Stochastic Control and Machine Learning Project - SciPhy RL for DOCTR-L

Constantin GLEYZE, Aurélien PEREZ, Léa MARINGER

Abstract

We present the DEEP DOCTR-L method introduced by Igor Halperin in Distributional Offline Continuous-Time Reinforcement Learning with Neural Physics-Informed PDEs (SciPhy RL for DOCTR-L). We apply it in the linear-quadratic (LQ) framework with a behavioral policy modeled as a Gaussian mixture. We describe its theoretical formulation and examine how it behaves by comparing it to a benchmark case with a semi-closed-form solution.

Introduction

Reinforcement Learning (RL) provides a framework for solving sequential decision-making problems in stochastic environments. Traditional RL methods typically rely on online interactions with the environment, allowing the agent to actively explore and refine its policy. However, in many practical settings, such exploration is either costly, risky, or infeasible. Offline RL, also known as batch-mode RL, addresses this challenge by learning solely from a fixed dataset of past experiences, collected under an unknown behavioral policy. While attractive, this paradigm introduces significant difficulties. Unlike online RL, where the agent can correct errors through trial and error, offline RL requires the model to generalize purely from observed data. This can lead to extrapolation errors, where poor generalization to unseen state-action pairs results in overestimation or underestimation of policy values.

A key complication in offline RL is that the available data is not necessarily optimal. Unlike imitation learning, where data comes from expert demonstrations, offline RL must learn from suboptimal or even adversarial policies. This means that naive value estimation can lead to highly misleading results, as there is no guarantee that observed actions were close to optimal. Many methods attempt to mitigate this issue by constraining the learned policy to remain close to the behavioral data, thereby limiting reliance on potentially unreliable extrapolations.

To address these challenges, Halperin (2023) proposes an approach based on partial differential equations (PDEs), casting the offline RL problem into a continuous-time optimal control framework. Unlike standard methods relying on Bellman iterations, this formulation learns directly from offline data by solving a neural PDE, reducing policy learning to a supervised learning problem. This PDE-based regularization helps to stabilize the optimization process, preventing some of the pitfalls of naive function approximation in high dimensions.

A particular feature of this approach is that it optimizes a transformation of the cost distribution rather than its expectation alone. Instead of directly minimizing the expected cumulative cost Z_T, the method minimizes $E[U(Z_T)]$ for a given function U, which can introduce sensitivity to the distribution's shape. Depending on the choice of U, this can lead to different forms of risk preferences, with convex choices penalizing high variability in returns.

This report presents and examines the DEEP DOCTRL method in the linear-quadratic (LQ) setting, where the behavioral policy is modeled as a Gaussian mixture. We describe its theoretical formulation and investigate its behavior by comparing it to a benchmark case with a semi-closed-form solution.

1 Stochastic Control under random policy

In the following subsections, we introduce the mathematical formulation of the problem (1.1), the cost function to be optimized (1.2), and then the derivation of the soft HJB equation used in this paper (1.3).

1.1 Mathematical formulation

We consider a stochastic control problem where the state process $(X_t)_{t\geq 0}$ takes values in $\mathcal{X} \subset \mathbb{R}^d$ and the control process $(\alpha_t)_{t\geq 0}$ takes values in $A \subset \mathbb{R}^m$. The state evolves according to the controlled stochastic differential equation (SDE):

$$dX_t = \mu(X_t, \alpha_t)dt + \sigma(X_t, \alpha_t)dW_t, \tag{1}$$

where:

- $\mu : \mathcal{X} \times A \to \mathcal{X}$ is the drift function,
- $\sigma : \mathcal{X} \times A \to \mathbb{R}^{d\times k}$ is the diffusion matrix,
- $(W_t)_{t\geq 0}$ is a k-dimensional standard Brownian motion.

The control process $\alpha = (\alpha_t)_t$ is governed by a stochastic policy π, which specifies a probability distribution over actions at each time and state:

$$\pi : [0, T] \times \mathcal{X} \to \mathcal{P}(A). \tag{2}$$

This policy admits a density function $a \mapsto \pi(t, x, a)$ with respect to a reference measure ν on A, leading to the notation:

$$\pi(t, x)(da) = \pi(t, x, a)\nu(da). \tag{3}$$

At each time t, the control α_t is sampled from $\pi(t, X_t)$, independently of the Brownian motion W driving the system. To ensure that α_t remains adapted to the filtration $\mathbb{F}$, the probability space is extended by introducing an independent sequence of i.i.d. uniform random variables $(\mathcal{U}_t)_t$, leading to:

$$\mathbb{F} = (\mathcal{F}_t)_{t\geq 0}, \quad \text{where} \quad \mathcal{F}_t = \mathcal{F}_t^W \vee \sigma(\mathcal{U}_s, s \leq t). \tag{4}$$

This formulation allows for randomized control policies, which are useful for exploration and robust decision-making in stochastic optimization problems.

1.2 Cost Function

For a given stochastic policy π, we define the running cost function $f : \mathcal{X} \times A \to \mathbb{R}$, which represents the instantaneous cost incurred at state X_s when taking action α_s.

The accumulated cost from time 0 to the final horizon T, as viewed from time t, is then given by:

$$Z_T(t) = \int_0^T e^{-r(s-t)} f(X_s, \alpha_s)ds. \tag{5}$$

The objective is to find an optimal policy π^* that minimizes a transformed cost function. The function $U : \mathbb{R} \to \mathbb{R}$ applies a transformation to the accumulated cost, allowing for different risk preferences. A convex choice of U penalizes high costs more severely, introducing a form of risk aversion.

Given a stochastic policy π, the expected transformed cost is defined as:

$$J_0^\pi(t,x,c) = \mathbb{E}\left[U(Z_T(t)) \mid X_t = x, C_t = c\right], \tag{6}$$

where C_t represents the cost accumulated up to time t as viewed from time t.

By definition, for any $s > t$, the accumulated cost satisfies the identity:

$$Z_T(t) = e^{-r(s-t)} Z_T(s). \tag{7}$$

Applying this to the objective function leads to:

$$J_0^\pi(t,x,c) = \mathbb{E}\left[U(e^{-r(s-t)} Z_T(s)) \mid X_t = x, C_t = c\right]. \tag{8}$$

In the original article, Equation (12) states that:

$$\begin{aligned} J_0^\pi(t,x,c) &= e^{-r(s-t)} \mathbb{E}[U(Z_T(s)) \mid X_t = x, C_t = c] \\ &= e^{-r(s-t)} \mathbb{E}[J_0^\pi(s, X_s, C_s) \mid X_t = x, C_t = c]. \end{aligned} \tag{9}$$

However, this equality does not hold in general. It is only valid under specific conditions:

- If U is homogeneous of degree 1, which effectively reduces the problem to a risk-neutral setting and removes the distributional aspect.
- If $r = 0$, which eliminates the discount factor and validates the equality.

The original derivation of Equation (12) starts from Equation (11), where a scaling factor appears to be missing in the density of $Z_T(s)$, leading to the erroneous equality. Throughout this work, we choose to assume that at least one of the above conditions holds, thus ensuring the validity of Equation (12) and enabling the derivation of the Soft HJB equation.

In addition to minimizing the transformed cost, a regularization term is introduced to control the deviation of the learned policy from a given behavioral policy π_0. Specifically, we augment the objective function with the following regularization:

$$\mathcal{R}^\pi(t,x) = \mathbb{E}\left[\int_t^T e^{-r(s-t)} KL(\pi||\pi_0)(s, X_s)\, ds \mid X_t = x\right], \tag{10}$$

where the Kullback-Leibler (KL) divergence is given by:

$$KL(\pi||\pi_0)(t,x) = \int_A \log\left(\frac{\pi(t,x,a)}{\pi_0(t,x,a)}\right) \pi(t,x,a)\nu(da). \tag{11}$$

This term measures the discrepancy between the candidate policy π and the behavioral policy π_0. This regularization is crucial in the offline RL context, as it prevents the optimized policy from excessively diverging from observed actions, thereby limiting reliance on potentially unreliable extrapolations.

This form of regularization generalizes the entropy-regularized RL methods commonly used in standard (online) RL settings, where one typically penalizes deviations from a uniform policy to promote exploration. Indeed, setting π_0 as the uniform distribution on the action space leads to the classical entropy-regularized RL formulation. Here, by replacing the uniform reference policy with a behavioral policy π_0 reflecting previously collected data, the regularization naturally enforces proximity to available observations rather than promoting exploration, which is more suited to the offline RL setting.

The objective is to determine the value function, defined as the minimal cost over all admissible policies:

$$J(t,x,c) = \inf_\pi J^\pi(t,x,c), \quad \text{with} \quad J^\pi(t,x,c) = J_0^\pi(t,x,c) + \lambda \mathcal{R}^\pi(t,x). \tag{12}$$

1.3 Soft Hamilton-Jacobi-Bellman (HJB) equation

Using the established relation for J_0^π (Eq. 9) and applying Itô's lemma to both J_0^π and $\mathcal{R}^\pi$, one obtains an integro-differential equation of the form:

$$\frac{\partial J^\pi}{\partial t} \;+\; \int_A H\big(x,a,\;\nabla_x J^\pi,\;D_x^2 J^\pi\big)\;\pi(t,x,a)\;\nu(da) \;+\; rc\;\frac{\partial J^\pi}{\partial c} \;-\; r\,J^\pi \;+\; \lambda\,KL(\pi\|\pi_0)(t,x) \;=\; 0, \quad (13)$$

where H is the usual Hamiltonian, *i.e.*

$$H(x,a,z,\gamma) = f(x,a) + \mu(x,a)\cdot z + \tfrac{1}{2}\,\mathrm{Tr}(\sigma\sigma^T\gamma). \quad (14)$$

Using the Bellman optimality principle, J satisfies the same PDE but with an infimum over π:

$$\frac{\partial J}{\partial t} \;-\; r\,J \;+\; r\,c\,\frac{\partial J}{\partial C} \;+\; \inf_{\pi\in\mathcal{P}(A)}\Big\{\int_A \Big(H\big(x,a,\nabla_x J, D_x^2 J\big) \;+\; \lambda\,\log\Big(\frac{\pi(a)}{\pi_0(a)}\Big)\,\pi(a)\Big)\,\pi(da)\Big\} \;=\; 0. \quad (15)$$

Carrying out the pointwise minimization in π (by setting the functional derivative to zero) yields:

$$\pi^*(t,x,a) \;\propto\; \pi_0(t,x,a)\,\exp\Big(-\tfrac{1}{\lambda}\;H\big(x,a,\nabla_x J, D_x^2 J\big)\Big). \quad (16)$$

Substituting π^* back into the PDE for J leads to the *Soft HJB equation*:

$$\frac{\partial J}{\partial t} \;-\; r\,J \;+\; r\,c\;\frac{\partial J}{\partial C} \;-\; \lambda\,\log\int_A \pi_0(t,x,a)\,\exp\Big(-\tfrac{1}{\lambda}\;H\big(x,a,\nabla_x J,\;D_x^2 J\big)\Big)\,\nu(da) \;=\; 0, \quad (17)$$

which encodes both optimality (via H) and proximity to the behavioral policy (via π_0).

2 Deep DOCTR-L in the Linear-Quadratic (LQ) Setting with Gaussian Mixture behavioural policy

2.1 LQ Formulation

We now specialize the problem to a linear-quadratic (LQ) setting where the state dynamics follow a controlled linear system:

$$dX_t = (\mu_0(X_t) + \mu_1(X_t)\alpha_t)dt + \sigma(X_t)dW_t, \quad (18)$$

where $\mu_0 : \mathcal{X} \to \mathbb{R}^d$ is the uncontrolled drift, $\mu_1 : \mathcal{X} \to \mathbb{R}^{d\times m}$ represents the control influence, and $\sigma : \mathcal{X} \to \mathbb{R}^{d\times k}$ is the diffusion matrix.
The running cost function is given by:

$$f(X_t,\alpha_t) = f_0(X_t) + \frac{1}{2} f_1(X_t)\alpha_t^T\alpha_t. \quad (19)$$

where:

- $f_0(x) \in \mathbb{R}$ is the state running cost.
- $f_1(x) \in \mathbb{R}$ is a weight for the control cost.

2.2 Behavioral Policy: Gaussian Mixture

We assume that the behavioral policy π_0 is a Gaussian mixture model:

$$\pi_0(t,x,a) = \sum_k \omega_k \phi_k(t,x,a), \tag{20}$$

where:

- ω_k are the mixture weights ($\sum_k \omega_k = 1$),
- $\phi_k(t,x,\cdot)$ is the density of a Gaussian component with mean $u_k(t,x)$ and diagonal covariance $\sigma_k^2 I_m$:

$$\phi_k(t,x)(a) = \frac{1}{(2\pi\sigma_k^2)^{m/2}} \exp\left(-\frac{1}{2\sigma_k^2}\|a - u_k(t,x)\|^2\right). \tag{21}$$

2.3 Optimal Policy as a Gaussian Mixture

Substituting LQ cost and dynamics and the behavioral policy into the expression for the optimal policy (Eq. 16), we obtain an optimal policy that is also a Gaussian mixture:

$$\pi^*(t,x,a) = \sum_k \omega_k[J]\phi_k[J](t,x,a), \tag{22}$$

where the updated weights and means are given by:

$$\omega_k[J] = \frac{\omega_k e^{-\frac{1}{\lambda}\mathcal{H}_k[J]}}{\sum_k \omega_k e^{-\frac{1}{\lambda}\mathcal{H}_k[J]}}, \quad u_k[J] = u_k(t,x) - \frac{\frac{1}{\lambda}\sigma_k^2 \mu_1(x)\cdot\nabla_x J}{1 + \frac{1}{\lambda}\sigma_k^2 f_1(x)\frac{\partial J}{\partial C}}. \tag{23}$$

where $\mathcal{H}_k[J]$ is given by:

$$\mathcal{H}_k[J] = \frac{1}{2}\left(\frac{u_k^T u_k f_1(x)\frac{\partial J}{\partial C}}{1 + \frac{1}{\lambda}\sigma_k^2 f_1(x)\frac{\partial J}{\partial C}} + u_k^T \mu_1 \cdot \nabla_x J - \frac{1}{\lambda}\sigma_k^2 (\mu_1 \cdot \nabla_x J)^2 + \lambda \log\left(1 + \frac{1}{\lambda}\sigma_k^2 f_1(x)\frac{\partial J}{\partial C}\right)\right). \tag{24}$$

2.4 Soft HJB in the LQ Setting

Injecting these results into the Soft HJB equation, we obtain:

$$\frac{\partial J}{\partial t} - rJ + (f_0(x) + rc)\frac{\partial J}{\partial C} + \mu_0(x)\cdot\nabla_x J + \frac{1}{2}\mathrm{Tr}(\sigma\sigma^T(x)\mathcal{D}_x^2 J) - \lambda \log\left(\sum_k \omega_k e^{-\frac{1}{\lambda}\mathcal{H}_k[J]}\right) = 0. \tag{25}$$

2.5 Hamilton-Jacobi (HJ) Equation

The Soft HJB equation provides a probabilistic formulation of the value function's evolution. However, in this form, both locality and causality of the system dynamics remain implicit, which can make optimization unstable. Instead, an alternative path-wise representation makes these aspects explicit, providing a more structured signal for learning.

This is achieved by expressing the evolution of J directly along sampled trajectories, enforcing consistency between successive time steps. With this formulation, the value function J can be parameterized by a neural network, where its gradients serve as constraints in training. This transformation enables learning in a supervised fashion.

Applying Itô's lemma to J and injecting the Soft HJB equation (Eq. 25), we obtain:

$$dJ = \left(rJ + \lambda \log\left(\sum_k \omega_k e^{-\frac{1}{\lambda}\mathcal{H}_k[J]}\right) + \mu_1(X_t)\mathbb{E}[\alpha_t]\cdot\nabla_x J\right)dt + \nabla_x J \cdot \sigma(X_t)dW_t. \tag{26}$$

In the original derivation of this equation (Halperin, 2023), the term $\mu_1(X_t)\mathbb{E}[\alpha_t]$ is missing.
Since we aim to learn from data, we rewrite this equation in terms of dX_t:

$$dJ = \left(rJ + \lambda \log \left(\sum_k \omega_k e^{-\frac{1}{\lambda}\mathcal{H}_k[J]} \right) - \mu_0(X_t) \cdot \nabla_x J \right) dt + \nabla_x J \cdot dX_t. \tag{27}$$

We rewrite the equation in its final form as:

$$dJ = \mathcal{H}_{\text{HJ}}(X_t, \dot{X}_t, J)dt, \tag{28}$$

where $\mathcal{H}_{\text{HJ}}$ represents the effective Hamiltonian governing the evolution of J along observed trajectories:

$$\mathcal{H}_{\text{HJ}}(X_t, \dot{X}_t, J) = rJ + \lambda \log \left(\sum_k \omega_k e^{-\frac{1}{\lambda}\mathcal{H}_k[J]} \right) + (\dot{X}_t - \mu_0(X_t)) \cdot \nabla_x J. \tag{29}$$

2.6 Learning the Value Function via Neural Networks

Suppose we have access to trajectory data (t, X_t, C_t) generated under the optimal policy. In this case, the true value function J satisfies the Hamilton-Jacobi equation along these optimal trajectories. To approximate J, we use a neural network J_θ, parameterized by θ, and enforce the following regression for each observed transition $(X_t, X_{t+\Delta t}, C_t, C_{t+\Delta t})$:

$$\Delta J_\theta(t, X_t, C_t) = \mathcal{H}_{\text{HJ}}(X_t, X_{t+\Delta t}, J_\theta)\Delta t + \varepsilon_t, \tag{30}$$

where $\varepsilon_t \sim \mathcal{N}(0, \nu^2)$ accounts for model uncertainty.

This means that for each observed state transition $X_t \to X_{t+\Delta t}$, we train the neural network so that the predicted difference

$$J_\theta(t + \Delta t, X_{t+\Delta t}, C_{t+\Delta t}) - J_\theta(t, X_t, C_t) \tag{31}$$

matches the evolution predicted by the Hamiltonian $\mathcal{H}_{\text{HJ}}$.

The likelihood function for a dataset of N such trajectories is proportional to:

$$\prod_{n=1}^{N} \prod_{t=0}^{T-1} P^{(J)}(X_{t+\Delta t}^{(n)} | X_t^{(n)}) \exp\left(-\frac{1}{2\nu^2} \left(\Delta J_\theta(t^{(n)}, X_t^{(n)}, C_t^{(n)}) - \mathcal{H}_{\text{HJ}}(X_t^{(n)}, X_{t+\Delta t}^{(n)}, J_\theta)\Delta t \right)^2 \right), \tag{32}$$

where $P^{(J)}(X_{t+\Delta t}|X_t)$ represents the transition probability under the optimal policy.

In the context of offline RL, we do not have access to data generated under the optimal policy. Instead, we observe trajectory data (t, X_t, C_t) collected under a behavioral policy π_0. Since the Hamilton-Jacobi equation is satisfied under the optimal policy, we must adjust for the mismatch between the behavioral and optimal distributions. This is done through a likelihood ratio correction, similar to importance sampling. To incorporate this correction, we introduce a change of measure using the quantity:

$$\Delta S(X_t, X_{t+\Delta t}, J_\theta) := \log \left(\frac{P^{(0)}(X_{t+\Delta t}|X_t)}{P^{(J)}(X_{t+\Delta t}|X_t)} \right), \tag{33}$$

where $P^{(0)}(X_{t+\Delta t}|X_t)$ represents the transition probability under the behavioral policy.
Using this reweighting, we define the loss function for training the neural network which is proportional to the negative log-likelihood:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T-1}\left(\frac{1}{2}\left(\Delta J_\theta(t^{(n)}, X_t^{(n)}, C_t^{(n)}) - \mathcal{H}_{\mathrm{HJ}}(X_t^{(n)}, X_{t+\Delta t}^{(n)}, J_\theta)\Delta t\right)^2 + \nu^2 \Delta S(X_t^{(n)}, X_{t+\Delta t}^{(n)}, J_\theta)\right). \tag{34}$$

This formulation ensures that the learned value function respects the Hamilton-Jacobi equation while compensating for the mismatch between the observed behavioral data and the distribution induced by the optimal policy.

3 Numerical Experiments

To analyze the behavior of DEEP DOCTR-L, we conducted numerical experiments within a linear-quadratic (LQ), risk-neutral framework. This setting was chosen because it provides a reference solution, allowing direct comparison between the learned policies and the known optimal solution obtained by solving a Riccati equation (see Appendix A).
Before evaluating the effectiveness of DEEP DOCTR-L, we validated our implementation by reproducing the experiments described in Halperin (2023). Details of this validation are provided in Appendix B.

3.1 Objectives and Methodology

We aimed to answer three main questions:

1. Can DEEP DOCTR-L reconstruct the optimal policy from a naive behavioral policy, or at least get closer to it? We compare the mean function of the DOCTR-L policy with that of the optimal policy, derived from the Riccati equation.
2. Does DEEP DOCTR-L improve the behavioral policy? To assess this, we compare the expected cost associated with the DEEP DOCTR-L policy, the behavioral policy, and the optimal policy. Expected costs are estimated via Monte Carlo simulations on newly generated trajectories under each policy.
3. If the initial behavioral policy is already near-optimal, can DEEP DOCTR-L reconstruct or further refine the optimal policy? Instead of using a fully naive policy, we approximate the Riccati solution with a simple pre-trained neural network and use it as the mean function of the behavioral policy.

The experimental setup, including parameter choices and numerical configurations, is detailed in Appendix C.

3.2 Results

Our numerical experiments led to the following main findings:

- When starting from a naive behavioral policy, DEEP DOCTR-L does not recover the optimal policy, nor does it systematically get closer to it. Despite extensive hyperparameter tuning, we could not obtain a learned policy whose mean function closely resembles that of the Riccati solution.
- However, in some cases, DEEP DOCTR-L successfully improves the behavioral policy by reducing the expected accumulated cost. While this improvement was not always consistent, it suggests that DOCTR-L can provide practical benefits even when it does not recover the optimal policy.
- When using a near-optimal behavioral policy, DEEP DOCTR-L did not further refine the policy. In fact, the learned policies often diverged from the optimal one. While we did not find hyperparameter configurations that systematically improved the near-optimal policy, we cannot rule out the possibility that a more extensive tuning process could yield different results.

We first present the results obtained when DEEP DOCTR-L is trained from a naive behavioral policy. The following figures illustrate the mean function of the learned policy, the evolution of state trajectories, and the associated accumulated costs.

Policy	Expected cost	95% CI Lower	95% CI Upper
Riccati	0.228	0.22	0.23
DOCTR-L	2.21	2.18	2.23
Behavioural	5.25	5.20	5.30

Table 1 Expected accumulated cost estimates for different policies.

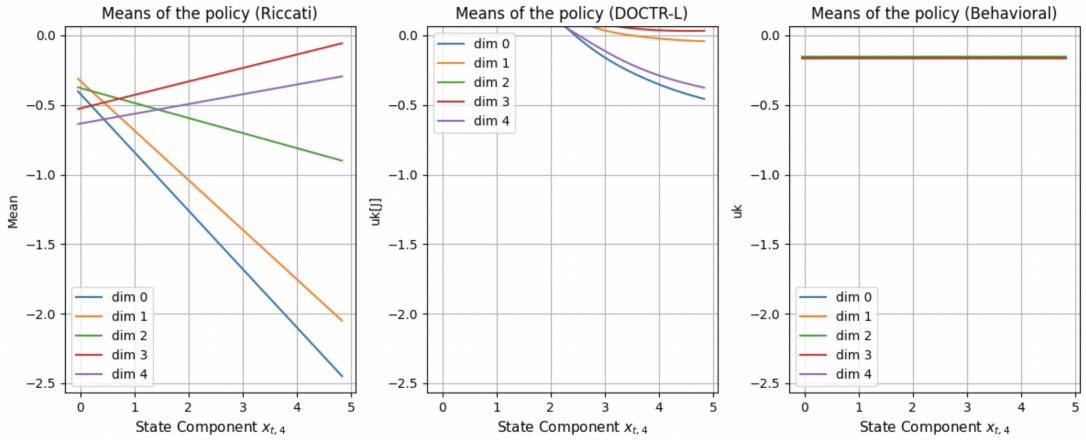

Fig. 1 Mean function of the control policies: Riccati (left), DEEP DOCTR-L (middle), and behavioral (right). The mean function of DEEP DOCTR-L differs significantly from the optimal Riccati solution and does not exhibit a clear structure.

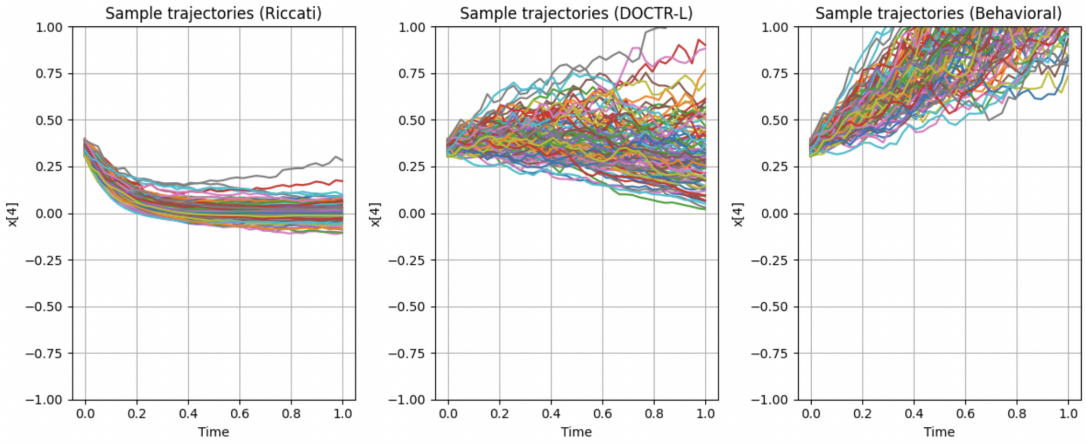

Fig. 2 Sampled state trajectories (100 randomly selected) for the Riccati policy (left), DEEP DOCTR-L (middle), and the behavioral policy (right). The Riccati-based policy stabilizes the trajectories effectively, keeping them close to the origin. The behavioral policy leads to significant state divergence. DEEP DOCTR-L exhibits intermediate behavior: while it does not fully stabilize the trajectories like the Riccati policy, it prevents excessive divergence observed in the behavioral policy.

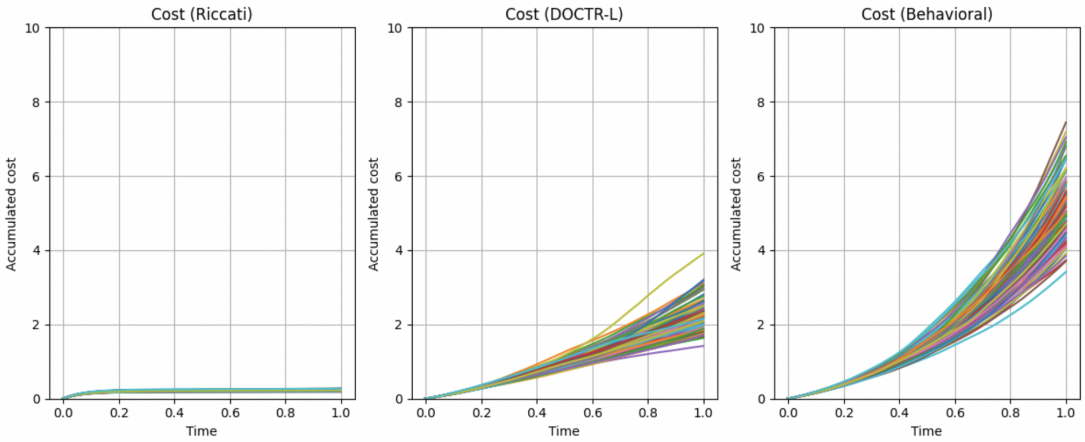

Fig. 3 Accumulated cost over time (100 randomly selected) for the Riccati policy (left), DEEP DOCTR-L (middle), and the behavioral policy (right). While DEEP DOCTR-L fails to approximate the optimal policy, it still achieves a significantly lower cost compared to the behavioral policy.

We now present the results obtained when DEEP DOCTR-L is trained from a near-optimal behavioral policy. The following figures illustrate the mean function of the learned policy, the evolution of state trajectories, and the associated accumulated costs.

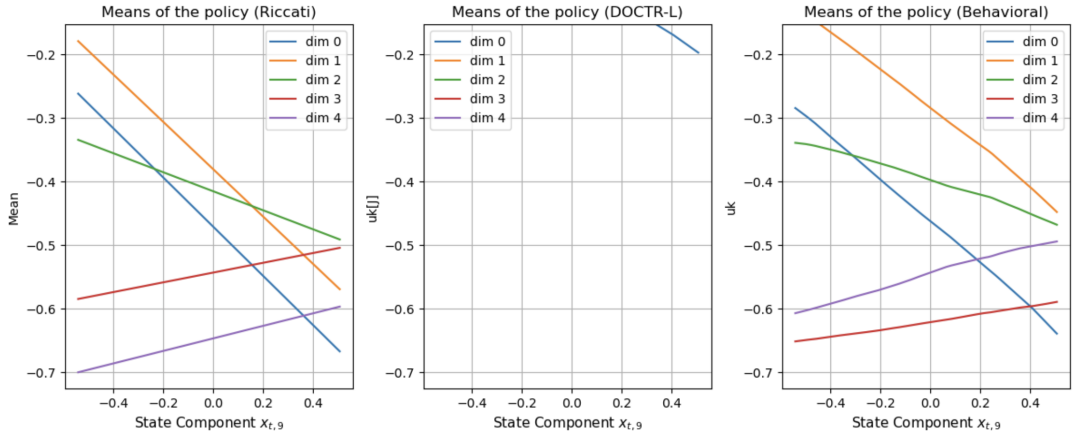

Fig. 4 Mean function of the control policies: Riccati (left), DEEP DOCTR-L (middle), and behavioral (right). The mean function of DEEP DOCTR-L does not align with the optimal Riccati policy. Even when starting from a near-optimal behavioral policy.

Policy	Expected cost	95% CI Lower	95% CI Upper
Riccati	0.228	0.226	0.229
DOCTR-L	1.627	1.619	1.635
Behavioral	0.331	0.329	0.333

Table 2 Expected accumulated cost estimates for different policies in the near-optimal case. Unlike the naive case, the behavioral policy already achieves a low expected cost. DEEP DOCTR-L does not improve it and sometimes performs worse.

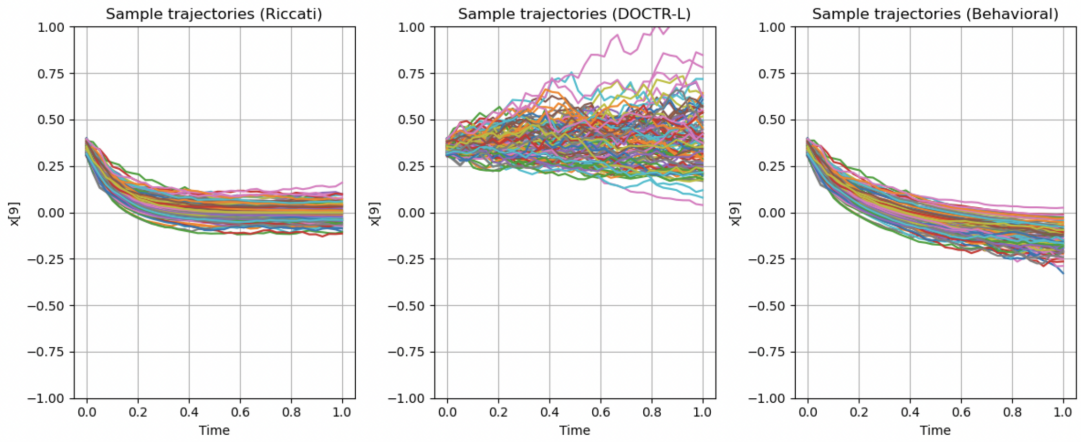

Fig. 5 Sampled state trajectories (100 randomly selected) for the Riccati policy (left), DEEP DOCTR-L (middle), and the behavioral policy (right). Unlike in the naive case, the near-optimal behavioral policy already stabilizes the trajectories. DEEP DOCTR-L does not improve upon this stabilization and does not exhibit behavior closer to the Riccati solution.

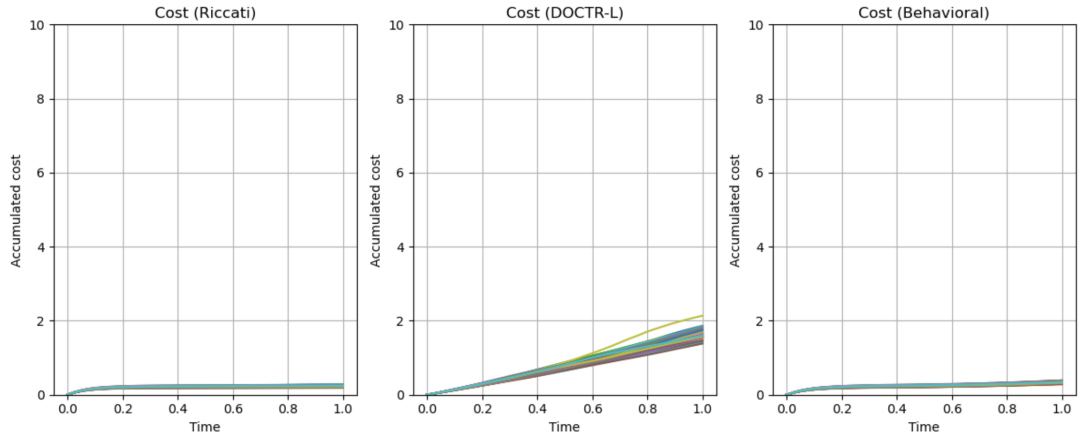

Fig. 6 Accumulated cost over time (100 randomly selected) for the Riccati policy (left), DEEP DOCTR-L (middle), and the behavioral policy (right). Unlike the naive case, the behavioral policy already achieves low cost. DEEP DOCTR-L performs worse than the behavioral policy.

3.3 Discussion

The difficulty of reconstructing the optimal policy from a naive behavioral policy was expected. Offline RL is inherently challenging because:

- The agent has no ability to actively explore the environment to generate informative data.
- The dataset is fixed and may only cover arbitrary regions of the state-action space, which might not be relevant for learning an optimal policy.

In practice, these limitations imply that DEEP DOCTR-L can only improve upon the observed behavioral policy but cannot be expected to learn the optimal policy unless the training data already contains sufficient information about it.

Some additional considerations arose during our study, suggesting potential future research directions:

- **Policy robustness to environmental changes:** We did not test whether a policy that improves over the behavioral policy remains effective under changes in drift or volatility. In practice, regime shifts are common, particularly in financial applications.
- **Data scaling sensitivity:** Our experiments relied on normalized trajectories maintained at scales near unity. Real-world data, particularly financial datasets, often involve substantial variations in magnitude, which could negatively impact training stability and policy quality. Developing robust normalization techniques or adaptive scaling strategies would be essential for practical deployment.

Appendix A Linear-Quadratic Risk-Neutral Case with Semi-Closed Solution

We consider a particular case of the framework described in Section 2, where the problem simplifies to a linear-quadratic (LQ) setting with a risk-neutral objective. Specifically, we assume:

- The cost function transformation is the identity: $U(x) = x$.
- No discounting: $r = 0$.
- The drift term has no constant component: $\mu_0^{(0)} = 0$.
- No state-action interaction in the drift: $\mu_1^{(1)} = 0$.
- The control weight in the running cost is a constant: $f_1(x) = 2c_1, \quad c_1 \in \mathbb{R}$.
- The state running cost is quadratic and isotropic: $f_0(x) = c_0\|x\|^2 I, \quad c_0 \in \mathbb{R}$.
- The diffusion term is state-dependent and diagonal: $\sigma(x) = \sigma_0 \operatorname{diag}(x), \quad \sigma_0 \in \mathbb{R}$.

Under these assumptions, the optimal policy is gaussian and its mean u_t can be explicitly characterized by solving a differential Riccati equation. We have:

$$u_t = -f_1^{-1}\mu_1^{(0)\top} K(t) X_t, \tag{A1}$$

where $K(t)$ is the solution of the Riccati equation:

$$K' = -f_0 - K\mu_0^{(1)} - \mu_0^{(1)\top} K - \sigma_0^2 K + K^\top \mu_1^{(0)} f_1^{-1} \mu_1^{(0)\top} K, \quad K(T) = 0. \tag{A2}$$

This differential equation is solved backward in time from T to 0 to obtain the time-dependent feedback gain.
We solve the Riccati equation numerically using a backward integration scheme. Given the terminal condition $K(T) = 0$, the equation is integrated backward using an adaptive solver, ensuring numerical stability. The resulting matrix $K(t)$ allows for the computation of the optimal control at any state X_t and time t.

Appendix B Reproducing the Experiments

To validate our implementation of DOCTR-L, we reproduced the numerical experiments presented in Halperin (2023). The objective was to verify whether our implementation correctly follows the described methodology and produces comparable results.

B.1 Experimental Setup

The experiments were conducted under the same conditions as in Halperin (2023), with two settings: a moderate-dimensional case ($d = 10$) and a high-dimensional case ($d = 100$). The policy learning was performed in an offline setting, using state-action trajectories generated from a behavioral policy.

State and control space:

- State space: $\mathcal{X} = \mathbb{R}^d$, with $d \in \{10, 100\}$.
- Control space: $A = \mathbb{R}^m$, with $m = 5$.

State dynamics: The controlled state process follows the stochastic differential equation:

$$dX_t = (\mu_0(X_t) + \mu_1(X_t)\alpha_t)dt + \sigma(X_t)dW_t. \tag{B3}$$

The coefficients are set as:

- $\mu_0(X_t) = \mu_0^{(0)} + X_t \cdot \mu_0^{(1)}$, with

$$\begin{aligned} \mu_0^{(0)} &= 0.1 I_d, \quad \mu_0^{(1)} = 0.2 I_{d\times d} \text{ (for } d = 10), \\ \mu_0^{(0)} &= 0.01 I_d, \quad \mu_0^{(1)} = 0.02 I_{d\times d} \text{ (for } d = 100). \end{aligned} \tag{B4}$$

- $\mu_1(X_t) = \mu_1^{(0)} + X_t \cdot \mu_1^{(1)}$, with

$$\begin{aligned} \mu_1^{(0)} &= 0.1 I_{d\times m}, \quad \mu_1^{(1)} = 0.2 I_{d\times m} \text{ (for } d = 10), \\ \mu_1^{(0)} &= 0.01 I_{d\times m}, \quad \mu_1^{(1)} = 0.02 I_{d\times m} \text{ (for } d = 100). \end{aligned} \tag{B5}$$

- The diffusion function $\sigma(X_t)$ was not specified in Halperin (2023). We set:

$$\sigma(X_t) = 0.4\,\mathrm{diag}(X_t), \tag{B6}$$

  ensuring a level of variability comparable to state values.

Cost function and utility transformation:

- Running cost:

$$f(X_t, \alpha_t) = f_0(X_t) + \frac{1}{2} f_1(X_t) \|\alpha_t\|^2, \tag{B7}$$

  with:

$$f_0(X_t) = \|X_t\|^2, \quad f_1(X_t) = 5.0 \|X_t\|^2. \tag{B8}$$

- The function $U(x)$ was not specified in Halperin (2023). Since the paper mentions a quadratic term x^2 in an earlier section, we assumed:

$$U(x) = 10x^2 \quad (\text{for } d = 10), \quad U(x) = x^2 \quad (\text{for } d = 100). \tag{B9}$$

Behavioral policy: The data was generated using a Gaussian mixture policy:

$$\pi_0(t, x, a) = \sum_{k=1}^{K} \omega_k \mathcal{N}(a; u_k, \sigma_k^2 I), \tag{B10}$$

with:

- Number of components: $K = 2$.
- Means: $u_k \sim \mathcal{U}([-0.5, 0.5]^m)$.
- Variances: $\sigma_k^2 \sim \mathcal{U}([0.2, 0.4]^m)$.
- Mixture weights: $\omega_k = 1/K$.

Training procedure:

- Horizon: $T = 1$.
- Number of time steps: 40.
- Number of simulated trajectories: $10,000$.
- KL temperature: $\lambda = 1$ (for $d = 10$), $\lambda = 0.2$ (for $d = 100$).
- Discount rate: $r = 0.03$.
- Variance of the learning noise: $\nu^2 = 100$ (for $d = 10$), $\nu^2 = 10$ (for $d = 100$).

Neural network architecture:

- Fully connected feedforward network with three hidden layers.
- Hidden layers: 100 neurons per layer, ReLU activations.
- Output layer: linear activation.

Optimization:

- Optimizer: Adam with learning rate 10^{-3}, weight decay 10^{-3}.
- Learning rate scheduler: OneCycleLR with max learning rate 5×10^{-3}.
- Number of training epochs: 30, batch size: 256.

B.2 Numerical Results and Comparison

The obtained results are broadly consistent with those reported in Halperin (2023). The loss function exhibits a similar decreasing trend, and the overall structure of the learned policy follows the expected patterns. However, several notable differences arise, primarily due to discrepancies in state trajectory distributions.

B.2.1 State Trajectories

One key observation is that the state trajectories obtained in our implementation cover a broader range of values compared to those in Halperin (2023). This results in a wider distribution of state inputs during training, which in turn affects the learned policy.

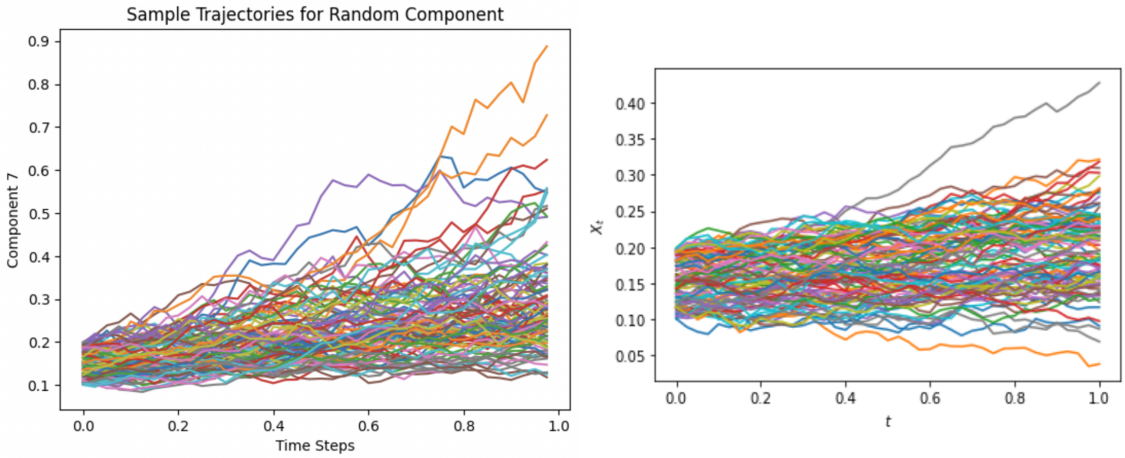

Fig. B1 Comparison of state trajectories over time. Left: our implementation. Right: results from Halperin (2023). The wider range of observed states in our experiments likely explains differences in the learned policy.

B.2.2 Learned Policy Structure

The structure of the learned policy is consistent with the expected behavior: the means of the Gaussian mixture policy exhibit a symmetric dependence on the state variable and converge asymptotically to constant values. However, in our case, convergence occurs at larger absolute values of x. In Halperin (2023), stabilization appears around $|x| \approx 1.5$, whereas in our experiments, it happens closer to $|x| \approx 10$. This can be attributed to the broader state distribution in our setup, which forces the model to extrapolate over a wider range of inputs. Additionally, this difference may also stem from the fact that our utility function is likely not the same as the one used in the original paper.

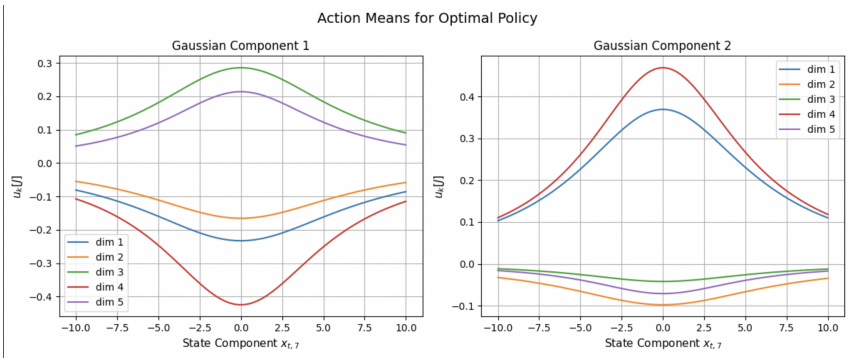

Fig. B2 State-dependence of the mean functions $u_k[J]$ of the learned policy for all action dimensions in our implementation. The overall shape is similar to Halperin (2023), but convergence occurs at different state ranges due to variations in observed trajectories.

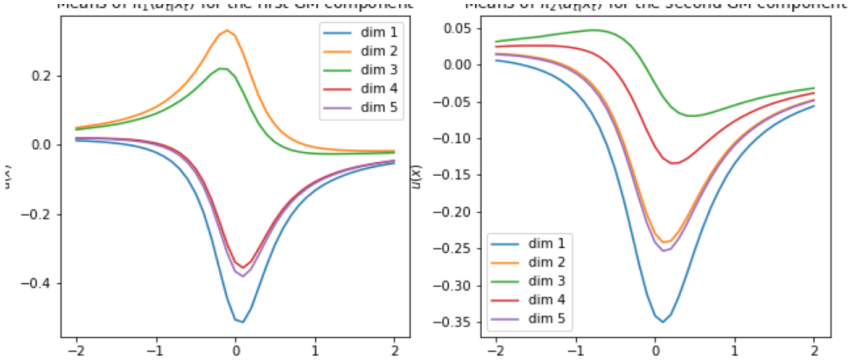

Fig. B3 State-dependence of the mean functions $u_k[J]$ in Halperin (2023). Convergence occurs at smaller absolute values of x compared to our results.

B.2.3 Loss Evolution

The loss function follows a similar decreasing trend, indicating successful training.

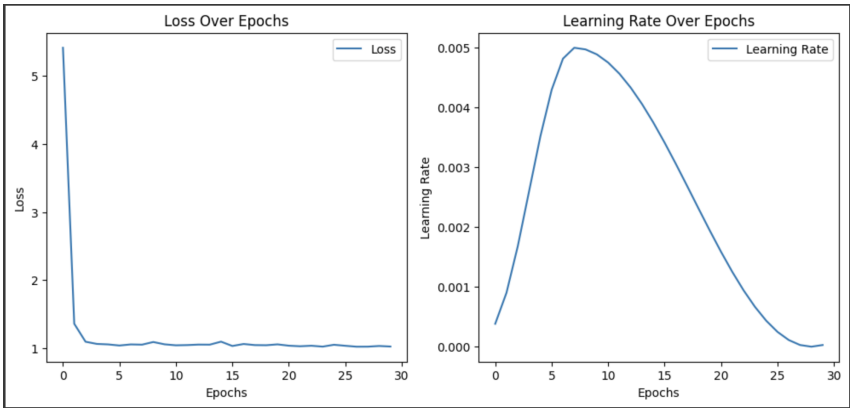

Fig. B4 Loss function evolution in our implementation. The trend is decreasing, showing successful training, though the absolute scale is higher in higher dimensions.

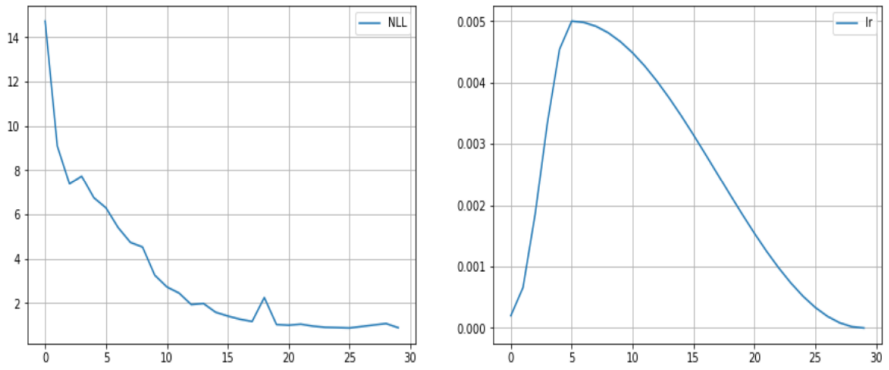

Fig. B5 Loss function evolution in Halperin (2023). The loss is lower in absolute terms, likely due to differences in dimensionality and trajectory distribution.

B.3 High-Dimensional Case (Dimension 100)

The same methodology was applied in the high-dimensional setting. To match Halperin (2023), the coefficients of the drift function were scaled down by a factor of 10, and the variance of the regression noise was reduced accordingly ($\nu^2 = 10$). The learned policy in this case closely follows the expected trends, with the same qualitative structure as in $d = 10$.

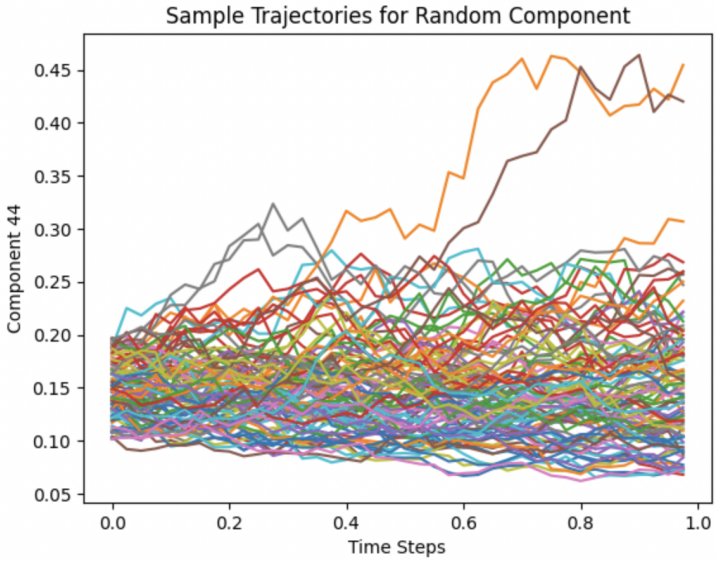

Fig. B6 State trajectories in our high-dimensional setting ($d = 100$).

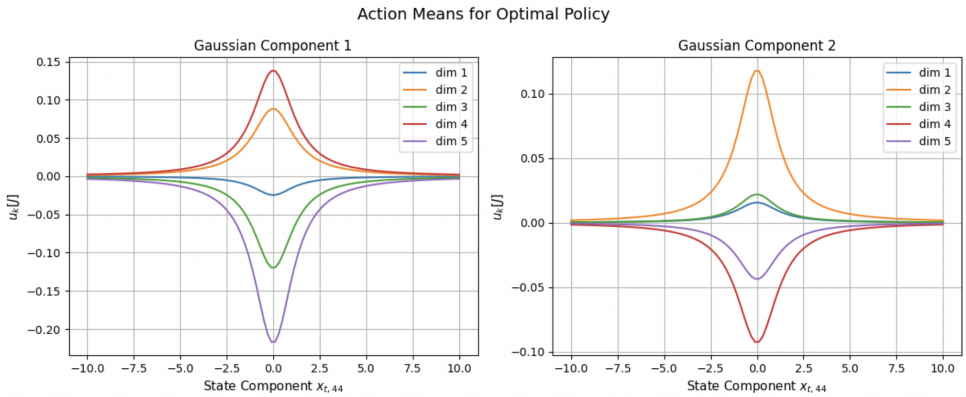

Fig. B7 Learned policy mean functions in the high-dimensional setting ($d = 100$).

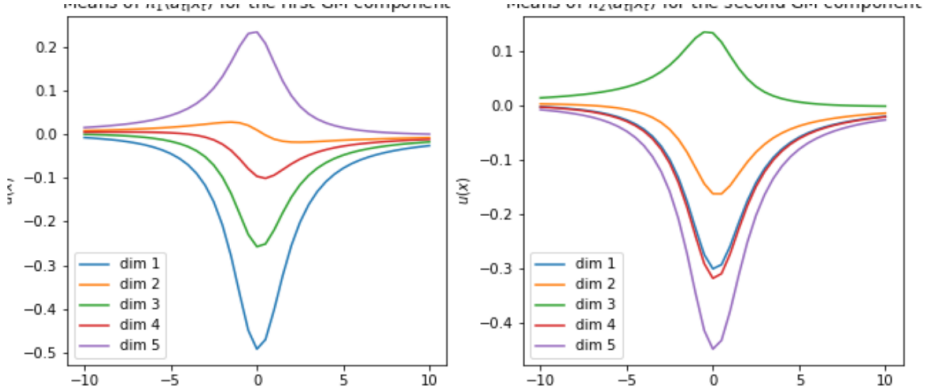

Fig. B8 Learned policy mean functions in Halperin (2023) for the high-dimensional case ($d = 100$).

The loss function in $d = 100$ follows the same decreasing trend but remains significantly larger in our case.

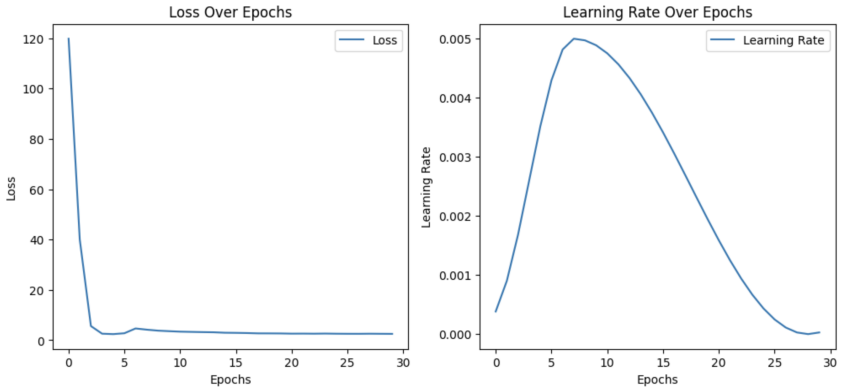

Fig. B9 Loss evolution in the high-dimensional setting ($d = 100$).

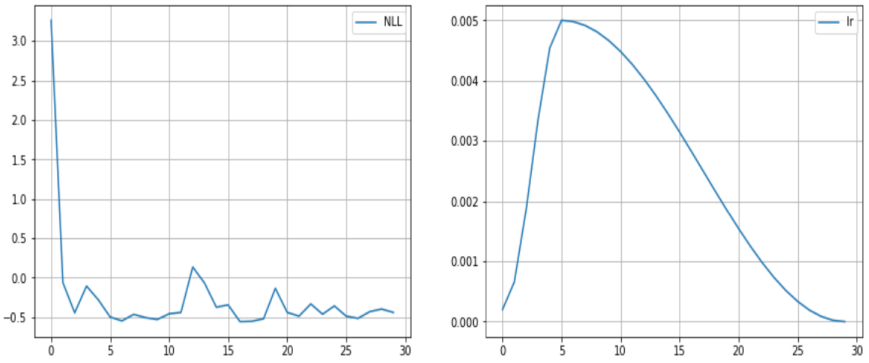

Fig. B10 Loss evolution in Halperin (2023) for the high-dimensional setting ($d = 100$). The trend remains similar, but absolute values differ.

Appendix C Experimental Setup

The experiments were conducted in two settings: a low-dimensional case ($d = 10$) and a high-dimensional case ($d = 100$). The policy learning was performed in an offline setting, using state-action trajectories generated from two behavioral policies:

- A naive Gaussian policy with fixed mean and variance.
- A learned policy, where the mean function was approximated using a neural network trained on Riccati-optimal controls.

State and control space:

- State space: $\mathcal{X} = \mathbb{R}^{10}$.
- Control space: $A = \mathbb{R}^5$.

State dynamics: The controlled state process follows the stochastic differential equation:

$$dX_t = (\mu_0 + \mu_1 \alpha_t)dt + \sigma(X_t)dW_t. \tag{C11}$$

where:

- $\mu_0 = 0.2 I_d X_t$, a state-dependent drift term.
- $\mu_1 = 1.0(\sin(i) + 1)_{i=1}^{d \times m}$, a deterministic control influence.
- $\sigma(X_t) = 0.4\,\mathrm{diag}(X_t)$, ensuring state-dependent volatility.

Cost function and utility transformation:

- Running cost:
$$f(X_t, \alpha_t) = \|X_t\|^2 + c_a \|\alpha_t\|^2, \tag{C12}$$
with $c_a = 2.0$ controlling the weight of the action penalty.
- The terminal cost function is set to:
$$U(x) = x. \tag{C13}$$

Behavioral policies:

- **Naive policy:** A Gaussian distribution with a fixed mean sampled uniformly from $[-0.5, 0.5]^m$ and variance sampled from $[0.2, 0.4]^m$.
- **Learned policy:** A neural network trained to approximate the Riccati-optimal control function using offline training data.

Training setup for the learned behavioral policy:

- Training dataset: $1,000$ samples of states and their corresponding Riccati-optimal actions.
- Model: A fully connected feedforward neural network with two hidden layers of 128 neurons each.
- Activation: ReLU.
- Optimizer: Adam with learning rate 10^{-3}, weight decay 0.01.
- Loss function: Mean Squared Error (MSE).
- Number of epochs: 50.
- Batch size: 256.

Training procedure:

- Time horizon: $T = 1$.
- Number of time steps: 40.
- Number of simulated trajectories: $1,000$.
- KL temperature: $\lambda = 0.01$ (naive policy), $\lambda = 1.0$ (learned policy).
- Discount rate: $r = 0.0$.
- Regression noise variance: $\nu^2 = 100$.

Neural network for value function learning:

- Architecture: A fully connected network with three hidden layers of 100 neurons each.
- Output layer: Linear activation.
- Optimizer: Adam with weight decay 0.001.
- Learning rate scheduler: OneCycleLR with a maximum learning rate of 5×10^{-3}.
- Number of epochs: 30, batch size: 256.

References

Halperin, I. (2023, 12). Distributional offline continuous-time reinforcement learning with neural physics-informed PDEs (SciPhy RL for DOCTR-L). *Neural Computing and Applications*, *36*(9), 4643–4659, https://doi.org/10.1007/s00521-023-09300-7