# Learning Convex Optimization Control Policies

Akshay Agrawal     Shane Barratt     Stephen Boyd

Bartolomeo Stellato*

December 19, 2019

**Abstract**

Many control policies used in various applications determine the input or action by solving a convex optimization problem that depends on the current state and some parameters. Common examples of such convex optimization control policies (COCPs) include the linear quadratic regulator (LQR), convex model predictive control (MPC), and convex control-Lyapunov or approximate dynamic programming (ADP) policies. These types of control policies are tuned by varying the parameters in the optimization problem, such as the LQR weights, to obtain good performance, judged by application-specific metrics. Tuning is often done by hand, or by simple methods such as a crude grid search. In this paper we propose a method to automate this process, by adjusting the parameters using an approximate gradient of the performance metric with respect to the parameters. Our method relies on recently developed methods that can efficiently evaluate the derivative of the solution of a convex optimization problem with respect to its parameters. We illustrate our method on several examples.

## 1 Introduction

### 1.1 Convex optimization control policies

We consider the control of a stochastic dynamical system with known dynamics, using a control policy that determines the input or action by solving a convex optimization problem. We call such policies *convex optimization control policies* (COCPs). Many practical policies have this form, including the first modern control policy, the linear quadratic regulator (LQR) [47]. In LQR, the convex optimization problem has quadratic objective and linear equality constraints, and so can be solved explicitly, yielding the familiar linear control policy. More modern examples, which rely on more complicated optimization problems such as quadratic programs (QPs), include convex model predictive control (MPC) [27] and convex approximate dynamic programming (ADP) [21]. These policies are used in many

---

*Authors listed in alphabetical order.

applications, including robotics [51], vehicle control [74], rocket landing [25], supply chain optimization [64], and finance [53, 37, 29].

Control policies in general, and COCPs in particular, are judged by application-specific metrics; these metrics are evaluated using simulation with historical or simulated values of the unknown quantities. In some but not all cases, the metrics have the traditional form of the average value of a given stage cost. We consider here more general metrics that can be functions of the whole state and input trajectories. An example of such a metric is the expected drawdown of a portfolio over some time period, *i.e.*, the expected value of the minimum future value of a portfolio.

In a few cases, the optimal policy for a traditional stochastic control problem has COCP form. A well-known example is LQR [47]. Another generic example is when the dynamics are affine and the stage cost is convex, in which case the Bellman value function is convex, and evaluating the optimal policy reduces to solving a convex optimization problem [48, §3.3.1]. While it is nice to know that in this case that the optimal policy has COCP form, we generally cannot express the value function in a form that allows us to evaluate the policy, so this observation is not useful in practice. In a far wider set of cases, a COCP policy is not optimal, but only a good, practical heuristic.

COCPs have some attractive properties compared to other parametrized control policies. When the convex problem to be solved is well chosen, the policy is at least reasonable for any choice of the parameter values over the allowed set. As a specific example, consider a linear control policy parametrized by the gain matrix, which indeed would seem to be the most natural parametrization of a linear policy. The set of gain matrices that lead to a stable closed-loop system (a very minimal performance requirement) can be very complex, even disconnected. In contrast, consider an LQR control policy parametrized by a state and control cost matrix (constrained to be positive definite). In this case any choice of policy yields a stable closed-loop system. It is far easier and safer to tune parameters when any feasible choice leads to at least a reasonable policy.

All control policies are tuned by choosing various parameters that appear in them. In the case of COCPs, the parameters are in the optimization problem that is solved to evaluate the policy. The tuning is usually done based on simulation with historical disturbances (called *back-testing*) or synthetic disturbances. It is often done by hand, or by a crude grid search. A familiar example of this is tuning the weights in an LQR controller to obtain good practical performance [8].

In this paper we present an automated method for tuning parameters in COCPs to achieve good values of a performance metric. Our method simulates the closed-loop system, *i.e.*, the system with the policy in the loop, and computes an approximate (stochastic) gradient of the expected performance with respect to the parameters. It uses this gradient to update the parameters via a projected stochastic gradient method. Central to our method is the fact that the solution map for convex optimization problems is often differentiable, and its derivative can be efficiently computed [2, 5]. This is combined with relatively new implementations of automatic differentiation, widely used in training neural networks [1, 62].

Our method is not guaranteed to find the best parameter values, since the performance

2

metric is not a convex function of the COCP parameter values, and we use a local search method. This is not a problem in practice, since in a typical use case, the COCP is initialized with reasonable parameters, and our method is used to tune these parameters to improve the performance (sometimes considerably).

## 1.2   Related work

**Dynamic programming.**   The Markov decision process (MDP) is a general stochastic control problem that can be solved in principle using dynamic programming (DP) [16, 17, 21]. The optimal policy is evaluated by solving an optimization problem, one that includes a current stage cost and the expected value of cost-to-go or value function at the next state. This optimization problem corresponds to a COCP when the system dynamics are linear or affine and the stage cost is convex [21]. Unfortunately, the value function can be found in a tractable form in only a few cases. A notable tractable case is when the cost is a convex extended quadratic and the dynamics are affine [14].

**Approximate dynamic programming.**   ADP [24, 63] refers to heuristic solution methods for stochastic control problems that replace the value function in DP with an approximation, or search over a parametric family of policies [22, §2.1].

In many ADP methods, an offline optimization problem is solved to approximate the value function. When there are a finite number of state and inputs, the approximation problem can be written as a linear program (LP) by relaxing the Bellman equation to an inequality [38]. When the dynamics are linear, the cost is quadratic, and the input is constrained to lie in a convex set, an approximate convex quadratic value function can be found by solving a particular semidefinite program (SDP) [78]. The quality of the approximation can also be improved by iterating the Bellman inequality [81, 73]. Because the approximate value function is convex quadratic and the dynamics are linear, the resulting policy is a COCP.

Other methods approximate the cost-to-go by iteratively adjusting the approximate value function to satisfy the Bellman equation. Examples of these methods include projected value iteration or fitted Q-iteration [43], temporal difference learning [75, 23], and approximate policy iteration [57]. Notable applications of COCPs here include the use of quadratic approximate cost-to-go functions for input-affine systems with convex cost, which can be approximately fit using projected value iteration [49], and modeling the state-action cost-to-go function as an input-convex neural network [7, §6.4]. Other approximation schemes fit nonconvex value functions, so the resulting policies are not necessarily COCPs. Notably, when the parametrization involves a featurization computed by a deep neural network, the ADP method is an instance of deep reinforcement learning.

Other ADP methods parametrize the policy and tune the parameters directly to improve performance; this is often referred to as policy search or policy approximation [22, §5.7]. The most common method is gradient or stochastic gradient search [63, §7.2], which is the method we employ in this paper, with a parametrized COCP as the policy. Historically, the most widely used of these policy approximation methods is the Proportional-Integral-Derivative (PID) controller [55], which indeed can be tuned using gradient methods [10].

**Reinforcement learning.** Reinforcement learning (RL) [76] and adaptive control [11] are essentially equivalent to ADP [22, §1.4], but with different notation and different emphasis. RL pays special attention to problems in which one does not possess a mathematical model of the dynamics or the expected cost, but has access to a computational simulator for both. Our method cannot be used directly in this setting, since we assume that we have mathematical descriptions of the dynamics and cost. However, our method might be used after learning a suitable model of the dynamics and cost. Alternatively, COCPs could be used as part of the policy in modern policy gradient or actor-critic algorithms [83, 52, 70].

**Learning optimization-based policies.** Other work has considered tuning optimization-based control policies. For example, there is prior work on learning for MPC, including nonconvex MPC controllers [6], cost function shaping [77], differentiable path integral control [60], and system identification of terminal constraint sets and costs [67]. As far as we are aware, our work is the first to consider the specific class of parametrized convex programs.

**Real-time optimization.** COCPs might be considered computationally expensive control policies compared to conventional analytical control policies such as the linear control policy prescribed by LQR. However, this is not the case in practice, thanks to fast embedded solvers [40, 72, 79] and code generation tools that emit solvers specialized to parametric problems [54, 33, 12]. For example, the aerospace and space transportation company SpaceX uses the QP code generation tool CVXGEN [54] to land its rockets [25]. COCPs based on MPC, which have many more variables and constraints than those based on ADP, can also be evaluated very efficiently [18, 80], even at MHz rates [46].

## 1.3 Outline

In §2, we introduce the controller tuning problem that we wish to solve. In §3, we describe some common forms of COCPs. In §4, we propose a heuristic for the controller tuning problem. In §5, we apply our heuristic for tuning COCPs to examples in portfolio optimization, vehicle control, and supply-chain management. We conclude in §6 by discussing extensions and variations.

## 2 Controller tuning problem

**System dynamics.** We consider a dynamical system with dynamics given by

$$x_{t+1} = f(x_t, u_t, w_t), \quad t = 0, 1, \dots. \tag{1}$$

At time period $t$, $x_t \in \mathbf{R}^n$ is the state, $u_t \in \mathbf{R}^m$ is the input or action, $w_t \in \mathcal{W}$ is the disturbance, and $f : \mathbf{R}^n \times \mathbf{R}^m \times \mathcal{W} \to \mathbf{R}^n$ is the state transition function. The initial state $x_0$ and the disturbances $w_t$ are random variables. In the traditional stochastic control problem, it is assumed that $x_0, w_0, w_1, \dots$ are independent, with $w_0, w_1, \dots$ identically distributed. We do not make this assumption.

The inputs are given by a state feedback control policy,

$$u_t = \phi(x_t), \tag{2}$$

where $\phi : \mathbf{R}^n \to \mathbf{R}^m$ is the policy. In particular, we assume the state $x_t$ at time period $t$ is fully observable when the input $u_t$ is chosen. It will be clear later that this assumption is not really needed, since our method can be applied to an estimated state feedback policy, either with a fixed state estimator, or with a state estimator that is itself a parametrized convex problem (see §6).

With the dynamics (1) and policy (2), the state and input trajectories $x_0, x_1, \ldots$ and $u_0, u_1, \ldots$ form a stochastic process.

**Convex optimization control policies.** We specifically consider COCPs, which have the form

$$
\begin{aligned}
\phi(x) \quad = \quad & \underset{u}{\text{argmin}} & & f_0(x, u; \theta) \\
& \text{subject to} & & f_i(x, u; \theta) \le 0, \quad i = 1, \ldots, k, \\
& & & g_i(x, u; \theta) = 0, \quad i = 1, \ldots, \ell,
\end{aligned}
\tag{3}
$$

where $f_i$ are convex in $u$ and $g_i$ are affine in $u$. To evaluate a COCP we must solve a convex optimization problem, which we assume has a unique solution. The convex optimization problem (3) is given by a *parametrized problem description* [30, §4.1.4], in which the vector $\theta \in \Theta \subseteq \mathbf{R}^p$ is the parameter ($\Theta$ is the set of allowable parameter values). The value of the parameter $\theta$ (and $x$) specifies a particular problem instance, and it can be adjusted to tune the control policy. The problem we address in this paper is the choice of the parameter $\theta$.

**Performance metric.** We judge the performance of a control policy, or choice of control policy parameter $\theta$, by the average value of a cost over trajectories of length $T$. Here the horizon $T$ is chosen large enough so that the average over $T$ time steps is close enough to the long term average. We denote the trajectories over $t = 0, \ldots, T$ as

$$
\begin{aligned}
X &= (x_0, x_1, \ldots, x_T) \in \mathbf{R}^N, \\
U &= (u_0, u_1, \ldots, u_T) \in \mathbf{R}^M, \\
W &= (w_0, w_1, \ldots, w_T) \in \mathcal{W}^{T+1},
\end{aligned}
$$

where $N = (T+1)n$ and $M = (T+1)m$. These state, input, and disturbance trajectories are random variables, with distributions that depend on the parameter $\theta$.

The cost is provided by a function $\psi : \mathbf{R}^N \times \mathbf{R}^M \times \mathcal{W}^{T+1} \to \mathbf{R} \cup \{+\infty\}$. Infinite values of $\psi$ can be interpreted as encoding constraints on the trajectories. A policy is judged by the expected value of this cost,

$$J(\theta) = \mathbf{E}\,\psi(X, U, W).$$

We emphasize that $J$ depends on the control policy parameter $\theta$, since $x_1, \ldots, x_T$ and $u_0, \ldots, u_T$ depend on $\theta$.

We mention that the traditional cost function is separable, with the form

$$\psi(X, U, W) = \frac{1}{T+1} \sum_{t=0}^{T} g(x_t, u_t, w_t), \tag{4}$$

where $g : \mathbf{R}^n \times \mathbf{R}^m \times \mathcal{W} \to \mathbf{R} \cup \{\infty\}$ is a stage cost function. However, we do not require a cost function that is separable across time.

**Evaluating $J(\theta)$.** We generally cannot evaluate $J(\theta)$ exactly. Instead, assuming that we can sample the initial state and the disturbances, we can compute a Monte Carlo approximation of it. In the simplest version, we generate $K$ independent trajectories

$$(X^1, U^1, W^1), \ldots, (X^K, U^K, W^K),$$

and form the approximation

$$\hat{J}(\theta) = \frac{1}{K} \sum_{i=1}^{K} \psi(X^i, U^i, W^i).$$

This computation requires carrying out $K$ simulations over $T$ time steps, which involves solving $K(T+1)$ convex optimization problems to evaluate $u_t^i$, $t = 0, \ldots, T$, $i = 1, \ldots K$.

Evidently, $\hat{J}(\theta)$ is an unbiased approximation of $J(\theta)$, meaning

$$\mathbf{E}\, \hat{J}(\theta) = J(\theta).$$

The quality of this approximation increases as $K$ increases, since

$$\mathbf{var}\, \hat{J}(\theta) = \frac{\mathbf{var}\, \psi(X, U, W)}{K},$$

where **var** denotes the variance; *i.e.*, the variance goes to $0$ as $K$ gets large. Of course more sophisticated methods can be used to approximately evaluate $J(\theta)$, *e.g.*, importance sampling (see [34]).

**Controller tuning problem.** The controller tuning problem has the form

$$\begin{array}{ll} \text{minimize} & J(\theta) \\ \text{subject to} & \theta \in \Theta, \end{array} \tag{5}$$

with variable $\theta$. This is the problem we seek to solve in this paper.

# 3   Examples of COCPs

In this section we describe some common COCPs.

**Optimal (dynamic programming) policy.** In the traditional stochastic control setting, the cost function is the average of stage costs computed by a function $g$, as in (4), and $x_0, w_0, w_1, \ldots$ are independent. Under some technical conditions, the optimal policy for $T \to \infty$, *i.e.*, the policy that minimizes $J$ over all possible state feedback policies, and not just those of COCP form, has the form

$$\phi(x) = \operatorname*{argmin}_{u} \mathbf{E}\left(g(x, u, w) + V(f(x, u, w))\right), \tag{6}$$

where $V : \mathbf{R}^n \to \mathbf{R}$ is the optimal cost-to-go or Bellman value function. This form of the optimal policy is sometimes called the dynamic programming (DP) form. When $f$ is affine in $x$ and $u$, and $g$ is convex in $x$ and $u$, it can be shown that the value function $V$ is convex [48, §3.3.1], so the expression to be minimized above is convex in $u$, and the optimal policy has COCP form (with no parameter $\theta$).

Unfortunately the optimal value function $V$ can be expressed in tractable form in only a few special cases. One well-known one is LQR [47], which has dynamics and stage cost

$$f(x, u, w) = Ax + Bu + w, \qquad g(x, u, w) = x^T Q x + u^T R u, \tag{7}$$

with $A \in \mathbf{R}^{n \times n}$, $B \in \mathbf{R}^{n \times m}$, $Q \in \mathbf{S}_+^n$ (the set of $n \times n$ symmetric positive semidefinite matrices), $R \in \mathbf{S}_{++}^m$ (the set of symmetric positive definite matrices), and $w \sim \mathcal{N}(0, \Sigma)$. In this special case we can compute the value function, which is a convex quadratic $V(x) = x^T P x$, and the optimal policy has the form

$$\phi(x) = \operatorname*{argmin}_{u} \left(u^T R u + (Ax + Bu)^T P(Ax + Bu)\right) = Kx,$$

with

$$K = -(R + B^T P B)^{-1} B^T P A.$$

Note that we can consider the policy above as a COCP, if we consider $P$ as our parameter $\theta$ (constrained to be positive semidefinite). Another option is to take $P = \theta^T \theta$, where $\theta \in \mathbf{R}^{n \times n}$, so the COCP has objective

$$f_0(x, u; \theta) = u^T R u + \|\theta(Ax + Bu)\|_2^2.$$

**Approximate dynamic programming policy.** An ADP [63] or control-Lyapunov [36] policy has the form

$$\phi(x) = \operatorname*{argmin}_{u} \mathbf{E}(g(x, u, w) + \hat{V}(f(x, u, w))), \tag{8}$$

where $\hat{V}$ is an approximation of the value function for which the minimization over $u$ above is tractable. When $g$ is convex in $u$, $f$ is affine in $u$, and $\hat{V}$ is convex, the minimization above is a convex optimization problem [30]. With a suitable parametrization of $\hat{V}$, this policy has COCP form [48].

**Model predictive control policy.** Suppose the cost function has the form (4), with stage cost $g$. In an MPC policy, the input is determined by solving an approximation to the control problem over a short horizon, where the unknown disturbances are replaced by predictions [65], and applying only the first input. A terminal cost function $g_H$ is often included in the optimization.

An MPC policy has the form

$$
\begin{aligned}
\phi(x) \quad = \quad \underset{u_0}{\text{argmin}} \quad & \sum_{t=0}^{H-1} g(x_t, u_t, \hat{w}_t) + g_H(x_H) \\
\text{subject to} \quad & x_{t+1} = f(x_t, u_t, \hat{w}_t), \quad t = 0, \ldots, H-1, \\
& x_0 = x,
\end{aligned}
$$

where $H$ is the planning horizon and $\hat{w}_0, \ldots, \hat{w}_{H-1}$ are the predicted disturbances. This optimization problem has variables $u_0, \ldots, u_{H-1}$ and $x_0, \ldots, x_H$; however, the argmin is over $u_0$ since in MPC we only apply the first input.

When $f$ is affine in $(x, u)$, $g$ is convex in $(x, u)$, and the terminal cost function $g_H$ is convex, the minimization above is a convex optimization problem. With a suitable parametrization of the terminal cost function $g_H$, the MPC policy has COCP form. When $f$ is not affine or $g$ is not convex, they can be replaced with parametrized convex approximations. The function that predicts the disturbances can also be parametrized (see §6).

# 4 Solution method

Solving the controller tuning problem (5) exactly is in general hard, especially when the number of parameters $p$ is large, so we will solve it approximately. Historically, many practitioners have used derivative-free methods to tune the parameters in control policies. Some of these methods include CMA-ES [45] and other evolutionary strategies [69], Bayesian optimization [56], grid search, and random search [9, 71, 20]. Many more methods are catalogued in [35]. These methods can certainly yield improvements over an initialization; however, they often converge very slowly.

**A gradient-based method.** It is well-known that first-order optimization methods, which make use of derivatives, can outperform derivative-free methods. In this paper, we apply the projected stochastic (sub)gradient method [66] to approximately solve (5). That is, starting with initial parameters $\theta^0$, at iteration $k$, we simulate the system and compute $\hat{J}(\theta^k)$. We then compute an unbiased stochastic gradient of $J$, $g^k = \nabla \hat{J}(\theta^k)$, by the chain rule or backpropagation through time (BPTT) [68, 82], and update the parameters according to the rule $\theta^{k+1} = \Pi_\Theta(\theta^k - \alpha^k g^k)$, where $\Pi_\Theta(\theta)$ denotes the Euclidean projection of $\theta$ onto $\Theta$ and $\alpha^k > 0$ is a step size. Of course more sophisticated methods can be used to update the parameters, for example, those that employ momentum, variance reduction, or second-order information (see [28] and the references therein for some of these methods).

**Computing $g^k$.** The computation of $g^k$ requires differentiating through the dynamics $f$, the cost $\psi$, and, notably, the solution map $\phi$ of a convex optimization problem. Methods for differentiating through special subclasses of convex optimization have existed for many decades; for example, literature on differentiating through QPs dates back to at least the 1960s [26]. Similarly, it is well known that if the objective function and constraint functions of a convex optimization problem are all smooth, and some regularity conditions are satisfied, then its derivative can be computed by differentiating through the KKT optimality conditions [50, 13]. Until very recently, however, it was not generically possible to differentiate through a convex optimization problem with nondifferentiable objective or constraints; recent work [31, 3, 2, 5] has shown how to efficiently and easily compute this derivative.

**Non-differentiability.** Until this point, we have assumed the differentiability of all of the functions involved ($f$, $\psi$, and $\phi$). In real applications, these functions very well may not be differentiable everywhere. So long as the functions are differentiable almost everywhere, however, it is reasonable to speak of applying a projected stochastic gradient method to (5). At non-differentiable points, we compute a heuristic quantity. For example, at some non-differentiable points of $\phi$, a certain matrix fails to be invertible, and we compute a least-squares approximation of the derivative instead, as in [3]. In this sense, we overload the notation $\nabla f(x)$ to denote a gradient when $f$ is differentiable at $x$, or some heuristic quantity (a "gradient") when $f$ is not differentiable at $x$. In practice, as our examples in §5 demonstrate, we find that this method works well. Indeed, most neural networks that are trained today are not differentiable (*e.g.*, the rectified linear unit or positive part is a non-differentiable activation function that is widely used) or even subdifferentiable (since neural networks are usually nonconvex), but it is nonetheless possible to train them, successfully, using stochastic "gradient" descent [42].

# 5 Examples

In this section, we present examples that illustrate our method. Our control policies were implemented using CVXPY [39, 4], and we used cvxpylayers [2] and PyTorch [62] to differentiate through them; cvxpylayers uses the open-source package SCS [58, 59] to solve convex optimization problems. For each example, we give the dynamics, the cost, the COCP under consideration, and the result of applying our method to a numerical instance.

In the numerical instances, we pick the number of simulations $K$ so that the variance of $\hat{J}(\theta)$ is sufficiently small, and we tune the step-size schedule $\alpha^k$ for each problem. BPTT is susceptible to exploding and vanishing gradients [19], which can make learning difficult. This issue can be mitigated by gradient clipping and regularization [61], which we do in some of our experiments.
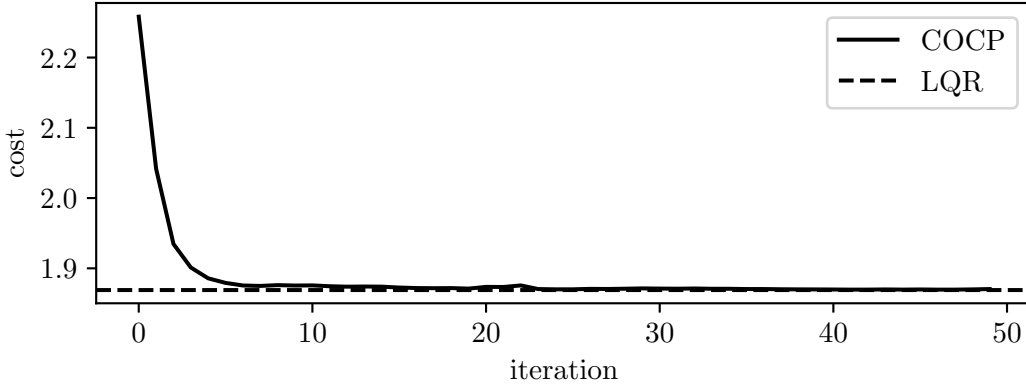
Figure 1: Tuning an LQR policy.

## 5.1 LQR

We first apply our method to the classical LQR problem, with dynamics and cost

$$f(x, u, w) = Ax + Bu + w, \quad \psi(X, U, W) = \frac{1}{T+1} \sum_{t=0}^{T} x_t^T Q x_t + u_t^T R u_t,$$

where $A \in \mathbf{R}^{n \times n}$, $B \in \mathbf{R}^{n \times m}$, $Q \in \mathbf{S}_+^n$, $R \in \mathbf{S}_{++}^m$, and $w \sim \mathcal{N}(0, \Sigma)$.

**Policy.**  We use the COCP

$$\phi(x) = \operatorname*{argmin}_{u} \left( u^T R u + \|\theta(Ax + Bu)\|_2^2 \right), \tag{9}$$

with parameter $\theta \in \mathbf{R}^{n \times n}$. This policy is linear, of the form $\phi(x) = Gx$, with

$$G = -(R + B^T \theta^T \theta B)^{-1} B^T \theta^T \theta A.$$

This COCP is clearly over-parametrized; for example, for any orthogonal matrix $U$, $U\theta$ gives the identical policy as $\theta$. If the matrix $\theta^T \theta$ satisfies a particular algebraic Riccati equation involving $A$, $B$, $Q$, and $R$, then (9) is optimal (over all control policies) for the case $T \to \infty$.

**Numerical example.**  We consider a numerical example with $n = 4$ states, $m = 2$ inputs, and $T = 100$. The entries of $A$ and $B$ were sampled from the standard normal distribution, and we scaled $A$ such that its spectral radius was one. The cost matrices are $Q = I$ and $R = I$, and the noise covariance is $W = (0.25)I$. We initialize $\theta$ with the identity. We trained our policy (9) for 50 iterations, using $K = 6$ simulations per step, starting with a step size of 0.5 that was decreased to 0.1 after 25 iterations. Figure 1 plots the average cost of the COCP during learning versus the average cost of the optimal LQR policy (in the case $T \to \infty$). Our method appears to converge to near the optimal cost in just 10 iterations.
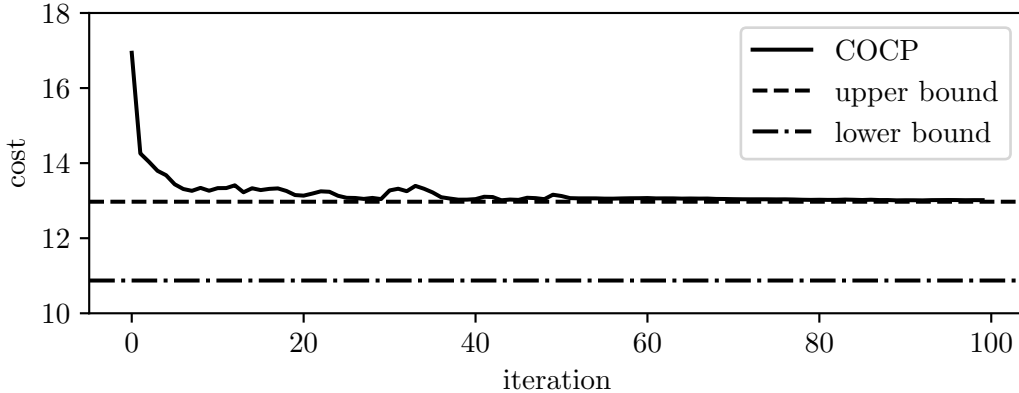
10

Figure 2: Tuning a box-constrained LQR policy.

## 5.2 Box-constrained LQR

A box-constrained LQR problem has the same dynamics and cost as LQR, with an additional constraint $\|u_t\|_\infty \le u_{\max}$:

$$\psi(X, U, W) = \frac{1}{T+1} \sum_{t=0}^{T} g(x_t, u_t, w_t), \qquad g(x_t, u_t, w_t) = \begin{cases} x_t^T Q x_t + u_t^T R u_t, & \|u_t\|_\infty \le u_{\max} \\ +\infty & \text{otherwise.} \end{cases}$$

Unlike the LQR problem, in general, there is no known exact solution to the box-constrained problem, analytical or otherwise. Sophisticated methods can be used, however, to compute a lower bound on the true optimal cost [78].

**Policy.** Our COCP is an ADP policy (8) with a quadratic value function:

$$\phi(x) = \underset{u}{\operatorname{argmin}} \quad u^T R u + \|\theta(Ax + Bu)\|_2^2$$
$$\text{subject to} \quad \|u\|_\infty \le u_{\max},$$

(10)

with parameter $\theta \in \mathbf{R}^{n \times n}$. The lower bound found in [78] yields a policy that has this same form, for a particular value of $\theta$.

**Numerical example.** We use $n = 8$ states, $m = 2$ inputs, $T = 100$, $u_{\max} = 0.1$, and data generated as in the LQR example above. The lower bounding technique from [78] yields a lower bound on optimal cost of around 11. It also suggests a particular value of $\theta$, which gives average cost around 13, an upper bound on the optimal cost that we suspect is the true optimal average cost. We initialize our COCP with $\theta = P^{1/2}$, where $P$ comes from the cost-to-go function for the unconstrained (LQR) problem. Figure 2 plots the expected cost of our COCP, and the expected cost of the upper and lower bounds suggested by [78]. Our method converges to roughly the same cost as the upper bound.

11

## 5.3 Tuning a Markowitz policy to maximize utility

In 1952, Markowitz introduced an optimization-based method for the allocation of financial portfolios [53], which trades off risk (measured as return variance), and (expected) return. While the original formulation involved only a quadratic objective and linear equality constraints (very much like LQR), with the addition of other constraints and terms, Markowitz's method becomes a sophisticated COCP [44, 29]. The parameters are the data that appear in the convex problem solved to determine the trades to execute in each time period.

In this example, we learn the parameters in a Markowitz policy to maximize a utility on the realized returns. We will use notation from [29], representing the state by $w_t$, the control by $z_t$, and the disturbance by $r_t$.

The portfolio under consideration has $n$ assets. The dollar value of the portfolio in period $t$ is denoted by $v_t$, which we assume to be positive. Our holdings in period $t$, normalized by the portfolio value, are denoted by $w_t \in \mathbf{R}^n$; the normalization ensures that $\mathbf{1}^T w_t = 1$. The number $v_t(w_t)_i$ is the dollar value of our position in asset $i$ ($(w_t)_i < 0$ corresponds to a short position). In each period, we re-allocate our holdings by executing trades $z_t \in \mathbf{R}^n$, which are also normalized by $v_t$. Selling or shorting asset $i$ corresponds to $(z_t)_i < 0$, and purchasing it corresponds to $(z_t)_i > 0$. Trades incur transaction costs $\kappa^T |z_t|$, where $\kappa \in \mathbf{R}^n_{++}$ (the set of positive $n$-vectors) is the vector of transaction cost rates and the absolute value is applied elementwise. Shorting also incurs a cost, which we express by $\nu^T (w_t + z_t)_-$, where $\nu \in \mathbf{R}^n_{++}$ is the vector of stock loan rates and $(\cdot)_-$ is the negative part. We impose the condition that trades are self-financing, i.e., we must withdraw enough cash to pay the transaction and shorting costs incurred by our trades. This can be expressed as $\mathbf{1}^T z_t + \kappa^T |z_t| + \nu^T (w_t + z_t)_- \leq 0$.

The holdings evolve according to the dynamics

$$w_{t+1} = r_t \circ (w_t + z_t)/r_t^T(w_t + z_t)$$

where $r_t \in \mathbf{R}^n_+$ are the total returns (which are IID) and $\circ$ is the elementwise product. The denominator in this expression is the return realized by executing the trade $z_t$.

Our goal is to minimize the average negative utility of the realized returns, as measured by a utility function $U : \mathbf{R} \to \mathbf{R}$. Letting $W$, $Z$ and $R$ denote the state, input, and disturbance trajectories, the cost function is

$$\psi(W, Z, R) = \frac{1}{T+1} \sum_{t=0}^{T} -U(r_t^T(w_t + z_t)) + I(z_t),$$

where $I : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ enforces the self-financing condition: $I(z_t)$ is 0 when $\mathbf{1}^T z_t + \kappa^T |z_t| + \nu^T (w_t + z_t)_- \leq 0$ and $+\infty$ otherwise.

**Policy.** We consider policies that compute $z_t$ as

$$
\begin{aligned}
\phi(w_t) \quad = \quad &\underset{z}{\operatorname{argmax}} \quad && \mu^T w^+ - \gamma \|S w^+\|_2^2 \\
&\text{subject to} \quad && w^+ = w_t + z \\
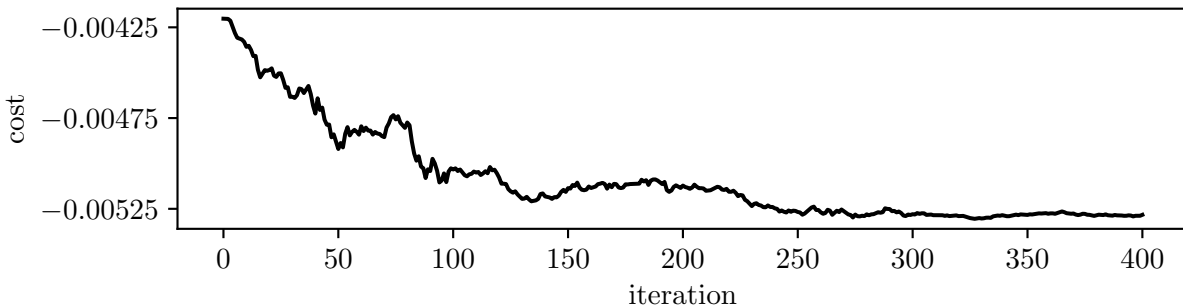& && \mathbf{1}^T z + \kappa^T |z| + \nu^T (w^+)_- \leq 0,
\end{aligned}
$$

Figure 3: Tuning a Markowitz policy.

with variables $w^+$ and $z$ and parameters $\theta = (\mu, \gamma, S)$, where $\mu \in \mathbf{R}^n$, $\gamma \in \mathbf{R}_+$, and $S \in \mathbf{R}^{n \times n}$. In a Markowitz formulation, $\mu$ is set to the empirical mean $\mu^{\text{mark}}$ of the returns, and $S$ is set to the square root of the return covariance $\Sigma^{\text{mark}}$. With these values for the parameters, the linear term in the objective represents the expected return of the post-trade portfolio $w^+$, and the quadratic term represents the risk. A trade-off between the risk and return is determined by the choice of the risk-aversion parameter $\gamma$. We mention that it is conventional to parametrize a Markowitz policy with a matrix $\Sigma \in \mathbf{S}^n_+$, rewriting the quadratic term as $w^{+T} \Sigma w^+$; as in the LQR example, our policy is over-parametrized.

In addition to the self-financing condition, there are many other constraints one may want to impose on the trade vector and the post-trade portfolio, including constraints on the portfolio leverage and turnover, many of which are convex. For various examples of such constraints, see [29, §4.4, §4.5].

**Numerical example.** We use $n = 12$ ETFs as the universe of assets,

AGG, VTI, VNQ, XLF, XLV, XLY, XLP, XLU, XLI, XLE, IBB, and ITA.

For the transaction rates and stock loan rates, we use $\kappa = \nu = (0.001)\mathbf{1}$, or 0.1 percent. We assume the investor is somewhat risk-averse, with utility function

$$U(r) = \min(2(r-1), r-1).$$

The policy is initialized with $\mu = \mu^{\text{mark}}$, $S = (\Sigma^{\text{mark}})^{1/2}$, and $\gamma = 15$. Each simulation starts with the portfolio obtained by solving

$$\begin{array}{ll} \text{maximize} & \mu^T w - \gamma \|Sw\|_2^2 - \nu^T(w)_- \\ \text{subject to} & \mathbf{1}^T w = 1, \end{array}$$

with variable $w \in \mathbf{R}^n$. The portfolio evolves according to returns sampled from a log-normal distribution. This distribution was fit to monthly returns (including dividends) from Dec. 2006 through Dec. 2018, retrieved from the Center for Research in Security Prices [32].
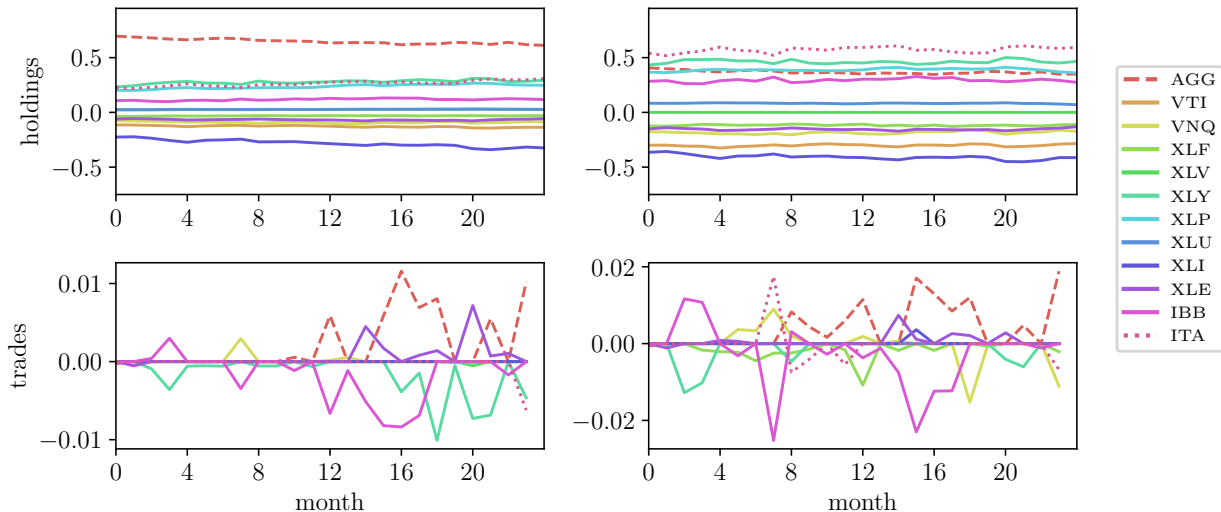
13

Figure 4: Simulated holdings (top row) and trades (bottom row) for untuned (left column) and tuned (right column) policies.

We train the policy using stochastic gradient descent over 400 iterations, with a horizon of $T = 24$ months and $K = 10$ simulations to evaluate $\hat{J}(\theta)$. (The step size is initialized to $10^{-3}$, halved every 100 iterations.) Figure 3 plots the per-iteration cost on a held-out random seed while training. The policy's performance improved by approximately 32 percent, decreasing from an initial cost of $-0.004$ to $-0.0053$.

Figure 4 plots simulated holdings and trades before and after tuning. Throughout the simulations, both the untuned and tuned policies regulated or re-balanced their holdings to track the initial portfolio, making small trades when their portfolios began to drift. The parameter $\mu$ was adjusted from its initial value,

$$(1.003, 1.006, 1.006, 1.002, 1.009, 1.009, 1.007, 1.006, 1.007, 1.004, 1.011, 1.011),$$

to

$$(0.999, 1.006, 1.005, 1.000, 1.001, 1.009, 1.008, 1.007, 1.009, 1.002, 1.014, 1.013).$$

In particular, the entry corresponding to AGG, a bond ETF, decreased from 1.003 to 0.999, and the entry for ITA, an aerospace and defense ETF, increased from 1.011 to 1.013; this observation is consistent with the plotted simulated holdings.

Tuning had essentially no effect on $\gamma$, which decreased from 15 to 14.99. The difference between $\Sigma^{\text{mark}}$ and $S^T S$, however, was significant: the median absolute percentage deviation between the entries of these two quantities was 2.6 percent.

## 5.4  Tuning a vehicle controller to track curved paths

We consider a vehicle moving relative to a smooth path, with state and input

$$x_t = (e_t, \Delta\psi_t, v_t, v_t^{\text{des}}, \kappa_t), \quad u_t = (a_t, z_t).$$

14

Here, at time period $t$, $e_t$ is the lateral path deviation (m), $\Delta\psi_t$ is the heading deviation from the path (rad), $v_t$ is the velocity (m/s), $v_t^{\text{des}}$ is the desired velocity (m/s), $\kappa_t$ is the current curvature (*i.e.*, inverse radius) of the path (1/m), $a_t$ is the acceleration (m/s$^2$), and $z_t := \tan(\delta_t) - L\kappa_t$, where $\delta_t$ is the wheel angle (rad) and $L$ is the vehicle's wheelbase (m).

**Dynamics.** We consider kinematic bicycle model dynamics in path coordinates [41], discretized at $h = 0.2\,\text{s}$, with random processes for $v_t^{\text{des}}$ and $\kappa_t$, of the form

$$
\begin{aligned}
e_{t+1} &= e_t + hv_t \sin(\Delta\psi_t) + w_1, \quad w_1 \sim \mathcal{N}(0, .01), \\
\Delta\psi_{t+1} &= \Delta\psi_t + hv_t\left(\kappa_t + \frac{z_t}{L} - \frac{\kappa_t}{1 - e_t\kappa_t}\cos(\Delta\psi_t)\right) + w_2, \quad w_2 \sim \mathcal{N}(0, .0001), \\
v_{t+1} &= v_t + ha_t + w_3, \quad w_3 \sim \mathcal{N}(0, .01), \\
v_{t+1}^{\text{des}} &= v_t^{\text{des}}w_4 + w_5(1 - w_4), \quad w_4 \sim \text{Bernoulli}(0.98), \quad w_5 \sim \mathcal{U}(3, 6), \\
\kappa_{t+1} &= \kappa_t w_6 + w_7(1 - w_6), \quad w_6 \sim \text{Bernoulli}(0.95), \quad w_7 \sim \mathcal{N}(0, .01).
\end{aligned}
$$

The disturbances $w_1, w_2, w_3$ represent uncertainty in our model, and $w_4, \ldots, w_7$ form the random process for the desired speed and path.

**Cost.** Our goal is to travel the desired speed ($v_t \approx v_t^{\text{des}}$), while tracking the path ($e_t \approx 0$, $\Delta\psi \approx 0$) and expending minimal control effort ($a_t \approx 0$, $z_t \approx 0$). We consider the cost

$$
\psi(X, U, W) = \frac{1}{T+1}\sum_{t=0}^{T}(v_t - v_t^{\text{des}})^2 + \lambda_1 e_t^2 + \lambda_2 \Delta\psi_t^2 + \lambda_3 |a_t| + \lambda_4 z_t^2 + I(a_t, z_t, \kappa_t),
$$

for positive $\lambda_1, \ldots, \lambda_4$ (with proper units), where

$$
I(a, z, \kappa) = \begin{cases} 0 & |a| \le a_{\max}, |z + L\kappa| \le \tan(\delta_{\max}), \\ +\infty & \text{otherwise,} \end{cases}
$$

for given maximum acceleration magnitude $a_{\max}$ (m/s$^2$) and maximum wheel angle magnitude $\delta_{\max}$ (rad).

**Policy.** We consider a COCP that computes $(a_t, z_t)$ as

$$
\begin{aligned}
\phi(x_t) \;=\; &\underset{a,z}{\operatorname{argmin}} \quad \lambda_3|a| + \lambda_4 z^2 + \|Sy\|_2^2 + q^T y \\
&\text{subject to} \quad y = \begin{bmatrix} e_t + hv_t \sin(\Delta\psi_t) \\ \Delta\psi_t + hv_t\left(\kappa_t + \frac{z}{L} - \frac{\kappa_t}{1 - e_t\kappa_t}\cos(\Delta\psi_t)\right) \\ v_t + ha - (0.98)v_t^{\text{des}} - (0.02)4.5 \\ y_1 + hv_t\sin(y_2 - hv_t\frac{z}{L}) + \frac{h^2 v_t^2}{L}z, \end{bmatrix} \\
&\qquad\qquad |a| \le a_{\max} \\
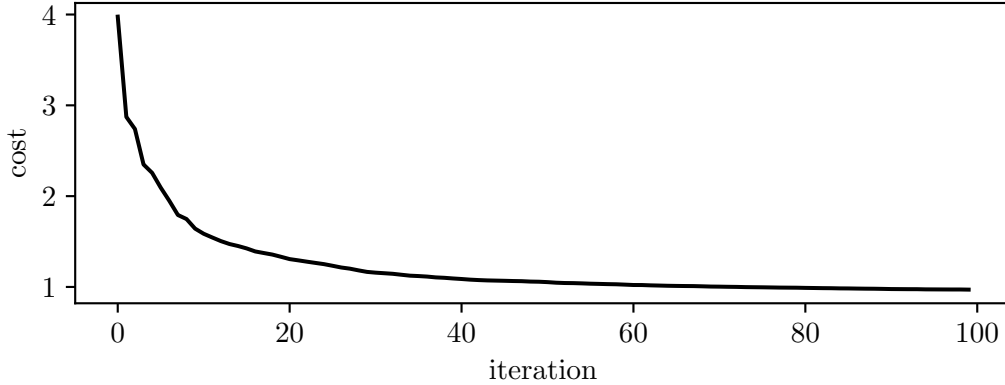&\qquad\qquad |z + L\kappa_t| \le \tan(\delta_{\max}),
\end{aligned}
$$

15

Figure 5: Tuning a vehicle controller.

with parameters $\theta = (S, q)$, where $S \in \mathbf{R}^{4 \times 4}$ and $q \in \mathbf{R}^4$. The additional variable $y \in \mathbf{R}^4$ represents relevant portions of the next state, since $y_1 = e_{t+1}$, $y_2 = \Delta\psi_{t+1}$, $y_3 = v_{t+1} - \mathbf{E}[v_{t+1}^{\mathrm{des}}]$, and $y_4 \approx e_{t+2}$ (since it assumes $a_t = 0$). Therefore, this COCP is an ADP policy and the term $\|Sy\|_2^2 + q^T y$ can be interpreted as the approximate value function.

**Numerical example.** We consider a numerical example with

$$L = 2.8\,\mathrm{m}, \quad \lambda_1 = \lambda_2 = 1, \quad \lambda_3 = \lambda_4 = 10, \quad a_{\max} = 2\,\mathrm{m/s^2}, \quad \delta_{\max} = 0.6\,\mathrm{rad}, \quad T = 100.$$

We use the initial state $x_0 = (.5, .1, 3, 4.5, 0)$. We run the stochastic gradient method for 100 iterations using $K = 6$ simulations and a step size of 0.1. We initialize the parameters with $S = I$ and $q = 0$. Over the course of learning, the cost decreased from 3.978 to 0.971. Figure 5 plots per-iteration cost on a held-out random seed while training. Figure 6 plots untuned and tuned sample paths on a single held-out instance. The resulting parameters are

$$S^T S = \begin{bmatrix} 1.12 & 1.17 & -0.75 & 0.85 \\ 1.17 & 3.82 & 0.46 & 3.13 \\ -0.75 & 0.46 & 13.07 & -0.29 \\ 0.85 & 3.13 & -0.29 & 3.96 \end{bmatrix}, \quad q = (-0, -0.04, -0.25, -0.04).$$

## 5.5 Tuning a supply chain policy to maximize profit

Supply chain management considers how to ship goods across a network of warehouses to maximize profit. In this example, we consider a single-good supply chain with $n$ nodes representing interconnected warehouses linked to suppliers and consumers by $m$ directed links over which goods can flow. There are $k$ links connecting suppliers to warehouses and $c$ links connecting warehouses to consumers. The remaining $m - k - c$ links are internode links.
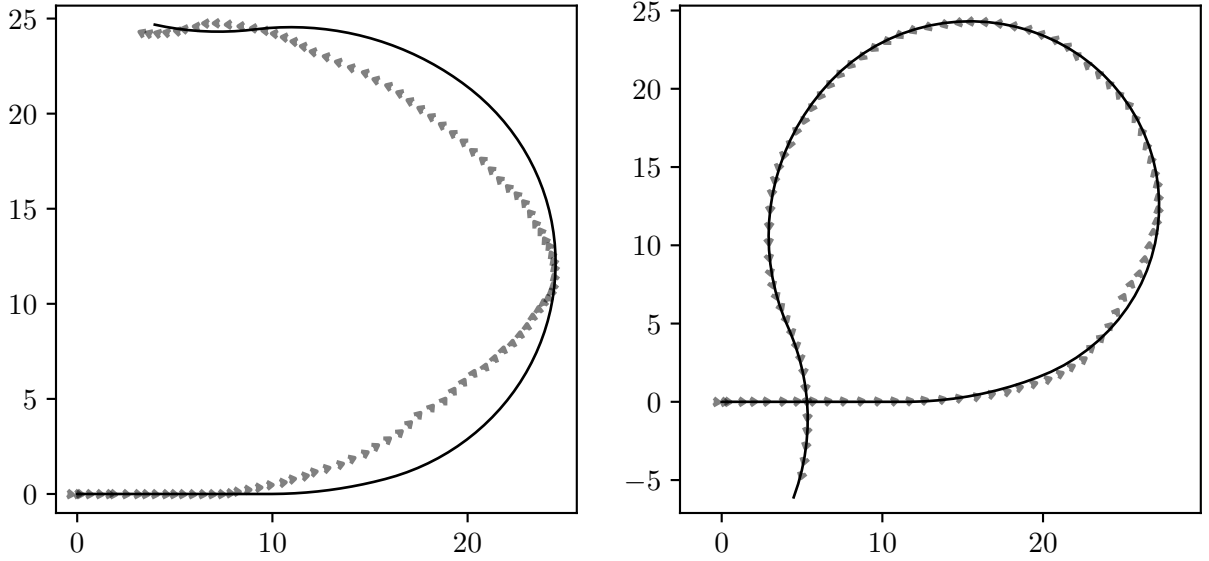
16

Figure 6: Left: untuned policy. Right: tuned policy. Black line is the path and the gray triangles represent the position and orientation of the vehicle. The tuned policy is able to track the path better and go faster.

We represent the amount of good held at each node as $h_t \in \mathbf{R}_+^n$ (the set of nonnegative $n$-vectors). The prices at which we can buy the good from the suppliers are denoted $p_t \in \mathbf{R}_+^k$, the (fixed) prices at which we can sell the goods to consumers are denoted $r \in \mathbf{R}_+^c$, and the customer demand is denoted $d_t \in \mathbf{R}_+^c$. Our inputs are $b_t \in \mathbf{R}_+^k$, the quantity of the good that we buy from the suppliers, $s_t \in \mathbf{R}_+^c$, the quantity that we sell to the consumers, and $z_t \in \mathbf{R}_+^{m-k-c}$, the quantity that we ship across the internode links. The state and inputs are

$$x_t = (h_t, p_t, d_t), \quad u_t = (b_t, s_t, z_t).$$

The system dynamics are

$$h_{t+1} = h_t + (A^{\mathrm{in}} - A^{\mathrm{out}})u_t,$$

where $A^{\mathrm{in}} \in \mathbf{R}^{n \times m}$ and $A^{\mathrm{out}} \in \mathbf{R}^{n \times m}$; $A_{ij}^{\mathrm{in(out)}}$ is 1 if link $j$ enters (exits) node $i$ and 0 otherwise.

The input and state are constrained in several ways. Warehouses have maximum capacities given by $h_{\max} \in \mathbf{R}_+^n$, i.e., $h_t \le h_{\max}$ (where the inequalities are elementwise), and links have maximum capacities given by $u_{\max} \in \mathbf{R}_+^m$, i.e., $u_t \le u_{\max}$. In addition, the amount of goods shipped out of a node cannot be more than the amount on hand, or $A^{\mathrm{out}}u_t \le h_t$. Finally, we require that we sell no more than the demand, or $s_t \le d_t$.

We model the unknown future supplier prices and demands as random disturbances $w_t = (p_{t+1}, d_{t+1})$ with joint log-normal distribution, i.e., $\log w_t = (\log p_{t+1}, \log d_{t+1}) \sim \mathcal{N}(\mu, \Sigma)$.
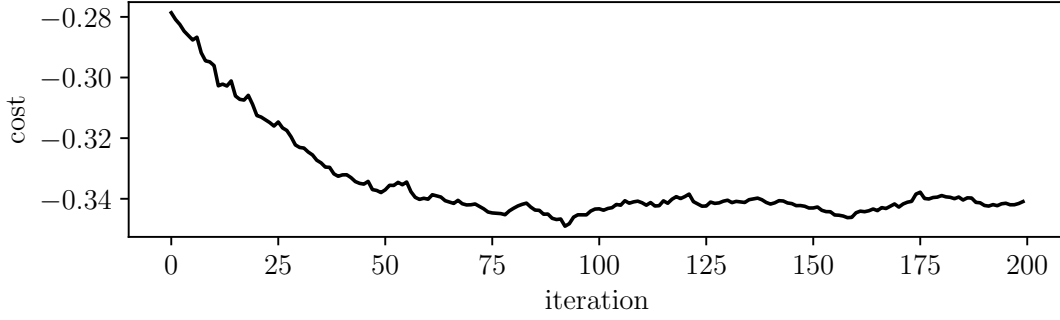
Figure 7: Tuning a supply chain policy.

The goal of our supply chain is to maximize profit, which depends on several quantities. Our payment to the suppliers is $p_t^T b_t$, we obtain revenues $r^T s_t$ for selling the good to consumers, and we incur a shipment cost $\tau^T z_t$, where $\tau \in \mathbf{R}_+^{m-k-c}$ is the cost of shipping a unit of good across the internode links. We also incur a cost for holding or storing $h_t$ in the warehouses; this is represented by a quadratic function $\alpha^T h_t + \beta^T h_t^2$, where $\alpha, \beta \in \mathbf{R}_{++}^n$ and the square is elementwise. Our cost is our average negative profit, or

$$\psi(X, U, W) = \frac{1}{T} \sum_{t=0}^{T-1} p_t^T b_t - r^T s_t + \tau^T z_t + \alpha^T h_t + \beta^T h_t^2 + I(x_t, u_t).$$

Here, $I(x_t, u_t)$ enforces the constraints mentioned above; $I(x_t, u_t)$ is 0 if $x_t$ and $u_t$ lie in the set

$$\left\{ 0 \le h_t \le h_{\max}, \quad 0 \le u_t \le u_{\max}, \quad A^{\mathrm{out}} u_t \le h_t, \quad s_t \le d_t \right\},$$

and $+\infty$ otherwise.

**Policy.** The policy seeks to maximize profit by computing $(b_t, s_t, z_t)$ as

$$
\begin{aligned}
\phi(h_t, p_t, d_t) \quad = \quad & \underset{b,s,z}{\operatorname{argmax}} \quad && -p_t^T b + r^T s - \tau^T z - \|Sh^+\|_2^2 - q^T h^+ \\
& \text{subject to} \quad && h^+ = h_t + (A^{\mathrm{in}} - A^{\mathrm{out}})(b, s, z) \\
& && 0 \le h^+ \le h_{\max}, \quad 0 \le (b, s, z) \le u_{\max}, \\
& && A^{\mathrm{out}}(b, s, z) \le h_t, \quad s \le d_t.
\end{aligned}
$$

where the parameters are $\theta = (S, q)$ with $S \in \mathbf{R}^{n \times n}$ and $q \in \mathbf{R}^n$. This COCP is an ADP policy and we can interpret the term $-\|Sh^+\|_2^2 - q^T h^+$ as our approximate value function applied to the next state.

**Numerical example.** We consider a supply chain over horizon $T = 20$ with $n = 4$ nodes, $m = 8$ links, $k = 2$ supply links, and $c = 2$ consumer links. The initial value of the network
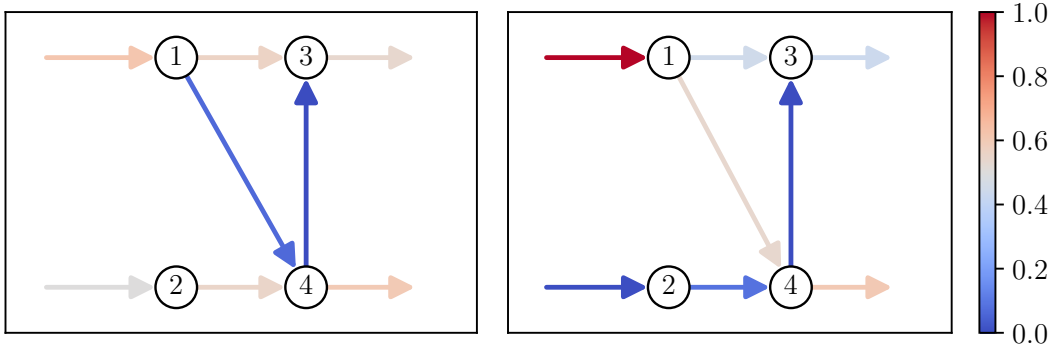
18

Figure 8: Supply chain network. Left: untuned policy. Right: tuned policy. Colors indicate the normalized shipments between 0 and 1.

storage is chosen uniformly between 0 and $h_{\max}$, *i.e.*, $h_0 \sim \mathcal{U}(0, h_{\max})$. The log supplier prices and consumer demands have mean and covariance

$$\mu = (0.0, 0.1, 0.0, 0.4), \quad \Sigma = 0.04I.$$

Therefore, the supplier prices have mean $(1.02, 1.13)$ and the consumer demands have mean $(1.02, 1.52)$. The consumer prices are $r = (1.4)\mathbf{1}$. We set the maximum nodes capacity to $h_{\max} = (3)\mathbf{1}$ and links capacity to $u_{\max} = (2)\mathbf{1}$. The storage cost parameters are $\alpha = \beta = (0.01)\mathbf{1}$. Node 1 is connected to the supplier with lower average price and node 4 to the consumer with higher demand.

We initialize the parameters of our policy to $S = I$ and $q = -h_{\max}$. In this way, the approximate value function is centered at $h_{\max}/2$ so that we try to keep the storage of each node at medium capacity.

We ran our method over 200 iterations, with $K = 10$ using the stochastic gradient method with step size 0.05. Figure 7 shows the per-iteration cost on a held-out random seed while training. Over the course of training the cost decreased by 22.35 percent from $-0.279$ to $-0.341$. The resulting parameters are

$$S^T S = \begin{bmatrix} 0.64 & 0.30 & 0.02 & -0.06 \\ 0.30 & 1.44 & 0.32 & 0.30 \\ 0.02 & 0.32 & 1.14 & 0.06 \\ -0.06 & 0.30 & 0.06 & 1.01 \end{bmatrix}, \quad q = (-3.05, -2.92, -2.97, -2.99).$$

The diagonal of $S^T S$ shows that the learned policy especially penalizes storing goods in nodes connected to more expensive suppliers, *e.g.*, node 2, or to consumers with lower demand, *e.g.*, node 3. Figure 8 shows the supply chain structure and displays the average shipment, normalized between 0 and 1; figure 9 the simulated storage $h_t$ for the untuned and tuned policy on a held-out random seed.
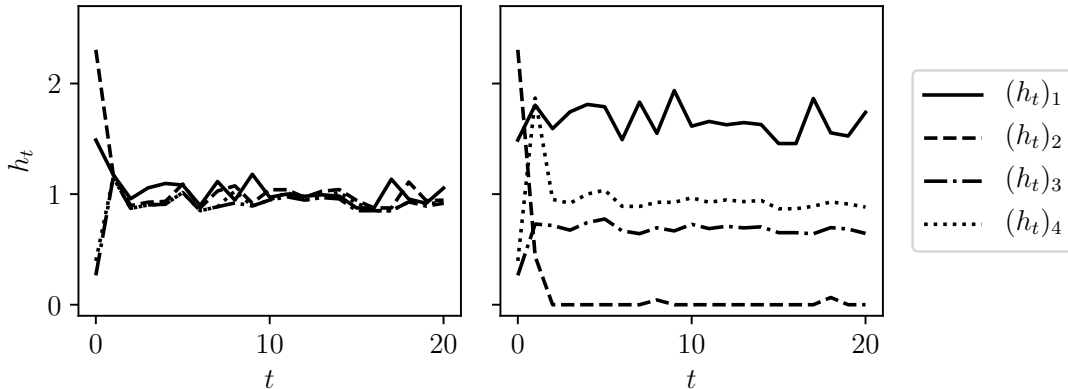
Figure 9: Left: untuned policy. Right: tuned policy. Supply chain storage $h_t$ for each node over time.

# 6  Extensions and variations

**Estimation.**  Our approach is not limited to tuning policies for control. As we alluded to before, our approach can also be used to learn convex optimization state estimators, for example Kalman filters or moving horizon estimators. The setup is exactly the same, in that we learn or tune parameters that appear in the state estimation procedure to maximize some performance metric. (A similar approach was adopted in [15], where the authors fit parameters in a Kalman smoother to observed data.) Also, since COCPs are applied to the estimated state, we can in fact jointly tune parameters in the COCP along with the parameters in the state estimator.

**Prediction.**  In an MPC policy, one could tune parameters in the function that predicts the disturbances together with the controller parameters. As a specific example, we mention that the parameters in a Markowitz policy, such as the expected return, could be computed using a parametrized prediction function, and this function could be tuned jointly with the other parameters in the COCP.

**Nonconvex optimization control policies (NCOCPs).**  An NCOCP is an optimization-based control policy that is evaluated by solving a *nonconvex* optimization problem. Parameters in NCOCPs can be tuned in the same way that we tune COCPs in this paper. Although the solution to a nonconvex optimization problem might be nonunique or hard to find, one can differentiate a local solution map to a smooth nonconvex optimization problem by implicitly differentiating the KKT conditions [50]. This is done in [6], where the authors define an MPC-based NCOCP.

# References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, pages 9558–9570, 2019.

[3] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 1(2):107–115, 2019.

[4] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[5] B. Amos. *Differentiable optimization-based modeling for machine learning*. PhD thesis, Carnegie Mellon University, 2019.

[6] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable MPC for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, pages 8299–8310, 2018.

[7] B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In *Proc. Intl. Conf. on Machine Learning*, pages 146–155, 2017.

[8] B. Anderson and J. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, Inc., 1990.

[9] R. Anderson. Recent advances in finding best operating conditions. *Journal of the American Statistical Association*, 48(264):789–798, 1953.

[10] K. Åström, T. Hägglund, C. Hang, and W. Ho. Automatic tuning and adaptation for pid controllers-a survey. *Control Engineering Practice*, 1(4):699–714, 1993.

[11] K. Åström and B. Wittenmark. *Adaptive Control*. Courier Corporation, 2013.

[12] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd. Embedded code generation using the OSQP solver. In *IEEE Conference on Decision and Control*, 2017.

[13] S. Barratt. On the differentiability of the solution to convex optimization problems. *arXiv preprint arXiv:1804.05098*, 2018.

[14] S. Barratt and S. Boyd. Stochastic control with affine dynamics and extended quadratic costs. *arXiv preprint arXiv:1811.00168*, 2018.

[15] S. Barratt and S. Boyd. Fitting a kalman smoother to data. *arXiv preprint arXiv:1910.08615*, 2019.

[16] R. Bellman. *Dynamic Programming*. Princeton University Press, 1 edition, 1957.

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

[18] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[19] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[20] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[21] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 4th edition, 2017.

[22] D. Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific, 1 edition, 2019.

[23] D. Bertsekas, V. Borkar, and A. Nedić. Improved temporal difference methods with linear function approximation. *Learning and Approximate Dynamic Programming*, pages 231–255, 2004.

[24] D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic Programming*, volume 5. Athena Scientific, 1996.

[25] L. Blackmore. Autonomous precision landing of space rockets. *The BRIDGE*, 26(4), 2016.

[26] J. Boot. On sensitivity analysis in convex quadratic programming problems. *Operations Research*, 11(5):771–786, 1963.

[27] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[28] L. Bottou, F. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[29] S. Boyd, E. Busseti, S. Diamond, R. Kahn, K. Koh, P. Nystrup, and J. Speth. Multi-period trading via convex optimization. *Foundations and Trends® in Optimization*, 3(1):1–76, 2017.

[30] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[31] E. Busseti, W. Moursi, and S. Boyd. Solution refinement at regular points of conic problems. *Computational Optimization and Applications*, 74:627–643, 2019.

[32] Center for Research in Security Prices. Stock and security files, 2019.

[33] E. Chu, N. Parikh, A. Domahidi, and S. Boyd. Code generation for embedded second-order cone programming. In *European Control Conference*, pages 1547–1552. IEEE, 2013.

[34] W. Cochran. *Sampling Techniques*. John Wiley & Sons, 2007.

[35] A. Conn, K. Scheinberg, and L. Vicente. *Introduction to Derivative-Free Optimization*, volume 8. SIAM, 2009.

[36] M. Corless and G. Leitmann. Controller design for uncertain systems via Lyapunov functions. In *American Control Conference*, pages 2019–2025. IEEE, 1988.

[37] G. Cornuejols and R. Tütüncü. *Optimization Methods in Finance*. Cambridge University Press, 2006.

[38] D. De Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[39] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[40] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference*, pages 3071–3076. IEEE, 2013.

[41] C. Gerdes. ME 227 vehicle dynamics and control course notes, 2019. Lectures 1 and 2.

[42] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[43] G. Gordon. Stable function approximation in dynamic programming. In *Machine Learning*, pages 261–268. Elsevier, 1995.

[44] R. Grinold and R. Kahn. *Active Portfolio Management: A Quantitative Approach for Producing Superior Returns and Controlling Risk*. McGraw Hill, 2000.

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

[46] J. Jerez, P. Goulart, S. Richter, G. Constantinides, E. C. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, 59(12):3238–3251, 2014.

[47] R. Kalman. Contributions to the theory of optimal control. *Boletin de la Sociedad Matematica Mexicana*, 5(2):102–119, 1960.

[48] A. Keshavarz. *Convex methods for approximate dynamic programming*. PhD thesis, Stanford University, 2012.

[49] A. Keshavarz and S. Boyd. Quadratic approximate dynamic programming for input-affine systems. *Intl. Journal of Robust and Nonlinear Control*, 24(3):432–449, 2014.

[50] H. Kuhn and A. Tucker. Nonlinear programming. In *Berkeley Symposium on Mathematical Statistics and Probability, 1950*, pages 481–492. University of California Press, 1951.

[51] S. Kuindersma, F. Permenter, and R. Tedrake. An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *Proc. Intl. on Robotics and Automation (ICRA)*, page 2589–2594, Hong Kong, China, 2014. IEEE, IEEE.

[52] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[53] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[54] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[55] N. Minorsky. Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309, 1922.

[56] J. Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

[57] A. Nedić and D. Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110, 2003.

[58] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[59] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 2.1.0. https://github.com/cvxgrp/scs, 2017.

[60] M. Okada, L. Rigazio, and T. Aoshima. Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*, 2017.

[61] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proc. Intl. Conf. on Machine Learning*, pages 1310–1318, 2013.

[62] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[63] W. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons, 2007.

[64] W. Powell, H. Simao, and B. Bouzaiene-Ayari. Approximate dynamic programming in transportation and logistics: a unified framework. *EURO Journal on Transportation and Logistics*, 1(3):237–284, 2012.

[65] J. Rawlings and D. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.

[66] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[67] U. Rosolia and F. Borrelli. Learning model predictive control for iterative tasks: A data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, 2017.

[68] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1, 1988.

[69] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

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

[71] F. Solis and R. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[72] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *arXiv preprint arXiv:1711.08013*, 2019.

[73] B. Stellato, T. Geyer, and P. Goulart. High-speed finite control set model predictive control for power electronics. *IEEE Transactions on Power Electronics*, 32(5):4007–4020, 2017.

[74] G. Stewart and F. Borrelli. A model predictive control framework for industrial turbodiesel engine control. In *IEEE Conference on Decision and Control (CDC)*, pages 5704–5711, 2008.

[75] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[76] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT press, 2018.

[77] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel. Learning from the hindsight plan–episodic MPC improvement. In *IEEE Intl. on Robotics and Automation (ICRA)*, pages 336–343, 2017.

[78] Y. Wang and S. Boyd. Performance bounds for linear stochastic control. *Systems & Control Letters*, 58(3):178–182, 2009.

[79] Y. Wang and S. Boyd. Fast evaluation of quadratic control-Lyapunov policy. *IEEE Transactions on Control Systems Technology*, 19(4):939–946, 2010.

[80] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010.

[81] Y. Wang, B. O'Donoghue, and S. Boyd. Approximate dynamic programming via iterated Bellman inequalities. *Intl. Journal of Robust and Nonlinear Control*, 25(10):1472–1496, 2014.

[82] P. Werbos. Backpropagation through time: What it does and how to do it. *Proc. IEEE*, 78(10):1550–1560, 1990.

[83] R. Williams. *Reinforcement-Learning Connectionist Systems.* College of Computer Science, Northeastern University, 1987.