmented Sparse Polynomial Regression". In: (2020).
- [AS20] Alessandro Alla and Luca Saluzzi. "A HJB-POD approach for the control of nonlinear PDEs on a tree structure". In: *Applied Numerical Mathematics* 155 (2020). Structural Dynamical Systems: Computational Aspects held in Monopoli (Italy) on June 12-15, 2018., pp. 192–207. ISSN: 0168-9274. DOI: <https://doi.org/10.1016/>

[j.apnum.2019.11.023](https://www.sciencedirect.com/science/article/pii/S0168927419303332). URL: <https://www.sciencedirect.com/science/article/pii/S0168927419303332>.

- [Atk+94] Christopher 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.
- [BC97] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*. Boston: Birkäuser, Jan. 1997. DOI: [10.1007/978-0-8176-4755-1](https://doi.org/10.1007/978-0-8176-4755-1).
- [BD97] Martino Bardi and Francesca Da Lio. “On the Bellman equation for some unbounded control problems”. In: *Nonlinear Differential Equations and Applications NoDEA* 4.4 (Oct. 1997), pp. 491–510. ISSN: 1420-9004. DOI: [10.1007/s000300050027](https://doi.org/10.1007/s000300050027). URL: <https://doi.org/10.1007/s000300050027>.
- [Bel57] Richard Bellman. *Dynamic programming*. Mineola, N.Y: Dover Publications, 1957. ISBN: 0-486-42809-5.
- [Bel66] Richard Bellman. “Dynamic programming”. In: *Science* 153.3731 (1966), pp. 34–37.
- [BSU16] Markus Bachmayr, Reinhold Schneider, and André Uschmajew. “Tensor Networks and Hierarchical Tensors for the Solution of High-Dimensional Partial Differential Equations”. In: *Found. Comput. Math.* 16.6 (Dec. 2016), pp. 1423–1472. ISSN: 1615-3375. DOI: [10.1007/s10208-016-9317-9](https://doi.org/10.1007/s10208-016-9317-9). URL: <https://doi.org/10.1007/s10208-016-9317-9>.
- [BT04] Bruno Bouchard and Nizar 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.
- [BTB00] Scott C Beeler, Hien T Tran, and Harvey Thomas Banks. “Feedback control methodologies for nonlinear systems”. In: *Journal of optimization theory and applications* 107.1 (2000), pp. 1–33.
- [CS01] Felipe Cucker and Steve Smale. “On the Mathematical Foundations of Learning”. In: *BULLETIN* 39 (Nov. 2001). DOI: [10.1090/S0273-0979-01-00923-5](https://doi.org/10.1090/S0273-0979-01-00923-5).
- [DJ14] Kristian Debrabant and Espen Jakobsen. “Semi-Lagrangian schemes for linear and fully non-linear Hamilton-Jacobi-Bellman equations”. In: *Hyperbolic Problems: Theory, Numerics, Applications*. Springer, Mar. 2014, pp. 483–490. ISBN: 978-1-60133-017-8.
- [DKK19] Sergey Dolgov, Dante Kalise, and Karl Kunisch. “Tensor decompositions for high-dimensional Hamilton-Jacobi-Bellman equations”. In: *arXiv preprint arXiv:1908.01533* (2019).

- [DLM20] Jerome Darbon, Gabriel P Langlois, and Tingwei Meng. “Overcoming the curse of dimensionality for some Hamilton–Jacobi partial differential equations via neural network architectures”. In: *Research in the Mathematical Sciences* 7.3 (2020), pp. 1–50.
- [Don+95] A. L. Dontchev et al. “Optimality, stability, and convergence in nonlinear control”. In: *Applied Mathematics and Optimization* 31.3 (May 1995), pp. 297–326. ISSN: 1432-0606. DOI: [10.1007/BF01215994](https://doi.org/10.1007/BF01215994). URL: <https://doi.org/10.1007/BF01215994>.
- [EHJ17] Weinan E, Jiequn Han, and Arnulf 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+20] Martin Eigel et al. “Adaptive stochastic Galerkin FEM for log-normal coefficients in hierarchical tensor representations”. In: *Numerische Mathematik* 145.3 (2020), pp. 655–692.
- [Fac+20] Konstantin Fackeldey et al. “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).
- [Fal87] M. Falcone. “A numerical approach to the infinite horizon problem of deterministic control theory”. In: *Applied Mathematics and Optimization* 15.1 (Jan. 1987), pp. 1–13. ISSN: 1432-0606. DOI: [10.1007/BF01442644](https://doi.org/10.1007/BF01442644). URL: <https://doi.org/10.1007/BF01442644>.
- [FF13] Maurizio Falcone and Roberto Ferretti. *Semi-Lagrangian approximation schemes for linear and Hamilton–Jacobi equations*. SIAM, 2013.
- [FLS94] Maurizio Falcone, Piero Lanucara, and Alessandra Seghini. “A splitting algorithm for Hamilton–Jacobi–Bellman equations”. In: *Applied Numerical Mathematics* 15.2 (1994), pp. 207–218. ISSN: 0168-9274. DOI: [https://doi.org/10.1016/0168-9274\(94\)00017-4](https://doi.org/10.1016/0168-9274(94)00017-4). URL: <http://www.sciencedirect.com/science/article/pii/0168927494000174>.
- [GKM18] Alex Gorodetsky, Sertac Karaman, and Youssef Marzouk. “High-dimensional stochastic optimal control using continuous tensor decompositions”. In: *The International Journal of Robotics Research* 37.2-3 (2018), pp. 340–377.
- [GLW+05] Emmanuel Gobet, Jean-Philippe Lemor, Xavier 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.

- [GP17] Lars Grüne and Jürgen Pannek. “Nonlinear model predictive control”. In: *Nonlinear Model Predictive Control*. Springer, 2017, pp. 45–69.
- [GPM89] Carlos E Garcia, David M Prett, and Manfred Morari. “Model predictive control: Theory and practice—A survey”. In: *Automatica* 25.3 (1989), pp. 335–348.
- [Hac14] Wolfgang Hackbusch. “Numerical tensor calculus”. In: *Acta numerica* 23 (2014), pp. 651–742. ISSN: 1474-0508. DOI: [10.1017/S0962492914000087](https://doi.org/10.1017/S0962492914000087).
- [HDB14] Matanya B Horowitz, Anil Damle, and Joel W Burdick. “Linear Hamilton Jacobi Bellman equations in high dimensions”. In: *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 5880–5887.
- [HK09] Wolfgang Hackbusch and Stefan Kühn. “A New Scheme for the Tensor Representation”. English. In: *Journal of Fourier Analysis and Applications* 15.5 (2009), pp. 706–722. ISSN: 1069-5869. DOI: [10.1007/s00041-009-9094-9](https://doi.org/10.1007/s00041-009-9094-9). URL: <http://dx.doi.org/10.1007/s00041-009-9094-9>.
- [HK10] Roland Herzog and Karl Kunisch. “Algorithms for PDE-constrained optimization”. In: *GAMM-Mitteilungen* 33.2 (2010), pp. 163–176.
- [How60] Ronald A Howard. “Dynamic programming and markov processes.” In: (1960).
- [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. DOI: [10.1137/100818893](https://doi.org/10.1137/100818893). eprint: <https://doi.org/10.1137/100818893>. URL: <https://doi.org/10.1137/100818893>.
- [HRS12b] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. “On manifolds of tensors of fixed TT-rank”. In: *Numerische Mathematik* 120.4 (2012), pp. 701–731.
- [HS14] Wolfgang Hackbusch and Reinhold Schneider. *Tensor Spaces and Hierarchical Tensor Representations*. Cham: Springer International Publishing, 2014, pp. 237–261. ISBN: 978-3-319-08159-5. DOI: [10.1007/978-3-319-08159-5_12](https://doi.org/10.1007/978-3-319-08159-5_12). URL: https://doi.org/10.1007/978-3-319-08159-5_12.
- [HW17] Benjamin Huber and Sebastian Wolf. *Xerus - A General Purpose Tensor Library*. <https://libxerus.org/>. 2014–2017.
- [IRZ20] Kazufumi Ito, Christoph Reisinger, and Yufei Zhang. “A Neural Network-Based Policy Iteration Algorithm with Global H^2 Superlinear Convergence for Stochastic Games on Domains”. In: *Foundations of Computational Mathematics* (2020), pp. 1–44.

- [KDK13] B. Kafash, A. Delavarkhalafi, and S.M. Karbassi. “Application of variational iteration method for Hamilton-Jacobi-Bellman”. In: *Applied Mathematical Modelling* 37.6 (2013), pp. 3917–3928. ISSN: 0307-904X. DOI: <https://doi.org/10.1016/j.apm.2012.08.013>. URL: <http://www.sciencedirect.com/science/article/pii/S0307904X12004696>.
- [KK18] Dante Kalise and Karl 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. ISSN: 1064-8275. DOI: [10.1137/17M1116635](https://doi.org/10.1137/17M1116635). URL: <https://doi.org/10.1137/17M1116635>.
- [Kle70] David Kleinman. “An easy way to stabilize a linear constant system”. In: *IEEE Transactions on Automatic Control* 15.6 (1970), pp. 692–692.
- [KW17] Wei Kang and Lucas C Wilcox. “Mitigating the curse of dimensionality: sparse grid characteristics method for optimal feedback control and HJB equations”. In: *Computational Optimization and Applications* 68.2 (2017), pp. 289–315.
- [Lan12] Joseph M Landsberg. “Tensors: geometry and applications”. In: *Representation theory* 381.402 (2012), p. 3.
- [LB98] J. Lawton and R. W. Beard. “Numerically efficient approximations to the Hamilton-Jacobi-Bellman equation”. In: *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*. Vol. 1. June 1998, 195–199 vol.1. DOI: [10.1109/ACC.1998.694657](https://doi.org/10.1109/ACC.1998.694657).
- [Luo+14] Biao Luo et al. “Data-based approximate policy iteration for affine nonlinear continuous-time optimal control design”. In: *Automatica* 50.12 (2014), pp. 3281–3290. ISSN: 0005-1098. DOI: <https://doi.org/10.1016/j.automatica.2014.10.056>. URL: <http://www.sciencedirect.com/science/article/pii/S0005109814004373>.
- [NGK19] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. “Adaptive deep learning for high-dimensional Hamilton-Jacobi-Bellman equations”. In: *arXiv preprint arXiv:1907.05317* (2019).
- [NR20] Nikolas Nüsken and Lorenz 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).
- [Ose11] Ivan V Oseledets. “Tensor-train decomposition”. In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.
- [OSS19] Mathias Oster, Leon Sallandt, and Reinhold Schneider. “Approximating the Stationary Hamilton-Jacobi-Bellman Equation by Hierarchical Tensor Products”. In: *arXiv preprint arXiv:1911.00279* (2019).

- [OT09] Ivan V Oseledets and Eugene E Tyrtyshnikov. “Breaking the curse of dimensionality, or how to use SVD in many dimensions”. In: *SIAM Journal on Scientific Computing* 31.5 (2009), pp. 3744–3759.
- [PB79] Martin L. Puterman and Shelby L. Brumelle. “On the Convergence of Policy Iteration in Stationary Dynamic Programming”. In: *Mathematics of Operations Research* 4.1 (1979), pp. 60–69. ISSN: 0364765X, 15265471. URL: <http://www.jstor.org/stable/3689239>.
- [Pha+05] Huy en Pham et al. “On some recent aspects of stochastic control and their applications”. In: *Probability Surveys* 2 (2005), pp. 506–549.
- [Pon+62] Lev Semenovich Pontryagin et al. *The mathematical theory of optimal processes*. New York, NY: Translated from the Russian by K. N. Trirogoff; edited by L. W. Neustadt. Wiley, 1962.
- [QB03] S Joe Qin and Thomas A Badgwell. “A survey of industrial model predictive control technology”. In: *Control engineering practice* 11.7 (2003), pp. 733–764.
- [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. ISSN: 0021-9991. DOI: <https://doi.org/10.1016/j.jcp.2018.10.045>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999118307125>.
- [RSN21] Lorenz Richter, Leon Sallandt, and Nikolas N usken. “Solving high-dimensional parabolic PDEs using the tensor train format”. In: *arXiv preprint arXiv:2102.11830* (2021).
- [SC08] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 0387772413.
- [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*. Dec. 1977, pp. 749–754. DOI: [10.1109/CDC.1977.271669](https://doi.org/10.1109/CDC.1977.271669).
- [SU09] Winfried Sickel and Tino 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] Szil ard Szalay et al. “Tensor product methods and entanglement optimization for ab initio quantum chemistry”. In: *International j. of quantum chemistry* 115.19 (2015), pp. 1342–1391. ISSN: 1097-461x. DOI: [10.1002/qua.24898](https://doi.org/10.1002/qua.24898).

- [TAK17] Daniela Tonon, Maria Aronna, and Dante Kalise. *Optimal Control: Novel Directions and Applications*. Springer, Jan. 2017. DOI: [10.1007/978-3-319-60771-9](https://doi.org/10.1007/978-3-319-60771-9).
- [Vap92] Vladimir Vapnik. “Principles of risk minimization for learning theory”. In: *Advances in neural information processing systems*. 1992, pp. 831–838.
- [ZHL21] Mo Zhou, Jiequn Han, and Jianfeng 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] Mingyuan Zhong et al. “Value function approximation and model predictive control”. In: *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE, 2013, pp. 100–107.

A Proof

proof of Lemma 5. Towards a contradiction assume that there exists an initial value x such that $|v_1(x) - v_2(x)| > \delta$. W.l.o.g. assume that $v_1(x) - v_2(x) > \delta$. Due to the definition of v_2 , for every $\varepsilon > 0$ there is a control u_2 such that

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

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

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

Now it follows that

$$\begin{aligned} \delta < v_1(x) - v_2(x) &\geq \tilde{v}_1(x) - v_2(x) \leq \int_{t_0}^{t_1} \ell(y_2, u_2) dt + c_1(y_2(t_1)) - \left(\int_{t_0}^{t_1} \ell(y_2, u_2) dt + c_2(y_2(t_1)) \right) + \varepsilon \\ &= c_1(y_2(t_1)) - c_2(y_2(t_1)) + \varepsilon. \end{aligned}$$

Since this holds for every $\varepsilon > 0$, this contradicts $\|c_1 - c_2\|_\infty < \delta$.

□

B Fast gradient evaluation

We now show how to compute the gradient of a function in TT-representation in complexity $\mathcal{O}(dmr^2)$. For this, we define the contractions

$$\psi_i^+(x_{i+1}, \dots, x_d) = u_{i+1} \circ \dots \circ u_d \circ \phi(x_d) \cdots \circ \phi(x_{i+1}) \quad (34)$$

$$\psi_i^-(x_1, \dots, x_{i-1}) = u_1 \circ \dots \circ u_{i-1} \circ \phi(x_{i-1}) \cdots \circ \phi(x_1). \quad (35)$$

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

$$\frac{\partial v}{\partial x_i}(x_1, \dots, x_d) = \psi_i^- \circ u_i \circ \psi_i^+ \circ \phi'(x_i). \quad (36)$$

Finally, the recursive properties

$$\psi_i^+ = u_{i+1} \circ \psi_{i+1}^+ \circ \phi(x_{i+1}), \quad \psi_i^- = \phi(x_{i-1}) \circ (\psi_{i-1}^- \circ u_{i-1}) \quad (37)$$

yields Algorithm 4.

Algorithm 4: Computing the gradient of a function v in TT-format.

input : A function v in TT-format, component tensor u_1, \dots, u_d , one-dimensional basis functions ϕ_1, \dots, ϕ_n , point $x = (x_1, \dots, x_d) \in \mathbb{R}^d$.

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

```

1 for  $\mu = 1, \dots, d - 1$  do
2   | Calculate  $\psi_i^-$  using the recursive formula (37).
3 end
4 for  $\mu = d, \dots, 1$  do
5   | Calculate  $\psi_i^+$  using the recursive formula (37).
6 end
7 for  $\mu = d, \dots, 1$  do
8   | Calculate  $\nabla v(x)[i] = \frac{\partial v}{\partial x_i}(x)$  by using (36).
9 end

```

Note that the computational complexity of (36) and (37) is $\mathcal{O}(mr^2)$. Thus, Algorithm 4 has computational complexity of $\mathcal{O}(dmr^2)$. This complexity compares to the complexity of the naive implementation $\mathcal{O}(d^2mr^2)$.