# Connecting stochastic optimal control and reinforcement learning

Cite as: J. Math. Phys. 65, 083512 (2024); doi: 10.1063/5.0140665
Submitted: 29 December 2022 • Accepted: 20 July 2024 •
Published Online: 19 August 2024

J. Quer[1] and Enric Ribera Borrell[1,2,a)]

AFFILIATIONS

[1] Institute of Mathematics, Freie Universität Berlin, 14195 Berlin, Germany
[2] Zuse Institute Berlin, 14195 Berlin, Germany

**Note:** Paper published as part of the Special Topic on Cascades of Scales: Applications and Mathematical Methodologies.
[a)] **Author to whom correspondence should be addressed:** ribera.borrell@fu-berlin.de

## ABSTRACT

In this paper the connection between stochastic optimal control and reinforcement learning is investigated. Our main motivation is to apply importance sampling to sampling rare events which can be reformulated as an optimal control problem. By using a parameterised approach the optimal control problem becomes a stochastic optimization problem which still raises some open questions regarding how to tackle the scalability to high-dimensional problems and how to deal with the intrinsic metastability of the system. To explore new methods we link the optimal control problem to reinforcement learning since both share the same underlying framework, namely a Markov Decision Process (MDP). For the optimal control problem we show how the MDP can be formulated. In addition we discuss how the stochastic optimal control problem can be interpreted in the framework of reinforcement learning. At the end of the article we present the application of two different reinforcement learning algorithms to the optimal control problem and a comparison of the advantages and disadvantages of the two algorithms.

## I. INTRODUCTION

Rare event sampling is an area of research with applications in many different fields, such as finance,[1] molecular dynamics[2] and many more. Very often the reason for the occurrence of rare events is that the dynamical system of interest exhibits metastable behavior. Metastability means that the underlying process remains in certain regions of the state space for a very long time and only rarely changes to another region. This change is particularly important for accurate sampling of rare events. The average waiting time for the occurrence of the rare event is orders of magnitude longer than the timescale of the process itself. This behavior is typically observed for dynamical systems following Langevin dynamics and moving in a potential with multiple minima. Here the metastable regions correspond to local minima of the potential. The minima are separated by barriers and the transitions between these regions are of interest. In molecular dynamics, for example, these quantities of interest correspond to the macroscopic properties of the molecules under consideration. Furthermore, it can be shown that the time to cross the barriers scales exponentially with the height of the barrier.[3] In terms of sampling it is observed that the variance of the Monte Carlo estimators associated with these rare transitions is often large. One idea to improve these estimators is the application of importance sampling but other methods such as splitting methods have been proposed. In this article we are going to focus on importance sampling. For a detailed discussion of splitting methods see Ref. 4 and the references therein.

One of the main challenges in importance sampling is to find a good bias so that the reweighted expectation has a low variance. The theory shows that the bias that would lead to a zero variance estimator is related to the quantity one wants to sample; see, e.g., Refs. 5 and 6. Therefore, many different variational methods have been proposed to find a good bias.[7] For importance sampling applications in stochastic differential equations it is well known that the optimal bias is actually given as the solution of a Hamilton-Jacobi-Bellman (HJB) equation, a nonlinear partial differential equation.[8] Since the HJB equation is the main equation of stochastic optimal control the importance sampling problem can be interpreted as stochastic optimal control. A stochastic optimization approach to approximate the bias using a parametric representation of the control has been proposed in Ref. 9. In the optimization approach the weights of the parametric representation are minimised to find the best approximation of the control. Methods for solving the related Hamilton-Jacobi-Bellman equation using deep learning based strategies in high dimensions have been developed; see, e.g., Refs. 10–15. The approximation of control functions with tensor trains has been presented in Ref. 16.

Although many methods have been proposed to approach the sampling problem from an optimal control point of view the stochastic optimization formulation offers the possibility to make a connection with reinforcement learning. Reinforcement learning (RL) is one of the three basic machine learning paradigms and has shown impressive results in high-dimensional applications such as Go and others.[17,18] The reinforcement learning literature is very rich and many interesting ideas such as model-free, data-driven methods and robust gradient estimation have been extensively explored. From a more abstract point of view optimal control and reinforcement learning are concerned with the development of methods for solving sequential decision problems[19] and their connection has been explored to some extent in Ref. 20. A general formulation can be given as follows: an intelligent agent should take different actions in an environment to maximise a so-called cumulative reward associated with a predefined goal. Applications of this formulation can be, for instance, a cleaning robot moving in a complex space, playing various games[21,22] or portfolio optimization.[8] The environment in which the agent moves is typically given in the form of a Markov decision process (MDP). Solution methods are often motivated by dynamic programming. The main difference between classical dynamic programming methods and reinforcement learning algorithms is that the latter do not require knowledge of an exact mathematical model of the MDP. In addition, reinforcement learning targets large MDPs where exact methods become infeasible. We make the link between the two fields in two ways. One way is to use the optimal control problem and formulate it as a Markov decision process which is the underlying theoretical framework of reinforcement learning. The other way is to formulate the reinforcement learning problem as a stochastic optimization problem. By comparing the resulting optimization problem with the optimization problem derived in Ref. 9 for the importance sampling stochastic optimal control problem, one can see that both agree. Although the link is known there are few papers that show how to explicitly establish this link and the different language in these fields makes it difficult to understand the overlap. For this reason we want to show this connection in more detail for our problem of interest namely the application of importance sampling. However, the underlying connection is much more general and can be used for other applications that use stochastic modeling. Moreover, making this connection has the advantage that ideas developed in one area can be transferred to the other.

The paper is structured as follows. In Sec. II we set the stage of rare event simulation, present the importance sampling problem and state its stochastic optimal control formulation. Section III is dedicated to the introduction of the reinforcement learning framework. In Secs. III A–III D we discuss Markov decision problems, which are the underlying theoretical framework of reinforcement learning. In Sec. III E we state the reinforcement learning optimization problem and in Sec. III F we recap the key ideas behind the main RL algorithms. Section IV is devoted to showing how optimal control and reinforcement learning are related. In Sec. IV A we show how the stochastic optimal control problem can be formulated as an MDP. In Sec. IV B we compare the two optimization formulations presented for both problems. In Sec. IV C we first discuss how a previously presented solution method in the framework of reinforcement learning can be understood as a model-based approach. We introduce the well-known model-free deterministic policy gradient (DPG) algorithms. In Sec. V we present an application of the presented methods to a small toy problem. The focus of this section is the approximation of the optimal control problem which we will discuss in more detail. We conclude the article with a summary of our results.

## II. IMPORTANCE SAMPLING SOC PROBLEM

The main goal of this paper is to show how stochastic optimal control (SOC) and reinforcement learning (RL) are related. We introduce a SOC problem related to an importance sampling diffusion problem. However, the importance sampling SOC formulation can be easily adapted to general SOC problems and is not restricted to the importance sampling application. First we motivate the importance sampling problem and present a rather formal formulation. Then we show the relationship to the corresponding Hamilton-Jacobi-Bellman (HJB) equation. Finally we show how this problem can be formulated as a stochastic optimization problem. This section will be formulated in continuous time to show the relation to the HJB equation. In the following sections we will return to the discrete-time problem for convenience as this is the viewpoint used in most of the reinforcement learning literature.

### A. The sampling problem

The problem we are going to consider in this paper is a particle moving in some potential landscape $V : \mathbb{R}^d \to \mathbb{R}$ on the bounded domain $\mathcal{D} \subset \mathbb{R}^d$. The movement of the particle follows the gradient of the potential plus a perturbation given by a Brownian motion noted as $w_t$ and scaled by a diffusion term. This type of movement can be described by a stochastic differential equation (SDE) known as the overdamped Langevin equation:

$$\mathrm{d}x_t = -\nabla V(x_t)\mathrm{d}t + \sigma(x_t)\mathrm{d}w_t, \quad x_0 = x, \tag{1}$$

where $x_t$ is the position of the particle at time $t$, $x \in \mathbb{R}^d$ is a deterministic starting position and the diffusion term is chosen to be $\sigma(x) = \sqrt{2\beta^{-1}}\,\mathrm{Id}$. Let us assume we are interested in computing the expectation

$$\Psi(x) := \mathbb{E}[I(x_{0:T}) \mid x_0 = x] \tag{2}$$

of path dependent characteristics which can be described as

$$I(x_{0:T}) := \exp\left(-g(x_T) - \int_0^T f(x_t)\mathrm{d}t\right), \tag{3}$$

where $T = \inf\{t > 0 \mid x_t \in \mathcal{T}\}$ is the first hitting time of a specific target set $\mathcal{T} \subset \mathcal{D}$ and $f, g : \mathbb{R}^d \to \mathbb{R}$ are some bounded and sufficiently smooth functions. In the literature $f$ and $g$ are often denoted as the *running cost* and the *terminal cost* respectively. For instance the quantity of interest which leads to estimating the moment generating function of $T$ can be achieved by setting $f = \lambda \in \mathbb{R}$ and $g = 0$, resulting in $\Psi(x) = \mathbb{E}[e^{\lambda T} \mid x_0 = x]$.

We usually consider a potential $V$ which has many local minima and which induces metastable dynamics in the model under consideration. This metastability leads to a high variance of the Monte Carlo estimator of the path dependent quantity of interest and thus it becomes unreliable. To improve the properties of the estimator we will apply an importance sampling strategy. The general idea of such a strategy is to sample a different dynamical system by introducing a bias and later correcting this effect in the expectation.

In the considered problem we are going to introduce a bias in the drift term of the SDE. This additional drift is known as a control in the SOC literature. By construction it does not influence the stochastic noise. The controlled dynamical system is given by

$$\mathrm{d}y_t = (-\nabla V(y_t) + \sigma(y_t)u(y_t))\mathrm{d}t + \sigma(y_t)\mathrm{d}w_t, \quad y_0 = x, \tag{4}$$

where $u : \mathbb{R}^d \to \mathbb{R}^d$ is the mentioned control, which belongs to the space of time-independent feedback controls $\mathcal{U}$ (see Ref. 23 for more technical assumptions on the control). The time-independence assumption on the feedback control comes from the fact that the quantity of interest (2) is time independent thus the overall problem we are trying to solve is stationary. The controlled dynamics can be related to the original dynamics by using the Girsanov formula, which is a change of measure in path space. Precisely, we have that the importance sampling quantity of interest is an unbiased estimator of (2)

$$\Psi(x) = \mathbb{E}[I(x_{0:T}) \mid x_0 = x] = \mathbb{E}[I(y_{0:T_u})\, m_{0:T_u} \mid y_0 = x], \tag{5}$$

where $m_{0:T_u}$ is an exponential Martingale given by

$$m_{0:T_u} := \exp\left(-\int_0^{T_u} u(y_t)\cdot \mathrm{d}w_t - \frac{1}{2}\int_0^{T_u} |u(y_t)|^2 \mathrm{d}t\right) \tag{6}$$

and $T_u$ is the first hitting time of $\mathcal{T}$ under the controlled process. Although the importance sampling relation (5) holds for any control $u \in \mathcal{U}$, the variance of the corresponding estimators significantly depends on the choice of $u$. Hence, one is tempted to aim for an optimal control $u^*$ which minimizes the variance of the importance sampling estimator over the space of controls

$$u^* = \arg\inf_{u\in\mathcal{U}} \{\mathrm{Var}\,(I(y_{0:T_u})\, m_{0:T_u} \mid y_0 = x)\}. \tag{7}$$

## B. Hamilton-Jacobi-Bellman equation

With this formulation at hand we can now search for the optimal control. It is well-known in the literature that the optimal control can be found by using the Feynman-Kac formula and the resulting partial differential equation (PDE) connection.[6,23] Via the Feynman-Kac formula $\Psi$ satisfies the following elliptic boundary value problem (BVP)

$$(\mathcal{L} - f)\Psi = 0, \quad \text{in } \mathcal{O}, \tag{8a}$$

$$\Psi = \exp(-g), \quad \text{on } \partial\mathcal{O}, \tag{8b}$$

on the domain $\mathcal{O} := \mathcal{D} \cap \mathcal{T}^c$ where $\mathcal{L}$ denotes the infinitesimal generator, which is given by

$$\mathcal{L} = \frac{1}{2}\sum_{i,j=1}^{d} (\sigma\sigma^\top)_{ij}(x)\frac{\partial^2}{\partial x_i \partial x_j} - \sum_{i=1}^{d}\left(\frac{\partial}{\partial x_i}V(x)\right)\frac{\partial}{\partial x_i}. \tag{9}$$

By using the Cole-Hopf transformation $\Phi = -\log\Psi$, one can derive the well-known Hamilton-Jacobi-Bellman equation

$$\mathcal{L}\Phi - \frac{1}{2}|\sigma^\top\nabla\Phi|^2 + f\Phi = 0, \quad \text{in } \mathcal{O}, \tag{10a}$$

$$\Phi = g, \quad \text{on } \partial\mathcal{O}. \tag{10b}$$

Moreover, it has been shown that the optimal control depends directly on the solution of the above PDE

$$u^* = -\sigma^\top \nabla \Phi = \sigma^\top \nabla \log \Psi \tag{11}$$

and that the corresponding importance sampling estimator achieves zero variance see e.g., Refs. 6 and 23.

A priori one can calculate the quantity that we originally wanted to estimate as a function of the initial position via e.g., a finite difference method. However, such a problem becomes not trivial to solve for high-dimensional settings due to the curse of dimensionality. Furthermore, we know that the problem we are trying to solve is hard because we are trying to find a solution for a nonlinear PDE.

## C. Stochastic optimization problem

To this end we are going to reformulate the problem as an optimization problem. Let us recall the value function $\Phi$ as a function of $\Psi$ which can be expressed in terms of the importance sampling estimator (5)

$$\Phi(x) = -\log\left(\mathbb{E}\left[\exp\left(-g(y_{T_u}) - \int_0^{T_u} f(y_t)\mathrm{d}t - \int_0^{T_u} u(y_t)\cdot \mathrm{d}w_t - \frac{1}{2}\int_0^{T_u} |u(y_t)|^2 \mathrm{d}t\right) \,\middle|\, y_0 = x\right]\right).$$

By using the Jensen's inequality and later the fact that an expectation of an Itô stochastic integral is zero we obtain the following upper bound for the value function

$$\Phi(x) \le \mathbb{E}\left[g(y_{T_u}) + \int_0^{T_u} f(y_t)\mathrm{d}t + \frac{1}{2}\int_0^{T_u} |u(y_t)|^2 \mathrm{d}t \,\middle|\, y_0 = x\right]. \tag{12}$$

In the literature it is shown with the help of some stochastic calculus that the equality is attained for the optimal control $u^*$.[6,23] We can understand the right-hand side of the Eq. (12) as a performance measure of the control applied to the dynamical system. The first two terms in the expectation are equal to the logarithm of the quantity of interest e.g., the stopping time in our example setting and the third term measures the force applied to the system. In the optimal control literature this expression is known as the *cost functional* conditional on the initial condition. On the one hand we can use the right-hand side of (12) as the objective function for the optimization problem since it is easier to handle numerically. On the other hand we see that there is still a connection to the optimal control problem. If we manage to find the optimal solution with the optimization procedure we have the solution to the HJB equation.

By minimizing this functional in terms of the control $u$ subject to the controlled dynamical system we have derived an optimization problem in order to find the optimal control. This optimal control problem is given by

$$u^* = \arg\inf_{u\in\mathcal{U}} J(u;x) \tag{13}$$

where the minimisation is taken again over the space of controls and $J(u;\, x)$ is the corresponding cost functional

$$J(u;x) := \mathbb{E}\left[g(y_{T_u}) + \int_0^{T_u} f(y_t)\mathrm{d}t + \frac{1}{2}\int_0^{T_u} |u(y_t)|^2 \mathrm{d}t \,\middle|\, y_0 = x\right] \tag{14a}$$

$$\text{s.t. } \mathrm{d}y_t = (-\nabla V(y_t) + \sigma(y_t)u(y_t))\mathrm{d}t + \sigma(y_t)\mathrm{d}w_t, \quad y_0 = x. \tag{14b}$$

Many different ideas have been proposed in the literature to solve this problem. One idea is to use a Galerkin projection of the control into a space of weighted finite ansatz functions and optimise over the weights using a gradient descent method or a cross-entropy method.[9,24] Another idea is to solve the deterministic control problem and use it to steer the dynamical system in the right direction.[25] A more detailed discussion of solution methods can be found in Ref. 6.

According to the theory using optimal control to sample the quantity of interest would result in a zero variance estimator. In principle one needs to sample a trajectory to find the quantity of interest. However, due to discretization and numerical issues this is not possible in the implementation. However, with a good approximation to the optimal control the sampling effort can be massively reduced and the estimator converges faster with a smaller relative error. Furthermore, it can be shown that the relative error scales exponentially with the approximation error.[26] This means that one should use the best possible approximation to find a good estimator of the quantity of interest. Because of this dependence, it is necessary to use methods that find a solution close to the optimum.

For mathematical completeness it is necessary to have a small remark on other possible types of time horizons. In this article $T_u$ is an a.s. finite stopping time with respect to the canonical filtration of the controlled process. In general we could consider

- a finite horizon time $T_u = T_{\text{end}}$, leading to a deterministic stopping time,
- a bounded stopping time $\widetilde{T}_u = \min(T_u, T_{\text{end}})$,
- or a general random stopping time $T_u \in \mathbb{R}^+ \cup \{+\infty\}$.

The first two types of stopping times are also a.s. finite and are of special interest since they guarantee the applicability of Girsanov's theorem for a control on a bounded domain. In this case the Novikov condition is satisfied. The reason is that with this assumption the boundedness of the control is guaranteed. However, one ends up with a different problem formulation. In these cases the corresponding BVPs are not elliptic but parabolic leading to time-dependent solutions and as a consequence the optimal control is also time-dependent.[23,27] The general case of random stopping time needs to be discussed in more detail and is beyond the scope of this article.[6]

## III. INTRODUCTION TO REINFORCEMENT LEARNING

Before showing the connection between RL and SOC for completeness we would like to give a brief overview of a typical model for a reinforcement learning problem. First, we look at the underlying theoretical framework namely Markov decision processes. Then, we will discuss key concepts of RL theory, such as types of policies, value and Q-value functions, and their recursive relations provided by the Bellman equations. Finally, we will introduce the reinforcement learning problem from an optimization point of view and discuss two different formulations of the RL problem.

As stated by Sutton and Barto: "Reinforcement learning is learning what to do - how to map situations to actions - in order to maximise a numerical reward signal."[21] So from this very first definition we can already see what it takes to define a reinforcement learning problem. To learn what "to do," we need someone to do something. This is usually called the *agent*. The agent experiences situations by interacting with an *environment*. This interaction is based on the *actions* the agent takes, the *reward signals* the agent receives from the environment and the *next state* the agent has moved to by following the controlled dynamics. Usually the interaction with the environment is considered to take place over some time interval (finite or infinite).

The goal of the agent is to reach a predefined goal that is part of the environment. The agent will reach the goal if it optimally chooses the sequence of actions so that the sum of the reward signals received along a trajectory is maximised. Given the dynamics of the environment and a chosen goal reinforcement learning assumes that there are reward functions that can lead the agent to success in the sense that the predefined goal is reached. The existence of a unique reward function that is certain to lead to the predefined goal is not given. Thus, the choice of the reward function can be flexible and depend on the task that the agent has to solve. Furthermore, it may have an influence on the learned action.

Environments with sparse reward functions are often difficult to handle, since the reward signal received is often negligible. For example this is often the case in games where the reward signal is 0 throughout the game and changes to −1 in the event of a loss or +1 in the event of a win. However, this is not the only possible reward function that makes the agent learn how to play. If a person plays a game of chess with a teacher who tells her how she played after every move the person will receive a much richer reward signal which will eventually lead to faster learning.

### A. Markov decision processes

The theoretical framework of all reinforcement learning problems is a Markov decision process. A typical RL problem has the following elements:

- the *state space* $\mathcal{S}$ is the set of states i.e. the set of all possible situations in the environment,
- the *action space* $\mathcal{A}$ is the set of actions the agent can choose at each state,
- the *set of decision epochs* $\mathbb{T} \subset \mathbb{R}^+$ is the set of time steps corresponding to the times where the agent acts. Let us assume the set of decision epochs is discrete. In this case, it can either be finite, i.e., $\mathbb{T} = \{0, \ldots, T\}$ with $T \in \mathbb{N}^+$, or infinite $\mathbb{T} = \mathbb{N}_0$.
- the *(state-action) transition probability function* $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ provides the probability of transitioning between states after having chosen a certain action. The transition probability function depends on the state where the agent is $s \in \mathcal{S}$, the action she chooses $a \in \mathcal{A}$ and the next state the agent moves into $s' \in \mathcal{S}$, i.e.,
$$p(s', s, a) = p(s'|s, a) := \mathbb{P}(s_{t+1} = s' \mid s_t = s,\ a_t = a). \tag{15}$$
The transition probability function given a state-action pair $p(\cdot|s,a)$ is a conditional probability mass function
$$\sum_{s' \in \mathcal{S}} p(s'|s,a) = 1. \tag{16}$$
If $\mathcal{S}$ is a continuous state space $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ represents the *(state-action) transition probability density*. Let $\Gamma \in \mathcal{B}(\mathcal{S})$ be a Borel set of the state space, then the probability of transitioning into $\Gamma$ conditional on being in the state $s \in \mathcal{S}$ and having chosen action $a \in \mathcal{A}$ is given by
$$\mathbb{P}(s_{t+1} \in \Gamma \mid s_t = s,\ a_t = a) = \int_\Gamma p(s'|s,a)\mathrm{d}s'.$$
For more details about continuous MDP we recommend to look at Ref. 28. For ease of notation, we will use the same symbol $p$ to denote the transition probability function and the transition probability density throughout the article.
- and the *reward function* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward signal the agent will receive after being in state $s \in \mathcal{S}$ and having taken action $a \in \mathcal{A}$, i.e.,
$$r_t := r(s_t, a_t). \tag{17}$$

Formally, the tuple $(\mathcal{S}, \mathcal{A}, \mathbb{T}, p, r)$ defines a Markov decision process. A more detailed introduction to MDPs can be found in Ref. 29.

### B. Reward and state-action transition probabilities

If an action $a_t$ is chosen in a state $s_t$ at time $t$, two things happen. First, the agent receives a reward signal $r_t = r(s_t, a_t)$. Second, the agent transitions to the next state $s_{t+1}$ according to the transition probability function $s_{t+1} \sim p(\cdot|s_t, a_t)$. In the literature one can find reward functions that depend on the state, the action and the next state. In this case the order changes. First the agent moves to the next state, and second, the agent receives the reward signal. We will consider the first case throughout this paper. Note that both the (state-action) transition probability function and the reward function depend only on the current state of the agent and the action chosen in that state. This is sufficient to describe the dynamics of the agent, since we assume that the agent is Markov. Recall that the Markov decision framework can be generalised to non-Markovian dynamics; see, e.g., Ref. 29.

The reward signal depends solely on the reward function. The reward function cannot be influenced by the agent and is considered to be part of the environment. We use the convention that the reward function is deterministic. However, for some environments the reward function may be described in a probabilistic way. In these cases it is useful to work with the so-called *dynamics function* $p_{\text{dyn}} : \mathcal{S} \times \mathbb{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ which gives us the probability that the agent is in the next state and has received a certain reward conditional on a state-action pair,[21] i.e.,

$$p_{\text{dyn}}(s', r, s, a) = p_{\text{dyn}}(s', r|s, a) := \mathbb{P}(s_{t+1} = s', r_t = r \mid s_t = s, a_t = a).$$

The *expected reward function* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ provides us the expected reward signal conditioned on being in a state and taking an action

$$\mathcal{R}(s, a) := \mathbb{E}[r_t \mid s_t = s,\ a_t = a].$$

If the dynamics of the environment is deterministic, the state transition probability function is replaced by the so-called *environment transition function*, denoted by $h : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. In this case the next state is given by $s_{t+1} = h(s_t, a_t)$. By introducing in the transition function a dependence on a random disturbance $\xi_t$ one can treat both stochastic and deterministic dynamical systems $s_{t+1} = h(s_t, a_t, \xi_t)$. To treat deterministic systems in this framework we just have to set $\xi$ to zero; see, e.g., Ref. 30.

### C. Policies

Policies are the most important part of reinforcement learning. A policy is a mapping that determines what action to take when the agent is in state $s_t$ at time $t$. This is why they are sometimes called the agent's brain. Policies can be either deterministic or probabilistic.

A deterministic policy is a function from the state space into the action space. Deterministic policies are usually denoted by $\mu$, and the action for state $s_t$ is given by $a_t = \mu(s_t)$. A stochastic policy is a conditional probability distribution over the action space. Stochastic policies are usually denoted by $\pi$, and the new action for state $s_t$ can be computed by sampling from this conditional probability distribution. The action for $s_t$ is given by $a_t \sim \pi(\cdot|s_t)$. The deterministic policies can be seen as special cases of stochastic policies where the probability distributions over the action space are degenerate.

In the following sections we will show how the goal of reinforcement learning can be expressed in terms of finding policies that maximise the reward signal received at each time step.

### D. Trajectories, return and value functions

Depending on the time horizon which is considered the reinforcement learning literature distinguishes between *infinite horizon problems* and *terminal problems*. We are only going to discuss terminal problems here since the SOC problem belongs to this class. For infinite time horizon problems a discounted factor has to be taken into account to make sure that the cumulative reward is finite. Details can be found in Ref. 31.

By interacting with the environment the agent generates a trajectory $\tau$ which is a sequence of states $(s_\cdot)$, actions $(a_\cdot)$, and rewards $(r_\cdot)$

$$\tau := s_0, a_0, r_0, s_1, a_1, r_1, \ldots s_k, a_k, r_k, \ldots. \tag{18}$$

The initial state $s_0 \in \mathcal{S}$ is either sampled from a start state distribution $s_0 \sim \rho_0$ or chosen to be constant $s_0 = s_{\text{init}}$, i.e., $s_0 \sim \rho_0 = \delta_{s_{\text{init}}}$. Recall that the state transitions only depend on the most recent state and action $s_{t+1} \sim p(\cdot|s_t, a_t)$.

The overall goal of the agent is to maximize the cumulative reward along a trajectory up to time $t$

$$G_t(\tau) := r_t + \cdots + r_T = \sum_{k=0}^{T-t} r_{t+k} \tag{19}$$

by choosing the actions or policy optimally.

In order to estimate how well the agent performs starting in a given state and following a certain policy we need to define a performance measure. For each policy this performance measure is called *value function* which is a function of the state only. It is defined as the expected return conditional on the agent starting at the state $s \in \mathcal{S}$

$$V^\pi(s) := \mathbb{E}[\,G_t(\tau) \mid s_t = s; \pi\,], \tag{20}$$

where the actions are chosen according to the policy $a_t \sim \pi(\cdot|s_t)$ for all $t \in \{0,\dots,T\}$. In optimal control theory the value function term is a synonym of the so-called *optimal cost-to-go* which refers to the optimal value of the cost functional (with respect to all possible controls) conditional on the initial condition. In RL the term value function is used for all policies. When the policy is optimal we use the term *optimal value function*. A crucial property of the value function is that it satisfies the following recursive relationship. For all $s \in \mathcal{S}$ and $t \in \{0,\dots,T-1\}$ we have

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left[\sum_{k=0}^{T-t} r_{t+k} \,|\, s_t = s; \pi\right] = \mathbb{E}[\, r_t + \cdots + r_T \,|\, s_t = s; \pi\,] \\ &= \mathbb{E}\left[ r_t + \mathbb{E}\left[\sum_{k=0}^{T-1-t} r_{t+1+k} \middle| s_{t+1} = s_{t+1}; \pi\right] \middle| s_t = s; \pi\right] = \mathbb{E}[\, r_t + V^\pi(s_{t+1}) \,|\, s_t = s; \pi\,]. \end{aligned}$$

In the reinforcement learning literature often stationary problems are considered; see, e.g., Ref. 21. However, from a theoretical view point is possible to extend this to time-dependent problems. Moreover, the value function gives us a partial ordering over policies. We say that a policy $\pi$ is better than or equal to another policy $\pi'$, $\pi \geq \pi'$, if $V^\pi(s) \geq V^{\pi'}(s)$ for all states $s \in \mathcal{S}$. If a policy is better than or equal to all other policies, this is the optimal policy. The optimal policy is denoted as $\pi^*$ and the optimal value function is defined by

$$V^*(s) = \max_\pi V^\pi(s), \quad \forall s. \tag{21}$$

For a small, finite MDP, policy iteration (PI) strategies offer in general convergence to the optimal policy.[21] They interlude value iteration where the value function is estimated by using the Bellman equation iteratively and policy improvement. A direct way to find the optimal actions is given by solving the Bellman equation. For any MDP results about the existence of an optimal policy can be found in Ref. 29.

Analogous to the motivation of the value function one might be interested in estimating how well the agent performs following a policy starting in a given state and taking a certain action. The performance measure for this is the so-called *Q-value function*. The Q-value function for a certain policy is a function of state and action and is given by

$$Q^\pi(s,a) := \mathbb{E}[G_t(\tau) \,|\, s_t = s, a_t = a; \pi]. \tag{22}$$

It provides us with the quality of the chosen action with respect to the given state. Since the Q-value function is an extension of the value function it also satisfies the Bellman expectation equation

$$Q^\pi(s,a) = \mathbb{E}\big[r_t + Q^\pi(s_{t+1}, a_{t+1}) \,|\, s_t = s, a_t = a; \pi\big] \tag{23}$$

and thus the recursive relationship. Last, let us define the *advantage function* with respect to a certain policy by

$$A^\pi(s,a) := Q^\pi(s,a) - V^\pi(s), \tag{24}$$

which tells us for each state-action pair the advantage of taking action $a$ in state $s$ with respect to the value function at $s$.

## E. RL as optimization problem

Solving the Bellman optimality equations is often not an easy task because the transition function has to be known beforehand or learned. Even if this is the case it becomes an unfeasible assignment for high-dimensional problems due to the curse of dimensionality. In reinforcement learning one way to circumvent this is to formulate the RL problem as an optimization problem. Very generally this optimization problem can be stated as

$$\pi^* = \arg\max_{\pi \in \Pi} \; J(\pi) \tag{25}$$

where $\Pi$ is the space of policies we consider and $J$ is the RL objective function

$$J(\pi) := \mathbb{E}[G_0(\tau) \,|\, \pi] \tag{26a}$$

$$\text{s.t. } s_0 \sim \rho_0, \; s_{t+1} \sim p(\cdot|s_t, a_t), \; a_t \sim \pi(\cdot|s_t), \tag{26b}$$

for all $t \in \{0,\dots,T-1\}$ where $a_t$ is the action chosen at the state $s_t$ following policy $\pi$ and $s_{t+1}$ is the next state of the agent given by the dynamics of the system. The expectation in (26a) denotes the expected return along all trajectories following the policy $\pi$

$$\mathbb{E}[G_0(\tau) \,|\, \pi] = \int_{\widetilde{\Omega}} \rho_{\text{traj}}(\tau \,|\, \pi) G_0(\tau) \mathrm{d}\tau,$$

**ALGORITHM 1.** Main RL Online model-free algorithm.

```
1: for $trajectory = 1, 2, \ldots$ do
2:     for $time\ step = 1, 2, \ldots, T$ do
3:         Evaluate the dynamical system with the current policy $\pi$ and calculate the reward.
4:     end for
5:     Optimize the policy.
6: end for
```

where $\widetilde{\Omega}$ denotes the space of all possible trajectories and $\rho_{\text{traj}}(\cdot|\pi)$ represents the probability distribution over the trajectories which follow the policy $\pi$

$$\rho_{\text{traj}}(\tau|\pi) = \rho_0(s_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t). \tag{27}$$

In general it is numerically infeasible to optimize over a function space or a space of probability distributions. Hence, one often considers a parametric representation of a policy and tries to optimize with respect to the chosen parameters.

## F. Brief summary of RL algorithms

Over the last few years many different algorithms have been developed to solve reinforcement learning problems. For a good but non-exhaustive review we refer to Ref. 32 and the references therein. Many of the algorithms share a general framework, which is summarized in Algorithm 1.

The main difference between the methods is how the policy optimization is computed which depends on the underlying problem. The proposed methods can be distinguished if they are model based or model-free and if the policy optimization is done via Q-learning, policy gradient or a combination of both methods.

Let us first discuss the difference between model-based and model-free approaches. An algorithm is said to be model-based if the transition function is explicitly known (tractable to evaluate) point-wise. In this case we can just take the transition function and sample $s'$ from $p(\cdot|s,a)$. The known transition function contains a lot of information about the underlying dynamics and is therefore very useful for possible solution methods. A method is called *model-free* when the transition function is not explicitly known. In this case the transition function cannot be used explicitly in the solution methods. The difference is related to the distinction between stochastic optimal control and reinforcement learning. In the case of stochastic optimal control the model is often known while reinforcement learning aims at a more general solution method that does not depend on the underlying dynamical system.

Let us briefly discuss the main underlying solution methods.

### *1. Q-learning*

One of the first methods which was proposed was Q-learning.[33] The main idea is to approximate the state-action value function and use greedy policy iteration until convergence to the optimal policy.[21] The Bellman optimality equation

$$Q^*(s,a) = \mathbb{E}\left[r_t + \max_{a'\in\mathcal{A}} Q^*(s_{t+1}, a') \,\middle|\, s_t = s, a_t = a, \pi^*\right]$$

provides a recursive formula for updating the Q-value function until convergence as long as some mild assumptions with respect to the learning rates and how often a state-action pair is visited are satisfied (see Refs. 21 and 33 for more details). If this is the case for any state $s \in \mathcal{S}$ the optimal policy is given by the action which maximizes the Q-value function

$$\pi^*(s) = \arg\max_{a\in\mathcal{A}} Q^*(s,a).$$

However, the Q-learning algorithm can only be applied to problems with finite action spaces since taking the maximum over a continuous action space is not a feasible task. It works well as long as the state space is small and discrete. For large or continuous state spaces deep Q-learning has been developed in Ref. 34 where the Q-value function is approximated by a parametric representation called the deep Q-value network (DQN). In addition Q-learning algorithms suffer from possible overestimation of the action values due to the fact that the maximum expected action value is approximated by the maximum action value which is biased. The idea of double Q-learning attempts to overcome this problem by storing two Q-value functions. The method has been introduced for both tabular and function approximation settings.[35,36]

An extension of the Q-learning idea for continuous action spaces is only possible by considering a separate policy parameterisation leading to an actor-critic setting which will be discussed in detail in Sec. IV C 2. We refer to Ref. 37 and the references therein for a comprehensive study of Q-learning in continuous time.

### 2. Policy gradient

The main idea is to approximate the policy by a parametric representation and then solve the RL optimization problem (25) with respect to the parameters. In this approach one would like to use a direct optimization method like gradient descent. But one big challenge is the gradient calculation of the expectation, which is a research field by its own; see, e.g., Ref. 1. One way to overcome this is to consider a parametrized stochastic policy. In this case the gradient calculation can be done explicitly and one can find a close formula. This was first presented in Ref. 38 and the so-called *policy gradient theorem* for stochastic policies was derived. The derived estimator is given by

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}\big[G_0(\tau)\nabla_\theta \log\rho_{\text{traj}}(\tau|\theta)\,\big|\,\pi_\theta\big]\\ &= \mathbb{E}\left[G_0(\tau)\left(\sum_{t=0}^{T-1}\nabla_\theta \log\,\pi_\theta(s_t,a_t)\right)\middle|\,\pi_\theta\right]\\ &= \mathbb{E}\left[\sum_{t=0}^{T-1}\left(\sum_{t'=0}^{T} r_{t'}\right)\nabla_\theta \log\,\pi_\theta(s_t,a_t)\,\middle|\,\pi_\theta\right],\end{aligned} \tag{28}$$

where one first uses the so-called log-derivative trick and then expands the expression for the gradient of the logarithm of the distribution of trajectories following the policy $\pi$. Notice that this gradient estimator does not depend on the transition probability density $p$ and hence it is considered to be model-free.

The methods presented here are the two main general ideas behind many variants of RL algorithms and were presented very early in the reinforcement learning community. Over the years, many drawbacks have been identified and extensions and combinations of solution methods have been proposed. Since policy evaluation is quite expensive, off-policy methods have been developed, i.e., trajectories simulated under a different policy are used for the current optimization step. This can be done, for example, by using an importance sampling approach; see, e.g., Ref. 39. Methods that only use policies that have been sampled with the current policy are called online. A combination of Q-learning and policy gradient has been proposed to overcome the high variance of a pure policy gradient.[40] These ideas have been further developed and methods such as TRPO[39] and PPO[41] have been proposed.

All of these algorithms have been developed for stochastic policies but methods for deterministic policies have also been derived. In the next section we will introduce the family of model-free deterministic policy gradient algorithms which provide a policy gradient without needing to know the model explicitly.

The selection presented here is far from complete but a more detailed discussion of RL is beyond the scope of this article. For a more detailed overview we refer to Refs. 21, 32, and 42. Most of the developed methods have a specific domain of application so one needs to carefully consider whether a method can be applied to a specific problem at hand.

## IV. THE SOC PROBLEM AS RL FORMULATION

In this section we show how the importance sampling SOC problem can be formulated as a reinforcement learning problem. First we show how to define an MDP for the stochastic control problem. This allows us to construct an RL environment based on the time-discretised stochastic optimal control problem. Then we compare the optimization approaches for both problems and argue that both formulations have a large overlap. After that we discuss how a previously presented method for the SOC problem can be categorised as a reinforcement learning algorithm. Finally we present a family of RL algorithms designed to deal with environments such as our stochastic optimal control problem.

### A. Importance sampling SOC problem as RL environment

To show how the stochastic control problem can be interpreted as a reinforcement learning environment we consider for simplicity its corresponding time-discretized formulation. For the time-discretized dynamics of (4) we use an Euler-Maruyama discretization of the SDE see e.g., Ref. 43

$$s_{t+1} = s_t + (-\nabla V(s_t) + \sigma u(s_t))\Delta t + \sigma\,\sqrt{\Delta t}\;\eta_{t+1}, \qquad s_0 = x \tag{29}$$

where $s_t$ represents the time evolution of the controlled dynamics for a time step $\Delta t$. The term $\sqrt{\Delta t}\;\eta_{t+1}$ is the so-called Brownian increment where $\eta_{t+1} \sim \mathcal{N}(0,1)$ is a normally distributed random variable. The time-discretized objective function is given by

$$J(u;x) := \mathbb{E}\left[g(s_{T_u}) + \sum_{t=0}^{T_u-1} f(s_t)\Delta t + \frac{1}{2}\sum_{t=0}^{T_u-1}|u(s_t)|^2\Delta t\,\middle|\, s_0 = x\right]. \tag{30}$$

The *state space* consists of all possible states $s \in \mathbb{R}^d$ in the domain and is therefore infinite and continuous, $\mathcal{S} = \mathbb{R}^d$. The *action space* is given by the codomain of the SOC admissible controls $u \in \mathcal{U}$ i.e., $\mathcal{A} = \mathbb{R}^d$. Recall that the set of decision epochs $\mathbb{T}$ is a discrete set. As for the uncontrolled process we assume that the stopping time for the controlled process $T_u$ is a.s. finite and hence for any trajectory, the set of decision epochs $\mathbb{T}$ is finite.

To get a better understanding of the policy under consideration let us first have a look at the control used in (29). The main idea to turn the SOC problem into an optimization problem is to use a Galerkin projection into a space of weighted approach functions. This was first proposed in the literature by Ref. 9. We will denote the approximation by $\mathrm{u}_\theta$ and it is given by

$$\mathrm{u}_\theta(x) = \sum_{m=0}^{M} \theta_m b_m(x) \tag{31}$$

where the $b_m(x)$ are some approach functions such as radial basis functions, polynomials and $\theta$ is the weight vector. It is also possible to approximate the control with nonlinear function approximations such as neural networks; see, e.g., Ref. 44. Regardless of the choice of approximation functions the resulting control is still deterministic which is usually called feedback form control in the optimal control literature. Due to the deterministic control the resulting policy is deterministic and the optimization will be over the weights of the ansatz function.

The time evolution for each control is given by the time-discretised dynamics (29) and this is sufficient to simulate trajectories. Moreover, for the considered SDE, the corresponding *transition probability density* for the deterministic policy can even be explicitly derived from (29) (see Ref. 44 for details)

$$p(s_{t+1}|s_t, a_t) = \left(\frac{\beta}{4\pi\Delta t}\right)^{d/2} \exp\left(-\frac{\beta\Delta t}{4}\left|\frac{s_{t+1}-s_t}{\Delta t} + \nabla V(s_t) - \sqrt{2\beta^{-1}}a_t\right|^2\right). \tag{32}$$

The *reward function* is defined such that the corresponding return along a trajectory equals the negative term inside the expectation of the time-discretized cost functional (30). The reward signal at time step $t$ reads

$$r_t = r(s_t, a_t) := \begin{cases} -f(s_t)\Delta t - \frac{1}{2}|a_t|^2\Delta t & \text{if} \quad s_t \notin \mathcal{T}, \\ -g(s_t) & \text{if} \quad s_t \in \mathcal{T}. \end{cases} \tag{33}$$

Notice that the reward signal is in general not sparse since the agent receives feedback at each time step but the choice of the running cost $f$ and the final cost $g$ can influence this statement. Moreover, the *return* along a trajectory $\tau$ looks like

$$G_0(\tau) = -g(s_T) - \sum_{t=0}^{T-1} f(s_t)\Delta t - \frac{1}{2}\sum_{t=0}^{T-1} |a_t|^2\Delta t. \tag{34}$$

To do this we have defined an MDP for the importance sampling SOC problem. So we change the viewpoint of the SOC problem to the RL viewpoint. Next let us have a brief look at the two optimization problems for reinforcement learning and stochastic optimal control.

## B. Comparison between the optimization approaches

Let us start by looking at the two optimization problems given in (25) and (13). Both problems are stochastic optimization problems. The RL optimization problem maximises the expected return and the importance sampling SOC problem minimises the time-discretized cost functional conditional on the initial position. The RL optimization problem is a bit more general as the forward time evolution is stated in a very general setting. The forward trajectories are determined by the probability transition density and the stochastic policy while for the stochastic optimal control problem the time evolution is explicitly given by a controlled SDE (29) for a chosen control. Furthermore, in the RL framework, technical conditions are rarely imposed on the policy. In contrast in the SOC optimization formulation it is known that the optimal control is deterministic and must satisfy some technical assumptions as shown above.

The time-discretized optimization problem for the SOC problem is given by

$$J(\pi_\theta) = \mathbb{E}\left[g(s_T) + \sum_{t=0}^{T-1} f(s_t)\Delta t + \frac{1}{2}\sum_{t=0}^{T-1}|a_t|^2\Delta t \,\middle|\, \pi_\theta\right] \to \min \tag{35a}$$

$$\text{s.t.} \quad s_{t+1} = s_t + (-\nabla V(s_t) + \sigma a_t)\Delta t + \sigma\sqrt{\Delta t}\,\eta_{t+1}, \quad s_0 = x, \quad a_t \sim \pi_\theta(\cdot|s_t), \tag{35b}$$

where $s_t$ is the time evolution of the controlled dynamics. Furthermore, we can turn the minimisation problem into a maximisation problem by multiplying the cost functional by $-1$ without making any further changes. Looking more closely at the reward signal for the SOC problem we see that the signal is a compromise between maximising the negative logarithm of the quantity of interest, which is given here by $-g(s_T) - \sum_{t=0}^{T-1} f(s_t)\Delta t$, and maximizing the negative of the control term, given by $-\frac{1}{2}\sum_{t=0}^{T-1}|a_t|^2\Delta t$. So the last part of the reward can be seen as a kind of regularisation which keeps the norm of the actions low.

In both cases the expectation is about trajectories. For the SOC case the probability density over the trajectories can be given in the same way as in (27). Note that in this particular case the reward along a trajectory depends on the chosen policy.

### C. Algorithms for SOC in a RL framework

The first method proposed to solve the importance sampling SOC problem[9] was a pure gradient descent but many other variants of this approach have been developed in the literature; see, e.g., Ref. 23 and the references therein. Nevertheless most methods are based on the idea of gradient descent. We will show a derivation of the gradient descent method in the RL framework and show that this can be interpreted as a deterministic model-based policy gradient method. For the deterministic policy setting methods in reinforcement learning have also been proposed namely the model-free DPG family of algorithms and its subsequent variants. We will introduce the main idea of these methods and in the next section we will present an application of both algorithms.

#### *1. Model-based deterministic policy gradient*

Let us again understand the expectation of the objective function (26a) taken over the distribution of trajectories and proceed with the gradient computation as we did in (28). First, notice that the probability distribution over the trajectories now reads

$$\rho_{\text{traj}}(\tau|\mu_\theta) = \prod_{t=0}^{T-1} p(s_{t+1}|s_t, \mu_\theta(s_t))$$

and the derivative of the logarithm of the probability distribution with respect to the parameters changes to

$$\begin{aligned}\nabla_\theta \log \rho_{\text{traj}}(\tau|\mu_\theta) &= \nabla_\theta \log\left(\prod_{t=0}^{T-1} p(s_{t+1}|s_t, \mu_\theta(s_t))\right)\\ &= \sum_{t=0}^{T-1} \nabla_\theta \log\left(p(s_{t+1}|s_t, \mu_\theta(s_t))\right)\\ &= \sum_{t=0}^{T-1} \nabla_a \log p(s_{t+1}|s_t, a)\Big|_{a=\mu_\theta(s_t)} \cdot \nabla_\theta \mu_\theta(s_t).\end{aligned}$$

Moreover, the return along a trajectory now depends directly on the policy parameter $\theta$

$$G_0(\tau;\theta) = \sum_{t=0}^{T} r_t = \sum_{t=0}^{T} r(s_t, \mu_\theta(s_t))$$

and therefore its derivative cannot be neglected

$$\nabla_\theta G_0(\tau;\theta) = \sum_{t=0}^{T} \nabla_\theta r(s_t, \mu_\theta(s_t)) = \sum_{t=0}^{T} \nabla_a r(s_t, a)\,|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s_t).$$

By putting everything together the deterministic policy gradient reads

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}\left[\nabla_\theta G_0(\tau;\theta) + G_0(\tau;\theta) \nabla_\theta \log \rho_{\text{traj}}(\tau|\mu_\theta) \,\middle|\, \mu_\theta\right] \tag{36}$$

$$\begin{aligned}&= \mathbb{E}\Bigg[\sum_{t=0}^{T} \nabla_a r(s_t, a)|_{a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s_t)\\ &\quad + G_0(\tau;\theta)\left(\sum_{t=0}^{T-1} \nabla_a \log p(s_{t+1}|s_t, a)\Big|_{a=\mu_\theta(s_t)} \cdot \nabla_\theta \mu_\theta(s_t)\right) \Bigg|\, \mu_\theta\Bigg].\end{aligned} \tag{37}$$

As mentioned by Ref. 45 this gradient estimator requires the knowledge of the (state-action) transition density and the reward function. Hence, in contrast with its analog for stochastic policies it is a model-based method. Moreover, notice that this gradient estimator has the same form as (28) with an extra term in the expectation corresponding to the gradient of the return. Thus an optimization method which uses this gradient estimator can be interpreted as a deterministic version of the well-known REINFORCE method. A different approach to derive the same gradient estimator for the continuous-time SOC problem can be found in Refs. 44 and 46.

For the importance sampling RL environment we can provide a close expression for this gradient estimator since the model is known

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}\left[-\Delta t \sum_{t=0}^{T-1} \mu_\theta(s_t) \cdot \nabla_\theta \mu_\theta(s_t) + G_0(\tau;\theta)\left(\sum_{t=0}^{T-1} \eta_{t+1} \cdot \nabla_\theta \mu_\theta(s_t)\right) \middle|\, \mu_\theta\right], \tag{38}$$

where the term $\nabla_a \log p(s_{t+1}|s_t, a)$ has been computed by using the Euler-Marujama time-discretized stochastic differential equation (29). Such an approach has already been successfully implemented for the mentioned environment in the context of stochastic optimal control optimization. In the original work[9] the deterministic policy i.e., the control is represented by a linear combination of Gaussian ansatz functions and later the approach is extended to a general deep representation of the control to avoid the curse of dimensionality.[44]

### 2. Model-free deterministic policy gradient

Due to the nature of the importance sampling SOC problem we are interested in exploring RL algorithms designed for continuous environments with optimal deterministic policies. The well-known deterministic policy gradient (DPG) algorithms presented in Ref. 22 attempt to find a deterministic policy and are suitable for environments with continuous actions. The authors extend the *policy gradient theorem*[31] to problems with deterministic policies and discounted infinite horizon. The gradient derived in this paper is given by

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s)\nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)}\right] \tag{39}$$

where $\rho^\mu$ is meant to be the so-called "(improper) discounted state distribution."[21,22,31] The *actor* representation $\mu_\theta$ approximates the deterministic policy and its parameters are updated by stochastic gradient ascent of (39). The *critic* representation $Q_\omega$ approximates the Q-value function $Q^{\mu_\theta}$ and can be estimated by using policy evaluation algorithms.

Furthermore, the authors show that this deterministic policy gradient is the limiting case of its stochastic analogue as the policy variance tends to zero. This is shown by using a stochastic policy $\pi_{\mu,\sigma}$, where $\mu$ is a parameter for the mean and $\sigma$ controls the variance of the underlying probability distribution. The variance parameter is then set to 0. Under certain assumptions, it can be shown that the following expression holds (see Ref. 22)

$$\lim_{\sigma\to 0} \nabla_\theta J(\pi_{\mu_\theta,\sigma}) = \nabla_\theta J(\mu_\theta). \tag{40}$$

The underlying idea is that the probability distribution degenerates and the resulting degenerated probability distribution is only optimized in the mean parameter.

Later this actor-critic approach was developed further resulting in a deep deterministic policy gradient (DDPG)[47] version of the algorithm where the policy and the Q-value function are approximated by deep neural networks. In this work the main ideas of deep Q-learning[34] have been adapted to problems with continuous actions. The critic network is updated by minimising the following loss function known as the *(mean squared) Bellman loss*

$$\mathcal{L}_{BE}(Q_\omega^\mu) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_\omega(s,a) - \left(r(s,a) + (1-d)Q_\omega\left(s',\mu(s')\right)\right)\right)^2\right] \tag{41}$$

where $\mathcal{D}$ is a given data set of *trajectory transitions* and $d$ tells us if the next state $s'$ is a terminal state i.e., $d = \mathbb{1}_{\mathcal{T}}(s')$. The Bellman loss tells us how close the approximation $Q_\omega$ is to satisfy the Bellman equation in the given data set of transitions.

Further ideas behind the success of DQN algorithms[34] such as off-policy training with samples from a *replay buffer* or the use of separate *target networks* have been implemented to provide more stable and robust learning. The replay buffer consists of a finite set of trajectory transitions $(s_t, a_t, r_t, s_{t+1}, d)$ that are stored online or offline. When the replay buffer is full the oldest tuples are discarded. The transition records required in (41) are randomly sampled from the replay buffer. The motivation for using target networks is to reduce the correlations between the action values $Q_\omega(s,a)$ and the corresponding targets $r + (1-d)Q_\omega(s', \mu(s))$. A separate network is used for the targets, and its weights are updated slowly according to the original network. By forcing the target networks to update slowly, the stability of the algorithm is improved.

Finally, the work of Ref. 48 addresses how to deal with a possible overestimation of the value function for continuous action space problems with ideas from double q-learning. They introduce the idea of clipped actions as a regularisation technique for deep value learning. The idea behind this is that similar actions should have similar action values. The algorithm developed is the latest developed algorithm in the DPG family and is called twin delayed deep deterministic policy gradient (TD3).

In general, the DPG family of algorithms combines the two ideas of policy optimization and Q-learning methods. On the one hand it is a type of policy gradient algorithm because it uses a parametric representation of the policy and updates its parameters by a gradient ascent method. On the other hand the required gradient depends on the Q-value function that needs to be approximated. Last let us conclude by highlighting that the resulting DPG algorithms are model-free, i.e., they do not require knowing the model transition density (see Refs. 22 and 47 for further details).

## V. EXAMPLES: ONE-DIMENSIONAL DOUBLE WELL POTENTIAL

In this section we compare the use of two different algorithms to solve the RL optimization problem for the importance sampling environment introduced in Sec. IV. We consider the two main possible approaches for environments with continuous states and actions where the desired optimal policy is deterministic. We restrict ourselves to deterministic policy methods because we know from the PDE connection of the SOC problem that the optimal solution is deterministic. The first is an online model-based policy gradient method and the second is an offline model-free actor-critic method.

Throughout, we will consider the paradigmatic one-dimensional double well potential $V_\alpha : \mathcal{D}_\mathcal{S} \subset \mathcal{S} \to \mathbb{R}$ given by

$$V_\alpha(s) = \alpha(s^2-1)^2,$$

where the parameter $\alpha$ is responsible for the height of the barrier and $\mathcal{D}_\mathcal{S} = [-2,2]$ is the domain of the state space. We assume that the dynamics of the environment is governed by the transition probability density (32) with $\sigma(x) = \sqrt{2\beta^{-1}}$ and time step $\Delta t = 0.005$. The parameter $\alpha$ and the inverse temperature $\beta$ encode the metastability of the system. The height of the barrier is set to $\alpha = 1.0$ so that the metastability

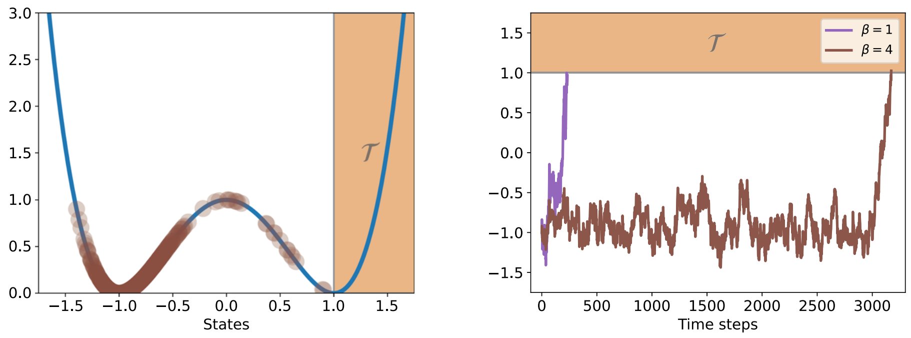

**FIG. 1.** Trajectories following the null policy for the two settings of study. The actions chosen along the trajectories are null. The trajectories are sampled starting at $s_{\text{init}} = -1$ until they arrive into the target set $\mathcal{T} = [1, \infty)$. Left panel: snapshots of the metastable trajectory $\beta = 4$. Right panel: trajectory positions as a function of the time steps.

is mainly influenced by the choice of $\beta$. To evaluate the performance of both algorithms we set a non-metastable setting $\beta = 1$ and a more metastable one $\beta = 4$. The initial value of the trajectories is set to $s_{\text{init}} = -1$ and the target set is chosen to be $\mathcal{T} = [1, \infty)$ for all experiments (Fig. 1). We set $f = 1$ and $g = 0$ so that the quantity of interest (3) is reduced to $I(x_T) = \exp(-T)$.

For the chosen environment the corresponding HJB Eq. (10) is one-dimensional and therefore we can compute reference solutions for the optimal control by a finite difference method. We call this reference control the HJB policy. The construction of our environment is motivated by finding the optimal policy with the intention of minimising the variance of the original importance sampling estimator (5). However, we do not focus on the performance of such variance reduction as it is sufficient to measure how close the approximated policy is to the optimal solution. To do this we track a $L^2$-type error of our deterministic policy approximations along the trajectories. Specifically, we define the empirical $L^2$ error along an ensemble of sampled trajectories with an arbitrary deterministic policy $\mu$ by

$$L^2(\mu) := \frac{1}{K_{\text{test}}} \sum_{k=1}^{K_{\text{test}}} \left( \sum_{t=0}^{T^{(k)}} |\mu - \mu_{\text{HJB}}|^2 (s_t^{(k)}) \Delta t \right), \tag{42}$$

where $K_{\text{test}}$ is the test batch size and the superscript $(k)$ denotes the index of each trajectory. This quantity tells us how close the policy $\mu$ is to the reference policy $\mu_{\text{HJB}}$ along the trajectories of the ensemble. For all experiments, the test batch size is chosen to be $K_{\text{test}} = 10^3$.

To approximate the optimal policy and the Q-value function we use *feed-forward neural networks*. These are essentially compositions of affine-linear maps and nonlinear activation functions which show remarkable approximation properties even in high dimensions. Let $d_{\text{in}}, d_{\text{out}} \in \mathbb{N}^+$ be the input and output dimensions of the feed-forward network $\varphi_\theta : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ which is defined recursively as follows

$$\varphi_\theta(x) = A_L \rho(A_{L-1}\rho(\cdots \rho(A_1 x + b_1)\cdots) + b_{L-1}) + b_L, \tag{43}$$

where $L$ is the number of layers $d_0 = d_{\text{in}}, d_L = d_{\text{out}}, A_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b_l \in \mathbb{R}^{d_l}, 1 \le l \le L$ are the weights and the bias for each layer and $\rho : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function applied componentwise. The collection of matrices $A_l$ and vectors $b_l$ contains the learnable parameters $\theta \in \mathbb{R}^p$. For all experiments we consider the policy representation $\mu_\theta = \varphi_\theta$ where $d_{\text{in}} = d_{\text{out}} = 1$ and the Q-value representation $Q_\omega = \varphi_\omega$ where $d_{\text{in}} = 2$ and $d_{\text{out}} = 1$ with both $L = 3$ layers, dimension of the hidden layers $d_1 = d_2 = 32$, and the activation function $\rho(x) = \tanh(x)$. To ensure that the initial output of the networks is close to zero the final layer weights and biases are initialised by sampling from the following uniform distributions $\mathcal{U}(-10^{-2}, 10^{-2})$, $\mathcal{U}(-10^{-3}, 10^{-3})$ respectively.

We repeat all our experiments several times with different random seeds to ensure generalisability. Each experiment requires only one CPU core, and the maximum value of allocated memory, is set to 1 GB unless otherwise stated.

## A. Model-based deterministic REINFORCE

First we present the results of the deterministic model-based version of the REINFORCE algorithm for the one-dimensional environment described above. We consider an online based implementation where the batch of trajectories is not reused after each gradient step. We summarise this method in Algorithm 2.

For both metastable and non-metastable settings we consider two different batch sizes $K = \{1, 10^3\}$ and use the Adam gradient based optimization algorithm[49] with corresponding learning rates $\lambda = \{5 \times 10^{-5}, 5 \times 10^{-4}\}$ respectively. The stopping criteria is set to be a fixed number of gradient steps $N = 10^4$ and the approximated policy is tested every 100 gradient updates.

Figures 2 and 3 show the $L^2$ empirical error as a function of the gradient updates and the policy approximation at different gradient steps for the two different problem settings. In the non-metastable case we can see for the experiment with batch size $K = 10^3$ that the policy

**ALGORITHM 2.** Model-based deterministic policy REINFORCE.

```
1: Initialize deterministic policy $\mu_\theta$.
2: Choose a batch size $K$, a gradient based optimization algorithm, a corresponding learning rate
   $\lambda > 0$, a time step size $\Delta t$ and a stopping criterion.
3: repeat
4:     Simulate $K$ trajectories by running the policy in the environment's dynamics.
5:     Estimate the policy gradient $\nabla_\theta J(\mu_\theta)$ by
           $\frac{1}{K}\sum_{k=1}^{K}\sum_{t=0}^{T^{(k)}-1}\left(-\Delta t\ \mu_\theta(s_t^{(k)})\cdot\nabla_\theta\mu_\theta(s_t^{(k)}) + G_0(\tau;\theta)\eta_{t+1}\cdot\nabla_\theta\mu_\theta(s_t^{(k)})\right).$
6:     Update the parameters $\theta$ based on the optimization algorithm.
7: until stopping criterion is fulfilled.
```

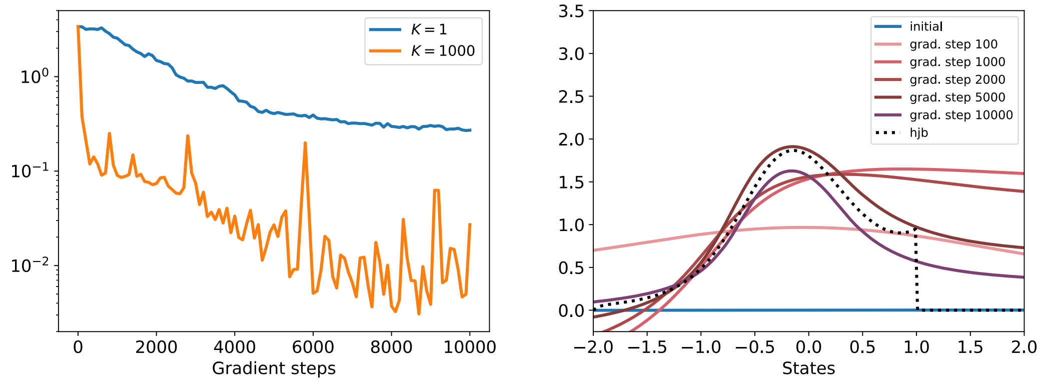

**FIG. 2.** Left panel: estimation of $L^2(\mu_\theta)$ at each gradient step for the non-metastable setting $\beta = 1$. Right panel: approximated policy for different gradient updates for the batch of trajectories case ($K = 10^3$).

approximation agrees well with the reference control already after ~5000 gradient steps. On the other hand with only one trajectory one has to rely on a lower learning rate and therefore learning is much slower. In the more metastable scenario we can see that the policy approximation for $K = 10^3$ is not as close to the HJB policy as in the less metastable case. One can see that after the same number of gradient steps the $L^2$ error differs by more than an order of magnitude. Moreover, for the non-metastable setting with $K = 10^3$ we need to increase the maximum value of allocated memory to 8 GB. For the metastable setting we end up allocating 4 GB for the one trajectory case $K = 1$ and around 100 GB for the batch case $K = 10^3$.

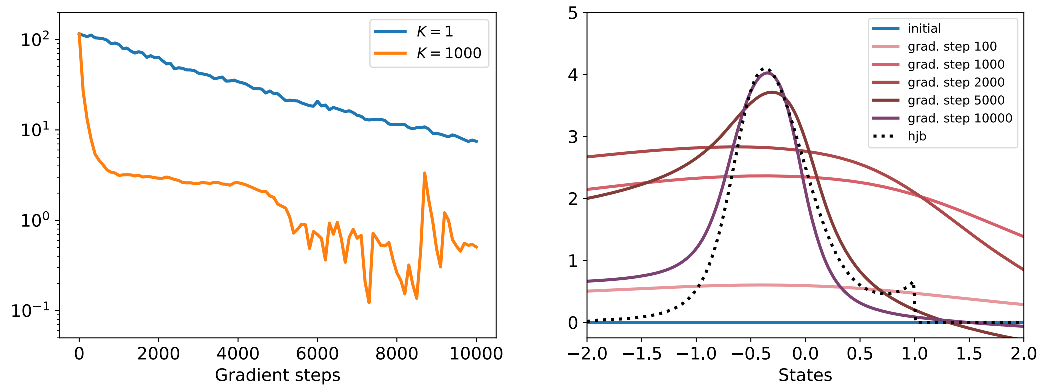

**FIG. 3.** Left panel: estimation of $L^2(\mu_\theta)$ as a function of the gradient steps for the metastable setting $\beta = 4$. Right panel: approximated policy for different gradient updates.

## B. Model-free deterministic policy gradient

Next we present the application of a model-free DPG method. In particular we will implement the TD3 variant of the DDPG method introduced at the end of Sec. IV C 2. First we consider a replay buffer and use separate target networks with slow updating so that the learning of the Q-value function is stabilised. We consider two critic networks to avoid overestimation of the value function. The replay buffer helps to reduce the amount of data that needs to be generated for each gradient estimation step. It uses caching of trajectories and random sampling of these cached trajectories for gradient estimation. The method is therefore considered an offline algorithm since it does not use the current control to estimate the gradient of the actor and critic networks. We summarise this method in Algorithm 3.

We consider a replay buffer that can allocate a maximum of $10^6$ trajectory transitions and we let the training start after $10^3$ time steps. Before the training phase starts the actions are chosen completely randomly and later they follow the actor policy with some exploration noise. For the non-metastable case, the standard deviation of the exploration is chosen to be $\sigma_{\text{expl}} = 1.0$ and the domain of the action space is set to $\mathcal{D}_{\mathcal{A}} = [a_{\text{low}}, a_{\text{high}}] = [-5, 5]$. For the metastable case we choose $\sigma_{\text{expl}} = 2.0$ and $\mathcal{D}_{\mathcal{A}} = [-10, 10]$. The clipping function is formally defined as $\text{clip}(\cdot, c, d) := \max(\min(\cdot, d), c)$. The maximum number of time steps per trajectory is set to $10^3$. We consider a batch size of $K = 10^3$ and

**ALGORITHM 3.** Model-free deterministic Policy Gradient (TD3).

```
1: Initialize actor network $\mu_\theta$ and critic networks $Q_{\omega_1}$ and $Q_{\omega_2}$.
2: Initialize corresponding target networks: $\theta' \leftarrow \theta$, $\omega'_1 \leftarrow \omega_1$, $\omega'_2 \leftarrow \omega_2$ and choose $\rho_{\text{p}} \in (0,1)$.
3: Initialize replay buffer $\mathcal{R}$.
4: Choose a batch size $K$, a gradient based optimization algorithm and a corresponding learning
   rate $\lambda_{\text{actor}}, \lambda_{\text{critic}} > 0$ for both optimization procedures, a time step size $\Delta t$ and a stopping criterion.
5: Choose standard deviation exploration noise $\sigma_{\text{expl}}$ and lower and upper action bounds $a_{\text{low}}, a_{\text{high}}$.
6: repeat
7:     Select clipped action and step the environment dynamics forward.
           $$a = \text{clip}(\mu_\theta(s) + \epsilon, a_{\text{low}}, a_{\text{high}}), \quad \epsilon \sim \mathcal{N}(0, \sigma_{\text{expl}}).$$
8:     Observe next state $s'$, reward $r$, and done signal $d$ and store the tuple $(s, a, r, s', d)$ in the replay buffer.
9:     if $s'$ is terminal then
10:        Reset trajectory.
11:    end if
12:    for $j$ in range (update frequency) do
13:        Sample batch $\mathcal{B} = \{(s^{(k)}, a^{(k)}, r^{(k)}, s'^{(k)}, d^{(k)})\}_{k=1}^{K}$ from replay buffer.
14:        Compute targets (Clipped Double Q-learning and policy smoothing).
           $$y(r, s', d) = r + (1-d)\min_{i=1,2}\{Q_{\omega'_i}(s', \tilde{a})\}, \quad \tilde{a} = \text{clip}(\mu_{\theta'}(s) + \epsilon, a_{\text{low}}, a_{\text{high}}), \quad \epsilon \sim \mathcal{N}(0, \sigma_{\text{target}}).$$
15:        Estimate critic gradient $\nabla_\omega L(Q_\omega^{\mu_\theta})$ by
           $$\nabla_{\omega_i}\left(\frac{1}{K}\sum_{k=1}^{K}\left(Q_{\omega_i}(s^{(k)}, a^{(k)}) - y(r^{(k)}, s'^{(k)}, d^{(k)})\right)^2\right), \quad \text{for} \quad i = 1, 2.$$
16:        Update the critic parameters $\omega_i$ based on the optimization algorithm.
17:        if $j$ mod policydelayfrequency $= 0$ then
18:            Estimate actor gradient $\nabla_\theta J(\mu_\theta)$ by
               $$\nabla_\theta\left(\frac{1}{K}\sum_{k=1}^{K} Q_{\omega_1}(s^{(k)}, \mu_\theta(s^{(k)}))\right).$$
19:            Update the actor parameters $\theta$ based on the optimization algorithm.
20:            Update target networks softly:
               $$\theta' \leftarrow \rho_{\text{p}}\theta' + (1-\rho_{\text{p}})\theta, \quad \omega'_i \leftarrow \rho_{\text{p}}\omega'_i + (1-\rho_{\text{p}})\omega_i, \quad \text{for} \quad i = 1, 2.$$
21:        end if
22:    end for
23: until stopping criterion is fulfilled.
```

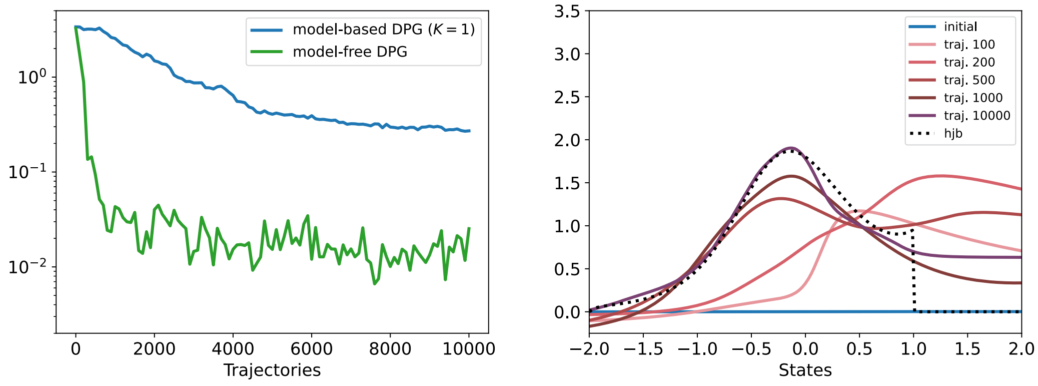

**FIG. 4.** Left panel: estimation of $L^2(\mu_\theta)$ as a function of the trajectories for the non-metastable setting $\beta = 1$. Right panel: approximated policy by the actor model after different trajectories.

use the Adam gradient based optimization algorithm with a learning rate of $\lambda_{\text{actor}} = \lambda_{\text{critic}} = 10^{-4}$ for both the actor and critic optimization procedures. The training takes place every ten time steps. During each training phase, the actor performs one gradient update for every two gradient updates performed by the critic, i.e. the critic performs ten gradient updates and the actor just 5. The standard deviation of the target noise is chosen to be $\sigma_{\text{target}} = 0.2$ and the Polyak averaging factor is chosen to be $\rho_{\text{p}} = 0.995$. The actor model is tested every 100 trajectories.

Figures 4 and 5 show the evolution of the $L^2$ error as a function of the sampled trajectories and the policy approximation at the end of different trajectories for the two chosen settings. The $L^2$ error is compared with the model-based approach with one trajectory $(K = 1)$.

For the non-metastable settings we can see that the policy approximation agrees well with the reference policy. We observe that the learning is quite fast. After around 2000 the $L^2$ error is already smaller than $10^{-1}$, but after this point the error does not decrease any further. Moreover, we observe that the model-free method learns faster than the model-based method in terms of generated data i.e., trajectories sampled. For the more metastable scenario we can see a similar pattern. Despite the metastability of the system the method learns quite fast and even manages to achieve a lower $L^2$ error than the model-based approach. However, it seems that the metastability does affect the stability of the method. Unfortunately this unstable behavior is observed for other choices of the hyperparameters of the algorithm, where the $L^2$ error can even blow up.

Finally in Fig. 6 we compare the $L^2$ error as a function of the computation time for the two considered methods. We see that the TD3 method learns a decent control much faster than the model-based approach, especially in the metastable setting. However, we observe that at a certain point the $L^2$-error stops decreasing. In contrast to that, for the model-based method learning is much slower but there is a steady decrease in the $L^2$-error.

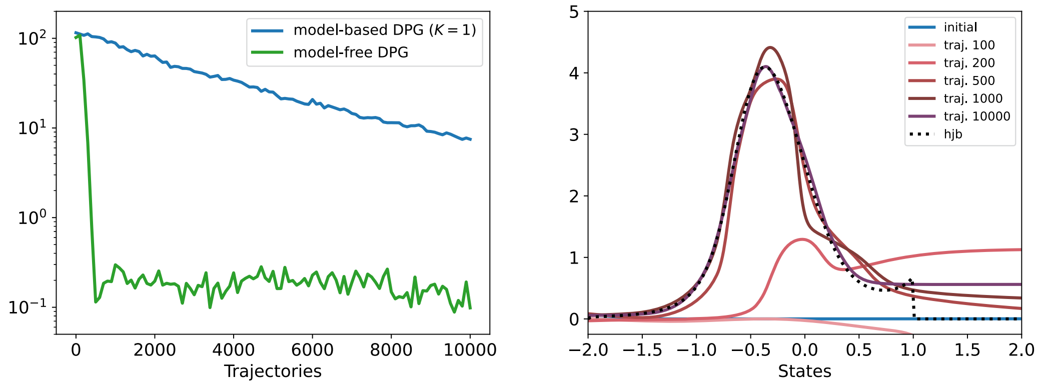

**FIG. 5.** Left panel: estimation of $L^2(\mu_\theta)$ as a function of the trajectories for the metastable setting $\beta = 4$. Right panel: approximated policy by the actor model after different trajectories.

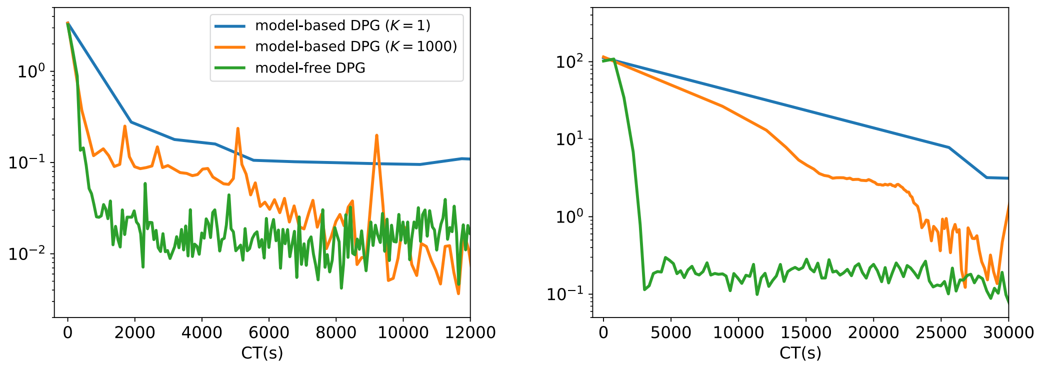

**FIG. 6.** Estimation of $L^2(\mu_\theta)$ as a function of the computation time for the two considered methods. Left panel: non-metastable setting $\beta = 1$. Right panel: metastable setting $\beta = 4$.

## C. Discussion

Let us now compare the two different methods used above. First we focus on the ingredients needed for each application. The model-based deterministic approach requires the model to be known. Without knowing the transition probability density this approach is not possible. For our importance sampling application with damped Langevin dynamics the transition probability density can be approximated after time discretization. However, we may be interested in general diffusion processes where this information is not given or cannot be trusted. Alternatively the model could be learned which is indeed a current area of research in model-based RL. On the other hand, the model-free alternative only requires knowledge of the reward function which is always the case for the importance sampling problem.

However, this model-free approach has its implications. The method relies on a good approximation of the Q-value function especially along the action axis because this determines the direction of the gradient that the actor will follow. Figure 7 shows the approximated Q-value and the advantage function in the space-action discretized grid for both settings. Note that the shape of the Q-value function has the following property: the difference between the Q-values for a given action along the state axis is orders of magnitude larger than the difference between the Q-values for a given state along the action axis. This difference may be the cause of the observed instabilities in the model-free

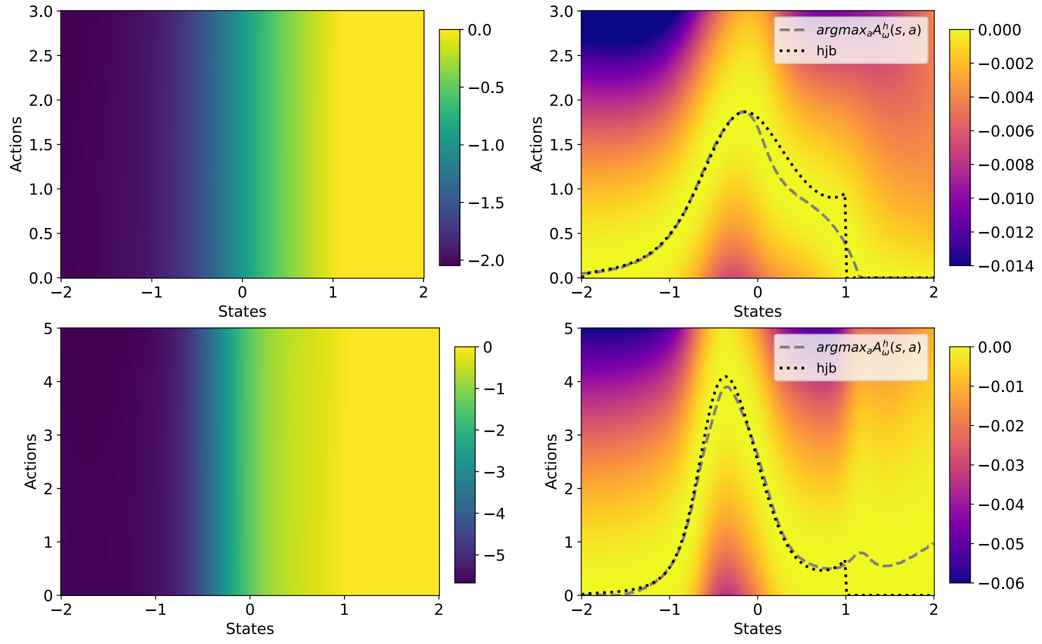

**FIG. 7.** Critic models after the last trajectory for both settings. Left panels: approximated Q-value function $Q_\omega$. Right panels: approximated optimal advantage function after action space discretization $A^h_\omega(s,a) = Q_\omega(s,a) - \max_{a \in \mathcal{A}_h} Q_\omega(s,a)$ and resulting greedy policy (gray dashed).

TD3 method. This problem is not specific to our importance sampling application and has been addressed in the field of advantage learning; see, e.g., Ref. 50. Advantage learning is an alternative approach to Q-learning where the advantage function is learned instead of the Q-value function. For future work it may therefore be interesting to exploit dueling network architecture approaches[51] where two separate estimators are maintained: one for the value function and one for the advantage function.

Regarding the performance of both approaches we have seen that the model-based method gets nearer to the optimal solution in the non-metastable settings. For high metastable dynamics this approach suffers from long running times and a high variance of the gradient estimator.[44] In our experimental analysis we observed a significant advantage of the TD3 algorithm over the deterministic reinforce algorithm in terms of learning a reasonable control faster. Our experiments suggest several reasons for this superiority. The TD3 algorithm performs a notably higher number of gradient steps per episode compared to the deterministic reinforce algorithm which relies on complete trajectory sampling before each gradient update, making bootstrapping impractical. This difference in gradient steps necessitates the deterministic gradient method to allocate considerably more memory for each update due to the extended length of trajectories. To ensure a fair comparison we conducted experiments for the case $K = 1$, allowing more gradient steps per data generated. The faster convergence observed in this scenario suggests that TD3 particularly benefits from the increased number of gradient steps especially in handling metastable problems. Another critical aspect contributing to TD3's effectiveness is its reliance on accurate Q-value function approximation. When the Q-value function is well-approximated TD3's gradient updates effectively guide the algorithm toward the optimal policy without requiring complete trajectory sampling, unlike the deterministic gradient estimator which lacks this correction term. Furthermore, this advantage enables TD3 to pursue off-policy learning enhancing its overall efficiency and adaptability. Moreover, TD3's third advantage lies in its integration of exploration mechanisms a feature absent in traditional gradient-based methods. By actively exploring the environment, TD3 efficiently uncovers novel and potentially rewarding state-action trajectories resulting in more informed policy discovery.

## VI. SUMMARY AND CONCLUSION

In this article we have shown that the stochastic optimization approach to importance sampling can be interpreted as a reinforcement learning problem. After presenting the importance sampling problem and a brief introduction to reinforcement learning we have shown how to formulate a MDP for the corresponding stochastic control problem. The MDP is the basic framework for reinforcement learning. By constructing the MDP we have established a first link. We then compared the optimization approaches given for both problems. The comparison has shown that the two optimization approaches are similar and that the optimization in the SOC case is a special case of the reinforcement learning formulation. In the SOC case the forward model of the controlled dynamical system is explicitly given while the reinforcement learning formulation is more general. A third connection has been shown by a detailed discussion of the algorithms developed for the SOC case. Here we have shown that a gradient-based method already proposed in the stochastic optimal control literature can be interpreted as the deterministic policy version of the well-known REINFORCE algorithm which turns out to be model-based. All in all, we have made three connections. We have introduced ideas from reinforcement learning that can be applied to problems seeking optimal deterministic policies namely DPG and its most popular variants DDPG and TD3. These algorithms are model-free policy gradient methods. They can be applied to the importance sampling SOC problem. We have presented the application of both algorithms used in a small dimensional setting and discussed their possible advantages and disadvantages. By applying TD3 to the SOC problem we have clearly shown that the importance sampling SOC problem can be interpreted as a reinforcement learning problem.

The main advantage of this is that ideas from reinforcement learning can now be applied to the stochastic optimal control approach to solving the importance sampling problem. For example reinforcement learning has already addressed the question of how to deal with the variance of the gradient estimator. The actor-critic method has been developed to solve this problem and it has been shown that the method achieves this goal. Especially for problems where the time evolution of the dynamical system is strongly influenced by metastable behavior this can be very helpful to reduce the sampling effort. Furthermore, the issue of efficient data usage has been discussed in the reinforcement learning community and various offline methods have been proposed to solve this problem. There are many other interesting ideas that have been addressed by the reinforcement learning community. Thus, this link can be used to efficiently design good and robust algorithms for high-dimensional settings of the importance sampling application.

We think that a combination of our model-based gradient estimator with an actor-critic design could be very interesting for the development of algorithms with fast convergence. Another research direction for us is the application of importance sampling to high-dimensional problems such as molecular dynamics. There is already related work exploring these ideas (see, e.g., Ref. 52). However, a stable application to real molecules is still lacking in the literature and would be a very helpful area of application. Another interesting line of research is the combination of model-free and model-based methods. As we have seen in the experiments with higher metastability learning with TD3 become unstable at a certain point. One could use TD3 to compute a good starting point so that the metastability is reduced and then switch to model-based optimization which seems to be much more stable. Similar ideas with pre-initialisation have been proposed in Ref. 44 where the optimization procedure is combined with an adapted version of the metadynamics algorithm.

## ACKNOWLEDGMENTS

The authors would like to thank the HPC Service of ZEDAT, Freie Universität Berlin, for computing time.[53] The research of J.Q. has been funded by the Einstein Foundation Berlin. The research of E.R.B. has been funded by Deutsche Forschungsgemeinschaft (DFG) through grant CRC 1114 "Scaling Cascades in Complex Systems," Project No. 235221301, Project A05 "Probing scales in equilibrated systems by optimal nonequilibrium forcing." Furthermore, we would like to thank the anonymous reviewer for his suggestions and improvements of the article and W. Quer for his linguistic improvements.

## AUTHOR DECLARATIONS

### Conflict of Interest

The authors have no conflicts to disclose.

### Author Contributions

**J. Quer**: Conceptualization (lead); Investigation (equal); Methodology (lead); Software (equal); Visualization (equal); Writing – original draft (lead); Writing – review & editing (equal). **Enric Ribera Borrell**: Conceptualization (supporting); Investigation (equal); Methodology (supporting); Software (equal); Visualization (equal); Writing – original draft (supporting); Writing – review & editing (equal).

## DATA AVAILABILITY

The code used for the numerical examples is available on GitHub at www.github.com/riberaborrell/rl-sde-is.

## REFERENCES

[1]E. Fournié, J.-M. Lasry, J. Lebuchoux, P. L. Lions, and N. Touzi, "Applications of Malliavin calculus to Monte Carlo methods in finance," Finance Stochastics **3**, 391–412 (1999).
[2]J. Quer, L. Donati, B. G. Keller, and M. Weber, "An automatic adaptive importance sampling algorithm for molecular dynamics in reaction coordinates," SIAM J. Sci. Comput. **40**, A653–A670 (2018).
[3]N. Berglund, "Kramers' law: Validity, derivations and generalisations," Markov Process. Relat. Fields **19**, 459–490 (2013).
[4]F. Cérou, A. Guyader, and M. Rousset, "Adaptive multilevel splitting: Historical perspective and recent results," Chaos **29**, 043108 (2019).
[5]T. Leliévre, M. Rousset, and G. Stoltz, *Free Energy Computations* (Imperical College Press, 2010); https://www.worldscientific.com/doi/pdf/10.1142/p579.
[6]T. Lelièvre and G. Stoltz, "Partial differential equations and stochastic methods in molecular dynamics," Acta Numer. **25**, 681–880 (2016).
[7]O. Valsson and M. Parrinello, "Variational approach to enhanced sampling and free energy calculations," Phys. Rev. Lett. **113**, 090601 (2014).
[8]W. Fleming and H. Soner, *Controlled Markov Processes and Viscosity Solutions*, *Applications of Mathematics* (U.S. Government Printing Office, 1993).
[9]C. Hartmann and C. Schütte, "Efficient rare event simulation by optimal nonequilibrium forcing," J. Stat. Mech. **2012**, P11004.
[10]E. Weinan, J. Han, and A. Jentzen, "Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations," Commun. Math. Stat. **5**, 349–380 (2017).
[11]N. Nüsken and L. Richter, "Solving high-dimensional Hamilton–Jacobi–Bellman PDEs using neural networks: Perspectives from the theory of controlled diffusions and measures on path space," Partial Differ. Equ. Appl. **2**, 48 (2021).
[12]E. Weinan, J. Han, and A. Jentzen, "Algorithms for solving high dimensional PDEs: From nonlinear Monte Carlo to machine learning," Nonlinearity **35**, 278 (2021).
[13]M. Zhou, J. Han, and J. Lu, "Actor-critic method for high dimensional static Hamilton–Jacobi–Bellman partial differential equations based on neural networks," SIAM J. Sci. Comput. **43**, A4043–A4066 (2021).
[14]J. Han, M. Nica, and A. R. Stinchcombe, "A derivative-free method for solving elliptic partial differential equations with deep neural networks," J. Comput. Phys. **419**, 109672 (2020).
[15]C. Martin, H. Zhang, J. Costacurta, M. Nica, and A. Stinchcombe, "Solving elliptic equations with Brownian motion: Bias reduction and temporal difference learning," Methodol. Comput. Appl. Probab. **24**, 1603–1626 (2022).
[16]K. Fackeldey, M. Oster, L. Sallandt, and R. Schneider, "Approximative policy iteration for exit time feedback control problems driven by stochastic differential equations using tensor train format," Multiscale Model. Simul. **20**, 379–403 (2022).
[17]D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," Nature **529**, 484–489 (2016).
[18]D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," arXiv:1712.01815 (2017).
[19]W. B. Powell, "From reinforcement learning to optimal control: A unified framework for sequential decisions," in *Handbook of Reinforcement Learning and Control* (Springer International Publishing, Cham, 2021), pp. 29–74.
[20]H. Wang, T. Zariphopoulou, and X. Zhou, "Reinforcement learning in continuous time and space: A stochastic control approach," J. Mach. Learn. Res. **21**, 1–34 (2020).
[21]R. S. Sutton and A. G. Barto, in *Reinforcement Learning: An Introduction*, 2nd ed. (The MIT Press, 2018).
[22]D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller,"Deterministic policy gradient algorithms," PMLR **32**, 387–395 (2014), https://proceedings.mlr.press/v32/silver14.html.
[23]C. Hartmann, L. Richter, C. Schütte, and W. Zhang, "Variational characterization of free energy: Theory and algorithms," Entropy **19**, 626 (2017).
[24]C. Hartmann, C. Schütte, and W. Zhang, "Model reduction algorithms for optimal control and importance sampling of diffusions," Nonlinearity **29**, 2298–2326 (2016).
[25]E. Vanden-Eijnden and J. Weare, "Rare event simulation of small noise diffusions," Commun. Pure Appl. Math. **65**, 1770–1803 (2012).
[26]C. Hartmann and L. Richter, "Nonasymptotic bounds for suboptimal importance sampling," SIAM/ASA J. Uncertain. Quantification **12**, 309–346 (2024).
[27]C. Hartmann, R. Banisch, M. Sarich, T. Badowski, and C. Schütte, "Characterization of rare events in molecular dynamics," Entropy **16**, 350–376 (2013).
[28]H. Van Hasselt, "Reinforcement learning in continuous state and action spaces," in *Reinforcement Learning: State-Of-The-Art* (Springer, 2012), pp. 207–251.

[29]M. L. Puterman, in *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. (John Wiley and Sons, Inc., USA, 1994).
[30]B. Recht, "A tour of reinforcement learning: The view from continuous control," Annu. Rev. Control, Robot., Autonom. Syst. **2**, 253–279 (2019).
[31]R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, edited by S. Solla, T. Leen, and K. Müller (MIT Press, 1999), Vol. 12.
[32]Y. Li, "Deep reinforcement learning," arXiv:1810.06339 (2018).
[33]C. J. C. H. Watkins and P. Dayan, "Q-learning," Mach. Learn. **8**, 279–292 (1992).
[34]V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv:1312.5602 (2013).
[35]H. van Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, edited by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta (Curran Associates, Inc., 2010), Vol. 23.
[36]H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI'16* (AAAI Press, 2016), pp. 2094–2100.
[37]Y. Jia and X. Zhou, "Q-learning in continuous time," J. Mach. Learn. Res. **24**, 1–61 (2023), https://jmlr.org/papers/v24/22-0755.html.
[38]R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Mach. Learn. **8**, 229–256 (1992).
[39]J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," PMLR **37**, 1889–1897 (2015), https://proceedings.mlr.press/v37/schulman15.html.
[40]J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," arXiv:1506.02438 [cs.LG] (2018).
[41]J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv:1707.06347 (2017).
[42]L. Weng (2018). "A (long) peek into reinforcement learning," https://lilianweng.github.io/posts/2018-02-19-rl-overview/.
[43]D. J. Higham, "An algorithmic introduction to numerical simulation of stochastic differential equations," SIAM Rev. **43**, 525–546 (2001).
[44]E. Ribera Borrell, J. Quer, L. Richter, and C. Schütte, "Improving control based importance sampling strategies for metastable diffusions via adapted metadynamics," SIAM J. Sci. Comput. **46**, S298–S323 (2024).
[45]J. Peters, "Policy gradient methods," Scholarpedia **5**, 3698 (2010), Revision #137199.
[46]H. C. Lie, "Fréchet derivatives of expected functionals of solutions to stochastic differential equations," arXiv:2106.09149 (2021).
[47]T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv:1509.02971 [cs.LG] (2019).
[48]S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," (2018).
[49]D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980 (2014).
[50]L. Baird and A. Moore, "Gradient descent for general reinforcement learning," in *Advances in Neural Information Processing Systems*, edited by M. Kearns, S. Solla, and D. Cohn (MIT Press, 1998), Vol. 11.
[51]Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," PMLR **48**, 1995–2003 (2016), https://proceedings.mlr.press/v48/wangf16.html.
[52]T. Leliévre, G. Robin, I. Sekkat, G. Stoltz, and G. V. Cardoso, "Generative methods for sampling transition paths in molecular dynamics," ESAIM: Proc. Surveys **73**, 238–256 (2023).
[53]L. Bennett, B. Melchers, and B. Proppe, "Curta: A general-purpose high-performance computer at ZEDAT," Freie Universität Berlin, 2020.