Accelerated Policy Search

Beschleunigte Strategiesuche
Master thesis by Jan Rathjens
Date of submission: January 24, 2021

1. Review: M.Sc. Boris Belousov
2. Review: Ph.D. Davide Tateo
3. Review: Prof. Jan Peters, Ph.D.
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

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

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 24. Januar 2021

J. Rathjens

Abstract

Policy search is among the most successful approaches to learning on real robots. While impressive results were achieved in the past, policy search methods are limited in their applicability as they are generally sample-inefficient and are known to suffer from local optima. A reasonable approach to tackle these problems emerges from advanced acceleration concepts in mathematical optimization. Acceleration techniques are proven to enhance the convergence properties of gradient-based optimization algorithms if certain mathematical conditions are satisfied. However, it is unclear how these concepts transfer to policy search. In this thesis, acceleration of policy search is studied on the basis of a particular acceleration scheme applicable to mirror descent. Inspired by Gaussian accelerated mirror descent search (G-AMDS), we derive two practical mirror descent variants operating in an infinite-dimensional setting: distributional mirror descent (DMD) and accelerated distributional mirror descent (ADMD), where the latter incorporates the acceleration scheme at hand. Based on these two algorithms the effects of accelerating policy search are researched in an experimental study on continuous control problems. Further, ADMD is compared to state-of-the-art policy search methods both practically and theoretically. Our results suggest, that while acceleration in policy search generally enhances convergence properties, acceleration techniques are more valuable in different optimization settings.

Zusammenfassung

Die Strategiensuche gehört zu den erfolgreichsten Ansätzen für das Lernen auf realen Robotern. Während in der Vergangenheit beeindruckende Ergebnisse erzielt wurden, sind Strategiensuchmethoden in ihrer Anwendbarkeit begrenzt, da sie im Allgemeinen stichprobenineffizient sind und bekanntermaßen unter lokalen Optima leiden. Ein sinnvoller Ansatz zur Lösung dieser Probleme ergibt sich aus fortgeschrittenen Beschleunigungskonzepten in der mathematischen Optimierung. Es ist erwiesen, dass Beschleunigungstechniken die Konvergenzeigenschaften von gradientenbasierten Optimierungsalgorithmen verbessern, wenn bestimmte mathematische Bedingungen erfüllt sind. Es ist jedoch unklar, wie sich diese Konzepte auf die Strategiensuche übertragen lassen. In dieser Arbeit wird die Beschleunigung der Strategiensuche auf der Basis eines speziellen Beschleunigungsschemas für den Mirror Descent Algorithmus untersucht. Inspiriert von dem Gaussian Accelerated Mirror Descent Search Algorithmus (G-AMDS) leiten wir zwei praktische Mirror Descent Varianten ab, die in einer unendlichdimensionalen Umgebung arbeiten: Distributional Mirror Descent (DMD) und Accelerated Distributional Mirror Descent (ADMD), wobei letzterer das vorliegende Beschleunigungsschema beinhaltet. Basierend auf diesen beiden Algorithmen werden in einer experimentellen Studie auf kontinuierlichen Regelungsproblemen die Auswirkungen einer beschleunigten Strategiensuche untersucht. Darüber hinaus wird ADMD sowohl praktisch als auch theoretisch mit State-of-the-Art Strategiensuchmethoden verglichen. Unsere Ergebnisse deuten darauf hin, dass die Beschleunigung der Strategiensuche zwar generell die Konvergenzeigenschaften verbessert, die Beschleunigungstechniken aber in anderen Optimierungsproblemen wertvoller sind.

Contents

1. Introduction

From quadrupedal machines walking in wooded terrain [1] to juggling robot arms [2]: with the advancements in reinforcement learning, robots today can learn impressive skills - without requiring a team of expert engineers as the robot learns everything it needs to know by itself.

Among the most successful approaches to learning on real robots is policy search [3]. Roughly speaking, instead of using a handcrafted controller, in policy search an agent explores its environment and adapts its behavior or policy based on made observations. The goal of policy search is to find an optimal policy for the agent with respect to solving an associated task [4].

However, policy search methods are considered sample inefficient, since the agent usually needs a large number of interactions with its environment to find good policies [5]. In addition, policy search is known to suffer from local optima, if not already in the solution basin [4]. For these reasons, policy search methods are limited in their applicability, and agents are typically pre-trained in a computer simulation before being deployed in a real-world scenario.

One way to tackle sample inefficiency of policy search methods might be transferring acceleration concepts from mathematical optimization or short optimization to policy search methods. Acceleration refers to techniques applied to gradient-based optimization algorithms in mathematics, to find optimal solutions with respect to an objective function more quickly and overcome local shallow optima [6]. It stands to reason that acceleration schemes can also be applied to policy search since in its essence policy search is an optimization problem. Therefore, accelerating policy search algorithms could potentially lead to better convergence properties.

This thesis aims at exploring the effects of acceleration in the context of policy search. For this purpose, we make use of the well understood-mirror descent algorithm [7], for which a generic acceleration scheme has been proposed in [8]. As the justification of acceleration

usually only depends on algebraic arguments and has no intuitive understanding [9], we carefully derive a mirror descent variant applicable to policy search which we call distributional mirror descent (DMD). Inspired by the work of [10], the generic acceleration scheme of [8] is incorporated into DMD, yielding an accelerated version of DMD which we call accelerated distributional mirror descent (ADMD). Based on these two algorithms, the effects of acceleration are researched in an experimental study.

This thesis is structured as follows. In chapter 2, we briefly review the policy search problem and formally describe mirror descent with a corresponding generic acceleration scheme. Chapter 3 derives DMD as a mirror descent variant in an infinite-dimensional setting and incorporates an acceleration scheme yielding ADMD. In chapter 4, methods used a conducted experimental study are presented. In chapter 5, we evaluate ADMD with a comparison to DMD and different state-of-the-art policy search algorithms. Chapter 6 discusses theoretical properties of DMD and ADMD. Besides, an outlook for future work is provided. Chapter 7 concludes with a summary of our findings.

2. Background

This chapter formally introduces concepts on which we build our approach to explore the effects of accelerated policy search. We start by briefly reviewing the policy search problem in a black-box setting. Then, the mirror descent algorithm is motivated and described, and a particular instance is presented, which we later extend to our needs. The chapter closes with a short introduction to acceleration principles and outlines the acceleration scheme used in our approach.

2.1. Policy Search in a Black-Box Setting

In the reinforcement learning setup, an agent is placed in an environment in which it has to solve a task. For example, a simple task could be for a point mass to walk from point A to point B. In general, the agent executes a number of actions during the interaction with its environment. The actions are chosen from a policy characterized by some parameters $\theta \in \Theta$, where $\Theta \subset \mathbb{R}^d$ is referred to as parameter space. The environment responds to the agent following a specific policy over time with a trajectory $\tau \in T$ from the set T of possible trajectories. A trajectory can be viewed as a sequence of different states an agent has gone through during his interactions. In reinforcement learning, each trajectory can be associated with a numerical value assessing its quality. This map is referred to as the return function $R : T \to \mathbb{R}$. To continue the above example, the return of a robot walking from one point to another could depend on the taken time and whether it tripped or not. Due to the stochastic nature of policies and most real-world environments, τ is a random variable on the set of trajectories T influenced by the specific policy the agent follows, i.e. $\tau \sim p(\tau|\theta)$. The goal of policy search is to find a policy's optimal parameters θ^* which maximize an agent's expected return. Formally, we want to find $\arg\max_{\theta\in\Theta} r(\theta)$, where $r(\theta) = \mathbb{E}_{\tau\sim p(\tau|\theta)}[R(\tau)]$. Hence, policy search in its essence is an optimization problem.

Policy search assumes a search distribution over the parameter space [4], which formally corresponds to a probability measure $p \in P$, where P is a set of probability measures on Θ. The search distribution has the benefit of providing a natural exploration strategy of the parameter space. Instead of directly optimizing over the policy's parameters, one instead optimizes the search distribution. Formally we want to find $p^* = \arg\max_{p \in P} J(p)$, where $J(p)$ is the defined as follows:

$$J(p) = \int_{\Theta} r \; dp \tag{2.1}$$

In this thesis the policy search problem is viewed from black-box perspective, i.e. given some parameters θ, we can observe an output of the return function in the form of $R_{\tau \sim p(\tau|\theta)}(\tau)$. In particular, the sampled trajectory $\tau \sim p(\tau|\theta)$ is not observed and we do not know the return function R, but can only observe its output. Policy search methods that only need a few samples to find an optimal search distribution over the parameter space are referred to as sample efficient.

2.2. Mirror Descent

Mirror descent is a family of optimization algorithms proposed in [7]. It can be viewed as a generalization of the popular gradient descent algorithm in the sense that it can be applied in more general vector spaces and thus, enabling the utilization of non-euclidean geometry for an optimization process. This section outlines mirror descent. We first motivate the algorithm with a general optimization objective, then provide a formal description and close with a specific instance of mirror descent, which is extended to our needs in chapter 3.

2.2.1. Motivation for mirror descent

To motivate mirror descent, let us assume a general optimization objective $J : X \to \mathbb{R}$, where X is a subset of a normed vector space V. If we want to apply the common gradient descent algorithm in order to minimize the objective, we start from an arbitrary point $x_k \in X$, compute or estimate the gradient of J at x_t denoted by $\nabla J(x_t)$, scale the gradient by some step size and subtract it from x_k such that we arrive at a new point x_{k+1}. This procedure is repeated until we converge to a minimum. From a linear algebra point of view,

the subtraction of the gradient $\nabla J(x_k)$ from a point x_k seems odd: while x_k is an element of V, the gradient is a linear map on V and as such an element of a different vector space, namely of V's dual space V^*. In general, this means there is no vector addition between elements of V and V^* defined and one can not subtract the gradient from points in V as it is required in gradient descent. The reason why gradient descent still works is that it is mostly applied in Euclidean vector spaces, in which V and V^* are isomorphic to each other as shown in the Riesz representation theorem [11]. If X is a subset of more general vector spaces, e.g. Banach spaces where V and V^* are not isomorphic, gradient descent can not be applied consistently. In order to still do gradient-based optimization in Banach spaces, mirror descent maps elements of X to a corresponding element in the dual space where the gradient can be subtracted. Then, the new element is mapped back to the original vector space also called primal space. The central element of mirror descent is the so-called mirror map used to map between the primal and dual vector space. They are formally introduced next following [12].

2.2.2. Formal description of mirror descent

Let D be a convex open subset of a Banach space V and X a compact convex optimization set, such that X is included in D's closure, i.e. $X \subset \overline{D}$, and $X \cap D \neq \emptyset$. A map $\phi : D \to \mathbb{R}$ is referred to as mirror map if it satisfies the following conditions: ϕ be must strictly convex and continuously differentiable, the gradient must take all possible values in the dual space and ϕ must diverge on the boundary of D.

With the help of a mirror map, the procedure of mirror descent illustrated in figure 2.1 can be described as follows: An element $x_k \in X$ is mapped to its corresponding dual element with $\nabla\phi(x_k)$. In the dual space the gradient subtraction takes place, i.e. $\nabla\phi(y_{k+1}) = \nabla\phi(x_k) - \eta\nabla J(x_k)$. This point is then mapped back to the primal with $y_{k+1} = (\nabla\phi(y_{k+1}))^{-1}$. Since ϕ acts on D and not only on X, y_{k+1} might lie outside of X. Hence, y_{k+1} is mapped to the element in $X \cap D$, which is closest to y_{k+1} in terms of the Bregman divergence w.r.t. ϕ, i.e. $x_{k+1} = \arg\min_{x \in X \cap D} B_\phi(x, y_{k+1})$.

One can show that mirror descent reduces to the following procedure [12]:

$$x_{k+1} = \underset{x \in X \cap D}{\arg\min}\, \eta\langle \nabla J(x_k), x\rangle + B_\phi\left(x, x_k\right) \tag{2.2}$$

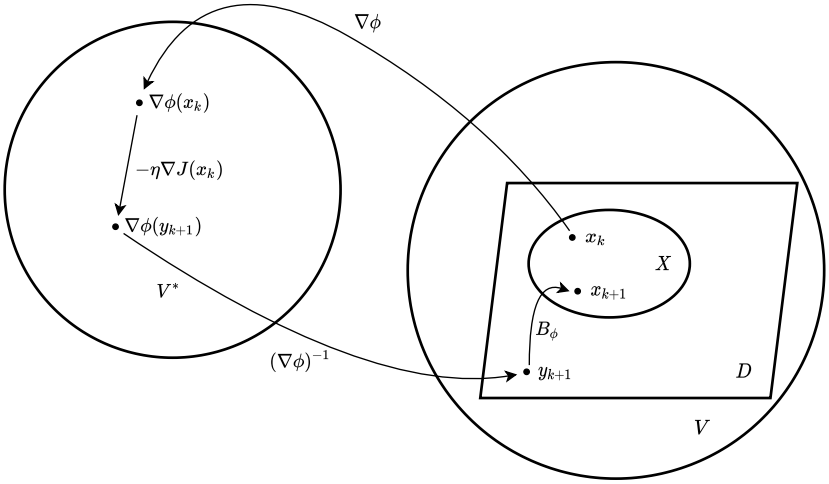


Figure 2.1.: Illustration of mirror descent. A point x_k in the optimization set X is mapped to a corresponding point in the dual space with help of a mirror map ϕ as $\nabla\phi(x_k) \in V^*$. In V^* the gradient update takes place yielding the point $\nabla\phi(y_{k+1})$. The point is mapped back to V with $(\nabla\phi)^{-1}(y_{k+1})$ and projected on X with the Bregman divergence w.r.t to ϕ.

where $\langle\cdot,\cdot\rangle : V \times V^* \to \mathbb{R}$ is the natural pairing of V and V^*. Coming from an optimization background where mostly gradient descent is used, mirror descent might seem unintuitive, however, the convergence can be proven if the objective and the step size satisfy certain conditions [12].

2.2.3. Example: entropic mirror descent

With different choices of Banach spaces and mirror maps, different instances of mirror descent can be obtained. For example, if X is a convex subset of an Euclidean vector space and the mirror map is equal to $\phi(x) = \frac{1}{2}\|x\|_2^2$, standard gradient descent is obtained. Another instance of mirror descent is entropic mirror descent [13], which is outlined next and later extended to our needs in chapter 3.

Entropic mirror descent can be applied, when the optimization set X is the d-dimensional unit simplex, i.e. $X = \left\{x \in \mathbb{R}^d : \sum_{i=1}^d x(i) = 1, x(i) \geqslant 0\right\}$ which formally can modeled as a subset of the Banach space ℓ^1 with norm $\|\cdot\|_1$, the space of sequences whose series is absolutely summable. One can show that in this case the negative Shannon entropy

$\phi_e(x) = \sum_{i=1}^{n} x(i) \ln x(i)$ acting on the domain D of d-dimensional sequences consisting of positive real numbers, can be used as a mirror map [13] yielding a mirror descent variant which authors refer to as entropic mirror descent. With the Karush–Kuhn–Tucker conditions a closed form solution for (2.2) can be obtained, e.g. for a maximization problem: $x_{k+1}(i) = \frac{x_k(i) e^{\eta(\nabla J(x_k)(i))}}{c}$, where c is a normalization constant, such that $x_{k+1} \in X$.

Since discrete probability distributions can be modeled as elements of a unit simplex [14], one could use entropic mirror descent to optimize over a set of discrete probability distributions making it interesting for our objective (2.1). However, as explained in section 2.1, policy search usually deals with continuous parameter spaces, which can not be covered by a discrete probability distribution. In [10] this problem is evaded by discretizing a continuous probability distribution in each iteration of mirror descent and refitting a continuous probability distribution to the updated discrete distribution. This makes it straightforward to apply entropic mirror descent to policy search and even accelerate with the acceleration scheme in [8]. However, while this approach works fine for low dimensional policies, it is infeasible for high dimensional policies, as the computational effort for discretizing a continuous probability distribution grows exponentially with the number of dimensions.

2.3. Acceleration

Acceleration in mathematical optimization refers to refinements of gradient-based optimization algorithms with the goal of improving their convergence rate. Among many techniques, well-known examples include Nesterov's accelerated gradient descent [15] or Adaptive Moment Estimation [16]. In general, it is hard to gain an intuition of why the acceleration helps the algorithm to converge faster. Aside from its success in practice, justification of acceleration is usually purely based on algebraic arguments, which are often specific to properties of the respective optimization problems [9]. For example, Nesterov's original proposal of accelerated gradient descent explicitly relies on arguments of Euclidean geometry and is thus not applicable in a general mirror descent setting. In [8] a generic acceleration scheme for mirror descent setting was proposed which is briefly presented next.

2.3.1. Accelerated mirror descent

The proposed acceleration of mirror descent in [8] is based on the observation that gradient-based optimization techniques, as well as their acceleration, can be interpreted as discretizations of ordinary differential equations (ODEs). Combining ODE interpretations of mirror descent and Nesterov's accelerated gradient descent, the authors come up with an ODE for an accelerated mirror descent scheme, which when discretized results in the following iterative procedure for minimizing an objective J with $x \in X$:

$$\begin{aligned} z_{k+1} &= \arg\min_{z \in X \cap D} \frac{k\eta}{v} \left\langle \nabla J\left(x_k\right), z \right\rangle + B_\phi\left(z, z_k\right) \\ y_{k+1} &= \arg\min_{y \in X \cap D} \gamma\eta \left\langle \nabla J\left(x_k\right), y \right\rangle + R\left(y, x_k\right) \\ x_{k+1} &= \lambda_k z_{k+1} + \left(1 - \lambda_k\right) y_{k+1}, \text{ with } \lambda_k = \frac{v}{v+k} \end{aligned} \tag{2.3}$$

where x_0 is an arbitrary point in the optimization set X, $y_0 = x_0, z_0 = x_0$, γ, η and v are hyperparameters, Φ is the same mirror function as in (2.2) and $R : X \times X \to \mathbb{R}$ is a called regularization function, satisfying the following conditions: For all $x, x' \in X$, there exist $0 < l_R \leq L_R$ such that $\frac{\ell_R}{2} \left\|x - x'\right\|^2 \leq R\left(x, x'\right) \leq \frac{L_R}{2} \left\|x - x'\right\|^2$, where $\|\cdot\|$ is the norm associated with the Banach space in which X is embedded. Note that in the original publication v is denoted by r and η is denoted by s. We changed the denotation to avoid confusion with the reward function and to stay consistent with the unaccelerated version of mirror descent. The authors show that if the gradient ∇J is Lipschitz continuous and some further assumptions on the hyperparameters accelerated mirror descent indeed converges faster than its non-accelerated counterpart.

The authors also include a restarting scheme for the algorithm, i.e. for a minimization problem if the gradient satisfies $\left\langle x_{k+1} - x_k, \nabla J\left(x_k\right)\right\rangle > 0$ or if $\left\|x_{k+1} - x_k\right\| < \left\|x_k - x_{k-1}\right\|$, k is set to 0 and z_k is set to x_k. While no mathematical analysis of the acceleration scheme is provided, it greatly further increased the performance rates in their conducted experiments.

3. Approach

While many algorithms are used in policy search, we choose to base our exploration of accelerated policy search on mirror descent for the following reasons: Mirror descent is a generic algorithm, making many commonly used policy search algorithms such as relative entropy search [17] special instances of mirror descent [18]. In addition, a generic acceleration scheme for mirror descent has been already proposed in [8]. However, to the best of our knowledge, no accelerated mirror descent algorithm applicable to policy search exists in the literature. In [10] first attempts towards an accelerated policy search algorithm have been made. However, as pointed out in section 2.2.3 the approach is limited to low dimensional policies. For these reasons, in this chapter we derive a mirror descent variant for policy search called distributional mirror descent (DMD) and incorporate the acceleration scheme in [8] yielding an accelerated variant of DMD called accelerated distributional mirror descent (ADMD). We begin by outlining the necessary mathematical structure in policy search in order to apply mirror descent. Then, based on these results DMD and ADMD are suggested, which will be used as a basis for an experimental study described in chapter 4.

3.1. Policy Search as a Mirror Descent Problem

Mirror descent was shown to be applicable if the optimization problem meets the following criteria outlined in section 2.2: the optimization set is a compact convex subset of a Banach space, we can obtain a gradient for our objective and there exists a convenient mirror map satisfying all conditions as described in section 2.2.2. Therefore, if we can frame policy search such that all listed criteria are met, an application of mirror descent is justified for policy search. This section outlines how policy search can be framed as an optimization problem suitable for mirror descent.

3.1.1. A suitable Banach space

We first want to show that our optimization set can be embedded as a compact convex subset in a Banach space. Recall our objective function from (2.1). We want to optimize over a set of measures.

Formally, a measure μ is associated with a measurable space (X, Σ), where X is a set and Σ is a a collection of subsets of X such that Σ is a σ-algebra. In the policy search case X is the set of parameters $\Theta \subset \mathbb{R}^d$ and Σ is a Borel σ-algebra on Θ denoted by $\mathcal{B}(\Theta)$. The Borel σ-algebra is a standard way of defining a σ-algebra on $\mathbb{R}^d$ and can be understood as all d-dimensional cuboids in $\mathbb{R}^d$. The measure μ is a function assigning positive real numbers to subsets of Σ. If $\mu(X) = 1$, μ is referred to as probability measure. Our optimisation set P can therefore be described as all measures p on $\mathcal{B}(\Theta)$ for which $p(\Theta) = 1$.

With these formalities, it stands to reason to embed P in a Banach space of measures. Well known examples for these spaces are $ba(\Sigma)$, $ca(\Sigma)$ and $rca(\Sigma)$ [19]. While it can be shown that P is a convex subset of all of theses spaces, a significant problem arises when applying mirror descent in any of these spaces. As described in section 2.2, the gradient needed in (2.2) is element of the dual space of the respective vector space. To the best of our knowledge, the dual spaces for the enumerated Banach spaces are unknown or can not be described in closed form. This makes precise analysis of mirror descent in these spaces difficult.

Fortunately, there exists an alternate way of expressing probability measures with the help of the Radon-Nikodym theorem: Let μ_1 and μ_2 be two σ-finite measures on a measure space (X, Σ). If μ_2 is absolutely continuous w.r.t. μ_1, then there exists a density function $f : X \to \mathbb{R}$ such that $\mu_2(S) = \int_S f d\mu_1$ for all $S \in \Sigma$. Therefore, when choosing a suitable base measure and a set of density functions, we can implicitly optimize over a set of probability measures that are absolutely continuous to the base measure by optimizing over a set of density functions. In policy search, the σ-algebra on which a base measure is defined is $\mathcal{B}(\Theta)$. Hence, a natural choice for a base measure is the Lebesgue measure λ which assigns the usual Euclidean volumes to the elements of $\mathcal{B}(\Theta)$. For the optimization set, we assume a set F of probability density functions, such that for each $f \in F$, $\mu(S) = \int_S f d\lambda$ is a probability measure as defined above. The constraint that μ has to be absolutely continuous w.r.t λ implies that we can only optimize over probability measures, which have a density function. However, this is sufficient in policy search, as we usually expect the reward function to have at least some smoothness properties. Thus, we

can rewrite our original objective in (2.1) as follows: Find $f^* = \arg\max_{f \in F} J(f)$, where F is a set of probability density functions and $J(f)$ is defined as follows:

$$J(f) = \int_{\Theta} f r d\lambda \tag{3.1}$$

The advantage of this reformulation lies in the fact that we can consider it easier to handle Banach spaces for the formulation of policy search as a mirror descent problem, i.e. instead of measure spaces, we can now consider spaces of measurable functions. In fact, all $f \in F$ form are also element of $L^1(\Theta, \mathcal{B}, \lambda)$ or short L^1, which is the Banach space of all integrable functions w.r.t to λ. Hence, we can easily embed F in L^1, which in addition is a convex set of this space [20].

In contrast to measure spaces, the dual space of $L^1(\Theta, \mathcal{B}, \lambda)$ can be expressed in closed form as $L^\infty(\Theta, \mathcal{B}, \lambda)$ or short L^∞ which consists of all essentially bounded measurable functions w.r.t λ. Strictly speaking, L^1 and other L^p spaces consist of equivalence classes of functions and not of functions themselves, i.e. f and g are equivalent to each other if $f = g$ a.e. for the measure λ. However, in practice, it is sufficient to think of functions being element of L^1 [21].

Unfortunately, in infinite dimensional vector spaces as L^1, subsets are usually not compact [22], which is also true for F. The compactness argument is used two times in the mirror descent framework. Firstly, it guarantees a unique solution for the update step in (2.2) and secondly it guarantees the existence of an optimizer in the optimisation set. To still justify the application of mirror descent over F, we invoke the following arguments: For (2.2) a stationary point in F can be found in a closed from solution, which is shown in section 3.2, i.e. (2.2) still has a unique solution if applied to policy search. We can not proof the existence of maximizer in F for (3.1). In fact, it is easy to verify that F does not always consist of a maximizer. For example, if the maximum reward is at θ_{max}, the maximum value of $f \in F$ should be at θ_{max}. However, one could always increase the value of $f(\theta_{\text{max}})$ such that f still is element of F. On the other hand (3.1) clearly is bounded by $r(\theta_{\text{max}})$. Hence, we are fine with finding a f for which $J(f)$ gets close to $r(\theta_{\text{max}})$.

3.1.2. A gradient in L^1

The second element we need for mirror descent is the gradient of the objective (3.1) $\nabla J(f)$. As the gradient's definition is typically based on an inner product which is unavailable in L^1 , we replace it by the Fréchet derivative of $J(f)$, which extends the gradient to maps between

arbitrary Banach spaces [23]. Following its definition, the Fréchet derivative g can be computed by solving $\lim_{\|h\|_1 \to 0} \frac{\|J(f+h) - J(f) - \langle h, g \rangle\|_2}{\|h\|_1} = 0$, where $\|\cdot\|_1$ is the norm associated to L^1, $\|\cdot\|_2$ is the usual euclidean norm and $\langle \cdot, \cdot \rangle$ is the natural duality pairing between L^1 and L^∞. By inserting (3.1) we obtain $\lim_{\|h\|_1 \to 0} \frac{\left\| \int_\Theta (f+h) r d\lambda - \int_\Theta f r d\lambda - \int_\Theta h g d\lambda \right\|_2}{\|h\|_1} = 0$, which reduces to $\lim_{\|h\|_1 \to 0} \frac{\left\| \int_\Theta h r d\lambda - \int_\Theta h g d\lambda \right\|_2}{\|h\|_1} = 0$. Clearly this equation is satisfied, whenever $g = r$, thus the Fréchet derivative of $J(f)$ is r. From now on we refer to the Fréchet derivative as gradient and denote it $\nabla J(f)$ to stay consistent with the notations used in the mirror descent framework.

There are interesting points to take away from this result. Firstly, even though we are using a gradient-based framework it is sufficient to use a zeroth-order approximation of the reward function instead of a first-order approximation which is a common requirement in gradient-based optimization. Secondly, the gradient does not depend on f as it is the same for every $f \in F$. Finally, the gradient is impossible to compute, since can not estimate $r(\theta)$ for each $\theta \in \Theta$. In fact, we would have no reason to apply a policy search algorithm if we already knew the function $r(\theta)$. However, it is sufficient to estimate $r(\theta)$ from only a few samples from the distribution induced by f_k in order calculate f_{k+1}, which is shown in section 3.2.

3.1.3. A mirror map in infinite dimensions

The negative Shannon entropy is used as a mirror in entropic mirror in finite dimensions [13]. As the negative differential entropy extends the negative Shannon entropy to infinite dimensions, it seems natural to employ it as a mirror map for our purposes. The negative differential entropy on integrable functions can be defined as $h : D \to \mathbb{R}, h(f) = \int_\Theta f \ln f d\lambda$, where $D \subset L^1$. Considering how D was defined in entropic mirror descent, again, it seems natural to also extend D to infinite dimensions as follows: $D = \{d \in L^1 : d(\theta) \in (0, \infty) \text{ for all } \theta \in \Theta\}$.

The theoretical justification of the negative differential entropy h as a mirror map is challenging and requires advanced tools from functional analysis. For our purpose, it is sufficient to see that the Bregman divergence with respect to h yields the Kullback–Leibler divergence (KL divergence) for a family of exponential density functions [24]. As we will work on exponential density functions such as Gaussian distribution in practice, the KL divergence serves as a proper Bregman divergence for mirror descent. For two functions $f_1, f_2 \in F$, the KL divergence is defined as $\mathrm{KL}(f_1 \| f_2) = \int_\Theta f_1 \ln(\frac{f_1}{f_2}) d\lambda$.

3.2. Suggested Algorithms

In the previous section, we have established a suitable mathematical structure for policy search problems in order to apply mirror descent. We now want to show how the theory can be translated into practice and suggest a mirror descent algorithm for policy search and an accelerated counterpart following the work of [10].

3.2.1. Distributional mirror descent (DMD)

To stay consistent with notations used in the mirror descent framework, we assume an initial probability density function $x_0 \in X$, where $X \subset L^1$ is the set of all probability density functions w.r.t. to the Lebesgue measure λ. In addition, we assume that we can sample parameters $\theta \in \Theta$ according to probability values implied by x_0. D consists of all function in L^1, which map to the open interval $(0, \infty)$. As established in the previous section, for the update step (2.2) we can use the negative differential entropy h as a mirror map and the expected return function r as gradient, leading to the following iterative procedure:

$$x_{k+1} = \arg\min_{x \in X \cap D} \eta\langle -r, x\rangle + \int_\Theta x \ln(\frac{x}{x_k}) \mathrm{d}\lambda \tag{3.2}$$

Note that we take the negative used the negative gradient since for policy search we want to maximize the objective instead of minimizing it. Differentiating Equation (3.2) w.r.t. to x and setting it to zero yields the following closed-form solution for x_{k+1}:

$$x_{k+1}(\theta) = \frac{x_k(\theta)\mathrm{e}^{\eta r(\theta)}}{c} \tag{3.3}$$

where c is a normalization constant, such that $x_{k+1} \in X$. Rewriting (3.3) as $\frac{\mathrm{e}^{\eta r(\theta)}}{c} = \frac{x_{k+1}(\theta)}{x_k(\theta)}$, we observe that we can now express the ratio between $\frac{x_{k+1}(\theta)}{x_k(\theta)}$, which we denote as $w_k(\theta)$. Being able to express a ratio between the two densities allows us to compute quantities of the updated probability distribution with the importance sampling framework [25]. For example, we can estimate the mean μ_{k+1} induced by x_{k+1} as follows: $\mu_{k+1} = \int_\Theta \theta x_{k+1}(\theta) d\lambda = \int_\Theta \theta \frac{x_{k+1}(\theta)}{x_k(\theta)} x_k(\theta) d\lambda = \int_\Theta \theta w_k(\theta) x_k(\theta) d\lambda = \mathbb{E}_{\theta \sim x_k}[\theta w_k(\theta)] \approx \frac{1}{n}\sum_{i=1}^n (\theta_i w_k(\theta_i))$. The covariance matrix induced by x_{k+1} can be computed with similar

arguments. We can use these quantities to obtain a new parameterized probability distribution, e.g. a Gaussian distribution which then can be used to obtain new samples in the next iteration of the algorithm. As we are in essence optimizing probability distributions with mirror descent, we call the algorithm at hand distributional mirror descent (DMD).

Data: initial Gaussian density function x_0, hyperparameter η
Result: Gaussian density function x_K
for $k = 0$ **to** K **do**
    for $m = 0$ **to** M **do**
        sample parameter $\theta_m \sim x_k$;
        evaluate policy with θ_m to obtain $r(\theta_m)$;
        compute ratio $w_k(\theta_m) = \exp(\eta r(\theta_m))$;
    Fit new Gaussian density function x_{k+1} with w_k.

Algorithm 1: Distributional mirror descent (DMD)

These remarks suggest the following procedure summarised in algorithm 1. We start with an initial Gaussian distribution with a density function x_0. For K iterations, we sample M parameters from the current distribution with density x_k. For each sampled parameter θ_m the return $r(\theta_m)$ is estimated by running the corresponding policy. We then use the returns and the step size η to compute the ratio w_k with (3.3). An iteration completes by obtaining a Gaussian density function x_{k+1} with μ_{k+1} and Σ_{k+1} which are computed as described above.

3.2.2. Accelerated distributional mirror descent (ADMD)

Similar to algorithm 1, we assume an initial distribution with density $x_0 \in X$, where $X \subset L^1$ is the set of all probability density functions w.r.t. to the Lebesgue measure λ. In addition, we assume that we can sample parameters $\theta \in \Theta$ according to probability values implied by x_0. D again consists of all function in L^1, which map to the open interval $(0, \infty)$. In contrast to DMD, for the acceleration scheme as given in (2.3) two additional density functions $y_0, z_0 \in X$ are needed. They are equal to x_0 in the first iteration. One iteration of accelerated mirror descent consists of three update steps as outlined in (2.3).

Taking the negative differential entropy as mirror map and the negative return function

as gradient, the first update step is given by the following equation:

$$z_{k+1} = \underset{z \in X \cap D}{\arg\min} \frac{k\eta}{v} \langle -r, z \rangle + \int_{\Theta} z \ln(\frac{z}{z_k}) \mathrm{d}\lambda \tag{3.4}$$

With the same arguments as for (3.2), the above equation expresses the ratio between z_k and z_{k+1}. Opposed to algorithm (1), we can not apply importance sampling straightforwardly to obtain quantities of z_{k+1} as the sampled parameters stem from x_k and not from z_k. However, since we know f_k and z_k we can compute the ratio between z_k and x_k as $\frac{z_{k+1}}{z_k} \frac{z_k}{x_k}$ which is denoted by w_k^z. Analogues to to (3.2), we can use this ratio to compute quantities of the updated probability distribution.

The second update step of (2.3) requires a ℓ_R-strongly convex and L_R-smooth regularisation function R. For the acceleration scheme of entropic mirror descent, in [8] authors set R to the Bregman divergence w.r.t a smoothed negative entropy function, since the Bregman divergence w.r.t the standard entropy function is not L_R-smooth. As a result, the second update step no longer has a closed-form solution. While this problem can be solved in a finite-dimensional setting with an additional optimization algorithm, it is unclear how a smoothed entropy translates to an infinite-dimensional setting. Instead, we argue that in practice the KL divergence serves as a proper regularisation function since two density functions are within limits in terms of their distances and the non-smoothness is not relevant. With these preliminaries, the second update step is given by the following equation:

$$y_{k+1} = \underset{y \in X \cap D}{\arg\min} \eta\lambda \langle -r, y \rangle + \int_{\Theta} y \ln(\frac{y}{x_k}) \mathrm{d}\lambda \tag{3.5}$$

With the same arguments as for 3.2, (3.5) expresses the ratio between x_k and y_{k+1}.

The final update step requires computing a linear combination of the two Gaussian density functions z_k and y_k. In general, this linear combination will not yield another Gaussian distribution, but a mixture of two Gaussians, i.e. a Gaussian mixture model. As pointed out in section 2.3.1 authors improve the performance of accelerated mirror descent by adaptively restarting the algorithm if the following conditions are satisfied: $\langle x_{k+1} - x_k, \nabla J(x_k) \rangle > 0$ or $\|x_{k+1} - x_k\| < \|x_k - x_{k-1}\|$. However, it is unclear how these conditions translate to an infinite-dimensional optimization set such as in our setting since the quantities needed in both conditions can not be computed in closed form. For these reasons we use a simple fixed restarting scheme, i.e. after a predefined number of

iterations the algorithm restarts setting k to 0 and z_0, y_0 to x_0. In general, this approach also boosts the performance of acceleration schemes as shown in [26].

Data: initial Gaussian density function x_K, hyperparameters η, γ, v
Result: density function x_K
initialize $y_0 = x_0$, $z_0 = x_0$;
for $k = 0$ **to** K **do**
  for $m = 0$ **to** M **do**
    sample parameter $\theta_m \sim x_k$;
    evaluate policy with θ_m to obtain $r(\theta_m)$;
    compute z-weight $w_k^z(\theta_m) = \frac{x_k(\theta_m)}{z_k(\theta_m)} \exp(\frac{(k+1)\eta r(\theta_m)}{v})$;
    compute y-weight $w_k^y(\theta_M) = \exp(\gamma \eta r(\theta_m))$;
  fit new Gaussian density functions z_{k+1} and y_{k+1} with w_k^z and w_k^y;
  set $x_{k+1} = \lambda z_{k+1} + (1-\lambda) y_{k+1}$ with $\lambda = \frac{v}{v+k+1}$ (ADMD-2);
  if desired, set x_{k+1} to a fitted density function of x_{k+1} (ADMD-1);
if restart set $x_0 = x_K$, $z_0 = x_0$, $y_0 = x_0$ and continue with first for loop;

Algorithm 2: Accelerated distributional mirror descent (ADMD)

These remarks suggest the following procedure summarised in algorithm 2.: We start with an initial Gaussian distribution with a density function x_0. For K iterations, we sample M parameters from the current distribution with density x_k. For each sampled parameter θ_m the return $r(\theta_m)$ is estimated by running the corresponding policy. We then use the returns, the current iteration number k, and the hyperparameters η, v and γ to compute the ratio w_k^z and w_k^y according to Equations (3.4) and (3.5) . The ratios are used to fit Gaussian distributions to z_{k+1} and y_{k+1} as in DMD. The updated distributions in turn are used to either obtain a Gaussian mixture model x_{k+1} as linear combination of y_{k+1} and z_{k+1} according to the λ factor in (2.3). Alternatively, a Gaussian distribution can be fit to x_{k+1} in order to stay closer to algorithm 1. If desired, the algorithm can be restarted after a predefined number of iterations by setting k to 0 and z_0, y_0 to x_0. As the suggested algorithm is an accelerated version of DMD, we simply call it accelerated distributional mirror descent (ADMD). In order to distinguish between fitting a Gaussian distribution to x_k in every time step or using a Gaussian mixture model, we split ADMD into ADMD-1 respectively ADMD-2.

4. Methods

Optimization algorithms can be analyzed mathematically and convergence guarantees can be given under certain conditions. For example, if the optimization set is convex and the objective function is convex and smooth one can prove that mirror descent converges to the optimum with a rate of $\mathcal{O}(1/k)$ [8]. For policy search, these conditions do not hold in general: reward functions are rarely convex as they consist of multiple local optima and are not as smooth as desired [27]. To evaluate the quality of policy search algorithms, it is therefore common practice to test algorithms on toy environments, for example classic control tasks such as the inverted pendulum [28]. In order to compare algorithms 1 and 2 to explore the benefits of accelerating policy search, we followed this practice and tested both algorithms on multiple environments with the MushroomRL [29] framework. This chapter is devoted to outlining the settings under which these experiments were conducted. We first describe which and how algorithms were executed on the environments. Then the policy for which optimal parameters have to be found is introduced. The chapter closes with a brief presentation of the used test environments and their corresponding settings.

4.1. General Execution

We tested algorithm 1 (DMD) and two versions of algorithm 2 (ADMD), one where the density function corresponds to a Gaussian density function (ADMD-1) and one where the density function yields a Gaussian Mixture Model (ADMD-2). In addition, we tested two popular black-box policy search algorithms on some environments in order to put the other algorithms into perspective, namely PGPE [30] and REPS [17].

For a specific combination of hyperparameters, an algorithm was run 25 times on an environment for a fixed number of epochs. Each epoch consisted of 200 episodes, where an episode corresponds to a run of a policy on the environment for as many time steps as specified by a horizon parameter. For each executed episode, the algorithm receives a

discounted reward controlled by a discount factor, which we fixed for each environment. All algorithms use a fixed number of iterations for every epoch. To keep the number of experiments within a reasonable amount, we use two iterations per epoch for each experiment, hence each iteration consisted of 100 episodes.

4.2. Policy

We used a linear policy over the state's Fourier feature or a linear policy over the state's variables in every experiment. The initial density function was a Gaussian density with zero mean and a diagonal covariance matrix where each element of the diagonal was 0.05. This section briefly explains and justifies this choice.

4.2.1. Features

Features in policy search refer to a map $\psi : S \to \mathbb{R}^m$, where $S \subset \mathbb{R}^n$ is the state space and $\psi(s)$ are called state features, i.e. features transform an environment's state into a different representation, which is shown to be favorable for many policy search tasks [31]. Commonly used features include radial basis functions, tile codes, polynomial features or Fourier features. One of the main advantages of feature representation of a state is the convenient experience that a good policy can be obtained by a linear function of the state features, i.e. $\pi_\theta(s) := \theta^T \psi(s)$ [32].

We followed these conventions for our experiments and learned linear policies over Fourier features, as they generally yield good performance for policy search methods [33]. For each environment we used a different amount of Fourier features depending on the size of the state space detailed in section 4.3.

4.2.2. Distributions

As previously mentioned, parameters are sampled from a distribution that is updated in every iteration, begging the question of how this distribution should be initialized. A standard approach in policy search is using an initial Gaussian distribution [4], since while we are uninformed about where good parameters lie in the state space, we only want to sample parameters that are are safe for the agent, i.e. we only want to sample parameters which are close to some safe point in the parameter space. Therefore, for all

experiments the initial distribution is set to a Gaussian distribution with zero mean and a diagonal covariance matrix, where each diagonal entry is 0.05. The zero mean is chosen as it seems like an unbiased initial distribution and, the covariance was chosen to balance the mentioned safety concerns while still allowing the agent to explore the parameter space.

4.3. Test Environments

We test the algorithms on simulated toy environments which are common for evaluating policy search algorithms. In this section, each environment is briefly introduced and the hyperparameters specific to each environment are outlined, e.g. the horizon for each episode and the reward's discount factor. For a more detailed presentation of each environment including the environment's dynamics or reward function, please refer to the provided references and the MushroomRL repository [29].

4.3.1. Pendulum

In the pendulum environment [34], a link is attached to a rotating motor. The agent's goal is to balance the link in an upright position. The state space is represented by the link's angular velocity and by the sine and cosine value of the link's angle. A continuous action corresponds to the torque applied to the link by the motor. We use two versions of the pendulum. One where the environment is initialized with a random state and one where the environment is initialized with a downward oriented link with zero angular velocity. For both versions we used 27 Fourier features, the horizon was set to 200 and the discount factor was set to 1.0.

4.3.2. Cart-Pole

In the cart-pole environment [35], a link with weight is attached to a cart. The agent's goal is to balance the link in an upright position by moving the cart in a horizontal direction. A continuous state is represented by the link's vertical angle and its angular velocity. As opposed to all other test environments, cart-pole consists of discrete action space, i.e. the agent can apply maximum force in the left or right direction or no force at all. The environment is initialized with zero angular velocity at a random position within a cone

above the cart. We use 9 Fourier features, but 27 parameters in a 3×9-dimensional matrix which when multiplied with a state's features yields a 3-dimensional vector. Each entry corresponds to one of the three actions and we simply choose the action with the highest number. The horizon of each episode was set to 300 and the discount factor to 0.95.

4.3.3. Ship Steering

In the ship steering environment [36], a ship is placed at a fixed position. The agent's goal is to steer through a gate in a minimum amount of time. A continuous state is represented by the ship's two-dimensional position, its orientation angle and its turning rate. Continuous actions correspond to the ship's turning rate. We used 81 Fourier features, the horizon was set to 400 and the discount factor to 0.99.

4.3.4. Segway

In the segway environment [37], the agent's objective is to balance a link of a segway in an upright position. A continuous state is represented by the link's angle and the angular velocities of the link and the segway's wheel. The environment is initialized with a fixed state of zero velocities and a slightly tilted upward oriented link. We fixed the horizon to 300 time steps and the discount factor to 0.97.

4.3.5. LQR

The LQR (linear-quadratic regulator) [38] environment refers to a linear system whose dynamics are determined by two d-dimensional quadratic matrices and whose reward function is determined by two different d-dimensional quadratic matrices. The continuous state and action spaces are represented by $\mathbb{R}^d$ vectors. We used three versions of the LQR environment, i.e. a 3, 5 and 7-dimensional LQR problem, leading to a 9, 25 and 49-dimensional linear policy which directly acts on the state variables. In all versions, the initial state is randomly initialized, the horizon is set to 50 and the discount factor is set to 0.9.

5. Evaluation

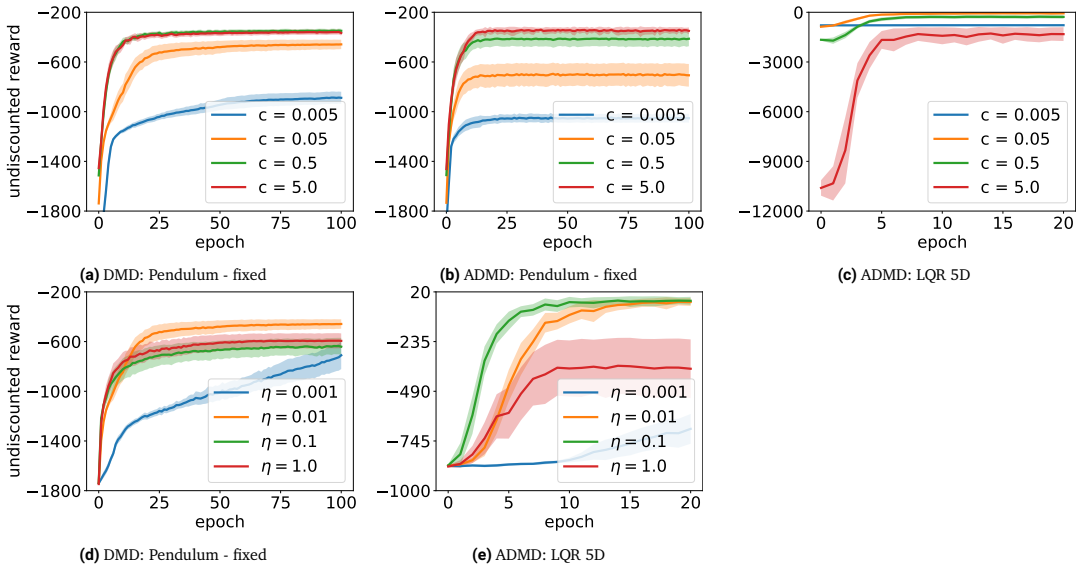
This chapter evaluates and discusses acceleration of policy search by analyzing ADMD with methods outlined in chapter 4. The evaluation is guided by the following questions: What are the properties of ADMD? How does ADMD compare to DMD? How does acceleration compare to other approaches? Each question is evaluated separately in the next sections. The chapter concludes with a summary of the evaluation. Used hyperparameters for displayed results are detailed in appendix A.

5.1. Ablation Study of ADMD

As outlined in chapter 2, acceleration schemes usually do not have an intuitive grasp as their justification purely relies on algebraic arguments. However, we still want to get a better understanding of the proposed accelerated mirror descent variant ADMD. For this purpose, we conducted an ablation study, i.e. we evaluated the algorithm's hyperparameters influence on its performance by varying the respective hyperparameter and keeping others fixed. The following section presents the results of this study with illustrative examples. For the remainder of this section, we summarize ADMD-1 and ADMD-2 under ADMD since both algorithms showed the same behavior in conducted experiments. We start with analyzing hyperparameters shared by both DMD and ADMD and then turn to hyperparameters exclusive to ADMD.

5.1.1. Effects of covariance and step size adaptation

In chapter 4, we established that the initial distribution used for sampling the policies' parameters is of Gaussian nature with zero mean and a medium-sized covariance matrix. While it is intuitive that the convergence of DMD and ADMD speeds up if the initial



(a) DMD: Pendulum - fixed **(b)** ADMD: Pendulum - fixed **(c)** ADMD: LQR 5D

(d) DMD: Pendulum - fixed **(e)** ADMD: LQR 5D

Figure 5.1.: Comparison of different covariances and step sizes for DMD and ADMD. The top row compares different covariance matrices of the initial distribution. The covariance matrix was equal to the provided c times the identity matrix. The bottom row compares different values for the step size η. All plots show the achieved undiscounted reward in a 95% confidence interval averaged over 25 runs.

distribution's mean is closer to the optimum, it is unclear how different sized covariance matrices influence the algorithms' performance.

To get a better understanding of the covariance matrix's role, we ran DMD and ADMD on multiple test environments with different initial Gaussian distributions with zero mean. The elements of the covariance matrices' diagonals corresponded to 0.005, 0.05, 0.5 and 5.0. Results are displayed in Figures 5.1a, 5.1b, 5.1c. Figures 5.1a and 5.1b compare the influence of the initial covariance on the fixed pendulum between DMD and ADMD. We observe that the choice of covariance matrix has a major impact on the performance for both algorithms and the best results are achieved with large covariances. ADMD seems to be affected more by premature convergence for small covariance matrices than DMD. Figure 5.1c illustrates the role of the initial covariance matrix on the 5D LQR environment for ADMD. Opposed to the fixed pendulum, large covariances drastically reduced the algorithm's performance. The difference between the 5D LQR environment and the fixed pendulum can be explained by the size of feasible parameter space: For LQR feasible parameters are smaller in terms of their numeric values, i.e. with large covariance matrices

infeasible parameters are sampled during the first iterations of the algorithm.

The step size η is a central hyperparameter in all gradient-based optimization algorithms. In general, moderate step sizes are desired as they bring the element sufficiently closer to the optimum in each iteration and prevent large updates which potentially decrease an element's quality. In DMD and ADMD η has a similar interpretation. It regulates how far an updated distribution x_{k+1} can be moved from a current distribution x_k in terms of the KL-divergence by weighting the gradient. The main issue with large step sizes in DMD can be deduced from (3.3): with large η, all weight will be put on a single sample resulting in a small covariance such that DMD is no longer to explore the parameter space, i.e. DMD converges prematurely.

We tested our intuition of the step size η and ran experiments on multiple environments with DMD and ADMD with varying step sizes. Illustrative results are shown in Figures 5.1d and 5.1e. For both algorithms, a moderate step size achieved the best results. Large values for η resulted in premature convergence, while small values led to slow convergence, which is consistent with the general intuition of a step size parameter. However, for ADMD the tuning of the step size is more complex, as we will see in the next section.

5.1.2. Effects of iteratively increasing step size and restart scheduling

Opposed to DMD, ADMD has an iteratively increasing step size for its z-distribution, compounded by $\frac{k\eta}{v}$. The linear growth factor of this step size is controlled by the number of the current iteration k and by the hyperparameter v, i.e. η is multiplied by $\frac{k}{v}$ each iteration. In addition, k and v also control the factor λ which determines how the z- and y-distribution are weighted in the x-distribution in each update step. For small v the z-distribution's step size grows quickly in each iteration, but the z-distribution's weight on the x-distribution decreases rapidly. For large v the step size grows slower and the z-distribution has a higher influence on x-distribution for more iterations.

As pointed out in [8], the hyperparameter v has to be greater or equal to 3 in order to guarantee convergence of accelerated mirror descent. We incorporated this insight into our study and conducted experiments with $v = 3$, $v = 10$, $v = 30$ and $v = 100$ on multiple environments. Illustrative results are shown in Figures 5.2a and 5.2b in which ADMD was run on the fixed pendulum and cart-pole environments. In both experiments, a small v led to faster convergence in the beginning but also introduced higher variance and premature convergence. On the other side, large v slowed down the convergence especially during the first epochs of the algorithm and also introduced high variance. Best results were

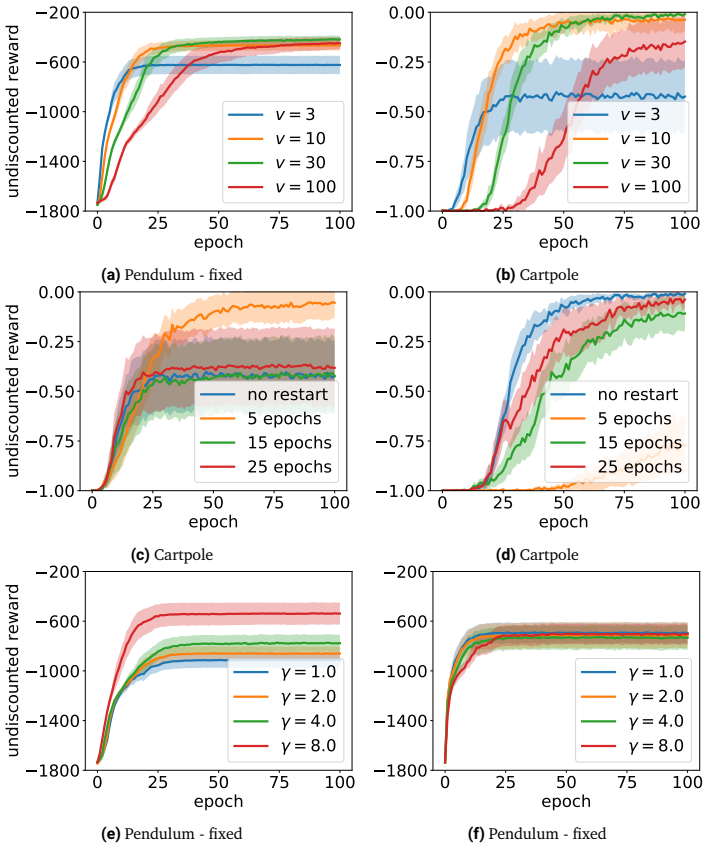


Figure 5.2.: Comparison of different settings for ADMD. The top row compares different values for the hyperparameter v on the fixed pendulum and cart-pole environment. The middle row compares different restart schedules on the cart-pole environment, i.e. ADMD was restarted every x epochs. In Figure 5.2c v was set to 3 and in Figure 5.2d v was set to 30. The bottom row compares different values for the hyperparameter γ on the fixed pendulum environment. In Figure 5.2e η was set to 0.001 and in Figure 5.2f to 0.1. All plots show the achieved undiscounted reward in a 95% confidence interval averaged over 25 runs.

obtained with moderate values. The outcomes are in line with what one would expect from v, as the hyperparameter directly influences the step size. Note that in the discussed figures the acceleration was not restarted, however, with different restarting schemes the influence of v varies more as we will see in the next section.

ADMD can be optionally restarted after a fixed number of iterations, i.e. the z and y-

distribution are set to the current x-distribution and the iteration count k is set to 0. We conducted experiments with different restarting schedules on several environments, i.e. we ran ADMD with no restarting schedule and with a restart every 5, 10 and 25 epochs. Illustrative results are shown in Figures 5.2c and 5.2d, where ADMD was run on the cart-pole environment. In Figure 5.2c ,the hyperparameter v was set to 3. We observe that restarting after a small number of epochs greatly enhanced the performance compared to no restarting while longer restarting schedules barely affected the performance. In Figure 5.2d, v was set to 30. We observe that here restarting had a counterproductive effect increased by a tighter restarting schedule. This behavior can be explained as follows: as discussed above, for small v the z-distribution's step size grows quickly. A tight restarting schedule prevents the step size from becoming too large, which would result in premature convergence. On the other hand, for large v the z-distribution's step grows slowly. In this case restarting prevents the step size from increasing resulting in worse performance.

The final hyperparameter of ADMD γ which controls the step size of the y-distribution compounded by $\eta\gamma$. Opposed to the z-distribution's step size, it is constant in every iteration. Therefore, one could replace the y-distribution's step size by a single constant. However, introducing γ allows relating the y-distribution's step size to the hyperparameter η, which is also used in the z-distribution's step size.

In [8], authors outlined, that in theory γ has to be greater or equal to 1 to guarantee convergence of the generic accelerated mirror descent scheme. Again, we incorporated this insight into our study and conducted experiments with $\gamma = 1$, $\gamma = 2$, $\gamma = 4$ and $\gamma = 8$ on multiple environments. Illustrative results are shown in Figures 5.2e and 5.2f, where ADMD was run on the fixed pendulum environment. In Figure 5.2e, the step size η was set to 0.001 and in Figure 5.2f to 0.1. We observe that for a small step size best results were achieved with large values for γ, while for a large step size different values for γ barely affected the algorithm's performance. This behavior can be explained as follows: during the first iterations of ADMD, the x-distribution was mostly influenced by the z-distribution due to the iteratively decreasing λ factor in Equation (2.3). Hence, γ barely has an effect on the x-distribution, since it does not directly affect the z-distribution's step size. Only during later iterations, γ has a greater effect on the x-distribution, as the λ value continues to decrease. However, for large step sizes η the x-distribution will have converged by that point, thus only for small step sizes γ significantly changes the performance of ADMD.

5.2. Comparison of ADMD with DMD

This section evaluates how acceleration affects the performance of policy search, i.e. how ADMD compares to DMD in terms of convergence rate and the quality of found policies. For this purpose, we individually tuned the hyperparameters of DMD, ADMD-1 and ADMD-2 on all test environments outlined in 4 and ran each algorithm with its best setting in order to get a fair comparison. Figure 5.3 displays the results.

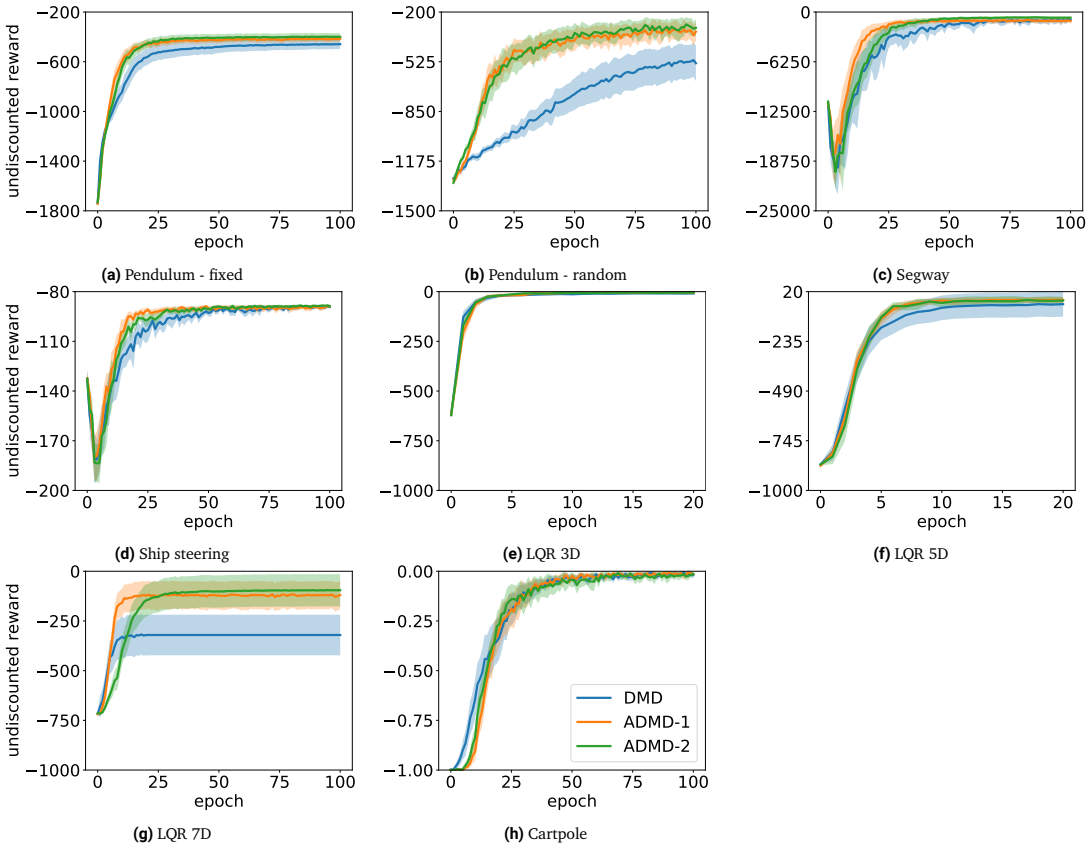


Figure 5.3.: Comparison of DMD and ADMD. Comparison of DMD (blue) with its accelerated counterpart ADMD. ADMD-1 (orange) makes use of Gaussian distributions while ADMD-2 (green) makes use of Gaussian mixture models. The plots show the achieved undiscounted reward in a 95% confidence interval averaged over 25 runs on multiple environments. Each algorithm was run with tuned hyperparameters.

On both pendulum environments (Figures 5.3a, 5.3b), ADMD-1 and ADMD-2 performed similarly and found better policies than DMD. In addition, the acceleration decreased the variance indicated by a smaller confidence interval. Particularly with a random starting position acceleration greatly increased the convergence rate of policy search on the pendulum. The difference between the two versions of the pendulum can be explained as follows: for the pendulum with a random starting position, the variance for the achieved reward is quite large as it highly depends on the starting position. Thus, DMD needs to run with small step size as otherwise the distribution may be moved to an undesired region in the parameter space. However, once the distribution gets closer to the optimum the policy will have learned to swing up the pendulum, but will not be able to balance it yet. Hence, the starting position becomes less relevant and one can take larger steps in the distribution space, which is well suited for ADMD's iteratively increasing step size. On the other hand, the fixed pendulum's reward function is deterministic making it safer to take large step sizes even in the first epochs of the algorithm.

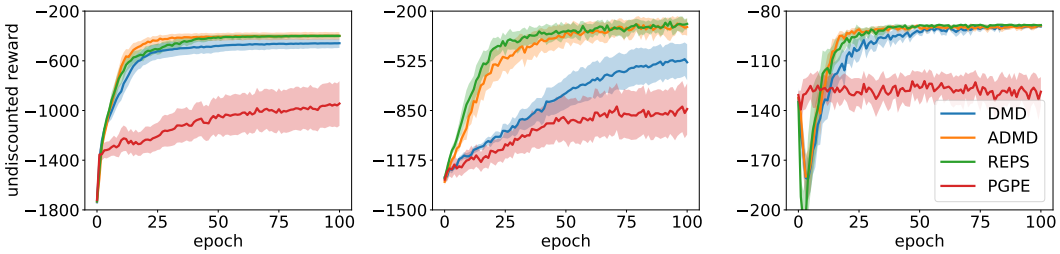
On the ship steering and segway environments (Figures 5.3d, 5.3c), both ADMD-1 and ADMD-2 performed better than DMD. While all algorithms found similar performing policies with acceleration the algorithm converged around 20 - 40 epochs earlier with ADMD-1 having a slight advantage over ADMD-2. The dent at the beginning of all algorithms on both environments is due to the nature of the reward function: the undiscounted reward decreases with longer trajectories. However, the algorithms that work with the discounted reward, i.e. intermediate rewards obtained at late time steps are not considered. Hence, for the algorithm the discounted reward increases in every epoch, but the undiscounted reward only starts to increase once the algorithms learn to pass a certain checkpoint, e.g. for ship steering to pass through the associated goal.

On the LQR environments (Figures 5.3e, 5.3f, 5.3g), the merit of acceleration increased with the number of dimensions. On the 3D LQR, all algorithms converged very similarly. On 5D LQR, ADMD-1 and ADMD-2 performed similarly and converged to better policies than DMD. In addition, ADMD ran more stable as the variance slightly decreased. On the 7D LQR, the difference between DMD and ADMD increased further, as the found policies with acceleration were significantly better than the ones without acceleration. We can also observe that on this particular environment ADMD-2 converges slower than ADMD-1, but finds a better policy. In our experiments, it turned out that on LQR 7 ADMD-2 ran better with higher v compared to ADMD-1, thus as explained in the previous section the step size increases slower.

On the cart-pole environment (Figure 5.3h), all algorithms performed similarly. DMD converged slightly faster in the first epochs, but ADMD-2 and ADMD-2 quickly drew level

with DMD yielding similarly performing policies.

5.3. Comparison of ADMD to other Policy Search Methods



(a) Pendulum - fixed **(b)** Pendulum - random **(c)** Ship steering

Figure 5.4.: Comparison of ADMD with DMD, PGPE and REPS Comparison of ADMD (orange) with DMD (blue), PGPE (red) and REPS (green). The plots show the achieved undiscounted reward in a 95% confidence interval averaged over 25 runs on multiple environments. Each algorithm was run with tuned hyperparameters.

To get a better assessment of accelerating policy search, we compared the performance of ADMD with the performance of two different popular black-box policy search algorithms, namely PGPE [30] and (episodic) REPS [17]. Both algorithms have a different approach to policy search than acceleration. REPS can be understood as a mirror descent variant utilizing a trust region approach [39], while PGPE makes use of the gradient with respect to the parameters of a distribution.

We tuned the hyperparameters of all algorithms within a reasonable amount and ran each algorithm with tuned hyperparameters on the pendulum and ship steering environments. Results of experiments are illustrated in Figure 5.4. In all experiments, PGPE performed significantly worse than the other algorithms. Both REPS and ADMD performed similarly with slight advantages for ADMD on the fixed pendulum and ship steering environment, and slight advantages for REPS on the random pendulum.

5.4. Summary of the Evaluation

We evaluated DMD, ADMD and other popular black-box policy search algorithms with different hyperparameter settings on multiple continuous control environments. The

evaluation was guided by the following questions: What are the properties of ADMD? How does ADMD compare to DMD? How does acceleration compare to other approaches? Our results can be summarized as follows: Due to the high number of hyperparameters in ADMD and their complex coherences, the algorithm is harder to tune than its non-accelerated counterpart DMD. Once the initial search distribution is chosen, DMD can be tuned by its step size only. In terms of convergence properties, ADMD increased the performance of DMD on all test environments. Especially on the random pendulum environments, the acceleration led to a significantly better convergence rate and final policy. However, on most environments, ADMD only had a minor benefit over DMD. When comparing ADMD to other black-box policy search, ADMD showed superior performance to PGPE and achieved the same convergence properties as REPS.

6. Discussion

In this chapter, we want to discuss accelerated policy search from a broader perspective and provide additional insights. First, ADMD is compared to other approaches used in policy search. Then, we assess the convergence rate of ADMD. The chapter concludes with an outlook on future work.

6.1. Comparison with different Approaches

In this section, we compare ADMD with two different approaches to policy search, namely Gaussian accelerated mirror descent search (G-AMDS) [10]m which is closely related to ADMD, and episodic relative entropy search (REPS) [17].

6.1.1. Comparison of ADMD with G-AMDS

DMD and ADMD are inspired by the work of [10] where an accelerated mirror descent algorithm called G-AMDS for black-box policy search was originally proposed. However, the authors' approach significantly differs from ours, even though the same generic mirror descent scheme in [8] was utilized. The difference can be made clear by comparing the unaccelerated version of G-AMDS called Gaussian mirror descent search (G-MDS) to DMD.

As for DMD, G-MDS begins an iteration by sampling parameters from a Gaussian probability distribution. The return of sampled parameters is then estimated by running the respective policy on an environment. Instead of using the returns to compute a ratio between the old and new distribution as done in DMD and ADMD, the authors model a discrete probability distribution over the sampled parameters. The discretization yields a straightforward way to apply entropic mirror descent respectively its acceleration to policy search and allows to omit a derivation for mirror descent on probability density functions as provided in chapter

3. At the end of each iteration, a continuous Gaussian distribution is fitted to the discrete distribution, while DMD and ADMD fit a Gaussian distribution to the ratio between the old and new distribution. However, the discretization approach suffers from computational problems: the effort for discretizing a continuous distribution grows exponentially with the dimensions of the parameter space, i.e. discretizing high dimensional policies is computationally impossible, as one would need an infinite number of samples. The authors avoid this problem by using the density values of the current density function x_k instead of true the true probability values for the discretization process, i.e. each sampled parameter θ_m in an iteration is assigned with the probability value $\frac{x_k(\theta_m)}{\sum x_k(\theta_m)}$. However, this leads to a different update for the new mean of x_{k+1} which is computed in closed form as $\mu_{k+1} = \frac{\sum \frac{x_k(\theta_i)}{\sum x_k(\theta_i)} e^{\eta * r(\theta_i)} * \theta_i}{c}$. On the other hand, the new mean in DMD can be computed as $\mu_{k+1} = \frac{\sum e^{\eta * r(\theta_i)} * \theta_i}{c}$. This basically means, that convergence is slowed down, as the distribution is inclined to stay close to the old distribution. In addition, the discretization scheme at hand makes it impossible to properly update the distribution's covariance matrix, since it unnecessarily shrinks in each iteration. Thus, in contrast to DMD and ADMD, G-MDS and G-AMDS keep the distribution's covariance fixed during the learning process and only update the mean.

6.1.2. Comparison of ADMD with REPS

As pointed out in [39], REPS can be formulated as an instance of mirror descent. In fact, the only difference in the update step between DMD and REPS is that the latter adapts the step size based on a maximum distance constraint in terms of the KL-divergence between the old and new distribution. Learning the distribution's covariance matrix in REPS has an interesting effect: an updated distribution x_{k+1} will always lay on the edge of the x_k's trust region. As a consequence, the step size scales the gradient down, if the update would move x_{k+1} outside of the trust region, and up if the update would leave x_{k+1} in the interior of the trust region. On the other hand, ADMD scales the step size of the z-distribution based on the current iteration and the hyperparameters η and v. From this perspective, both REPS and ADMD can be viewed as adaptations of DMD with the goal to improve the convergence rate utilizing a dynamic step size. As both algorithms showed similar convergence properties in conducted experiments in chapter 5, it stands to reason, that an adaptive step size is a major factor in improving the convergence rate in policy search.

6.2. Assessment of ADMD

In chapter 5, we showed that ADMD in general enhances the performance of DMD. However, on most environments the acceleration only yielded slight advantages over DMD. This result is somewhat counterintuitive as the used acceleration scheme in [8] is proven to change the convergence rate of mirror descent from $\mathcal{O}(1/k)$ to $\mathcal{O}(1/k^2)$. In this section, we want to point out two reasons for this behavior, namely the experimental setup and the convergence in L^1.

6.2.1. Experimental setup

When evaluating the influence of acceleration on the convergence rate in an experimental study, several setups are reasonable. For example, in [10] G-MDS and its accelerated counterpart G-AMDS were initialized with the same step size. Their experiments suggested, that G-AMDS significantly increases the convergence rate of G-MDS.

We used a different setup for comparing the convergence rates of DMD and ADMD. Instead of fixing a step size for both algorithms, we tuned the hyperparameters for both algorithms individually to a reasonable extent and then conducted experiments as described in chapter 4. As a result, the convergence rate of DMD was much closer to the one of ADMD than for their counterparts in [10]. These results suggest, that while acceleration in general achieves better performance in an untuned setting, in practice non-accelerated algorithms achieve similar convergence rates as their accelerated counterparts in policy search. We tested if this suggestion is also true for G-MDS and G-AMDS and achieved the same convergence rates for both algorithms on the environments described in [10].

6.2.2. Convergence rates in L^1

Convergence rates can be expressed from two different perspectives. One can either express how quickly a point x converges to an optimizer x^* or how quickly the objective value $J(x_k)$ converges to an optimum $J(x^*)$. For the accelerated mirror descent scheme we used, the authors showed that the convergence rate in terms of the distance between x_k and x^* is $\mathcal{O}(1/k^2)$ [8]. We argue that a convergence rate in terms of the distance between x_k and x^* for optimization over distributions is not as expressive as for different optimization sets. As an illustrative example, consider an arbitrary policy whose optimal

parameters are θ. Now, consider two distributions p_1 and p_2 whose mean is θ. One distribution is associated with a small-sized covariance matrix, while the other one is associated with an infinitely small covariance. For both distributions, the objective $J(p_1)$ and $J(p_2)$ will approximately output the same value, as sampled parameters are close to θ, due to the small-sized covariances. However, in terms of the L^1-norm on the difference between the distributions' density functions, the distributions have a significant distance between them. Hence, for probability distributions improving the convergence rate of an optimizer is not necessarily reflected in the convergence rate to an optimum. This suggest, that acceleration techniques might not as valuable for policy search as for different optimisation scenarios.

6.3. Outlook

In this section, we provide an outlook for what we believe contributes to a deeper understanding of acceleration in policy search, namely improving the convergence properties of ADMD, applying acceleration to step-based approaches, and trying alternative acceleration schemes.

6.3.1. Improving DMD and ADMD

We believe that the performance of both DMD and ADMD can be enhanced by using a different family of search distribution and finding an adaptive restarting scheme.

Firstly, both DMD and ADMD use Gaussian distributions or Gaussian mixture models as search distributions over the parameter space Θ. Gaussian distributions are particularly simple since they are fully parameterized by a mean and covariance matrix. While Gaussian distributions are commonly used as search distributions in policy search [4], they are also limited in their expressiveness. For example, consider a parameter space, where parameters of interest can be clustered into two regions. In Gaussian distributions sampled parameters are centered around the distribution's mean. Therefore, one can either only explore one of the regions or both regions, but with a large covariance matrix. Both options are undesirable. Alternatively, a search distribution could be modeled with normalizing flows [40] as they can express different distributions than Gaussian distributions, but also satisfy the demands for probability distribution in policy search, i.e. normalizing flows can be used for sampling and density estimation. Especially in the update steps of ADMD,

normalizing flows might enhance the convergence properties due to a large number of distribution fittings during a run of the algorithm.

Secondly, the authors of the generic acceleration scheme we used for ADMD proposed an adaptive restarting scheme [8]. Based on properties of the gradient and the distance between an optimization element and its updated version, the algorithm is restarted. Since it was unclear how these properties translate to an infinite-dimensional optimization setting, we used a simple fixed restarting schedule, i.e. after a predefined number of iterations the algorithm was restarted. Fixed restarting schedule generally enhance the convergence properties of acceleration schemes [26]. However, adaptive restarting schemes are usually superior to fixed restarting schemes [26]. Therefore, finding proper restarting conditions for ADMD can potentially enhance its performance.

6.3.2. Accelerating step-based approaches

In this thesis, we viewed the policy search problem from a black-box perspective omitting information about the return function and trajectories. On the other hand, there also exist step-based approaches to policy search that model the temporal structure of sampled trajectories. A particularly well-known step-based policy search algorithm is PPO [41]. Recently, a step-based mirror descent algorithm related to PPO was proposed in [42]. Therefore, it stands to reason, that the acceleration scheme used for our approach can also be transferred to step-based algorithms, potentially increasing the convergence rates.

6.3.3. Exploring alternative acceleration schemes

We based our exploration of accelerated policy search on the generic acceleration scheme for mirror descent proposed in [8], as mirror descent is well suited for a black-box approach to policy search. In order to fulfill the algebraic structure to the application of this acceleration scheme, we framed the policy search problem as an optimization problem on a subset of the Banach space L^1. However, policy search can be framed in alternative ways. For example, a set of probability distributions can be modeled as Riemann manifold [43] enabling to apply acceleration justified with the structure of Riemann manifolds, e.g. [44].

In our opinion, a particular interesting acceleration scheme applicable to policy search is described in [45] where acceleration is applied to particle-based variational inference. In the algorithm, the distance of two distributions is measured with the Wasserstein

distance [43] instead of the KL-divergence. The Wasserstein distance was interpreted as a horizontal distance [46] and as such should not suffer as much from premature convergence due to small covariances in updated distributions.

7. Conclusion

This thesis aimed at transferring advanced acceleration concepts from mathematical optimization to policy search in a black-box scenario and study its effects. Inspired by G-AMDS, we derived a mirror descent variant operating in an infinite-dimensional setting called distributional mirror descent (DMD). Then, we incorporated a particular acceleration scheme suitable for mirror descent algorithms into DMD yielding an accelerated version of DMD called accelerated distributional mirror descent (ADMD). Based on these two algorithms, we studied the effects of the acceleration scheme at hand in an experimental study on continuous control problems. We further discussed the theoretical properties of DMD and ADMD and their implications for optimizing over a set of probability distributions. Our results suggest that acceleration in policy search generally enhances convergence properties in the sense that better optima can be found in fewer iterations. However, the benefit of acceleration is limited in many cases and does not yield advantage over simpler trust-region approaches. Nevertheless, we believe that a deeper understanding of accelerated policy search can be obtained by future research. The following approaches are particularly interesting: Firstly, acceleration can be transferred to step-based algorithms. Secondly, probability distributions can be modeled as elements of a Riemann manifolds enabling the utilization of different acceleration schemes.

Bibliography

[1] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, p. eabc5986, Oct 2020.

[2] K. Ploeger, M. Lutter, and J. Peters, "High acceleration reinforcement learning for real-world juggling with binary rewards," *arXiv preprint arXiv:2010.13483*, 2020.

[3] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A survey on policy search algorithms for learning robot controllers in a handful of trials," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 328–347, 2019.

[4] M. P. Deisenroth, G. Neumann, J. Peters, and Others, "A survey on policy search for robotics," *Foundations and Trends®in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[5] Y. Yu, "Towards sample efficient reinforcement learning," in *IJCAI*, pp. 5739–5743, 2018.

[6] A. Wibisono and A. C. Wilson, "On accelerated methods in optimization," *arXiv preprint arXiv:1509.03616*, 2015.

[7] C. Blair, "Problem complexity and method efficiency in optimization (nemirovsky and yudin)," *SIAM Review*, vol. 27, no. 2, p. 264, 1985.

[8] W. Krichene, A. M. Bayen, and P. L. Bartlett, "Accelerated mirror descent in continuous and discrete time," *Advances in Neural Information Processing Systems*, vol. 2015-Janua, pp. 2845–2853, 2015.

[9] D. Scieur, *Acceleration in optimization*. PhD thesis, PSL Research University, 2018.

[10] M. Miyashita, S. Yano, and T. Kondo, "Mirror descent search and its acceleration," *Robotics and Autonomous Systems*, vol. 106, pp. 107–116, 2018.

[11] R. K. Goodrich, "A riesz representation theorem," *Proceedings of the American Mathematical Society*, vol. 24, no. 3, pp. 629–636, 1970.

[12] S. Bubeck, "Convex optimization: Algorithms and complexity," *Foundations and Trends® in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.

[13] A. Beck and M. Teboulle, "Mirror descent and nonlinear projected subgradient methods for convex optimization," *Operations Research Letters*, vol. 31, no. 3, pp. 167–175, 2003.

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

[15] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate o(1/kˆ2)," in *Dokl. akad. nauk Sssr*, vol. 269, pp. 543–547, 1983.

[16] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[17] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Conference on Artificial Intelligence (AAAI)*, 2010.

[18] J. Bas-Serrano, S. Curi, A. Krause, and G. Neu, "Logistic q-learning," *arXiv preprint arXiv:2010.11151*, 2020.

[19] N. Dunford and J. T. Schwartz, *Linear operators part I: general theory*, vol. 243. Interscience publishers New York, 1958.

[20] J. J. Egozcue, J. L. Díaz–Barrero, and V. Pawlowsky–Glahn, "Hilbert space of probability density functions based on aitchison geometry," *Acta Mathematica Sinica, English Series*, vol. 22, no. 4, pp. 1175–1182, 2006.

[21] E. M. Stein and R. Shakarchi, *Functional analysis: introduction to further topics in analysis*, vol. 4. Princeton University Press, 2011.

[22] "Chapter 5 compact sets in banach spaces," in *Applications of Functional Analysis and Operator Theory* (V. Hutson and J. Pym, eds.), vol. 146 of *Mathematics in Science and Engineering*, pp. 138 – 147, Elsevier, 1980.

[23] B. A. Frigyik, S. Srivastava, and M. R. Gupta, "An introduction to functional derivatives," *Dept. Electr. Eng., Univ. Washington, Seattle, WA, Tech. Rep*, vol. 1, 2008.

[24] F. Nielsen and R. Nock, "Entropies and cross-entropies of exponential families," in *2010 IEEE International Conference on Image Processing*, pp. 3621–3624, 2010.

[25] S. T. Tokdar and R. E. Kass, "Importance sampling: a review," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[26] B. O'Donoghue and E. Candès, "Adaptive Restart for Accelerated Gradient Schemes," *Foundations of Computational Mathematics*, vol. 15, no. 3, pp. 715–732, 2015.

[27] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[28] C. W. Anderson, "Learning to control an inverted pendulum using neural networks," *IEEE Control Systems Magazine*, vol. 9, no. 3, pp. 31–37, 1989.

[29] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters, "Mushroomrl: Simplifying reinforcement learning research." https://github.com/MushroomRL/mushroom-rl, 2020.

[30] F. Sehnke, C. Osendorfer, T. Rückstieß Thomas, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010.

[31] H. Hachiya and M. Sugiyama, "Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 474–489, Springer, 2010.

[32] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[33] H. Van Hoof, G. Neumann, and J. Peters, "Non-parametric policy search with limited information loss," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2472–2517, 2017.

[34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. cite arxiv:1606.01540.

[35] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of machine learning research*, vol. 4, no. Dec, pp. 1107–1149, 2003.

[36] M. Ghavamzadeh and S. Mahadevan, "Hierarchical policy gradient algorithms," *Computer Science Department Faculty Publication Series*, p. 173, 2003.

[37] X. Jia, "Deep learning for actor-critic reinforcement learning," Master's thesis, TU Delft, 2015.

[38] S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli, "Policy gradient approaches for multi-objective sequential decision making," in *2014 International Joint Conference on Neural Networks (IJCNN)*, pp. 2323–2330, IEEE, 2014.

[39] G. Neu, A. Jonsson, and V. Gómez, "A unified view of entropy-regularized markov decision processes," *arXiv preprint arXiv:1705.07798*, 2017.

[40] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[42] M. Tomar, L. Shani, Y. Efroni, and M. Ghavamzadeh, "Mirror descent policy optimization," *arXiv preprint arXiv:2005.09814*, 2020.

[43] F. Otto, "The geometry of dissipative evolution equations: The porous medium equation," *Communications in Partial Differential Equations*, vol. 26, 04 2000.

[44] K. Ahn and S. Sra, "From nesterov's estimate sequence to riemannian acceleration," *arXiv preprint arXiv:2001.08876*, 2020.

[45] C. Liu, J. Zhuo, P. Cheng, R. Zhang, and J. Zhu, "Understanding and accelerating particle-based variational inference," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 4082–4092, PMLR, 09–15 Jun 2019.

[46] F. Santambrogio, "{Euclidean, metric, and Wasserstein} gradient flows: an overview," *Bulletin of Mathematical Sciences*, vol. 7, no. 1, pp. 87–154, 2017.

A. Hyperparameters

figure	η	c
5.1a	0.01	0.005
	0.01	0.05
	0.01	0.5
	0.01	5.0
5.1d	0.001	0.05
	0.01	0.05
	0.1	0.05
	1.0	0.05
5.3a	0.01	0.05
5.3b	0.001	0.05
5.3c	0.01	0.05
5.3d	1.0	0.05
5.3e	0.1	0.05
5.3f	0.1	0.05
5.3g	0.1	0.05
5.3h	10.0	0.05
5.4a	0.01	0.05
5.4b	0.01	0.05
5.4c	1.0	0.05

Table A.1.: DMD hyperparameters

figure	η	c	v	γ	restart
5.1b	0.01	0.005	3	1	-
	0.01	0.05	3	1	-
	0.01	0.5	3	1	-
	0.01	5.0	3	1	-
5.1c	0.1	0.005	3	1	-
	0.1	0.05	3	1	-
	0.1	0.5	3	1	-
	0.1	5.0	3	1	-
5.1e	0.001	0.05	3	1	-
	0.01	0.05	3	1	-
	0.1	0.05	3	1	-
	1.0	0.05	3	1	-
5.2a	0.1	0.05	3	1	-
	0.1	0.05	10	1	-
	0.1	0.05	30	1	-
	0.1	0.05	100	1	-
5.2b	5.0	0.05	3	1	-
	5.0	0.05	10	1	-
	5.0	0.05	30	1	-
	5.0	0.05	100	1	-
5.2c	5.0	0.05	3	1	-
	5.0	0.05	3	1	-
	5.0	0.05	3	1	-
	5.0	0.05	3	1	-
5.2d	5.0	0.05	3	1	-
	5.0	0.05	30	1	-
	5.0	0.05	30	1	-
	5.0	0.05	30	1	-
5.2e	0.001	0.05	3	1	-
	0.001	0.05	3	2	-
	0.001	0.05	3	4	-
	0.001	0.05	3	8	-
5.2f	0.1	0.05	3	1	-
	0.1	0.05	3	2	-
	0.1	0.05	3	4	-
	0.1	0.05	3	8	-
5.3a	0.1	0.05	30	1	50
5.3b	0.01	0.05	30	1	30
5.3c	0.1	0.05	10	1	30
5.3d	1.0	0.05	10	1	30
5.3e	0.1	0.05	3	1	30
5.3f	0.1	0.05	10	1	-
5.3g	0.1	0.05	10	1	30
5.3h	20.0	0.05	30	1	-
5.4c	1.0	0.05	10	1	30

Table A.2.: ADMD-1 hyperparameters

figure	η	c	v	γ	restart
5.3a	0.1	0.05	30	1	5
5.3b	0.01	0.05	10	1	5
5.3c	0.1	0.05	30	1	-
5.3d	1.0	0.05	10	1	30
5.3e	0.1	0.05	3	1	-
5.3f	0.1	0.05	10	1	50
5.3g	0.1	0.05	30	1	10
5.3h	20.0	0.05	30	1	50
5.4a	0.1	0.05	30	1	5
5.4b	0.01	0.05	10	1	5

Table A.3.: ADMD-2 hyperparameters

figure	η	c
5.4a	0.1	0.05
5.4b	0.1	0.05
5.4c	1.0	0.05

Table A.4.: PGPE hyperparameters

figure	ϵ	c
5.4a	1.0	0.05
5.4b	0.5	0.05
5.4c	4.0	0.05

Table A.5.: REPS hyperparameters