A MCMC Sampling Approach for Learning Continuous-Time Optimal Control Laws

Balduin Dettling (D-ITET, dbalduin@ethz.ch)
Supervision:
Lenart Treven (D-INFK, lenart.treven@inf.ethz.ch),
Bhavya Sukhija (D-INFK, bhavya.sukhija@inf.ethz.ch)

Semester Project – Spring 2023

Abstract

We present an approach for approximately solving continuous-time, deterministic optimal control problems over large regions of the state space. The task is separated into data generation and supervised learning, where our main contribution is a principled and efficient approach to data generation. We avoid any optimisation over policies or input sequences, and instead use Pontryagin's minimum principle (PMP) together with a MCMC sampling algorithm to directly find many optimal trajectories. We apply the presented methods to solve two simple problems, and discuss how they might be extended to handle more challenging tasks.

1 Introduction

Optimal control is a general, yet intuitive approach for specifying control laws for general nonlinear feedback systems and is widespread in practical applications. However, explicit evaluation of the optimal control input is nontrivial. Practical approaches mostly fall into two categories, namely Model Predictive Control (MPC) and Reinforcement Learning (RL).

MPC is the practice of repeatedly solving an approximation of the optimal control problem at hand in real-time, and applying the first input. For general systems, this is nontrivial and guarantees about convergence, optimality and runtime are hard to obtain. Therefore, in practice, MPC is limited to either relatively simple systems, or ones where significant computational power is available online.

RL on the other hand seeks to learn a controller while interacting with the system. Depending on the specific formulation and solution method, it can work with very general systems. Typically, a discrete-time formulation with infinite horizon and discounted rewards is used. However, most RL methods are computationally quite expensive. If online computational power is limited, RL methods can be used to train a controller purely in simulation, which is then deployed on the real system without further learning and adaptation. If applied in this style, an RL algorithm can be considered a tool for model-based optimal control, with the simulator being the model.

Here we explore an alternative approach of solving continuous-time, deterministic, finite-horizon optimal control problems for a known dynamics model. Our main design goals are cheap online evaluation of the optimal controller, and reliable data generation for training. Using Pontryagin's minimum principle together with a MCMC sampling algorithm, we efficiently find many optimal trajectories without optimising over policies or input sequences, enabling us to then use standard supervised learning tools to learn the optimal controller.

1.1 Literature Review

Numerous research efforts have aimed at widening the class of problems for which approximate optimal control can be practically calculated. We provide a short (non-exhaustive) overview here.

One theme is the combination of MPC and machine learning, where an MPC control law is evaluated at many points, to then use supervised learning for predicting the control law at new points [1]. When an MPC control law is available but computationally too heavy for online evaluation, this approach is especially appealing.

There are works in the RL domain which aim at solving continuous-time problems. Examples include continuous-time pendants of policy gradient methods [2, 3] or Q-learning [4]. For an overview of foundational theory and the differences to the discrete-time setting, we refer to [5] and [6].

Another group of methods attempts to directly find an approximate solution to the HJB equation based on an available dynamics model, instead of taking the "detour" through closed-loop simulations. They use physics-informed neural networks (PINNs) [7] to find a function that minimises a loss related to the satisfaction of the HJB equation and its boundary condition. These tools have been used to address finite-horizon optimal control or reachability problems [8, 9, 10], to find a stationary point of the infinite-horizon discounted version of the HJB equation [11], or to learn a change of variables that simplifies planning of optimal trajectories [12]. However, a larger body of research is concerned with characteristics-based approaches. The characteristic curves of the HJB equation coincide with the solutions of the well-known Pontryagin minimum principle ([13], Section 7.2), and thus can be used to obtain locally optimal trajectories while maintaining a very close connection HJB theory.

Recent works include [14, 15, 16, 17, 18, 19] – most of these methods involve first selecting a set of initial states, then solving a two-point boundary value problem at each state to find optimal control inputs. [19] is the only one we found that does away with solving two-point boundary value problems in favour of directly calculating a set of suitable characteristic curves. However, they provide only a problem-specific heuristic for selecting a suitable set of characteristics. We develop this approach further and introduce several improvements.

1.2 Problem Statement

We consider a state-feedback control system with some state variable $x \in \mathcal{X} \subseteq \mathbb{R}^n$, and a control input $u \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$. For simplicity, we assume that it is control affine and time-invariant:

$$\dot{x} = f(x) + g(x)u \tag{1}$$

We want to find a practical, fast approximation of the optimal feedback controller $u^\star : [0, T] \times \mathcal{X} \to \mathcal{U}$ that solves the following deterministic, finite horizon optimal control problem (OCP):

$$\min_{u:[0,T]\times\mathcal{X}\to\mathcal{U}} \quad \int_0^T l(x, u(t, x))dt \tag{2a}$$
$$\text{s.t.} \quad \dot{x} = f(x) + g(x)u(t, x) \quad \forall t \in [0, T] \tag{2b}$$
$$x(T) = 0 \tag{2c}$$

We restrict our attention to approximating the optimal controller $u^\star$ only at $t = 0$. This is motivated by the widespread practice of solving a finite horizon optimal control problem to obtain feedback controls at every time step, commonly known as model predictive control or MPC. We will mostly omit the argument t and write $u^\star(x)$ as a shorthand for $u^\star(0, x)$, and do the same for V and λ.

For this reason, we choose to use terminal constraint (2c) instead of a terminal cost function. Using the former makes the stability proof of MPC-type control simpler.

Moreover, as many states might be completely irrelevant, we further restrict our attention to a compact subset of the state space $\mathcal{S} \subseteq \mathcal{X}$, which in a practical application should contain every state we reasonably expect to visit. In our examples, this is either an ellipsoid or a value sublevel set.

$$\mathcal{S}_e = \{x | x^\top Q_\mathcal{S} x \leq 1\} \tag{3a}$$
$$\mathcal{S}_v = \{x | V(x) \leq V_{\max}\} \tag{3b}$$

The latter choice has the advantage that under optimal control inputs, it is forward invariant. To ensure well-posedness of (2), we make a few assumptions on the problem parameters.

Assumption 1 *f and g are globally Lipschitz over $\mathcal{S}$.*

Assumption 2 *The function $t \mapsto u(t, x(t))$ (with x given by some initial condition $x(0)$ and the ODE (1)) piecewise continuous over $[0, T]$.*

Assumption 3 *l is a convex quadratic form in its second argument u, and $\mathcal{U}$ is a convex polytope.*

Assumption 4 *The system has a stabilisable equilibrium at $(x, u) = (0, 0)$. From any initial condition $x(0) \in \mathcal{S}$, there exists an input $u : [0, T] \to \mathcal{U}$ that, when applied to the system, leads to $x(T) = 0$. Furthermore, $l(0, 0) = 0$ and $l(x, u) > 0$ for $(x, u) \neq (0, 0)$*

According to [13], Section 3.3.1, Assumptions 1 and 2 guarantee existence and uniqess of the solutions of (1). Assumption 3 makes the minimisation the Hamiltonian over u (eq. 5) a simple constrained convex optimisation problem. Assumption 4 ensures that the control task is feasible, and that $\lambda_T = 0$ leads to the trajectory $x(t) = 0, u(t) = 0$ being the solution of (7), so we can use it as a starting point for sampling other λ_T.

We will introduce further assumptions after exposing the necessary theory. In summary, Assumption 5 ensures that the partial derivatives of the Hamiltonian H (Eq. 4) exist, which is needed for stating Pontryagin's minimum principle. Assumption 6 (section 2.1) ensures that we can find locally optimal trajectories, and Assumption 7 (section 2.2.1) narrows down the class of problems, so that all locally optimal trajectories are globally optimal.

2 Methods

Continuing a recent line of research [18, 19], we split the problem into two main parts: *data generation* and *supervised learning*.

For the first part, *data generation*, we use the Pontryagin minimum principle (PMP) to efficiently find optimal controls for many initial states x_0 (Section 2.1). We frame this whole step as taking samples from a certain probability distribution, which we achieve by a Markov Chain Monte Carlo

(MCMC) method specifically adapted to the unique challenges of this situation (Section 2.2).

The second part, *supervised learning* consists of finding a function that fits the generated data and generalises to unseen data points. This type of problem is very well-researched at this point – we use mostly the same approach as that presented in [18] (Section 2.3).

2.1 Pontryagin Minimum Principle

The PMP [20, 13] arises as an ODE describing the characteristic curves of the HJB PDE, with its solution being a locally optimal control input and state trajectory.

To state it, we first define the Hamiltonian H and the optimal control input $u^\star$:

$$H(x, u, \lambda) = l(x, u) + \lambda^\top (f(x) + g(x)u) \tag{4}$$

$$u^\star(x, \lambda) = \arg\min_u H(x, u, \lambda) \tag{5}$$

Most often [13, 18], the PMP is presented as a boundary value problem with split boundary conditions. Here is the version with initial and terminal state constraints:

$$\dot{x} = \frac{\partial H(x, u^\star, \lambda)}{\partial \lambda}, \quad x(0) = x_0, \quad x(T) = 0 \tag{6a}$$

$$\dot{\lambda} = -\frac{\partial H(x, u^\star, \lambda)}{\partial x} \tag{6b}$$

We assume that the necessary derivatives are well defined:

Assumption 5 *l is continuously differentiable w.r.t. its first argument x over $\mathcal{X} \times \mathcal{U}$, and f and g are continuously differentiable over $\mathcal{X}$.*

In (6), we omit obvious arguments of the functions x, $u^\star$, and λ to shorten notation. To obtain locally optimal control inputs at some state x_0, (6) can be solved by root finding methods. These generally work by finding an initial condition $\lambda(0)$ that leads to satisfaction of the terminal constraint $x(T) = 0$ after solving the ODE. These approaches are known as *shooting methods.*

The above formulation is most natural when searching for trajectories starting at some specific state, which is the usual situation in predictive control applications. However, in this work we are not concerned with finding optimal controls for a single initial state, but for a large set of initial states. Therefore we find it more convenient to move all boundary conditions to $t = T$, an idea previously seen in [19]. We simultaneously solve for the value function along the trajectory with $v(t) = V(t, x(t))$[1]:

$$\dot{x} = \frac{\partial H(x, u^\star, \lambda)}{\partial \lambda}, \quad x(T) = 0 \tag{7a}$$

$$\dot{\lambda} = -\frac{\partial H(x, u^\star, \lambda)}{\partial x}, \quad \lambda(T) = \lambda_T \tag{7b}$$

$$\dot{v} = -l(x, u^\star), \quad v(T) = 0 \tag{7c}$$

Thus, we can find $V(x_0)$ and $\lambda(x_0)$ at some initial state x_0 with a single ODE solver call (solving the ODE backwards in time). This is at the expense of not knowing exactly which initial state we will end up at – to compensate we will have to sample many terminal costates λ_T, which is needed anyway.

We assume both existence and uniqueness of the solution to (7) over $[0, T]$. We define the function $\mathrm{PMP} : \lambda_T \mapsto x_0$, given by evaluating the resulting state trajectory at $t = 0$. A sufficient condition for existence and uniqueness of solutions to some ODE is global lipschitzness of the its RHS [21].

Alternatively, we can relax the requirement and be content with local lipschitzness. In that case, solutions are well defined if they don't leave the domain $\mathcal{S}$. This is hard to verify in advance, but we found in practice that the methods presented here are quite successful at avoiding problems related to finite escape time and will work with this assumption from now on.

Assumption 6 *The function $(x, \lambda, v) \mapsto \left(\frac{\partial H}{\partial \lambda}, -\frac{\partial H}{\partial x}, -l\right)$ (with arguments according to Eq. 7) is globally Lipschitz over the domain $\mathcal{S} \times \mathbb{R}^n \times [0, \infty)$.*

2.2 Sampling of optimal trajectories

The Pontryagin principle allows us to go from some terminal costate λ_T to an initial state $x_0 = \mathrm{PMP}(\lambda_T)$, by solving (7). In doing so, we find the value $V(x_0)$ and costate $\lambda(x_0)$, from which we can find the desired optimal control input with (5). However, it is still unclear which terminal costates λ_T we should use to arrive at relevant x_0.

Previous works have applied different heuristics for this kind of task. In [19], the problem of stabilising a nominal interplanetary transfer trajectory with optimal feedback control is addressed. The authors perform backward integration of the PMP, just like in our work. They use terminal costates λ_T sampled from some small ball – This works well for the problem at hand, but for general systems, there is no reason to expect that this procedure leads to a useful distribution of initial states. In [17], the authors set up a random walk that traverses the space of initial conditions x_0, and because at each iteration the initial condition only changes slightly, they are able to update λ_T by solving the two-point BVP with the previous solution as initial guess.

We combine ideas from these two works, and present the resulting method in the following subsections.

[1]Starting from $V_t(t, x(t)) = -l(x, u^\star) - V_x(t, x(t))\dot{x}(t)$ (the HJB equation, [13], eq. 5.11, after inserting optimal u), add the $V_x\ \dot{x}$ term on both sides, giving the total derivative $\frac{d}{dt}V(t, x(t))$ on the left side.

2.2.1 Sampling problem

We view the problem of finding relevant λ_T as that of drawing samples from a suitable probability distribution. We start by specifying some distribution $p(x_0) = \frac{1}{Z}\exp(r(x_0))$ over the state space, from which we would like the training data to come from. It is determined by its log probability density function r, which can encode different goals. With a simple exterior penalty term, we approximate the uniform distribution over the relevant state space subset $\mathcal{S}$ (3):

$$r_e(x) = -10\max(0, x^\top Q_{\mathcal{S}} x - 1) \tag{8a}$$

$$r_v(x) = -10\max(0, V(x) - V_{\max}) \tag{8b}$$

To find V, λ and $u^\star$ at many initial states, we cannot just sample $x_0 \sim p$, instead we need to sample suitable λ_T and evaluate $x_0 = \mathrm{PMP}(\lambda_T)$. To simplify this task, we assume that the function PMP is bijective. More precisely:

Assumption 7 *For all $x_0 \in \mathcal{S}$, there exists exactly one λ_T such that $x_0 = \mathrm{PMP}(\lambda_T)$. In addition, the Jacobian $\frac{d\,\mathrm{PMP}(\lambda_T)}{d\lambda_T}$ also invertible.*

This makes it possible to uniquely define the probability density g such that sampling $\lambda_T \sim g$ leads to $x_0 = \mathrm{PMP}(\lambda_T) \sim p$ as desired. We apply the usual change of variables formula to PMP^{-1}, obtaining:

$$g(\lambda_T) = p(\mathrm{PMP}(\lambda_T)) \left| \det\left[\frac{d\,\mathrm{PMP}(\lambda_T)}{d\lambda_T}\right]\right| \tag{9}$$

Unfortunately, this bijectivity assumption rules out many problems, namely all those for which there are distinct locally optimal trajectories. In addition, it is hard to verify practically whether the assumption holds. Although we assumed its existence in the derivation, PMP^{-1} itself does not show up in (9). This begs the question: What would happen if we were to naively apply our method to a problem violating the bijectivity assumption? In Section 4.3.1 we discuss briefly what might happen in this case.

2.2.2 Solution of the sampling problem

We want to draw samples from this distribution g. For this we start by considering a standard Metropolis-Hastings algorithm, a relatively simple MCMC method, given in Algorithm 1 [22].

Algorithm 1 Standard Metropolis-Hastings algorithm

```
Require: Target distribution $g(\lambda)$
Require: Proposal distribution $q(\lambda'|\lambda)$
Require: Initial point $\lambda_0$
 1: for $k = 0, 1, \dots$ do
 2:     Draw proposal $\hat{\lambda}_{k+1} \sim q(\cdot|\lambda_k)$
 3:     Calculate $H = \frac{g(\hat{\lambda}_{k+1})q(\lambda_k|\hat{\lambda}_{k+1})}{g(\lambda_k)q(\hat{\lambda}_{k+1}|\lambda_k)}$
 4:     Draw $u \sim \mathrm{Uniform}([0,1])$
 5:     if $u < H$ then
 6:         $\lambda_{k+1} \leftarrow \hat{\lambda}_{k+1}$
 7:     else
 8:         $\lambda_{k+1} \leftarrow \lambda_k$
 9:     end if
10: end for
```

The accept-reject step ensures the detailed balance condition, which in turn ensures the correct stationary distribution. This iteration therefore converges asymptotically to the desired distribution g by design. However, it is difficult to make concrete statements about how quickly the distribution is reached.

Practically, the success of an MCMC sampler depends crucially on the choice of proposal distribution. Guided by intuition, we choose a specific proposal distribution, which we will explain here.

The main challenge in writing a sampler for the density g is its variable geometry. More precisely, the PMP function and its inverse can significantly alter the shape of sets. In Fig. 3, an example is shown where sampling from an ellipsoid-like set in the state space necessitates sampling from a pre-image in the λ_T space that looks very different. Particularly the long narrow funnels are challenging for standard algorithms with constant proposal distributions.

However, we can gain insights about the local distortion of the state space through the jacobian $\frac{d\,\mathrm{PMP}(\lambda)}{d\lambda}$. Intuitively, it tells us how a small distribution centred at λ transforms through PMP to another small distribution over x_0 centred at $\mathrm{PMP}(\lambda)$. We invert this linear relationship, allowing us to define a desired proposal distribution over x_0 and then convert it to a corresponding distribution over λ. We choose to approximate a Langevin diffusion over x_0, discretised with the Euler-Maruyama scheme and discretisation time τ:

$$q(\lambda'|\lambda) = \mathcal{N}(\lambda'; \mu(\lambda), \Sigma(\lambda)) \tag{10a}$$

$$\mu(\lambda) = \lambda + J\left(\tau \nabla_x \log p(x)|_{x=\mathrm{PMP}(\lambda)}\right) \tag{10b}$$

$$\Sigma(\lambda) = J\,(2\tau\Sigma_x)\,J^\top \tag{10c}$$

where $J = \left[\frac{d\,\mathrm{PMP}(\lambda)}{d\lambda}\right]^{-1}$, and Σ_x is a matrix containing (very roughly) the variances and covariances of the state vectors we visit. The proposal distribution over the state space is linearly transformed to one over λ through the inverse Jacobian J. As $\tau \to 0$, this linear approximation becomes exact, but more importantly, we don't need it to be exact: The accept-reject step ensures that we have the desired stationary distribution over λ_T (and thus by construction also the desired distribution over x_0). We can thus choose relatively large discretisation times τ to ensure good mixing.

This proposal density essentially makes the problem of MCMC sampling from some distribution whose shape we don't really know (the pre-image of $\mathcal{S}$ under the map PMP) as easy as that of sampling from the much more well-behaved set $\mathcal{S}$ itself.

The forward proposal density at iteration k, $q(\cdot|\lambda_k)$ is already calculated in the previous iteration: If in iteration $k-1$ the proposal is accepted, then the new forward proposal density is equal to the previous backward transition density $q(\cdot|\hat{\lambda}_{(k-1)+1})$. If the proposal is rejected, we have $\lambda_k = \lambda_{k-1}$, so the proposal distribution stays the same. Therefore, we reuse the transition information from the previous iteration, cutting runtime in half compared to the naive implementation.

2.3 Gradient-enabled NN ensemble

To approximate the underlying value function and optimal controls using NNs, we use mostly standard methods, which we briefly go over here. We refer to the NN-approximated value function as $V_{\mathrm{NN}}(x;\theta)$, where $x \in \mathcal{X}$ is a point in state space and θ is a vector collecting all weights and biases of the NN.

We have training data $\mathcal{D} = \{(x^{(i)}, V^{(i)}, \lambda^{(i)})\}$. If we wanted to just approximate the value function, then a loss function of $l(\theta) = \mathbb{E}_i\left[(V_{\mathrm{NN}}(x^{(i)},\theta) - V^{(i)})^2\right]$ would be appropriate. However, we are mainly interested in the value gradient $\nabla_x V(x) = \lambda(x)$, rather than the value function – this becomes apparent when considering (5), where the optimal control directly depends on the costate.

We could directly approximate the costate $\lambda(x)$ using a NN with n-dimensional output. Similarly, we could directly approximate the optimal input $u^\star$ after evaluating (5) at all training data points. But then there might not exist a value function V such that has our approximated costates as its gradients. We would be leaving information on the table by ignoring $V(x)$. For moderately sized datasets, both [18] and [19] report the best results by modelling a scalar value function, but incorporating gradient information into the loss function:

$$l(\theta) = \mathbb{E}\left[\frac{(V_{\mathrm{NN}}(x;\theta) - V)^2 + \mu||\nabla_x V_{\mathrm{NN}}(x;\theta) - \lambda||_2^2}{1+\mu}\right] \quad (11)$$

We adopt this choice, choosing $\mu > 1$ to reflect the fact that we prefer a good approximation of λ over that of V. The existence of a value function acts as an inductive bias.

The expectation in Eq. 11 is taken over training samples $(x, V, \lambda) \in \mathcal{D}$ and approximated during training with minibatches. The gradient $\nabla_x V_{\mathrm{NN}}$ is calculated with the same autodifferentiation framework used to calculate $\nabla_\theta V_{\mathrm{NN}}$. Other names for this type of NN model include *physics-informed machine learning*, *gradient regularisation* [18], or *deep differential networks* [11].

To get an estimate of model uncertainty, we use an NN ensemble, instead of just one NN. The ensemble is constructed by applying the same NN architecture, but with N_{ens} different parameter vectors $\Theta = \{\theta_0, ..., \theta_{N_{\mathrm{ens}}}\}$. They are trained together, but starting at a different random initialization. Predictions are then formed by evaluating the NN for all $\theta \in \Theta$, to obtain a predictive distribution consisting of N_{ens} particles. We are mainly interested in the mean and variance of these predictions, and will from now on overload the notation $V_{\mathrm{NN}}(x;\Theta)$ to denote the mean prediction $\frac{1}{N_{\mathrm{ens}}}\sum_{\theta\in\Theta} V_{\mathrm{NN}}(x;\theta)$.

The NN architecture and training hyperparameters are listed in Table 1 in Appendix A. We perform no data normalization, although we suspect it may be beneficial if the scale of the data is farther away from unity.

3 Results

We test the presented methods with two small toy examples: a linear double integrator with unconstrained input, and one with cubic friction and an additional input constraint.

3.1 Evaluation procedure

For each of the two test problems, we evaluate our methods as follows. First, we use the MCMC sampling approach from section 2.2 with the value-based reward function (8b) to sample from the uniform distribution over $\mathcal{S} = \{x | V(x) \leq V_{\max}\}$. This gives us a large training dataset $\mathcal{D} = \{(x^{(i)}, V^{(i)}, \lambda^{(i)})\}$ with 16384 data points.

We then approximate the value function and its gradients using NN ensemble approach from section 2.3. To get an empirical estimate of the sample complexity, we train the ensemble using increasing amounts of training data, all subsampled from our full dataset $\mathcal{D}$. To ensure that each NN ensemble is trained properly, we always use a training data set the size of the complete dataset, but construct it by repeating a small subsample the appropriate number of times. We then find hyperparameters that work well in terms of test error on the full dataset using well known tuning heuristics. This means that we spend more computation than necessary, especially for the smaller data sets. In turn, we have relatively comparable training dynamics across all experiments.

For every trained NN ensemble, we calculate two performance metrics. First, the costate test loss $\mathbb{E}\left[||\nabla_x V_{\mathrm{NN}}(x;\Theta) - \nabla_x V(x)||_2^2\right]$ is evaluated on a fixed test set of 256 points, which we set aside before all training experiments. Second, we estimate the expected closed loop control cost, defined by:

$$J_{\mathrm{control}} = \mathbb{E}_{x_0}\left[\int_0^{T_{\mathrm{sim}}} l(x(t), \tilde{u}^\star(x(t))\right] \quad (12)$$

where the trajectories $x(t)$ are found by forward simulation of $\dot{x} = f(x, \tilde{u}^\star(x))$, $x(0) = x_0$. The expectation over x_0 is approximated by a Monte Carlo estimate over a set of 32 different initial conditions, which are drawn from the test set before all experiments and then kept constant. $\tilde{u}^\star$ refers to our approximate optimal controller, obtained by substituting $\lambda(x) = \nabla_x V_{\text{NN}}(x; \Theta))$ into Eq. (5). Instead of J_{control} itself, we show $(J_{\text{control}} - J_{\text{baseline}})/J_{\text{baseline}}$ in the plots, a measure of suboptimality with respect to the optimal baseline sometimes called *relative regret*.

Notice that J_{control} approximates the infinite-horizon control cost with respect to l, whereas the original OCP (2) demands minimisation of the finite-horizon control cost subject to a terminal constraint. This is commonplace in MPC applications, and the practical implications of this mismatch are well understood. However, this means that in principle, an approximate optimal controller $\tilde{u}^\star$ with just "the right" approximation errors might achieve a lower cost J_{cl} than the exact optimal controller $u^\star$.

To assess repeatability, we repeat this whole experiment with 16 different seeds for the pseudorandom number generator (PRNG). These are responsible for initialization of the NN ensemble, and for shuffling the training data before each experiment run. An overview of all the algorithm parameters used is given in Table 1 (Appendix A).

3.2 Linear example

We introduce a simple linear control problem, to compare the results of our approach to the well-known LQR solution. We have a two-dimensional state variable $x = [p \quad v]^\top$ and a scalar input u. The dynamics are

$$\dot{p} = v \qquad \dot{v} = u \tag{13}$$

We specify the control task of stabilising this system to the origin by setting $l(x, u) = x^\top x + u^\top u$ with a time horizon of $T = 8$. Because of the terminal constraint $x(T) = 0$, we do not need a terminal cost function.

We generate a training dataset using the MCMC sampling approach, a subsample of which is shown in Fig. 1.

Because this first problem is linear and unconstrained, we can find the LQR solution and compare it to the output of our methods. Our problem is readily solved by the continuous-time algebraic Riccati equation [23]:

$$-\dot{P} = A^\top P + PA - PBR^{-1}B^\top P + Q \tag{14a}$$
$$P(T) = P_T \tag{14b}$$

$P(t)$ parameterises the value function $V(t, x) = x^\top P(t)x$, and the optimal control input at $t = 0$ is $u^\star(x) = -R^{-1}B^\top P(0)x = -Kx$. We solve this ODE with the `diffrax.Tsit5` [24] solver and a very small step size of 1/1024. To approximate the terminal constraint in our original problem formulation, we set the boundary condition P_T to a relatively high $10^5\ \mathbf{I}_2$. However, we notice that in this specific example, the boundary condition barely influences the results at $t = 0$, which are very similar to the infinite-horizon LQR controller K_∞. We get numerical values of $K = [1.00000026, 1.73205015]$ and $K_\infty = [1.0, 1.73205081]$. This indicates that our horizon T is long enough for the control problem to very closely approximate the infinite-horizon control problem.

We evaluate the finite-horizon LQR controller K with the same procedure and the same set of initial conditions as our NN controllers. The resulting costate test losses and relative regrets are shown in Fig. 2.

3.3 Nonlinear example

We modify the problem introduced in the previous section to make it slightly harder. First, we add a nonlinear friction term:

$$\dot{p} = v \qquad \dot{v} = -v^3 + u \tag{15}$$

Additionally, we impose an input constraint $u \in \mathcal{U} = [-1, 1]$. This changes the expression for the optimal input $u^\star$ (Eq. 5) to a constrained convex optimisation problem. Because it is one-dimensional it is easily solved by finding the unconstrained solution and projecting it onto $\mathcal{U}$. The control cost, time horizon and terminal constraint are the same as in the linear example.

Like before, we generate training data with our MCMC sampling approach. The resulting data set is visualised in Fig. 3. To compare our approach to a baseline, we also make an attempt to find the actual optimal control inputs at each x. This is done by solving the BVPs associated with the PMP (7) to reach the desired x at $t = 0$. The ODE solver used is still `diffrax.Tsit5`, but with a lower step size of 1/128 to reduce the integration error. To solve the root finding problem, we apply a batched Newton method, initialised with the terminal costate of 8 different samples from the training data set with x_0 closest to x. Unfortunately, the Newton iteration is not completely reliable: sometimes it diverges and sometimes it oscillates around the desired x. Therefore, if we don't find any λ_T with $||x - \text{PMP}(\lambda_T)||_2 \leq 10^{-6}$, we apply a fallback strategy of taking the 8 best samples encountered during the Newton iteration, fitting a linear function to the corresponding costates, and using that to produce a sensible guess of the costate $\lambda(x)$. After finding $\lambda(x)$ with either method, we find the optimal control input using (5). While this does not guarantee that we find the optimal control input, it comes quite close and presents a sensible baseline with which we can compare our approach.

We again conduct the same experiment to evaluate the performance of controllers obtained with different amounts of training data. The results of this experiment can be seen in 4.

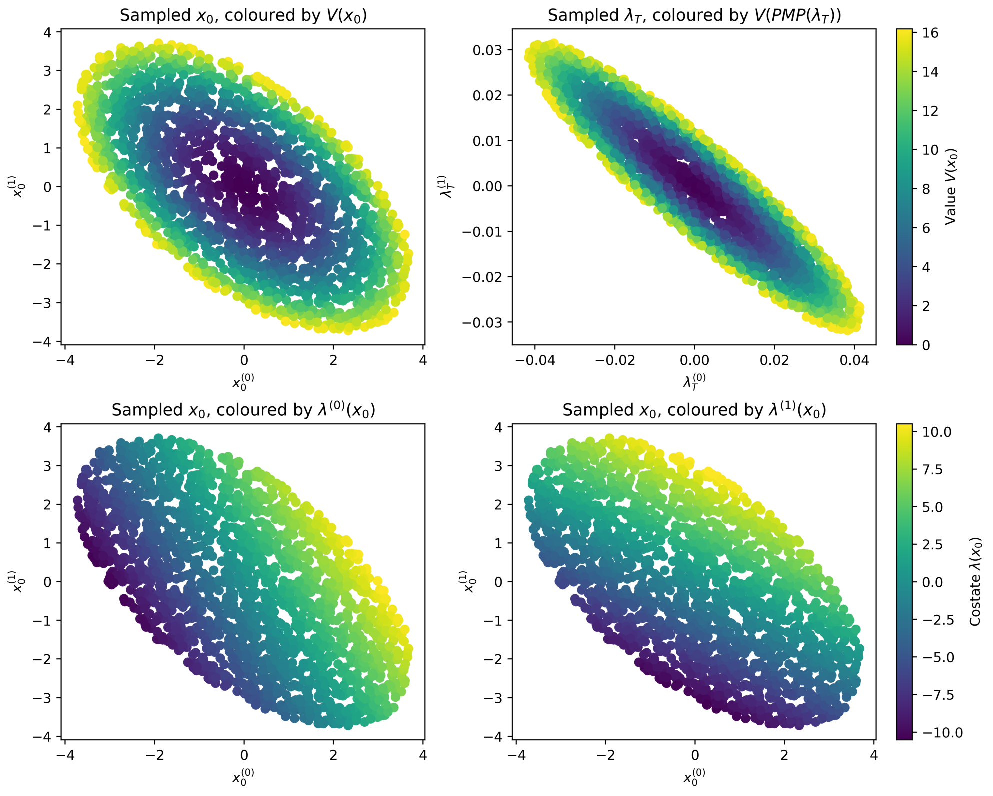

Figure 1: A size 2048 uniform subsample of the generated training data for the linear control problem.

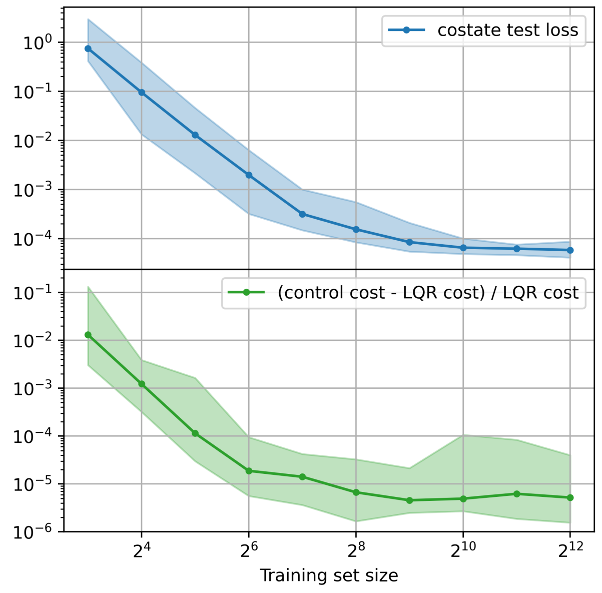

Figure 2: Costate test losses and relative regrets for the linear example. The lines are medians over 16 PRNG seeds, and shaded areas show the min/max range.

4 Discussion

4.1 Data generation

We observe in the two experiments that the MCMC sampler for generating training data works well. We confirm visually that the samples cover the interesting set $\mathcal{S}$ uniformly.

As expected for a linear problem (Fig. 1), the value sublevel set looks like an ellipse, the value function is quadratic in x and the costates are linear in x. The set of sampled λ_T has these same properties, as in this case the whole PMP function is linear. For this task, the MCMC sampler works very well – we see high acceptance rates and great mixing. This is to be expected, as the distribution $g(\lambda_T)$ is uniform over an ellipse, just like $p(x_0)$.

The nonlinear problem (Fig. 3) leads to a more complicated shape, which was unknown a priori. Interestingly, to achieve a uniform distribution over $\mathcal{S}$, we need a distribution over λ_T that is far from uniform. For example, we sample at a much lower density around $\lambda_T = 0$ than in some regions near the edge of the pre-image of $\mathcal{S}$. It is also apparent that thanks to the non-stationary proposal distribution, the long thin spikes in the distribution over λ_T are also well-explored.

Still, the sampling task for this problem proved challenging. Even though our non-stationary proposal distribution helped, we observed low acceptance rates between 0.1 and 0.12, necessitating rather long MCMC chains, small τ, and

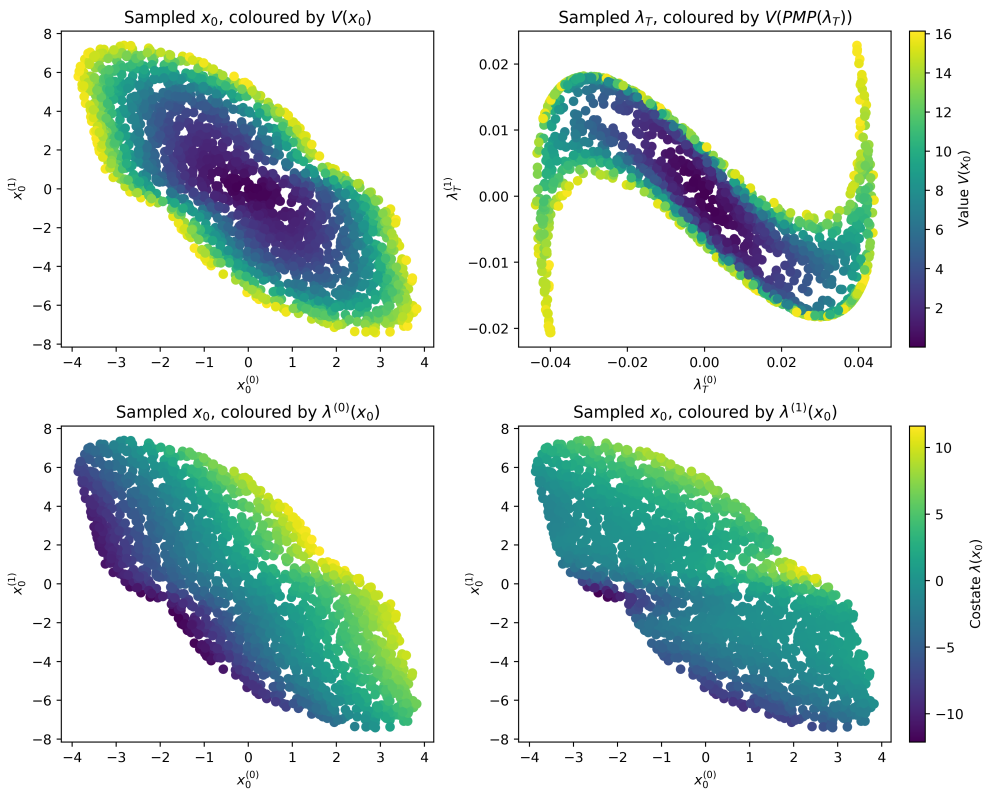

Figure 3: A size 2048 uniform subsample of the generated training data for the nonlinear control problem.

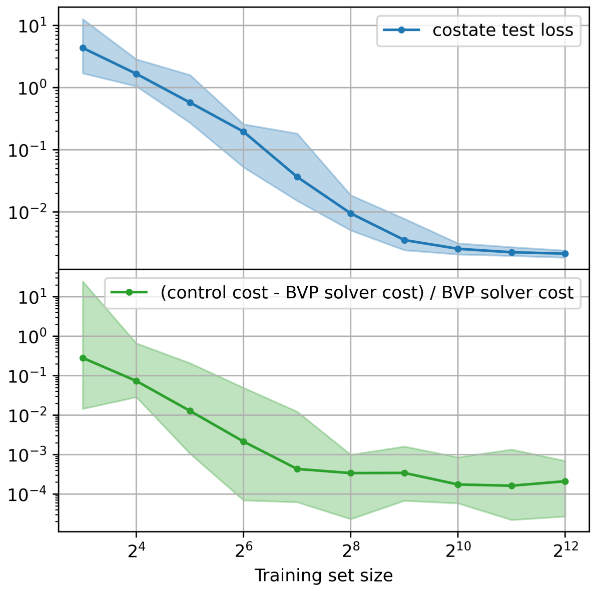

Figure 4: Costate test losses and relative regrets for the nonlinear example. The lines are medians over 16 PRNG seeds, and shaded areas show the min/max range.

large subsampling rates to generate a good dataset.

Additionally, we observed that for the nonlinear example, we need a relatively small ODE solver time step of 1/64 to solve (7). Otherwise, we get training data with obvious numerical artefacts like discontinuities or outliers.

These two issues made the data generation task for the nonlinear example relatively expensive, taking about 25 minutes on an i5-4570 quad-core CPU running at 3.20GHz for the needed 16384 samples (with the use of `jax.jit`). This could be improved by further optimising the sampler, by using a more efficient adaptive ODE solver, or with the use of parallel computing.

4.2 Learning & control

In Figs. 2 and 4, we see that the costate test loss decreases both times as we add more training data, then stabilises at low values after about 1024 training data points.

The low test loss indeed translates to low control cost, as we would hope. In the linear example, we achieve a median relative regret of 10^{-4} with only 32 training points, and beyond 256 points the median is consistently below 10^{-5} (meaning at most 0.001% worse than the LQR solution). This serves as a sanity check and confirms that our approach can recreate the optimal solution for such a simple problem very accurately.

For the nonlinear example, the results look very similar, except that both the costate test loss and the relative regret do not go quite as low. Still, we find a controller that performs very close to the baseline.

Interestingly, in each example, there seems to be a range of training set sizes where the costate approximation error still decreases with more data, but the relative regret already starts flattening out. While we do not have a good explanation for this phenomenon, it raises hope that our approach might do well for higher dimensional systems, where we might not be able to generate so much training data that adding more won't improve the test loss.

4.3 Improvements

These results are great and indicate that larger problems could also be solved similarly. In principle, nothing stops us from applying our approach to larger problems, as it is completely grid-free and data generation is easily parallelisable. However, practical and theoretical challenges must still be addressed before our approach can reliably handle complex real-world control problems.

Here, we discuss some limitations of our methods, which could be improved in further research. For anyone interested, we published all code used for our experiments: `https://github.com/mbjd/approximate_optimal_control`.

4.3.1 Local vs. global optimality

The main drawback is Assumption 7 requiring that the PMP function is bijective. This means that all locally optimal trajectories are also globally optimal. Nonconvexities and resulting multiple distinct local solutions are commonplace in real-world optimal control problems, and solving them in an even somewhat sound way would be very useful.

We can try to apply our approach anyway when PMP is not invertible. Several problems will arise: First, we lose the guarantee that sampling from $\lambda_T \sim g$ (Eq. 9) leads to $x_0 = \mathrm{PMP}(\lambda_T)$ haing the desired distribution p. Moreover, while g is still well-defined, the proposal distributions $q(\lambda'|\lambda)$ from (10) are not necessarily so, because the Jacobian $\frac{d\,\mathrm{PMP}(\lambda_T)}{d\lambda_T}$ might not be invertible anymore. This also leads to g being zero at some points, making its log-pdf undefined, which is needed for MCMC sampling.

While there are general formulas to propagate probability probability densities through a non-invertible change of variables, they require the explicit use of the whole pre-image of a single point – in our case, we would have to know all λ_T with $\mathrm{PMP}(\lambda_T) = x_0$, which is not practically feasible. All in all, relaxing the assumption that PMP is invertible necessitates either a significant overhaul of the presented sampling procedure, a drastic decrease in theoretical soundness, or both.

The other difficulty is that at $t = 0$, we will get conflicting samples of $V(x_0)$ and $\lambda(x_0)$ when distinct locally optimal trajectories start at the same x_0. In principle, if we could evaluate the PMP function *for all* infinitely many boundary conditions λ_T, the rest of the problem would be simple – just select any desirable set of initial conditions x_0, and for each one, take the solution with the lowest $V(x_0)$ from those that start at x_0.

We conjecture that MCMC sampling from *some* suitable distribution over λ_T can be part of a realistic approximation of this "perfect" algorithm, maybe with some form of iterative re-sampling to encourage sampling solutions that could plausibly be the global optimum under a partially learned value function. Some interesting thoughts on this are also discussed after a talk [25] by a co-author of [18]. We hope to inspire future research that addresses the challenge of global optimality.

4.3.2 Implementation aspects

Our MCMC sampler is not very efficient, and we have to throw away much of the data because it is too strongly correlated. This could be improved by further optimising the parameters of the sampler, or by using a different sampling algorithm altogether. We think specifically that the choice of proposal distribution (10) was based on misguided intuition: Because we approximate a Langevin diffusion in x_0, we get proposals centred around the current sample when $p(x_0)$ is uniform. However, the distribution from which λ_T is sampled is generally not uniform (See Fig. 3). We think a better sampler could be achieved by approximating a Langevin diffusion over λ_T, biasing the proposals towards regions with larger density, while still using some pre-conditioning based on the Jacobian J.

Another reason that data generation is slow is the small ODE solver timestep. We think that this is mostly needed due to the nondifferentiability introduced by the input constraint, when it changes between being active and being inactive. Using an adaptive ODE solver could speed up the process without losing significant accuracy.

Currently, we evaluate the Jacobian determinant in (9) naively using automatic differentiation. However, theorem 1 from [26] presents an alternative way of accounting for the volume change due to a continuous-time transformation of a random variable. To properly apply it, though, we have to consider probability densities over the combined (x_T, λ_T) and (x_0, λ_0). This might lead to challenges as the densities over (x_T, λ_T) we need are usually singular, as we are either always setting $x_T = 0$ (as we do here, due to the terminal constraint), or λ_T is a deterministic function of x_T (in case of terminal value function). However, the full transformation $(x_T, \lambda_T) \mapsto (x_0, \lambda_0)$ is now obviously invertible (with the inverse given by solving the ODE in the other direction), which might aid in relaxing the restrictive Assumption 7.

Our loss function (11) penalises mostly the costate approximation error $\nabla_x V_{\mathrm{NN}}(x;\theta) - \lambda$. We suspect that for more challenging problems this might be suboptimal. We propose to use the input approximation error $u^\star(x, \nabla_x V_{\mathrm{NN}}(x;\theta)) - u^\star(x, \lambda)$ as the main loss signal, while keeping the present loss terms but lowering their importance. This loss most closely reflects our actual goals of approximating the input and is physically more intuitive than the costate error.

4.3.3 Uncertainty sampling

While our NN ensemble gives us uncertainty estimates, we do not use them at the moment. An improvement over our method could include some form of uncertainty sampling, where a second sampling step collects more data in regions of large model uncertainty to optimally inform the NN ensemble.

In our small examples, we see satisfactory performance when using a single dataset generated by MCMC sampling from a uniform distribution. However, when we go to higher dimensional problems, we cannot afford the luxury of covering the state space with samples as densely as we do here. In this case, uncertainty sampling might prove effective.

4.3.4 Data augmentation

A practical and easy improvement to the data generation process could be data augmentation via symmetries of the dynamics model. For example, for a physical system with a left-right symmetry, we could double the number of training points at essentially no cost, by adding a mirrored copy of the training dataset. A similar idea in the context of neural PDE solvers has been explored in [27].

4.3.5 Larger input spaces

In this work we only tested systems with a scalar input, which means that finding $u^\star$ according to (5) is straightforward. But many relevant control problems have more inputs. As long as H is convex in u, $u^\star$ is still easy to calculate using modern convex optimisers, some of which play very nicely with autodifferentiation tools [28]. However, the cost associated with solving small convex problems could still be prohibitive, as they have to be solved many times within the ODE solver. In that case, tools for explicit convex optimisation such as MPT [29] or PPOPT [30] might reduce the total computational effort.

4.3.6 State constraints

We only treat unconstrained problems, whereas practical problems often demand the satisfaction of state constraints. They could be addressed in two ways: Penalty functions or constrained versions of the PMP.

Penalty functions are conceptually straightforward, but increase the sensitivity of the locally optimal trajectory w.r.t. λ_T, making the sampling distribution $g(\lambda_T)$ less well-behaved[2]. We expect that avoiding these difficulties while still approximating the constrained problem well enough will be a hard balancing act.

State-constrained versions of the PMP are somewhat obscure and mostly involve an additional lagrange multiplier function for each constraint (see for example [31, 32, 33]). This increases the dimensionality of the problem, and the sampling approach presented here would need to be significantly generalised to handle state constraints in this way.

5 Conclusion

This work presents an approach for solving deterministic, finite-horizon, unconstrained optimal control problems over a large region of the state space. We avoid explicit optimisation over control laws or input sequences and instead rely on the PMP and a MCMC sampling algorithm to directly find many optimal trajectories. For the small example problems we look at, the approach works as desired.

In contrast to previous similar works [18, 19], we streamline the data generation process by not attempting to solve two-point BVPs and instead letting one single tool handle it: MCMC sampling. We believe this solution is a better fit for the task at hand, as we generate many optimal trajectories without spending much effort to reach any specific initial condition. This reduces the amount of guesswork involved, and makes data generation easier to analyse and debug.

In contrast to MPC approaches, we find an approximation of the optimal controller that is available with a small, deterministic computational effort. By moving the effort offline, we pave the way towards using higher-fidelity models and closer approximations of the original control problem. From this angle, our method can be seen as a nonlinear and continuous-time, but approximate pendant of explicit MPC. In contrast to previous methods that approximate an MPC control law, we make the data generation process simpler and more efficient by avoiding explicit optimisation over inputs, thus enabling the generation of a larger training dataset per computational effort.

In comparison to RL approaches, we solve a different, much narrower class of problems. This allows us to make full use of the fact that a dynamics model is available to do the necessary simulations in backward time, where optimal trajectories are found very efficiently.

[2]Penalty functions generally involve high values of the Hessian $\nabla_x^2 l(x,u)$ in regions near the constraint boundary. From the sensitivity equations ([21], 2.3) we see that the sensitivity of an ODE solution w.r.t. initial conditions grows along its solution in proportion to the Jacobian of the RHS w.r.t. the solution. The RHS in (7) contains the term $dl(x,u)/dx$ as a part of $-\partial H/\partial x$, which when differentiated again w.r.t. x gives the Hessian mentioned above. The resulting high sensitivity will make the density $g(\lambda_T)$ very large in regions where the corresponding trajectories travel along a constraint boundary, causing difficulties with MCMC sampling and maybe even floating point arithmetic.

These differences could – in certain situations – make our methods preferrable over the more popular approaches. However, challenges remain to be addressed until our methods can be applied to larger real-world problems. Whether our approach presents a practical alternative to other methods of approximate optimal control remains to be seen.

A Parameters

	PMP solver parameters	
	ODE solver	`diffrax.Tsit5`
Δt	Time step	1/64
	MCMC sampler parameters	
τ	Discretisation time	1/64
	Chain length	65536
	Number of parallel chains	4
	Subsampling rate	16
V_{max}	Value level	16
Σ_x	Expected state covariance	$\text{diag}([4,8])^2$
	NN ensemble parameters	
N_{ens}	Ensemble size	16
	Hidden layer sizes	(64, 64, 64)
	Activation function	softplus
	Optimiser	ADAM
	Minibatch size	128
	Training epochs	10
	Initial learning rate	0.05
	Final learning rate	0.005
	Learning rate schedule	Exp. decay
μ	Value gradient penalty	50
	Closed-loop simulation parameters	
	ODE solver	`diffrax.Tsit5`
	Time step	1/16
T_{sim}	Simulation time	16
	Number of simulations	32

Table 1: Algorithm parameters used for the reported results.

References

[1] Michael Hertneck et al. "Learning an Approximate Model Predictive Controller With Guarantees". In: *IEEE Control Systems Letters* 2 (2018), pp. 543–548.

[2] Samuel Ainsworth et al. "Faster Policy Learning with Continuous-Time Gradients". In: *Conference on Learning for Dynamics & Control.* 2020.

[3] Jae Young Lee and Richard S. Sutton. "Policy iterations for reinforcement learning problems in continuous time and space - Fundamental theory and methods". In: *Autom.* 126 (2020), p. 109421.

[4] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015).

[5] Kenji Doya. "Reinforcement Learning in Continuous Time and Space". In: *Neural Computation* 12 (2000), pp. 219–245.

[6] Rémi Munos. "A Study of Reinforcement Learning in the Continuous Case by the Means of Viscosity Solutions". In: *Machine Learning* 40 (2000), pp. 265–299.

[7] 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. DOI: `https://doi.org/10.1016/j.jcp.2018.10.045`.

[8] Saviz Mowlavi and Saleh Nabi. "Optimal control of PDEs using physics-informed neural networks". In: *J. Comput. Phys.* 473 (2021), p. 111731.

[9] Somil Bansal and Claire J. Tomlin. "DeepReach: A Deep Learning Approach to High-Dimensional Reachability". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 1817–1824.

[10] Anastasia Borovykh et al. "Data-driven initialization of deep learning solvers for Hamilton-Jacobi-Bellman PDEs". In: *IFAC-PapersOnLine* (2022).

[11] Michael Lutter et al. "HJB Optimal Feedback Control with Deep Differential Value Functions and Action Constraints". In: *ArXiv* abs/1909.06153 (2019).

[12] Tingwei Meng et al. "SympOCnet: Solving optimal control problems with applications to high-dimensional multi-agent path planning problems". In: *ArXiv* abs/2201.05475 (2022).

[13] Daniel Liberzon. "Calculus of Variations and Optimal Control Theory: A Concise Introduction". In: Princeton University Press, 2012.

[14] Cheng-Kui Gu and Yize Chen. "Pontryagin Optimal Controller via Neural Networks". In: *ArXiv* abs/2212.14566 (2022).

[15] 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 (2015), pp. 289–315.

[16] Behzad Azmi, Dante Kalise, and Karl Kunisch. "Optimal Feedback Law Recovery by Gradient-Augmented Sparse Polynomial Regression". In: *J. Mach. Learn. Res.* 22 (2021), 48:1–48:32.

[17] Carlos Sánchez-Sánchez and Dario Izzo. “Real-time optimal control via Deep Neural Networks: study on landing problems”. In: *ArXiv* abs/1610.08668 (2016).

[18] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. “Adaptive Deep Learning for High Dimensional Hamilton-Jacobi-Bellman Equations”. In: *ArXiv* abs/1907.05317 (2019).

[19] Dario Izzo, Ekin Öztürk, and Marcus Märtens. “Interplanetary transfers via deep representations of the optimal policy and/or of the value function”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (2019).

[20] L. S. Pontryagin et al. “Mathematical Theory of Optimal Processes”. In: 1962.

[21] Peter Al Hokayem and Eduardo Gallestey. *Lecture Notes on Nonlinear Systems and Control.* 2020. URL: `https://people.ee.ethz.ch/~apnoco/Lectures2020/2020_NLSC_lecture_notes.pdf`.

[22] David Luengo et al. “A survey of Monte Carlo methods for parameter estimation”. In: *EURASIP Journal on Advances in Signal Processing* 2020 (2020), pp. 1–62.

[23] Stephen Boyd. *EE363: Linear Dynamical Systems.* 2008. URL: `https://stanford.edu/class/ee363/`.

[24] Patrick Kidger. “On Neural Differential Equations”. PhD thesis. University of Oxford, 2021.

[25] Wei Kang. *Data Development and Deep Learning for HJB Equations.* Video containing a talk about the academic paper ”Paper Title” followed by a discussion. YouTube. 2020. URL: `https://www.youtube.com/watch?v=ABY-Wo6w77I`.

[26] Tian Qi Chen et al. “Neural Ordinary Differential Equations”. In: *Neural Information Processing Systems.* 2018.

[27] Johannes Brandstetter, Max Welling, and Daniel E. Worrall. “Lie Point Symmetry Data Augmentation for Neural PDE Solvers”. In: *ArXiv* abs/2202.07643 (2022).

[28] Akshay Agrawal et al. “Differentiable Convex Optimization Layers”. In: *Neural Information Processing Systems.* 2019.

[29] M. Kvasnica, P. Grieder, and M. Baotić. *Multi-Parametric Toolbox (MPT).* 2004. URL: `http://control.ee.ethz.ch/~mpt/`.

[30] Dustin Kenefake and Efstratios N Pistikopoulos. “PPOPT-Multiparametric Solver for Explicit MPC”. In: *Computer Aided Chemical Engineering.* Vol. 51. Elsevier, 2022, pp. 1273–1278.

[31] Richard F. Hartl, Suresh P. Sethi, and Raymond G. Vickson. “A Survey of the Maximum Principles for Optimal Control Problems with State Constraints”. In: *OPER: Continuous (Topic)* (1995).

[32] Thijs van Keulen. “Indirect Solution for Optimal Control Problems with a Pure State Constraint”. In: *IFAC Proceedings Volumes* 47 (2014), pp. 2456–2461.

[33] Adam Korytowski and Maciej Szymkat. “Necessary Optimality Conditions for a Class of Control Problems with State Constraint”. In: *Games* 12 (2021), p. 9.