Benchmarking Reinforcement Learning Algorithms on Tetherball Games

Analyse von Reinforcement Learning Methoden für Tetherball

Bachelor-Thesis von Nourhan Khaled aus Ägypten

Tag der Einreichung:

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Hany Abdulsamad

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Benchmarking Reinforcement Learning Algorithms on Tetherball Games
Analyse von Reinforcement Learning Methoden für Tetherball

Vorgelegte Bachelor-Thesis von Nourhan Khaled aus Ägypten

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Hany Abdulsamad

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den September 26, 2018

(Nourhan Khaled)

Abstract

Robotic applications in the real world are faced with the stochasticity and uncertainty of the environment, which renders the approaches of having hand-tuned controllers impractical. Reinforcement learning offers a more flexible alternative due to its ability to adapt to the ever-changing environment dynamics, as opposed to rigid engineered approaches, where each scenario has to be accounted for separately. Policy search methods, a sub-field of reinforcement learning, provide a potentially more flexible alternative that is scalable to robotic applications by optimizing the robot's policy parameters to the task at hand. In a real world scenario, the same task can have a multitude of settings or *contexts*, and instead of engineering a solution for each context, contextual policy search methods generalize their knowledge of the task across different contexts making the robot's policy more versatile. This thesis reviews and benchmarks four contextual policy search algorithms and empirically evaluates their sample efficiency, scalability and performance on multidimensional benchmark functions and on a simulated robot tetherball task.

Zusammenfassung

Roboter sind einem großen Maß von Variabilität und Unsicherheit in der realen Welt ausgesetzt. Dies schränkt den Einsatz von klassischen, fein abgestimmten Lösungen ein, da es unmöglich ist alle Szenarien zu berücksichtigen. Reinforcement Learning bietet eine flexible Alternative, indem es Robotern die Fähigkeit verleiht aus Interaktionen zu lernen, und sich an eine dynamische Umgebung anzupassen. Policy Search Methoden bilden eine Kategorie im Bereich von Reinforcement Learning. Solche Methoden zeigen zurzeit großes Potential in der Robotik, indem sie einem Roboter ermöglichen viele Variationen einer Aufgabe in verschiedenen *Kontexten* gleichzeitig zu lernen, und auf unbekannte Situationen zu generalisieren. In dieser Thesis werden sogenannte Contextual Policy Search Methoden untersucht und im Hinblick auf Effizienz, Skalierbarkeit und Optimalität empirisch ausgewertet. Dabei werden sowohl verschiedene hochdeminsionale Testfunktionen als auch eine simulierte Tetherball-Aufgabe analysiert.

Acknowledgements

I would like to thank Svenja, Joni, Daniel and Doro for the ice-cream, Macadamia nuts, Nutella Pretzels, early breakfasts and unplanned late dinners. Good food is central to successful projects.
I am genuinely grateful for having Hany as a supervisor. He has been patient with me and provided me with valuable advice along the way. Thank you for pushing me to be more self reliant. And most of all, thank you for propagating the importance and beauty of understanding how things work together, this will stay with me.
I am immensely thankful for my family's support. The funny video calls kept me motivated.
Last and certainly not least, I would like to thank Jan for giving me this opportunity. I have gained so much more than academic knowledge.

Contents

List of Figures

List of Algorithms

1 Introduction

1.1 Motivation

From as far as the 8th century BC, when Homer wrote about *mechanical waiters* in Iliad [Lattimore et al., 1961], humans have aspired to create intelligent machines [Buchanan, 2005], now referred to as robots. In particular, robots with complex motor skills that have the ability to cope in the real world environment and assist humans in their daily lives. Despite the extensive efforts, and even successes, that have gone in the direction of building hand-crafted robot controllers [Yoshiike et al., 2017], [Sakagami et al., 2002], one can only go so far with hard-coded controllers in a real world environment that is packed with unpredictability. The element of intelligence has been the missing piece that would enable robots to adapt to the uncertainty of the real world. With the rise of reinforcement learning algorithms, especially Policy Search (PS) methods [Deisenroth et al., 2013] and the introduction of motor primtives [Ijspeert et al., 2003], having autonomous robots equipped with complex motor skills is becoming more attainable. Robots are now more capable of performing dynamic motor tasks that were once deemed unlikely such as bipedal locomotion [Xie et al., 2018], ball-in-a-cup [Kober and Peters, 2009], table-tennis [Muelling et al., 2010] and even pancake-flipping [Kormushev et al., 2010], and they do so solely by *learning* as opposed to being hard-coded for the task.

Key to versatile robots is their ability to generalize their experience across similar tasks. One example of such generalization is learning tasks with variable settings. For instance, if the robot's task is to throw a ball to a certain goal position, instead of relearning the whole task for a different goal position or *context*, the robot should adapt its current knowledge of the task to the new setting or *context*. This notion of task contextualization results in faster adaptive robots and generally more efficient learning. The task of robot tetherball has been introduced as an application of contextual robot learning [Parisi et al., 2015]. In a typical game of tetherball, the player should be able to react to the ball coming from different positions with different velocities. Even more so, the player could intercept the ball at a range of positions. These variables could all be incorporated into the context of the task to create a flexible robot player. Following the work of [Parisi et al., 2015], [Abdulsamad et al., 2014], this thesis explores some state-of-the-art reinforcement learning algorithms that can be applied to this task. From the multitude of policy search algorithms out there, a few are reviewed in this thesis and four are selected for benchmarking on standard optimization test functions and on a simulated robot tetherball task.

Figure 1.1: Picture (left) shows a typical game of tetherball with a ball at the end of a rope attached to a pole. Players try to hit the ball in the one way and a player scores if his/her opponent doesn't hit the ball. Picture (right) shows the robot tetherball setup used in [Parisi et al., 2015].

1.2 Overview

This section provides an overview of the structure of the thesis and gives a brief summary of the topics discussed in each chapter. The thesis is structured as follows:

Chapter 2 discusses the foundations and fundamentals needed in the scope of this thesis. First reinforcement learning is introduced as an approach to solve robotic tasks. Then an overview of policy search methods, a class of approaches to solve reinforcement learning problems, is given. The Kullback-Leibler divergence (KL), an information theoretic distance metric between probability distributions is breifly introduced to be later used in policy search methods. Finally, Dynamic Movement Primitives (DMPs) are introduced as a parametrized policy representation, along with imitation learning that allows us to bootstrap the learning of complex tasks.

Chapter 3 thoroughly discusses and derives the four selected algorithms for this thesis. For each algorithm, the standard version is first derived, then the derivation is extended to the contextual version to allow for generalizing to different task settings. The chosen algorithms are Relative Entropy Policy Search (REPS), Model-based Relative Entropy (MORE), Covariance Matrix Adaptation Evolution Strategies (CMA-ES) and finally regularized versions of REPS.

Chapter 4 gives the details of the implementation starting with the dissection of the tetherball task, then introducing the software used and finally diving into the details of the implemented setup.

Chapter 5 evaluates the performance of the selected algorithms on standard benchmark functions. Some algorithm-specific parameters are examined and more general metrics such as sample efficiency and scalability are evaluated. Then results on the simulated task of robot tetherball are presented.

Chapter 6 concludes the thesis and discusses the challenges encountered throughout.

Appendix presents some thorough derivations of the selected algorithms.

2 Background and Foundations

This chapter presents the basic concepts used throughout this thesis. First, Markov Decision Processes (MDPs), the classical framework for reinforcement learning will be introduced along with some basic principles of reinforcement learning. Then, to harness the power of reinforcement learning into robotic applications, the robot control problem will be formalized as a reinforcement learning problem. To solve the reinforcememt learning problem Policy Search (PS) methods, a popular approach for robotic problems will be discussed. The Kullback-Leibler divergence (KL), an information theoretic distance metric between probability distributions is briefly introduced. Finally, Dynamic Motor Primitives (DMPs) will be presented for policy representation.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning, which unlike supervised learning, where the training data is labeled with *the right answer*, the agent (ex: robot) has to learn how to behave in the environment and perform its task without any prior knowledge. The agent does so by learning from its own experience; it performs some actions in the environment and based on a reward function, the agent can identify how good, or not so good, the action was. The goal of the agent is then to maximize the accumulation of rewards from the beginning till the end of its task, which is referred to as the *long-term reward* in literature. The typical interaction loop between agent and environment is illustrated in Figure 2.1.

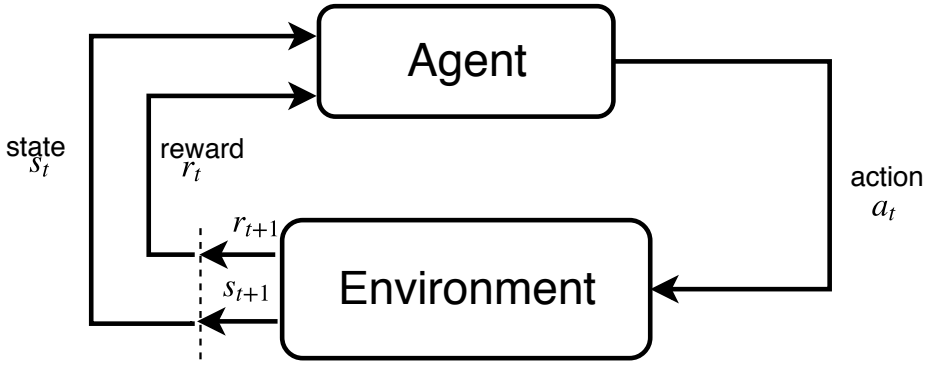


Figure 2.1: The classic Reinforcement Learning interaction loop between the agent and environment. The agent takes an action a_t at state s_t. The environment then responds with the corresponding reward r_{t+1} and the new state s_{t+1}, which are fed back to the agent. Drawing recreated from [Sutton et al., 1998]

A sub-field where reinforcement learning (or machine learning in general) and robotics are intersect is *Robot Learning* [Connell and Mahadevan, 2012]. It aims to bridge the gap between programmed robots, where the controllers for the robots are tuned and designed by engineers, and fully autonomous robots, where robots perform tasks, e.g., household assistance, without explicit human control. Robot learning harnesses the power of machine learning techniques to extract relevant information for solving a robotic task [Deisenroth et al., 2013].

2.1.1 Markov Decision Processes

The classic setting for RL are Markov Decision Processes (MDPs). MDPs [Bellman, 1957] are a mathematical framework for decision making in deterministic and stochastic environments i.e., environments that have some degree of randomness. MDPs satisfy the Markov property [Markov, 1961], which states that "the future is independent of the past given the present", meaning that the next state only depends on the current state, no previous states are required, which is sometimes, and quite conveniently, referred to as *memorylessness*. A Markov Decision Process (MDP) can be formally defined as a tuple $(\mathcal{S}, \mathcal{A}, P, r)$:

- A set of states $s \in \mathcal{S}$ that describe the environment.
- A set of actions $a \in \mathcal{A}$ that can be be performed by the agent in the environment.

- A transition function $P(s_{t+1}|s_t, a_t)$ that gives the probability of a new state s_{t+1} after an action a_t has been taken in state s_t.
- A reward function $r(s_t, a_t)$ that specifies the immediate reward after taking action a_t in state s_t.

The Markov property is expressed as

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}...) = P(s_{t+1}|s_t, a_t) \ .$$

Typically, the objective in a reinforcement learning problem is to maximize the total discounted long-term reward

$$\max \ J = \sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \ ,$$

where γ is the *discount factor*, which defines how much the agent values immediate rewards versus long-term rewards. In some works [Arulkumaran et al., 2017], [Schulman et al., 2015], [Duan et al., 2016], γ is considered as a part of the MDP.
In this setting, the agent observes the current state s_t and based on some criterion it takes an action a_t. This mapping from state to action is called a *policy*. A deterministic policy denoted by $\pi(s) = a$, is one which returns one action for a given state. It will always return the same action given the same state. A stochastic policy $\pi(a|s)$ on the other hand, can give different actions given the same state determined by a distribution over actions. There are two main approaches to solving reinforcement learning problems, value-based methods and policy search methods. In value-based methods, a value function returns a value $V(s)$ i.e., the total expected reward, of being in a certain state and following the current policy. The value computed by the value function is then used to select the best actions. There are various methods in value function approaches such as Dynamic Programming based methods, Monte Carlo methods and Temporal Difference methods. The alternative method to value-function learning is policy search (PS), where the parameters $\boldsymbol{\theta}$ of a policy $\pi(\theta)$ are optimized. This approach will be further elaborated in section 2.2.

One of the most fundamental dilemmas in reinforcement learning is the *exploration-exploitation* trade-off. It is the choice between *exploiting* the safe actions that have been tried before and shown promising results and *exploring* new actions that may or may not give better results. As it turns out, exploration is key in the success of reinforcement learning algorithms as it gives the agent the opportunity to discover new, and possibly more rewarding, areas of the environment as opposed to staying in a sub-optimal area. However, this should be taken into account when using reinforcement learning in the context of robotics, because exploration is not always desirable. Now, to use reinforcement learning in robotic applications, the robot control problem must first be formulated as a reinforcement learning problem.

2.1.2 Robot Control as a RL Problem

As proposed in [Deisenroth et al., 2013], robot control can be modeled as a reinforcement learning problem. The robots' state space s comprises of the robot's state (e.g., joint angles, joint velocities, position of end-effectors, etc) as well as the state of the environment (e.g. location of targets for task). Based on a control policy π, the robot chooses some action a, which, in the context of robotics, is a motor command. The state of the robot is changed by the motor commands according to the probabilistic function $p(s_{t+1}|s_t, a_t)$. A sequence of states and actions forms a trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$, sometimes referred to as a rollout or path. The quality of the robot's trajectory is evaluated by a numeric scoring system that returns an accumulated reward signal $R(\tau)$. A positive reward for example would indicate that the robot performed the task, and a negative reward would penalize missing a target. Tasks in which rewards are returned every time-step of the trajectory are called *step-based* tasks. Alternatively, in *episodic* tasks (e.g., stroke based movements such as swinging racket or throwing dart), the rewards are accumulated and returned at the end of the episode after T time steps. The accumulated reward for a trajectory τ is given as

$$R(\tau) = \sum_{t=0}^{T} r_t(s_t, a_t) \ ,$$

where r_t is the immediate reward given by the reward function. The goal of robotic task is choosing an optimal policy π^* that maximizes the expected return J. For finite horizon i.e., finite number of timesteps T, the expected return under a certain policy π is given as the accumulated reward $R(\tau)$

$$J(\pi) = \mathbb{E}[\sum_{t=0}^{T} r_t|\pi] \ .$$

Alternatively, an infinite horizon can also be represented given a discount factor γ, as the discounted accumulated reward

$$J(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | \pi] \,,$$

where γ specifies the long-term/short-term rewards trade-off. The optimal policy is given by

$$\pi^* = \arg\max_{\pi} J(\pi) \,.$$

After setting up the reinforcement learning problem, the algorithm has to deal with high-dimensional state-action spaces, real-time requirements and the non-trivial cost of the robot interacting with the environment. Furthermore, in classical RL algorithms, exploration is essential to successful learning because it allows the agent to discover new and possibly more rewarding states and trajectories. When it comes to robots however, unrestrained exploration is highly discouraged due to hardware constraints; it might be dangerous for the robot and the environment. To solve a RL problem, value-based methods try to estimate the expected long-term reward of a policy, which is also known as the value function. As previously stated, the value function gives an indication of how good an action is for a given state and is hence used for selecting actions and updating the policy. This approach however, doesn't scale well to robot learning problems. As mentioned above, a robotic task usually has a large state-action space, and even more so, continuous actions, making the value functions not a feasible solution since they require filling up the state-space with data [Deisenroth et al., 2013]. Value-based methods are also faced with additional limitations such as discontinuous value-functions and the non-trivial task of optimizing highly non-linear approximations of the value function. Proposed in the next section, policy search methods alleviate some of the challenges encountered by value-based methods. Additionally PS methods have desirable properties such as using parameterized policies that make the robot-learning problem simpler and make it easier to apply RL, which makes it a popular approach in robotics.

2.2 Policy Search

As indicated by the name, policy search looks for the optimal policy. A policy π is a mapping from state s to action a. If the state and action set are finite, there will be a finite number of policies in the policy space to optimize over i.e., searching for the optimal policy among all possible policies. On the other hand, if the state and action set are continuous, which is typically the case for robot control, there will be infinitely many possible policies. In that case, the policy π is parameterized with some parameter $\boldsymbol{\theta}$ and and the search is performed in the parameter space $\boldsymbol{\Theta}$, $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. So, as opposed to value-based methods, policy search methods use parameterized policies $\pi(\boldsymbol{\theta})$ and work directly in the parameter space $\boldsymbol{\Theta}$, $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ of the policy. This significant reduction of the search space allows for the scaling of the RL problem to meet the high-dimensional continuous state and action space needs of a robotic task. Another advantage of using parameterized-policies is that predefined task-appropriate policy representations can be used as will be shown in Section 2.4.
One categorization of policy search methods is the distinction between model-free policy search methods and model-based methods. In model-free policy search methods, sampled trajectories $\tau^{[i]}$ and immediate rewards $r^{[i]}$ for the trajectories are obtained from interactions with the real robot. The parameters $\boldsymbol{\theta}$ of the policy are then updated in such a way to increase the likelihood of trajectories with higher rewards and thereby increasing the average return

$$J(\boldsymbol{\theta}) = \mathbb{E}[R(\tau)|\boldsymbol{\theta}] = \int R(\tau) p_\theta(\tau) \mathrm{d}\tau. \tag{2.1}$$

In model-based policy search methods, a model of the robot's and environment's dynamics is first constructed using sampled trajectories, then this model is used for improving the policy. Model-free approaches will be the main focus of this thesis.
One very popular group of algorithms in policy search methods is stochastic search algorithms or *black-box optimizers* [Hansen et al., 2003], [Sun et al., 2009], [Rückstieß et al., 2008]. Black-box optimization is a setup where an algorithm optimizes an objective function without any underlying knowledge of its landscape i.e., the algorithm has no knowledge of the shape or gradient of the objective function. This approach is particularly useful when the objective function is unknown or is too complex to model. Stochastic search algorithms maintain a search distribution over the parameters $\boldsymbol{\theta}$ that are to be optimized. This search distribution is then used to draw samples of parameter vectors and then evaluated using some objective function. This data set of samples and evaluations is then used to update the current search distribution using gradient-based updates [Sun et al., 2009], [Rückstieß et al., 2008], evolution strategies[Hansen et al., 2003], cross entropy method [Mannor et al., 2003], path integrals [Stulp and Sigaud, 2012] or information theoretic approaches [Peters et al., 2010]. The simplicity and ease of use of stochastic search algorithms resulted in their popularity. Chapter 3 reviews a few algorithms based on this approach.

Algorithm 1 Generic Stochastic Policy Search Algorithms

```
while not converged do
    for i=1 .. N do
        Draw parameters θ_i ~N(μ, Σ)
        Evaluate parameters θ_i and get returns R(θ_i)
    end for
    Update policy π(θ): ex: REPS, MORE, CMA-ES.
end while
```

2.3 Kullback-Leibler (KL) Divergence as a Distance Metric

An important factor to consider with policy updates on real robotic systems is that there should not be a large difference between the old policy and the new updated policy. Intuitively, we don't want the robot to drift far away from where it starts and take large exploration steps; it might be dangerous for both the robot and the environment. This is achieved by introducing a constraint on the Kullback-Leibler (KL) divergence [Kullback and Leibler, 1951] between the current and the updated policy distributions, which is also known as the relative entropy. The KL divergence quantifies *the distance* or *difference* between two probability distributions in terms of the information encoded in each distribution. More specifically, it measures the the shannon entropy [Shannon, 2001] of one distribution relative to the other, hence the name relative-entropy. More formally, it measures how much one probability distribution diverges from another. The KL divergence from q to p is defined as

$$\mathrm{KL}(p||q) = \int_{-\infty}^{\infty} p(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} \, \mathrm{d}\boldsymbol{\theta} \ ,$$

where p and q are continuous probability distributions. It can be seen as an expectation of the logarithmic difference between p and q. As will be shown in the next chapter, this metric will prove to be quite useful in the policy updates of the discussed algorithms, as it is usually used as a contraint in bounding two subsequent policy distributions in an attempt to limit information loss between updates.

2.4 Dynamic Motor/Movement Primitives (DMPs)

As mentioned in Section 2.2, policy search methods enable us to work with parameterized policies. Therefore, we can benefit from using policy representations that are relevant to a given task. To represent the parametric policy, Dynamic Movement Primitives or Dynamic Motor Primitives (DMPs) [Ijspeert et al., 2003], [Ijspeert et al., 2013]; a time-dependent policy representation that is commonly used in robotics, is used. Then, imitation learning, a method that enables learning from demonstrations [Osa et al., 2018], is used to bootstrap the parameters of the DMPs.

2.4.1 DMPs for Policy Representation

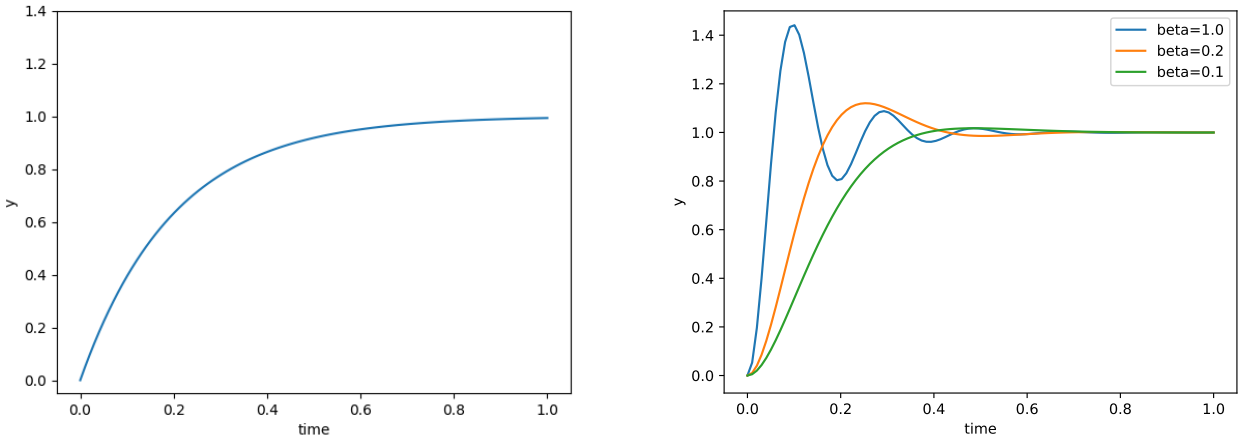
Formalized as second-order dynamic systems, DMPs are a mathematical framework that offers a compact representation of basic movements, such as hitting and grasping. They are used as trajectory generators for the real system. DMPs are essentially dynamical systems. A dynamical system is defined to be a system in which a function describes the evolution of a point in a geometrical space over time. This makes it convenient to represent trajectories. A basic example of a dynamical system could be a first order linear dynamical system

$$\dot{y} = \alpha(c - y), \tag{2.2}$$

with y describing the state of the system, c being the end state we would want the system to reach and α is a constant that determines how quickly the system reaches the end state. To add more complexity to the trajectory it can be shown in Figure 2.3 that a second order linear dynamical system

$$\ddot{y} = \alpha(\beta(c - y) - \dot{y}), \tag{2.3}$$

has more versatility, where β acts as the gain. Second-order linear dynamical systems have well defined behaviours but they still can't encode various classes of movements. DMPs opt for the stability of second-order dynamical systems and



(a) First-order linear dynamical system.

(b) Second-order linear dynamical system

Figure 2.2: Plots of first and second-order linear dynamical systems, showing the evolution of the state y of the system over time. It is evident that second order-linear dynamical sytems can represent more versatile trajectories, however they still can't represent various classes of movements.

introduce further complexity to the trajectory by adding a non-linear forcing function f. This system behaves like a linear spring-damper system with a non-linear forcing function modulating it.

$$\ddot{y} = \alpha(\beta(c - y) - \dot{y}) + f \ . \tag{2.4}$$

After adding a temporal scaling factor τ, that determines the speed of the execution and a phase variable z_t for time modulation, the DMPs can be fully defined as

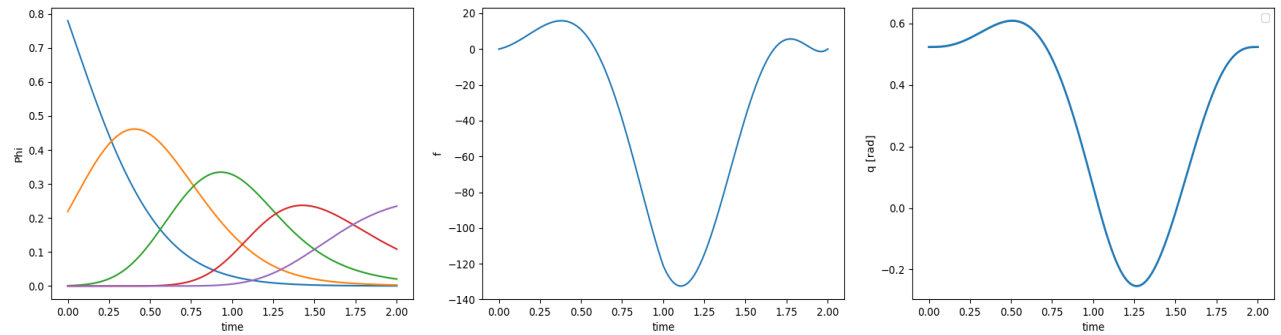
$$\ddot{\boldsymbol{q}}_t = \tau^2(\alpha(\beta(\boldsymbol{g} - \boldsymbol{q}) - \frac{\dot{\boldsymbol{q}}}{\tau}) + f(z_t, \boldsymbol{\theta})) \ , \tag{2.5}$$

$$\dot{z}_t = -\tau \alpha_z z_t, \tag{2.6}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ are joint positions, velocities and accelerations, $\boldsymbol{g}$ is the goal attractor i.e., the desired end position of the trajectory and z_t is a phase variable. α and β are constants that define the spring-damping. Equation (2.6) is referred to as the *canonical system* because it models the generic behaviour of the system. Complexity of the desired movement is encoded in the forcing function $f(z_t, \boldsymbol{\theta})$, which is defined as a function of the canonical system using a weighted sum of N basis functions ϕ_i

$$f(z, \boldsymbol{\theta}) = \frac{\sum_{i=1}^{N} \phi_i(z)\theta_i}{\sum_{i=1}^{N} \phi_i(z)} z, \quad \phi_i(z) = \frac{\exp(-h_i(z - c_i)^2)}{\sum_{j=1}^{N} \exp(-h_j(z - c_j^2))}, \tag{2.7}$$

where ϕ_i defines a Gaussian centered at c_i with variances/bandwidths defined by h_i and the weighting of each Gaussian is given by θ_i. The forcing function is linear in the weights θ but non-linear in the phase z_t, i.e., $f(z, \boldsymbol{\theta}) = \Phi(z)^T \boldsymbol{\theta}$. With this representation we can easily generate trajectories and specify the desired movement of the robot.



(a) Basis functions

(b) Forcing function

(c) Trajectory

Figure 2.3: Components of a DMP.

2.4.2 Imitation Learning of DMPs

While it is possible for a robot to learn a control policy from scratch using reinforcement learning, in a real system, it is often undesirable to have the robot explore its way to the region of feasible solutions. Usually, it is easier to have the robot learn from *demonstrations* of the desired tasks shown by a teacher or an expert. That way, the initial search space of the robot would shrink to the region around where it observed the expert demonstrations. This process of learning from demonstrations is called *behavioural cloning* or *imitation learning* [Osa et al., 2018]. This method is used to *bootstrap* i.e., initialize the parameters $\boldsymbol{\theta}$ of the DMPs. In order to perform imitation learning, given is a demonstrated trajectory and its derivatives $\boldsymbol{q}_d$, $\dot{\boldsymbol{q}}_d$, $\ddot{\boldsymbol{q}}_d$, a goal $\boldsymbol{g}$, the constant DMP parameters α, β, α_z and a temporal scaling factor τ. By simply rearranging terms from equation (2.5), we can compute the forcing function and learn the weights by linear ridge regression

$$f(z,\boldsymbol{\theta}) = \frac{\ddot{\boldsymbol{q}}_d}{\tau^2} - \alpha\Big(\beta(\boldsymbol{g}-\boldsymbol{q}_d) - \frac{\dot{\boldsymbol{q}}_d}{\tau}\Big)\,, \tag{2.8}$$

$$\boldsymbol{\theta} = \Big(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}\Big)^{-1}\boldsymbol{\Phi}^T f\ . \tag{2.9}$$

Algorithm 2 Imitation Learning of DMPs

Input:
Desired trajectory $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$
Goal attractor $\boldsymbol{g}$
Parameters $\alpha, \beta, \alpha_z, \tau$

1: Compute forcing function for each timestep
2: $f(z,\boldsymbol{\theta}) = \frac{\ddot{\boldsymbol{q}}}{\tau^2} - \alpha\Big(\beta(\boldsymbol{g}-\boldsymbol{q}_d) - \frac{\dot{\boldsymbol{q}}}{\tau}\Big)$
3: Compute shape parameters by linear ridge regression
4: $\boldsymbol{\theta} = \Big(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \sigma^2\boldsymbol{I}\Big)^{-1}\boldsymbol{\Phi}^T f$

3 Policy Search Algorithms

In this chapter, three stochastic search episode based algorithms that all share the same objective (2.1) will be discussed. First, Relative Entropy Policy Search (REPS), an algorithm that bounds the KL divergence between subsequent updates, will be presented in Section 3.1. Section 3.2 will discuss a related algorithm, Model-based Relative Entropy (MORE), which introduces an additional bound on the entropy loss. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) an evolution strategy (ES) based approach will be presented in Section 3.3. Finally, Contextual Relative Entropy with Covariance Matrix Update (CREPS-CMA) a hybrid algorithm is presented. Each algorithm will be extended to the *contextual* version, that allows for more versatile learning.

3.1 Relative Entropy Policy Search (REPS)

REPS formulates the policy search problem as a constrained optimization problem [Boyd and Vandenberghe, 2004] and uses the KL, introduced in Section 2.3, as a distance metric. The problem is formulated as follows:

$$\begin{aligned} &\max_{\pi} \int \pi(\boldsymbol{\theta}) R(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}, \\ &\text{s.t. } \epsilon \geq \mathrm{KL}(\pi(\boldsymbol{\theta}) || q(\boldsymbol{\theta})), \\ &1 = \int \pi(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}, \end{aligned} \tag{3.1}$$

where the KL is given by $\int \pi(\boldsymbol{\theta}) \log(\pi(\boldsymbol{\theta})/q(\theta)) \mathrm{d}\boldsymbol{\theta}$. Putting it into plain words, the aim is to find a new policy π parameterized by $\boldsymbol{\theta}$ that maximizes the objective function $R(\boldsymbol{\theta})$ while staying close to the old policy q and having the policy as a proper distribution. The analytic solution for the policy update is derived using the method of Lagrangian multipliers (see Appendix 7.2), which yields the new policy in closed form as

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}) \exp\Big(\frac{R(\boldsymbol{\theta})}{\eta}\Big), \tag{3.2}$$

where η is the Lagrangian multiplier for the KL constraint, which is responsible for the exploration i.e., how far does the new search distribution move away from the current one to search for the optimal policy parameters. The dual function of the optimization problem (7.1)

$$g(\eta) = \eta\epsilon + \eta \log \int q(\boldsymbol{\theta}) \exp\Big(\frac{R(\boldsymbol{\theta})}{\eta}\Big) \, \mathrm{d}\boldsymbol{\theta} \ , \tag{3.3}$$

is then minimized to get the parameter η. Usually samples are used to estimate the integral

$$g(\eta) = \eta\epsilon + \eta \log \sum_i \frac{1}{N} \exp\Big(\frac{R(\boldsymbol{\theta}^{[i]})}{\eta}\Big) \mathrm{d}\boldsymbol{\theta} \ . \tag{3.4}$$

To estimate the new policy we need to fit a parametric distribution $\pi(\boldsymbol{\theta})$ to the samples $\boldsymbol{\theta}^{[i]}$ for which the rewards have been computed $R(\boldsymbol{\theta}^{[i]})$. This can be achieved using weighted maximum likelihood estimate on the samples, with weights given by $d^{[i]} = \exp(R(\boldsymbol{\theta}^{[i]})/\eta)$. The distribution $q(\boldsymbol{\theta})$ is not included in the weighting because the samples were already drawn from q. Since the search distribution is chosen to be a Gaussian, then the policy parameters to be updated are the mean and covariance. The new mean and covariance are given by

$$\mu_\pi = \frac{\sum_i^N d^{[i]} \boldsymbol{\theta}^{[i]}}{\sum_i^N d^{[i]}} \quad \Sigma_\pi = \frac{\sum_i^N d^{[i]} (\boldsymbol{\theta}^{[i]} - \boldsymbol{\mu}_\pi)(\boldsymbol{\theta}^{[i]} - \boldsymbol{\mu}_\pi)^T}{Z}, \tag{3.5}$$

where

$$Z = \frac{(\sum_i^N d^{[i]})^2 - \sum_i^N (d^{[i]})^2}{\sum_i^N d^{[i]}} \ , \tag{3.6}$$

is used to compute an unbiased estimator for the covariance matrix. Refer to Appendix 7.4 for derivation of weighted maximum likelihood estimate.

3.1.1 Contextual REPS

The problem formulation of REPS can be extended to multiple contexts [Kupcsik et al., 2013]. The variables that do not change for a given task but vary from one task to another give the notion of the context. For example, for the task of tetherball, the initial position of the ball and the starting position of the robot describe the context of one task but may vary in another. It is desirable to have the policy generalize to different contexts without re-learning the whole task i.e., adapt the parameters to the context. To select the parameters based on some context s the problem statement becomes

$$\begin{aligned} \max_{\pi} & \int \mu(\boldsymbol{s}) \int \pi(\boldsymbol{\theta}|\boldsymbol{s})\, R(\boldsymbol{s},\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}\, \mathrm{d}\boldsymbol{s}, \\ \text{s.t. } \epsilon \geq & \int \mu(\boldsymbol{s})\, \mathrm{KL}(\pi(\boldsymbol{\theta}|\boldsymbol{s})||q(\boldsymbol{\theta}|\boldsymbol{s}))\mathrm{d}\boldsymbol{s}\ , \\ \forall \boldsymbol{s}\ : 1 = & \int \pi(\boldsymbol{\theta}|\boldsymbol{s})\, \mathrm{d}\boldsymbol{\theta}, \end{aligned} \tag{3.7}$$

where $\mu(\boldsymbol{s})$ is the context distribution. This formulation however indicates that for each context $\boldsymbol{s}^{[i]}$, we have many parameter vector samples $\boldsymbol{\theta}^{[i,j]}$. Contextual REPS alleviates this problem by optimizing over the the joint probabilities $p(\boldsymbol{s},\boldsymbol{\theta}) = \mu(\boldsymbol{s})\pi(\boldsymbol{\theta}|\boldsymbol{s})$ under the condition that the marginal distribution $p(\boldsymbol{s}) = \int p(\boldsymbol{s},\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}$ reproduces the context distribution i.e., $\forall \boldsymbol{s} : p(\boldsymbol{s}) = \mu(\boldsymbol{s})$, so we can now use one sample per context $[\boldsymbol{s}^{[i]}, \boldsymbol{\theta}^{[i]}]$ as opposed to many parameters samples per context.
This formulation is still problematic however, because as a result of continuous spaces, we end up with an infinite amount of constraints. So, instead of matching the context distribution exactly, only feature expectations are matched i.e., The final formulation is given by

$$\begin{aligned} \max_{p} & \int\int p(\boldsymbol{s},\boldsymbol{\theta})\, R(\boldsymbol{s},\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{s}, \\ \text{s.t. } \epsilon \geq & \int\int \mathrm{KL}\left(p(\boldsymbol{s},\boldsymbol{\theta})||q(\boldsymbol{s},\boldsymbol{\theta})\right)\mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{s}\ , \\ \forall \boldsymbol{s}\ : 1 = & \int\int p(\boldsymbol{s},\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{s}, \\ \hat{\boldsymbol{\phi}} = & \int\int p(\boldsymbol{s},\boldsymbol{\theta})\phi(s)\mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{s}, \end{aligned} \tag{3.8}$$

where $\hat{\boldsymbol{\phi}}$ is the observed average feature vector. Once again, using the method of Lagrangian multipliers, the new policy is given by

$$p(\boldsymbol{s},\boldsymbol{\theta}) \propto \mu(\boldsymbol{s}) q(\boldsymbol{\theta}|\boldsymbol{s}) \exp\Big(\frac{R(\boldsymbol{s},\boldsymbol{\theta}) - V(\boldsymbol{s})}{\eta}\Big), \tag{3.9}$$

where $V(\boldsymbol{s}) = \phi(\boldsymbol{s})^T \boldsymbol{\nu}$ is the context-dependent baseline i.e., we subtract how good the reward is based on the context. For example, given the tetherball task, a reward for hitting and a penalty for joint movement and a context describing the initial position of the robot, if there were no baseline, a robot hitting the ball from a difficult initial position would get a lower reward than a robot positioned in such a way that it doesn't move a lot to reach the ball. It allows is to evaluate the parameter $\boldsymbol{\theta}$ independently from the context $\boldsymbol{s}$. The dual function for the optimizaton problem is

$$g(\eta, \boldsymbol{\nu}) = \eta\epsilon + \boldsymbol{\nu}^T \hat{\boldsymbol{\phi}} + \eta \log \int q(\boldsymbol{\theta}) \exp\Big(\frac{R(\boldsymbol{s},\boldsymbol{\theta}) - \boldsymbol{\nu}^T \phi(\boldsymbol{s})}{\eta}\Big)\, \mathrm{d}\boldsymbol{\theta}. \tag{3.10}$$

Similarly as in standard REPS, a new parametric distribution is obtained by performing a weighted maximum likelihood (ML) estimate. Typically, a Gaussian policy with a linear mean is used to represent the policy $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\varphi(\boldsymbol{s}), \boldsymbol{\Sigma})$ [1]. The weighted ML solution is given by

$$\boldsymbol{K} = (\boldsymbol{\Phi}\boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{D}\boldsymbol{\Theta}\ , \tag{3.11}$$

where $\boldsymbol{\Phi}$ is the feature vector, $\boldsymbol{D}$ is the diagonal weighting matrix with weights given by $d^{[i]} = \exp(R^{[i]} - V(\boldsymbol{s}^{[i]})/\eta)$ and the covariance matrix is the same as in equation (3.5).

[1] Note that $\phi(\boldsymbol{s})$ is typically different from $\varphi(\boldsymbol{s})$.

Algorithm 3 Contextual REPS

while not converged **do**
  for i=1 .. N **do**
    Observe context $\boldsymbol{s}_i \sim \mu(\boldsymbol{s})$
    Draw parameters $\boldsymbol{\theta}_i \sim \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\varphi(\boldsymbol{s}_i), \boldsymbol{\Sigma})$
    Evaluate parameters and get $R(\boldsymbol{s}_i, \boldsymbol{\theta}_i)$
  end for
  Optimize dual function (3.10) to get η and $\boldsymbol{\nu}$
  Update policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ using Weighted Maximum Likelihood Estimate
end while

3.2 Model-based Relative Entropy (MORE)

In the previous section, REPS was introduced along with the idea of information loss bounding using the KL divergence. Despite its simplicity and ease of use, REPS uses only the values of the objective function which tend to be very noisy, require an exhaustive amount of evaluations for the objective/ need many samples to approximate KL and suffers from premature convergence [Abdolmaleki et al., 2015a], [Stulp and Sigaud, 2012]. Proposed by [Abdolmaleki et al., 2015b], MORE is also a black-box optimizer in which the objective function is locally approximated by a quadratic model and that model is used to locally maximize objective function. On top of that it adds a lower bound constraint on the entropy of the new distribution to avoid the problem of premature convergence caused by a collapsing variance. Similar to REPS the problem is formulated as a constrained optimization problem with the addition of the entropy bound

$$\begin{aligned} &\max_{\pi} \int \pi(\boldsymbol{\theta}) R(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}, \\ &\text{s.t.} \quad \epsilon \geq \mathrm{KL}(\pi(\boldsymbol{\theta}) || q(\boldsymbol{\theta})), \\ &\qquad \beta \leq H(\pi), \\ &1 = \int \pi(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} \ , \end{aligned} \tag{3.12}$$

where $H(\pi) = -\int \pi(\boldsymbol{\theta}) \log(\pi(\boldsymbol{\theta})) \, \mathrm{d}\boldsymbol{\theta}$ is the entropy of π. The upper-bound for the KL ϵ is not changed during the learning process, while β is updated every iteration as

$$\beta = \gamma(\mathbb{E}[H(q)] - H(\pi_0)) + H(\pi_0) \ , \tag{3.13}$$

where γ specifies the percentage of the relative difference between the old policy's expected entropy and the minimal entropy $H(\pi_0)$ set for the policy distribution. Given the above optimization problem, the new policy is given by

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})^{\frac{\eta}{\eta+\omega}} \exp\Big(\frac{R(\boldsymbol{\theta})}{\eta+\omega}\Big), \tag{3.14}$$

where $\eta > 0$ and $\omega > 0$ are the Lagrangian multipliers for the KL and entropy constraint respectively. This would be the same formulation for REPS if $\omega = 0$. The dual function given by

$$g(\eta, \omega) = \eta\epsilon - \omega\beta + (\eta+\omega) \log \int q(\boldsymbol{\theta})^{\frac{\eta}{\eta+\omega}} \exp\Big(\frac{R(\boldsymbol{\theta})}{\eta+\omega}\Big) \, \mathrm{d}\boldsymbol{\theta} \ , \tag{3.15}$$

is minimized to get η and ω. For the quadratic model as

$$R(\boldsymbol{\theta}) \approx \boldsymbol{\theta}^T \boldsymbol{R} \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{r} + r_0, \tag{3.16}$$

and a Gaussian search distribution $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ we end up with a closed form solution for equation (3.15), (see full derivation in Appendix 7.3)

$$g(\eta, \omega) = \eta\epsilon - \omega\beta + \frac{1}{2}\Big(\boldsymbol{f}^T \boldsymbol{F} \boldsymbol{f} - \eta \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} - \eta \log|2\pi\boldsymbol{\Sigma}| + (\eta+\omega)\log|2\pi(\eta+\omega)\boldsymbol{F}|\Big), \tag{3.17}$$

where $\boldsymbol{F} = (\eta\boldsymbol{\Sigma}^{-1} - 2\boldsymbol{R})^{-1}$ and $\boldsymbol{f} = \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{r}$. The policy update is

$$\pi\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{F}\boldsymbol{f}, (\eta+\omega)\boldsymbol{F}) \tag{3.18}$$

3.2.1 Contextual-MORE

For versatile task objectives, once again we would like to incorporate some context into the problem statement. The problem formulation remains the same, with the addition of a context distribution over the search distribution as described in [Tangkaratt et al., 2017]

$$\begin{aligned} \max_{\pi} & \int\int \mu(\boldsymbol{s})\pi(\boldsymbol{\theta}|\boldsymbol{s})R(\boldsymbol{\theta},\boldsymbol{s})\,\mathrm{d}\boldsymbol{\theta}\,\mathrm{d}\boldsymbol{s}, \\ \text{s.t.} \quad \epsilon \geq & \int \mu(\boldsymbol{s})\mathrm{KL}(\pi(\boldsymbol{\theta}|\boldsymbol{s})||q(\boldsymbol{\theta}|\boldsymbol{s}))\,\mathrm{d}\boldsymbol{s}, \\ \beta \leq & H(\pi), \\ 1 = & \int\int \mu(\boldsymbol{s})\pi(\boldsymbol{\theta}|\boldsymbol{s})\,\mathrm{d}\boldsymbol{\theta}\,\mathrm{d}\boldsymbol{s}\ , \end{aligned} \tag{3.19}$$

where $\mu(\boldsymbol{s})$ is the distribution over the context and β is defined similarly as in standard MORE (3.13). Unlike the contextual REPS formulation, where the optimization is done over the joint distribution $p(\theta,s)$, CMORE optimizes over the conditional distribution $\pi(\boldsymbol{\theta}|\boldsymbol{s})$, which implies that CMORE assumes if the search distribution does not change so much in consecutive updates i.e., KL is bounded, then the context distribution also does not drastically change between bounded updates.
The quadratic model in terms of $\boldsymbol{\theta}$ and $\boldsymbol{s}$ is approximated by

$$R(\boldsymbol{\theta},\boldsymbol{s}) \approx \boldsymbol{\theta}^T\boldsymbol{R}_{\theta\theta}\boldsymbol{\theta} + \phi(\boldsymbol{s})^T\boldsymbol{R}_{ss}\phi(\boldsymbol{s}) - 2\boldsymbol{\theta}^T\boldsymbol{R}_{\theta s}\phi(\boldsymbol{s}) + \boldsymbol{\theta}^T\boldsymbol{r}_1 + \phi(\boldsymbol{s})^T\boldsymbol{r}_2 + r_0 \tag{3.20}$$

where $\boldsymbol{R}_{\theta\theta}$, $\boldsymbol{R}_{ss}$, $\boldsymbol{R}_{\theta s}$, $\boldsymbol{r_1}$, $\boldsymbol{r_2}$ and r_0 are the model parameters [2]. The matricies $\boldsymbol{R}_{\theta\theta}$, $\boldsymbol{R}_{ss}$ are symmetric, $\boldsymbol{R}_{\theta\theta}$ is enforced to be negative definite and $\phi(\boldsymbol{s})$ denotes the features of the context $\boldsymbol{s}$. The policy distribution is a Gaussian with a linear mean

$$q(\boldsymbol{\theta}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\phi(\boldsymbol{s}) + \boldsymbol{b}, \boldsymbol{\Sigma}). \tag{3.21}$$

The dual function for (3.19) is expressed as

$$\begin{aligned} g(\eta,\omega) = \eta\epsilon - \omega\beta + \frac{1}{2}\Big(\boldsymbol{f}^T\boldsymbol{F}\boldsymbol{f} - \eta\boldsymbol{b}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{b} + (\eta+\omega)\log|2\pi\boldsymbol{F}(\eta+\omega)| - \eta\log|2\pi\boldsymbol{\Sigma}|\Big) \\ + \int \mu(\boldsymbol{s})\Big(\phi(\boldsymbol{s})^T\boldsymbol{m} + \frac{1}{2}\phi(\boldsymbol{s})^T\boldsymbol{M}\phi(\boldsymbol{s})\Big)d\boldsymbol{s} \end{aligned} \tag{3.22}$$

where

$$\begin{aligned} \boldsymbol{f} &= \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{b} + \boldsymbol{r}_1, \\ \boldsymbol{F} &= (\eta\boldsymbol{\Sigma}^{-1} - 2\boldsymbol{R})^{-1}, \\ \boldsymbol{m} &= \boldsymbol{L}^T\boldsymbol{F}\boldsymbol{L} - \eta\boldsymbol{K}\boldsymbol{\Sigma}^{-1}\boldsymbol{b}, \\ \boldsymbol{M} &= \boldsymbol{L}^T\boldsymbol{F}\boldsymbol{L} - \eta\boldsymbol{K}\boldsymbol{\Sigma}^{-1}\boldsymbol{K}, \\ \boldsymbol{L} &= \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{K} + 2\boldsymbol{R}_{\theta s}. \end{aligned} \tag{3.23}$$

The expectation in Equation (3.22) is approximated by sample averages since the context distribution $\mu(\boldsymbol{s})$ is unknown. The new policy is given by

$$\pi(\boldsymbol{\theta}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{F}\boldsymbol{L}\phi(\boldsymbol{s}) + \boldsymbol{F}\boldsymbol{f}, \boldsymbol{F}(\eta+\omega)), \tag{3.24}$$

with $\boldsymbol{f}$, $\boldsymbol{F}$ and $\boldsymbol{L}$ given in (3.23).

[2] When learning the model parameters using linear regression, after $\boldsymbol{R}_{\theta\theta}$ is enforced to be negative definite, the remaining model parameters need to be re-learned.

Algorithm 4 Contextual MORE

while not converged **do**
  for i=1 .. N **do**
    Observe context $\boldsymbol{s}_i \sim \mu(\boldsymbol{s})$
    Draw parameters $\boldsymbol{\theta}_i \sim \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\phi(\boldsymbol{s}_i), \boldsymbol{\Sigma})$
    Evaluate parameters and get $R(\boldsymbol{\theta}_i, \boldsymbol{s}_i)$
  end for
  Fit quadratic model (3.20)
  Optimize dual function (3.22)
  Update policy $\pi(\boldsymbol{\theta}|\boldsymbol{s})$ (3.24)
end while

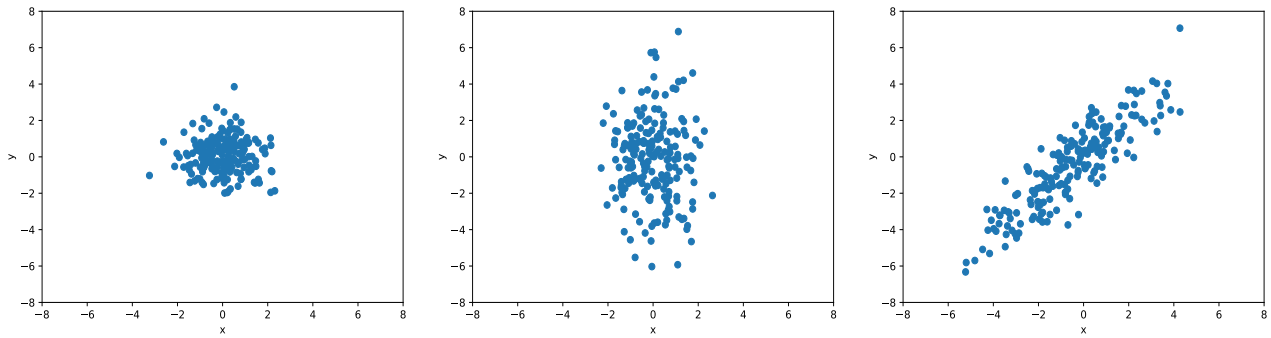
3.3 Covariance Matrix Adaptation Evolution Strategies (CMA-ES)

CMA-ES [Auger and Hansen, 2012] is yet another stochastic search algorithm that descends from a class of algorithms called evolution strategies (ES) [Rechenberg, 1978]. Inspired by the principle of biological evolution, the main idea of ES is to generate a population of candidate solutions, evaluate the candidates' *fitness*, then perform *selection*; choose the fittest individuals as parents for the next generation, *mutate* the current distribution and get the offspring of the next generation from the selected parents. Like most stochastic search algorithms, CMA-ES aims to find the optimal policy parameters that maximize the likelihood of favoured observations i.e., increase the probability of successful solutions. Furthermore, CMA-ES exploits the correlation between consecutive steps captured by the *evolution paths* to adapt the step-size and efficiently update the covariance matrix of the distribution.

The Covariance Matrix: Typically, stochastic search algorithms maintain a Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{m}, \boldsymbol{\Sigma})$ [3] on individuals where $\boldsymbol{m}$ denotes the mean and $\boldsymbol{\Sigma}$ denotes the covariance. The covariance matrix is responsible for the exploration of the search distribution. It determines the shape of the distribution as demonstrated in Figure 3.1. The covariance matrix is defined as

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \dots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \dots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{m1} & \sigma_{m2} & \dots & \sigma_{mn} \end{bmatrix},$$

where the diagonals denote the variance of the corresponding random variables i.e., how far variables change from the mean, and the off-diagonal entries capture the *covariance* between each pair of variables; i.e., how the variables are correlated to each other.



(a) $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (b) $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$ (c) $\boldsymbol{\Sigma} = \begin{bmatrix} 5 & 4 \\ 4 & 6 \end{bmatrix}$

Figure 3.1: An illustration of how the shape of a 2-dimensional Gaussian distribution is determined by the covariance matrix. Figure 3.1a, shows equal horizontal (along x-axis) and vertical (along y-axis) spreads of data points and no correlation between the two variables x and y. Figure 3.1b captures a larger vertical spread, hence the elongated shape. Figure 3.1c a symmetric correlation of between x and y, i.e., if the value of x increases the value of y also increases.

[3] The mean is denoted by $\boldsymbol{m}$ instead of $\boldsymbol{\mu}$ to avoid confusion with the algorithm's terminology.

The basic sampling equation from which new offspring (search points) is/are generated is

$$\boldsymbol{x}_k^{(g+1)} \sim \boldsymbol{m}^{(g)} + \sigma^{(g)} \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}^{(g)}) \quad \text{for } k = 1, ..., \lambda \,, \tag{3.25}$$

where g denotes the generation, $\boldsymbol{m} \in \mathbb{R}^n$ denotes the mean, λ denotes the population size/ number of offspring, $\boldsymbol{\Sigma} \in \mathbb{R}^{nxn}$ denotes the covariance matrix of the multivariate search distribution and $\sigma \in \mathbb{R}_+$ is the step-size/standard deviation. The number of selected individuals/parents is denoted by μ. The equation shows that the new search points are sampled as normally distributed perturbations of $\boldsymbol{m}$. There are two main strategies for selection, the plus/elitist selection $(\mu/\mu + \lambda)$, where μ parents are selected from $\lambda + \mu$ individuals, and comma/non-elistist selection $(\mu/\mu, \lambda)$, where μ parents are selected only from the offspring λ. The latter will be used for the remainder of this section.

Fitness Transformation: CMA-ES ranks the samples/offspring based on their fitness using a rank preserving transformation of the fitness values. First the samples are sorted ascendingly [4]based on their fitness values, such that $f(\boldsymbol{x}_{1:\lambda}) \leq .. \leq f(\boldsymbol{x}_{\lambda:\lambda})$ and the weight of the j^{th} best sample is set to

$$w_j = \ln(\mu + \frac{1}{2}) - \ln(j) \,, \tag{3.26}$$

such that all weights sum to 1. This transformation makes the algorithm invariant under monotone transformations of the fitness function [Hansen, 2016] rendering it quite robust.

To update the mean of the population, the solution to the weighted maximum likelihood estimate is used and the new mean is given by

$$\begin{aligned} \boldsymbol{m}^{(g+1)} &= \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}, \\ w_1 \geq w_2 \geq ... \geq w_\mu > 0, &\quad \sum_{i=1}^{\mu} w_i = 1 \,. \end{aligned} \tag{3.27}$$

The step-size σ in CMA-ES is equivalent to the learning rate of the algorithm. There are various methods for step-size control in ES, such as 1/5 success rule [Vent, 1975], which increases the step size if more than 20% of the new solutions are successful and decrease it other, σ-self-adaptation [Schwefel, 1987] and path length control, also referred to as cumulative step-size control or Cumulative Step-length Adaptation (CSA) [Hansen and Ostermeier, 2001], which is used for this algorithm.

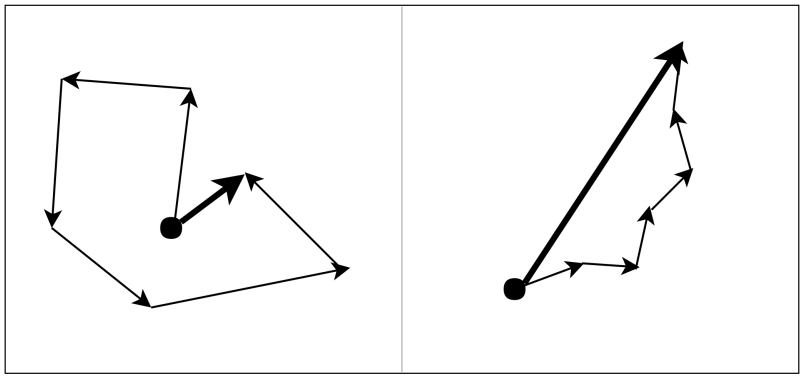


Figure 3.2: An illustration of step-size control in effect. Left figure shows the scenario where the updates are uncorrelated. In that case the step size should be decreased as shown by the bold arrow. Right figure shows updates all in the same direction i.e., correlated, in that case the step size should be increased.

Evolution Paths: The crux of this algorithm lies in the usage of *evolution paths*. Evolution paths store information about consecutive update steps and allow the algorithm to exploit the correlation or decorrelation between update steps to adapt its parameters. The idea is that if two consecutive update steps are towards the same direction i.e., they are correlated, the updates add up. Likewise, if the two updates are in opposite directions, the updates cancel out. Exploiting this correlation speeds up convergence time of the algorithm [Hansen, 2016].

[4] In coherence with the referenced papers, a minimization objective is assumed for this algorithm.

Step-size control is used to exploit the correlation between consecutive updates in order to adapt the step-size. As previously mentioned, if two consecutive updates are correlated, then the step size should increase. Similarly, if the two updates are in opposite directions, the step-size should be decreased. This is depicted in Figure 3.2. In order to achieve that, the step-size adaptation control of CMA-ES uses the evolution path $\boldsymbol{p}_\sigma$. The evolution path for step-size control is given by

$$\begin{aligned} \boldsymbol{p}_\sigma^{(g+1)} &= (1-c_\sigma)\boldsymbol{p}_\sigma^{(g)} + \sqrt{c_\sigma(2-c_\sigma)\mu_{\text{eff}}}\, \boldsymbol{\Sigma}^{(g)-\frac{1}{2}} \frac{\boldsymbol{m}^{(g+1)}-\boldsymbol{m}^{(g)}}{\sigma^{(g)}}, \\ \mu_{\text{eff}} &\approx \frac{1}{\sum_{i=1}^{\mu} w_i^2}, \quad c_\sigma = \frac{\mu_{\text{eff}}+2}{n+\mu_{\text{eff}}+5}, \end{aligned} \tag{3.28}$$

where $c_\sigma < 1$, the term $(1-c_\sigma)$ can be interpreted as a discount factor for the previous evolution path $\boldsymbol{p}_\sigma^{(g)}$, μ_{eff} is the number of effective samples [5]. The step size is then given by

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||\boldsymbol{p}_\sigma^{(g+1)}||}{\mathbb{E}||\mathcal{N}(0,\boldsymbol{I})||}\right)\right), \tag{3.29}$$

where d_σ is a damping parameter that determines how much the step-size can vary and $||X||$ is the Euclidean norm of the vector $X^n = \sqrt{X_1^2 + X_2^2 + .. + X_{(n)}^2}$.
The covariance matrix update consists of two parts: the *rank-μ* update and the *rank-one* update. Rank-one update considers the change of population mean over time as encoded in the evolution path $\boldsymbol{p}_c$ i.e., it morphs the variance of the distribution in the direction of previous successful steps. Rank-μ update incorporates information about the successful steps in the previous generation (g). The evolution path $\boldsymbol{p}_c$ used by the rank-one update is given by

$$\boldsymbol{p}_c^{(g+1)} = (1-c_c)\boldsymbol{p}_c^{(g)} + \sqrt{c_c(2-c_c)\mu_{\text{eff}}}\, \frac{\boldsymbol{m}^{(g+1)}-\boldsymbol{m}^{(g)}}{\sigma^{(g)}}. \tag{3.30}$$

The full covariance matrix update is

$$\begin{aligned} \boldsymbol{\Sigma}^{(g+1)} &= (1-c_1-c_\mu)\boldsymbol{\Sigma}^{(g)} + \underbrace{c_1 \boldsymbol{p}_c \boldsymbol{p}_c^T}_{rank-one} + \underbrace{c_\mu \sum_{i=1}^{\mu} w_i \frac{x_{i:\lambda}-m^{(g)}}{\sigma^{(g)}}\left(\frac{x_{i:\lambda}-m^{(g)}}{\sigma^{(g)}}\right)^T}_{rank-\mu}, \\ c_1 &= \frac{2}{(n+1.3)^2+\mu_{\text{eff}}}, \quad c_\mu = \min\left(1-c_1, \frac{2(\mu_{\text{eff}}-2+1/\mu_{\text{eff}})}{(n+2)^2+\mu_{\text{eff}}}\right), \end{aligned} \tag{3.31}$$

where c_1 is the learning rate for the rank-one update, c_μ is the learning rate for the rank-μ update.

TR-CMA-ES: It has been noted that the update rules of CMA-ES are based on heuristics. In recent works [Akimoto et al., 2012], [Ollivier et al., 2017], some theoretical foundation has been set to justify those rules. However, it was not all set in a single mathematical framework. A new algorithm called Trust Regions for Covariance Matrix Adaptation (TR-CMA-ES) [Abdolmaleki et al., 2017a] fully derives the update rules of CMA-ES in the Expectation-Maximization framework, that optimizes a well defined objective i.e., maximizing the lower bound. By adding a trust region (KL bound) to the optimization of the lower bound in EM, this approach reaches the same update rules for the mean and the covariance matrix as CMA-ES, but reaches a different update rule for the step size, which yields more promising results.

3.3.1 Contextual CMA-ES

CMA-ES is a robust stochastic search algorithm with many favourable properties such as premature convergence avoidance, multiple invariance properties and adaptive learning rates [Hansen, 2016]. It is regarded as one of the most efficient evolutionary algorithms [Beyer, 2007]. Contextualizing this algorithm would allow us to inherit its beneficial properties while gaining a more adaptive task learning algorithm. The contextual version of CMA-ES [Abdolmaleki et al., 2017b] adopts a similar approach in contextualization as in contextual REPS i.e., using a linear Gaussian policy for the search distribution and subtracting a context-dependent baseline from the rewards before assigning weights to samples.

[5] There are well established heuristics for setting parameters for CMA-ES, refer to [Hansen, 2016] for more details.

Just like the standard version, CCMAES ranks samples based on their fitness values. With the introduction of the context however, the fitness values first need to be rectified in order to be able to rank the samples solely based on their fitness independent of the quality of the context $\boldsymbol{s}$. This is achieved by subtracting a context-dependent baseline $V(\boldsymbol{s})$ from the fitness R to obtain the *advantage*

$$A_k = R_k - V(\boldsymbol{s_k}). \tag{3.32}$$

The baseline function $V(\boldsymbol{s})$ can be interpreted as a value function that gives the expected return of the search distribution for a given context. It can be obtain by linear ridge regression as

$$\begin{aligned} \boldsymbol{\beta} &= \left(\boldsymbol{\phi}(\boldsymbol{s})^T \boldsymbol{\phi}(\boldsymbol{s}) + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\phi}(\boldsymbol{s})^T R, \\ V(\boldsymbol{s}) &= \boldsymbol{\beta}^T \phi(\boldsymbol{s}) \,, \end{aligned} \tag{3.33}$$

where $\boldsymbol{\phi}(\boldsymbol{s})$ are the context features, λ is the regularization factor for the ridge regression and R are the fitness values for the samples.

Algorithm 5 Contextual CMA-ES

Input: parameter dimensionality n, context dimensionality n_s, number of offspring λ, number of parents μ

Initialize $\boldsymbol{K}, \sigma^{(g=0)}, \boldsymbol{p}_{\boldsymbol{\sigma}}^{(g=0)}, \boldsymbol{p}_{\boldsymbol{c}}^{(g=0)}, \boldsymbol{\Sigma}^{(g=0)}$
while stopping criterion not met **do** [6]
  for k = 1, ... , λ **do**
    Observe context $\boldsymbol{s}_k$
    Compute mean $\boldsymbol{m}(\boldsymbol{s}_k) = \boldsymbol{K}\varphi(\boldsymbol{s}_k)$
    Draw samples $\boldsymbol{\theta_k} = \boldsymbol{m}(\boldsymbol{s}_k) + \sigma\, \mathcal{N}(\boldsymbol{0}, \Sigma^{(g)})$
    Evaluate fitness $R_k = f(\boldsymbol{\theta_k})$
  end for
  Compute baseline $V(s)$ **for fitness** Eq. (3.33)
  Compute weights $d^{[k]}$ **for** $\{\boldsymbol{s_k}, \boldsymbol{\theta_k}, A_k\}$ Eq. (3.26)
  Set hyperparameters $\mu_{\text{eff}}, c_1, c_\mu, c_c, c_\sigma, d_\sigma$
  Update mean function

$$\boldsymbol{K}^{(g+1)} = (\boldsymbol{\Phi D \Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{D \Theta}$$

  Update evolution paths

$$\begin{aligned} \boldsymbol{y} &= \frac{\boldsymbol{K}^{(g+1)}\hat{\varphi} \;-\; \boldsymbol{K}^{(g)^T}\hat{\varphi}}{\sigma^{(g)}} \\ p_c^{(g+1)} &= (1-c_c)p_c^{(g)} + h_\sigma \sqrt{c_c(2-c_c)\mu_{\text{eff}}}\;\; \boldsymbol{y} \\ p_\sigma^{(g+1)} &= (1-c_\sigma)p_\sigma^{(g)} + \sqrt{c_\sigma(2-c_\sigma)\mu_{\text{eff}}}\;\; \boldsymbol{\Sigma}^{(g)\,\frac{-1}{2}}\, \boldsymbol{y} \end{aligned}$$

  Update step size

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||\boldsymbol{p}_\sigma^{(g+1)}||}{\mathbb{E}||\mathcal{N}(0,\boldsymbol{I})||}\right)\right)$$

  Update covariance matrix

$$\begin{aligned} \boldsymbol{S} &= \tfrac{1}{\sigma^2}\textstyle\sum_{i=1}^{\mu} d_k(\boldsymbol{\theta}_k - \boldsymbol{K}^{(g)}\varphi(\boldsymbol{s}_k))(\boldsymbol{\theta}_k - \boldsymbol{K}^{(g)}\varphi(\boldsymbol{s}_k))^T \\ \boldsymbol{\Sigma}^{(g+1)} &= (1-c_1-c_\mu)\boldsymbol{\Sigma}^{(g)} + \underbrace{c_\mu \boldsymbol{S}}_{rank-\mu} + \underbrace{c_1 \boldsymbol{p}_c^{(g+1)} \boldsymbol{p}_c(g+1)^T}_{rank-one} \end{aligned}$$

end while

3.4 Contextual Relative Entropy Policy Search with Covariance Matrix Adaptation (CREPS-CMA)

Some stochastic search algorithms (ex: REPS) use a weighted maximum likelihood estimate (WMLE), which can be an unreliable estimator when the task at hand has a few number of samples and high dimensionality. This causes the WMLE to overfit the covariance matrix making the entropy/variance collapse quickly, biasing the search distribution towards a certain region. This loss of exploration causes the algorithm to converge prematurely. This issue of premature convergence has been addressed by several works [Abdolmaleki et al., 2015a], [Stulp and Sigaud, 2012] that have proposed

[6] Refer to [Hansen, 2016] for suggested stopping criteria.

various methods to alleviate this problem. Contextual Relative Entropy Policy Search with Covariance Matrix Adaptation is a hybrid algorithm proposed by [Abdolmaleki et al., 2016]. This algorithm combines the light weight advantages of contextual REPS while addressing the problem of premature convergence by using a CMA-ES inspired update for the covariance matrix.

Inspired by CMA-ES, the old covariance matrix is combined with the sample covariance matrix from Equation (3.5) such that

$$\boldsymbol{\Sigma} = (1-\lambda)\boldsymbol{\Sigma}_q + \lambda \boldsymbol{S}, \tag{3.34}$$

where $\lambda \in [0,1]$ and it balances the old covariance matrix $\boldsymbol{\Sigma}_q$ and the sample covariance matrix $\boldsymbol{S}$. It is set as

$$\lambda = \min\Big(1, \frac{\phi_{\text{eff}}}{p^2}\Big), \quad \phi_{\text{eff}} = \frac{1}{\sum_{i=1}^{N}(d^{[i]})^2}, \tag{3.35}$$

where p is the dimensionality of parameters and $d^{[i]}$ are the weights computed for each sample as in Section 3.1.1. This update reduces the entropy of the distribution by a certain amount and scales it by the number of effective samples ϕ_{eff}. The intuition is, in the early stages of the search distribution, there is a low number of effective samples i.e., ones with high weights, so information retained in the old covariance matrix is needed, therefore it is combined with the sample covariance matrix. On the other hand, when the search distribution is close to convergence, the number of effective samples increases so it is sufficient to use only the sample covariance matrix by setting λ to 1.

Algorithm 6 Contextual REPS-CMA

Given data set $\mathcal{D}$ of parameters $\boldsymbol{\theta}$, context $\boldsymbol{s}$ and rewards $\boldsymbol{R}$

1: **Compute weights** $d^{[i]}$ **for data set** $\mathcal{D}$
2: **Optimize dual function to get optimum** η and $\boldsymbol{v}$
$$g(\eta, \boldsymbol{v}) = \eta\epsilon + \boldsymbol{v}^T\hat{\boldsymbol{\phi}} + \eta \log \int q(\boldsymbol{\theta}) \exp\Big(\frac{\boldsymbol{R}(\boldsymbol{s},\boldsymbol{\theta}) - \boldsymbol{v}^T\phi(\boldsymbol{s})}{\eta}\Big)$$
3: **Normalize weights**
$\sum_{i=1}^{N} d^{[i]} = 1$
4: **Compute mean**
$\boldsymbol{K} = (\boldsymbol{\Phi}\boldsymbol{D}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{D}\boldsymbol{\Theta}$
$\boldsymbol{\mu} = \boldsymbol{K}^T\phi(\boldsymbol{s})$
5: **Compute sample covariance** $\boldsymbol{S}$
$\boldsymbol{S} = \sum_{i=1}^{N} d^{[i]}(\boldsymbol{\theta}^{[i]} - \boldsymbol{\mu})(\boldsymbol{\theta}^{[i]} - \boldsymbol{\mu})^T / Z$
6: **Compute parameters** λ **and** ϕ_{eff} **for covariance matrix update** Eq. (3.35)
7: **Compute new covariance matrix**
$\boldsymbol{\Sigma} = (1-\lambda)\boldsymbol{\Sigma}_q + \lambda\boldsymbol{S}$

3.5 Covariance Estimation with Controlled Entropy Reduction (CECER)

Similar to the previous algorithm, CECER [Abdolmaleki et al., 2015a] offers another way to regularize the covariance matrix update in REPS that also linearly interpolates the old covariance $\boldsymbol{\Sigma}_q$ matrix with the sample covariance matrix $\boldsymbol{S}$

$$\boldsymbol{\Sigma} = \lambda\boldsymbol{\Sigma}_q + (1-\lambda)\boldsymbol{S}, \tag{3.36}$$

where $\lambda \in [0,1]$ is an interpolation factor that is chosen to reduce the entropy by a controlled amount ΔH. The entropy reduction is scaled by the number of effective samples as follows

$$\phi_{\text{eff}} = \frac{1}{\sum_{i=1}^{N}(d^{[i]})^2}, \quad \Delta H = \alpha\phi_{\text{eff}}, \tag{3.37}$$

where $d^{[i]}$ defines the sample weights of REPS, and α is a constant that determines the amount of entropy reduction. The idea is the more number of effective samples there is, the more the entropy can be reduced, i.e., we can rely more on the sample covariance matrix $\boldsymbol{S}$. The choice of λ is done to satisfy the following equality

$$H(\boldsymbol{\Sigma}_q) - H(\lambda\boldsymbol{\Sigma}_q + (1-\lambda)\boldsymbol{S}) = \Delta H. \tag{3.38}$$

In order to find λ, direct search is used.

4 Setup and Implementation

In this chapter, the full details of the setup will be introduced. First, the task objective and model will be discussed. The components of the software framework will be then presented. Then the modules of the software framework will be throughly explained.

4.1 The Tetherball Task and Setup

Using a game-based task is convenient for benchmarking, since the performance i.e., game score, provides a simple success measure for comparison. In a typical two-player tetherball game, two players stand facing each other and a pole is centered between them. A ball is attached to the top of the pole using a rope. A player scores if he/she hits the ball without the opponent unwinding it. This is the full setup that was presented in [Parisi et al., 2015] to compare a programmed player to a *learned* opponent. The task setup in this thesis follows a slightly simpler setup. First of all, a single-player setup is chosen, so a single BioRob is used in simulation. In the single-player setup, the robot scores a point for every time it hits the ball. Additionally, only four degrees of freedom of the robot out of the six are controlled. The remaining two joints almost nearly do not move in this task, so they are ignored for the sake of shrinking the parameter space of the policy. A simpler model is used for the ball; a pendulum setup is used as opposed to pole setup (see Figure 4.1) , where there is an additional height parameter (distance of rope wind around the pole). To have a non-trivial task, the ball starts at a random position and velocity determined by φ, θ, $d\varphi$ and $d\theta$ in the spherical coordinate system. These would be our *context* variables in the state. The state for the MDP of the task is $\boldsymbol{x} = \{\boldsymbol{q}, \boldsymbol{b}, \dot{\boldsymbol{b}}\}$, where $\boldsymbol{q} \in \mathbb{R}^4$ denotes the joint positions and $\boldsymbol{b}, \dot{\boldsymbol{b}} \in \mathbb{R}^4$ are the ball positions and velocities respectively. The control actions $\boldsymbol{u} \in \mathbb{R}^4$ are the torques generated by the controller given the desired joint positions.

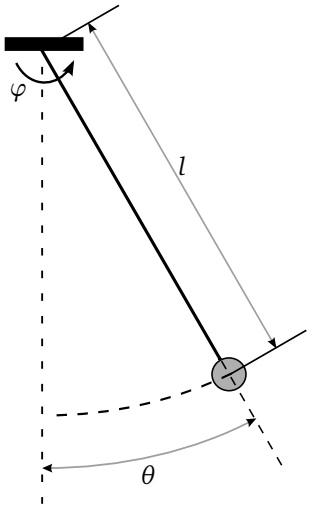


Figure 4.1: Diagram of the pendulum setup with ball position in spherical coordinates. The initial values of φ, θ, $d\varphi$ and $d\theta$ determine the context for the task, while l remains constant. Figure from [Abdulsamad et al., 2014].

4.2 The Software

The Simulator: The most essential component of the software is SL the simulator. Simulation Lab (SL) is an OpenGL based physical simulation environment for robotics, originally developed by Stefan Schaal and colleagues at USC [Schaal, 2009]. SL is particularly useful in simulation mode, to simulate tasks on the robot before implementing them on the real system, thereby preventing any harm to the real robot. It provides a rigid-body physics simulation environment that accurately models the real system. It is implemented using the C programming language.

Robcom: Robcom is an API that communicates with SL in a variety of languages/frameworks such as Matlab, C++, Python and ROS. For the scope of this project, the Python API is used. It eases the implementation of some functionalities

such as recording demonstrations from the simulator and trajectory execution as will be shown in the next section.

The Shared Memory Interface: Offered by SL, the shared memory interface provides another way for SL to communicate with Matlab and Python. Similar to Robcom, it is also used to send trajectories to SL. On top of that, it can receive replies specific to each task. It is also easier to send and receive arbitrary data using its predefined set of buffers.

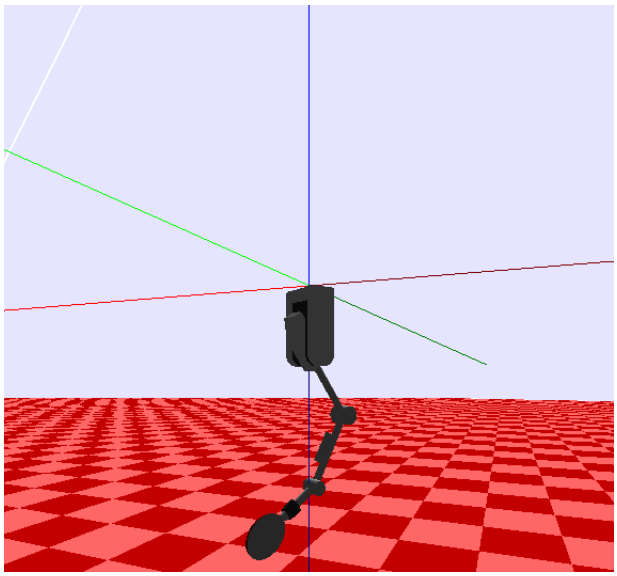


Figure 4.2: The Biorob robot in the simulation environment SL.

4.3 Implementation Details

Demonstrations Recording: Demonstration data is essential for policy initialization, which will be discussed shortly. The aim is to save trajectory data starting with the ball being released, the robot moving in attempt to hit the ball and finally the robot going back to its starting position. The initial position and velocity of the ball are also recorded and saved because they define the context of the task, which is needed for imitation learning. To obtain these trajectories, the simulation of the programmed player from [Parisi et al., 2015] is used. For some initial position and velocity of the ball, it estimates the ball's trajectory using ordinary differential equations (ODEs). The robot player then uses this prediction of the ball's trajectory to find a hitting point. With the desired hitting point, a low-level controller solves a constrained inverse kinematics problem and generates the desired trajectory. Fifth order polynomials are used to represent the trajectory, which is then followed using a PD controller with gravity compensation. SL's utility functions, such as *invKin* and *ias_calculate_min_jerk_next_step_single_joint*, are used to attain a robust programmed player. Robcom is used to record the demonstrations.

Imitation Learning and Policy Initialization: To speed up the learning process, it is often useful to narrow down the initial search space by initializing it around where the optimal parameters are, i.e., not have the robot start very far off from the optimal solution. In order to achieve that, imitation learning, discussed in 2.4.2, is used. The data saved from the recorded demonstrations stores the joint positions $\boldsymbol{q}$, velocities $\dot{\boldsymbol{q}}$ and accelerations $\ddot{\boldsymbol{q}}$, which are needed for imitation learning of the DMP parameters. The goal g is set to the initial position of the trajectory and the number of steps of the complete trajectory is 1200 steps. To learn the weights of the DMPs' forcing function, Equation (2.8) is used. In addition to the joints' data, the initial position and velocity of the ball are also retrieved from the recorded demonstrations. Together, they define the context of the task $\boldsymbol{s} = \{\boldsymbol{b}, \dot{\boldsymbol{b}}\}$. The ball data is saved in cartesian space, so before performing calculations, it is first transformed into spherical coordinates. Since sample parameters are drawn from Gaussian policies with a linear mean $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\boldsymbol{\varphi}(\boldsymbol{s}), \boldsymbol{\Sigma})$, where $\boldsymbol{\varphi}(\boldsymbol{s})$ denotes features of the context s, the policy parameters $\boldsymbol{K}$ can be estimated from demonstrations by ridge regression as

$$\boldsymbol{K} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}\boldsymbol{\Theta},$$

where $\boldsymbol{\Phi}^T = [\boldsymbol{\varphi}(\boldsymbol{s}_1), .., \boldsymbol{\varphi}(\boldsymbol{s}_N)]$ is the context (feature) vector collected from demonstrations, $\boldsymbol{\Theta} = [\boldsymbol{\theta}_1, .., \boldsymbol{\theta}_N]$ are the samples obtained by imitation learning and λ is the ridge factor that specifies the strength of regularization.

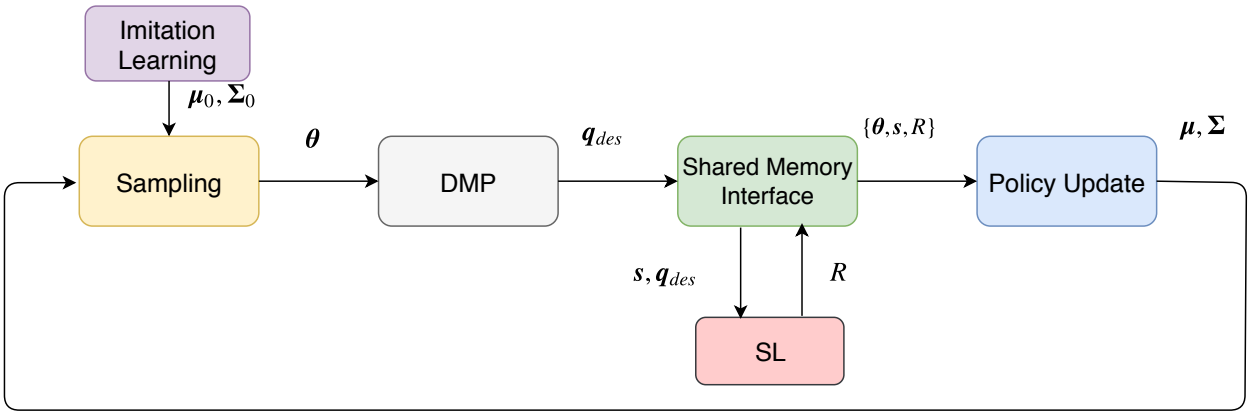


Figure 4.3: A figure depicting the general flow of the framework. The initial policy parameters μ_0 and Σ_0 are obtained by imitation learning. They are then passed to the sampling distribution $\mathcal{N}(\theta|\mu, \Sigma)$ where the samples θ are drawn. The samples are the DMP parameters, which are used to generate the desired trajectory q_{des}. The desired trajectory q_{des} and the context s (independently sampled from a uniform distribution) are sent to SL through the shared memory interface. The context specifies the initial state of the ball and the desrired trajectory is specified by q_{des}. After the trajectory execution, the reward R for the episode is sent by SL. Using the data set $\{\theta, s, R\}$, the learning algorithm updates the policy parameters μ, Σ which are fed back into the sampling distribution. This loop is repeated until satisfactory performance is reached.

Trajectory Generation: To generate trajectories Equation (2.5) is used. The weights θ of the forcing function (2.7) are obtained by sampling from a Gaussian distribution with the policy parameters. For each step of the trajectory, first the forcing function is computed using Equation (2.7), then the desired joint acceleration $\ddot{q}_{t+1}$ is obtained using Equation (2.5). In order to obtain the joint positions q_{t+1} and velocities $\dot{q}_{t+1}$, Euler's method/integration scheme is used

$$\begin{aligned} \dot{q}_{t+1} &= \dot{q}_t + \ddot{q}_{t+1} dt, \\ q_{t+1} &= q_t + \dot{q}_{t+1} dt, \\ \dot{z}_{t+1} &= -\tau \alpha_z z_t, \\ z_{t+1} &= z_t + \dot{z}_{t+1} dt \ . \end{aligned} \tag{4.1}$$

Sampling: To sample from a Gaussian distribution, the mean μ and covariance Σ of the distribution are needed, and since the mean is dependent on the context, its value is computed for each context with the current policy parameters, i.e., $\mu_i = K\varphi(s_i)$. First a random context is drawn from a uniform distribution. Using this context, the mean μ_i is computed, then samples are drawn from the current search distribution.

Trajectory Execution: The samples drawn from the search distrbution, as previously mentioned are the weights of the forcing function of the DMPs. They are then sent to the trajectory generator (reference trajectory generation point somehow), which returns the desired joint positions of the robot. In order to execute this trajectory, first the current context, which specifies the initial position and velocity of the ball is sent along with the initial state/home position of the robot which is arbitratily as the home position using the shared memory interface. Next, the whole desired trajectory is sent for execution via shared memory. An execution loop awaits commands and moves the robot every timestep to the corresponding trajectory step.

Reward Computation: The reward used for the tetherball task is computed in SL. It is computed as $p_1 + p_2 + p_3$ where

$$\begin{aligned} p_1 &= 100(1 - \exp(d^2)) \ , \\ p_2 &= -J \ , \\ p_3 &= 500c \quad c \in \{0, 1\} \ , \end{aligned}$$

in which d denotes the minimum distance between the ball and the racket during the whole episode/trajectory, J is the average of the load torque of the robot during the episode and c is the contact flag indicating whether the robot hit the ball or not.

5 Experiments

This chapter presents the experimental evaluations performed on the selected algorithms and discusses the results. The algorithms are first assessed on standard optimization test functions, then they are evaluated on the tetherball task from the previous chapter. The experimental setup on the standard optimization functions is provided followed by the results and likewise for the tetherball task.

5.1 Benchmarking on Standard Optimization Test Functions

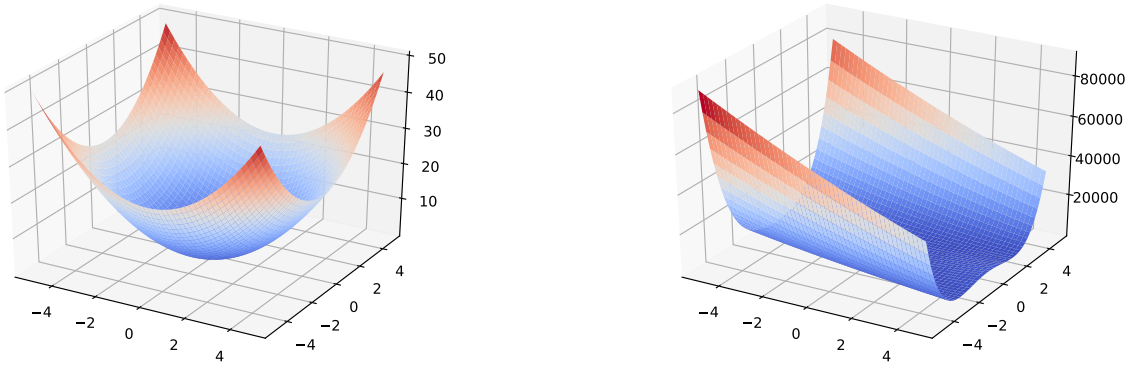
Test functions provide *artificial landscapes* to evaluate the performance of optimization algorithms. They are used to assess robustness, convergence speed and general performance of the algorithms. The two classical benchmarking functions chosen are the Sphere and Rosenbrock functions [Molga and Smutnicki, 2005].
The Sphere function is a unimodel convex function denoted by

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{x}_i^2. \tag{5.1}$$

The Rosenbrock function, sometimes referred to as the *banana function*, is a classic optimization problem, which is also unimodal and convex. It has one global optimum which is inside a parabolic shaped flat valley. It is usually easy to find the valley, however finding the global optimum is particularly difficult, which makes this function popular with testing algorithms' performance. It is defined as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N-1} [100(\boldsymbol{x}_{i+1} - \boldsymbol{x}_i^2)^2 + (1 - \boldsymbol{x}_i)^2]. \tag{5.2}$$

The objective is usually to minimize the functions above, however it is easy to obtain a maximization objective by simply inverting the sign of the function. In order to incorporate the context $\boldsymbol{s}$ into the optimization functions, the samples $\boldsymbol{\theta}$ are linearly transformed with the context $\boldsymbol{x} = \boldsymbol{\theta} + \boldsymbol{G}\boldsymbol{s}$, where $\boldsymbol{G}$ is a constant matrix sampled from a normal distribution. Both functions have their global optima at zero. The context is drawn from a uniform distribution $1 \leq \boldsymbol{s}_i \leq 2$. In all algorithms, quadratic features are used for $\phi(\boldsymbol{s})$. The samples are drawn from a linear Gaussian policy $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\varphi(\boldsymbol{s}), \boldsymbol{\Sigma})$, where $\varphi(\boldsymbol{s})$ is just a linear transformation of the context i.e., adds a bias term, except for CMORE where $\varphi(\boldsymbol{s}) = \phi(\boldsymbol{s})$. For each experiment, the average cost over 5 trials is plotted.



(a) 2D Sphere

(b) 2D Rosenbrock

Figure 5.1: Standard Test Functions.

5.1.1 Evaluating CREPS & CREPS-CMA on Number of Samples and Initial Variance

In this experiment, CREPS is compared against its closely related counterpart CREPS-CMA. The performance of the algorithms is examined under different number of samples. The experiment is performed on the 15-dimensional sphere and rosenbrock functions with a 1-dimensional context. The initial distributon is initialized as $\mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, 1000\boldsymbol{I})$. The KL is set to 0.5.

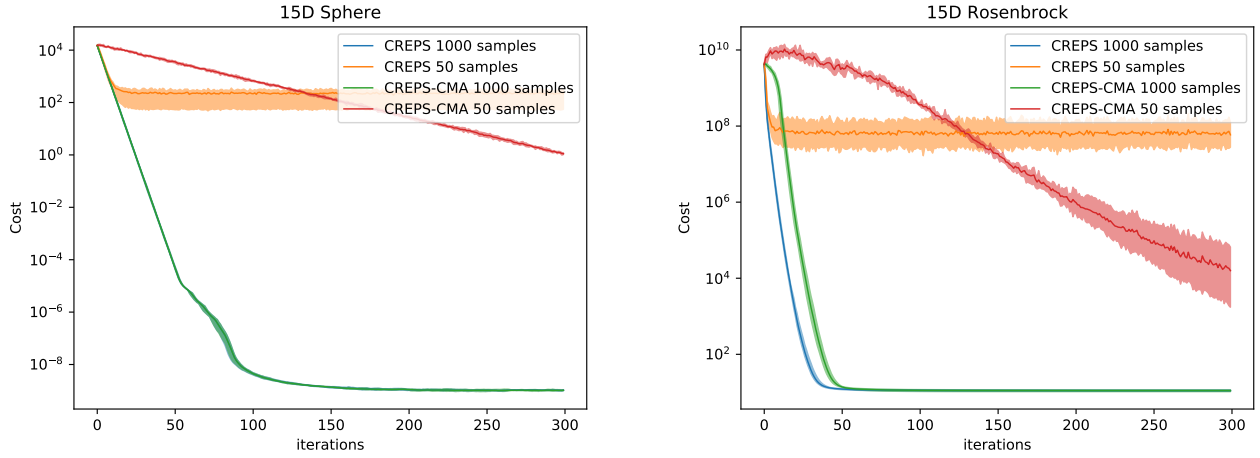


Figure 5.2: Plots of experiment comparing the performance of CREPS and CREPS-CMA with a high and low number of samples.The experiment is performed on a 15-dimensional sphere (left) and rosenbrock (right). It is evident that CREPS-CMA outperforms standard CREPS with a lower number of samples and has almost the exact performance otherwise. This is due to CREPS-CMA compensating the low number of samples by incorporating information from the old covariance matrix to the sample covariance matrix.

Figure 5.2 shows that CREPS is outperformed by CREPS-CMA when given a lower number of samples. This is due to CREPS-CMA incorporating information from the old covariance matrix when the number of effective samples is low. However, when both algorithms are given sufficiently many samples, the achieve the same performance.

The next experiment examines the effect of the initial variance in the search distribution on the performance of CREPS. This experiment is run on an 8-dimensional sphere and rosenbrock with a 4-dimensional context. 200 samples are generated every iteration, the KL is set to 0.5 and the mean function $\boldsymbol{K}$ is initialized to $\mathbf{0}$. The two initializations under evaluation are $\mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, 1000\boldsymbol{I})$ and $\mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \boldsymbol{I})$.

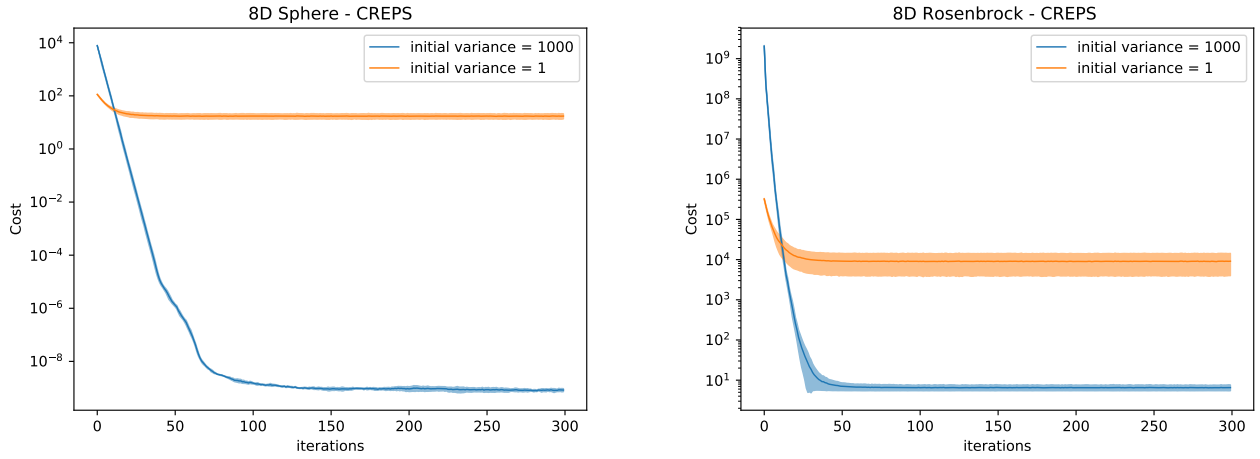


Figure 5.3: Results showing performance of CREPS assessed on different initial variances of the distribution. The experiment is performed on an 8-dimensional sphere(left) and rosenbrock(right) with a 4-dimensional context. The plots show the significant increase in the performance of the algorithm when given a larger initial variance. A larger initial variance indicates a larger initial distribution, which gives the algorithm room for exploration, which could allow the algorithm to converge to a better optimum.

Results in Figure 5.3 clearly show that algorithm with higher initial variance significantly outperforms the other. This highlights the importance of the initial variance for some algorithms. A larger initial search distribution might sometimes benefit a search algorithm by providing it with some exploration, which is often needed at the beginning of the search. Unlike CMORE, CREPS has no lower bound on the entropy of the distribution, so the entropy is prone to drastically dropping, so having a larger initial variance delays possible premature convergence.

5.1.2 Evaluating CMORE with Minimum Entropy

This experiment examines the outcome of different values of the minimum entropy H_0 on the performance of CMORE. H_0 is used in setting the lower bound on the entropy in CMORE as $\beta = \gamma(\mathbb{E}[H(q)] - H(\pi_0)) + H(\pi_0)$. CMORE was evaluated on the 8-dimensional sphere and rosenbrock with a 4-dimensional context. The KL is set to 0.5, γ is set to 0.80 and 1500 samples are generated each iteration. To fit the quadratic model (3.20), linear ridge regression is used.

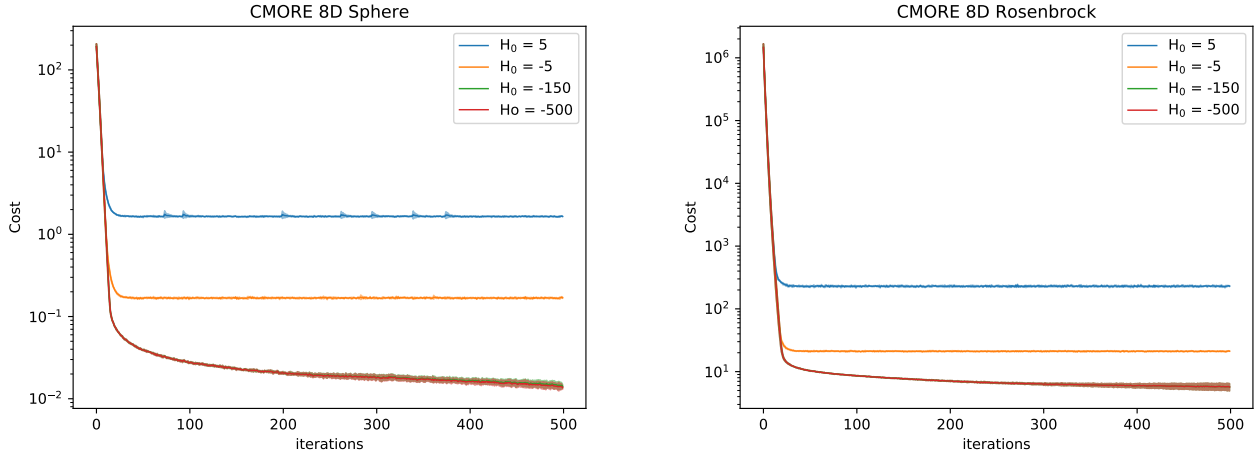


Figure 5.4: The results of different values of the minimum entropy H_0 on the performance of CMORE. The experiment is performed on an 8-dimensional sphere (right) and rosenbrock (left) with a 4-dimensional context. As shown in the plots, a lower minimum entropy yields a better performance.

Results in Figure 5.4 show that a lower minumum entropy yields better performance. Intuitively, the more entropy there is in the distribution, the harder it is to narrow the distibution around the optimum parameters. Additionally, having less entropy in the distribution result in *safer* update steps towards the optimum parameters.

Experiments were also run to examine the effect of the initial variance and initial mean of the distribution on the performance of CMORE, but due to no effect on the performance, the plots are not included. Also, varying γ between 0.99 and 0.10 did not affect the performance with H_0 = -150.

5.1.3 Evaluating CCMAES with Initial Variance and Step Size

The following experiment demonstrates the effect of varying the initial variance on the performance of CCMAES. 200 samples are drawn every iteration and the initial step-size σ_0 is 1. The experiment is performed on the 8-dimensional sphere and rosenbrock with a 4-dimensional context. For the mean given by $\boldsymbol{m}(s) = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{b}$, the matrix $\boldsymbol{A}$ is initialized to $\boldsymbol{0}$ and the bias $\boldsymbol{b}$ is drawn from a normal distribution. The initial covariance matrices in comparison are $\boldsymbol{I}$ and $10\boldsymbol{I}$.

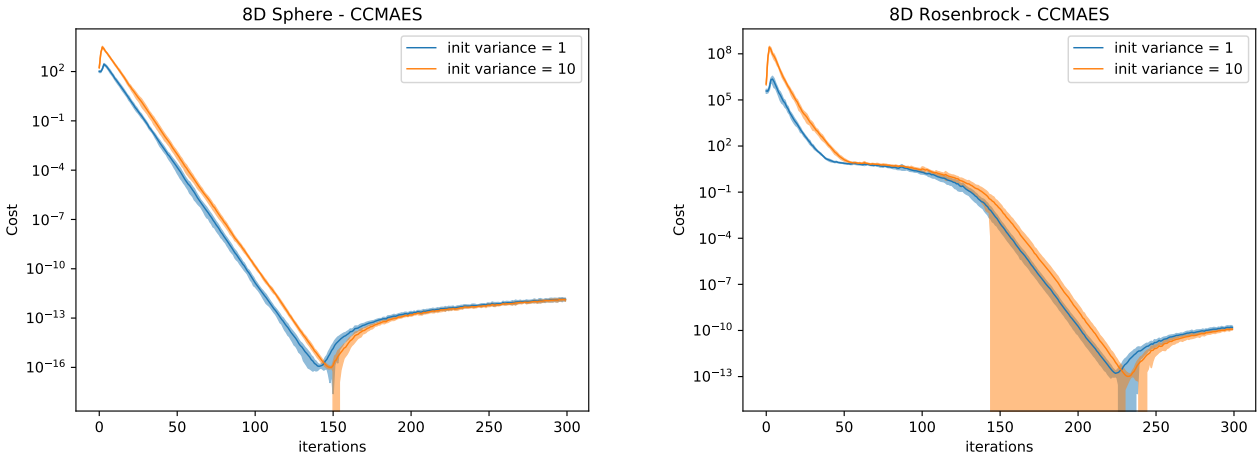


Figure 5.5: The effect of adding initial variance on the performance of CCMAES. It is clear that adding noise to the initial covariance matrix in the context of these optimization problems only hinders the performance of CCMAES. The inital policy might already be near the optimum parameters, so expanding the search distribution would only slow down convergence.

Results in Figure 5.5 show that no performance gain is achieved with a larger initial variance. On contrary, it deters the performance for this problem. This might be due to the search distribution already starting near the optimal parameters, so increasing the search space would only slow down convergence.[1]

This experiment has the same settings as the previous ones, except the initial covariance matrix is initialized to the identity matrix $\boldsymbol{I}$ and the initial step-size σ_0 is varied.

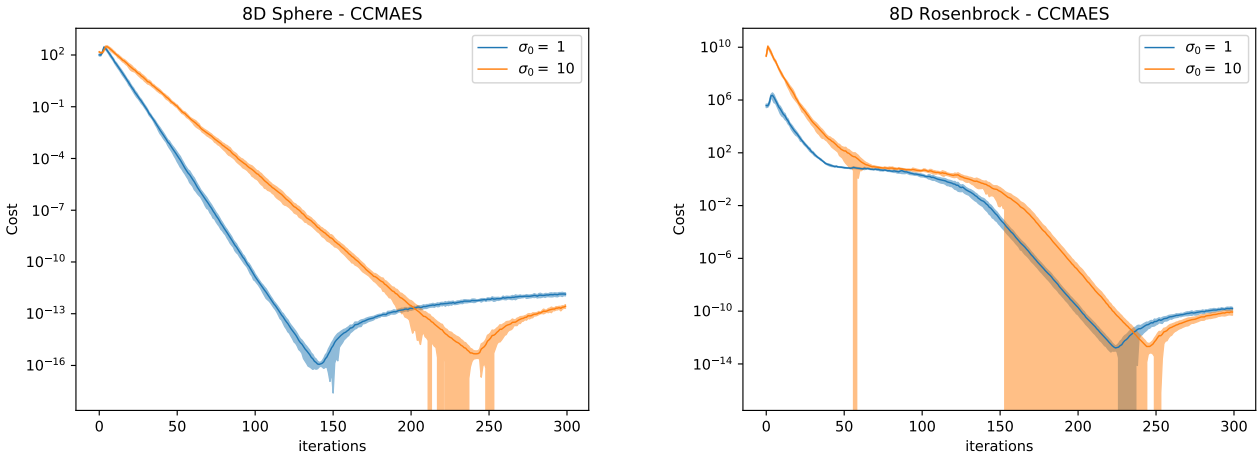


Figure 5.6: The effect of the initial step-size σ_0 on the performance of CCMAES on the 8-dimensional sphere (left) and rosenbrock (right). Increasing the step-size deters the performance in the context of these optimization problems. With a larger initial step-size the distribution may drift away from the optimum solution in the early stages, which slows down the convergence.

Results in Figure 5.6 are consistent with the previous experiment, since both experiments scale the variance of the distribution. This can be seen by referring to the sampling equation for CCMAES 3.25. A larger initial step-size σ_0 does not achieve performance gain as, in this scenario, the initial search distribution is initialized near the optimum parameters, increasing the initial step-size causes the distribution to drift away and therefore take longer to converge.

[1] The blip at the end of the curves might be due to numerical errors.

5.1.4 Benchmarking on Sample efficiency

In this experiment, the sample efficiency of each algorithm is assessed. For all algorithms, the covariance matrix is initialized to $\boldsymbol{I}$. The KL for CREPS and CREPS-CMA is set to 0.5. For CMORE, the minimum entropy H_0 is set to -150 and $\gamma = 0.80$. CCMAES starts with step size $\sigma = 1$, the mean functions are initialized to $\mathbf{0}$ and the bias $\boldsymbol{b}$ for CCMAES is drawn from a normal distribution. The average cost at convergence is assessed against the number of samples generated.

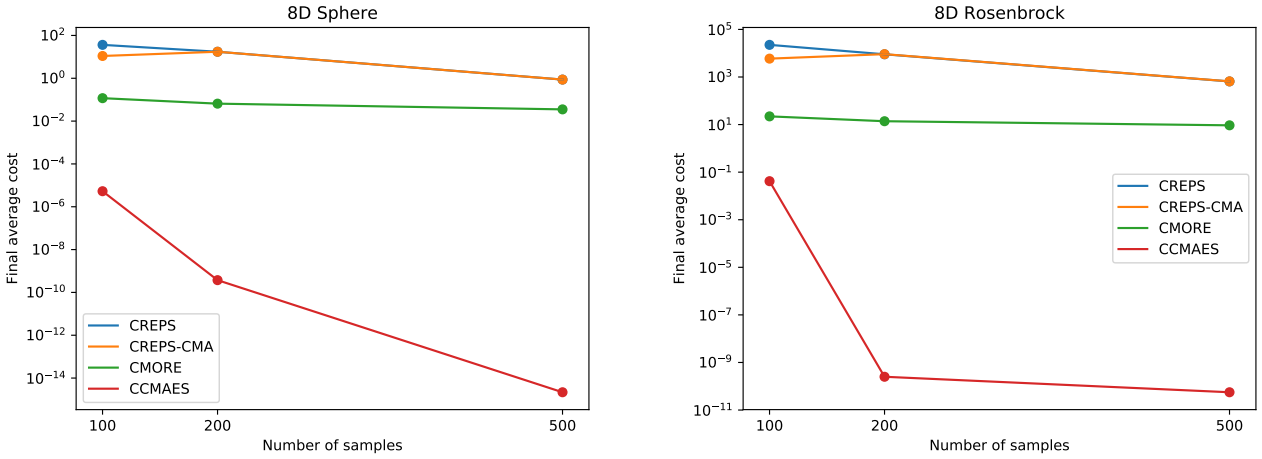


Figure 5.7: Results showing performance of the algorithms assessed using 100, 200 and 500 number of samples. CCMAES demonstrates the largest change in performace with different number of samples. CMORE is least sensitive, with trivial improvement given more samples. CREPS-CMA out performs CREPS with a lower number of samples and performs the same otherwise. For clarity, only the mean is plotted.

Figure 5.7 shows the the results of the sample efficiency experiment. CCMAES shows a significant increase in performance with higher number of samples, so it appears to be the most sensitive. CMORE on the other hand is the least affected by the number of samples. It should be noted however, that for higher dimensional problems the performance of CMORE can be strongly affected because it requires a sufficient number of samples to learn the quadratic model (3.20). CREPS-CMA slightly out performs CREPS with a lower number of samples, consistent with the results presented in 5.2. However, given 200 samples, the minimization (3.35) that chooses the interpolation factor is 1, which only utilizes the sample covariance matrix as in REPS, so both algorithms achieve the same performace.

5.1.5 Benchmarking on Scalability

This experiment compares the scalability of the algorithms. The evaluations are carried on the 2, 8 and 15-dimensional sphere and rosenbrock with 2, 4 and 7 dimensional contexts respectively. 500 samples where generated each iteration. The covariance matrix is intialized to $\boldsymbol{I}$. The settings from the previous experiment are maintained.

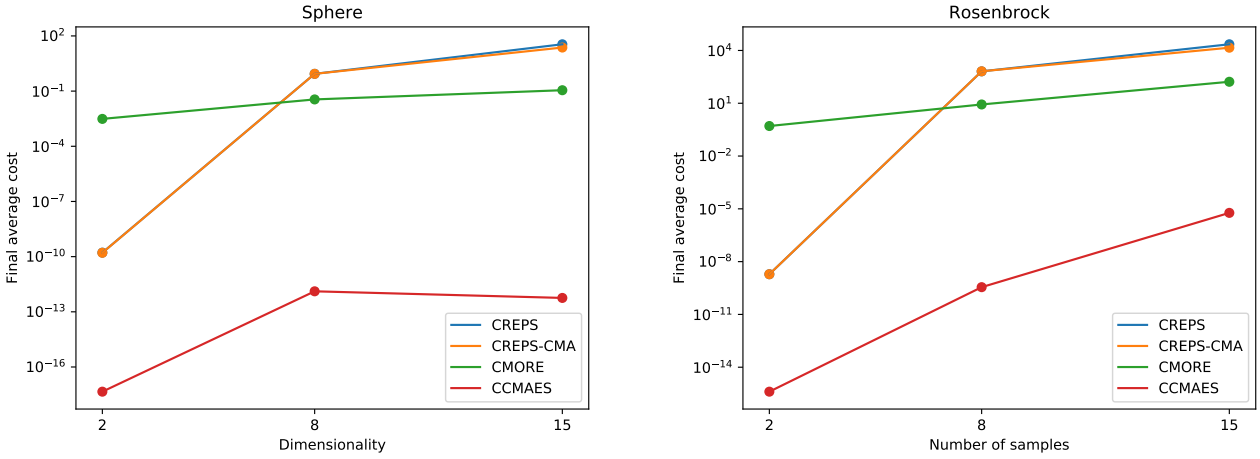


Figure 5.8: Results showing performance of algorithms assessed on the 2, 8 and 15-dimensional sphere (left) and rosenbrock(right). The performance of all algorithms decreases as the dimensionality increases. CMORE being the most stable followed by CCMAES then CREPS and CREPS-CMA. CREPS-CMA shows no significant advantage over CREPS in this case. For clarity, only the mean is plotted.

As one might expect, increasing the dimensionality of the problem makes it tougher to solve. CMORE observes the least change in performance followed by CCMAES then CREPS and CREPS-CMA. One can see a trivial outperfomance of CREPS-CMA over CREPS at 15 dimensions. This is due to the division over the square of the dimensionality in Equation (3.35) that determines the interpolation factor.

5.1.6 Benchmarking on Optimum Performance

In this experiment all parameters are tuned to achieve the optimum performance by each algorithm. 200 samples are drawn per iteration. The KL for CREPS, CREPS-CMA and CMORE is set to 0.5. For CMORE the minimum entropy H_0 is set to -150 and $\gamma = 0.80$. The initial covariance matrix for CREPS and CREPS-CMA is $1000\boldsymbol{I}$ and $\boldsymbol{I}$ for CMORE and CCMAES. The initial step-size for CCMAES $\sigma_0 = 1$.

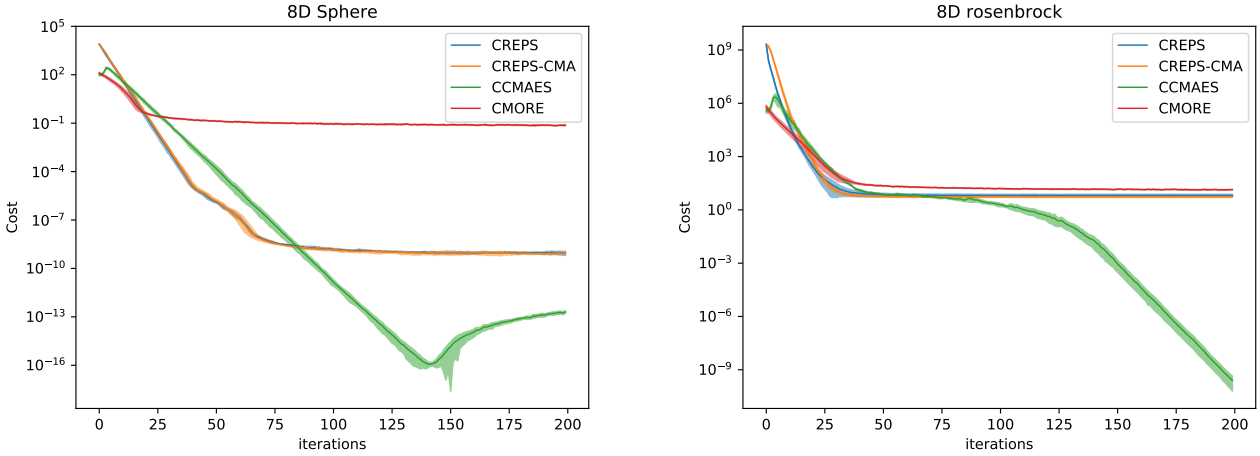


Figure 5.9: The four policy search algorithms evaluated on an 8-dimensional sphere (left) and rosenbrock(right) with a 4-dimensional context. The best performance is achieved by CCMAES, followed by CREPS and CREPS-CMA that observe the same performance. CMORE achieves the least optimum performance. For clarity, only the mean is plotted.

Figure 5.9 shows the best performance achieved by CCMAES, followed by CREPS-CMA and CREPS and finally CMORE. One possible cause to justify CMORE's performance is the method of model-fitting. In [Tangkaratt et al., 2017], acceler-

ated proximal gradient is used to learn the model, whearas in this experiment ridge regression is used. This is consistent with the evaluations in [Tangkaratt et al., 2017], which show that ridge regression does not achieve a satisfying results.

5.2 Benchmarking on Tetherball

For the task of tetherball, 5 DMP basis functions are used to generate the trajectories for all algorithms except CMORE. In CMORE, the feature matrix used to estimate the parameters of the quadratic model (3.20) is very high-dimensional, as it includes quadratic features of the parameters of the task, quadratic features of the context features, linear features of the parameters and the context features and cross features. In an attempt to reduce the dimensionality of the problem of CMORE only 4 DMP basis were used, which reduces the parameter space of the task. With 4 joints and the DMP parameters for each joint, the task has a 20-dimensional parameter space and 16-dimensional for CMORE. The DMP parameters are set as follows: the goal g is set as an arbitrary starting position of the robot $[0.5236, -0.6, 1.2, 0.5]$ rad, α and β are constants set to 25 and $\alpha/4$ respectively, $\tau = 1$ and $\alpha_z = 0.5$. The canonical system/ initial state of the system z_t is initially set to be 1. The number of steps for the trajectory is set to $T = 1200$ with step size $dt = 0.002$, which is the time period for the simulator. The context is drawn from a uniform distribution as

$$\begin{aligned} \theta &\sim \mathcal{U}\Big(\frac{-\pi}{20}, \frac{\pi}{20}\Big) + \frac{\pi}{10}, \\ \varphi &\sim \mathcal{U}\Big(0, \frac{\pi}{6}\Big) + \frac{\pi}{2}, \\ d\theta &\sim \mathcal{U}(-0.1, 0.1), \\ d\varphi &\sim \mathcal{U}(-0.15, 0.15) + 0.7, \end{aligned}$$

which gives a range of 18°, 30°, 0.2 rad/s, 0.3 rad/s for θ, φ, $d\theta$ and $d\varphi$ respectively. For all the upcoming experiments, CREPS and CREPS-CMA reuse old samples in a sample pool of 250 and CMORE uses a sample pool of 4500 samples. Results of all experiments are averaged over 3 trials. Plots of the average reward and the hit rate (the percentage of hitting the ball during the iteration) are used to assess the performance.

5.2.1 Evaluating CREPS with KL, Number of DMP Basis Functions and Value Functions

In the following experiment, the effect of different values for the KL bound ϵ on CREPS is presented. The distribution is initialized using imitation learning. 50 fourier features are used for the value function $\phi(\boldsymbol{s})$. 25 samples are generated every iteration and a sample pool of 250 samples is maintained. The different values for the KL used are $\epsilon = \{0.1,\ 0.3,\ 0.5,\ 0.7\}$.

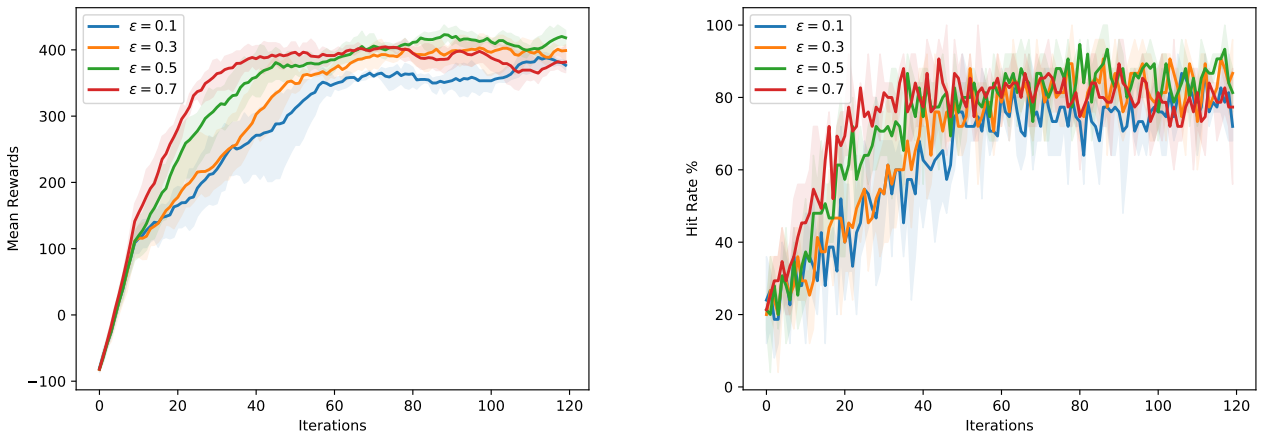


Figure 5.10: This experiment evaluates the performance of CREPS given different KL bounds ϵ. Lower values for ϵ converge slower than higher values because the new distribution remains closer to the previous distribution with lower ϵ.

The KL determines how far the updated policy distribution moves from the previous policy distribution. In a way it can be thought of as a step-size. Results in Figure 5.10 show how the magnitude of ϵ affects convergence speed. A high

ϵ may cause unsafe update steps or cause the search to overshoot an optimum solution. This can be seen with $\epsilon = 0.7$, where the average reward drops after the 60th iteration.

The following experiment was used to choose the best number of basis functions for trajectory generation using DMPs. The same settings from the previous experiment are maintained and the KL is set to 0.5. The experiment evaluates the performance using 3, 4, 5 and 8 DMP basis functions.

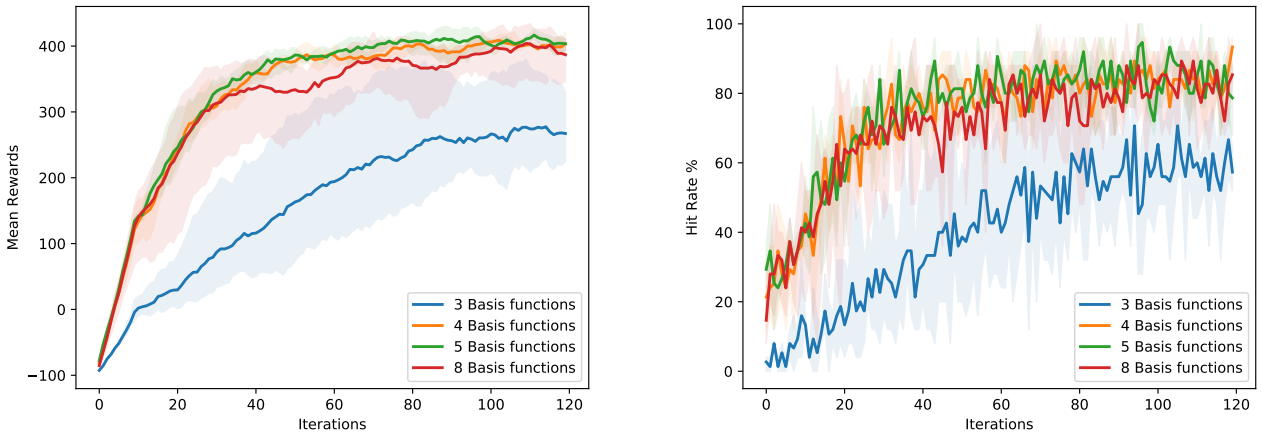


Figure 5.11: Results of varying the number of DMP basis functions used to generate trajectories for the robot hitting movement. The lowest performance is achieved by the lowest number of DMP basis functions. A better average reward is achieved by increasing the number of DMPs a certain threshold is reached. The performance using 8 DMP basis functions is lower than when using only 4 or 5, due to the redundant added complexity.

Results in Figure 5.11 show how the performance is affected with a varying number of DMP basis functions. As depicted in the results, while increasing the number of basis functions could enable generating more complex trajectories, this is also a tuned parameter because it is prone to overfitting. At some threshold, adding more basis functions adds more complexity where it is not needed, increasing the parameter space of the problem rendering it harder without achieving any performance gain.

The next experiment evaluates the performance of different features to represent the value function or the context-dependent baseline (3.9). The settings from the previous experiment are maintained and 5 DMP basis functions are used. To represent the value function, quadratic features, linear features and fourier features are examined.

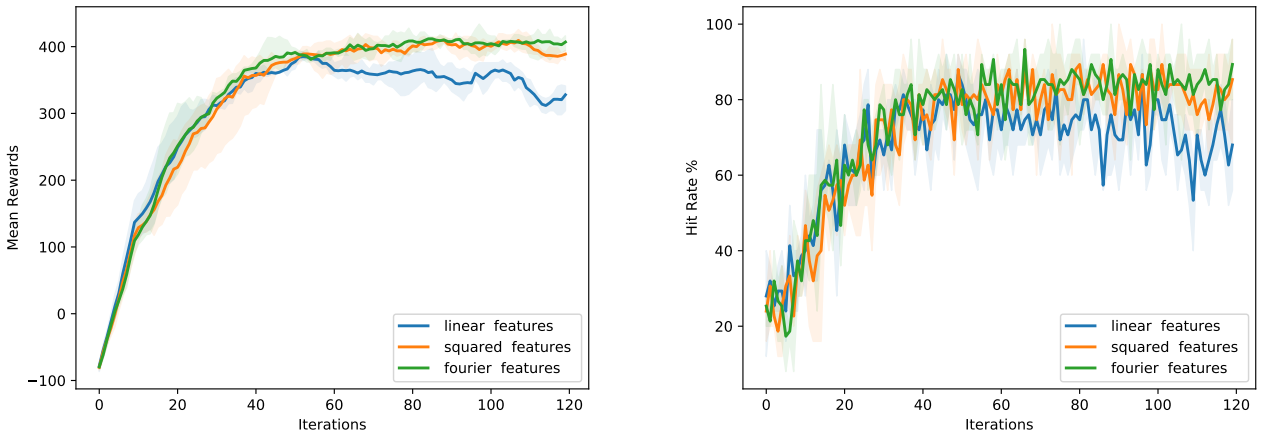


Figure 5.12: Performance of CREPS using different features to estimate the value function is shown. Fourier features and quadratic features perform equally, while performance using linear features collapses after 50 iterations.

The choice of the features of value functions is considered a tunable parameter in the context of the given task. On one hand complex features are powerful and can give a good approximation to the value function and on the other hand, complexity may increase the dimensionality and hardness of the problem. Figure 5.12 shows the performance of CREPS using different features to estimate the value function. Fourier features and quadratic features demonstrate similar performace, while linear features seem to lack the complexity to estimate the value function.

5.2.2 Evaluating CCMAES with Initial Policy and Population Size

The following experiment examines the effect of providing a policy initialized using prior knowledge of the task i.e., using imitation learning, and a policy initialized randomly.

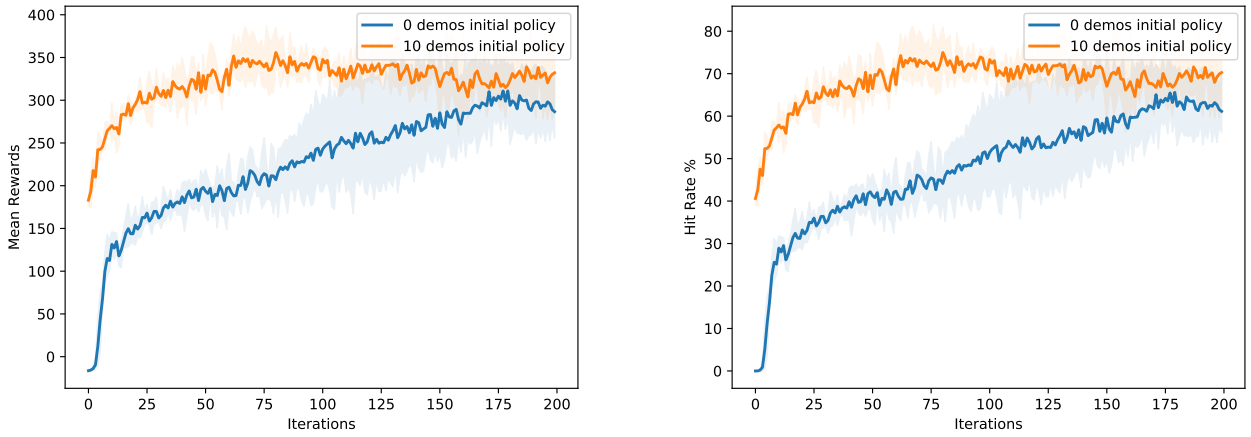


Figure 5.13: The performance of CCMAES is evaluated against two policy initializations, one that is initialized randomly with no prior knowledge and the other is initialized by imitation learning using 10 demonstrations. The policy initialized from the demonstrations clearly outperforms the randomly initialized policy. The policy with prior knowledge achieves faster convergence and convergence to a better optimum.

Results in Figure 5.13 emphasize the importance of a good initial policy for robotic tasks. Evidently, the more prior knowledge the algorithm has about the task, the more likely it is to converge faster and to a possibly better solution.

The following experiment examines the effect of the population size on the performance of CCMAES. The initial step-size σ_0 is 1 and the covariance matrix is initialized to $100\boldsymbol{I}$.

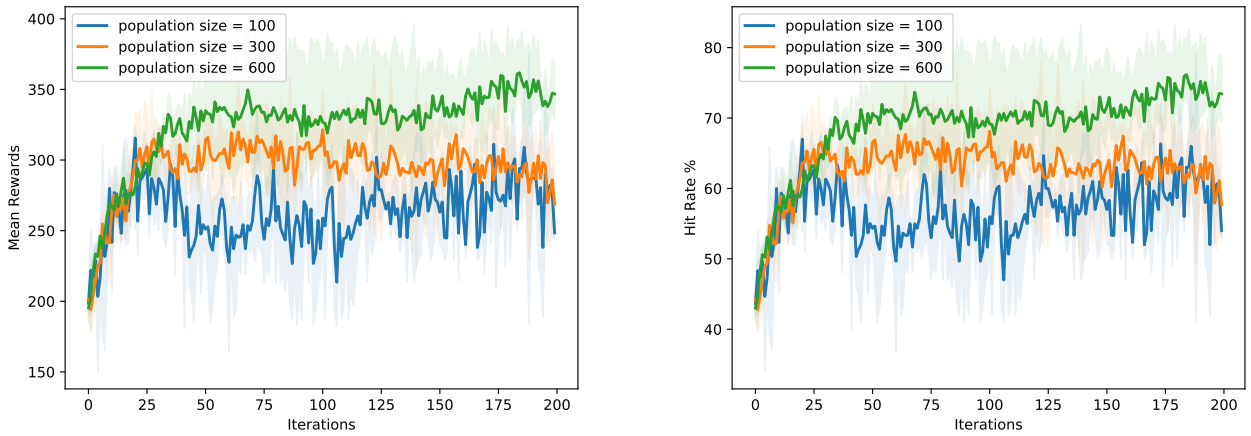


Figure 5.14: Evaluating the performance of CCMAES with various population sizes. Results show that CCMAES needs a large number of samples to reach satisfactory performance.

Congruent to the results on the standard functions. CCMAES's performance is highly affected by the population size or number of samples. The best performance is achieved with the highest number of samples. In comparison to CREPS, CCMAES needs more samples to reach a satisfactory solution. While not presented, experimental evaluations on CREPS that vary the number of samples drawn each iteration have been carried out. Results showed that, unlike CCMAES, CREPS usually converges to the same solution at different speeds depending on the number of samples.

5.2.3 Benchmarking on Optimum Performance

The settings for the final experiment are as follows. The KL for CREPS, CREPS-CMA and CMORE is set to 0.5. For CMORE, the minimum entropy H_0 = -75 and γ = 0.99. The initial mean function $\boldsymbol{K}$ of the distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{K}\varphi(\boldsymbol{s}), \boldsymbol{\Sigma})$ is initialized by imitation learning using 10 demonstrations for all algorithms. The initial covariance matrices $\boldsymbol{\Sigma}$ for CREPS and CREPS-CMA are also initialized by imitation learning. However, that was not well suited for CMORE due to numerical instabilities, so it is instead initialized as $\boldsymbol{\Sigma}_0 = 10\boldsymbol{I}$. For CCMAES, the algorithm already has a precise way of constructing the covariance matrix, so only extra variance was added and $\boldsymbol{\Sigma}_0 = 100\boldsymbol{I}$. [2]. For the context features $\phi(\boldsymbol{s})$, CREPS and CREPS-CMA use 50 fourier features. CMORE uses 60 fourier features and CCMAES uses quadratic features.[3] For CREPS, CREPS-CMA and CCMAES $\varphi(\boldsymbol{s})$ is a linear transformation of the context, i.e., only a bias term is added. In CMORE $\varphi(\boldsymbol{s}) = \phi(\boldsymbol{s})$. 250 samples are generated each iteration.

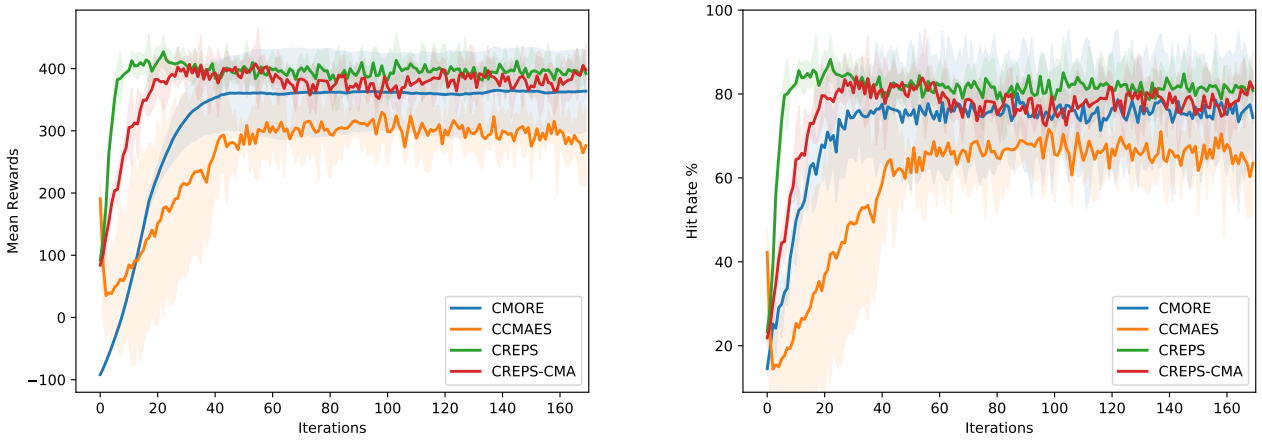


Figure 5.15: An evaluation of the optimum performance achieved by all algorithms on the tetherball task.. Results show the best performance achieved by CREPS and CREPS-CMA reaching a hit rate of 80% followed by CMORE converging at 75% and finally CCMAES converging around 62%.

While all algorithms reach a satisfactory performance i.e., all manage to achieve a hit rate above 60%, some outperform others. Results in Figure 5.15 show the best performance achieved by CREPS and CREPS-CMA followed by CMORE and finally CCMAES. Better results might be achieve by CMORE if a different model-fitting approach is used instead of ridge regression. Additionally, tuning the features of the value function and initial policy parameters might achieve better performance, however due to the lengthiness of the evaluation process, it was not possible to try out many different parameter settings. The same argument can be extended for CCMAES. Another factor that might lead to a higher performance from CMORE and CCMAES is the initialization of the covariance matrix. While CREPS and CREPS-CMA use the initial covariance matrix initialized by imitation learning, CMORE and CCMAES start with a scaled identity matrix.

[2] Some experiments were run to choose $\boldsymbol{\Sigma}_0$. The algorithm did not learn with $\boldsymbol{\Sigma}_0 = \boldsymbol{I}$ nor with $\boldsymbol{\Sigma}_0 = 1000\boldsymbol{I}$.

[3] Several experiments were run to obtain the value function that achieved a good performance for each algorithm. radial basis functions, fourier features, quadratic features and linear features were examined. For CCMAES, fourier features performed equally as quadratic features.

6 Conclusion and Future Work

In this thesis, a group of policy search methods were reviewed as a possible solution to complex robotic tasks. The main focus was on contextual policy search methods, which offer an additional level of generalization that gives us more versatile learning. The selected algorithms were first derived as from the literature, then benchmarked on standard test optimization functions and on a simulated robot tetherball task. First, the background needed to implement these tasks was presented. The fundementals of reinforcement learning were introduced along with Markov Decision Processes (MDPs), the classical mathematical framework for reinforcement learning. Then, the robot control problem was formulated as a reinforcement learning problem to enable us to harness the power of RL techniques and apply it to robotic tasks. To solve the reinforcement learning problem, a class of approaches called policy search (PS) methods was presented. Then the KL-divergence, a metric that is exhaustively used in policy search methods was breifly introduced. The discussion of the background is then concluded with the introduction of Dynamic Movement Primitives (DMPs) which act as trajectory generators. In the scope of this thesis, they are used as a parametric policy representation. Finally, to bootstrap the parameters of the DMPs, imitation learning is used, which is often favourable in robotic applications since they are too challenging to initialize. Folowing the background, a group of policy search algorithms were thoroughly discussed. The standard version of each algorithm was first introduced followed by its contextual version. The algorithms presented were: Relative Entropy Policy Search (REPS) that bounds the KL between subsequent policy distributions, Model-based Relative Entropy (MORE), which adds and additional lower bound on the entropy and fits a quadratic model for the objective function, Covariance Matrix Adaptation Evolution Strategies (CMA-ES), another black box optimizer that exploits evolution paths to update its parameters, and finally, a couple of regularized versions of REPS that linearly interpolate the old covariance matrix with the sample covariance matrix. Then, after elaborating the experimental setup for the tetherball task and the standard functions, results of the experimental evaluations on the standard functions were first presented. The algorithms were assessed on metrics such as sample efficiency, scalability and optimum performance. The performance of the algorithms was also examined with various settings of algorithm-specific hyperparameters. This was followed by benchmarking on the simulated task of robot tetherball.

After noticing the significant difference between the results of the experimental evaluations on the standard benchmark functions and the simulated robot tetherball task, one can easily reach the conclusion that using reinforcement learning to find solutions to robotic tasks proves to be a challenging task met with several limitations. One challege that was encountered in this thesis is reward shaping. Crafting a good reward function is quite difficult and not often intuitive, however it determines the success of the learning. Additionally, parameter sensitivity of reinforcement learning algorithms render some problems more tedious as it is not always practical to try out all possible parameter settings.

A simulated robotic task was used in this thesis. In addition, it would be interesting to evaluate these algorithms on a real robot setup, because simulators often under-model the real-world system. Moreover, this thesis explored only a limimited class of algorithms i.e., black box optimizers. Having a comparison with other other classes of algorithms such as natural gradient approaches would provide a broader outlook and a richer comparison. Furthermore, to add an additional layer of versatility and generalizability to the robot's policy, it would also be intersting to investigate hierarchal approaches on top of contextual approaches.

7 Appendix

7.1 Solving Constrained Optimization Problems

A constrained optimization problem is the problem of optimizing some function J under some constraints. It is generally denoted as

$$\begin{aligned} &\max_{\theta} \; J(\boldsymbol{\theta}), \\ \text{s.t} \;\; &f(\boldsymbol{\theta}) = a, \\ &g(\boldsymbol{\theta}) \leq b, \end{aligned}$$

where f and g are the constraints. The formulation above is usually referred to as the *primal* problem. The method of Lagrangian multipliers [insert citation] is a technique that finds the optima of a mutivariable function that is subject to some constraints. The Lagrangian function L for the problem above is formulated as

$$L(\boldsymbol{\theta}, \lambda, \eta) = J(\boldsymbol{\theta}) + \lambda(f(\boldsymbol{\theta}) - a) + \eta(g(\boldsymbol{\theta}) - b),$$

where λ and η the *lagrangian multipliers*, which are constants that penalize violating the constraints. For the example above, to penalize the violation the Lagrangian L is minimized by setting the derivative of L w.r.t. $J(\theta)$ to 0.

$$g(\lambda, \eta) = \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda, \eta).$$

The function g is called the *dual* function, which is usually easier to optimize than the primal. Additionally, if the primal problem is maximized then the dual is minimized and vice versa.

7.2 Dual function derivation of REPS

The following constrained optimization problem fro REPS given by

$$\begin{aligned} &\max_{\pi} \int \pi(\boldsymbol{\theta}) R(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}, \\ &\text{s.t. } \epsilon \geq \mathrm{KL}(\pi(\boldsymbol{\theta}) || q(\boldsymbol{\theta})), \\ &1 = \int \pi(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}, \end{aligned} \tag{7.1}$$

is solved using the method of Lagrangian multipliers. First the Langrangian is formulated as

$$L = \int R(\boldsymbol{\theta}) \pi(\boldsymbol{\theta}) d\boldsymbol{\theta} + \lambda\big(1 - \int \pi(\boldsymbol{\theta}) \, d\boldsymbol{\theta}\big) + \eta\big(-\int \pi(\boldsymbol{\theta}) \log \frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} \, d\boldsymbol{\theta} + \epsilon\big) \, . \tag{7.2}$$

Rearranging terms

$$L = \int \pi(\boldsymbol{\theta}) \big(R(\boldsymbol{\theta}) - \lambda - \eta \log \frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} \, \big) \, d\boldsymbol{\theta} + \lambda + \eta\epsilon \, . \tag{7.3}$$

To get the optimum policy, L is minimized; taking the derivative w.r.t. $\pi(\boldsymbol{\theta})$ and setting it to zero.

$$\begin{aligned} L' &= R(\boldsymbol{\theta}) - \lambda - \eta - \eta \log \frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} = 0 \, . \\ \pi(\boldsymbol{\theta}) &= q(\boldsymbol{\theta}) \exp\Big(\frac{R_{\boldsymbol{\theta}} - \lambda - \eta}{\eta}\Big) \, , \\ \pi(\boldsymbol{\theta}) &= q(\boldsymbol{\theta}) \exp\Big(\frac{R(\boldsymbol{\theta})}{\eta}\Big) \exp\Big(\frac{-\lambda - \eta}{\eta}\Big) \, . \end{aligned} \tag{7.4}$$

From the last constraint in Equation (7.1) and (7.4) λ can be obtained as

$$\lambda = \eta \log \int q(\boldsymbol{\theta}) \exp\Big(\frac{R(\boldsymbol{\theta})}{\eta}\Big) d\boldsymbol{\theta} - \eta \,. \tag{7.5}$$

Substituting the last constraint in Equation (7.1) and λ from above in L, the dual is

$$\begin{aligned} g(\eta, \lambda) &= \eta + \eta\epsilon + \lambda \,, \\ g(\eta) &= \eta\epsilon + \eta \log \int q(\boldsymbol{\theta}) \exp\Big(\frac{R(\boldsymbol{\theta})}{\eta}\Big) d\boldsymbol{\theta}. \end{aligned} \tag{7.6}$$

7.3 Closed form solution for MORE

The constrained optimization problem for MORE is given by

$$\begin{aligned} &\max_{\pi} \int \pi(\boldsymbol{\theta}) R(\boldsymbol{\theta}) \, \mathrm{d}\theta, \\ &\text{s.t.} \;\; \epsilon \geq \mathrm{KL}(\pi(\boldsymbol{\theta}) || q(\boldsymbol{\theta})), \\ &\qquad \beta \leq H(\pi), \\ &\qquad 1 = \int \pi(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} \,. \end{aligned} \tag{7.7}$$

The dual function for MORE is derived similarly as Section 7.2. Given the dual function and the quadratic model

$$\begin{aligned} g(\eta, \omega) &= \eta\epsilon - \omega\beta + (\eta + \omega) \log \int q(\boldsymbol{\theta})^{\frac{\eta}{\eta+\omega}} \exp\Big(\frac{R(\boldsymbol{\theta})}{\eta + \omega}\Big) \, d\boldsymbol{\theta} \,, \\ R(\boldsymbol{\theta}) &\approx \boldsymbol{\theta}^T \mathbf{R} \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{r} + r_0, \end{aligned}$$

and knowing that the policy distribution is a Gaussian $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$, we substitute $R(\boldsymbol{\theta})$ with the quadratic model and rearrange terms to get

$$\begin{aligned} g(\eta, \omega) &= \eta\epsilon - \omega\beta + (\eta + \omega) \log \Bigg(\int C_\pi^{\eta/\eta+\omega} \exp\Big(\frac{-0.5\eta}{\eta + \omega} (\boldsymbol{\theta} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu})\Big) \\ &\qquad \exp\Big(\frac{\boldsymbol{\theta}^T \mathbf{R} \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{r} + r_0}{\eta + \omega}\Big)\Big) d\boldsymbol{\theta} \Bigg), \\ &= \eta\epsilon - \omega\beta + (\eta + \omega) \log \Bigg(\int C_\pi^{\eta/\eta+\omega} \exp\Big(\frac{\eta}{\eta + \omega} (-0.5\theta^T \Sigma^{-1} \theta + \mu^T \Sigma^{-1} \theta - 0.5\mu^T \Sigma^{-1} \mu)\Big) \\ &\qquad \exp\Big(\frac{-0.5(\boldsymbol{\theta}^T \boldsymbol{R} \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{r})}{\eta + \omega}\Big) d\boldsymbol{\theta} \Bigg), \\ &= \eta\epsilon - \omega\beta + (\eta + \omega) \log \Bigg(\int C_\pi^{\eta/\eta+\omega} \exp\Big(\frac{1}{\eta + \omega} \Big(-0.5\boldsymbol{\theta}^T \eta \boldsymbol{\Sigma}^{-1} \boldsymbol{\theta} + \boldsymbol{\mu} \eta \boldsymbol{\Sigma}^{-1} \boldsymbol{\theta} - 0.5\eta \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + \boldsymbol{\theta}^T \boldsymbol{R} \boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{r} \Big)\Big) d\boldsymbol{\theta} \Bigg), \\ &= \eta\epsilon - \omega\beta + (\eta + \omega) \log \Bigg(\int C_\pi^{\eta/\eta+\omega} \exp\Big(\frac{1}{\eta + \omega} \Big(-0.5\boldsymbol{\theta}^T (\eta \boldsymbol{\Sigma}^{-1} - 2\boldsymbol{R}) \boldsymbol{\theta} + (\eta \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + \boldsymbol{r}) \boldsymbol{\theta} - 0.5\eta \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \Big)\Big) d\boldsymbol{\theta} \Bigg), \\ &= \eta\epsilon - \omega\beta + (\eta + \omega) \log \Bigg(\int C_\pi^{\eta/\eta+\omega} \exp\Big(\frac{1}{\eta + \omega} \Big(-0.5\boldsymbol{\theta}^T \boldsymbol{F}^{-1} \boldsymbol{\theta} + \boldsymbol{f} \boldsymbol{\theta} - 0.5\eta \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \Big)\Big) d\boldsymbol{\theta} \Bigg). \end{aligned}$$

Completing the square

$$g(\eta, \omega) = \eta\epsilon - \omega\beta + (\eta + \omega) \log \Bigg(\int C_\pi^{\eta/\eta+\omega} \exp\Big(-0.5(\boldsymbol{\theta} - \boldsymbol{F}\boldsymbol{f})^T \boldsymbol{F}^{-1} (\boldsymbol{\theta} - \boldsymbol{F}\boldsymbol{f}) + 0.5\boldsymbol{f}^T \boldsymbol{F} \boldsymbol{f} - 0.5\eta \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} \Big) d\boldsymbol{\theta} \Bigg).$$

Then plugging in the integration of a Gaussian and expanding C_π

$$g(\eta,\omega) = \eta\epsilon - \omega\beta + (\eta+\omega)\ \log\left(\exp\left(\frac{1}{\eta+\omega}\left(0.5(\boldsymbol{f}^T\boldsymbol{F}\boldsymbol{f} - \eta\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})\right)\right)\frac{\left((2\pi)^{-k/2}|\Sigma|^{-0.5}\right)^{\eta/\eta+\omega}}{(2\pi)^{-k/2}|(\eta+\omega)\boldsymbol{F}|^{-0.5}}\right).$$

Explanding the log

$$\begin{aligned}
g(\eta,\omega) &= \eta\epsilon - \omega\beta + (\eta+\omega)\left(\frac{1}{\eta+\omega}\left(0.5(\boldsymbol{f}^T\boldsymbol{F}\boldsymbol{f} - \eta\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})\right) + \frac{\eta}{\eta+\omega}\log\left((2\pi)^{-k/2}|\Sigma|^{-0.5}\right) - \log\left((2\pi)^{-k/2}|(\eta+\omega)\boldsymbol{F}|^{-0.5}\right)\right),\\
&= \eta\epsilon - \omega\beta + \left(0.5(\boldsymbol{f}^T\boldsymbol{F}\boldsymbol{f} - \eta\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}) + \eta\log\left((2\pi)^{-k/2}|\boldsymbol{\Sigma}|^{-0.5}\right) - (\eta+\omega)\log\left((2\pi)^{-k/2}|(\eta+\omega)\boldsymbol{F}|^{-0.5}\right)\right),\\
&= \eta\epsilon - \omega\beta + \left(0.5(\boldsymbol{f}^T\boldsymbol{F}\boldsymbol{f} - \eta\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}) - 0.5\eta\log|2\pi\boldsymbol{\Sigma}| + 0.5(\eta+\omega)\log|2\pi(\eta+\omega)\boldsymbol{F}|\right),\\
&= \eta\epsilon - \omega\beta + \frac{1}{2}\left(\boldsymbol{f}^T\boldsymbol{F}\boldsymbol{f} - \eta\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} - \eta\log|2\pi\boldsymbol{\Sigma}| + (\eta+\omega)\log|2\pi(\eta+\omega)\boldsymbol{F}|\right),
\end{aligned}$$

with $\boldsymbol{F} = (\eta\boldsymbol{\Sigma}^{-1} - 2\boldsymbol{R})^{-1}$ and $\boldsymbol{f} = \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{r}$. The new policy is given as $\pi(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{F}\boldsymbol{f}, \boldsymbol{F}(\eta+\omega))$.

7.4 Weighted Maximum-Likelihood Updates

Maximum-likelihood estimation (MLE) is a probability density estimation method that estiamates the unknown parameters $\boldsymbol{\theta}$ of a parametric distribution given observations $X_1, X_2, ..X_n$. Intuitively speaking, it tries to get a good estimate of $\boldsymbol{\theta}$ that *maximizes* the probability or *likelihood* of reproducing the observed data. The likelihood function is denoted by L and is given as

$$L(\boldsymbol{\theta}) = P(X_1 = \boldsymbol{x}_1, X_2 = \boldsymbol{x}_2, ..X_n = \boldsymbol{x}_n).$$

With i.i.d data points we can expand this into

$$P(X_1 = \boldsymbol{x}_1).P(X_2 = \boldsymbol{x}_2)..P(X_n = \boldsymbol{x}_n) = f(\boldsymbol{x}_1; \boldsymbol{\theta}).f(\boldsymbol{x}_2; \boldsymbol{\theta})..f(\boldsymbol{x}_n; \boldsymbol{\theta}) = \Pi_{i=1}^{n} f(\boldsymbol{x}_i; \boldsymbol{\theta}).$$

where f is the probability density function and $f(x_i; \boldsymbol{\theta})$ is read as the probability of getting $\boldsymbol{x}_i$ given parameters $\boldsymbol{\theta}$. Now we do the well-known 'log trick' to make maximizing L easier. [1] Maximize the log-likelihood

$$\max_{\theta} \log L(\boldsymbol{\theta}) = \sum_{i=1}^{n} f(\boldsymbol{x}_i; \boldsymbol{\theta}).$$

The probability density function of a multivariate Gaussian distribution is defined as

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp\Big(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\Big),$$

where n denotes the dimensionality of parameters of the distribution. And addition of a diagonal weighting matrix $\boldsymbol{D}$ is added to obtain the weighted version of the distribution. The aim is to estimate the parameters of the distribution $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ using the MLE. The log-likelihood function for the Gaussian is

$$\begin{aligned}
\log\ L(\boldsymbol{\theta} = \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \log \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \\
&= \sum_{i=1}^{N} \log\Big(\frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp\Big(-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu})^T d_i \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu})\Big), \\
&= \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}}\right) + \sum_{i=1}^{N} -\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu})^T d_i \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}), \\
&= -\frac{N}{2} \log |\boldsymbol{\Sigma}| + \sum_{i=1}^{N} -\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu})^T d_i \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}) + \text{const.}
\end{aligned}$$

To obtain the solution for the constant mean $\boldsymbol{\mu}$, maximize the log-likelihood function w.r.t $\boldsymbol{\mu}$

$$\frac{\partial \log L}{\partial \mu} = \sum_{i=1}^{N} d_i (\boldsymbol{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1},$$

and setting to zero the result is

$$\hat{\boldsymbol{\mu}} = \frac{\sum_{i=1}^{N} d_i \boldsymbol{x}_i}{\sum_{i=1}^{N} d_i},$$

For the covariance $\boldsymbol{\Sigma}$, using the 'trace trick' $\frac{\partial}{\partial \boldsymbol{A}} \text{tr}(\boldsymbol{AB}) = \frac{\partial}{\partial \boldsymbol{A}} \text{tr}(\boldsymbol{BA}) = \boldsymbol{B}^T$

$$\begin{aligned}
\log\ L(\boldsymbol{\theta}) &= -\frac{N}{2} \log |\boldsymbol{\Sigma}| + \sum_{i=1}^{N} -\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu})^T d_i \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}), \\
&\propto -\frac{N}{2} \log |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{i=1}^{N} \text{tr}\Big((\boldsymbol{x}_i - \boldsymbol{\mu})^T d_i \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu})\Big), \\
&= -\frac{N}{2} \log |\boldsymbol{\Sigma}| - \frac{1}{2} \sum_{i=1}^{N} \text{tr}\Big(\boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu})^T d_i (\boldsymbol{x}_i - \boldsymbol{\mu})\Big), \\
&= -\frac{N}{2} \log |\boldsymbol{\Sigma}| - \frac{1}{2} \text{tr}\Big(\boldsymbol{\Sigma}^{-1} \sum_{i=1}^{N} (\boldsymbol{x}_i - \boldsymbol{\mu})^T d_i (\boldsymbol{x}_i - \boldsymbol{\mu})\Big).
\end{aligned}$$

Finally, using $\frac{\partial}{\partial \boldsymbol{A}} \log |\boldsymbol{A}| = \boldsymbol{A}^{-T}$

$$\hat{\boldsymbol{\Sigma}} = \frac{\sum_{i=1}^{N} d_i (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^T}{\sum_{i=1}^{N} d_i}.$$

[1] We are able to do this because we are maximizing, and log is a monotonously increasing function which doesn't change the outcome of the maximization.

Bibliography

[Abdolmaleki et al., 2015a] Abdolmaleki, A., Lau, N., Reis, L. P., Neumann, G., et al. (2015a). Regularized covariance estimation for weighted maximum likelihood policy search methods. In *IEEE-RAS International Conference on Humanoid Robots*, volume 2015, pages 154–159.

[Abdolmaleki et al., 2015b] Abdolmaleki, A., Lioutikov, R., Peters, J. R., Lau, N., Reis, L. P., and Neumann, G. (2015b). Model-based relative entropy stochastic search. In *Advances in Neural Information Processing Systems*, pages 3537–3545.

[Abdolmaleki et al., 2017a] Abdolmaleki, A., Price, B., Lau, N., Reis, L. P., and Neumann, G. (2017a). Deriving and improving cma-es with information geometric trust regions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 657–664. ACM.

[Abdolmaleki et al., 2017b] Abdolmaleki, A., Price, B., Lau, N., Reis, P., Neumann, G., et al. (2017b). Contextual cma-es.

[Abdolmaleki et al., 2016] Abdolmaleki, A., Simões, D., Lau, N., Reis, L. P., and Neumann, G. (2016). Contextual relative entropy policy search with covariance matrix adaptation. In *Autonomous Robot Systems and Competitions (ICARSC), 2016 International Conference on*, pages 94–99. IEEE.

[Abdulsamad et al., 2014] Abdulsamad, H., Buchholz, T., Croon, T., and Khoury, M. E. (2014). Playing tetherball with compliant robots.

[Akimoto et al., 2012] Akimoto, Y., Nagata, Y., Ono, I., and Kobayashi, S. (2012). Theoretical foundation for cma-es from information geometry perspective. *Algorithmica*, 64(4):698–716.

[Arulkumaran et al., 2017] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.

[Auger and Hansen, 2012] Auger, A. and Hansen, N. (2012). Tutorial cma-es: evolution strategies and covariance matrix adaptation. In *GECCO (Companion)*, pages 827–848.

[Bellman, 1957] Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684.

[Beyer, 2007] Beyer, H.-G. (2007). Evolution strategies. *Scholarpedia*, 2(8):1965.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

[Buchanan, 2005] Buchanan, B. G. (2005). A (very) brief history of artificial intelligence. *Ai Magazine*, 26(4):53.

[Connell and Mahadevan, 2012] Connell, J. H. and Mahadevan, S. (2012). *Robot learning*, volume 233. Springer Science & Business Media.

[Deisenroth et al., 2013] Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142.

[Duan et al., 2016] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338.

[Hansen, 2016] Hansen, N. (2016). The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.

[Hansen et al., 2003] Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18.

[Hansen and Ostermeier, 2001] Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195.

[Ijspeert et al., 2013] Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373.

[Ijspeert et al., 2003] Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1547–1554.

[Kober and Peters, 2009] Kober, J. and Peters, J. R. (2009). Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856.

[Kormushev et al., 2010] Kormushev, P., Calinon, S., and Caldwell, D. G. (2010). Robot motor skill coordination with em-based reinforcement learning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3232–3237. IEEE.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

[Kupcsik et al., 2013] Kupcsik, A. G., Deisenroth, M. P., Peters, J., Neumann, G., et al. (2013). Data-efficient generalization of robot skills with contextual policy search. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013*, pages 1401–1407.

[Lattimore et al., 1961] Lattimore, R., Baskin, L., et al. (1961). *The Iliad of Homer*. University of Chicago Press.

[Mannor et al., 2003] Mannor, S., Rubinstein, R. Y., and Gat, Y. (2003). The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 512–519.

[Markov, 1961] Markov, A. (1961). Theory of algorithms (academy of sciences of the ussr, moscow, 1954). *English translation by the Israel Program for Scientific Translations*.

[Molga and Smutnicki, 2005] Molga, M. and Smutnicki, C. (2005). Test functions for optimization needs. *Test functions for optimization needs*, 101.

[Muelling et al., 2010] Muelling, K., Kober, J., and Peters, J. (2010). Learning table tennis with a mixture of motor primitives. In *Humanoids Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 411–416. IEEE.

[Ollivier et al., 2017] Ollivier, Y., Arnold, L., Auger, A., and Hansen, N. (2017). Information-geometric optimization algorithms: A unifying picture via inivariance principles. *Journal of Machine Learning Research*, 18(18):1–65.

[Osa et al., 2018] Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., Peters, J., et al. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179.

[Parisi et al., 2015] Parisi, S., Abdulsamad, H., Paraschos, A., Daniel, C., and Peters, J. (2015). Reinforcement learning vs human programming in tetherball robot games. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 6428–6434. IEEE.

[Peters et al., 2010] Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *AAAI*, pages 1607–1612. Atlanta.

[Rechenberg, 1978] Rechenberg, I. (1978). Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer.

[Rückstieß et al., 2008] Rückstieß, T., Felder, M., and Schmidhuber, J. (2008). State-dependent exploration for policy gradient methods. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 234–249. Springer.

[Sakagami et al., 2002] Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N., and Fujimura, K. (2002). The intelligent asimo: System overview and integration. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2478–2483. IEEE.

[Schaal, 2009] Schaal, S. (2009). The sl simulation and real-time control software package. *University of Southern California*.

[Schulman et al., 2015] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

[Schwefel, 1987] Schwefel, H.-P. (1987). Collective phenomena in evolutionary systems.

[Shannon, 2001] Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55.

[Stulp and Sigaud, 2012] Stulp, F. and Sigaud, O. (2012). Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*.

[Sun et al., 2009] Sun, Y., Wierstra, D., Schaul, T., and Schmidhuber, J. (2009). Efficient natural evolution strategies. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 539–546. ACM.

[Sutton et al., 1998] Sutton, R. S., Barto, A. G., et al. (1998). *Reinforcement learning: An introduction*. MIT press.

[Tangkaratt et al., 2017] Tangkaratt, V., van Hoof, H., Parisi, S., Neumann, G., Peters, J., Sugiyama, M., et al. (2017). Policy search with high-dimensional context variables. In *AAAI*, pages 2632–2638.

[Vent, 1975] Vent, W. (1975). Rechenberg, ingo, evolution strategy optimization of technical systems according to principles of biological evolution. 170 p. with 36 illustrations by frommann-holzboog-verlag. stuttgart 1973. paperback. *Feddes Repertory*, 86(5):337–337.

[Xie et al., 2018] Xie, Z., Berseth, G., Clary, P., Hurst, J., and van de Panne, M. (2018). Feedback control for cassie with deep reinforcement learning. *arXiv preprint arXiv:1803.05580*.

[Yoshiike et al., 2017] Yoshiike, T., Kuroda, M., Ujino, R., Kaneko, H., Higuchi, H., Iwasaki, S., Kanemoto, Y., Asatani, M., and Koshiishi, T. (2017). Development of experimental legged robot for inspection and disaster response in plants. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 4869–4876. IEEE.