A Pontryagin Perspective on Reinforcement Learning

Onno Eberhard[1,2] **Claire Vernade**[2] **Michael Muehlebach**[1]

[1]Max Planck Institute for Intelligent Systems, Tübingen, Germany

[2]University of Tübingen

Abstract

Reinforcement learning has traditionally focused on learning state-dependent policies to solve optimal control problems in a *closed-loop* fashion. In this work, we introduce the paradigm of *open-loop reinforcement learning* where a fixed action sequence is learned instead. We present three new algorithms: one robust model-based method and two sample-efficient model-free methods. Rather than basing our algorithms on Bellman's equation from dynamic programming, our work builds on *Pontryagin's principle* from the theory of open-loop optimal control. We provide convergence guarantees and evaluate all methods empirically on a pendulum swing-up task, as well as on two high-dimensional MuJoCo tasks, demonstrating remarkable performance compared to existing baselines.

1 Introduction

Reinforcement learning (RL) refers to "the optimal control of incompletely-known Markov decision processes" [Sutton and Barto, 2018, p. 2]. It has traditionally focused on applying dynamic programming algorithms, such as value iteration or policy iteration, to situations where the environment is unknown. These methods solve optimal control problems in a closed-loop fashion by learning feedback policies, which map states x_t to actions u_t. In contrast, this work introduces the paradigm of *open-loop reinforcement learning* (OLRL), in which entire action sequences $u_{0:T-1}$, over a horizon T, are learned instead. The closed-loop and open-loop control paradigms are illustrated in Fig. 1.

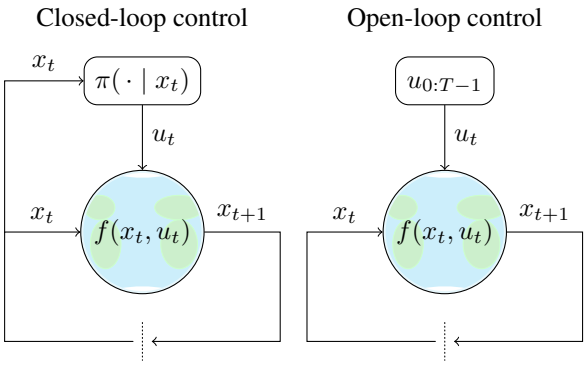

Figure 1: Comparison of closed-loop (feedback) and open-loop (feedforward) control. In closed-loop reinforcement learning (RL), the goal is to learn a policy (π). In open-loop RL, a fixed sequence of actions ($u_{0:T-1}$) is learned instead, with the action u_t independent of the states $x_{0:t}$.

There are good reasons why RL research has historically focused on closed-loop control. Many problems, particularly those with stochastic or unstable dynamics, cannot be satisfactorily solved in an open-loop fashion. However, in some settings, open-loop control is the only viable option, for example when economic reasons prevent the use of sensors. A feedback policy may also produce unpredictable behavior if the environment changes, whereas an open-loop solution, which cannot detect environmental changes, is robust in such cases. In open-loop control, we optimize over action sequences, and so we need to find one action per time step. In finite-horizon settings, this can be considerably cheaper than closed-loop control, as optimizing over policies amounts to finding one action for each state of the system. Finally, there are also fundamental control-theoretic reasons

ICML 2024 Workshop on Foundations of Reinforcement Learning and Control.

for combining a feedback policy that attenuates disturbances with a feedforward controller that is optimized for reference tracking. In many situations, such a design significantly outperforms a solution purely based on feedback [e.g., Åström and Murray, 2021, Sec. 12.4].

Indeed, for these reasons, there exists a large body of literature on open-loop optimal control theory [Pontryagin et al., 1962]. In this work, we introduce a family of three new open-loop RL algorithms by adapting this theory to the setting of unknown dynamics. Whereas closed-loop RL is largely based on approximating the Bellman equation, the central equation of dynamic programming, we base our algorithms on approximations of *Pontryagin's principle*, the central equation of open-loop optimal control. We provide a model-based method whose convergence we prove to be robust to modeling errors. We then extend this algorithm to settings with completely unknown dynamics and propose two fully online model-free methods. Finally, we empirically demonstrate the robustness and sample efficiency of our methods on an inverted pendulum swing-up task and on two complex MuJoCo tasks.

2 Background

We consider a reinforcement learning setup with state space $\mathcal{X} \subset \mathbb{R}^D$, action space $\mathcal{U} \subset \mathbb{R}^K$, and a fixed horizon $T \in \mathbb{N}$. An episode always starts with the same initial state $x_0 \in \mathcal{X}$ and follows the deterministic dynamics $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, such that $x_{t+1} = f(x_t, u_t)$ for all times $t \in [T-1]_0$.[1] After every transition, a deterministic reward $r(x_t, u_t) \in \mathbb{R}$ is received, and at the end of an episode, an additional terminal reward $r_T(x_T) \in \mathbb{R}$ is computed as a function of the terminal state x_T. We can thus define the value of the state x_t at time t as the sum of future rewards

$$V_t(x_t; u_{t:T-1}) \doteq \sum_{\tau=t}^{T-1} r(x_\tau, u_\tau) + r_T(x_T) = r(x_t, u_t) + V_{t+1}\{f(x_t, u_t); u_{t+1:T-1}\},$$

where we defined V_T as the terminal reward function r_T. Our goal is to find a sequence of actions $u_{0:T-1} \in \mathcal{U}^T$ maximizing the total sum of rewards $J(u_{0:T-1}) \doteq V_0(x_0; u_{0:T-1})$. We will tackle this trajectory optimization problem using gradient ascent.

Pontryagin's principle. The gradient of the objective function J with respect to the action u_t is

$$\nabla_{u_t} J(u_{0:T-1}) = \nabla_u r(x_t, u_t) + \nabla_u f(x_t, u_t) \underbrace{\nabla_x V_{t+1}(x_{t+1}; u_{t+1:T-1})}_{\lambda_{t+1} \in \mathbb{R}^D}, \tag{1}$$

where the terms of J related to the earlier time steps $\tau \in [t-1]_0$ vanish, as they do not depend on u_t. We denote Jacobians as $(\nabla_y f)_{i,j} \doteq \frac{\partial f_j}{\partial y_i}$. The *costates* $\lambda_{1:T}$ are defined as the gradients of the value function along the given trajectory. They can be computed through a backward recursion:

$$\lambda_t \doteq \nabla_x V_t(x_t; u_{t:T-1}) = \nabla_x r(x_t, u_t) + \nabla_x f(x_t, u_t)\lambda_{t+1} \tag{2}$$

$$\lambda_T \doteq \nabla V_T(x_T) = \nabla r_T(x_T). \tag{3}$$

The gradient (1) of the objective function can thus be obtained by means of one forward pass through the dynamics f (a rollout), yielding the states $x_{0:T}$, and one backward pass through Eqs. (2) and (3), yielding the costates $\lambda_{1:T}$. The stationarity condition arising from setting Eq. (1) to zero, where the costates are computed from Eqs. (2) and (3), is known as *Pontryagin's principle*. (Pontryagin's principle in fact goes much further than this, as it generalizes to infinite-dimensional and constrained settings.) We re-derive Eqs. (1) to (3) using the method of Lagrange multipliers in Appendix B.

3 Method

If the dynamics are known, then the trajectory can be optimized as described above, by performing gradient ascent with the gradients computed according to Pontryagin's equations (1) to (3). In this work, we adapt this idea to the domain of reinforcement learning, where the dynamics are unknown. In RL, we are able to interact with the environment, so the forward pass through the dynamics f is not an issue. However, the gradient computation according to Pontryagin's principle requires the Jacobians $\nabla_x f_t \doteq \nabla_x f(x_t, u_t)$ and $\nabla_u f_t \doteq \nabla_u f(x_t, u_t)$ of the unknown dynamics. In our

[1] For $n \in \mathbb{N}$, we write $[n] \doteq \{1, 2, \ldots, n\}$ and $[n]_0 \doteq \{0, 1, \ldots, n\}$. Unless explicitly mentioned, all time-dependent equations hold for all $t \in [T-1]_0$.

methods, which follow the structure of Algorithm 1, we therefore replace these Jacobians by estimates $A_t \simeq \nabla_x f_t$ and $B_t \simeq \nabla_u f_t$. We now show that, given sufficiently accurate estimates A_t and B_t, this algorithm still converges to a local maximum of the objective J. In the following sections we then discuss model-based and model-free methods to obtain such estimates.

3.1 Convergence of Algorithm 1

Before discussing concrete methods for open-loop RL, whose main concern is the construction of appropriate estimates A_t and B_t, we first show that replacing $\nabla_x f_t$ and $\nabla_u f_t$ with such estimates is a good idea. In particular, we show that, under certain assumptions on the accuracy of A_t and B_t, Algorithm 1 converges to an unbiased local optimum of the true objective J. Our convergence result relies on the following three assumptions.

Algorithm 1: Open-loop reinforcement learning

Input: Optimization steps $N \in \mathbb{N}$, step size $\eta > 0$

Initialize $u_{0:T-1}$ (initial action sequence)
for $k = 1, 2, \ldots, N$ **do**
  $x_{0:T} \leftarrow \text{rollout}(u_{0:T-1})$ `// Forward pass`
  `// Backward pass`
  $\tilde{\lambda}_T \leftarrow \nabla r_T(x_T)$
  for $t = T-1, T-2, \ldots, 0$ **do**
    `// Jacobian estimation`
    $A_t, B_t \simeq \nabla_x f(x_t, u_t), \nabla_u f(x_t, u_t)$
    `// Pontryagin update`
    $\tilde{\lambda}_t \leftarrow \nabla_x r(x_t, u_t) + A_t \tilde{\lambda}_{t+1}$
    $g_t \leftarrow \nabla_u r(x_t, u_t) + B_t \tilde{\lambda}_{t+1}$
    $u_t \leftarrow u_t + \eta g_t$ `// Gradient ascent`
  end
end

Assumption 3.1. *All rewards are encoded in the terminal reward r_T. In other words, $r(x, u) = 0$ for all $x \in \mathcal{X}$ and $u \in \mathcal{U}$.*

This assumption is without loss of generality, since we can augment the state x_t by a single real variable ρ_t that captures the sum of the running rewards (i.e., $\rho_0 = 0$ and $\rho_{t+1} = \rho_t + r(x_t, u_t)$). An equivalent setup that satisfies Assumption 3.1 is then obtained by defining a new terminal reward function $r'_T(x_T, \rho_T) \doteq r_T(x_T) + \rho_T$ and setting the running rewards r' to zero.

Assumption 3.2. *There exist constants $\gamma, \zeta > 0$ with $\gamma + \zeta + \gamma\zeta < 1$ such that for any trajectory $(u_{0:T-1}, x_{0:T})$ encountered by Algorithm 1, the following properties hold for all $t \in [T-1]_0$:*

(a) The error of A_{t+s} is bounded, for all $s \in [T-t]$, in the following way:

$$\|A_{t+s} - \nabla_x f_{t+s}\| \leq \frac{\gamma}{3^s} \frac{\underline{\sigma}(\nabla_u f_t)}{\bar{\sigma}(\nabla_u f_t)} \left\{ \prod_{i=1}^{s-1} \frac{\underline{\sigma}(\nabla_x f_{t+i})}{\bar{\sigma}(\nabla_x f_{t+i})} \right\} \underline{\sigma}(\nabla_x f_{t+s}).$$

(b) The error of B_t is bounded in the following way: $\|B_t - \nabla_u f_t\| \leq \zeta \underline{\sigma}(\nabla_u f_t)$.

Here, $\underline{\sigma}(A)$ and $\|A\| \doteq \bar{\sigma}(A)$ denote the minimum and maximum singular value of A, respectively.

This assumption restricts the errors of the estimates A_t and B_t which are used in place of the true Jacobians $\nabla_x f_t$ and $\nabla_u f_t$ in Algorithm 1. Although the use of the true system for collecting rollouts prevents a buildup of error in the forward pass, any error in the approximate costate $\tilde{\lambda}_t$ can still be amplified by the Jacobians of earlier time steps, $\nabla_x f_\tau$ for $\tau \in [t-1]$, during the backward pass. Thus, to ensure convergence to a stationary point of the true objective function, the errors of these Jacobians need to be small. This is particularly important for t close to T, as the errors will be amplified over more time steps. Assumption 3.2 provides a quantitative characterization of this intuition.

Assumption 3.3. *There exists a constant $L > 0$ such that, for all action sequences $u^A_{0:T-1}, u^B_{0:T-1} \in \mathcal{U}^T$ and all times $t \in [T-1]_0$, $\|\nabla_{u_t} J(u^A_{0:T-1}) - \nabla_{u_t} J(u^B_{0:T-1})\| \leq L\|u^A_t - u^B_t\|$.*

This final assumption states that the objective function J is L-smooth with respect to the action u_t at each time step $t \in [T-1]_0$, which is a standard assumption in nonconvex optimization. This implies that the dynamics f are smooth as well. We are now ready to state the convergence result.

Theorem 3.4. *Suppose Assumptions 3.1 to 3.3 hold with γ, ζ, and L. Let $\mu \doteq 1 - \gamma - \zeta - \gamma\zeta$ and $\nu \doteq 1 + \gamma + \zeta + \gamma\zeta$. If the step size η is chosen small enough such that $\alpha \doteq \mu - \frac{1}{2}\eta L \nu^2$ is positive, then the iterates $(u^{(k)}_{0:T-1})_{k=0}^{N-1}$ of Algorithm 1 satisfy, for all $N \in \mathbb{N}$ and $t \in [T-1]_0$,*

$$\frac{1}{N} \sum_{k=0}^{N-1} \|\nabla_{u_t} J(u^{(k)}_{0:T-1})\|^2 \leq \frac{J^\star - J(u^{(0)}_{0:T-1})}{\alpha \eta N},$$

where $J^\star \doteq \sup_{u \in \mathcal{U}^T} J(u)$ is the optimal value of the initial state.

Proof. See Appendix D. □

3.2 Model-based open-loop RL

The most direct way to approximate the Jacobians $\nabla_x f_t$ and $\nabla_u f_t$ is by using a (learned or manually designed) differentiable model $\tilde{f} : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ of the dynamics f and setting $A_t = \nabla_x \tilde{f}(x_t, u_t)$ and $B_t = \nabla_u \tilde{f}(x_t, u_t)$ in Line 6 of Algorithm 1. Theorem 3.4 guarantees that this *model-based* open-loop RL method (see Algorithm A.1) is robust to a certain amount of modeling error. In contrast to this, consider the more naive *planning* method of using the model to directly obtain the gradient by differentiating

$$J(u_{0:T-1}) \simeq r(x_0, u_0) + r\{\underbrace{\tilde{f}(x_0, u_0)}_{\tilde{x}_1}, u_1\} + \cdots + r_T\{\underbrace{\tilde{f}(\tilde{f}(\cdots \tilde{f}(\tilde{f}(x_0, u_0), u_1) \cdots), u_{T-1})}_{\tilde{x}_T}\}$$

with respect to the actions $u_{0:T-1}$ using the backpropagation algorithm. In Appendix C, we show that this approach is exactly equivalent to an approximation of Algorithm 1 where, in addition to setting $A_t = \nabla_x \tilde{f}(x_t, u_t)$ and $B_t = \nabla_u \tilde{f}(x_t, u_t)$, the forward pass of Line 3 is replaced by an imagined forward pass through the model $\tilde{f}$. In Section 4, we empirically demonstrate that this planning method, whose convergence is not guaranteed by Theorem 3.4, is much less robust to modeling errors than the open-loop RL approach.

3.3 Model-free on-trajectory open-loop RL

Access to a reasonably accurate model may not always be feasible, and as Algorithm 1 only requires the Jacobians of the dynamics along the current trajectory, a global model is also not necessary. In the following two sections, we propose two methods that directly estimate the Jacobians $\nabla_x f_t$ and $\nabla_u f_t$ from rollouts in the environment. We call these methods *model-free*, as the estimated Jacobians are only valid along the current trajectory, and thus cannot be used for planning.

Our goal is to estimate the Jacobians $\nabla_x f(\bar{x}_t, \bar{u}_t)$ and $\nabla_u f(\bar{x}_t, \bar{u}_t)$ that lie along the trajectory induced by the action sequence $\bar{u}_{0:T-1}$. These Jacobians measure how the next state ($\bar{x}_{t+1}$) changes if the current state or action ($\bar{x}_t$, $\bar{u}_t$) are slightly perturbed. More formally, the dynamics f may be linearized about the reference trajectory $(\bar{u}_{0:T-1}, \bar{x}_{0:T})$ as

$$\underbrace{f(x_t, u_t) - f(\bar{x}_t, \bar{u}_t)}_{\Delta x_{t+1}} \simeq \nabla_x f(\bar{x}_t, \bar{u}_t)^\top \underbrace{(x_t - \bar{x}_t)}_{\Delta x_t} + \nabla_u f(\bar{x}_t, \bar{u}_t)^\top \underbrace{(u_t - \bar{u}_t)}_{\Delta u_t},$$

which is a valid approximation if the perturbations $\Delta x_{0:T}$ and $\Delta u_{0:T-1}$ are small. By collecting a dataset of $M \in \mathbb{N}$ rollouts with slightly perturbed actions, we can thus estimate the Jacobians by solving the (analytically tractable) least-squares problem

$$\underset{[A_t^\top \;\; B_t^\top] \in \mathbb{R}^{D \times (D+K)}}{\arg\min} \sum_{i=1}^{M} \| A_t^\top \Delta x_t^{(i)} + B_t^\top \Delta u_t^{(i)} - \Delta x_{t+1}^{(i)} \|^2. \tag{4}$$

This technique is illustrated in Fig. 2a (dashed purple line). Using these estimates in Algorithm 1 yields a model-free method we call *on-trajectory*, as the gradient estimate relies only on data generated based on the current trajectory (see Algorithm A.2 for details). We see a connection to on-policy methods in closed-loop reinforcement learning, where the policy gradient estimate (or the Q-update) similarly depends only on data generated under the current policy. Like on-policy methods, on-trajectory methods will benefit greatly from the possibility of parallel environments, which could reduce the effective complexity of the forward pass stage from $M+1$ rollouts to that of a single rollout.

Exploiting the Markovian structure. Consider a direct linearization of the objective function J about the current trajectory. Writing the action sequence as a vector $\bar{\boldsymbol{u}} \doteq \mathrm{vec}(\bar{u}_{0:T-1}) \in \mathbb{R}^{TK}$, this linearization is given, for $\boldsymbol{u} \in \mathbb{R}^{TK}$ close to $\bar{\boldsymbol{u}}$, by

$$J(\boldsymbol{u}) \simeq J(\bar{\boldsymbol{u}}) + \nabla J(\bar{\boldsymbol{u}})^\top (\boldsymbol{u} - \bar{\boldsymbol{u}}).$$

We can thus estimate the gradient of the objective function by solving the least squares problem

$$\nabla J(\bar{\boldsymbol{u}}) \simeq \underset{\boldsymbol{g} \in \mathbb{R}^{TK}}{\arg\min} \sum_{i=1}^{M} \{J(\boldsymbol{u}_i) - J(\bar{\boldsymbol{u}}) - \boldsymbol{g}^\top (\boldsymbol{u}_i - \bar{\boldsymbol{u}})\}^2,$$

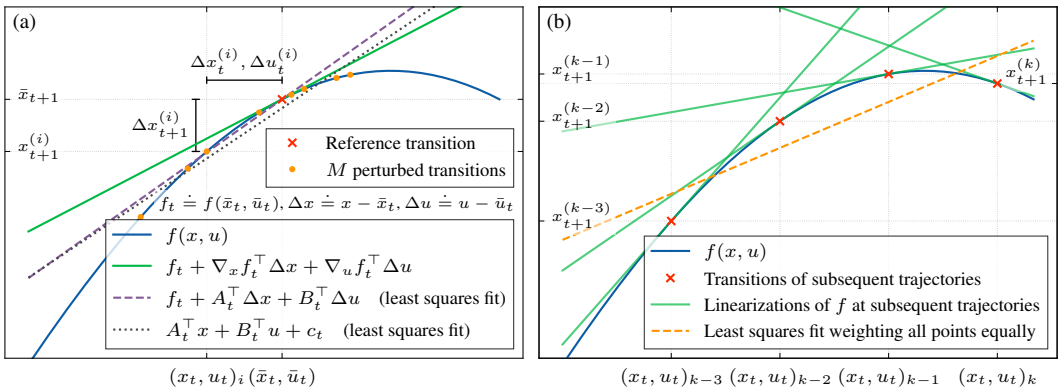

Figure 2: (a) The Jacobians of f (slope of the green linearization) at the reference point $(\bar{x}_t, \bar{u}_t)$ can be estimated from the transitions $\{(x_t^{(i)}, u_t^{(i)}, x_{t+1}^{(i)})\}_{i=1}^M$ of M perturbed rollouts. (b) The Jacobians of subsequent trajectories (indexed by k) remain close. To estimate the Jacobian at iteration k, the most recent iterate $(k-1)$ is more relevant than older iterates.

where $\{\boldsymbol{u}_i\}$ are $M \in \mathbb{N}$ slightly perturbed action sequences. Due to the dimensionality of $\bar{\boldsymbol{u}}$, this method requires $\mathcal{O}(TK)$ rollouts to estimate the gradient. In contrast to this, our approach leverages the Markovian structure of the problem, including the fact that we observe the states $x_{0:T}$ in each rollout. As the Jacobians at all time steps are estimated jointly, we can expect to get a useful gradient estimate from only $\mathcal{O}(D^2 + DK)$ rollouts, which significantly reduces the sample complexity if T is large. This gain in efficiency is demonstrated empirically in Section 4.

3.4 Model-free off-trajectory open-loop RL

The on-trajectory algorithm is sample-efficient in the sense that it leverages the problem structure, but a key inefficiency remains: the rollout data sampled at each iteration is discarded after the action sequence is updated. In this section, we propose an *off-trajectory* method that implicitly uses the data from previous trajectories to construct the Jacobian estimates. Our approach is based on the following observation. If the dynamics f are smooth and the step size η is small, then the updated trajectory $(u_{0:T-1}^{(k)}, x_{0:T}^{(k)})$ will remain close to the previous iterate $(u_{0:T-1}^{(k-1)}, x_{0:T}^{(k-1)})$. Furthermore, the Jacobians along the updated trajectory will be similar to the previous Jacobians, as illustrated in Fig. 2b. Thus, we propose to estimate the Jacobians along the current trajectory from *a single rollout only* by bootstrapping our estimates using the Jacobian estimates from the previous iteration.

Consider again the problem of estimating the Jacobians from multiple perturbed rollouts, illustrated in Fig. 2a. Instead of relying on a reference trajectory and Eq. (4), we can estimate the Jacobians by fitting a linear regression model to the dataset of M perturbed transitions. Solving

$$\underset{[A_t^\top \; B_t^\top \; c_t] \in \mathbb{R}^{D \times (D+K+1)}}{\arg\min} \sum_{i=1}^{M} \| A_t^\top x_t^{(i)} + B_t^\top u_t^{(i)} + c_t - x_{t+1}^{(i)} \|^2 \tag{5}$$

yields an approximate linearization $f(x_t, u_t) \simeq A_t^\top x_t + B_t^\top u_t + c_t = F_t z_t$, with $F_t \doteq [A_t^\top \; B_t^\top \; c_t]$ and $z_t \doteq (x_t, u_t, 1) \in \mathbb{R}^{D+K+1}$. This approximation is also shown in Fig. 2a (dotted gray line).[2] At iteration k, given the estimate $F_t^{(k-1)}$ and a new point $z_t^{(k)} = (x_t^{(k)}, u_t^{(k)}, 1)$ and corresponding target $x_{t+1}^{(k)}$, computing the new estimate $F_t^{(k)}$ is a problem of online linear regression. We solve this regression problem using an augmented version of the *recursive least squares* (RLS) algorithm [e.g., Ljung, 1999, Sec. 11.2]. By introducing a prior precision matrix $Q_t^{(0)} \doteq q_0 I$ for each time t, where $q_0 > 0$, in Algorithm A.3 we compute the update at iteration $k \in \mathbb{N}$ as

$$\begin{aligned} Q_t^{(k)} &= \alpha Q_t^{(k-1)} + (1-\alpha) q_0 I + z_t^{(k)} \{z_t^{(k)}\}^\top \\ F_t^{(k)} &= F_t^{(k-1)} + \{Q_t^{(k)}\}^{-1} z_t^{(k)} \{x_{t+1}^{(k)} - F_t^{(k-1)} z_t^{(k)}\}^\top . \end{aligned} \tag{6}$$

[2] If we replace Eq. (4) by Eq. (5) in Algorithm A.2, we get a slightly different on-trajectory method that has a similar performance.

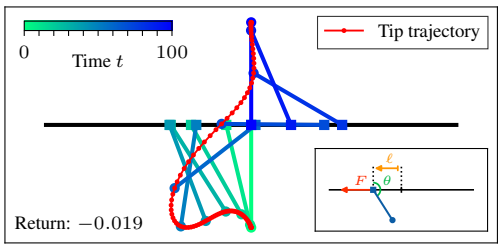

Figure 3: The inverted pendulum swing-up problem. The goal is to control the force F such that the tip of the pendulum swings up above the base. The shown solution was found by the on-trajectory method of Section 3.3.

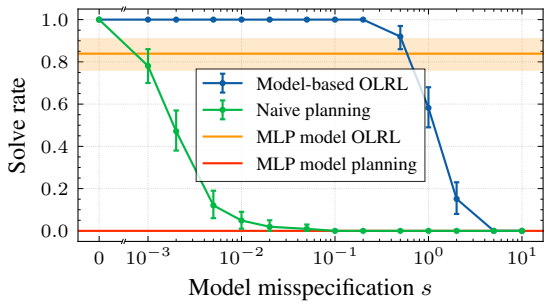

Figure 4: The model-based open-loop RL algorithm can solve the pendulum problem reliably even with a considerable model error.

Forgetting and stability. The standard RLS update of the precision matrix corresponds to Eq. (6) with $\alpha = 0$. In the limit as $q_0 \to 0$, the RLS algorithm is equivalent to the batch processing of Eq. (5), which treats all points equally. However, as illustrated in Fig. 2b, points from recent trajectories should be given more weight, as transitions that happened many iterations ago will give little information about the Jacobians along the current trajectory. We thus incorporate a *forgetting factor* $\alpha \in (0, 1)$ into the precision update with the effect that past data points are exponentially downweighted:

$$Q_t^{(k)} = \alpha Q_t^{(k-1)} + z_t^{(k)} \{z_t^{(k)}\}^\top \quad \rightsquigarrow \quad Q_t^{(k)} = \alpha^k q_0 I + \sum_{i=1}^{k} \alpha^{k-i} z_t^{(i)} \{z_t^{(i)}\}^\top. \tag{7}$$

This forgetting factor introduces a new problem: instability. If subsequent trajectories lie close to each other, then the sum of outer products may become singular (clearly, if all z_i are identical, then the sum has rank 1). As the prior $q_0 I$ is downweighted, at some point Q may become singular. Our modification in Eq. (6) adds $(1 - \alpha) q_0 I$ in each update, which has the effect of removing the α^k coefficient in front of $q_0 I$ in Eq. (7). If the optimization procedure converges, then eventually subsequent trajectories will indeed lie close together. Although Eq. (6) prevents issues with instability, the quality of the Jacobian estimates will still degrade, as this estimation inherently requires perturbations (see Section 3.3). In Algorithm A.3, we thus slightly perturb the actions used in each rollout to get more diverse data.

4 Experiments

4.1 Inverted pendulum swing-up

We empirically evaluate our algorithms on the inverted pendulum swing-up problem shown in Fig. 3. The state at time t is $x_t = (\ell, \dot{\ell}, \theta, \dot{\theta})_t \in \mathbb{R}^4$, where ℓ is the position of the cart on the bar and θ is the pendulum angle. The action u_t is the horizontal force F applied to the cart at time t. Episodes are of length $T = 100$, the running reward $r(x, u) \propto -u^2$ penalizes large forces, and the terminal reward $r_T(x) = -\|x\|_1$ defines the goal state to be at rest in the upright position.

Performance metrics. We monitor several performance criteria, all of which are based on the return J. First, we define $J_{\max} \doteq \max_{k \in [N]_0} J(u_{0:T-1}^{(k)})$ as the return achieved by the best action sequence over a complete learning process of N optimization steps. Based on this quantity, we say that the swing-up task is *solved* if $J_{\max} > -0.03$. This threshold was determined empirically. If the algorithm or the model is randomized, then $[J_{\max} > -0.03]$ is a Bernoulli random variable whose mean, which we call the *solve rate*, depends on the quality of the learning algorithm. We repeat all experiments with 100 random seeds and show 95% bootstrap confidence intervals in all plots.

Robustness: model-based open-loop RL. We start with the model-based method for open-loop RL of Section 3.2 (Algorithm A.1). In Theorem 3.4, we proved that this method can accommodate some model error and still converge to a local maximum of the true objective. In Fig. 4, we empirically analyze this robustness property on the pendulum system. This system has five parameters: the masses of the cart and the pendulum tip, the length of the pendulum, and the friction coefficients for linear and rotational motion. To test the robustness of our algorithm against model misspecification,

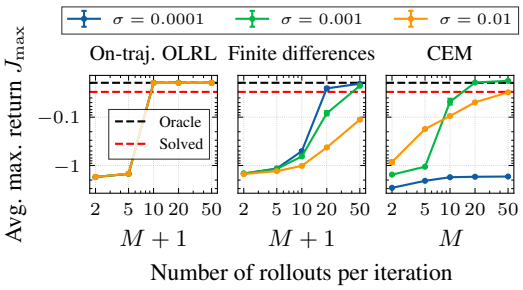

Figure 5: The on-trajectory open-loop RL method is more sample-efficient than the finite-difference and cross-entropy methods. It is also much less sensitive to the noise scale σ.

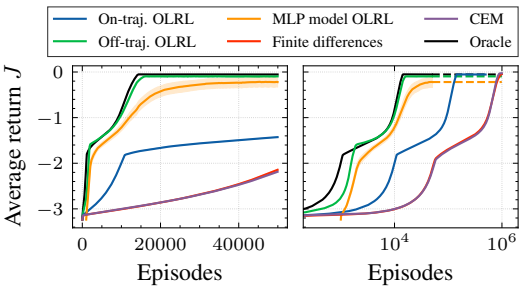

Figure 6: Learning curves on the pendulum task. On the right, we show a longer time period in log scale. The off-trajectory open-loop RL method converges almost as fast as the oracle method.

we use a pendulum system with inaccurate parameters as the model $\tilde{f}$. Concretely, if m_i is the ith parameter of the true system, we sample the corresponding model parameter $\tilde{m}_i$ from a log-normal distribution centered at m_i, such that $\tilde{m}_i = \xi m_i$, with $\ln \xi \sim \mathcal{N}(0, s^2)$. The severity of the model error is then controlled by the scale parameter s. In Fig. 4, we compare the performance of our method with the planning procedure described in Section 3.2, in which the forward pass is performed through the model $\tilde{f}$ instead of the real system f. Whereas the planning method only solves the pendulum reliably with the true system as the model ($s = 0$), the open-loop RL method can accommodate a considerable model misspecification.

In a second experiment, we represent the model $\tilde{f}$ by a small multi-layer perceptron (MLP). The model is learned from 1000 rollouts on the pendulum system, with the action sequences sampled from a pink noise distribution, as suggested by Eberhard et al. [2023]. Figure 4 shows the performance achieved with this model by our proposed algorithm and by the planning method. As the MLP model represents a considerable misspecification of the true dynamics, only the open-loop RL method manages to solve the pendulum task.

Structure: on-trajectory open-loop RL. In Section 3.3, we proposed a model-free approach (Algorithm A.2) that uses rollouts to directly estimate the Jacobians needed to update the action sequence. It is clear from Eq. (4) that more rollouts (i.e., larger M) will give more accurate Jacobian estimates, and therefore increase the quality of the gradient approximation. In Fig. 5, we analyze the sample efficiency of the on-trajectory OLRL algorithm by comparing the performance achieved at different values of M, where the number N of optimization steps remains fixed. We compare our method to the finite-difference approach described at the end of Section 3.3 and to the gradient-free cross-entropy method [CEM; Rubinstein, 1999]. Both these methods also update the action sequence on the basis of M perturbed rollouts in the environment. As in our method, the M action sequences are perturbed using Gaussian white noise with noise scale σ. We describe both baselines in detail in Appendix F. The *oracle* performance shown in Fig. 5 corresponds to Algorithm 1 with the true gradient, i.e., $A_t = \nabla_x f_t$ and $B_t = \nabla_u f_t$.

We note two things in Fig. 5. First, the performance of both the finite-difference method and CEM heavily depends on the choice of the noise scale σ, whereas our method performs identically for all three values of σ. Second, even for tuned values of σ, the finite-difference method and CEM still need approximately twice as many rollouts per iteration as the open-loop RL method to reliably swing up the pendulum. At 10 rollouts per iteration, our method matches the oracle's performance, while both baselines are below the *solved* threshold of $J_{\max} = -0.03$. This empirically confirms our theoretical claims at the end of Section 3.3, where we argue that exploiting the Markovian structure of the problem leads to increased sample efficiency.

Efficiency: off-trajectory open-loop RL. Finally, we turn to the method proposed in Section 3.4 (Algorithm A.3), which promises increased sample efficiency by estimating the Jacobians in an off-trajectory fashion. The performance of this algorithm is shown in Fig. 6, where the learning curves of all our methods as well as the two baselines and the oracle are plotted. For the on-trajectory methods compared in Fig. 5, we chose for each the minimum number of rollouts M such that, under the best choice for σ, the method would reliably solve the swing-up task. The hyperparameters for all

methods are summarized in Appendix G. It can be seen that the off-trajectory method, which only requires one rollout per iteration, converges much faster than the on-trajectory open-loop RL method.

4.2 MuJoCo

While the inverted pendulum system is an illustrative example for exploring the different merits of our algorithms, it is a relatively simple task with smooth, low-dimensional, and deterministic dynamics. In this section, we test our method in two considerably more challenging environments: the `Ant-v4` and `HalfCheetah-v4` tasks provided by the OpenAI Gym library [Brockman et al., 2016, Towers et al., 2023], implemented in MuJoCo [Todorov et al., 2012]. These environments are high-dimensional, they exhibit non-smooth contact dynamics, and the initial state is randomly sampled at the beginning of each episode.

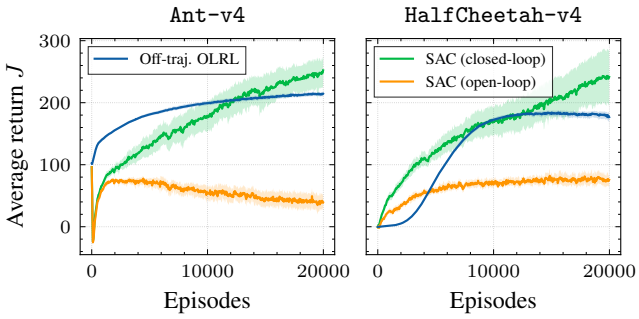

Figure 7: Learning curves of our off-trajectory open-loop RL method and soft actor-critic for two MuJoCo tasks. All experiments were repeated with 20 random seeds, and we show 95%-bootstrap confidence intervals for the average return. The horizon is fixed to $T = 100$.

We tackle these two tasks with our model-free off-trajectory method (Algorithm A.3). The results are shown in Fig. 7, where we compare to the closed-loop RL baseline soft actor-critic [SAC; Haarnoja et al., 2018a]. It can be seen that the open-loop RL method performs comparably to SAC, even though SAC learns a closed-loop policy that is capable of adapting its behavior to the initial condition.[3] In the figure, we also analyze the open-loop performance achieved by SAC. Whereas the closed-loop performance is the return obtained in a rollout where the actions are taken according to the mean of the Gaussian policy, the open-loop return is achieved by blindly executing exactly the same actions in a new episode. The discrepancy in performance is thus completely due to the stochasticity in the initial state. In Appendix E, we show that our method also works with a longer horizon T.

The results demonstrate that the open-loop RL method is robust to a certain level of stochasticity in the initial state of stable dynamical systems. Additionally, while our convergence analysis depends on the assumption of smooth dynamics, these experiments empirically demonstrate that the algorithms are also able to tackle non-smooth contact dynamics. Finally, we see that the high dimensionality of the MuJoCo systems is handled without complications. While soft actor-critic is an elegant and powerful algorithm, the combination with deep function approximation can make efficient learning more difficult. Our methods are considerably simpler and, because they are based on Pontryagin's principle rather than dynamic programming, they evade the curse of dimensionality by design, and thus do not require any function approximation.

5 Related work

Our work is inspired by numerical optimal control theory [Betts, 2010, Geering, 2007], which deals with the numerical solution of trajectory optimization problems, a setting that has also been studied in machine learning [Schaal and Atkeson, 2010, Howe et al., 2022]. As in our approach, an important aspect of solution methods is to exploit the Markovian structure of the dynamics to reduce computation [Carraro et al., 2015, Schäfer et al., 2007]. However, all these methods deal with situations where the dynamics are known, whereas our algorithms only requires an approximate model (model-based open-loop RL) or no model at all (model-free open-loop RL). Another set of related methods is known as *iterative learning control* [Moore, 1993, Ma et al., 2022, 2023], which is a control-theoretic framework that iteratively improves the execution of a task by optimizing over feedforward trajectories. However, these methods are often formulated for *trajectory tracking* tasks, while we consider a more general class of reinforcement learning problems. Chen and Braun [2019]

[3] In this comparison, our method is further disadvantaged by the piecewise constant "health" terms in the reward function of `Ant-v4`. Our method, exclusively relying on the gradient of the reward function, ignores these.

explore an idea similar as in our Algorithm A.1; their model-based control algorithm combines a rollout in a real system with an inaccurate model to construct an iterative LQR feedback controller.

Recently, deep neural networks have been used to learn representations of complex dynamical systems [Fragkiadaki et al., 2015] and Pontryagin's principle was leveraged in the optimization of control tasks based on such models [Jin et al., 2020, Böttcher et al., 2022]. However, these methods only consider the setting of closed-loop control. The combination of an exact forward pass with an approximate backward pass, which our methods are based on, has also been explored in different settings in the deep learning literature, such as spiking [Lee et al., 2016] or physical [Wright et al., 2022] neural networks, or networks that include nondifferentiable procedures, for example used for rendering [Niemeyer et al., 2020] or combinatorial optimization [Vlastelica et al., 2020].

6 Discussion

While open-loop control is a well-established field with countless applications [Diehl et al., 2006, van Zundert and Oomen, 2018, Sferrazza et al., 2020], the setting of incompletely-known dynamics has received little attention. This paper makes an important first step towards understanding how principles from open-loop optimal control can be combined with ideas from reinforcement learning while preserving convergence guarantees. We propose three algorithms that address this *open-loop RL* problem, from robust trajectory optimization with an approximate model to sample-efficient learning under fully unknown dynamics. This work focuses on reinforcement learning in continuous state and action spaces, a class of problems known to be challenging [Recht, 2019]. Although this setting allows us to leverage continuous optimization techniques, we expect that most ideas will transfer to the discrete setting and would be interested to see further research on this topic.

It is interesting to note that there are many apparent parallels between our open-loop RL algorithms and their closed-loop counterparts. The distinction between model-based and model-free methods is similar as in closed-loop RL. Similarly, the on-trajectory and off-trajectory methods we present show a tradeoff between sample efficiency and stability that is reminiscent of the tradeoffs between on-policy and off-policy methods in closed-loop RL. The question of exploration, which is central to reinforcement learning, also arises in our case. We do not address this complex problem thoroughly here but instead rely on additive Gaussian noise to sample diverse trajectories.

Limitations of Open-Loop RL. Another important open question is how open-loop methods fit into the reinforcement learning landscape. An inherent limitation of these methods is that an open-loop controller can, by definition, not react to unexpected changes in the system's state, be it due to random disturbances or an adversary. If these disturbances are small, and the system is not sensitive to small changes in the initial condition or the action sequence (roughly speaking, if the system is stable and non-chaotic), then such a reaction is not necessary. In this situation, open-loop RL works even for long time-horizons T, as we highlight in our MuJoCo experiments (see Appendix E). However, an open-loop controller cannot balance an inverted pendulum in its unstable position[4], track a reference trajectory in noisy conditions, or play Go, where reactions to the opponent's moves are constantly required. In these situations open-loop RL is not viable or only effective over a very small horizon T.

Open-loop control can be viewed as a special case of closed-loop control, and therefore it is clear that closed-loop control is much more powerful. Historically, control theory has dealt with both closed-loop and open-loop control [Åström and Murray, 2021, Skogestad and Postlethwaite, 2005, Åström and Hägglund, 1995] and there is a broad consensus in the control community that both are important [Betts, 2010, Verscheure et al., 2009, Horn et al., 2013]. However, the setting of open-loop control with unknown dynamics, which is naturally formulated through the lens of reinforcement learning, has so far received much less consideration from the community. Our algorithms provide a first solution to the open-loop RL problem and are not intended to replace any of the existing closed-loop RL algorithms. Instead, we think open-loop and closed-loop methods should complement each other. In control engineering, it is common practice to combine feedback and feedforward techniques, and such approaches are also thinkable in RL. For example, a high-level policy could execute low-level feedforward trajectories using the options framework [Sutton et al., 1999]; a related idea is explored by Hansen et al. [1996]. We believe that ultimately a combination of open-loop and closed-loop techniques will be fruitful in reinforcement learning and think that this is an important direction for future research.

[4]Except with a clever trick called *vibrational control* [Meerkov, 1980].

Acknowledgments

We thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for their support. C. Vernade is funded by the German Research Foundation (DFG) under both the project 468806714 of the Emmy Noether Programme and under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. M. Muehlebach is funded by the German Research Foundation (DFG) under the project 456587626 of the Emmy Noether Programme.

References

K. J. Åström and T. Hägglund. *PID Controllers: Theory, Design, and Tuning*. International Society for Measurement and Control, second edition, 1995. 9

K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, second edition, 2021. 2, 9

J. T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, 2010. 8, 9

L. Böttcher, N. Antulov-Fantulin, and T. Asikis. AI Pontryagin or how artificial neural networks learn to control dynamical systems. *Nature communications*, 13(333), 2022. URL https://doi.org/10.1038/s41467-021-27590-0. 9

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018. URL https://doi.org/10.1137/16M1080173. 15

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv:1606.01540*, 2016. URL http://arxiv.org/abs/1606.01540. 8

T. Carraro, M. Geiger, S. Körkel, and R. Rannacher, editors. *Multiple Shooting and Time Domain Decomposition Methods*. Springer, 2015. 8

Y. Chen and D. J. Braun. Hardware-in-the-loop iterative optimal feedback control without model-based future prediction. *IEEE Transactions on Robotics*, 35(6):1419–1434, 2019. URL https://doi.org/10.1109/TRO.2019.2929014. 8

M. Diehl, H. Bock, H. Diedam, and P.-B. Wieber. Fast direct multiple shooting algorithms for optimal robot control. In M. Diehl and K. Mombaur, editors, *Fast Motions in Biomechanics and Robotics: Optimization and Feedback Control*, pages 65–93. Springer, 2006. URL https://doi.org/10.1007/978-3-540-36119-0_4. 9

O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning. In *Proceedings of the Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hQ9V5QN27eS. 7

K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4346–4354, 2015. URL https://doi.org/10.1109/ICCV.2015.488. 9

H. P. Geering. *Optimal Control with Engineering Applications*. Springer, 2007. 8

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018a. URL http://proceedings.mlr.press/v80/haarnoja18b.html. 8, 21

T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *arXiv:1812.05905*, 2018b. URL http://arxiv.org/abs/1812.05905. 21

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 2019. URL https://proceedings.mlr.press/v97/hafner19a.html. 20

E. Hansen, A. Barto, and S. Zilberstein. Reinforcement learning for mixed open-loop and closed-loop control. *Advances in Neural Information Processing Systems*, 9, 1996. URL https://proceedings.neurips.cc/paper/1996/hash/ab1a4d0dd4d48a2ba1077c4494791306-Abstract.html. 9

G. Horn, S. Gros, and M. Diehl. Numerical trajectory optimization for airborne wind energy systems described by high fidelity aircraft models. In U. Ahrens, M. Diehl, and R. Schmehl, editors, *Airborne Wind Energy*, pages 205–218. Springer, 2013. URL https://doi.org/10.1007/978-3-642-39965-7_11. 9

N. Howe, S. Dufort-Labbé, N. Rajkumar, and P.-L. Bacon. Myriad: a real-world testbed to bridge trajectory optimization and deep learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 29801–29815, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/c0b91f9a3587bf35287f41dba5d20233-Abstract-Datasets_and_Benchmarks.html. 8

W. Jin, Z. Wang, Z. Yang, and S. Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. In *Advances in Neural Information Processing Systems*, volume 33, pages 7979–7992, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/5a7b238ba0f6502e5d6be14424b20ded-Abstract.html. 9

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the Third International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1412.6980. 21

J. H. Lee, T. Delbruck, and M. Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10, 2016. URL https://doi.org/10.3389/fnins.2016.00508. 9

L. Ljung. *System Identification: Theory for the User*. Prentice Hall, 1999. 5

H. Ma, D. Büchler, B. Schölkopf, and M. Muehlebach. A learning-based iterative control framework for controlling a robot arm with pneumatic artificial muscles. In *Proceedings of Robotics: Science and Systems*, 2022. URL https://www.roboticsproceedings.org/rss18/p029.html. 8

H. Ma, D. Büchler, B. Schölkopf, and M. Muehlebach. Reinforcement learning with model-based feedforward inputs for robotic table tennis. *Autonomous Robots*, 47:1387–1403, 2023. URL https://doi.org/10.1007/s10514-023-10140-6. 8

S. Meerkov. Principle of vibrational control: Theory and applications. *IEEE Transactions on Automatic Control*, 25(4):755–762, 1980. URL https://doi.org/10.1109/TAC.1980.1102426. 9

K. L. Moore. *Iterative Learning Control for Deterministic Systems*. Springer, 1993. URL https://doi.org/10.1007/978-1-4471-1912-8. 8

M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Niemeyer_Differentiable_Volumetric_Rendering_Learning_Implicit_3D_Representations_Without_3D_Supervision_CVPR_2020_paper.html. 9

C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius. Sample-efficient cross-entropy method for real-time planning. In *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 1049–1065. PMLR, 2021. URL https://proceedings.mlr.press/v155/pinneri21a.html. 20

L. Pontryagin, V. Boltayanskii, R. Gamkrelidze, and E. Mishchenko. *Mathematical Theory of Optimal Processes*. Wiley, 1962. 2

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html. 21

B. Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019. URL https://doi.org/10.1146/annurev-control-053018-023825. 9

R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1:127–190, 1999. URL https://doi.org/10.1023/A:1010091220143. 7, 19

S. Schaal and C. G. Atkeson. Learning control in robotics. *IEEE Robotics & Automation Magazine*, 17(2):20–29, 2010. URL https://doi.org/10.1109/MRA.2010.936957. 8

A. Schäfer, P. Kühl, M. Diehl, J. Schlöder, and H. G. Bock. Fast reduced multiple shooting methods for nonlinear model predictive control. *Chemical Engineering and Processing: Process Intensification*, 46(11):1200–1214, 2007. URL https://doi.org/10.1016/j.cep.2006.06.024. 8

C. Sferrazza, M. Muehlebach, and R. D'Andrea. Learning-based parametrized model predictive control for trajectory tracking. *Optimal Control Applications and Methods*, 41(6):2225–2249, 2020. URL https://doi.org/10.1002/oca.2656. 9

S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. Wiley, second edition, 2005. 9

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, second edition, 2018. 1

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999. URL https://doi.org/10.1016/S0004-3702(99)00052-1. 9

E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. URL https://doi.org/10.1109/IROS.2012.6386109. 8

M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium, 2023. URL https://github.com/Farama-Foundation/Gymnasium. 8

J. van Zundert and T. Oomen. On inversion-based approaches for feedforward and ILC. *Mechatronics*, 50:282–291, 2018. URL https://doi.org/10.1016/j.mechatronics.2017.09.010. 9

D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327, 2009. URL https://doi.org/10.1109/TAC.2009.2028959. 9

M. Vlastelica, A. Paulus, V. Musil, G. Martius, and M. Rolínek. Differentiation of blackbox combinatorial solvers. In *Proceedings of the Eighth International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkevoJSYPB. 9

L. G. Wright, T. Onodera, M. M. Stein, T. Wang, D. T. Schachter, Z. Hu, and P. L. McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601(7894):549–555, 2022. URL https://doi.org/10.1038/s41586-021-04223-6. 9

A Algorithms

In this section, we provide detailed descriptions of the three open-loop RL algorithms presented in the main text. The model-based algorithm of Section 3.2 is listed in Algorithm A.1, the model-free on-trajectory method of Section 3.3 is listed in Algorithm A.2, and the off-trajectory method of Section 3.4 is listed in Algorithm A.3. The hyperparameters we use in these algorithms are discussed in Appendix G.

Algorithm A.1: Model-based open-loop RL

Input: Model $\tilde{f} : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, optimization steps $N \in \mathbb{N}$, step size $\eta > 0$

Initialize $u_{0:T-1}$ (initial action sequence)
for $k = 1, 2, \ldots, N$ **do**
  `// Forward pass`
  $x_{0:T} \leftarrow \text{rollout}(u_{0:T-1})$
  `// Backward pass`
  $\tilde{\lambda}_T \leftarrow \nabla r_T(x_T)$
  for $t = T-1, T-2, \ldots, 0$ **do**
    $\tilde{\lambda}_t \leftarrow \nabla_x r(x_t, u_t) + \nabla_x \tilde{f}(x_t, u_t)\tilde{\lambda}_{t+1}$
    $g_t \leftarrow \nabla_u r(x_t, u_t) + \nabla_u \tilde{f}(x_t, u_t)\tilde{\lambda}_{t+1}$
    $u_t \leftarrow u_t + \eta g_t$ `// Gradient ascent`
  end
end

Algorithm A.2: Model-free on-trajectory open-loop RL

Input: Number of rollouts $M \in \mathbb{N}$, noise scale $\sigma > 0$, optimization steps $N \in \mathbb{N}$, step size $\eta > 0$

Initialize $\bar{u}_{0:T-1}$ (initial action sequence)
for $k = 1, 2, \ldots, N$ **do**
  `// Forward passes`
  $\bar{x}_{0:T} \leftarrow \text{rollout}(\bar{u}_{0:T-1})$
  for $i = 1, 2, \ldots, M$ **do**
    $u^{(i)}_{0:T-1} \sim \mathcal{N}(\bar{u}_{0:T-1}, \sigma I)$
    $x^{(i)}_{0:T} \leftarrow \text{rollout}(u^{(i)}_{0:T-1})$
    $\Delta u^{(i)}_{0:T-1} \leftarrow u^{(i)}_{0:T-1} - \bar{u}_{0:T-1}$
    $\Delta x^{(i)}_{0:T} \leftarrow x^{(i)}_{0:T} - \bar{x}_{0:T}$
  end
  `// Backward pass`
  $\tilde{\lambda}_T \leftarrow \nabla r_T(\bar{x}_T)$
  for $t = T-1, T-2, \ldots, 0$ **do**
    `// Jacobian estimation`
    $A_t, B_t \leftarrow \arg\min_{A_t \in \mathbb{R}^{D\times D}, B_t \in \mathbb{R}^{K\times D}} \sum_{i=1}^{M} \| A_t^\top \Delta x_t^{(i)} + B_t^\top \Delta u_t^{(i)} - \Delta x_{t+1}^{(i)} \|^2$
    `// Pontryagin update`
    $\tilde{\lambda}_t \leftarrow \nabla_x r(\bar{x}_t, \bar{u}_t) + A_t \tilde{\lambda}_{t+1}$
    $g_t \leftarrow \nabla_u r(\bar{x}_t, \bar{u}_t) + B_t \tilde{\lambda}_{t+1}$
    $\bar{u}_t \leftarrow \bar{u}_t + \eta g_t$ `// Gradient ascent`
  end
end

Algorithm A.3: Model-free off-trajectory open-loop RL

Input: Forgetting factor $\alpha \in [0, 1]$, noise scale $\sigma > 0$, initial precision $q_0 > 0$, optimization steps $N \in \mathbb{N}$, step size $\eta > 0$

Initialize $\bar{u}_{0:T-1}$ (initial action sequence)
Initialize $F_t \in \mathbb{R}^{D\times(D+K+1)}, \forall t \in [T-1]_0$
$Q_t \leftarrow q_0 I \in \mathbb{R}^{(D+K+1)\times(D+K+1)}, \forall t \in [T-1]_0$
for $k = 1, 2, \ldots, N$ **do**
  `// Forward pass`
  $u_{0:T-1} \sim \mathcal{N}(\bar{u}_{0:T-1}, \sigma I)$
  $x_{0:T} \leftarrow \text{rollout}(u_{0:T-1})$
  `// Backward pass`
  $\tilde{\lambda}_T \leftarrow \nabla r_T(x_T)$
  for $t = T-1, T-2, \ldots, 0$ **do**
    `// Jacobian estimation`
    $z_t \leftarrow [x_t^\top \; u_t^\top \; 1]^\top$
    $Q_t \leftarrow \alpha Q_t + (1-\alpha) q_0 I + z_t z_t^\top$
    $F_t \leftarrow F_t + Q_t^{-1} z_t (x_{t+1} - F_t z_t)^\top$
    $[A_t^\top \; B_t^\top \; c_t] \leftarrow F_t$
    `// Pontryagin update`
    $\tilde{\lambda}_t \leftarrow \nabla_x r(x_t, u_t) + A_t \tilde{\lambda}_{t+1}$
    $g_t \leftarrow \nabla_u r(x_t, u_t) + B_t \tilde{\lambda}_{t+1}$
    $\bar{u}_t \leftarrow \bar{u}_t + \eta g_t$ `// Gradient ascent`
  end
end

B Derivation of Pontryagin's principle

In this section, we will derive Pontryagin's principle, Eqs. (1) to (3), using the method of Lagrange multipliers. In the following we view the objective J as a function of states and actions, that is

$$J(x_{0:T}, u_{0:T-1}) \doteq \sum_{t=0}^{T-1} r(x_t, u_t) + r_T(x_T).$$

We maximize J with respect to $x_{0:T}$ and $u_{0:T-1}$ subject to the constraint that $x_{t+1} = f(x_t, u_t)$ for all $t = [T-1]_0$. The corresponding Lagrangian is

$$L(x_{0:T}, u_{0:T-1}, \lambda_{1:T}) \doteq \sum_{t=0}^{T-1} \{r(x_t, u_t) + \lambda_{t+1}^\top (f(x_t, u_t) - x_{t+1})\} + r_T(x_T),$$

where the constraints are included through the multipliers $\lambda_{1:T}$. The costate equations are then obtained by setting the partial derivatives of the Lagrangian with respect to $x_{0:T}$ to zero:

$$\begin{aligned}
\nabla_{x_t} L &= \nabla_x r(x_t, u_t) + \nabla_x f(x_t, u_t)\lambda_{t+1} - \lambda_t \doteq 0 \\
\implies \quad \lambda_t &= \nabla_x r(x_t, u_t) + \nabla_x f(x_t, u_t)\lambda_{t+1} \\
\nabla_{x_T} L &= \nabla r_T(x_T) - \lambda_T \doteq 0 \\
\implies \quad \lambda_T &= \nabla r_T(x_T).
\end{aligned}$$

Setting the partial derivatives of the Lagrangian with respect to $\lambda_{1:T}$ to zero yields the dynamics equations, and the partial derivatives of the Lagrangian with respect to $u_{0:T-1}$ are

$$\nabla_{u_t} L = \nabla_u r(x_t, u_t) + \nabla_u f(x_t, u_t)\lambda_{t+1},$$

which is the same expression for the gradient of the objective as in Eq. (1).

C Pontryagin's principle from backpropagation

In Section 3.2, we mention that computing the gradient of

$$\begin{aligned}
J(u_{0:T-1}) &= \sum_{t=0}^{T-1} r(x_t, u_t) + r_T(x_T) \\
&= r(x_0, u_0) + r\{\underbrace{f(x_0, u_0)}_{x_1}, u_1\} + r\{\underbrace{f(f(x_0, u_0), u_1)}_{x_2}, u_2\} + \cdots \\
&\qquad + r\{\underbrace{f(f(\cdots f(f(x_0, u_0), u_1)\cdots), u_{T-2})}_{x_{T-1}}, u_{T-1}\} \\
&\qquad\qquad + r_T\{\underbrace{f(f(\cdots f(f(x_0, u_0), u_1)\cdots), u_{T-1})}_{x_T}\}
\end{aligned}$$

by means of the backpropagation algorithm, i.e. a repeated application of the chain rule, leads naturally to Pontryagin's principle. The chain rule states that for $g : \mathbb{R}^n \to \mathbb{R}^k$, $h : \mathbb{R}^k \to \mathbb{R}^m$ and $x \in \mathbb{R}^n$,

$$\nabla(h \circ g)(x) = \nabla g(x)\, \nabla h\{g(x)\},$$

where $\nabla g : \mathbb{R}^n \to \mathbb{R}^{n\times k}$, $\nabla h : \mathbb{R}^k \to \mathbb{R}^{k\times m}$ and $\nabla(h \circ g) : \mathbb{R}^n \to \mathbb{R}^{n\times m}$. From this, we can compute the gradient of the objective function with respect to the action u_t at time $t \in [T-1]_0$ as

$$\begin{aligned}
\nabla_{u_t} J(u_{0:T-1}) &= \nabla_u r(x_t, u_t) + \nabla_u f(x_t, u_t)\nabla_x r(x_{t+1}, u_{t+1}) \\
&\quad + \nabla_u f(x_t, u_t)\nabla_x f(x_{t+1}, u_{t+1})\nabla_x r(x_{t+2}, u_{t+2}) \\
&\quad + \cdots \\
&\quad + \nabla_u f(x_t, u_t)\nabla_x f(x_{t+1}, u_{t+1}) \cdots \nabla_x f(x_{T-2}, u_{T-2})\nabla_x r(x_{T-1}, u_{T-1}) \\
&\quad + \nabla_u f(x_t, u_t)\nabla_x f(x_{t+1}, u_{t+1}) \cdots \nabla_x f(x_{T-1}, u_{T-1})\nabla r_T(x_T) \\
&= \nabla_u r(x_t, u_t) + \nabla_u f(x_t, u_t)\lambda_{t+1},
\end{aligned}$$

where we have introduced the shorthand λ_{t+1} for the blue part. This is the same expression for the gradient as in Eq. (1), and it can easily be seen that this definition of λ_t satisfies the costate equations (2) and (3).

D Proof of Theorem 3.4

In this section, we prove Theorem 3.4, our convergence result of Algorithm 1. The main part of the proof is contained in the proof of Theorem D.3, which provides a lower bound for the inner product between the approximate and true gradients as well as an upper bound for the norm of the approximate gradients. Intuitively, this theorem turns Assumption 3.2, which is a statement about the error of the approximate Jacobians, into a statement about the error of the approximate gradient. We then show that Theorem 3.4 follows by making use of the L-smoothness (Assumption 3.3) of the objective function. This latter part is a standard result in the analysis of stochastic gradient methods [e.g., Bottou et al., 2018].

Before coming to the main result, we introduce the following shorthand notation. Given a fixed trajectory $(u_{0:T-1}, x_{0:T})$, we define

$$\nabla J_t \doteq \nabla_u f_t \lambda_{t+1}, \quad \varepsilon_t \doteq A_t - \nabla_x f_t, \quad \varepsilon'_t \doteq B_t - \nabla_u f_t \quad \text{and} \quad \delta_{t+1} \doteq \tilde{\lambda}_{t+1} - \lambda_{t+1}$$

for all times $t \in [T-1]_0$. By Eq. (1) and Assumption 3.1, the first quantity defines the true gradient and the approximate gradient is given by $g_t = B_t\tilde{\lambda}_{t+1}$. We also state two small lemmas, which we will use routinely in the following proof.

Lemma D.1. *Let $x, y \in \mathbb{R}^n$ for $n \in \mathbb{N}$ and $\alpha \in \mathbb{R}$ such that $\|x\| \le \alpha\|y\|$. Then, $|x^\top y| \le \alpha\|y\|^2$.*

Proof. $\|x\| \le \alpha\|y\| \implies \|x\|\|y\| \le \alpha\|y\|^2 \implies |x^\top y| \le \alpha\|y\|^2$ (by Cauchy-Schwarz). □

Lemma D.2. *Let $A, B \in \mathbb{R}^{m\times n}$ for some $m, n \in \mathbb{N}$ and $x, y \in \mathbb{R}^n$ such that $\bar{\sigma}(A)\|x\| \le \underline{\sigma}(B)\|y\|$. Then, $\|Ax\| \le \|By\|$.*

Proof. This is a simple corollary of the Courant-Fischer (min-max) theorem. The min-max theorem states that, for a symmetric matrix $C \in \mathbb{R}^{n\times n}$, the minimum and maximum eigenvalues $\underline{\lambda}(C)$ and $\bar{\lambda}(C)$ are characterized in the following way:

$$\underline{\lambda}(C) = \min_{\substack{z\in\mathbb{R}^n \\ \|z\|=1}} z^\top C z \quad \text{and} \quad \bar{\lambda}(C) = \max_{\substack{z\in\mathbb{R}^n \\ \|z\|=1}} z^\top C z.$$

This can be extended to a characterization of the singular values , $\bar{\sigma}(A)$ and $\underline{\sigma}(B)$ by relating them to the eigenvalues of $A^\top A$ and $B^\top B$, respectively:

$$\bar{\sigma}(A) = \sqrt{\bar{\lambda}(A^\top A)} = \max_{\substack{z\in\mathbb{R}^n \\ \|z\|=1}} \sqrt{z^\top A^\top A z} = \max_{\substack{z\in\mathbb{R}^n \\ \|z\|=1}} \|Az\| \geq \frac{1}{\|x\|}\|Ax\|,$$

$$\underline{\sigma}(B) = \sqrt{\underline{\lambda}(B^\top B)} = \min_{\substack{z\in\mathbb{R}^n \\ \|z\|=1}} \sqrt{z^\top B^\top B z} = \min_{\substack{z\in\mathbb{R}^n \\ \|z\|=1}} \|Bz\| \leq \frac{1}{\|y\|}\|By\|.$$

Combining these inequalities, we get:

$$\|Ax\| \leq \bar{\sigma}(A)\|x\| \leq \underline{\sigma}(B)\|y\| \leq \|By\|.$$

□

Theorem D.3. *Suppose Assumptions 3.1 and 3.2 hold with γ and ζ and define $\mu \doteq 1-\gamma-\zeta-\gamma\zeta$ and $\nu \doteq 1+\gamma+\zeta+\gamma\zeta$. Then,*

$$g_t^\top \nabla J_t \geq \mu\|\nabla J_t\|^2 \quad \textit{and} \quad \|g_t\| \leq \nu\|\nabla J_t\|,$$

for all $t \in [T-1]_0$.

Proof. Let $t \in [T-1]_0$ be fixed. Decomposing the left-hand side of the first inequality, we get

$$\begin{aligned} g_t^\top \nabla J_t &= \tilde{\lambda}_{t+1}^\top B_t^\top \nabla_u f_t \lambda_{t+1} \\ &= (\lambda_{t+1} + \delta_{t+1})^\top (\nabla_u f_t + \varepsilon_t')^\top \nabla_u f_t \lambda_{t+1} \\ &= \|\nabla J_t\|^2 + \underbrace{\lambda_{t+1}^\top \varepsilon_t'^\top \nabla_u f_t \lambda_{t+1}}_{a} + \underbrace{\delta_{t+1}^\top \nabla_u f_t^\top \nabla_u f_t \lambda_{t+1}}_{b} + \underbrace{\delta_{t+1}^\top \varepsilon_t'^\top \nabla_u f_t \lambda_{t+1}}_{c} \\ &\geq \|\nabla J_t\|^2 - |a| - |b| - |c|. \end{aligned}$$

We will now show that

$$|a| \leq \zeta\|\nabla J_t\|^2 \quad \text{and} \quad |b| \leq \gamma\|\nabla J_t\|^2 \quad \text{and} \quad |c| \leq \gamma\zeta\|\nabla J_t\|^2,$$

which, when taken together, will give us

$$g_t^\top \nabla J_t \geq (1-\gamma-\zeta-\gamma\zeta)\|\nabla J_t\|^2 = \mu\|\nabla J_t\|^2.$$

We first derive the bound on $|a|$:

$$\begin{aligned} & & \bar{\sigma}(\varepsilon_t') &\leq \zeta\underline{\sigma}(\nabla_u f_t) && \text{(Assumption 3.2b)} \\ &\implies & \|\varepsilon_t'\lambda_{t+1}\| &\leq \zeta\|\nabla_u f_t \lambda_{t+1}\| && \text{(Lemma D.2)} \\ &\implies & |\underbrace{\lambda_{t+1}^\top \varepsilon_t'^\top \nabla_u f_t \lambda_{t+1}}_{a}| &\leq \zeta\|\nabla J_t\|^2. && \text{(Lemma D.1)} \end{aligned} \tag{8}$$

The expression for b involves δ_{t+1}, which is the error of the approximate costate $\tilde{\lambda}_{t+1}$. This error comes from the cumulative error build-up due to $\varepsilon_{t+1:T-1}$, the errors of the approximate Jacobians used in the backward pass. To bound $|b|$ we therefore first need to bound this error build-up. To this end, we now show that for all $s \in [T-t]$,

$$\|\delta_{t+s}\| \leq \frac{\gamma}{3^{s-1}}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s-1}\kappa^{-1}(\nabla_x f_{t+i})\|\lambda_{t+s}\|, \tag{9}$$

where we write the inverse condition number of a matrix A as $\kappa^{-1}(A) \doteq \underline{\sigma}(A)/\bar{\sigma}(A)$. To prove this bound, we perform a backward induction on s. First, consider $s = T-t$. The right-hand side of Eq. (9) is clearly nonnegative. The left-hand side is

$$\|\delta_T\| = \|\tilde{\lambda}_T - \lambda_T\| = 0,$$

as $\tilde{\lambda}_T = \lambda_T$. Thus, the inequality holds for $s = T - t$. We now complete the induction by showing that it holds for any $s \in [T - t - 1]$, assuming that it holds for $s + 1$. We start by decomposing δ_{t+s}:

$$\begin{aligned}
\delta_{t+s} &= \tilde{\lambda}_{t+s} - \lambda_{t+s} \\
&= A_{t+s}\tilde{\lambda}_{t+s+1} - \nabla_x f_{t+s}\lambda_{t+s+1} \\
&= (\nabla_x f_{t+s} + \varepsilon_{t+s})(\lambda_{t+s+1} + \delta_{t+s+1}) - \nabla_x f_{t+s}\lambda_{t+s+1} \\
&= \varepsilon_{t+s}\lambda_{t+s+1} + \nabla_x f_{t+s}\delta_{t+s+1} + \varepsilon_{t+s}\delta_{t+s+1}.
\end{aligned}$$

Now, we can bound $\|\delta_{t+s}\|$ by bounding these individual contributions:

$$\|\delta_{t+s}\| \leq \|\underbrace{\varepsilon_{t+s}\lambda_{t+s+1}}_{a'}\| + \|\underbrace{\nabla_x f_{t+s}\delta_{t+s+1}}_{b'}\| + \|\underbrace{\varepsilon_{t+s}\delta_{t+s+1}}_{c'}\|.$$

We start with $\|a'\|$:

$$\bar{\sigma}(\varepsilon_{t+s}) \;\leq\; \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s-1}\kappa^{-1}(\nabla_x f_{t+i})\underline{\sigma}(\nabla_x f_{t+s}) \qquad \text{(Assumption 3.2a)}$$

$$\implies \quad \|\underbrace{\varepsilon_{t+s}\lambda_{t+s+1}}_{a'}\| \;\leq\; \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s-1}\kappa^{-1}(\nabla_x f_{t+i})\|\underbrace{\nabla_x f_{t+s}\lambda_{t+s+1}}_{\lambda_{t+s}}\|. \qquad \text{(Lemma D.2)}$$

Now, $\|b'\|$:

$$\|\delta_{t+s+1}\| \;\leq\; \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s}\kappa^{-1}(\nabla_x f_{t+i})\|\lambda_{t+s+1}\|$$

(Induction hypothesis)

$$\iff \quad \bar{\sigma}(\nabla_x f_{t+s})\|\delta_{t+s+1}\| \;\leq\; \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s-1}\kappa^{-1}(\nabla_x f_{t+i})\underline{\sigma}(\nabla_x f_{t+s})\|\lambda_{t+s+1}\|$$

(Definition of κ^{-1})

$$\implies \quad \|\underbrace{\nabla_x f_{t+s}\delta_{t+s+1}}_{b'}\| \;\leq\; \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s-1}\kappa^{-1}(\nabla_x f_{t+i})\|\underbrace{\nabla_x f_{t+s}\lambda_{t+s+1}}_{\lambda_{t+s}}\|.$$

(Lemma D.2)

And finally, $\|c'\|$:

$$\bar{\sigma}(\varepsilon_{t+s}) \;\leq\; \underline{\sigma}(\nabla_x f_{t+s}) \;\leq\; \bar{\sigma}(\nabla_x f_{t+s}) \tag{10}$$

$$\implies \quad \bar{\sigma}(\varepsilon_{t+s})\|\delta_{t+s+1}\| \;\leq\; \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s}\kappa^{-1}(\nabla_x f_{t+i})\bar{\sigma}(\nabla_x f_{t+s})\|\lambda_{t+s+1}\|$$

(Induction hypothesis)

$$\iff \quad \bar{\sigma}(\varepsilon_{t+s})\|\delta_{t+s+1}\| \;\leq\; \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s-1}\kappa^{-1}(\nabla_x f_{t+i})\underline{\sigma}(\nabla_x f_{t+s})\|\lambda_{t+s+1}\|$$

(Definition of κ^{-1})

$$\implies \quad \|\underbrace{\varepsilon_{t+s}\delta_{t+s+1}}_{c'}\| \;\leq\; \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s-1}\kappa^{-1}(\nabla_x f_{t+i})\|\underbrace{\nabla_x f_{t+s}\lambda_{t+s+1}}_{\lambda_{t+s}}\|. \qquad \text{(Lemma D.2)}$$

Here, Eq. (10) follows from Assumption 3.2a by noting that that the constant before $\underline{\sigma}(\nabla_x f_{t+s})$ on the right-hand side is not greater than 1. We can now put all three bounds together to give us Eq. (9):

$$\|\delta_{t+s}\| \leq \|a'\| + \|b'\| + \|c'\| \leq 3 \cdot \frac{\gamma}{3^s}\kappa^{-1}(\nabla_u f_t)\prod_{i=1}^{s-1}\kappa^{-1}(\nabla_x f_{t+i})\|\lambda_{t+s}\|.$$

Equipped with a bound on δ_{t+s}, we are ready to bound $|b|$ and $|c|$. Starting with $|b|$, we have:

$$\|\delta_{t+1}\| \;\leq\; \gamma\kappa^{-1}(\nabla_u f_t)\|\lambda_{t+1}\| \qquad \text{(Eq. (9) for } s = 1)$$

$$
\begin{aligned}
\iff \quad & \bar{\sigma}(\nabla_u f_t)\|\delta_{t+1}\| \le \gamma \underline{\sigma}(\nabla_u f_t)\|\lambda_{t+1}\| && \text{(Definition of } \kappa^{-1}) \\
\implies \quad & \|\nabla_u f_t \delta_{t+1}\| \le \gamma \|\nabla_u f_t \lambda_{t+1}\| && \text{(Lemma D.2)} \\
\implies \quad & |\underbrace{\delta_{t+1}^\top \nabla_u f_t^\top \nabla_u f_t \lambda_{t+1}}_{b}| \le \gamma \|\nabla J_t\|^2. && \text{(Lemma D.1)}
\end{aligned} \tag{11}
$$

And finally, we can bound $|c|$:

$$
\begin{aligned}
& \bar{\sigma}(\varepsilon_t') \le \zeta \underline{\sigma}(\nabla_u f_t) \le \zeta \bar{\sigma}(\nabla_u f_t) && \text{(Assumption 3.2b)} \\
\implies \quad & \bar{\sigma}(\varepsilon_t')\kappa^{-1}(\nabla_u f_t)\|\lambda_{t+1}\| \le \zeta \underline{\sigma}(\nabla_u f_t)\|\lambda_{t+1}\| && \text{(Definition of } \kappa^{-1}) \\
\implies \quad & \bar{\sigma}(\varepsilon_t')\|\delta_{t+1}\| \le \gamma\zeta \underline{\sigma}(\nabla_u f_t)\|\lambda_{t+1}\| && \text{(Eq. (9) for } s = 1) \\
\implies \quad & \|\varepsilon_t' \delta_{t+1}\| \le \gamma\zeta \|\nabla_u f_t \lambda_{t+1}\| && \text{(Lemma D.2)} \\
\implies \quad & |\underbrace{\delta_{t+1}^\top \varepsilon_t'^\top \nabla_u f_t \lambda_{t+1}}_{c}| \le \gamma\zeta \|\nabla J_t\|^2. && \text{(Lemma D.1)}
\end{aligned} \tag{12}
$$

This concludes the proof of the first inequality showing that

$$
g_t^\top \nabla J_t \ge \mu \|\nabla J_t\|^2.
$$

The second inequality,

$$
\|g_t\| \le \nu \|\nabla J_t\|,
$$

follows easily from the work we have already done. To show this, we start by decomposing g_t:

$$
\begin{aligned}
g_t &= B_t \tilde{\lambda}_{t+1} \\
&= (\nabla_u f_t + \varepsilon_t')(\lambda_{t+1} + \delta_{t+1}) \\
&= \nabla J_t + \nabla_u f_t \delta_{t+1} + \varepsilon_t' \lambda_{t+1} + \varepsilon_t' \delta_{t+1}.
\end{aligned}
$$

To bound the norm of g_t, we again make use of the triangle inequality:

$$
\begin{aligned}
\|g_t\| &\le \|\nabla J_t\| + \|\nabla_u f_t \delta_{t+1}\| + \|\varepsilon_t' \lambda_{t+1}\| + \|\varepsilon_t' \delta_{t+1}\| \\
&\le (1 + \gamma + \zeta + \gamma\zeta)\|\nabla J_t\| \\
&= \nu \|\nabla J_t\|,
\end{aligned}
$$

where we have used Eqs. (8), (11) and (12). □

Proof of Theorem 3.4. Let $N \in \mathbb{N}$ and $t \in [T-1]_0$ be fixed. In Algorithm A.1, the iterates are computed, for all $k \in [N-1]_0$, as

$$
u_t^{(k+1)} = u_t^{(k)} + \eta g_t^{(k)},
$$

where $g_t^{(k)}$ is the approximate gradient at iteration k. We denote the true gradient at iteration k by $\nabla J_t^{(k)}$. From the L-smoothness of the objective function (Assumption 3.3), it follows that

$$
\begin{aligned}
J(u_{0:T-1}^{(k+1)}) &\ge J(u_{0:T-1}^{(k)}) + \nabla_{u_t} J(u_{0:T-1}^{(k)})^\top (u_t^{(k+1)} - u_t^{(k)}) - \frac{L}{2}\|u_t^{(k+1)} - u_t^{(k)}\|^2 \\
&= J(u_{0:T-1}^{(k)}) + \nabla J_t^{(k)\top}(\eta g_t^{(k)}) - \frac{L}{2}\|\eta g_t^{(k)}\|^2 \\
&\ge J(u_{0:T-1}^{(k)}) + \eta\mu\|\nabla J_t^{(k)}\|^2 - \frac{\eta^2 L \nu^2}{2}\|\nabla J_t^{(k)}\|^2 && \text{(Theorem D.3)} \\
&= J(u_{0:T-1}^{(k)}) + \eta\Big(\underbrace{\mu - \frac{\eta L \nu^2}{2}}_{\alpha}\Big)\|\nabla J_t^{(k)}\|^2.
\end{aligned}
$$

Theorem 3.4 demands that $\eta > 0$ is set small enough such that $\alpha > 0$, which is possible because $0 < \mu < \nu$ and $L > 0$. Thus, we get

$$
\begin{aligned}
\eta\alpha\|\nabla J_t^{(k)}\|^2 &\le J(u_{0:T-1}^{(k+1)}) - J(u_{0:T-1}^{(k)}) \\
\implies \quad \frac{1}{N}\sum_{k=0}^{N-1}\|\nabla J_t^{(k)}\|^2 &\le \frac{1}{\alpha\eta N}\sum_{k=0}^{N-1}\Big\{J(u_{0:T-1}^{(k+1)}) - J(u_{0:T-1}^{(k)})\Big\}
\end{aligned}
$$

$$= \frac{1}{\alpha\eta N}\left\{J(u_{0:T-1}^{(N)}) - J(u_{0:T-1}^{(0)})\right\}$$
$$\leq \frac{J^\star - J(u_{0:T-1}^{(0)})}{\alpha\eta N},$$

where $J^\star \doteq \sup_{u\in\mathcal{U}^T} J(u)$ is the optimal value of the initial state. □

E Further experiments

In this section, we repeat the MuJoCo experiments of Section 4.2 with longer time-horizons T. The results are shown in Fig. 8. Our algorithms are sensitive to the horizon T, as the costates, which are propagated backward in time, attain some error at each propagation step due to the approximation errors of the Jacobians. For this reason, Theorem 3.4 demands (through Assumption 3.2) more accurate Jacobians at later time steps. Thus, for large T, our convergence result requires more accurate Jacobian estimates. However, in Fig. 8, we see that Algorithm A.3 is able to cope with longer horizons for the two MuJoCo environments. The reason for this discrepancy between our theoretical and empirical result is that Theorem 3.4 does not consider the stability of the system under consideration. The two MuJoCo systems, `Ant-v4` and `HalfCheetah-v4`, are stable along the trajectories encountered during training, which prevents an exponential build-up of error in the costate propagation.

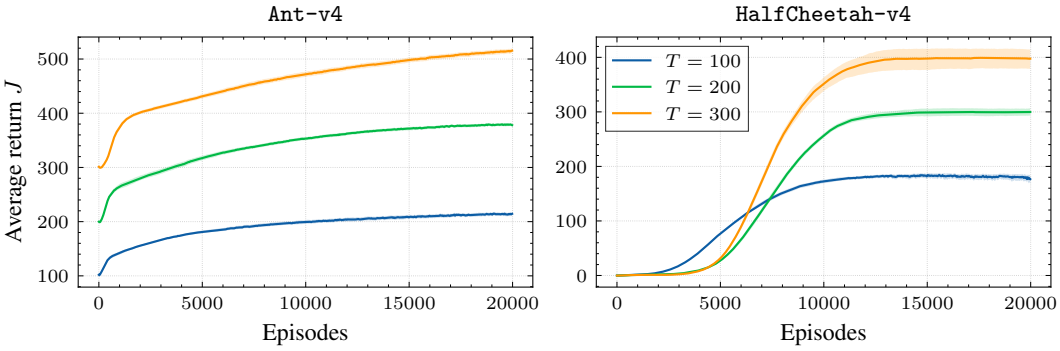

Figure 8: Learning curves of our off-trajectory algorithm for the two MuJoCo tasks. All experiments were repeated with 20 random seeds, and we show 95%-bootstrap confidence intervals for the average return.

F Baselines

We compare our algorithms against two baselines: the finite-difference approach discussed at the end of Section 3.3 and the gradient-free cross-entropy method [CEM; Rubinstein, 1999]. These methods are listed in Algorithms F.1 and F.2.

Similarly to the finite-difference method and our on-trajectory algorithm, in CEM, we also perform $M \in \mathbb{N}$ rollouts of perturbed action sequences $\{\boldsymbol{u}_i \sim \mathcal{N}(\bar{\boldsymbol{u}}, \sigma I)\}_{i=1}^M$. Here, $\bar{\boldsymbol{u}}$ is the current action sequence and $\sigma > 0$ is a noise scale parameter. We then construct the *elite* set S of the $L < M$ perturbed action sequences with the highest returns, where $L \in \mathbb{N}$ is a hyperparameter. Finally, the current action sequence $\bar{\boldsymbol{u}}$ is updated to be the mean of the elite sequences, such that $\bar{\boldsymbol{u}} \leftarrow \frac{1}{L}\sum_{\boldsymbol{u}\in S} \boldsymbol{u}$.

This method can be considerably more efficient than the simple finite-difference method. Here, we are not trying to estimate the gradient anymore, so we can potentially improve the action sequence with far fewer rollouts than would be needed in the finite-difference approach. However, this method suffers from the same fundamental deficiency as the finite-difference method: it ignores the Markovian structure of the RL problem and treats the objective function J as a black box. CEM is commonly used in model-based closed-loop reinforcement learning for planning. In this setting, the rollouts are hallucinated using the approximate model. Instead of executing the complete open-loop trajectory, the model-predictive control framework is typically employed. The planning procedure is repeated after each step in the real environment with the executed action being the first item in the planned

action sequence. Thus, this setting is very different from our open-loop RL objective. For this reason, we slightly modify the CEM algorithm to better fit our requirements. In model-based RL, typically both mean $\bar{u}$ and standard variation σ are adapted in CEM [Hafner et al., 2019, Pinneri et al., 2021]. In our experiments, this approach led to very fast convergence ($\sigma \to 0$) to suboptimal trajectories. We thus only fit the mean and keep the noise scale fixed, which we empirically observed to give much better results.

Algorithm F.1: Finite-difference method

Input: Number of rollouts $M \in \mathbb{N}$, noise scale $\sigma > 0$, step size $\eta > 0$

Initialize $\bar{u}_{0:T-1}$ (initial action sequence)
$\bar{\boldsymbol{u}} \leftarrow \mathrm{vec}(\bar{u}_{0:T-1}) \in \mathbb{R}^{TK}$
for $k = 1, 2, \ldots, N$ **do**
  `// Forward passes`
  $\bar{x}_{0:T} \leftarrow \mathrm{rollout}(\bar{u}_{0:T-1})$
  $\bar{J} \leftarrow \sum_{t=0}^{T-1} r(\bar{x}_t, \bar{u}_t) + r_T(\bar{x}_T)$
  for $i = 1, 2, \ldots, M$ **do**
    $u_{0:T-1} \sim \mathcal{N}(\bar{u}_{0:T-1}, \sigma I)$
    $x_{0:T} \leftarrow \mathrm{rollout}(u_{0:T-1})$
    $\boldsymbol{u}_i \leftarrow \mathrm{vec}(u_{0:T-1}) \in \mathbb{R}^{TK}$
    $J_i \leftarrow \sum_{t=0}^{T-1} r(x_t, u_t) + r_T(x_T)$
  end
  `// Gradient estimation`
  $\boldsymbol{g} \leftarrow \arg\min_{\boldsymbol{g} \in \mathbb{R}^{TK}} \sum_{i=1}^{M} \{J_i - \bar{J} - \boldsymbol{g}^\top (\boldsymbol{u}_i - \bar{\boldsymbol{u}})\}^2$
  `// Gradient ascent`
  $\bar{\boldsymbol{u}} \leftarrow \bar{\boldsymbol{u}} + \eta \boldsymbol{g}$
  $\bar{u}_{0:T-1} \leftarrow \mathrm{reshape}(\bar{\boldsymbol{u}}) \in \mathbb{R}^{T \times K}$
end

Algorithm F.2: Cross-entropy method

Input: Number of rollouts $M \in \mathbb{N}$, noise scale $\sigma > 0$, size of elite set $L \in \mathbb{N}$

Initialize $\bar{u}_{0:T-1}$ (initial action sequence)
$\bar{\boldsymbol{u}} \leftarrow \mathrm{vec}(\bar{u}_{0:T-1}) \in \mathbb{R}^{TK}$
for $k = 1, 2, \ldots, N$ **do**
  `// Forward passes`
  for $i = 1, 2, \ldots, M$ **do**
    $u_{0:T-1} \sim \mathcal{N}(\bar{u}_{0:T-1}, \sigma I)$
    $x_{0:T} \leftarrow \mathrm{rollout}(u_{0:T-1})$
    $\boldsymbol{u}_i \leftarrow \mathrm{vec}(u_{0:T-1}) \in \mathbb{R}^{TK}$
    $J_i \leftarrow \sum_{t=0}^{T-1} r(x_t, u_t) + r_T(x_T)$
  end
  `// Elite set computation`
  $S \leftarrow \arg\mathrm{partition}_L \{(-J_i)_{i=1}^{M}\}_{1:L}$
  `// Action sequence update`
  $\bar{\boldsymbol{u}} \leftarrow \frac{1}{L} \sum_{i \in S} \boldsymbol{u}_i$
  $\bar{u}_{0:T-1} \leftarrow \mathrm{reshape}(\bar{\boldsymbol{u}}) \in \mathbb{R}^{T \times K}$
end

G Hyperparameters

Unless stated otherwise, we used the hyperparameters listed in Table 1 in the inverted penulum experiments of Section 4.1, and those listed in Table 2 in the MuJoCo experiments of Section 4.2

Parameter	Value
Number of optimization steps N	50000
Step size η	0.001
Noise scale σ	0.001
Number of perturbed rollouts M	10
Forgetting factor α	0.8
Initial precision q_0	0.001
Cross-entropy method: M[5]	20
Finite-difference method: M[5]	20
Finite-difference method: σ[5]	0.0001
MLP model: hidden layers	$[16, 16]$
MLP model: training rollouts	1000
MLP model training: epochs	10
MLP model training: batch size	100
MLP model training: step size	0.002
MLP model training: weight decay	0.001

Table 1: Pendulum experiments hyperparameters

Parameter	Value
Number of optimization steps N	20000
Step size η	0.0001
Noise scale σ	0.03
Initial precision q_0	0.0001
Forgetting factor α	
`HalfCheetah-v4`, $T = 100$	0.9
`HalfCheetah-v4`, $T = 200$	0.8
`HalfCheetah-v4`, $T = 300$	0.8
`Ant-v4`, $T = 100$	0.95
`Ant-v4`, $T = 200$	0.9
`Ant-v4`, $T = 300$	0.85

Table 2: MuJoCo experiments hyperparameters

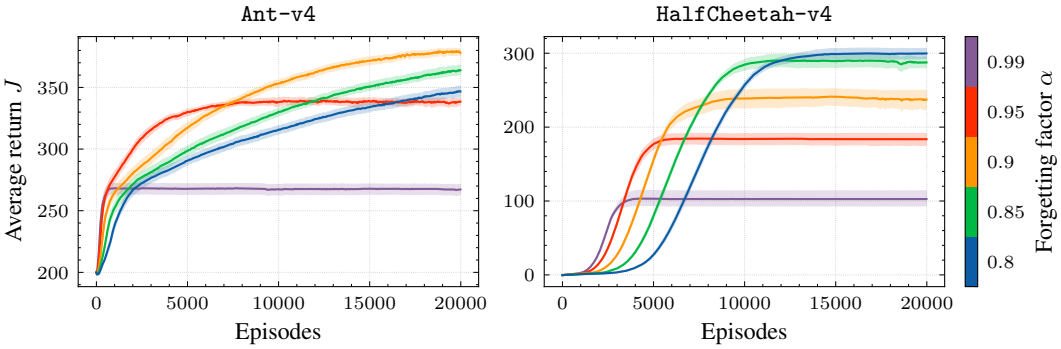

Figure 9: Analysis of the influence of the forgetting factor α on the performance of the off-trajectory method (Algorithm A.3) in the MuJoCo environments ($T = 200$). All experiments were repeated with 20 random seeds, and we show 95%-bootstrap confidence intervals for the average return.

and Appendix E. In each experiment, all actions in the initial action trajectory $u^{(0)}_{0:T-1}$ are sampled from a zero-mean Gaussian distribution with standard deviation 0.01. We use the Adam optimizer [Kingma and Ba, 2014] both for training the MLP model and for performing the gradient ascent steps in Algorithms A.1 to A.3 and F.1. We did not optimize the hyperparameters of soft actor-critic (SAC), but kept the default values suggested by Haarnoja et al. [2018a], as these are already optimized for the MuJoCo environments. The entropy coefficient of the SAC algorithm is tuned automatically according to the procedure described by Haarnoja et al. [2018b]. In our experiments, we make use of the Stable-Baselines3 [Raffin et al., 2021] implementation of SAC.

For our off-trajectory method, we found it worthwile to tune the forgetting factor α to the specific task at hand. Large α means that data is retained for longer, which both makes the algorithm more sample efficient (i.e., faster convergece) and the Jacobian estimates more biased (i.e., convergence to a worse solution). In Fig. 9, we show this trade-off in the learning curves for the MuJoCo tasks (with the horizon $T = 200$). We found that the performance is much less senstitive to the choice of noise scale σ and initial precision q_0 than to the choice of the forgetting factor α.

[5] This value was chosen on the basis of the experiment presented in Fig. 5.