Optimal and Learning-Based Control
Course Notes

James Harrison[1]

April 25, 2020

[1]Contact: jharrison@stanford.edu

Foreword

These notes accompany the newly revised (Spring 2019 to current) version of *AA 203: Optimal and Learning-Based Control* at Stanford. The goal of this new course is to present a unified treatment of optimal control and reinforcement learning (RL), with an emphasis on model-based reinforcement learning. The goal of the instructors is to unify the subjects as much as possible, and to concretize connections between these research communities.

How is this course different from a standard class on Optimal Control? First, we will emphasize practical computational tools for real world optimal control problems, such as model predictive control and sequential convex programming. Beyond this, the last third of the course focuses on the case in which an exact model of the system is not available. We will discuss this setting both in the online context (typically referred to as adaptive optimal control) and in the episodic context (the typical setting for reinforcement learning).

How is this course different from a standard class on Reinforcement Learning? Many courses on reinforcement learning focus primarily on the setting of discrete Markov Decision Processes (MDPs), whereas we will focus primarily on continuous MDPs. More importantly, the focus on discrete MDPs leads planning with a known model (which is typically referred to as "planning" or "control" in RL) to be relatively simple. In this course, we will spend considerably more time focusing on planning with a known model in both continuous and discrete time. Finally, the focus of this course will primarily be on model-based methods. We will touch briefly on model-free methods at the end, and combinations of model-free and model-based approaches.

A Note on Notation

The notation and language used in the control theory and reinforcement learning communities vary substantially, as so we will state all of the notational choices we make in this section. First, optimal control problems are typically stated in terms of minimizing a cost function, whereas reinforcement learning problems aim to maximize a reward. These are mathematically identical statements, where one is simply the negation of the other. Herein, we will use the control theoretic approach of cost minimization. We write c for the cost function, f for the system dynamics, and denote the state and action at time t as $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ respectively. We write scalars as lower case letters, vectors as bold lower case letters, and matrices as upper case letters. We write a deterministic policy as $\pi(\boldsymbol{x})$, and a stochastic policy as $\pi(\boldsymbol{u} \mid \boldsymbol{x})$. We write the cost-to-go (negation of the value function) associated with policy π at time t and state $\boldsymbol{x}$ as $J_t^\pi(\boldsymbol{x})$. We will also sometimes refer to the cost-to-go as the value, but in these notes we are always referring to the expected sum of future costs. For an in-depth discussion of the notational and language differences between the artificial intelligence and control theory communities, we refer the reader to [Pow12].

For notational convenience, we will write the Hessian of a function $f(x)$, evaluated at x^*, as $\nabla^2 f(x^*)$.

Prerequisites

While these notes aim to be almost entirely self contained, familiarity with undergraduate level calculus, differential equations, and linear algebra (equivalent to CME 102 and EE 263 at Stanford) are assumed. We will briefly review nonlinear optimization in the first section of these notes, but previous experience with optimization (e.g. EE 364A) will be helpful. Finally, previous experience with machine learning (at the level of CS 229) is beneficial.

Omissions

This course (and these notes) aim to cover the content of at least three distinct fields, each with many papers published every year[1]. As a consequence, we skip over many topics. At present, we avoid covering:

- **Motion planning beyond trajectory optimization**, including sampling-based motion planning. For this we refer the reader to the excellent book by LaValle [LaV06].
- **Lyapunov analysis and stability analysis in adaptive control**. We refer the reader to [ÅW13, IS12].
- **Imitation learning**.

Acknowledgments

We acknowledge the students of the 2019 iteration of AA 203, who pointed out many typos. We also acknowledge the former course assistants of AA 203 who helped in the preparation of the material covered in this class and development of this class—in particular, Ed Schmerling, Federico Rossi, Sumeet Singh, and Jonathan Lacotte.

[1]We primarily include references to literature in adaptive control, optimal control, and reinforcement learning, but related work is also published in economics, neuroscience, operations research and quantitative finance, as well as many other fields and sub-fields.

Contents

Part I

Optimization and Optimal Control

Chapter 1

Nonlinear Optimization

In this section we discuss the generic nonlinear optimization problem that forms the basis for the rest of the material presented in this class. We write the minimization problem as

$$\min_{\boldsymbol{x} \in \mathcal{X}} \quad f(\boldsymbol{x})$$

where f is the cost function, usually assumed twice continuously differentiable, $x \in \mathbb{R}^n$ is the optimization variable, and $\mathcal{X} \subset \mathbb{R}^n$ is the constraint set. The special case in which the cost function is linear and the constraint set is specified by linear equations and/or inequalities is *linear optimization*, which we will not discuss.

1.1 Unconstrained Nonlinear Optimization

We will first address the unconstrained case, in which $\mathcal{X} = \mathbb{R}^n$. A vector $\boldsymbol{x}^*$ is said to be an unconstrained *local minimum* if there exists $\epsilon > 0$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^*\| \leq \epsilon\}$, and $\boldsymbol{x}^*$ is said to be an unconstrained *global minimum* if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $x \in \mathbb{R}^n$.

1.1.1 Necessary Conditions for Optimality

For a differentiable cost function, we can compare the cost of a point to its neighbors by considering a small variation $\Delta \boldsymbol{x}$ from $\boldsymbol{x}^*$. By using Taylor expansions, this yields a first and second order cost variation

$$f(\boldsymbol{x}^* + \Delta \boldsymbol{x}) - f(\boldsymbol{x}^*) \approx \nabla f(\boldsymbol{x}^*)^T \Delta \boldsymbol{x} + \frac{1}{2} \Delta \boldsymbol{x}^T \nabla^2 f(\boldsymbol{x}^*) \Delta \boldsymbol{x}. \tag{1.1}$$

Setting $\Delta \boldsymbol{x}$ to be equal to positive and negative multiples of the unit coordinate vector, we have

$$\frac{\partial f(\boldsymbol{x}^*)}{\partial x_i} \geq 0 \tag{1.2}$$

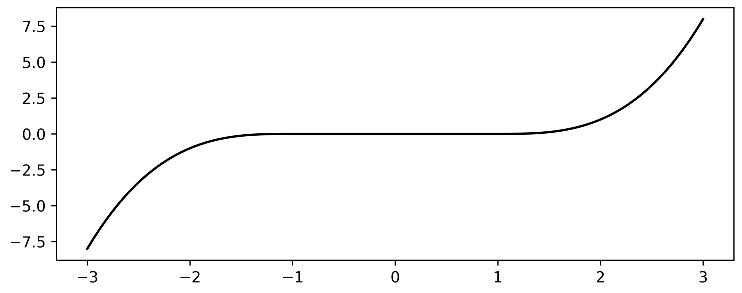


Figure 1.1: An example of a function for which the necessary conditions of optimality are satisfied but the sufficient conditions are not.

where x_i denotes the i'th coordinate of $\boldsymbol{x}$, and

$$\frac{\partial f(\boldsymbol{x}^*)}{\partial x_i} \leq 0 \tag{1.3}$$

for all i, which is only satisfied by $\nabla f(\boldsymbol{x}^*) = 0$. This is referred to as the *first order necessary condition for optimality.* Looking at the second order variation, and noting that $\nabla f(\boldsymbol{x}^*) = 0$, we expect

$$f(\boldsymbol{x}^* + \Delta\boldsymbol{x}) - f(\boldsymbol{x}^*) \geq 0 \tag{1.4}$$

and thus

$$\Delta\boldsymbol{x}^T \nabla^2 f(\boldsymbol{x}^*) \Delta\boldsymbol{x} \geq 0 \tag{1.5}$$

which implies $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite. This is referred to as the *second order necessary condition for optimality.* Stating these conditons formally,

Theorem 1.1.1 (Necessary Conditions for Optimality (NOC))**.** *Let $\boldsymbol{x}^*$ be an unconstrained local minimum of $f : \mathbb{R}^n \to \mathbb{R}$ and $f \in C^1$ in an open set S containing $\boldsymbol{x}^*$. Then,*

$$\nabla f(\boldsymbol{x}^*) = 0. \tag{1.6}$$

If $f \in C^2$ within S, $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite.

Proof. See section 1.1 of [Ber16].

1.1.2 Sufficient Conditions for Optimality

If we strengthen the second order condition to $\nabla^2 f(\boldsymbol{x}^*)$ being positive definite, we have the sufficient conditions for $\boldsymbol{x}^*$ being a local minimum. Why is the second order necessary conditions not sufficient? An example function is given in figure 1.1. Formally,

Theorem 1.1.2 (Sufficient Conditions for Optimality (SOC))**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be C^2 in an open set S. Suppose a vector $\boldsymbol{x}^*$ satisfies the conditions $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ is positive definite. Then $\boldsymbol{x}^*$ is a strict unconstrained local minimum of f.*

Proof is again given in Section 1.1 of [Ber16]. There are several reasons why the optimality conditions are important. In a general nonlinear optimization setting, they can be used to filter candidates for global minima. They can be used for sensitivity analysis, in which the sensitivity of $\boldsymbol{x}^*$ to model parameters can be quantified [Ber16]. This is common in e.g. microeconomics. Finally, these conditions often provide the basis for the design and analysis of optimization algorithms.

1.1.3 Special case: Convex Optimization

A special case within nonlinear optimization is the set of *convex optimization* problems. A set $S \subset \mathbb{R}^n$ is called *convex* if

$$\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y} \in S, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in S, \forall \alpha \in [0,1]. \tag{1.7}$$

For S convex, a function $f : S \to \mathbb{R}$ is called convex if

$$f(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{y}). \tag{1.8}$$

This class of problems has several important characteristics. If f is convex, then

- A local minimum of f over S is also a global minimum over S. If in addition f is strictly convex (the inequality in (1.8) is strict), there exists at most one global minimum of f.
- If $f \in C^1$ and convex, and the set S is open, $\nabla f(\boldsymbol{x}^*) = 0$ is a necessary and sufficient condition for a vector $\boldsymbol{x}^* \in S$ to be a global minimum over S.

Convex optimization problems have several nice properties that make them (usually) computationally efficient to solve, and the first property above gives a certificate of having obtained global optimality that is difficult or impossible to obtain in the general nonlinear optimization setting. For a thorough treatment of convex optimization theory and algorithms, see [BV04].

1.1.4 Computational Methods

In this subsection we will discuss the class of algorithms known as *gradient methods* for finding local minima in nonlinear optimization problems. These approaches, rely (roughly) on following the gradient of the function "downhill", toward the minima. More concretely, these algorithms rely on taking steps of the form

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k \tag{1.9}$$

where if $\nabla f(\boldsymbol{x}) \neq 0$, $\boldsymbol{d}^k$ is chosen so that

$$\nabla f(\boldsymbol{x})^T \boldsymbol{d}^k < 0 \tag{1.10}$$

and $\alpha > 0$. Typically, the step size α^k is chosen such that

$$f(\boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k) < f(\boldsymbol{x}^k), \tag{1.11}$$

but generally, the step size and the direction of descent ($\boldsymbol{d}^k$) are tuning parameters.

We will look at the general class of descent directions of the form

$$\boldsymbol{d}^k = -D^k \nabla f(\boldsymbol{x}^k) \tag{1.12}$$

where $D^k > 0$ (note that this guarantees $\nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k < 0$).

Steepest descent, $D^k = I$. The simplest choice of descent direction is directly following the gradient, and ignoring second order function information. In practice, this often leads to slow convergence (figure 1.2a) and possible oscillation (figure 1.2b).

Newton's Method, $D^k = (\nabla^2 f(\boldsymbol{x}^k))^{-1}$. The underlying idea of this approach is to at each iteration, minimize the quadratic approximation of f around $\boldsymbol{x}^k$,

$$f^k(\boldsymbol{x}) = f(\boldsymbol{x}^k) + \nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x} - \boldsymbol{x}^k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^k)^T \nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k). \tag{1.13}$$

Setting the derivative of this to zero, we obtain

$$\nabla f(\boldsymbol{x}^k) + \nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k) = 0 \tag{1.14}$$

and thus, by setting $\boldsymbol{x}^{k+1}$ to be the $\boldsymbol{x}$ that satisfies the above, we get the

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - (\nabla^2 f(\boldsymbol{x}^k))^{-1} \nabla f(\boldsymbol{x}^k) \tag{1.15}$$

or more generally,

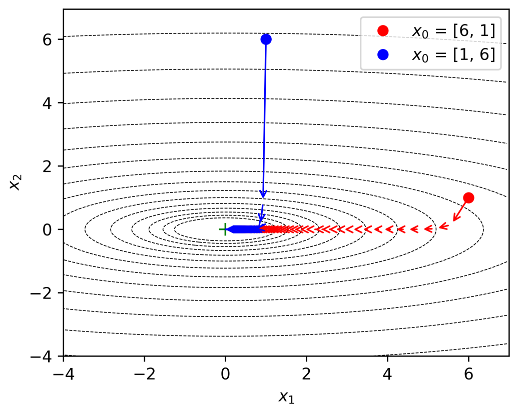
$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha(\nabla^2 f(\boldsymbol{x}^k))^{-1} \nabla f(\boldsymbol{x}^k). \tag{1.16}$$

Note that this update is only valid for $\nabla^2 f(\boldsymbol{x}^k) \succ 0$. When this condition doesn't hold, $\boldsymbol{x}^{k+1}$ is not a minimizer of the second order approximation (as a result of the SOCs). See figure 1.2d for an example where Newton's method converges in one step, as a result of the cost function being quadratic.

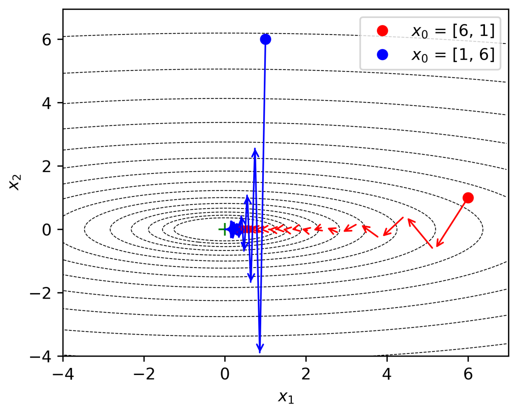
Diagonally scaled steepest descent, $D^k = \mathbf{diag}(d_1^k, \ldots, d_n^k)$. Have $d_i^k > 0 \forall i$. A popular choice is

$$d_i^k = \left(\frac{\partial^2 f(\boldsymbol{x}^k)}{\partial x_i^2}\right)^{-1} \tag{1.17}$$

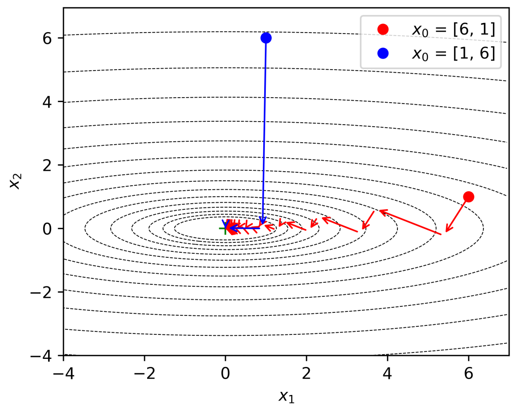
which is a diagonal approximation of the Hessian



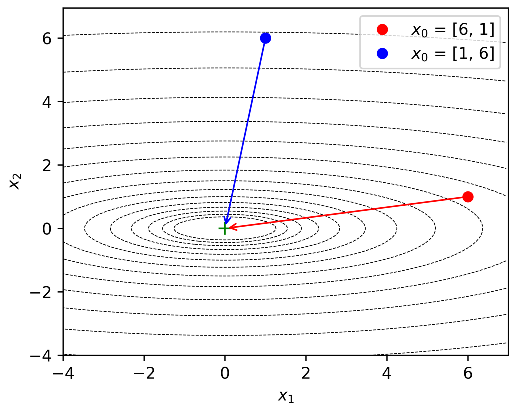
(a) Steepest descent, small fixed step size.



(b) Steepest descent, large fixed step size.



(c) Steepest descent, step size chosen via line search.



(d) Newton's method. Note that the method converges in one step.

Figure 1.2: Comparison of steepest descent methods with various step sizes, and Newton's method, on the same quadratic cost function.

Modified Newton's method, $D^k = (\nabla^2 f(\boldsymbol{x}^0))^{-1}$. Requires $\nabla^2 f(\boldsymbol{x}^0) > 0$. For cases in which one expects $\nabla^2 f(\boldsymbol{x}^0) \approx \nabla^2 f(\boldsymbol{x}^k)$, this removes having to compute the Hessian at each step.

In addition to choosing the descent direction, there also exist a variety of methods to choose the step size α. A computationally intensive but efficient (in terms of the number of steps taken) is using a minimization rule of the form

$$\alpha^k = \text{argmin}_{\alpha \geq 0} f(\boldsymbol{x}^k + \alpha \boldsymbol{d}^k) \tag{1.18}$$

which is usually solved via line search (figure 1.2c). Alternative approaches include a limited minimization rule, in which you constrain $\alpha^k \in [0, s]$ during the line search, or simpler approach such as a constant step size (which may not guarantee convergence), or a diminishing scheduled step size. In this last case, schedules are typically chosen such that $\alpha^k \to 0$ as $k \to \infty$, while $\sum_{k=0}^{\infty} \alpha^k = +\infty$.

1.2 Constrained Nonlinear Optimization

In this section we will address the general constrained nonlinear optimization problem,

$$\min_{\boldsymbol{x} \in \mathcal{X}} \quad f(\boldsymbol{x})$$

which may equivalently be written

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

where the set $\mathcal{X}$ is usually specified in terms of equality and inequality constraints. To operate within this problem structure, we will develop a set of optimality conditions involving auxiliary variables called *Lagrange multipliers*.

1.2.1 Equality Constrained Optimization

We will first look at optimization with equality constraints of the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & h_i(\boldsymbol{x}) = 0, \quad i = 1, \dots, m \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $h_i : \mathbb{R}^n \to \mathbb{R}$ are C^1. We will write $\boldsymbol{h} = [h_1, \dots, h_m]^T$. For a given local minimum $\boldsymbol{x}^*$, there exist scalars $\lambda_1, \dots, \lambda_m$ called Lagrange multipliers such that

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\boldsymbol{x}^*) = 0. \tag{1.19}$$

There are several possible interpretations for Lagrange multipliers. First, note that the cost gradient $f(\boldsymbol{x}^*)$ is in the subspace spanned by the constraint gradients at $\boldsymbol{x}^*$. Equivalently, $f(\boldsymbol{x}^*)$ is orthogonal to the subspace of first order feasible variations

$$V(\boldsymbol{x}^*) = \{\Delta \boldsymbol{x} \mid \nabla h_i(\boldsymbol{x}^*)^T \Delta \boldsymbol{x} = 0, i = 1, \dots, m\}. \tag{1.20}$$

This subspace is the space of variations $\Delta \boldsymbol{x}$ for which $\boldsymbol{x} = \boldsymbol{x}^* + \Delta \boldsymbol{x}$ satisfies the constraint $\boldsymbol{h}(\boldsymbol{x}) = 0$ up to first order. Therefore, at a local minimum, the first order cost variation $\nabla f(\boldsymbol{x}^*)^T \Delta \boldsymbol{x}$ is zero for all variations $\Delta \boldsymbol{x}$ in this space.

Given this informal understanding, we may now precisely state the necessary conditions for optimality in constrained optimization.

Theorem 1.2.1 (NOC for equality constrained optimization)**.** *Let $\boldsymbol{x}^*$ be a local minimum of f subject to $\boldsymbol{h}(\boldsymbol{x}) = 0$, and assume that the constraint gradients $\nabla h_1(\boldsymbol{x}^*), \dots, \nabla h_m(\boldsymbol{x}^*)$ are linearly independent. Then there exists a unique vector $\boldsymbol{\lambda}^* = [\lambda_1^*, \dots, \lambda_m^*]^T$ called a Lagrange multiplier vector, such that*

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\boldsymbol{x}^*) = 0. \tag{1.21}$$

If in addition f and $\boldsymbol{h}$ are C^2, we have

$$\boldsymbol{y}^T(\nabla^2 f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla^2 h_i(\boldsymbol{x}^*))\boldsymbol{y} \geq 0, \quad \forall \boldsymbol{y} \in V(\boldsymbol{x}^*) \tag{1.22}$$

where

$$V(\boldsymbol{x}^*) = \{\boldsymbol{y} \mid \nabla h_i(\boldsymbol{x}^*)^T \boldsymbol{y} = 0, i = 1, \dots, m\}. \tag{1.23}$$

Proof. See [Ber16] Section 3.1.1 and 3.1.2. □

We will sketch two possible proofs for the NOC for equality constrained optimization.

Penalty approach. This approach relies on adding a to the cost function a large penalty term for constraint violation. This is the same approach that will be used in proving the necessary conditions for inequality constrained optimization, and is the basis of a variety of practical numerical algorithms.

Elimination approach. This approach views the constraints as a system of m equations with n unknowns, for which m variables can be expressed in terms of the remaining $m - n$ variables. This reduces the problem to an unconstrained optimization problem.

Note that in theorem 1.2.1, we assumed the gradients of the constraint functions were linearly independent. A feasible vector for which this holds is called *regular*. If this condition is violated, a Lagrange multiplier for a local minimum may not exist.

For convenience, we will write the necessary conditions in terms of the Lagrangian function $L : \mathbb{R}^{m+n} \to \mathbb{R}$,

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{x}). \tag{1.24}$$

This function allows the NOC conditions to be succinctly stated as

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{1.25}$$

$$\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{1.26}$$

$$\boldsymbol{y}^T \nabla^2_{\boldsymbol{xx}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)\boldsymbol{y} \geq 0, \quad \forall \boldsymbol{y} \in V(\boldsymbol{x}^*). \tag{1.27}$$

which form a system of $n + m$ equations with $n + m$ unknowns. Given this notation, we can state the sufficient conditions.

Theorem 1.2.2 (SOC for equality constrained optimization)**.** *Assume that f and $\boldsymbol{h}$ are C^2 and let $\boldsymbol{x}^* \in \mathbb{R}^n$ and $\boldsymbol{\lambda}^* \in \mathbb{R}^m$ satisfy*

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{1.28}$$

$$\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{1.29}$$

$$\boldsymbol{y}^T \nabla^2_{\boldsymbol{xx}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \boldsymbol{y} > 0, \quad \forall \boldsymbol{y} \neq 0, \boldsymbol{y} \in V(\boldsymbol{x}^*). \tag{1.30}$$

Proof. See [Ber16] Section 3.2. □

Note that the SOC does not include regularity of $\boldsymbol{x}^*$.

1.2.2 Inequality Constrained Optimization

We will now address the general case, including inequality constraints,

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & h_i(\boldsymbol{x}) = 0, \quad i = 1, \ldots, m \\ & g_j(\boldsymbol{x}) \leq 0, \quad j = 1, \ldots, r \end{aligned}$$

where f, h_i, g_i are C^1. The key intuition for the case of inequality constraints is based on realizing that for any feasible point, some subset of the constraints will be active (for which $g_j(\boldsymbol{x}) = 0$), while the complement of this set will be inactive. We define the active set of inequality constraints, which we denote

$$A(\boldsymbol{x}) = \{j \mid g_j(\boldsymbol{x}) = 0\}. \tag{1.31}$$

A constraint is active at $\boldsymbol{x}$ if it is in $A(\boldsymbol{x})$, otherwise it is inactive. Note that if $\boldsymbol{x}^*$i is a local minimum of the inequality constrained problem, then $\boldsymbol{x}^*$ is a local minimum of the identical problem with the inactive constraints removed. Moreover, at this local minimum, the constraints may be treated as equality constraints. Thus, if $\boldsymbol{x}^*$ is regular, there exists Lagrange multipliers $\lambda_1^*, \ldots, \lambda_m^*$ and $\mu_j^*, j \in A(\boldsymbol{x}^*)$ such that

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\boldsymbol{x}^*) + \sum_{j \in A(\boldsymbol{x}^*)} \mu_j^* \nabla g_j(\boldsymbol{x}^*) = 0. \tag{1.32}$$

We will define the Lagrangian

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{x}) + \sum_{j=1}^{r} \mu_j g_j(\boldsymbol{x}), \tag{1.33}$$

which we will use to state the necessary and sufficient conditions.

Theorem 1.2.3 (Karush-Kuhn-Tucker NOC). *Let $\boldsymbol{x}^*$ be a local minimum for the inequality constrained problem where f, h_i, g_j are C^1 and assume $\boldsymbol{x}^*$ is regular (equality and active inequality constraint gradients are linearly independent). Then, there exists unique Lagrange multiplier vectors $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that*

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = 0 \tag{1.34}$$

$$\boldsymbol{\mu} \geq 0 \tag{1.35}$$

$$\mu_j^* = 0, \quad \forall j \notin A(\boldsymbol{x}^*) \tag{1.36}$$

If in addition, $f, \boldsymbol{h}, \boldsymbol{g}$ are C^2, we have

$$\boldsymbol{y}^T \nabla^2_{\boldsymbol{xx}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \boldsymbol{y} \geq 0 \tag{1.37}$$

for all $\boldsymbol{y}$ such that

$$\nabla h_i(\boldsymbol{x}^*)^T \boldsymbol{y} = 0, \quad i = 1, \ldots, m \tag{1.38}$$

$$\nabla g_j(\boldsymbol{x}^*)^T \boldsymbol{y} = 0, \quad j \in A(\boldsymbol{x}^*) \tag{1.39}$$

Proof. See [Ber16] Section 3.3.1. □

The SOC are obtained similarly to the equality constrained case.

1.3 Further Reading

In this section we have addressed the necessary and sufficient conditions for constrained and unconstrained nonlinear optimization. This section is based heavily on [Ber16], and we refer the reader to this book for further details. We have avoided discussing linear programming, which is itself a large topic of study, about which many books have been written (we refer the reader to [BT97] as a good reference on the subject).

Convex optimization has become a powerful and widespread tool in modern optimal control. While we have only addressed it briefly here, [BV04] offers a fairly comprehensive treatment of the theory and practice of convex optimization. For a succinct overview with a focus on machine learning, we refer the reader to [Kol08].

Chapter 2

Optimal Control Fundamentals

In this section we will introduce the fundamental concepts of optimal control in discrete and continuous time. In particular, we introduce the principle of optimality, which enables dynamic programming. Dynamic programming allows us to solve optimal control problems by recursively solving sub-problems, and this approach is (arguably) the most important concept in our study of optimal control. In addition to studying dynamic programming and the principle of optimality in discrete and continuous time for both deterministic and stochastic systems, we will introduce optimal control algorithms for discrete state spaces.

2.1 The Optimal Control Problem

2.1.1 Optimal Control in Continuous Time

In this section, we will outline the deterministic continuous-time optimal control problem that we will aim to solve. We will denote the state at time t as $\boldsymbol{x}(t) \in \mathbb{R}^n$, and the control as $\boldsymbol{u}(t) \in \mathbb{R}^m$. We will also occasionally write these as $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$, respectively. We will write the continuous-time systems dynamics as

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t). \tag{2.1}$$

We will refer to a history of control input values during an interval $[t_0, t_f]$ as a control history, and we will refer to a history of state values over this interval as a state trajectory.

Different control problems may call for various constraints. For example, we may constrain a quadrotor to only fly in space not occupied by obstacles. Examples of constraints we will see are

- Initial and final conditions, $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$, $\boldsymbol{x}(t_f) = \boldsymbol{x}_f$
- Trajectory constraints, $\underline{\boldsymbol{x}} \leq \boldsymbol{x}(t) \leq \bar{\boldsymbol{x}}$
- Control limits, $\underline{\boldsymbol{u}} \leq \boldsymbol{u}(t) \leq \bar{\boldsymbol{u}}$.

A state trajectory and control history that satisfy the constraints during the entire time interval $[t_0, t_f]$ are called admissible trajectories and admissible controls, respectively.

Finally, we will define the performance measure,

$$J = c_f(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \tag{2.2}$$

where c is the instantaneous cost function, and c_f is the terminal state cost. We are now able to state the continuous-time optimal control problem. We aim to find an admissible control, $\boldsymbol{u}^*$, which causes the system (2.1) to follow an admissible trajectory, $\boldsymbol{x}^*$, that minimizes the performance measure given by (2.2). The minimizer $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ is called an optimal trajectory-control pair.

Note, first of all, that this is an extremely general problem formulation. We have not fixed our system dynamics, cost function, or specific constraints. We can't, in general, guarantee the existence or uniqueness of the optimal solution.

There are two possible solution forms for the optimal control. The first, $\boldsymbol{u}^* = e(\boldsymbol{x}(t_0), t)$ is referred to as an open-loop solution. This is an input function that is applied to the system, without using feedback. Practically, such solutions usually require augmentation with a feedback controller, as small model mismatch may lead to compounding errors. The second possible solution form is a feedback policy, $\boldsymbol{u}^* = \pi(\boldsymbol{x}(t), t)$. This feedback law maps all state-time pairs to an action and thus is usually more robust to possible model mismatch. However, depending on the particular problem formulation, open-loop solutions may be easier to compute.

2.1.2 Optimal Control in Discrete Time

In the previous section, we have developed an optimal control problem statement in continuous time. This approach to system modeling is likely familiar to readers coming from a background in the physical sciences; in particular, physical models from mechanical and electrical engineering will often be stated as differential equations, and so a problem formulation in continuous time is natural. Readers from backgrounds in computer science or operations research on the other hand may be more familiar with dynamics expressed as discrete time difference equations. Moreover, such an approach is more natural for implementation on a digital computer, and thus fluency mapping between these two settings is an important skill. Finally, while work in the optimal control literature discusses both the continuous and discrete time case, the literature in reinforcement learning and artificial intelligence typically presents problems in discrete time.

We will index time with k, where one increment of this index is equal to some increment of time Δt. Thus, $\boldsymbol{x}_k = \boldsymbol{x}(k\Delta t)$. We will write N for the final time. Then, we will write our system dynamics as

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, k) \tag{2.3}$$

and our performance measure as

$$J = c_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} c(\boldsymbol{x}_k, \boldsymbol{u}_k, k) \tag{2.4}$$

where c is now a stage-wise cost function, c_N is a terminal cost function, and the integral of the continuous time formulation is replaced with a sum.

2.1.3 Markovian Decision Problems

Before moving to the principle of optimality, we will formalize the setting in which we will operate for the duration of this class. In particular, we will outline *Markovian* decision problems, variations in their presentation, as well as extensions of this framework. We will first discuss the *perfect state information* case, in which the system state summarizes the full history of the system. We will then briefly discuss the *imperfect state information* case. Typically, finding optimal solutions in this case is much harder than in the perfect state information case. We will introduce it briefly, and in the next chapter we will discuss one practical case in which the imperfect case can be solved effectively.

MDPs

We will first present the Markov decision process in discrete time, and the continuous time case will look very similar. The fundamental idea behind Markov Decision Processes (MDPs) is that the state dynamics are *Markovian*, or obey the *Markov property.* This property says that all information about the previous history of the system is summarized by its current state. While we have previously considered deterministic dynamics, we will consider probabilistic dynamics of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\omega}_k, k) \tag{2.5}$$

where $\boldsymbol{\omega}_k$ is a stochastic disturbance, with $\boldsymbol{\omega}_j$ independent of $\boldsymbol{\omega}_i$ for $i \neq j$. This noise term at time k may depend on the state and action at time k, but it is independent of the state and action at other time steps. Given this probabilistic dynamics model, we can reason about our conditional distribution of $\boldsymbol{x}_{k+1}$ given our current state $\boldsymbol{x}_k$ and action $\boldsymbol{u}_k$, which we will write $p(\boldsymbol{x}_{k+1} \mid \boldsymbol{x}_k, \boldsymbol{u}_k)$. Let

$$\boldsymbol{h}_k = (\boldsymbol{x}_0, \boldsymbol{u}_0, c_0, \ldots, \boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.6}$$

where c_i is the accrued cost at timestep i. Then, a system is said to be Markovian if

$$p(\boldsymbol{x}_{k+1} \mid \boldsymbol{x}_k, \boldsymbol{u}_k) = p(\boldsymbol{x}_{k+1} \mid \boldsymbol{h}_k). \tag{2.7}$$

In addition to Markovian dynamics, we require an *additive* cost function, $c(\boldsymbol{x}_k, \boldsymbol{u}_k, k)$. Finally, we will define the state space $\mathcal{X}$ and action space $\mathcal{U}$, such that $\boldsymbol{x}_k \in \mathcal{X}$, $\boldsymbol{u}_k \in \mathcal{U}$ for all k. Given these ingredients, we will define the Markov decision process as the tuple $(\mathcal{X}, \mathcal{U}, \boldsymbol{f}, c)$.

Are these ingredients sufficient to completely define the optimal control problem introduced in the last section? Or equivalently, given the elements of this tuple, can we specify everything about an optimal control problem? The answer is no: we have not included the constraints (for example, control constraints), the final time, or the initial state. Are these elements necessary for the specification of an optimal control problem? We will break these elements down individually.

Constraints: The control literature frequently includes constraints in the optimal control problem setting. For example, control limits are necessary to ensure commanded actions are physically realizable. In contrast, the artificial intelligence community typically does not consider constraints. For deterministic dynamics, we can include constraints in the cost function: if a constraint is violated, a cost of ∞ is returned. Such an approach is common in the literature on optimization [BV04]. However, this leads to a problem in the stochastic case. Imagine some deterministic dynamics with an additive Gaussian noise term. If we impose state constraints on this system, the infinite support of the Gaussian noise will lead to a non-zero probability of constraints violation at every time step. The tension between stochastic dynamics and constraints is resolved in part by the constrained MDP formulation [Alt99], in which a budget on constraint violation is allowed. The objective, then, is satisfying the constraint violation budget while minimizing the cost. Generally, however, designing meaningful constraints with stochastic dynamics is an active research problem in the control community.

Final time: A final time may be desired by a system designer, and thus they may choose to include this in their optimal control problem. But, is it necessary to include? Imagine an infinite-horizon optimal control problem, in which we plan a feedback policy that will execute for all time. If there exists a state that the system can reach with finite cost, for which the cost for the rest of the problem is finite (for example, a state for which we can achieve zero cost for the rest of time), then the total cost will be finite. If this is not satisfied, then the total cost is infinite. This is problematic, as it removes the ability to distinguish between different control policies, as both will receive the same (infinite) total cost. This can be resolved in several ways. As discussed in the previous section, we can limit the problem setting to a finite final time. In the infinite horizon case, we could also scale the cost accrued at each timestep so that the total cost is finite. One approach is discounting, in which cost is scaled by a time-varying term. An alternative (but less common) approach is to consider *average* cost, as opposed to total cost.

Initial State: In the optimal control problem defined previously, we assumed knowledge of the initial state. This knowledge is important if we are designing an *open-loop* sequence of controls. If we are searching for an optimal *control policy*, we do not require the initial state, as we will find a policy that is optimal for all states. The difference between these two approaches will be discussed in depth later in this section. In reinforcement learning where you assume episodic interaction with a system, it is typical to assume a known initial state

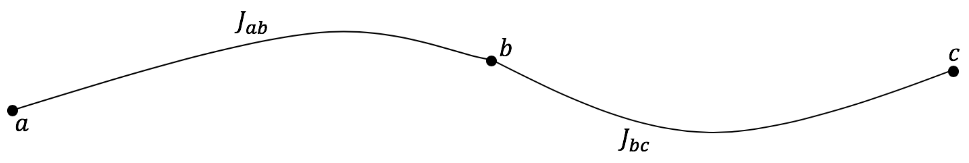


Figure 2.1: An optimal trajectory connecting point a to point c. There are no better (lower cost) trajectories than the sub-trajectory connecting b and c, by the principle of optimality.

or a distribution over initial states. This is important as these algorithms typically result in sub-optimal control policies that perform better close to the regions of the state space where they have observed data, and perform worse further from their previous experience.

POMDPs

In the definition of the Markov decision process, we assumed two necessary features of the dynamics. First, we assumed the dynamics were Markovian; the state contained all information about the history of the system. Second, we assumed that we could directly observe the state of the system (the *perfect information* assumption. In the partially observed MDP (POMDP) setting, we will assume the former but not the later. Specifically, in the POMDP framework, instead of observing the state directly, we observe a (possibly noisy) measurement $\boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\nu}_k, k)$, where $\boldsymbol{\nu}_k$ is a noise term. For example, we may only noisily measure the position of a robot and not the velocity, and thus have incomplete state information. We will typically assume the measurement is independent of the action. Typically, in this setting the measurements are not Markovian. Two strategies are possible: one may design an estimation algorithm to attempt to infer the underlying state $\boldsymbol{x}_k$, and choose control actions based on this quantity. Alternatively, we can look for a (sub-optimal) control policy that maps observations to actions, ignoring state information. We will address the first case in the next section when we discuss the Linear Quadratic Gaussian (LQG) problem setting. Generally however, this is computationally extremely challenging, and our discussion will be primarily focused on the perfect information case.

2.2 Dynamic Programming and the Principle of Optimality

In this chapter we will outline the principle of optimality, and the method of dynamic programming (DP), one of two main approaches to solving the optimal control problem. The second, so-called variational approaches based on Pontryagin's Maximum Principle (PMP) will be discussed in future chapters. While dynamic programming has the strong advantage compared to variational methods of yielding a feedback policy, exactly solving the dynamic programming problem is infeasible for many systems. We will address special cases in which the DP problem can be solved exactly, and approximate methods that work for a wide variety of systems. In particular, in this chapter we will discuss exhaustive approaches to dynamic programming for discrete state space problems; such methods may be used as an approximation method for continuous state space problems. In the next chapter we discuss a special case in which dynamic programming is tractable for continuous state spaces. Again, this special case is used for useful approximations for intractable problems. We will first introduce DP for discrete time systems, before extending these methods to the continuous time setting in the next section.

The principle of optimality is as follows. Figure 2.1 shows a trajectory from point a to c. If the cost of the trajectory, $J_{ac} = J_{ab} + J_{bc}$, is minimal, then J_{bc} is also a minimum cost trajectory connecting b and c. The proof of this principle, stated informally, is simple. Assume there exists an alternative trajectory connecting b and c, for which we will write the cost as $\tilde{J}_{bc}$, that achieves $\tilde{J}_{bc} < J_{bc}$. Then, we have

$$\begin{aligned}\tilde{J}_{ac} &= J_{ab} + \tilde{J}_{bc} && (2.8)\\ &< J_{ab} + J_{bc} && (2.9)\\ &= J_{ac}, && (2.10)\end{aligned}$$

and thus J_{ac} isn't minimal. More formally,

Theorem 2.2.1 (Discrete-time Principle of Optimality: Deterministic Case)**.** *Let $\pi^* = (\pi_0^*, \ldots, \pi_{N-1}^*)$ be an optimal policy. Assume state $\boldsymbol{x}_k$ is reachable. Consider the subproblem whereby we are at $\boldsymbol{x}_k$ at time k and we wish to minimize the cost-to-go from time k to time N. Then the truncated policy $(\pi_k^*, \ldots, \pi_{N-1}^*)$ is optimal for the subproblem.*

Dynamic programming, intuitively, proceeds backwards in time, first solving simpler shorter horizon problems. If we have found the optimal policy for times $k+1$ to $N-1$, along with the associated cost-to-go for each state, choosing the optimal policy for time k is a one step optimization problem. More concretely, dynamic programming iterates backward in time, from $N-1$ to 0, with

$$\begin{aligned}J_N(\boldsymbol{x}_N) &= c_N(\boldsymbol{x}_N) && (2.11)\\ J_k(\boldsymbol{x}_k) &= \min_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)} \left\{ c_k(\boldsymbol{x}_k, \boldsymbol{u}_k, k) + J_{k+1}(f(\boldsymbol{x}_k, \boldsymbol{u}_k, k)) \right\}. && (2.12)\end{aligned}$$

Note that here we have considered only deterministic dynamical systems (there is no stochastic disturbance). Equation (2.12) is one form of the *Bellman equation*, one of the most important relations in optimal control. Critically, this approach changes the optimization problem associated with the optimal control problem from one in which optimization is performed over a sequence of actions of length N, to a collection of N one step optimization problems.

Dynamic programming raises many practical issues if one were to attempt to apply it directly in general. To perform the recursion, J_{k+1} must be known for all $\boldsymbol{x}_{k+1}$ (or more precisely, all $\boldsymbol{x}_{k+1}$ that are reachable from $\boldsymbol{x}_k$). If the state space is discrete (and relatively small), this is tractable as the cost-to-go may just be maintained in tabular form. We will address this case later in this section. However, for general systems, we can not expect to be able to compute the cost-to-go for all states. Possible approaches to make the DP approach tractable are discretizing the state space, approximating the cost-to-go (i.e. restricting the family of functions that J_{k+1} may be in), or interpolating between cost-to-go computed for a finite set of states.

2.2.1 Generalizing the Principle of Optimality: Stochastic Case

We consider systems of the form

$$\boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\omega}_k), \;\; k = 0, \ldots, N-1 \tag{2.13}$$

where $\boldsymbol{\omega}_k \sim p(\cdot \mid \boldsymbol{x}_k, \boldsymbol{u}_k)$ is the disturbance or noise. We write the expected cost under policy $\pi = \{\pi_0, \ldots, \pi_{N-1}\}$ as

$$J^{\pi}(\boldsymbol{x}_0) = \mathbb{E}_{\boldsymbol{\omega}_{0:N-1}}\left[c_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} c_k(\boldsymbol{x}_k, \pi_k(\boldsymbol{x}_k), \boldsymbol{\omega}_k)\right]. \tag{2.14}$$

Then, the stochastic control problem we wish to solve is to find

$$J^*(\boldsymbol{x}_0) = \min_{\pi} J^{\pi}(\boldsymbol{x}_0). \tag{2.15}$$

In contrast to the deterministic optimal control problem, we are specifically interested in finding the optimal closed-loop *policy* in the stochastic case. Closed-loop policies can achieve lower cost than open-loop action sequences when disturbances are present, as they take advantage of current state information in their action selection. Thus, if disturbances cause a system to leave the nominal open-loop trajectory, a policy will continue to act optimally from that new state, whereas an open-loop sequence of actions will potentially act sub-optimally.

We can now state the principle of optimality for the stochastic optimal control problem. Note that this is a strict generalization of the deterministic case.

Theorem 2.2.2 (Discrete-time Principle of Optimality: Stochastic Case)**.** *Let $\pi^* = (\pi_0^*, \ldots, \pi_{N-1}^*)$ be an optimal policy. Assume state $\boldsymbol{x}_k$ is reachable. Consider the tail subproblem*

$$\mathbb{E}_{w_{i:N-1}}\left[c_T(\boldsymbol{x}_N) + \sum_{k=i}^{N-1} c_k(\boldsymbol{x}_k, \pi_k(\boldsymbol{x}_k), \boldsymbol{\omega}_k)\right]. \tag{2.16}$$

Then the truncated policy $(\pi_i^*, \ldots, \pi_{N-1}^*)$ is optimal for the subproblem.

The intuition behind the stochastic principle of optimality is effectively the same as for the deterministic, and the proof is also based on decomposition of the total cost into two cost terms. This is possible due to the linearity of expectation. Stated simply, if a better policy existed for the tail problem, this would imply π^* is suboptimal.

The stochastic version of the principle of optimality leads to a concomitant dynamic programming algorithm, which takes the form

$$J_N(\boldsymbol{x}_N) = c_N(\boldsymbol{x}_N) \tag{2.17}$$

$$J_k(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)} \mathbb{E}_{w_k}\left[c_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\omega}_k) + J_{k+1}(f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\omega}_k))\right] \tag{2.18}$$

and the optimal policy is

$$\pi_k^*(\boldsymbol{x}_k) = \operatorname{argmin}_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)} \mathbb{E}_{w_k}\left[c_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\omega}_k) + J_{k+1}(f(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\omega}_k))\right]. \tag{2.19}$$

2.2.2 Optimal Control in Discrete State Spaces

We will now look at practical algorithms that leverage the previously described dynamic programming principle. The problem setting we consider for these methods are *discrete state space* and *discrete action space* methods. We will first discuss *policy evaluation*: given some policy π and MDP $\mathcal{M}$, we aim to determine the associated expected total cost (or value function), J_π. In this section we will assume a deterministic policy, but this assumption is easily relaxed.

Policy Evaluation

The total expected cost associated with a state $\boldsymbol{u}$ may be written as

$$J_k^\pi(\boldsymbol{x}) \leftarrow c_k(\boldsymbol{x}, \pi(\boldsymbol{x})) + \sum_{\boldsymbol{x}' \in \mathcal{X}} p(\boldsymbol{x}' \mid \boldsymbol{x}, \pi(\boldsymbol{x})) J_{k+1}^\pi(\boldsymbol{x}'). \tag{2.20}$$

Given a model of the dynamics, which we (for now) will assume known, we exactly have access to the density $p(\boldsymbol{x}' \mid \boldsymbol{x}, \pi(\boldsymbol{x}))$. Thus, the algorithm proceeds backward in time, exactly computing this expectation each time.

While this is intuitive for the finite horizon setting, what about the infinite horizon problem setting? This same backward iteration can be repeated until convergence. It will, in the limit, converge to the true value function. Note that for infinite horizon problems, the value function is not time varying.

Value Iteration

Having discussed the computation of the value function for a fixed policy, we now proceed to the policy improvement setting, in which we aim to compute the optimal policy. *Value iteration* follows closely from the policy evaluation setting. In particular, for each $\boldsymbol{x} \in \mathcal{X}$, we perform

$$J_k^*(\boldsymbol{x}) \leftarrow \min_{\boldsymbol{u} \in \mathcal{U}} \left\{ c_k(\boldsymbol{x}, \boldsymbol{u}) + \sum_{\boldsymbol{x}' \in \mathcal{X}} p(\boldsymbol{x}' \mid \boldsymbol{x}, \boldsymbol{u}) J_{k+1}^*(\boldsymbol{x}') \right\}. \tag{2.21}$$

This process is iterated backward in time, for all states following the dynamic programming process. This yields the optimal value function, but has not given us the optimal policy. This may be computed via

$$\pi_k^*(\boldsymbol{x}) \leftarrow \operatorname{argmin}_{\boldsymbol{u}\in\mathcal{U}} \left\{ c_k(\boldsymbol{x}, \boldsymbol{u}) + \sum_{\boldsymbol{x}'\in\mathcal{X}} p(\boldsymbol{x}' \mid \boldsymbol{x}, \boldsymbol{u}) J_{k+1}^*(\boldsymbol{x}') \right\}. \tag{2.22}$$

Thus while performing value iteration, we store only the value function. Given this, we can extract the policy from a one step optimization problem.

As a useful tool that will appear throughout our discussion of reinforcement learning in particular, we will define the state-action value function (or Q function as it is more typically called) as

$$Q_k^\pi(\boldsymbol{x}, \boldsymbol{u}) = c_k(\boldsymbol{x}, \boldsymbol{u}) + \sum_{\boldsymbol{x}'\in\mathcal{X}} p(\boldsymbol{x}' \mid \boldsymbol{x}, \boldsymbol{u}) J_{k+1}^\pi(\boldsymbol{x}'). \tag{2.23}$$

This represents the total expected cost associated with taking action $\boldsymbol{u}$, followed by acting according to policy π for the remainder of the problem. The Q function is extremely useful because of how it relates to the previously introduced quantities that appear throughout this section. First, note that

$$J_k^\pi(\boldsymbol{x}) = Q_k^\pi(\boldsymbol{x}, \pi(\boldsymbol{x})) \tag{2.24}$$

and

$$J_k^*(\boldsymbol{x}) = \min_{\boldsymbol{u}\in\mathcal{U}} Q_k^*(\boldsymbol{x}, \boldsymbol{u}). \tag{2.25}$$

Note also that

$$\pi_k^*(s) = \operatorname{argmin}_{\boldsymbol{u}\in\mathcal{U}} Q_k^*(\boldsymbol{x}, \boldsymbol{u}). \tag{2.26}$$

A similar approach to value iteration may be performed using the Q function. Note that (2.21) is equivalent to

$$J_k^*(\boldsymbol{x}) \leftarrow \min_{\boldsymbol{u}\in\mathcal{U}} Q_k^*(\boldsymbol{x}, \boldsymbol{u}) \tag{2.27}$$

and thus we can iterate backward in time computing the Q function for each state action pair using

$$Q_k^*(\boldsymbol{x}, \boldsymbol{u}) = c_k(\boldsymbol{x}, \boldsymbol{u}) + \sum_{\boldsymbol{x}'\in\mathcal{X}} p(\boldsymbol{x}' \mid \boldsymbol{x}, \boldsymbol{u}) \min_{\boldsymbol{u}'\in\mathcal{U}} Q_{k+1}^*(\boldsymbol{x}', \boldsymbol{u}'). \tag{2.28}$$

We will see versions of this relation later in the course that replace the expectation over $\boldsymbol{x}'$ with experience to learn Q functions from data.

Policy Iteration

Monte Carlo Policy Improvment

2.3 Continuous-Time Optimal Control

In this section, we will extend the ideas of dynamic programming to the continuous time setting. Restating the continuous time optimal control problem, we assume dynamics

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{2.29}$$

and cost

$$J(\boldsymbol{x}(0)) = c_f(\boldsymbol{x}(t_f), t_f) + \int_0^{t_f} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau) d\tau. \tag{2.30}$$

where t_f is fixed.

2.3.1 Hamilton-Jacobi-Bellman

As in the discrete time principle of optimality, consider the tail problem

$$J(\boldsymbol{x}(t), \{\boldsymbol{u}(\tau)\}_{\tau=t}^{t_f}, t) = c_f(\boldsymbol{x}(t_f), t_f) + \int_t^{t_f} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau) d\tau \tag{2.31}$$

where $t \leq t_f$ and $\boldsymbol{x}(t)$ is an admissible state value. The optimal solution to this tail problem comes from the functional minimization

$$J^*(\boldsymbol{x}(t), t) = \min_{\{\boldsymbol{u}(\tau)\}_{\tau=t}^{t_f}} \left\{ c_f(\boldsymbol{x}(t_f), t_f) + \int_t^{t_f} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau) d\tau \right\}. \tag{2.32}$$

Note, then, that due to the additivity of cost we can split the problem up over time,

$$J^*(\boldsymbol{x}(t), t) = \min_{\{\boldsymbol{u}(\tau)\}_{\tau=t}^{t_f}} \left\{ \int_t^{t+\Delta t} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau) d\tau + c_f(\boldsymbol{x}(t_f), t_f) + \int_{t+\Delta t}^{t_f} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau) d\tau \right\} \tag{2.33}$$

which by applying the principle of optimality to the tail cost,

$$J^*(\boldsymbol{x}(t), t) = \min_{\{\boldsymbol{u}(\tau)\}_{\tau=t}^{t+\Delta t}} \left\{ \int_t^{t+\Delta t} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau) d\tau + J^*(\boldsymbol{x}(t+\Delta t), t+\Delta t) \right\}. \tag{2.34}$$

Let $J_t^*(\boldsymbol{x}(t), t) = \nabla_t J^*(\boldsymbol{x}(t), t)$ and $J_{\boldsymbol{x}}^*(\boldsymbol{x}(t), t) = \nabla_{\boldsymbol{x}} J^*(\boldsymbol{x}(t), t)$. Taylor expanding, we have

$$\begin{aligned} J^*(\boldsymbol{x}(t), t) = \min_{\{\boldsymbol{u}(\tau)\}_{\tau=t}^{t+\Delta t}} \{ & c(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\Delta t + J^*(\boldsymbol{x}(t), t) + (J_t^*(\boldsymbol{x}(t), t))\Delta t \\ & + (J_{\boldsymbol{x}}^*(\boldsymbol{x}(t), t))^T (\boldsymbol{x}(t+\Delta t) - \boldsymbol{x}(t)) + o(\Delta t) \} \end{aligned} \tag{2.35}$$

for small Δt. The first term is a result of Taylor expanding the integral and applying the fundamental theorem of calculus. Note that we can pull $J^*(\boldsymbol{x}(t), t)$ out of the minimization over cost, as this quantity will not vary under different choices of future actions. Dividing through by Δt and taking the limit $\Delta t \to 0$, we obtain the *Hamilton-Jacobi-Bellman* equation

$$0 = J_t^*(\boldsymbol{x}(t), t) + \min_{\boldsymbol{u}(t)} \left\{ c(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + (J_{\boldsymbol{x}}^*(\boldsymbol{x}(t), t))^T f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \right\} \tag{2.36}$$

with terminal condition

$$J^*(\boldsymbol{x}(t_f), t_f) = c_f(\boldsymbol{x}(t_f), t_f). \tag{2.37}$$

For convenience, we will define the Hamiltonian

$$\mathcal{H}(\boldsymbol{x}(t), \boldsymbol{u}(t), J^*_{\boldsymbol{x}}, t) := c(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + (J^*_{\boldsymbol{x}}(\boldsymbol{x}(t), t))^T f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{2.38}$$

which allow us to compactly write the HJB equation as

$$0 = J^*_t(\boldsymbol{x}(t), t) + \min_{\boldsymbol{u}(t)} \left\{\mathcal{H}(\boldsymbol{x}(t), \boldsymbol{u}(t), J^*_{\boldsymbol{x}}, t)\right\}. \tag{2.39}$$

The HJB equation is a partial differential equation that, for cost-to-go $J^*(\boldsymbol{x}(t), t)$, will satisfy all time-state pairs $(\boldsymbol{x}(t), t)$. The previous informal derivation assumed differentiability of $J^*(\boldsymbol{x}(t), t)$, which we do not know a priori. This assumption is rectified by the following theorem on solutions to the HJB equation.

Theorem 2.3.1 (Sufficiency Theorem)**.** *Suppose $V(\boldsymbol{x}, t)$ is a solution to the HJB equation, that $V(\boldsymbol{x}, t)$ is C^1 in $\boldsymbol{x}$ and t, and that*

$$\begin{aligned} 0 &= V_t(\boldsymbol{x}, t) + \min_{\boldsymbol{u} \in \mathcal{U}} \left\{c(\boldsymbol{x}, \boldsymbol{u}, t) + (V_{\boldsymbol{x}}(\boldsymbol{x}, t))^T f(\boldsymbol{x}, \boldsymbol{u}, t)\right\} \\ V(\boldsymbol{x}, t_f) &= c_f(\boldsymbol{x}, t_f) \; \forall \boldsymbol{x} \end{aligned}$$

Suppose also that $\pi^*(\boldsymbol{x}, t)$ attains the minimum in this equation for all t and $\boldsymbol{x}$. Let $\{\boldsymbol{x}^*(t) \mid t \in [t_0, t_f]\}$ be the state trajectory obtained from the given initial condition $\boldsymbol{x}(0)$ when the control trajectory $\boldsymbol{u}^*(t) = \pi^*(\boldsymbol{x}^*(t), t), t \in [t_0, t_f]$ is used. Then V is equal to the optimal cost-to-go function, i.e.,

$$V(\boldsymbol{x}, t) = J^*(\boldsymbol{x}, t) \; \forall \boldsymbol{x}, t. \tag{2.40}$$

Furthermore, the control trajectory $\{\boldsymbol{u}^*(t) \mid t \in [t_0, t_f]\}$ is optimal..

Proof. [Ber12], Volume 1, Section 7.2. □

2.3.2 Differential Games

We have so far addressed the case in which we aim to solve the optimal control problem for a single agent. We will now consider an adversarial game setting, in which there exists another player that aims to maximally harm the first agent. In particular, we will consider zero sum games in which the second agent aims to maximize the cost of the first agent. While the differential game setting is not restricted to this case — agents may have separate cost functions that partially interfere or aid each other — the zero-sum case lends itself to useful analytical tools.

Differential Games and Information Patterns

We consider the two player differential game with dynamics

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{d}(t)) \tag{2.41}$$

where the first player takes action $\boldsymbol{u}(t)$ at time t, and the second player takes action $\boldsymbol{d}(t)$. The state $\boldsymbol{x}(t)$ is the joint state of both players. We write the cost as

$$J(\boldsymbol{x}(t)) = c_f(\boldsymbol{x}(0)) + \int_t^0 c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{d}(\tau)) d\tau \tag{2.42}$$

which the first agent aims to maximize, and the second agent aims to minimize.

To fully specify the differential game, we must specify what each agent knows, and when. This is referred to as the *information pattern* of the game. In addition to capturing the knowledge of the state available to each agent, the information pattern also captures the knowledge of each other agents' strategies available to each agent.

Hamilton-Jacobi-Isaacs

The key idea in building the multi-agent equivalent of the HJB equation will again be to apply the principle of optimality. We consider the information pattern in which the adversary has access to the instantaneous control action of the first agent, so the cost takes the form

$$J(\boldsymbol{x}(t), t) = \min_{\Gamma(\boldsymbol{u})(\cdot)} \max_{\boldsymbol{u}(\cdot)} \left\{ \int_t^0 c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{d}(\tau)) d\tau + c_f(\boldsymbol{x}(0)) \right\}. \tag{2.43}$$

Applying the dynamic programming principle, we have

$$J(\boldsymbol{x}(t), t) = \min_{\Gamma(\boldsymbol{u})(\cdot)} \max_{\boldsymbol{u}(\cdot)} \left\{ \int_t^{t+\Delta t} c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{d}(\tau)) d\tau + J(\boldsymbol{x}(t+\Delta t), t+\Delta t) \right\}. \tag{2.44}$$

We can take the same strategy as with the informal derivation of the HJB equation, and Taylor expand both terms to yield

$$\begin{aligned} J(\boldsymbol{x}(t), t) = \min_{\Gamma(\boldsymbol{u})(\cdot)} \max_{\boldsymbol{u}(\cdot)} \{ & c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{d}(\tau)) \Delta t + J(\boldsymbol{x}(t), t) \\ & + (J_{\boldsymbol{x}}(\boldsymbol{x}(t), t))^T f(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{d}(t)) \Delta t + J_t(\boldsymbol{x}(t), t) \Delta t \}. \end{aligned} \tag{2.45}$$

Note that we are optimizing over instantaneous actions, and so we optimizing over finite dimensional quantities as opposed to functions. Dividing through by Δt and removing redundant terms, we get the *Hamilton-Jacobi-Isaacs* (HJI) equation

$$0 = J_t(\boldsymbol{x}, t) + \max_{\boldsymbol{u}} \min_{\boldsymbol{d}} \left\{ c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d}) + (J_{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d}))^T f(\boldsymbol{x}, \boldsymbol{d}, \boldsymbol{u}) \right\} \tag{2.46}$$

with boundary condition

$$J(\boldsymbol{x}, 0) = c_f(\boldsymbol{x}). \tag{2.47}$$

Note that we have switched the order of the min/max.

Reachability

Differential games have applications in multi-agent modeling (both in the context of autonomous systems engineering and, e.g., economics and operations research). One concrete application in engineering is reachability analysis. In this setting, an agent aims to compute the set of states in which there exists a policy that either avoids a target set or enters a target set, subject to adversarial disturbances. The former case, in which we would like to avoid a target set, is useful for safety verification. If we are able to, even in the worst case, guarantee e.g. collision avoidance, we have guarantees on safety (subject of course to our system assumptions). The latter case is useful for task satisfaction. For example, we would like a quadrotor to reach a set of safe hovering poses, even under adversarial disturbances. Finding the backward reachable set in this case would find all states such that there exists a policy that succeeds in reaching the target set.

More concretely, the first case aims to find a set

$$\mathcal{A}(t) = \{\bar{\boldsymbol{x}} : \exists \Gamma(\boldsymbol{u})(\cdot), \forall \boldsymbol{u}(\cdot), \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d}), \boldsymbol{x}(t) = \bar{\boldsymbol{x}}, \boldsymbol{x}(0) \in \mathcal{T}\} \tag{2.48}$$

where $\mathcal{T}$ is the unsafe set which we aim to avoid. Breaking this down, $\mathcal{A}(t)$ is the set of states at time t such that there exists $\Gamma(\boldsymbol{u})$ that maps action $\boldsymbol{u}$ to a disturbance such that, following the dynamics induced by the disturbance and the action sequence, the state is in $\mathcal{T}$ at time 0 (note that we are considering $t \leq 0$).

The second case aims to find a set

$$\mathcal{R}(t) = \{\bar{\boldsymbol{x}} : \forall \Gamma(\boldsymbol{u})(\cdot), \exists \boldsymbol{u}(\cdot), \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d}), \boldsymbol{x}(t) = \bar{\boldsymbol{x}}, \boldsymbol{x}(0) \in \mathcal{T}\}, \tag{2.49}$$

where in this case $\mathcal{T}$ is the set that we wish to reach. In this setting, we wish to find all states that, no matter what strategy the disturbance takes, there exist control actions that can steer the system to the goal state. Because the disturbance is adversarial (we reason over all adversary strategies), this is an extremely conservative form of safety analysis.

Computation of the backward reachable set results from solving a differential *game of kind* in which the outcome is Boolean (i.e. whether or not $\boldsymbol{x}(0) \in \mathcal{T}$). This boolean outcome can be encoded by removing the running cost and choosing a particular form for the final cost. In particular, we can choose a final cost where

$$\boldsymbol{x} \in \mathcal{T} \iff c_f(\boldsymbol{x}) \leq 0. \tag{2.50}$$

As a result, the agent should aim to maximize c_f to avoid $\mathcal{T}$, whereas the disturbance should aim to minimize it. The two settings then take the following forms:

- Set avoidance: $J(\boldsymbol{x}, t) = \min_{\Gamma(\boldsymbol{u})} \max_{\boldsymbol{u}} c_f(\boldsymbol{x}(0))$
- Set reaching: $J(\boldsymbol{x}, t) = \max_{\Gamma(\boldsymbol{u})} \min_{\boldsymbol{u}} c_f(\boldsymbol{x}(0))$

Sets vs. Tubes. We have so far considered avoidance or reachability problems for which we care about set membership at time $t = 0$. However, for something like collision avoidance, we would like to stay collision free at every time as opposed to a particular time. *Backward reachable sets* capture the case in which only the final time set membership matters, and states for times $t < 0$ do not matter. *Backward reachable tubes* capture the entire time duration of the problem. Any state that passes through the target at any time in the problem duration is included. This yields a modified value function of the form

$$J(\boldsymbol{x}, t) = \min_{\Gamma(\boldsymbol{u})} \max_{\boldsymbol{u}} \min_{\tau \in [t,0]} c_f(\boldsymbol{x}(\tau)). \tag{2.51}$$

If the target set membership holds at any time τ', then $\min_{\tau \in [t,0]} c_f(\boldsymbol{x}(\tau)) \leq c_f(\boldsymbol{x}(\tau')) \leq 0$.

2.4 Further Reading

Our coverage of reachability analysis is based on the [MBT05], which is an important early work in the field, in addition to being a relatively comprehensive coverage of the method. For a review of differential games with a (slight) emphasis on economics and management science, we refer the reader to [Bre10]. For a review of HJB and continuous time LQR, we refer the reader to [Ber12] and [Kir12].

Chapter 3

Linear Quadratic Optimal Control

In this section we will address an important subclass of continuous state and action space problems for which dynamic programming can be applied exactly. In this setting, we assume linear dynamics and quadratic costs, and the problem setting is referred to as the *linear quadratic regulator* (LQR) problem. This LQR setting is important for several reasons. First, as a local stabilizing controller, it is a core tool that is often a first (effective) approach for a wide variety of problems. Setting, as we build up open-loop trajectory optimization methods later in the class, the LQR approach will often be used to provide local tracking of these trajectories. Finally, tracking LQR paired with a forward rollout step will form the basis of the first (and one of the most effective) nonlinear trajectory optimization methods that we will see in this class.

We will discuss the LQR setting in both continuous and discrete time. Additionally, we will further our discussion on the incomplete state estimation case in the linear quadratic setting, in which the dynamics and observation function are linear and the cost is quadratic. This results in the so-called *linear quadratic Gaussian* (LQG) setting, which is an important example of the *separation* principle, in which state observers and feedback controllers can be designed independently. As a consequence, the LQG approach forms the foundation of many algorithms in incomplete state information optimal control.

3.1 Discrete Linear Quadratic Regulator

We will fix the dynamics of the system to be discrete time (possibly time-varying) linear,

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k \tag{3.1}$$

and the cost function as quadratic

$$c(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}(\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \boldsymbol{u}_k^T R_k \boldsymbol{u}_k) \tag{3.2}$$

$$c_N(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_N \boldsymbol{x}_k \tag{3.3}$$

where $Q_k \in \mathbb{R}^{n\times n}$ is positive semi-definite and $R_k \in \mathbb{R}^{m\times m}$ is positive definite for all $k = 0,\ldots,N$. Importantly, we assume $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$ are unconstrained for all k. To perform DP recursion, we initialize

$$J_N^*(\boldsymbol{x}_N) = \frac{1}{2}\boldsymbol{x}_N^T Q_N \boldsymbol{x}_N := \frac{1}{2}\boldsymbol{x}_N^T V_N \boldsymbol{x}_N. \tag{3.4}$$

Then, applying (2.12), we have

$$J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m}\left\{\boldsymbol{x}_{N-1}^T Q_{N-1}\boldsymbol{x}_{N-1} + \boldsymbol{u}_{N-1}^T R_{N-1}\boldsymbol{u}_{N-1} + \boldsymbol{x}_N^T V_N \boldsymbol{x}_N\right\} \tag{3.5}$$

which, applying the dynamics,

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m}\{&\boldsymbol{x}_{N-1}^T Q_{N-1}\boldsymbol{x}_{N-1} + \boldsymbol{u}_{N-1}^T R_{N-1}\boldsymbol{u}_{N-1} \\ &+ (A_{N-1}\boldsymbol{x}_{N-1} + B_{N-1}\boldsymbol{u}_{N-1})^T V_N (A_{N-1}\boldsymbol{x}_{N-1} + B_{N-1}\boldsymbol{u}_{N-1})\}. \end{aligned} \tag{3.6}$$

Rearranging, we have

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m}\{&\boldsymbol{x}_{N-1}^T (Q_{N-1} + A_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \\ &+ \boldsymbol{u}_{N-1}^T (R_{N-1} + B_{N-1}^T V_N B_{N-1})\boldsymbol{u}_{N-1} \\ &+ 2\boldsymbol{u}_{N-1}^T (B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1}\}. \end{aligned} \tag{3.7}$$

Note that this optimization problem is convex in $\boldsymbol{u}_{N-1}$ as $R_{N-1} + B_{N-1}^T V_N B_{N-1} > 0$. Therefore, any local minima is a global minima, and therefore we can simply apply the first order optimality conditions. Differentiating,

$$\frac{\partial J_{N-1}^*}{\partial \boldsymbol{u}_{N-1}}(\boldsymbol{x}_{N-1}) = (R_{N-1} + B_{N-1}^T V_N B_{N-1})\boldsymbol{u}_{N-1} + (B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \tag{3.8}$$

and setting this to zero yields

$$\boldsymbol{u}_{N-1}^* = -(R_{N-1} + B_{N-1}^T V_N B_{N-1})^{-1}(B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \tag{3.9}$$

which we write

$$\boldsymbol{u}_{N-1}^* = L_{N-1}\boldsymbol{x}_{N-1} \tag{3.10}$$

which is a time-varying linear feedback policy. Plugging this feedback policy into (3.6),

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) =&\boldsymbol{x}_{N-1}^T (Q_{N-1} + L_{N-1}^T R_{N-1} L_{N-1} \\ &+ (A_{N-1} + B_{N-1}L_{N-1})^T V_N (A_{N-1} + B_{N-1}L_{N-1}))\boldsymbol{x}_{N-1}. \end{aligned} \tag{3.11}$$

Critically, this implies that the cost-to-go is always a positive semi-definite quadratic function of the state. Because the optimal policy is always linear, and the optimal cost-to-go is always quadratic, the DP recursion may be recursively performed backward in time and the minimization may be performed analytically.

Following the same procedure, we can write the DP recursion for the discrete-time LQR controller:

1. $V_N = Q_N$
2. $L_k = -(R_k + B_k^T V_{k+1} B_k)^{-1}(B_k^T V_{k+1} A_k)$
3. $V_k = Q_k + L_k^T R_k L_k + (A_k + B_k L_k)^T V_{k+1}(A_k + B_k L_k)$
4. $\boldsymbol{u}_k^* = L_k \boldsymbol{x}_k$
5. $J_k^*(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T V_k \boldsymbol{x}_k$

There are several implications of this recurrence relation. First, even if A, B, Q, R are all constant (not time-varying), the policy is still time-varying. Why is this the case? Control effort invested early in the problem will yield dividends over the remaining length of the horizon, in terms of lower state cost for all future time steps. However, as the remaining length of the episode becomes shorter, this tradeoff is increasingly imbalanced, and the control effort will decrease. However, for a linear time-invariant system, if (A, B) is controllable, the feedback gain L_k approach a constant as the episode length approaches infinity. This time-invariant policy is practical for long horizon control problems, and may be approximately computed by running the DP recurrence relation until approximate convergence.

3.1.1 LQR with (Bi)linear Cost and Affine Dynamics

In the previous subsection, we have derived the common formulation of the LQR controller. In this subsection, we will derive the discrete time LQR controller for a more general system with bilinear/linear terms in the cost and affine terms in the dynamics. This derivation will be the basis of algorithms we will build up in the following subsections. More concretely, we consider systems with stage-wise cost

$$c(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T R_k \boldsymbol{u}_k + \boldsymbol{u}_k^T H_k \boldsymbol{x}_k + \boldsymbol{q}_k^T \boldsymbol{x}_k + \boldsymbol{r}_k^T \boldsymbol{u}_k + q_k, \tag{3.12}$$

terminal cost

$$c_N(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_N \boldsymbol{x}_k + \boldsymbol{q}_N^T \boldsymbol{x}_k + q_N, \tag{3.13}$$

and dynamics

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + d_k. \tag{3.14}$$

The cost-to-go will take the form

$$J_k(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T V_k \boldsymbol{x}_k + \boldsymbol{v}_k^T \boldsymbol{x}_k + v_k. \tag{3.15}$$

Repeating our approach from the last subsection, we have

$$\begin{aligned} J_k^*(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathbb{R}^m} \{ &\frac{1}{2}\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T R_k \boldsymbol{u}_k + \boldsymbol{u}_k^T H_k \boldsymbol{x}_k + \boldsymbol{q}_k^T \boldsymbol{x}_k + \boldsymbol{r}_k^T \boldsymbol{u}_k + q_k \\ &+ \frac{1}{2}(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k)^T V_{k+1}(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k) \\ &+ \boldsymbol{v}_{k+1}^T (A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_{\boldsymbol{k}}) + v_{k+1} \}. \end{aligned} \tag{3.16}$$

Rearranging, we have

$$\begin{aligned} J_k^*(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathbb{R}^m} \{ & \frac{1}{2}\boldsymbol{x}_k^T(Q_k + A_k^T V_{k+1} A_k)\boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T(R_k + B_k^T V_{k+1} B_k)\boldsymbol{u}_k \\ & + \boldsymbol{u}_k^T(H_k + B_k^T V_{k+1} A_k)^T \boldsymbol{x}_k + (\boldsymbol{q}_k + A_k^T V_{K+1}\boldsymbol{d}_k + A_k^T \boldsymbol{v}_{k+1})^T \boldsymbol{x}_k \\ & + (\boldsymbol{r}_k + B_k^T V_{k+1}\boldsymbol{d}_k + B_k^T \boldsymbol{v}_{k+1})\boldsymbol{u}_k + (v_{k+1} + \frac{1}{2}\boldsymbol{d}_k^T V_{k+1}\boldsymbol{d}_k + \boldsymbol{v}_{k+1}^T \boldsymbol{d}_k)\}. \end{aligned} \tag{3.17}$$

Solving this minimization problem, we see that our optimal controller takes the form

$$\boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \boldsymbol{x}_k. \tag{3.18}$$

We will define the following useful terms which will be used throughout the remainder of this section

$$S_{\boldsymbol{u},k} = \boldsymbol{r}_k + \boldsymbol{v}_{k+1}^T B_k + \boldsymbol{d}_k^T V_{k+1} B_k \tag{3.19}$$

$$S_{\boldsymbol{uu},k} = R_k + B_k^T V_{k+1} B_k \tag{3.20}$$

$$S_{\boldsymbol{ux},k} = H_k + B_k^T V_{k+1} A_k. \tag{3.21}$$

Given this notation, all necessary terms can be computed via the following relations

1. $V_N = Q_N$; $\boldsymbol{v}_N = \boldsymbol{q}_N$; $v_N = q_N$

2.
$$L_k = -S_{\boldsymbol{uu},k}^{-1} S_{\boldsymbol{ux},k} \tag{3.22}$$
$$\boldsymbol{l}_k = -S_{\boldsymbol{uu},k}^{-1} S_{\boldsymbol{u},k} \tag{3.23}$$

3.
$$V_k = Q_k + A_k^T V_{k+1} A_k - L_k^T S_{\boldsymbol{uu},k} L_k \tag{3.24}$$
$$\boldsymbol{v}_k = \boldsymbol{q}_k + A_k^T(\boldsymbol{v}_{k+1} + V_{k+1}\boldsymbol{d}_k) + S_{\boldsymbol{ux},k}^T \boldsymbol{l}_k \tag{3.25}$$
$$v_k = v_{k+1} + q_k + \boldsymbol{d}_k^T \boldsymbol{v}_{k+1} + \frac{1}{2}\boldsymbol{d}_k^T V_{k+1}\boldsymbol{d}_k + \frac{1}{2}\boldsymbol{l}_k^T S_{\boldsymbol{u},k} \tag{3.26}$$

4. $\boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \boldsymbol{x}_k$

5. $J_k(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T V_k \boldsymbol{x}_k + \boldsymbol{v}_k^T \boldsymbol{x}_k + v_k$.

Note that in the following subsections (specifically in our discussion of differential dynamic programming) we will introduce more convenient (and compact) notation.

3.1.2 LQR Tracking around a Linear Trajectory

In the previous subsections we have considered the generic linear quadratic control problem, in which we want to regulate to a fixed point, and deviations from this point are penalized. In this section, we will address the case in which we want to track a pre-specified trajectory. Let us assume (for now) that we have been given a nominal trajectory of the form $(\bar{\boldsymbol{x}}_0, \ldots, \bar{\boldsymbol{x}}_N)$ and $(\bar{\boldsymbol{u}}_0, \ldots, \bar{\boldsymbol{u}}_{N-1})$. We will also assume that this trajectory satisfies our given dynamics, such that

$$\bar{\boldsymbol{x}}_{k+1} = A_k \bar{\boldsymbol{x}}_k + B_k \bar{\boldsymbol{u}}_k + \boldsymbol{d}_k,\ \forall k = 0, \ldots, N-1. \tag{3.27}$$

Then, we can rewrite our dynamics in terms of deviations from the nominal trajectory,

$$\delta \boldsymbol{x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k \tag{3.28}$$

$$\delta \boldsymbol{u}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k. \tag{3.29}$$

Rewriting, we have

$$\delta \boldsymbol{x}_{k+1} = A_k \delta \boldsymbol{x}_k + B_k \delta \boldsymbol{u}_k. \tag{3.30}$$

Thus, tracking the nominal trajectory reduces to driving the state deviation, $\delta \boldsymbol{x}_k$, to zero. Note that solving this problem requires rewriting the original cost function in terms of the deviations $\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k$.

3.1.3 LQR Tracking around a Nonlinear Trajectory

Despite LQR being an extremely powerful approach to optimal control, it suffers from a handful of limitations. First and foremost, it assumes the dynamics are (possibly time-varying) linear, and the cost function is quadratic. While most systems are in fact nonlinear, a typical approach to designing feedback controllers is to linearize around some operating point. This is an effective method for designing regulators, which aim to control the system to some particular state. If, in contrast, we wish to track a trajectory, we must instead linearize around this trajectory. We will assume we are given a nominal trajectory which satisfies the nonlinear dynamics, such that

$$\bar{\boldsymbol{x}}_{k+1} = f(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k),\ \forall k = 0, \ldots, N-1. \tag{3.31}$$

Given this, we can linearize our system at each timestep by Taylor expanding,

$$\boldsymbol{x}_{k+1} \approx f(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{A_k}(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \underbrace{\frac{\partial f}{\partial \boldsymbol{u}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{B_k}(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k) \tag{3.32}$$

which allows us to again rewrite the system in terms of deviations, to get

$$\delta \boldsymbol{x}_{k+1} = A_k \delta \boldsymbol{x}_k + B_k \delta \boldsymbol{u}_k \tag{3.33}$$

which is linear in $\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k$. Note that design of systems of this type often require careful design and analysis, as deviating from the nominal trajectory results in the loss of accuracy of the local model linearization.

In designing this tracking system, a second question now occurs: how do we choose our cost function? One possible option is arbitrary choice of Q and R by the system designer. This has the advantage of being easily customizable to change system behavior, and we can guarantee the necessary conditions on these matrices. A second option, if we are given some arbitrary (possibly non-quadratic) cost function c, is to locally quadratize the cost function. Writing

$$c_k := c(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{3.34}$$

$$c_{i,k} := \frac{\partial c}{\partial i}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{3.35}$$

$$c_{ij,k} := \frac{\partial^2 c}{\partial i \partial j}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{3.36}$$

we can second order Taylor expand our cost function around our nominal trajectory

$$c(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix}^T \begin{bmatrix} 2c_k & c_{\boldsymbol{x},k}^T & c_{\boldsymbol{u},k}^T \\ c_{\boldsymbol{x},k} & c_{\boldsymbol{xx},k} & c_{\boldsymbol{ux},k}^T \\ c_{\boldsymbol{u},k} & c_{\boldsymbol{ux},k} & c_{\boldsymbol{uu},k} \end{bmatrix} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix}. \tag{3.37}$$

Here $c_{\boldsymbol{xx},k}$ and $c_{\boldsymbol{uu},k}$ replace Q_k and R_k from the previous section, respectively. There are two primary concerns with this approach to choosing the cost function. First, we require the quadratic form in (3.37) to be positive semi-definite and $c_{\boldsymbol{uu},k}$ to be positive definite, for all k. Second, we have an implicit cost that we would like to stay close to the nominal trajectory to ensure our linearized model does not become inaccurate. As a result of this implicit cost, we may wish to tune the cost terms to yield tracking that is better suited to the nonlinear model that we are tracking.

3.2 Iterative LQR and Differential Dynamic Programming

3.2.1 Iterative LQR

We have addressed the case in which we wish to track a given trajectory with LQR. A natural question, now, is whether we can use LQR to improve on this nominal trajectory? Iterative LQR augments tracking LQR with a forward pass in which the nominal trajectory is updated. As a consequence, it can be used to improve trajectories and in most cases, can be used as a practical trajectory generation and control algorithm for nonlinear systems. We

Algorithm 1 iLQR

```
Require: Nominal control sequence, $(\bar{\boldsymbol{u}}_0, \ldots, \bar{\boldsymbol{u}}_{N-1})$
1: $\delta\boldsymbol{u}_k = 0$ for all $k$
2: while not converged do
   Forward pass:
3:     Compute nominal trajectory $\bar{\boldsymbol{x}}_{k+1} = f(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k + \delta\boldsymbol{u}_k)$ and set $\bar{\boldsymbol{u}}_k \leftarrow \bar{\boldsymbol{u}}_k + \delta\boldsymbol{u}_k$
   Backward pass:
4:     Compute $Q$ terms around $(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$ for all $k$ via (3.38 – 3.43)
5:     Update feedback law via (3.45 – 3.46)
6:     Update value approximation via (3.47 – 3.49)
7: end while
8: Compute control law $\pi_k(\boldsymbol{x}_k) = \bar{\boldsymbol{u}}_k + \boldsymbol{l}_k + L_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)$
9: return $\{\pi_k\}_{k=0}^{N-1}$
```

will define the following useful terms

$$Q_k = c_k + v_{k+1} \tag{3.38}$$
$$Q_{\boldsymbol{x},k} = c_{\boldsymbol{x},k} + f_{\boldsymbol{x},k}^T \boldsymbol{v}_{k+1} \tag{3.39}$$
$$Q_{\boldsymbol{u},k} = c_{\boldsymbol{u},k} + f_{\boldsymbol{u},k}^T \boldsymbol{v}_{k+1} \tag{3.40}$$
$$Q_{\boldsymbol{xx},k} = c_{\boldsymbol{xx},k} + f_{\boldsymbol{x},k}^T V_{k+1} f_{\boldsymbol{x},k} \tag{3.41}$$
$$Q_{\boldsymbol{uu},k} = c_{\boldsymbol{uu},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{u},k} \tag{3.42}$$
$$Q_{\boldsymbol{ux},k} = c_{\boldsymbol{ux},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{x},k} \tag{3.43}$$

where $f_{\boldsymbol{x},k} = A_k$ and $f_{\boldsymbol{u},k} = B_k$. In this form, the optimal control perturbation is

$$\delta\boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \delta\boldsymbol{x}_k \tag{3.44}$$

where

$$\boldsymbol{l}_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{u},k} \tag{3.45}$$
$$L_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{ux},k}. \tag{3.46}$$

Finally, the local backward recursion can be completed by updating the value function terms via

$$v_k = Q_k - \frac{1}{2}\boldsymbol{l}_k^T Q_{\boldsymbol{uu},k} \boldsymbol{l}_k \tag{3.47}$$
$$\boldsymbol{v}_k = Q_{\boldsymbol{x},k} - L_k^T Q_{\boldsymbol{uu},k} \boldsymbol{l}_k \tag{3.48}$$
$$V_k = Q_{\boldsymbol{xx},k} - L_k^T Q_{\boldsymbol{uu},k} L_k. \tag{3.49}$$

So far, we have simply derived an alternative method for performing a quadratic approximation of the DP recursion around some nominal trajectory. The iterative LQR (iLQR) algorithm differs by introducing a forward pass that updates the trajectory that is being tracked. The algorithm alternates between forward passes, in which the control policy is applied to the nonlinear dynamics, and backward passes in which the cost function and dynamics are linearized around the new nominal trajectory, and the quadratic approximation of the value, as well as the new control law, is computed. The iterative LQR algorithm is outlined in Algorithm 1. Critically, note that this algorithm returns both a nominal trajectory, in terms of the $\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k$, as well as a feedback policy that stabilizes around this trajectory.

3.2.2 Differential Dynamic Programming

Iterative LQR performs trajectory optimization by first linearizing the dynamics and quadratizing the cost function, and then performing the dynamic programming recursion to compute optimal controls. While this linearization/quadratization approach is sufficient for approximating the Bellman equation such that it may be solved analytically, an alternative approach is to directly approximate the Bellman equation. *Differential dynamic programming* (DDP) directly builds a quadratic approximation of the right hand side of the Bellman equation (as opposed to first approximating the dynamics and the cost function), which may then be solved analytically. We will first define the change in the value of J_k under a perturbation $\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k$,

$$Q(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k) := c(\bar{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta\boldsymbol{u}_k) + J_{k+1}(f(\bar{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta\boldsymbol{u}_k)). \tag{3.50}$$

Note that Q here is different from the Q matrix in Section 3.1. Using the same notation as in (3.34), we can write the quadratic expansion of (3.50) as

$$Q(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k) \approx \frac{1}{2}\begin{bmatrix} 1 \\ \delta\boldsymbol{x}_k \\ \delta\boldsymbol{u}_k \end{bmatrix}^T \begin{bmatrix} 2Q_k & Q_{\boldsymbol{x},k}^T & Q_{\boldsymbol{u},k}^T \\ Q_{\boldsymbol{x},k} & Q_{\boldsymbol{xx},k} & Q_{\boldsymbol{ux},k}^T \\ Q_{\boldsymbol{u},k} & Q_{\boldsymbol{ux},k} & Q_{\boldsymbol{uu},k} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\boldsymbol{x}_k \\ \delta\boldsymbol{u}_k \end{bmatrix} \tag{3.51}$$

where

$$Q_k = c_k + v_{k+1} \tag{3.52}$$
$$Q_{\boldsymbol{x},k} = c_{\boldsymbol{x},k} + f_{\boldsymbol{x},k}^T \boldsymbol{v}_{k+1} \tag{3.53}$$
$$Q_{\boldsymbol{u},k} = c_{\boldsymbol{u},k} + f_{\boldsymbol{u},k}^T \boldsymbol{v}_{k+1} \tag{3.54}$$
$$Q_{\boldsymbol{xx},k} = c_{\boldsymbol{xx},k} + f_{\boldsymbol{x},k}^T V_{k+1} f_{\boldsymbol{x},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{xx},k} \tag{3.55}$$
$$Q_{\boldsymbol{uu},k} = c_{\boldsymbol{uu},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{u},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{uu},k} \tag{3.56}$$
$$Q_{\boldsymbol{ux},k} = c_{\boldsymbol{ux},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{x},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{ux},k}. \tag{3.57}$$

Note that these terms differ only from iLQR via the last term in (3.55 – 3.57), which are second order approximation of the dynamics. Note that the dot notation denotes tensor contraction.

Given this, we can partially minimize this quadratic form over the control deviation,

$$\delta\boldsymbol{u}_k^* = \text{argmin}_{\delta\boldsymbol{u}} Q(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}) = \boldsymbol{l}_k + L_k \delta\boldsymbol{x}_k \tag{3.58}$$

where

$$\boldsymbol{l}_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{u},k} \tag{3.59}$$

$$L_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{ux},k}. \tag{3.60}$$

The DDP algorithm is identical to Algorithm 1, just with the alternative definitions for $Q_{\boldsymbol{xx},k}, Q_{\boldsymbol{uu},k}$ and $Q_{\boldsymbol{ux},k}$. The main philosophical difference between iLQR and DDP is that iLQR first approximates the dynamics and cost, and then solves the Bellman equation directly, whereas DDP directly approximates the Bellman equation. While DDP yields a more accurate approximation, computing the second order dynamics terms is expensive in practice. Practically, iLQR is sufficient for most applications.

3.2.3 Algorithmic Details for iLQR and DDP

Algorithm 1 leaves out several details that would be critical for implementing the algorithm. First, what convergence criteria should we use? In [TL05], the authors stop when the update to the nominal control action sequence is sufficiently small. In [LK14], the authors iterate until the cost of the trajectory (with some additional penalty terms) increases. Finally, a variety of convergence criteria are based on expected trajectory improvement, computed via line search [JM70, TET12]. In the forward pass, standard iLQR computes an updated nominal control sequence via $\bar{\boldsymbol{u}}_k \leftarrow \bar{\boldsymbol{u}}_k + \boldsymbol{l}_{\boldsymbol{k}} + L_k \delta \boldsymbol{x}_k$. Instead we can weight $\boldsymbol{l}_{\boldsymbol{k}}$ with a scalar $\alpha \in [0,1]$ for which we perform line search. This results in increased stability (as with standard line search for step size determination in nonlinear optimization) and possibly faster convergence. When α is close to zero, or alternative conditions (such as expected improvement being small) are met, we terminate. For a further discussion of this approach, we refer the reader to [TET12], which also features a discussion of step size determination in the DDP literature.

Iterative LQR and DDP rely on minimizing a second order approximation of the cost-to-go perturbation. However, we do not have any guarantees on the convexity of $Q(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k)$ for arbitrary cost functions. Note that DDP is performing a Newton step [LS92] (iLQR is performing a Newton step with an approximation of the Hessian) via decomposing the optimization problem over controls into N smaller optimization problems. As such, standard approaches from Newton methods for regularization have been applied, such as replacing $Q_{\boldsymbol{uu},k}$ with $Q_{\boldsymbol{uu},k} + \mu I$, which is convex for sufficiently large μ. Alternative approaches have been explored in [TET12, TMT14], based on regularizing the quadratic term in the approximate cost-to-go.

Both iLQR and DDP are local methods. Full dynamic programming approaches yield globally optimal feedback policies. In contrast, iLQR and DDP yield nominal trajectories and local stabilizing controllers. However, these local controllers are often sufficient for tracking the trajectory. As they are local method, choice of initial control sequence is important, and poor choice may result in poor convergence. Additionally, we have not considered constraints on either state or action in the derivation of iLQR or DDP. This is currently an active area of research [XLH17, TMT14, GB17].

3.3 Stochastic LQR and LQG

3.3.1 LQR with Additive Noise

We will first address the stochastic LQR problem. The system dynamics are

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{\omega}_k \tag{3.61}$$

where $\boldsymbol{\omega}_k \sim \mathcal{N}(0, \Sigma_\omega)$, and the stage-wise cost is

$$c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}(\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \boldsymbol{u}_k^T R_k \boldsymbol{u}_k). \tag{3.62}$$

with terminal cost $\frac{1}{2}\boldsymbol{x}_N^T Q_N \boldsymbol{x}_N$. We wish to minimize the expected cost. The cost-to-go, as in the deterministic case, will be quadratic. Thus, plugging into the Bellman equation, we have

$$\begin{aligned}
J_k^*(\boldsymbol{x}_k) &= \min_{\boldsymbol{u}_k \in \mathbb{R}^m} \mathbb{E}[\frac{1}{2}\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T R_k \boldsymbol{u}_k \qquad (3.63)\\
&+ \frac{1}{2}(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{\omega}_k)^T V_{k+1} (A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{\omega}_k)]\\
&= \min_{\boldsymbol{u}_k \in \mathbb{R}^m} \{\frac{1}{2}\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T R_k \boldsymbol{u}_k \qquad (3.64)\\
&+ \mathbb{E}[\frac{1}{2}(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{\omega}_k)^T V_{k+1} (A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{\omega}_k)]\}.
\end{aligned}$$

Following the same minimization procedure as for LQR, we see that the policy is identical to that in Section 3.1. The Riccati equation, however, is

$$\begin{aligned}
V_k &= Q_k + L_k^T R_k L_k + \mathbb{E}[(A_k + B_k L_k + \boldsymbol{\omega}_k)^T V_{k+1} (A_k + B_k L_k + \boldsymbol{\omega}_k)] \qquad (3.65)\\
&= Q_k + L_k^T R_k L_k + (A_k + B_k L_k)^T V_{k+1} (A_k + B_k L_k) + \mathrm{tr}(\Sigma_\omega V_{k+1}) \qquad (3.66)
\end{aligned}$$

where $\mathrm{tr}(\cdot)$ denotes the trace. The equality between (3.65) and (3.66) holds as

$$\mathbb{E}[(A_k + B_k L_k)^T V_{k+1} \boldsymbol{\omega}_k] = 0 \tag{3.67}$$

for zero-mean $\boldsymbol{\omega}_k$, and $\mathbb{E}[\boldsymbol{\omega}_k^T V_{k+1} \boldsymbol{\omega}_k] = \mathrm{tr}(\Sigma_\omega V_{k+1})$. Note that this is identical to the deterministic case, other than the additive trace term at the end.

3.3.2 Problems with Imperfect State Information

We now consider the case in which direct, perfect state information isn't available. Instead, we have a noise-corrupted measurements

$$\begin{aligned}
\boldsymbol{y}_0 &= h_0(\boldsymbol{x}_0, \boldsymbol{\nu}_0) \qquad (3.68)\\
\boldsymbol{y}_k &= h(\boldsymbol{x}_k, \boldsymbol{\nu}_k),\ k = 0, \dots, N-1. \qquad (3.69)
\end{aligned}$$

The observation disturbance is characterized by distribution

$$p(\cdot \mid \boldsymbol{x}_k, \dots, \boldsymbol{x}_0, \boldsymbol{u}_{k-1}, \dots, \boldsymbol{u}_0, \boldsymbol{\omega}_{k-1}, \dots, \boldsymbol{\omega}_0, \boldsymbol{\nu}_{k-1}, \dots, \boldsymbol{\nu}_0) \tag{3.70}$$

and the initial state $\boldsymbol{x}_0$ is distributed according to $p(\boldsymbol{x}_0)$. We will define the information vector as

$$\boldsymbol{i}_k = [\boldsymbol{y}_0^T, \dots, \boldsymbol{y}_k^T, \boldsymbol{u}_0^T, \dots, \boldsymbol{u}_{k-1}^T]^T. \tag{3.71}$$

Armed with this, we will consider *admissible* policies $\pi(\boldsymbol{i}_k) \in \mathcal{U}_k$, which implies they are *causal* — they do not rely on information only available in the future. The goal of the control problem, then is to minimize

$$\mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{\omega}_{0:N-1}, \boldsymbol{\nu}_{0:N-1}} \left[c_T(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} c(\boldsymbol{x}_k, \pi(\boldsymbol{i}_k), \boldsymbol{\omega}_k) \right]. \tag{3.72}$$

Treating this problem as a perfect state information problem and attempting to solve directly results in several problems. In addition to the standard difficulties associated with applying DP to generic problems, $\boldsymbol{i}_k$ has expanding dimension over the length of the problem. Alternatively, we may reason in terms of sufficient statistics: quantities that summarize all of the informational content of $\boldsymbol{i}_k$. For example, if we can construct a conditional distribution over state, $p(\boldsymbol{x}_k \mid \boldsymbol{i}_k)$, we can design a policy of the form $\pi_k(p(\boldsymbol{x}_k \mid \boldsymbol{i}_k))$.

3.3.3 LQG and the Separation Principle

Given the generic imperfect state information control problem, we will address an extremely important special case. We will again consider quadratic cost of the form

$$\frac{1}{2}\mathbb{E}\left[\boldsymbol{x}_N^T Q_N \boldsymbol{x}_N + \sum_{k=0}^{N-1} \boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \boldsymbol{u}_k^T R_k \boldsymbol{u}_k \right] \tag{3.73}$$

subject to dynamics

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{\omega}_k \tag{3.74}$$

and measurements

$$\boldsymbol{y}_k = C_k \boldsymbol{x}_k + \boldsymbol{\nu}_k. \tag{3.75}$$

The initial state $\boldsymbol{x}_0$, and process and measurements noise $\boldsymbol{\omega}_{0:N-1}, \boldsymbol{\nu}_{0:N-1}$ independent, zero-mean Gaussians. We will write the covariance of $\boldsymbol{\omega}_k$ and $\boldsymbol{\nu}_k$ as $\Sigma_{\boldsymbol{\omega},k}$ and $\Sigma_{\boldsymbol{\nu},k}$, respectively. We will write $\Sigma_{\boldsymbol{x},0}$ for the covariance of $\boldsymbol{x}_0$.

We will skip the lengthy derivation, but the optimal control policy takes the form

$$\boldsymbol{u}_k^* = L_k \hat{\boldsymbol{x}}_k \tag{3.76}$$

where $\hat{\boldsymbol{x}}_k$ is the state estimate from the Kalman filter, and L_k is the standard gain from LQR. Note that the Kalman filter maintains a Gaussian estimate of the state, and so the sufficient statistics are the mean and variance. However, the policy depends only on the mean. The policy can be designed as if access to perfect state information is available, while the estimator can be designed without considering the controller. This is known as the *separation principle.*

3.4 Continuous-Time LQR

As a useful result of the HJB equations, we will derive LQR in continuous time. We aim to minimize

$$J(\boldsymbol{x}(0)) = \frac{1}{2}\boldsymbol{x}^T(t_f)Q_f\boldsymbol{x}(t_f) + \frac{1}{2}\int_0^{t_f} \boldsymbol{x}^T(t)Q(t)\boldsymbol{x}(t) + \boldsymbol{u}^T(t)R(t)\boldsymbol{u}(t)dt \tag{3.77}$$

subject to dynamics

$$\dot{\boldsymbol{x}}(t) = A(t)\boldsymbol{x}(t) + B(t)\boldsymbol{u}(t). \tag{3.78}$$

As in discrete LQR, we will assume $Q_f, Q(t)$ are positive semidefinite, and $R(t)$ is positive definite. We will also assume t_f is fixed, and the state and action are unconstrained.

We will write the Hamiltonian,

$$\mathcal{H} = \frac{1}{2}\boldsymbol{x}^T(t)Q(t)\boldsymbol{x}(t) + \frac{1}{2}\boldsymbol{u}^T(t)R(t)\boldsymbol{u}(t) + J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)^T(A(t)\boldsymbol{x}(t) + B(t)\boldsymbol{u}(t)) \tag{3.79}$$

which yields necessary optimality conditions

$$0 = \nabla_{\boldsymbol{u}}\mathcal{H} = R(t)\boldsymbol{u}(t) + B^T(t)J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t). \tag{3.80}$$

Since $\nabla^2_{\boldsymbol{uu}}\mathcal{H} = R(t) > 0$, the control that satisfies the necessary conditions is the global minimizer. Rearranging, we have

$$\boldsymbol{u}^*(t) = -R^{-1}(t)B^T(t)J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t) \tag{3.81}$$

which we can plug back into the Hamiltonian to yield

$$\begin{aligned}\mathcal{H} &= \frac{1}{2}\boldsymbol{x}^T(t)Q(t)\boldsymbol{x}(t) + \frac{1}{2}J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)^TB(t)R^{-1}(t)B^T(t)J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t) \qquad (3.82)\\ &\quad + J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)^TA(t)\boldsymbol{x}(t) - J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)^TB(t)R^{-1}(t)B^T(t)J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)\\ &= \frac{1}{2}\boldsymbol{x}^T(t)Q(t)\boldsymbol{x}(t) - \frac{1}{2}J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)^TB(t)R^{-1}(t)B^T(t)J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t) + J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)^TA(t)\boldsymbol{x}(t). \qquad (3.83)\end{aligned}$$

This gives the HJB equation

$$\begin{aligned}0 = J^*_t(\boldsymbol{x}(t),t) + \frac{1}{2}\boldsymbol{x}^T(t)Q(t)\boldsymbol{x}(t) - \frac{1}{2}J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)^TB(t)R^{-1}(t)B^T(t)J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t) \qquad (3.84)\\ + J^*_{\boldsymbol{x}}(\boldsymbol{x}(t),t)^TA(t)\boldsymbol{x}(t)\end{aligned}$$

with boundary condition

$$J^*(\boldsymbol{x}(t_f),t_f) = \frac{1}{2}\boldsymbol{x}^T(t_f)Q_f\boldsymbol{x}(t_f). \tag{3.85}$$

It may appear as if we are stuck here, as this form of the HJB doesn't immediately yield $J^*(\boldsymbol{x}(t),t)$. Armed with the knowledge that the discrete time LQR problem has a quadratic cost-to-go, we will cross our fingers and guess a solution of the form

$$J^*(\boldsymbol{x}(t),t) = \frac{1}{2}\boldsymbol{x}^T(t)V(t)\boldsymbol{x}(t). \tag{3.86}$$

Substituting, we have

$$\begin{aligned}0 = &\frac{1}{2}\boldsymbol{x}^T(t)\dot{V}(t)\boldsymbol{x}(t) + \frac{1}{2}\boldsymbol{x}^T(t)Q(t)\boldsymbol{x}(t) \\ &- \frac{1}{2}\boldsymbol{x}^T(t)V(t)B(t)R^{-1}(t)B^T(t)V(t)\boldsymbol{x}(t) + \boldsymbol{x}^T(t)V(t)A(t)\boldsymbol{x}(t)\end{aligned} \tag{3.87}$$

Note that we will decompose

$$\boldsymbol{x}^T(t)V(t)A(t)\boldsymbol{x}(t) = \frac{1}{2}\boldsymbol{x}^T(t)V(t)A(t)\boldsymbol{x}(t) + \frac{1}{2}\boldsymbol{x}^T(t)A^T(t)V(t)\boldsymbol{x}(t) \tag{3.88}$$

which yields

$$0 = \frac{1}{2}\boldsymbol{x}^T(t)\left(\dot{V}(t) + Q(t) - V(t)B(t)R^{-1}(t)B^T(t)V(t) + V(t)A(t) + A^T(t)V(t)\right)\boldsymbol{x}(t). \tag{3.89}$$

This equation must hold for all $\boldsymbol{x}(t)$, so

$$-\dot{V}(t) = Q(t) - V(t)B(t)R^{-1}(t)B^T(t)V(t) + V(t)A(t) + A^T(t)V(t) \tag{3.90}$$

with boundary condition $V(t_f) = Q_f$.

Therefore, the HJB PDE has been reduced to a set of matrix ordinary differential equations (the Riccati equation). This is integrated backwards in time to find the full control policy as a function of time. One we have found $V(t)$, the control policy is

$$\boldsymbol{u}^*(t) = -R^{-1}(t)B^T(t)V(t)\boldsymbol{x}(t). \tag{3.91}$$

Similarly to the discrete case, the feedback gains tend toward constant in the limit of the infinite horizon problem, under some technical assumptions.

3.5 Further Reading

A comprehensive coverage of linear quadratic methods for optimal control is Anderson and Moore [AM07]. LQG is covered in discrete time in [Ber12]. The original, comprehensive reference on DDP is [JM70], but a large body of literature on the method has been produced since then. The original papers on iLQR are [TL05, LT04].

Chapter 4

Indirect Methods

4.1 Calculus of Variations

We will begin by restating the optimal control problem. We will to find an admissible control sequence $\boldsymbol{u}^*$ which causes the system

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{4.1}$$

to follow an *admissible* trajectory $\boldsymbol{x}^*$ that minimizes the functional

$$J = c_f(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt. \tag{4.2}$$

To find the minima of functions of a finite number of real numbers, we rely on the first order optimality conditions to find candidate minima, and use higher order derivatives to determine whether a point is a local minimum. Because we are minimizing a function that maps from some n dimensional space to a scalar, candidate points have zero gradient in each of these dimensions. However, in the optimal control problem, we have a cost *functional*, which maps functions to scalars. This is immediately problematic for our first order conditions — we are required to check the necessary condition at infinite points. The necessary notion of optimality conditions for functionals is provided by calculus of variations.

Concretely, we define a functional J as a rule of correspondence assigning each function $\boldsymbol{x}$ in a class Ω (the domain) to a unique real number. The functional J is linear if and only if

$$J(\alpha_1\boldsymbol{x}_1 + \alpha_2\boldsymbol{x}_2) = \alpha_1 J(\boldsymbol{x}_1) + \alpha_2 J(\boldsymbol{x}_2) \tag{4.3}$$

for all $\boldsymbol{x}_1, \boldsymbol{x}_2, \alpha_1\boldsymbol{x}_1 + \alpha_2\boldsymbol{x}_2$ in Ω. We must now define a notion of "closeness" for functions. Intuitively, two points being close together has an immediate geometric interpretation. We first define the norm of a function. The norm of a function is a rule of correspondence that assigns each $\boldsymbol{x} \in \Omega$, defined over $t \in [t_0, t_f]$, a real number. The norm of $\boldsymbol{x}$, which we denote $\|\boldsymbol{x}\|$, satisfies:

1. $\|\boldsymbol{x}\| \geq 0$, and $\|\boldsymbol{x}\| = 0$ iff $\boldsymbol{x}(t) = 0$ for all $t \in [t_0, t_f]$

2. $\|\alpha \boldsymbol{x}\| = |\alpha|\|\boldsymbol{x}\|$ for all real numbers α

3. $\|\boldsymbol{x}_1 + \boldsymbol{x}_2\| \leq \|\boldsymbol{x}_1\| + \|\boldsymbol{x}_2\|$.

To compare the closeness of two functions $\boldsymbol{y}, \boldsymbol{z}$, we let $\boldsymbol{x}(t) = \boldsymbol{y}(t) - \boldsymbol{z}(t)$. Thus, for two identical functions, $\|\boldsymbol{x}\|$ is zero. Generally, a norm will be small for "close" functions, and large for "far apart" functions. However, there exist many possible definitions of norms that satisfy the above conditions.

4.1.1 Extrema for Functionals

A functional J with domain Ω has a local minimum at $\boldsymbol{x}^* \in \Omega$ if there exists an $\epsilon > 0$ such that $J(\boldsymbol{x}) \geq J(\boldsymbol{x}^*)$ for all $\boldsymbol{x} \in \Omega$ such that $\|\boldsymbol{x} - \boldsymbol{x}^*\| < \epsilon$. Maxima are defined similarly, just with $J(\boldsymbol{x}) \leq J(\boldsymbol{x}^*)$.

Analogously to optimization of functions, we define the variation of the functional as

$$\Delta J(\boldsymbol{x}, \delta\boldsymbol{x}) := J(\boldsymbol{x} + \delta\boldsymbol{x}) - J(\boldsymbol{x}) \tag{4.4}$$

where $\delta\boldsymbol{x}(t)$ is the *variation* of $\boldsymbol{x}(t)$. The increment of a functional can be written as

$$\Delta J(\boldsymbol{x}, \delta\boldsymbol{x}) = \delta J(\boldsymbol{x}, \delta\boldsymbol{x}) + g(\boldsymbol{x}, \delta\boldsymbol{x})\|\delta\boldsymbol{x}\| \tag{4.5}$$

where δJ is linear in $\delta\boldsymbol{x}$. If

$$\lim_{\|\delta\boldsymbol{x}\| \to 0} \{g(\boldsymbol{x}, \delta\boldsymbol{x})\} = 0 \tag{4.6}$$

then J is said to be differentiable on $\boldsymbol{x}$ and δJ is the variation of J at $\boldsymbol{x}$. We can now state the *fundamental theorem of the calculus of variations.*

Theorem 4.1.1 (Fundamental Theorem of CoV). *Let $\boldsymbol{x}(t)$ be a vector function of t in the class Ω, and $J(\boldsymbol{x})$ be a differentiable functional of $\boldsymbol{x}$. Assume that the functions in Ω are not constrained by any boundaries. If $\boldsymbol{x}^*$ is an extremal, the variation of J must vanish at $\boldsymbol{x}^*$, that is $\delta J(\boldsymbol{x}^*, \delta\boldsymbol{x}) = 0$ for all admissible $\delta\boldsymbol{x}$ (i.e. such that $\boldsymbol{x} + \delta\boldsymbol{x} \in \Omega$).*

Proof. [Kir12], Section 4.1. □

We will now look at how calculus of variations may be leveraged to approach practical problems. Let x be a scalar continuous function in C^1. We would like to find a function x^* for which the functional

$$J(s) = \int_{t_0}^{t_f} g(x(t), x(t), t)dt \tag{4.7}$$

has a relative extremum. We will assume $g \in C^2$, that t_0, t_f are fixed, and x_0, x_f are fixed. Let $\boldsymbol{x}$ be any curve in Ω, and we will write the variation δJ from the increment

$$\begin{aligned} \Delta J(x, \delta x) &= J(x + \delta x) - J(x) & (4.8)\\ &= \int_{t_0}^{t_f} g(x + \delta x, \dot{x} + \delta\dot{x}, t)dt - \int_{t_0}^{t_f} g(x, \dot{x}, t)dt & (4.9)\\ &= \int_{t_0}^{t_f} g(x + \delta x, \dot{x} + \delta\dot{x}, t) - g(x, \dot{x}, t)dt. & (4.10) \end{aligned}$$

Expanding via Taylor series, we get

$$\Delta J(x, \delta x) = \int_{t_0}^{t_f} g(x, \dot{x}, t) + \underbrace{\frac{\partial g}{\partial x}}_{g_x}(x, \dot{x}, t)\delta x + \underbrace{\frac{\partial g}{\partial \dot{x}}}_{g_{\dot{x}}}(x, \dot{x}, t)\delta \dot{x} + o(\delta x, \delta \dot{x}) - g(x, \dot{x}, t)dt \quad (4.11)$$

which yields the variation

$$\delta J = \int_{t_0}^{t_f} g_x(x, \dot{x}, t)\delta x + g_{\dot{x}}(x, \dot{x}, t)\delta \dot{x} \; dt. \quad (4.12)$$

Integrating by parts, we have

$$\delta J = \int_{t_0}^{t_f} \left[g_x(x, \dot{x}, t) - \frac{d}{dt} g_{\dot{x}}(x, \dot{x}, t) \right] \delta x \delta t + [g_{\dot{x}}(x, \dot{x}, t)\delta x(t)]_{t_0}^{t_f}. \quad (4.13)$$

We have assumed $x(t_0), x(t_f)$ given, and thus $\delta x(t_0) = 0$, $\delta x(t_f) = 0$. Considering an extremal curve, applying the CoV theorem yields

$$\int_{t_0}^{t_f} \left[g_x(x, \dot{x}, t) - \frac{d}{dt} g_{\dot{x}}(x, \dot{x}, t) \right] \delta x \delta t. \quad (4.14)$$

We can now state the fundamental lemma of CoV. We will state it for vector functions, although our derivation was for the scalar case.

Lemma 4.1.2 (Fundamental Lemma of CoV)**.** *If a function h is continuous and*

$$\int_{t_0}^{t_f} h(t)\delta \boldsymbol{x}(t)dt = 0 \quad (4.15)$$

for every function $\delta \boldsymbol{x}$ that is continuous in the interval $[t_0, t_f]$, then h must be zero everywhere in the interval $[t_0, t_f]$.

Proof. [Kir12], Section 4.2. □

Applying the fundamental lemma, we find that a necessary condition for $\boldsymbol{x}^*$ being an extremal is

$$g_{\boldsymbol{x}}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) - \frac{d}{dt} g_{\dot{\boldsymbol{x}}}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = 0 \quad (4.16)$$

for all $t \in [t_0, t_f]$, which is the *Euler equation.* This is a nonlinear, time-varying second-order ordinary differential equation with split boundary conditions (at $\boldsymbol{x}(t_0)$ and $\boldsymbol{x}(t_f)$).

4.1.2 Generalized Boundary Conditions

In the previous subsection, we assumed that $t_0, t_f, \boldsymbol{x}(t_0), \boldsymbol{x}(t_f)$ were all given. We will now relax that assumption. In particular, t_f may be fixed or free, and each component of $\boldsymbol{x}(t_f)$ may be fixed or free.

We begin by writing the variation around $\boldsymbol{x}^*$

$$\begin{aligned}\delta J = &\left[g_{\dot{\boldsymbol{x}}}(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f)\right]^T \delta\boldsymbol{x}(t_f) + \left[g(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f)\right]^T \delta t_f \\ &+ \int_{t_0}^{t_f} \left[g_{\boldsymbol{x}}(\boldsymbol{x}^*, \dot{\boldsymbol{x}}^*, t) - \frac{d}{dt} g_{\dot{\boldsymbol{x}}}(\boldsymbol{x}^*, \dot{\boldsymbol{x}}^*, t)\right]^T \delta\boldsymbol{x}\delta t\end{aligned} \tag{4.17}$$

by using the same integration by parts approach as before. Note that for fixed t_f and $\boldsymbol{x}(t_f)$, the variations δt_f and $\delta\boldsymbol{x}(t_f)$ vanish, and so we are left with (4.14). Because δt_f and $\delta\boldsymbol{x}(t_f)$ do not vanish in this case, we are left with additional boundary conditions that must be satisfied. Note that

$$\delta\boldsymbol{x}_f = \delta\boldsymbol{x}(t_f) + \dot{\boldsymbol{x}}^*(t_f)\delta t_f \tag{4.18}$$

and substituting this, we have

$$\begin{aligned}\delta J = &\left[g_{\dot{\boldsymbol{x}}}(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f)\right]^T \delta\boldsymbol{x}_f + \left[g(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f) - g_{\dot{\boldsymbol{x}}}^T(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f)\dot{\boldsymbol{x}}^*(t_f)\right] \delta t_f \\ &+ \int_{t_0}^{t_f} \left[g_{\boldsymbol{x}}(\boldsymbol{x}^*, \dot{\boldsymbol{x}}^*, t) - \frac{d}{dt} g_{\dot{\boldsymbol{x}}}(\boldsymbol{x}^*, \dot{\boldsymbol{x}}^*, t)\right] \delta\boldsymbol{x}\delta t.\end{aligned} \tag{4.19}$$

Stationarity of this variation thus requires

$$g_{\dot{\boldsymbol{x}}}(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f) = 0 \tag{4.20}$$

if $\boldsymbol{x}_f$ is free, and

$$g(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f) - g_{\dot{\boldsymbol{x}}}^T(\boldsymbol{x}^*(t_f), \dot{\boldsymbol{x}}^*(t_f), t_f)\dot{\boldsymbol{x}}^*(t_f) = 0 \tag{4.21}$$

if t_f is free, in addition to the Euler equation being satisfied. For a complete reference on the boundary conditions associated with a variety of problem specifications, we refer the reader to Section 4.3 of [Kir12].

4.1.3 Constrained Extrema

Previously, we have not considered constraints in the variational problem. However, constraints (and in particular, dynamics constraints) are central to most optimal control problems. Let $\boldsymbol{w} \in \mathbb{R}^{n+m}$ be a vector function in C^1. As previously, we would like to find a function $\boldsymbol{w}^*$ for which the functional

$$J(\boldsymbol{w}) = \int_{t_0}^{t_f} g(\boldsymbol{w}(t), \dot{\boldsymbol{w}}(t), t)dt \tag{4.22}$$

has a relative extremum, although we additionally introduce the constraints

$$f_i(\boldsymbol{w}(t), \dot{\boldsymbol{w}}(t), t) = 0, \quad i = 1, \ldots, n. \tag{4.23}$$

We will again assume $g \in C^2$ and that $t_0, \boldsymbol{w}(t_0)$ are fixed. Note that as a result of these n constraints, only m of the $n + m$ components of $\boldsymbol{w}$ are independent.

One approach to solving this constrained problem is re-writing the n dependent components of $\boldsymbol{w}$ in terms of the m independent components. However, the nonlinearity of the constraints typically makes this infeasible. Instead, we will turn to Lagrange multipliers. We will write our *augmented functional* as

$$\hat{g}(\boldsymbol{w}(t), \dot{\boldsymbol{w}}(t), \boldsymbol{p}(t), t) := g(\boldsymbol{w}(t), \dot{\boldsymbol{w}}(t), t) + \boldsymbol{p}^T(t)\boldsymbol{f}(\boldsymbol{w}(t), \dot{\boldsymbol{w}}(t), t) \tag{4.24}$$

where $\boldsymbol{p}(t)$ are Lagrange multipliers that are functions of time. Based on this, a necessary condition for optimality is

$$\hat{g}_{\boldsymbol{w}}(\boldsymbol{w}^*(t), \dot{\boldsymbol{w}}^*(t), \boldsymbol{p}^*(t), t) - \frac{d}{dt}\hat{g}_{\dot{\boldsymbol{w}}}(\boldsymbol{w}^*(t), \dot{\boldsymbol{w}}^*(t), \boldsymbol{p}^*(t), t) = 0 \tag{4.25}$$

with

$$\boldsymbol{f}(\boldsymbol{w}^*(t), \dot{\boldsymbol{w}}^*(t), t) = 0. \tag{4.26}$$

4.2 Indirect Methods for Optimal Control

Having built the foundations of functional optimization via calculus of variations, we will now derive the necessary conditions for optimal control under the assumption that the admissible controls are not bounded. The problem, as previously stated, is to find an *admissible control* $\boldsymbol{u}^*$ which causes the system

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{4.27}$$

to follow an *admissible trajectory* $\boldsymbol{x}^*$ that minimizes the functional

$$J(\boldsymbol{u}) = c_f(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \tag{4.28}$$

under the assumptions that $c_f \in C^2$, the state and control are unconstrained, and $t_0, \boldsymbol{x}(t_0)$ are fixed. We define the *Hamiltonian* as

$$\mathcal{H}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}(t), t) := c(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{p}^T(t)f(\boldsymbol{x}(t), \boldsymbol{u}(t), t). \tag{4.29}$$

Then, the necessary conditions are

$$\dot{\boldsymbol{x}}^*(t) = \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t) \tag{4.30}$$

$$\dot{\boldsymbol{p}}^*(t) = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t) \tag{4.31}$$

$$0 = \frac{\partial \mathcal{H}}{\partial \boldsymbol{u}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t) \tag{4.32}$$

which must hold for all $t \in [t_0, t_f]$. Additionally, the boundary conditions

$$[\frac{\partial c_f}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t_f), t_f) - \boldsymbol{p}^*(t_f)]^T \delta \boldsymbol{x}_f \\ + [\mathcal{H}(\boldsymbol{x}^*(t_f), \boldsymbol{u}^*(t_f), \boldsymbol{p}^*(t_f), t_f) + \frac{\partial c_f}{\partial t}(\boldsymbol{x}^*(t_f), t_f)]\delta t_f = 0 \tag{4.33}$$

must be satisfied. Note that as in the previous section, they are automatically satisfied if the terminal state and time are fixed. Based on these necessary conditions, we have a set of $2n$ *first-order* differential equations (for the state and co-state), and a set of m algebraic equations (control equations). The solution to the state and co-state equations will contain $2n$ constants of integration. To solve for these constants, we use the initial conditions $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$ (of which there are n), and an additional n (or $n+1$) equations from the boundary conditions. We are left with a two-point boundary value problem, which are considerably more difficult to solve than initial value problems which can just be integrated forward. For a full review of boundary conditions, we again refer the reader to [Kir12].

4.2.1 Proof of the Necessary Conditions

We will now prove the necessary conditions, (4.30 – 4.32), along with the boundary conditions (4.42). For simplicity, assume that the terminal cost is zero, and that $t_f, \boldsymbol{x}(t_f)$ are fixed and given. Consider the augmented cost function

$$\hat{c}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \boldsymbol{u}(t), \boldsymbol{p}(t), t) := c(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{p}^T(t)[f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) - \dot{\boldsymbol{x}}(t)]. \tag{4.34}$$

When the constraint holds, this augmented cost function is exactly equal to the original cost function. The augmented total cost is then

$$\hat{J}(\boldsymbol{u}) = \int_{t_0}^{t_f} \hat{c}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), \boldsymbol{u}(t), \boldsymbol{p}(t), t)dt. \tag{4.35}$$

Applying the fundamental theorem of CoV on an extremal, we have

$$0 = \delta \hat{J}(\boldsymbol{u}) = \int_{t_0}^{t_f} \left[\overbrace{\frac{\partial \hat{c}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t)}^{\frac{\partial c}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t) + \frac{\partial f}{\partial \boldsymbol{x}}^T(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t)\boldsymbol{p}^*(t)} - \overbrace{\frac{d}{dt}\frac{\partial \hat{c}}{\partial \dot{\boldsymbol{x}}}(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t)}^{-\frac{d}{dt}(-\boldsymbol{p}^*(t))} \right]^T \delta \boldsymbol{x}(t) \tag{4.36}$$

$$+ \left[\frac{\partial \hat{c}}{\partial \boldsymbol{u}}(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t)\right]^T \delta \boldsymbol{u}(t) + \underbrace{\left[\frac{\partial \hat{c}}{\partial \boldsymbol{p}}(\boldsymbol{x}^*(t), \dot{\boldsymbol{x}}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t)\right]^T}_{f(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t) - \dot{\boldsymbol{x}}^*(t)} \delta \boldsymbol{p}(t)dt.$$

Considering each term in sequence, we have:

- $f(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t) - \dot{\boldsymbol{x}}^*(t) = 0$ on an extremal.
- The Lagrange multipliers are arbitrary, so we can select them to make the coefficients of $\delta\boldsymbol{x}(t)$ equal to zero, giving $\dot{\boldsymbol{p}}(t) = -\frac{\partial c}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t) - \frac{\partial f}{\partial \boldsymbol{x}}^T(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t)\boldsymbol{p}^*(t)$.
- The remaining variation $\delta\boldsymbol{u}(t)$ is independent, so its coefficient must be zero, thus $\frac{\partial c}{\partial \boldsymbol{u}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t) + \frac{\partial f}{\partial \boldsymbol{u}}^T(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t)\boldsymbol{p}^*(t) = 0$.

These conditions exactly give the necessary conditions as previously stated, when recast with the Hamiltonian formalism.

4.3 Pontryagin's Minimum Principle

So far, we have assumed that the admissible controls and states are unconstrained. This assumption is frequently violated for real systems—physical actuators have limits on their realizable outputs, and state constraints may occur due to safety considerations. The control $\boldsymbol{u}^*$ causes the functional J to have a relative minimum if

$$J(\boldsymbol{u}) - J(\boldsymbol{u}^*) = \Delta J \geq 0 \tag{4.37}$$

for all admissible controls "close" to $\boldsymbol{u}^*$. Letting $\boldsymbol{u} = \boldsymbol{u}^* + \delta\boldsymbol{u}$, the increment can be expressed as

$$\Delta J(\boldsymbol{u}^*, \delta\boldsymbol{u}) = \delta J(\boldsymbol{u}^*, \delta\boldsymbol{u}) + \text{higher order terms.} \tag{4.38}$$

The variation $\delta\boldsymbol{u}$ is arbitrary only if the extremal control is strictly within the boundary for all time in the interval $[t_0, t_f]$. In general, however, an extremal control lies on a boundary during at least subinterval in the interval $[t_0, t_f]$. As a consequence, admissible control variations $\delta\boldsymbol{u}$ exist whose negatives are not admissible. This implies that a necessary condition for $\boldsymbol{u}^*$ to minimize J is $\delta J(\boldsymbol{u}^*, \delta\boldsymbol{u}) \geq 0$ for all admissible variations with $\|\delta\boldsymbol{u}\|$ small enough. The reason why the equality in the fundamental theorem of CoV (in which we explicitly assumed no constraints) is replaced with an inequality is the presence of the control constraints. This result has an analogue in calculus, where the necessary condition for a scalar function f to have a relative minimum at the end point is that the differential $df \geq 0$.

Assuming bounded controls $\boldsymbol{u} \in \mathcal{U}$, the necessary optimality conditions are

$$\dot{\boldsymbol{x}}^*(t) = \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t) \tag{4.39}$$

$$\dot{\boldsymbol{p}}^*(t) = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t) \tag{4.40}$$

$$\mathcal{H}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t) \leq \mathcal{H}(\boldsymbol{x}^*(t), \boldsymbol{u}(t), \boldsymbol{p}^*(t), t) \;\forall \boldsymbol{u} \in \mathcal{U} \tag{4.41}$$

along with the boundary conditions

$$\begin{aligned} &[\frac{\partial c_f}{\partial \boldsymbol{x}}(\boldsymbol{x}^*(t_f), t_f) - \boldsymbol{p}^*(t_f)]^T \delta\boldsymbol{x}_f \\ &\qquad + [\mathcal{H}(\boldsymbol{x}^*(t_f), \boldsymbol{u}^*(t_f), \boldsymbol{p}^*(t_f), t_f) + \frac{\partial c_f}{\partial t}(\boldsymbol{x}^*(t_f), t_f)]\delta t_f = 0. \end{aligned} \tag{4.42}$$

The control $\boldsymbol{u}^*(t)$ causes $\mathcal{H}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t), t)$ to assume its global minimum. This is a harder condition, in general, to analyze. Finally, we have additional necessary conditions. If the final time is fixed and the Hamiltonian does not explicitly depend on time,

$$\mathcal{H}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t)) = c \ \forall t \in [t_0, t_f] \tag{4.43}$$

and if the final time is free and the Hamiltonian does not depend explicitly on time,

$$\mathcal{H}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*(t)) = 0 \ \forall t \in [t_0, t_f]. \tag{4.44}$$

Note that in general, uniqueness and existence are not guaranteed in the constrained setting.

4.4 Numerical Aspects of Indirect Optimal Control

4.5 Further Reading

For a practical treatment of indirect methods, we refer the reader to [BH75]. For a more theoretical treatment, we refer the reader to [LM67].

Chapter 5

Direct Methods for Optimal Control

In the previous section we considered indirect methods to optimal control, in which the necessary conditions for optimality were first applied, yielding a two-point boundary value problem that was solved numerically. We will now consider the class of direct methods, in which the optimal control problem is first discretized, and then the resulting discrete optimization problem is solved numerically.

5.1 Direct Methods

We will write our original continuous optimal control problem,

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \int_0^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \\ \text{s.t.} \quad & \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t), t \in [0, t_f] \\ & \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ & \boldsymbol{x}(t_f) \in \mathcal{M}_f \\ & \boldsymbol{u}(t) \in \mathcal{U}, t \in [0, t_f] \end{aligned} \tag{5.1}$$

where $\mathcal{M}_f = \{\boldsymbol{x} \in \mathbb{R}^n : F(\boldsymbol{x}) = 0\}$ and where we have, for simplicity, assumed zero terminal cost and $t_0 = 0$. We will use forward Euler discretization of the dynamics. We select a discretization $0 = t_0 < t_1 < \ldots < t_N = t_f$ for the interval $[0, t_f]$, and we will write $\boldsymbol{x}_{i+1} \approx \boldsymbol{x}(t), \boldsymbol{u}_i \approx \boldsymbol{u}(t)$ for $t \in [t_i, t_{i+1}]$, and $\boldsymbol{x}_0 \approx \boldsymbol{x}(0)$. Denoting $h_i = t_{i+1} - t_i$, the continuous time optimal control problem is transcibed into the nonlinear constrained optimization problem

$$\begin{aligned} \min_{\boldsymbol{x}, \boldsymbol{u}} \quad & \sum_{i=0}^{N-1} h_i c(\boldsymbol{x}_i, \boldsymbol{u}_i, t_i) \\ \text{s.t.} \quad & \boldsymbol{x}_{i+1} = \boldsymbol{x}_i + h_i f(\boldsymbol{x}_i, \boldsymbol{u}_i, t_i), i = 0, \ldots, N-1 \\ & \boldsymbol{x}_N \in \mathcal{M}_f \\ & \boldsymbol{u}_i \in \mathcal{U}, i = 0, \ldots, N-1 \end{aligned} \tag{5.2}$$

5.1.1 Consistency of Time Discretization

Having performed this discretization, a reasonable (and important) sanity check on the validity of the direct approach is whether we recover the original problem in the limit of $h_i \to 0$. For simplicity, we will drop the time-dependence of the cost and dynamics. We will write the Lagrangian for (5.2) as

$$\mathcal{L} = \sum_{i=0}^{N-1} h_i c(\boldsymbol{x}_i, \boldsymbol{u}_i) + \sum_{i=0}^{N-1} \boldsymbol{\lambda}_i^T (\boldsymbol{x}_i + h_i f(\boldsymbol{x}_i, \boldsymbol{u}_i) - \boldsymbol{x}_{i+1}). \tag{5.3}$$

Then, the KKT conditions are

$$0 = h_i \frac{\partial c}{\partial \boldsymbol{x}_i}(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{\lambda}_i - \boldsymbol{\lambda}_{i-1} + h_i \frac{\partial f}{\partial \boldsymbol{x}_i}^T (\boldsymbol{x}_i, \boldsymbol{u}_i)\boldsymbol{\lambda}_i \tag{5.4}$$

$$0 = h_i \frac{\partial c}{\partial \boldsymbol{u}_i}(\boldsymbol{x}_i, \boldsymbol{u}_i) + h_i \frac{\partial f}{\partial \boldsymbol{u}_i}^T (\boldsymbol{x}_i, \boldsymbol{u}_i)\boldsymbol{\lambda}_i \tag{5.5}$$

Rearranging, we have

$$\frac{\boldsymbol{\lambda}_i - \boldsymbol{\lambda}_{i-1}}{h_i} = -\frac{\partial f}{\partial \boldsymbol{x}_i}^T (\boldsymbol{x}_i, \boldsymbol{u}_i)\boldsymbol{\lambda}_i - \frac{\partial c}{\partial \boldsymbol{x}_i}(\boldsymbol{x}_i, \boldsymbol{u}_i) \tag{5.6}$$

$$0 = \frac{\partial f}{\partial \boldsymbol{u}_i}^T (\boldsymbol{x}_i, \boldsymbol{u}_i)\boldsymbol{\lambda}_i + \frac{\partial c}{\partial \boldsymbol{u}_i}(\boldsymbol{x}_i, \boldsymbol{u}_i). \tag{5.7}$$

Let $\boldsymbol{p}(t) = \boldsymbol{\lambda}_i$ for $t \in [t_i, t_{i+1}], i = 0, \ldots, N-1$ and $p(0) = \lambda_0$. Then, the above are direct discretizations of the necessary conditions for (6.22),

$$\dot{\boldsymbol{p}}(t) = -\frac{\partial f}{\partial \boldsymbol{x}}^T (\boldsymbol{x}(t), \boldsymbol{u}(t))\boldsymbol{p}_i - \frac{\partial c}{\partial \boldsymbol{x}}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{5.8}$$

$$0 = \frac{\partial f}{\partial \boldsymbol{u}}^T (\boldsymbol{x}(t), \boldsymbol{u}(t))\boldsymbol{p}(t) + \frac{\partial c}{\partial \boldsymbol{u}}(\boldsymbol{x}(t), \boldsymbol{u}(t)). \tag{5.9}$$

5.2 Transcription Methods

A fundamental choice in the design of numerical algorithms for direct optimization of the discretized optimal control problem is whether to optimize over the state and action variables (a method known as collocation or simultaneous optimization) or strictly over the action variables (known as shooting).

5.2.1 Collocation Methods

Collocation methods optimize both the state variables and the control input at a fixed, finite number of times, $t_0, \ldots, t_i, \ldots, t_N$. Moreover, the dynamics constraints are enforced at these points. As such, it is necessary to choose a finite-dimensional representation of the trajectory between these points. This rough outline leaves unspecified a large number of algorithmic design choices.

First, how are the dynamics constraints enforced? Both derivative and integral constraints exist. The derivative approach enforces that the derivative of the state with respect to time of the parameterized trajectory is equal to the given system dynamics. The integral approach relies on integrating the given dynamics and enforcing agreement between this and the trajectory parameterization. In these notes, we will focus on the derivative approach.

Second, a choice of trajectory parameterization is required. We will primarily discuss Hermite-Simpson methods in herein, which parameterize each subinterval of the trajectory (in $[t_i, t_{i+1}]$) with a cubic polynomial. Note that the choice of a polynomial results in integral and derivative constraints being relatively simple to evaluate. However, a wide variety of parameterizations exist. For example, pseudospectral methods represent the entire trajectory as a single high-order polynomial.

We will now outline the Hermite-Simpson method as one example of direct collocation. Having selected a discretization $0 = t_0 < t_1 < \ldots < t_N = t_f$, we denote $h_i = t_{i+1} - t_i$. In every subinterval $[t_i, t_{i+1}]$, we approximate $\boldsymbol{x}(t)$ with a cubic polynomial

$$\boldsymbol{x}(t) = \boldsymbol{c}_0^i + \boldsymbol{c}_1^i(t - t_i) + \boldsymbol{c}_2^i(t - t_i)^2 + \boldsymbol{c}_3^i(t - t_i)^3 \tag{5.10}$$

which yields derivative

$$\boldsymbol{x}(t) = \boldsymbol{c}_1^i + 2\boldsymbol{c}_2^i(t - t_i) + 3\boldsymbol{c}_3^i(t - t_i)^2. \tag{5.11}$$

Writing $\boldsymbol{x}_i = \boldsymbol{x}(t_i), \boldsymbol{x}_{i+1} = \boldsymbol{x}(t_{i+1}), \dot{\boldsymbol{x}}_i = \dot{\boldsymbol{x}}(t_i), \dot{\boldsymbol{x}}_{i+1} = \dot{\boldsymbol{x}}(t_{i+1})$, we may write

$$\begin{bmatrix} \boldsymbol{x}_i \\ \dot{\boldsymbol{x}}_i \\ \boldsymbol{x}_{i+1} \\ \dot{\boldsymbol{x}}_{i+1} \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ I & h_i I & h_i^2 I & h_i^3 I \\ 0 & I & 2h_i I & 3h_i^2 I \end{bmatrix} \begin{bmatrix} \boldsymbol{c}_0^i \\ \boldsymbol{c}_1^i \\ \boldsymbol{c}_2^i \\ \boldsymbol{c}_3^i \end{bmatrix} \tag{5.12}$$

which in turn results in

$$\begin{bmatrix} \boldsymbol{c}_0^i \\ \boldsymbol{c}_1^i \\ \boldsymbol{c}_2^i \\ \boldsymbol{c}_3^i \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ -\frac{3}{h_i^2} I & -\frac{2}{h_i} I & \frac{3}{h_i^2} I & -\frac{1}{h_i} I \\ \frac{2}{h_i^2} I & \frac{1}{h_i^2} I & -\frac{2}{h_i^3} I & \frac{1}{h_i^2} I \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_i \\ \dot{\boldsymbol{x}}_i \\ \boldsymbol{x}_{i+1} \\ \dot{\boldsymbol{x}}_{i+1} \end{bmatrix}. \tag{5.13}$$

Choosing intermediate times $t_i^c = t_i + \frac{h_i}{2}$ (collocation points), we can define interpolated controls $\boldsymbol{u}_i^c = \frac{\boldsymbol{u}_i + \boldsymbol{u}_{i+1}}{2}$. From the above, we have

$$\boldsymbol{x}_i^c := \boldsymbol{x}(t_i + \frac{h_i}{2}) = \frac{1}{2}(\boldsymbol{x}_i + \boldsymbol{x}_{i+1}) + \frac{h_i}{8}(f(\boldsymbol{x}_i, \boldsymbol{u}_i, t_i) - f(\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}, t_{i+1})) \tag{5.14}$$

$$\dot{\boldsymbol{x}}_i^c := \dot{\boldsymbol{x}}(t_i + \frac{h_i}{2}) = -\frac{3}{2h_i}(\boldsymbol{x}_i + \boldsymbol{x}_{i+1}) - \frac{1}{4}(f(\boldsymbol{x}_i, \boldsymbol{u}_i, t_i) + f(\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}, t_{i+1})). \tag{5.15}$$

Thus, we can write our discretized problem as

$$\begin{aligned} \min_{\boldsymbol{u}_{0:N-1},\boldsymbol{x}_{0:N}} \quad & \sum_{i=0}^{N-1} h_i c(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\ \text{s.t.} \quad & \dot{\boldsymbol{x}}_i^c - f(\boldsymbol{x}_i^c, \boldsymbol{u}_i^c, t_i^c) = 0, i = 0, \ldots, N-1 \\ & F(\boldsymbol{x}_N) = 0 \\ & \boldsymbol{u}(t) \in \mathcal{U}, i = 0, \ldots, N-1 \end{aligned} \tag{5.16}$$

5.2.2 Shooting Methods

Shooting methods solve the discrete optimization problem via optimizing only over the control inputs, and integrating the dynamics forward given these controls. A simple approach to the forward integration is the approach we have discussed above, in which forward Euler integration is used. Single-shooting methods directly optimize the controls for the entire problem. These approaches are fairly efficient for low dimension, short horizon problems, but typically struggle to scale to larger problems. Multiple shooting methods, on the other hand, optimize via shooting over subcomponents of the problem, and enforce agreement between the trajectory segments generated via shooting within each subproblem. These methods are therefore a combination of shooting methods and collocation methods. Generally, numerical solvers for shooting problems will, given an initial action sequence, linearize the trajectory and optimize the objective function with respect to those linearized dynamics to obtain new control inputs.

5.2.3 Sequential Convex Programming

Direct optimization of the discretized nonlinear control problem typically results in a non-convex optimization problem, for which finding a good solution may be difficult or impossible. The source of this non-convexity is typically the dynamics (and sometimes the cost function). The key idea of sequential convex programming (SCP) is to iterative re-linearize the dynamics (and construct a convex approximation of the cost function, if it is non-convex) around a nominal trajectory.

First, we will assume for this outline that the cost c is convex. Let $(\boldsymbol{x}_0(\cdot), \boldsymbol{u}_0(\cdot))$ be a nominal tuple of trajectory and control (which is not necessarily feasible). We linearize the dynamics around this trajectory:

$$f_1(\boldsymbol{x}, \boldsymbol{u}, t) = f(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t), t) + \frac{\partial f}{\partial \boldsymbol{x}}(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t), t)(\boldsymbol{x} - \boldsymbol{x}_0(t)) + \frac{\partial f}{\partial \boldsymbol{u}}(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t), t)(\boldsymbol{u} - \boldsymbol{u}_0(t)). \tag{5.17}$$

We can then solve the linear optimal control problem (with $k = 0$, initially),

$$\begin{aligned} \min \quad & \int_0^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \\ \text{s.t.} \quad & \dot{\boldsymbol{x}}(t) = f_{k+1}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), t \in [0, t_f] \\ & \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ & \boldsymbol{x}(t_f) = \boldsymbol{x}_f \\ & \boldsymbol{u}(t) \in \mathcal{U}, t \in [0, t_f] \end{aligned} \tag{5.18}$$

where the dynamics are linear and the cost function is quadratic. Discretizing this continuous control problem yields a tractable convex optimization problem with dynamics $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + h_i f(\boldsymbol{x}_i, \boldsymbol{u}_i, t_i), i = 0, \ldots, N-1$. We then iterate this procedure until convergence is achieved with the new trajectory.

5.3 Further Reading

A broad introduction to direct methods for trajectory optimization is presented in [Kel17b]. This tutorial also features a discussion of trajectory optimization for hybrid systems, which we have not discussed in this section, as well as numerical solver features. For a more comprehensive review of direct methods for trajectory optimization by the same author with an emphasis on collocation methods, see [Kel17a].

Chapter 6

Model Predictive Control

Both direct and indirect methods for open-loop control result in trajectories that must be tracked with an auxiliary controller, if there is any mismatch between the systems model and the true system. This often results in a decoupling of the auxiliary controller from the original optimal control problem, which may result in performance degradation. Alternatively, the auxiliary controller may not be able to take into account other problem considerations such as state or control constraints. In this section, we introduce model predictive control, which applies the ideas from direct methods for trajectory generation online to iteratively replan, and thus results in a closed-loop controller.

6.1 Overview of MPC

Model predictive control entails solving finite-time optimal control problems in a receding horizon fashion (and thus is also frequently referred to as *receding horizon control*). The rough structure of model predictive control algorithms is

- At each sampling time t, solve an *open-loop* optimal control problem over a finite horizon
- Apply the generated optimal input signal during the subsequent sampling interval $[t, t+1)$
- At the next time step $t+1$, solve the new optimal control problem based on new measurements of the state over a shifted horizon

Consider the problem of regulating to the origin the discrete-time linear time-invariant system

$$\boldsymbol{x}(t+1) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t) \tag{6.1}$$

for $\boldsymbol{x}(t) \in \mathbb{R}^n$, $\boldsymbol{u}(t) \in \mathbb{R}^m$, subject to constraints $\boldsymbol{x}(t) \in \mathcal{X}, \boldsymbol{u}(t) \in \mathcal{U}, t \geq 0$, where $\mathcal{X}, \mathcal{U}$ are polyhedra. We will assume the full state measurement is available at time t. Given this, we can state the finite-time optimal control problem solved at each stage, t, as

$$\begin{aligned}
\min_{\boldsymbol{u}_{t|t},\dots,\boldsymbol{u}_{t+N-1|t}} \quad & c_f(\boldsymbol{x}_{t+N|t}) + \sum_{k=0}^{N-1} c(\boldsymbol{x}_{t+k|t}, \boldsymbol{u}_{t+k|t}) \\
\text{s.t.} \quad & \boldsymbol{x}_{t+k+1|t} = A\boldsymbol{x}_{t+k|t} + B\boldsymbol{u}_{t+k|t}, \quad k = 0,\dots,N-1 \\
& \boldsymbol{x}_{t+k|t} \in \mathcal{X}, \quad k = 0,\dots,N-1 \\
& \boldsymbol{u}_{t+k|t} \in \mathcal{U}, \quad k = 0,\dots,N-1 \\
& \boldsymbol{x}_{t+N|t} \in \mathcal{X}_f, \\
& \boldsymbol{x}_{t|t} = \boldsymbol{x}(t)
\end{aligned} \tag{6.2}$$

for which we write the solution as $J_t^*(\boldsymbol{x}(t))$. In this problem, $\boldsymbol{x}_{t+k|t}$ and $\boldsymbol{u}_{t+k|t}$ are the state and action predicted at time $t+k$ from time t. Letting $U^*_{t\to t+N|t} := \{\boldsymbol{u}^*_{t|t}, \dots, \boldsymbol{u}^*_{t+N-1|t}\}$ denote the optimal solution, we take $\boldsymbol{u}(t) = \boldsymbol{u}^*_{t|t}(\boldsymbol{x}(t))$. This optimization problem is then repeated at time $t+1$, based on the new state $\boldsymbol{x}_{t+1|t+1} = \boldsymbol{x}(t+1)$. Defining the closed-loop control policy as $\pi_t(\boldsymbol{x}(t)) := \boldsymbol{u}^*_{t|t}(\boldsymbol{x}(t))$, we have the closed-loop dynamics

$$\boldsymbol{x}(t+1) = A\boldsymbol{x}(t) + B\pi_t(\boldsymbol{x}(t)). \tag{6.3}$$

Thus, the central question of this formulation becomes characterizing the behavior of the closed-loop system defined by this iterative re-optimization. As the problem is time-invariant, we can rewrite the closed-loop dynamics as

$$\boldsymbol{x}(t+1) = A\boldsymbol{x}(t) + B\pi(\boldsymbol{x}(t)). \tag{6.4}$$

The rough structure of the online model predictive control framework is then as follows:

1. Measure the state $\boldsymbol{x}(t)$ at every time t
2. Obtain $U_0^*(\boldsymbol{x}(t)$ by solving finite-time optimal control problem
3. If $U_0^*(\boldsymbol{x}(t) = \emptyset$ then 'problem infeasible', stop
4. Apply the first element $\boldsymbol{u}_0^*$ of $U_0^*(\boldsymbol{x}(t)$ to the system
5. Wait for the new sampling time $t+1$

This framework leads to two main implementation issues. First, the controller may lead us into a situation where after a few steps the finite-time optimal control problem is infeasible, which we refer to as the *persistent feasibility issue.* Even if the feasibility problem does not occur, the generated control inputs may not lead to trajectories that converge to the origin, which we refer to as the *stability issue.* The key question in the analysis of MPC algorithms is how we may guarantee that our "short-sighted" control strategy leads to effective long-term behavior. While one possible approach is directly analyzing the closed-loop dynamics, this is in practice very difficult. Our approach will instead be to derive conditions on the terminal function c_f and terminal constraint set $\mathcal{X}_f$ so that the persistent feasibility and closed-loop stability are guaranteed.

6.2 Feasibility

Model predictive control simplifies the online control optimization problem by solving a shorter horizon problem, as opposed to solving the full optimal control problem online at each timestep. This myopic optimization leads to the possibility that after several steps, the problem may no longer be feasible. As such, in this section we will discuss approaches to impose constraints on so-called *recursive feasibility* to avoid this problem.

Let

$$\begin{aligned}\mathcal{X}_0 := \{\boldsymbol{x} \in \mathcal{X} \mid \exists(\boldsymbol{u}_0, \dots, \boldsymbol{u}_{N-1}) \text{ s.t. } \boldsymbol{x}_k \in \mathcal{X}, \boldsymbol{u}_k \in \mathcal{U}, k = 0, \dots, N-1, \\ \boldsymbol{x}_N \in \mathcal{X}_f, \text{ where } \boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k, k = 0, \dots, N-1\}\end{aligned} \tag{6.5}$$

be the set of feasible initial states. Simply, this set is the set of initial states for which a sequence of control inputs exist that cause the final state to satisfy the terminal constraint. For the autonomous system $\boldsymbol{x}(t+1) = \phi(\boldsymbol{x}(t))$ with constraints $\boldsymbol{x}(t) \in \mathcal{X}, \boldsymbol{u}(t) \in \mathcal{U}$, the one-step controllable set to set $\mathcal{S}$ is defined as

$$\operatorname{Pre}(\mathcal{S}) := \{\boldsymbol{x} \in \mathbb{R}^n : \phi(\boldsymbol{x}) \in \mathcal{S}\}. \tag{6.6}$$

For the system $\boldsymbol{x}(t+1) = \phi(\boldsymbol{x}(t), \boldsymbol{u}(t))$ with constraints $\boldsymbol{x}(t) \in \mathcal{X}, \boldsymbol{u}(t) \in \mathcal{U}$, the one-step controllable set to set $\mathcal{S}$ is defined as

$$\operatorname{Pre}(\mathcal{S}) := \{\boldsymbol{x} \in \mathbb{R}^n : \exists \boldsymbol{u} \in \mathcal{U} \text{ s.t. } \phi(\boldsymbol{x}, \boldsymbol{u}) \in \mathcal{S}\}. \tag{6.7}$$

A set $\mathcal{C} \subseteq \mathcal{X}$ is said to be a control invariant set for the system $\boldsymbol{x}(t+1) = \phi(\boldsymbol{x}(t), \boldsymbol{u}(t))$ with constraints $\boldsymbol{x}(t) \in \mathcal{X}, \boldsymbol{u}(t) \in \mathcal{U}$, if

$$\boldsymbol{x}(t) \in \mathcal{C} \implies \exists \boldsymbol{u} \in \mathcal{U} \text{ s.t. } \phi(\boldsymbol{x}(t), \boldsymbol{u}(t)) \in \mathcal{C}, \forall t. \tag{6.8}$$

The set $\mathcal{C}_\infty \subset \mathcal{X}$ is said to the maximal control invariant set for the system $\boldsymbol{x}(t+1) = \phi(\boldsymbol{x}(t), \boldsymbol{u}(t))$ with constraints $\boldsymbol{x}(t) \in \mathcal{X}, \boldsymbol{u}(t) \in \mathcal{U}$, if it control invariant all control invariant sets contained in $\mathcal{X}$[1].

We will now proceed to derive critical results on recursive feasibility for linear dynamical systems. We will define the "truncated" feasibility set

$$\begin{aligned}\mathcal{X}_1 := \{\boldsymbol{x} \in \mathcal{X} \mid \exists(\boldsymbol{u}_1, \dots, \boldsymbol{u}_{N-1}) \text{ s.t. } \boldsymbol{x}_k \in \mathcal{X}, \boldsymbol{u}_k \in \mathcal{U}, k = 1, \dots, N-1, \\ \boldsymbol{x}_N \in \mathcal{X}_f, \text{ where } \boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k, k = 1, \dots, N-1\}.\end{aligned} \tag{6.9}$$

Then, we may state the following result on feasibility.

Lemma 6.2.1 (Persistent Feasibility)**.** *If set $\mathcal{X}_1$ is a control invariant set for system $\boldsymbol{x}(t+1) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t)$, $\boldsymbol{x}(t) \in \mathcal{X}, \boldsymbol{u}(t) \in \mathcal{U}$, then the MPC law is persistently feasible.*

[1]Control invariant sets can be computed using the MPT toolbox: `www.mpt3.org`

Proof. Note that

$$\mathrm{Pre}(\mathcal{X}_1) := \{\boldsymbol{x} \in \mathbb{R}^n : \exists \boldsymbol{u} \in \mathcal{U} \text{ s.t. } A\boldsymbol{x} + B\boldsymbol{u} \in \mathcal{X}_1\}. \tag{6.10}$$

Since $\mathcal{X}_1$ is control invariant, there exists $\boldsymbol{u} \in \mathcal{U}$ such that $A\boldsymbol{x} + B\boldsymbol{u} \in \mathcal{X}_1$ for all $\boldsymbol{x} \in \mathcal{X}_1$. Thus, $\mathcal{X}_1 \subseteq \mathcal{X}_1 \cap \mathcal{X}$. One may write

$$\mathcal{X}_0 = \{\boldsymbol{x}_0 \in \mathcal{X} \mid \exists \boldsymbol{u}_0 \in \mathcal{U} \text{ s.t. } A\boldsymbol{x}_0 + B\boldsymbol{u}_0 \in \mathcal{X}_1\} = \mathrm{Pre}(\mathcal{X}_1) \cap \mathcal{X}. \tag{6.11}$$

This then implies $\mathcal{X}_1 \subseteq \mathcal{X}_0$. Choose some $\boldsymbol{x}_0 \in \mathcal{X}_0$. Let U_0^* be the solution to the finite-time optimization problem, and $\boldsymbol{u}_0^*$ be the first control. Let $\boldsymbol{x}_1 = A\boldsymbol{x}_0 + B\boldsymbol{u}_0^*$. Since U_0^* is feasible, one has $\boldsymbol{x}_1 \in \mathcal{X}_1$. Since $\mathcal{X}_1 \subseteq \mathcal{X}_0$, $\boldsymbol{x}_1 \in \mathcal{X}_0$, and hence the next optimization problem is feasible. □

For $N = 1$, we may set $\mathcal{X}_f = \mathcal{X}1$. If the terminal set is chosen to be control invariant, then MPC problem will be persistently feasible *independent* of chosen control objectives and parameters. The system designer may then choose the parameters to affect the system performance. The logical question, then, is how to extent this result to $N > 1$, for which we have the following result.

Theorem 6.2.2 (Persistent Feasibility)**.** *If $\mathcal{X}_f$ is a control invariant set for the the system $\boldsymbol{x}(t+1) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t)$, $\boldsymbol{x}(t) \in \mathcal{X}, \boldsymbol{u}(t) \in \mathcal{U}, t \geq 0$, then the MPC law is persistently feasible.*

Proof. We will begin by defining the "truncated" feasibility set at step $N-1$,

$$\begin{aligned}\mathcal{X}_{N-1} := \{\boldsymbol{x}_{N-1} \in \mathcal{X} \mid \exists \boldsymbol{u}_{N-1} \text{ s.t. } &\boldsymbol{x}_{N-1} \in \mathcal{X}, \boldsymbol{u}_{N-1} \in \mathcal{U} \\ &\boldsymbol{x}_N \in \mathcal{X}_f, \text{ where } \boldsymbol{x}_N = A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1}\}.\end{aligned} \tag{6.12}$$

Due to the terminal constraints, have $A\boldsymbol{x}_{N-1} + B\boldsymbol{u}_{N-1} = \boldsymbol{x}_N \in \mathcal{X}_f$. Since $\mathcal{X}_f$ is a control invariant set, there exists a $\boldsymbol{u} \in \mathcal{U}$ such that $\boldsymbol{x}^+ = A\boldsymbol{x}_N + B\boldsymbol{u} \in \mathcal{X}_f$. This is the requirement that $\boldsymbol{x}_N \in \mathcal{X}_{N-1}$. Thus, $\mathcal{X}_{N-1}$ is control invariant. Repeating this argument, one can recursively show that $\mathcal{X}_{N-2}, \ldots, \mathcal{X}_1$ are control invariant, and the persistent feasibility lemma then applies. □

Practically, we introduce the terminal set $\mathcal{X}_f$ artificially for the purpose of leading to a sufficient condition for persistent feasibility. We would like to choose it to be large, so that is avoids compromising closed-loop performance.

6.3 Stability

Persistent feasibility does not guarantee that the closed-loop trajectories converge toward the desired equilibrium point. One of the most popular approaches to guarantee persistent feasibility and stability of the MPC law makes use of a control invariant terminal set $\mathcal{X}_f$ for feasibility, and a terminal function $c_f(\cdot)$ for stability. To prove stability, we leverage Lyapunov stability theory.

Theorem 6.3.1 (Lyapunov Stability)**.** *Consider the equilibrium point $\boldsymbol{x} = 0$ for the autonomous system $\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k)$ (with $f(0) = 0$). Let $\Omega \subset \mathbb{R}^n$ be a closed and bounded set containing the origin. Let $V : \mathbb{R}^n \to \mathbb{R}$ be a function, continuous at the origin, such that*

$$V(0) = 0 \textit{ and } V(\boldsymbol{x}) > 0,\ \forall \boldsymbol{x} \in \Omega \setminus \{0\} \tag{6.13}$$

$$V(\boldsymbol{x}_{k+1}) - V(\boldsymbol{x}_k) < 0,\ \forall \boldsymbol{x} \in \Omega \setminus \{0\}. \tag{6.14}$$

Then $\boldsymbol{x} = 0$ is asymptotically stable in Ω.

We will utilize this result to show that with appropriate choices of $\mathcal{X}_f$ and $c_f(\cdot)$, J_0^* is a Lyapunov function for the closed-loop system.

Theorem 6.3.2 (MPC Stability (for Quadratic Cost))**.** *Assume*

1. $Q = Q^T > 0, R = R^T > 0, Q_f > 0$
2. *Sets $\mathcal{X}, \mathcal{X}_f$, and $\mathcal{U}$ contain the origin in their interior and are closed*
3. *$\mathcal{X}_f \subseteq \mathcal{X}$ is control invariant*
4. $\min_{\boldsymbol{v}\in\mathcal{U}, A\boldsymbol{x}+B\boldsymbol{v}\in\mathcal{X}_f} \{-c_f(\boldsymbol{x}) + c(\boldsymbol{x}, \boldsymbol{v}) + c_f(A\boldsymbol{x} + B\boldsymbol{v})\} \leq 0, \forall \boldsymbol{x} \in \mathcal{X}_f.$

Then, the origin of the closed-loop system is asymptotically stable with domain of attraction $\mathcal{X}_f$.

Proof. Note that via assumption 3, persistent feasibility is guaranteed for any Q_f, Q, R. We want to show that J_0^* is a Lyapunov function for the closed-loop system $\boldsymbol{x}(t+1) = f_{cl}(\boldsymbol{x}(t)) = A\boldsymbol{x}(t) + B\pi(\boldsymbol{x}(t))$, with respect to the equilibrium $f_{cl}(0) = 0$ (the origin is indeed an equilibrium as $0 \in \mathcal{X}, 0 \in \mathcal{U}$, and the cost is positive for any non-zero control sequence. Note also that $\mathcal{X}_0$ is closed and bounded, and $J_0^*(0) = 0$, both by assumption. Note also that $J_0^*(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathcal{X}_0 \setminus \{0\}$.

We will now show the decay property. Since the setup is time-invariant, we can study the decay property between $t = 0$ and $t = 1$. Let $\boldsymbol{x}(0) \in \mathcal{X}_0$, let $U_0^{[0]} = [\boldsymbol{u}_0^{[0]}, \ldots, \boldsymbol{u}_{N-1}^{[0]}]$ be the optimal control sequence, and let $[\boldsymbol{x}(0), \ldots, \boldsymbol{x}_N^{[0]}]$ be the corresponding trajectory. After applying $\boldsymbol{u}_0^{[0]}$, one obtains $\boldsymbol{x}(1) = A\boldsymbol{x}(0) + B\boldsymbol{u}_0^{[0]}$. Now, consider the sequence of control $[\boldsymbol{u}_1^{[0]}, \ldots, \boldsymbol{u}_{N-1}^{[0]}, \boldsymbol{v}]$, where $\boldsymbol{v} \in \mathcal{U}$ and the corresponding state trajectory is $[\boldsymbol{x}(1), \ldots, \boldsymbol{x}_N^{[0]}, A\boldsymbol{x}_N^{[0]} + B\boldsymbol{V}]$. Since $\boldsymbol{x}_N^{[0]} \in \mathcal{X}_f$ (by the terminal constraint), and since $\mathcal{X}_f$ is control invariant,

$$\exists \bar{\boldsymbol{v}} \in \mathcal{U} \mid A\boldsymbol{x}_N^{[0]} + B\bar{\boldsymbol{v}} \in \mathcal{X}_f. \tag{6.15}$$

With such a choice of $\bar{\boldsymbol{v}}$, the sequence $[\boldsymbol{u}_1^{[0]}, \ldots, \boldsymbol{u}_{N-1}^{[0]}, \boldsymbol{v}]$ is feasible for the MPC optimization problem at time $t = 1$. Subce tgus sequence is not necessarily optimal,

$$J_0^*(\boldsymbol{x}(1)) \leq c_f(A\boldsymbol{x}_N^{[0]} + B\bar{\boldsymbol{v}}) + \sum_{k=1}^{N-1} c(\boldsymbol{x}_k^{[0]}, \boldsymbol{u}_k^{[0]}) + c(\boldsymbol{x}_N^{[0]}, \boldsymbol{v}). \tag{6.16}$$

Equivalently,

$$J_0^*(\boldsymbol{x}(1)) \leq c_f(A\boldsymbol{x}_N^{[0]} + B\bar{\boldsymbol{v}}) + J_0^*(\boldsymbol{x}(0)) - c_f(\boldsymbol{x}_N^{[0]}) - c(\boldsymbol{x}(0), \boldsymbol{u}_0^{[0]}) + c(\boldsymbol{x}_N^{[0]}, \bar{\boldsymbol{v}}) \tag{6.17}$$

Since $\boldsymbol{x}_N^{[0]} \in \mathcal{X}_f$ by assumption, we can select $\bar{\boldsymbol{v}}$ such that

$$J_0^*(\boldsymbol{x}(1)) \leq J_0^*(\boldsymbol{x}(0)) - c(\boldsymbol{x}(0), \boldsymbol{u}_0^{[0]}). \tag{6.18}$$

Since $c(\boldsymbol{x}(0), \boldsymbol{u}_0^{[0]}) > 0$ for all $\boldsymbol{x}(0) \in \mathcal{X}_0 \setminus \{0\}$,

$$J_0^*(\boldsymbol{x}(1)) - J_0^*(\boldsymbol{x}(0)) < 0. \tag{6.19}$$

The last step is to prove continuity, for which we omit the details and refer the reader to [BBM17]. □

6.3.1 Choosing $\mathcal{X}_f$ and Q_f

We will look at two cases. First, we will assume that A is asymptotically stable. Then, we set $\mathcal{X}_f$ as the maximally positive invariant set $\mathcal{O}_\infty$ for the system $\boldsymbol{x}(t+1) = A\boldsymbol{x}(t), \boldsymbol{x}(t) \in \mathcal{X}$. The set $\mathcal{X}_f$ is a control invariant set for the system $\boldsymbol{x}(t+1) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t)$ as $\boldsymbol{u} = 0$ is a feasible control. As for stability, $\boldsymbol{u} = 0$ is feasible and $A\boldsymbol{x} \in \mathcal{X}_f$ if $\boldsymbol{x} \in \mathcal{X}_f$, thus assumption 4 of Theorem 6.3.1 becomes

$$-\boldsymbol{x}^T Q_f \boldsymbol{x} + \boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{x}^T A^T Q_f A \boldsymbol{x} \leq 0, \forall \boldsymbol{x} \in \mathcal{X}_f \tag{6.20}$$

which is true since, due to the fact that A is asymptotically stable,

$$\exists Q_f > 0 \mid -Q_f + Q + A^T Q_f A = 0 \tag{6.21}$$

Next, we will look at the general case. Let L_∞ be the optimal gain for the infinite-horizon LQR controller. Set $\mathcal{X}_f$ as the maximal positive invariant set for system $\boldsymbol{x}(t+1) = (A + BL_\infty)\boldsymbol{x}(t)$ (with constraints $\boldsymbol{x}(t) \in \mathcal{X}, L_\infty \boldsymbol{x}(t) \in \mathcal{U}$). Then, set Q_f as the solution Q_∞ to the discrete-time Riccati equation.

6.3.2 Explicit MPC

In some cases, the MPC law can be pre-computed, which removes the need for online optimization. An important case of this is that of constrained LQR, in which we wish to solve the optimal control problem

$$\begin{aligned}
\min_{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}} \quad & \boldsymbol{x}_N^T Q_f \boldsymbol{x}_N + \sum_{k=0}^{N-1} \boldsymbol{x}_k^T Q \boldsymbol{x}_k + \boldsymbol{u}_k^T R \boldsymbol{u}_k \\
\text{s.t.} \quad & \boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k, \quad k = 0, \ldots, N-1 \\
& \boldsymbol{x}_k \in \mathcal{X}, \quad k = 0, \ldots, N-1 \\
& \boldsymbol{u}_k \in \mathcal{U}, \quad k = 0, \ldots, N-1 \\
& \boldsymbol{x}_N \in \mathcal{X}_f, \\
& \boldsymbol{x}_0 = \boldsymbol{x}
\end{aligned} \tag{6.22}$$

The solution to the constrained LQR problem is a control $\boldsymbol{u}^*$ which is a continuous piecewise affine function on polyhedral partition of the state space $\mathcal{X}$, that is $\boldsymbol{u}^* = \pi(\boldsymbol{x})$, where

$$\pi(\boldsymbol{x}) = L^j\boldsymbol{x} + \boldsymbol{l}^j \text{ if } H^j\boldsymbol{x} \leq K^j,\ j = 1, \ldots, N^r. \tag{6.23}$$

Thus, online, one has to locate in which cell of the polyhedral partition the state $\boldsymbol{x}$ lies, and then one obtains the optimal control via a look-up table query.

6.4 Further Reading

We refer the reader to [BBM17] and [RMD17] for two broad and comprehensive treatments of the topic.

Part II

Adaptive Control and Reinforcement Learning

Chapter 7

Introduction

7.1 Learning in Optimal Control

7.1.1 What Should we Learn?

7.1.2 Episodes and Data Collection

Chapter 8

Regression

In this section we will develop the statistical tools behind regression in the linear and non-linear setting. In the regression setting we aim to estimate the function that maps a set of *inputs* (equivalently, independent variables, covariates or features) to a *continuous*[1] output (or dependent variables). Regression models are the foundation upon which the rest of the discussion of learning-based control will be built: we will typically pose the problem of learning dynamics models, value functions, or policies as regression problems. For example, given a set of measurements of the system state and the control action, as well as the subsequent state, we can fit a dynamics model that captures the system dynamics.

Concretely, we will assume we are provided d pairs of the form $\boldsymbol{x}_i, y_i$. We will assume throughout our discussion on linear and Gaussian process regression that y is a scalar, but the methods we will develop can easily be generalized to vector outputs. Let $\hat{y}_i$ denote our prediction given input $\boldsymbol{x}_i$ Then, we aim to find some function f such that, for prediction $\hat{y}_i = f(\boldsymbol{x}_i)$, $\hat{y}_i$ is close to y_i. While one could imagine posing this problem as an optimization problem or a question of statistical inference, it leaves several questions unanswered. First, we have not defined what it means for our predictions to be "close" to the measured output. Beyond this, and more subtly, we haven't addressed how the function f is represented.

We will begin with the least squares linear regression setting, in which $f(\boldsymbol{x})$ is linear. While we expect that this will be a review for most readers, we will use this setting to highlight several fundamental concepts in frequentist statistics—in which model parameters are assumed fixed and we do not specify prior beliefs over these parameters—and Bayesian statistics, which focuses on computing the posterior belief over model parameters given a prior. We will then discuss Gaussian process regression and neural network models, which are two highly influential approaches to nonlinear regression in reinforcement learning and adaptive control. Highlighting both the frequentist and the Bayesian approaches is critical, as neural network regression models are typically presented from the frequentist point of view, whereas Gaussian process models are Bayesian models. Moreover, both of these approaches have strengths and weaknesses that may be relevant to the downstream control task.

[1]This is in contrast to the classification setting in which we aim to map input variables to a (usually) finite set of output variables

8.1 Linear Regression

8.1.1 Minimum Mean Squared Error

In this section we will discuss regression with linear functions,

$$\hat{y} = f(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x}. \tag{8.1}$$

Here, $\boldsymbol{\theta}$ are the parameters of the model. Note that the linear function above has an intercept at zero—there is no offset term. Thus, we will assume throughout our discussion of linear regression that the input vector is augmented with a 1, so that

$$\hat{y} = \theta_0 + \boldsymbol{\theta}^T \boldsymbol{x} = \begin{bmatrix} \theta_0 & \boldsymbol{\theta}^T \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}. \tag{8.2}$$

Thus when we write $\boldsymbol{x}$, simply imagine the input augmented with a 1 at the first entry.

Now that we have fixed our function class, we need to fix an objective (or some sort of concrete metric for our predictions being "close" to the measured data). We will choose the *mean squared error* (MSE) cost function,

$$J(\theta) = \frac{1}{d} \sum_{i=1}^{d} \left(y_i - \hat{y}\right)^2. \tag{8.3}$$

The MSE cost (or equivalently, loss) function is the most common loss function in regression problems due to several favorable properties, both intuitive and computational (which we will see later in this section). Simply, minimizing this loss encodes the objective of minimizing the squared error in our prediction. Given our linear model parameterization and this loss function, we can now fully specify our regression problem as

$$\min_{\boldsymbol{\theta}} \quad \frac{1}{d} \sum_{i=1}^{d} \left(y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i\right)^2 \tag{8.4}$$

which we will rewrite as

$$\min_{\boldsymbol{\theta}} \quad \|\boldsymbol{y} - X\boldsymbol{\theta}\|_2^2 \tag{8.5}$$

where we have multiplied the objective by d to simplify notation, and where $\boldsymbol{y}^T = [y_1, \dots, y_d]$ and $X^T = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_d]$[2].

First, note that this loss is convex in the model parameters and is differentiable with continuous derivative (in class C^1). From this, we know any point satisfying the first order necessary conditions for optimality is a global minimizer. Thus, to solve this problem, we will compute the gradient

$$\nabla_{\boldsymbol{\theta}} \|\boldsymbol{y} - X\boldsymbol{\theta}\|_2^2 = 2X^T X\boldsymbol{\theta} - 2X^T \boldsymbol{y} \tag{8.6}$$

[2]This matrix is typically referred to as the *design matrix*.

which, setting this gradient to zero, we have

$$X^T X\boldsymbol{\theta} = X^T \boldsymbol{y}. \tag{8.7}$$

If $X^T X$ is invertible then the optimal set of parameters is

$$\hat{\boldsymbol{\theta}} = (X^T X)^{-1} X^T \boldsymbol{y}. \tag{8.8}$$

We will write the set of parameters that solve the least squares problem in the overdetermined form (as above) as $\hat{\boldsymbol{\theta}}_{LS}$. When is $X^T X$ invertible? Note that

$$X^T X = \sum_{i=1}^{d} \boldsymbol{x}_i \boldsymbol{x}_i^T. \tag{8.9}$$

Let n denote the dimension of the input. Then, we require $d > n$ for $X^T X$ to be full rank. However, at least n input is not a sufficient condition for $X^T X$ to be full rank; we require at least n linearly independent inputs.

An intuitive understanding of the requirements for the exact computation of $\hat{\boldsymbol{\theta}}_{LS}$ is provided by considering the problem geometrically. The invertibility of $X^T X$ implies the unique existence of a set of parameters $\boldsymbol{\theta}$ that minimizes (8.5). In the case that we have fewer linearly independent data points than n, (8.7) is not unique. Thus, there exist multiple values of $\boldsymbol{\theta}$ that result in zero error, and the system is underdetermined. A common approach to this is to optimize a regularized objective

$$\min_{\boldsymbol{\theta}} \quad \|\boldsymbol{y} - X\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_2^2 \tag{8.10}$$

where λ is a positive scalar. By adding in the regularization term $\|\boldsymbol{\theta}\|_2^2$ (typically referred to as Ridge or Tikhanov regression, or L_2 regression due to the use of the 2-norm), we bias the optimal value of $\boldsymbol{\theta}$ toward having a small norm. Following the approach of (8.11), we have

$$\hat{\boldsymbol{\theta}} = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y} \tag{8.11}$$

for which the $X^T X + \lambda I$ is always full rank and thus invertible. We will write the solution to the ridge regularized problem as $\hat{\boldsymbol{\theta}}_{RR}$. For positive λ, $\|\boldsymbol{y} - X\boldsymbol{\theta}\|_2^2$ may not be zero, and the estimator of the parameters is *biased*. As λ gets smaller, this bias will decrease. In the limit of $\lambda \to 0$, we have

$$(X^T X + \lambda I)^{-1} X^T \to X^T (X X^T)^{-1} \tag{8.12}$$

yielding the *least norm* solution to (8.7), which we write

$$\hat{\boldsymbol{\theta}}_{LN} = X^T (X X^T)^{-1} \boldsymbol{y}. \tag{8.13}$$

The least norm solution is the set of parameters that achieves zero loss on (8.5) while simultaneously minimizing the 2-norm of the parameter vector. So, is the least norm solution strictly preferred to a set of parameters computed via the ridge regression approach with a positive value of λ? Surprisingly, the answer is no. The added bias from the regularization helps generalize to new data points without *overfitting* to the training dataset. Regularization is an important concept in machine learning, and we will discuss it further throughout this section, including showing the ridge regression problem arises naturally in Bayesian inference.

8.1.2 Maximum Likelihood Estimation

Our previous discussion of least squares linear regression has avoided formalizing several implicit, necessary assumptions. In this section, we will investigate the method of *maximum likelihood estimation*, which we will use to derive the least squares estimator from the previous section. Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^d$ denote the training dataset. The likelihood function is the conditional probability of the data given the parameter,

$$L(\boldsymbol{\theta}) = p(\mathcal{D} \mid \boldsymbol{\theta}). \tag{8.14}$$

The goal of maximum likelihood estimation is to choose the parameter that maximizes the probability of the training data

$$\boldsymbol{\theta}^* = \text{argmax}_{\boldsymbol{\theta}} L(\boldsymbol{\theta}). \tag{8.15}$$

In practice, we will typically consider the *log likelihood*, $\ell(\boldsymbol{\theta}) = \log L(\boldsymbol{\theta})$. Note that the maximizer of the log likelihood is equivalent to the maximizer of the likelihood due to the monotonicity of the log function. The log likelihood is more practical for several reasons. First, we will make the standard assumption that our training data are independent (one $(\boldsymbol{x}, y)$ pair we observe have no impact on any other pair) and identically distributed (i.i.d.). Then, the joint distribution is

$$p(\mathcal{D} \mid \boldsymbol{\theta}) = \prod_{i=1}^{d} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \tag{8.16}$$

where we have discarded the distribution $p(\boldsymbol{x}_i \mid \boldsymbol{\theta}) = p(\boldsymbol{x})$ as we assume it contains no information about $\boldsymbol{\theta}$. For the log likelihood, this joint takes the form

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{i=1}^{d} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) \tag{8.17}$$

resulting in the more convinient summation as opposed to the product. There are several other important properties that we will not detail here regarding the log likelihood; for example, due to the log-concavity of the exponential family, optimization of the log likelihood leads to a convex optimization problem for many common distributions. In practice, we will write maximum likelihood problems as minimization of the negative log likelihood to unify the notation.

Linear regression via MLE

We will assume the underlying generative model of our data takes the form

$$y = \boldsymbol{x}^T \boldsymbol{\theta}^* + \epsilon \tag{8.18}$$

where $\boldsymbol{\theta}^*$ is the true underlying set of parameters and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is a Gaussian, zero-mean noise term. Given this model, the likelihood

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}^T \boldsymbol{\theta}, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y - \boldsymbol{x}^T \boldsymbol{\theta})^2}{2\sigma^2}} \tag{8.19}$$

resulting in log likelihood of the dataset

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{d} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}) = \sum_{i=1}^{d} \left(-\log(\sqrt{2\pi}) - \frac{(y_i - \boldsymbol{x}_i^T \boldsymbol{\theta})^2}{2\sigma^2} \right) \tag{8.20}$$

which, discarding the constant term $\log(\sqrt{2\pi})$ and the multiplicative term $2\sigma^2$ as they do not effect the optimization problem, yields exactly the mean squared error criterion we considered in the previous section. The least squares model we have developed in this section (in particular, under the i.i.d.assumption in combination with the assumption that we receive noise-free measurements of the input variable $\boldsymbol{x}$) is referred to as the *ordinary least squares* (OLS) method.

While in the process of this derivation we have assumed Gaussian additive noise, such as assumption is not necessary. We will refer to $\hat{\boldsymbol{\theta}}_{LS}$ as an estimator, which maps training data to a parameter estimate. The quality of this estimator can be evaluated under a loss function (note that this is a loss on the parameter estimate as opposed to a loss function on the prediction of the model). We will fix a MSE loss function on the parameter estimate of the form

$$\mathbb{E}_{\mathcal{D}} \left[\|\boldsymbol{\theta}^* - \hat{\boldsymbol{\theta}}\|_2^2 \right]. \tag{8.21}$$

An estimator is said to be unbiased if

$$\mathbb{E}_{\mathcal{D}}[\hat{\boldsymbol{\theta}}] = \boldsymbol{\theta}^*. \tag{8.22}$$

Finally, note that an estimator is *linear* if it is a linear combination of the output measurements. Then, we have the following result for the OLS estimator. If ϵ is sampled *i.i.d.* with zero mean, then the OLS estimator is the best (in terms of MSE loss) linear unbiased estimator (BLUE). This is referred to as the Gauss-Markov theorem. This result provides a basis for using the OLS estimator for a wide variety of settings, as it is flexible, interpretable, and performs well in a variety of settings.

Basis Function Regression

We have so far restricted the class of models under consideration to functions linear in the input, $\boldsymbol{x}$. For many problems, this is an unrealistic model, and so it is desirable to consider a more expressive class of models. Because we have noise-free measurements of the input $\boldsymbol{x}$, we can instead consider nonlinear functions of these inputs. We will refer to these as *features*, and write them as $\phi(\boldsymbol{x})$, where ϕ is some nonlinear function. Note here that we fix ϕ is advance, and then optimize the weightings of these features. This approach will be generalized when we consider neural network models, for which we can directly optimize the features as a function of the training data.

Which features should we use? In general, it depends on your problem setting. For example, if we consider oscillatory pendulum dynamics, we should include sinusoidal features. Other common choices include polynomials of the inputs, indicators for certain regions, or more complex functions like radial basis functions. While increasing the number of features results in a more expressive model, it also increases the risk of overfitting, and so the features should be chosen carefully, for example via cross validation. We will not discuss feature selection in detail in this work, and we refer the reader to [FHT08].

8.1.3 Bayesian Inference and Bayesian Linear Regression

We have so far considered a frequentist parameter estimation setting in which we aim to choose a point estimate of some parameter to minimize a loss function. We will now look at Bayesian inference. In this setting, we will assume access to a prior distribution (or belief) over the parameter, which we will write $p(\boldsymbol{\theta})$. This prior captures any information we have about the parameter before observing any other data. This could be influenced by knowledge available to a system designer (for example, rough estimates of dynamics parameters) or from data from other, related problems. Given this prior distribution combined with a likelihood function, we will use Bayes' rule to compute the posterior distribution over the parameters, via

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{D})}. \tag{8.23}$$

This approach has several strengths and weaknesses. Whereas frequentist estimation gives us a point estimate of a parameter (possibly with confidence intervals), Bayesian inference returns a full distribution over the parameter. This full representation of the uncertainty over a parameter can be incorporated downstream into decision-making algorithms, potentially resulting in increased robustness. However, the major limitation of Bayesian inference is that it is analytically intractable in most cases: for arbitrary models for the prior and the likelihood, the posterior density likely does not have a convenient, closed-form representation. This is a result of the need to compute the marginal likelihood, $p(\mathcal{D})$. This can be computed via

$$\int p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{8.24}$$

which is typically an intractable integral. In general, practitioners must therefore turn to sampling-based or discrete approximations. However, one special case for which the posterior can be computed exactly is the case for which the prior and the likelihood are Gaussian. This fact underlies, for example, the Kalman filter, and will be critical to our development of Bayesian least squares.

Bayesian Least Squares

We will now show how the Bayesian inference can be applied to the least squares problem. Concretely, we will consider the *conditional* density estimation problem

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}, X) = \frac{p(\boldsymbol{y} \mid \boldsymbol{\theta}, X) p(\boldsymbol{\theta})}{p(\boldsymbol{y} \mid X)}. \tag{8.25}$$

As previously, nonlinear transformations of the input variables may be used, but for simplicity of notation we will use un-transformed inputs. Let

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_0|}} \exp(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_0)^T \Sigma_0^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu}_0)) \tag{8.26}$$

denote the prior over $\boldsymbol{\theta}$, where $\boldsymbol{\mu}_0$ and Σ_0 are the mean and covariance respectively. As in the previous section, we will assume $\epsilon \sim \mathcal{N}(0, \sigma^2)$, and that this noise term is sampled i.i.d.. We will assume that the noise covariance, σ^2, is known, but in general this assumption is fairly easily relaxed.

To compute the posterior over $\boldsymbol{\theta}$, note that

$$p(\boldsymbol{\theta} \mid \mathcal{D}) \propto \exp(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_0)^T \Sigma_0^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu}_0)) \exp(-\frac{1}{2\sigma^2}(\boldsymbol{y} - X\boldsymbol{\theta})^T (\boldsymbol{y} - X\boldsymbol{\theta})) \tag{8.27}$$

where we have ignored the normalizing constants. Looking at the exponentiated terms, we have

$$\begin{aligned} & -\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_0)^T \Sigma_0^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu}_0) + -\frac{1}{2\sigma^2}(\boldsymbol{y} - X\boldsymbol{\theta})^T (\boldsymbol{y} - X\boldsymbol{\theta})) && (8.28) \\ \propto & -\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_d)^T \Sigma_d^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu}_d) && (8.29) \end{aligned}$$

where

$$\begin{aligned} \Sigma_d &= (\frac{1}{\sigma^2} X^T X + \Sigma_0^{-1})^{-1} && (8.30) \\ \boldsymbol{\mu}_d &= \Sigma_d (\frac{1}{\sigma^2} X^T y + \Sigma_0^{-1} \boldsymbol{\mu}_0). && (8.31) \end{aligned}$$

The above can be shown by completing the square for (8.28). We can now notice that this is an unnormalized Gaussian density for $\boldsymbol{\theta}$ with mean and variance given by (8.31) and (8.30), or

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}_d, \Sigma_d). \tag{8.32}$$

This result is quite remarkable: we have incorporated our measurements to give a full posterior distribution over our model parameters, as opposed to a simple point estimate. However, for applications in decision-making and control, we care about the *predictive distribution*, or $p(y^* \mid \boldsymbol{x}, \mathcal{D})$ where we write $\boldsymbol{x}^*, y^*$ to denote *test points*, or an arbitrary prediction problem given the training dataset. To derive the posterior predictive, we will turn to following identity.

Lemma 8.1.1. *Let* $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{x}}, \Sigma_{\boldsymbol{x}})$ *and* $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{y}}, \Sigma_{\boldsymbol{y}})$, *with* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *independent. Then* $A\boldsymbol{x} + B\boldsymbol{y} \sim \mathcal{N}(A\boldsymbol{\mu}_{\boldsymbol{x}} + B\boldsymbol{\mu}_{\boldsymbol{y}}, A\Sigma_{\boldsymbol{x}}A^T + B\Sigma_{\boldsymbol{y}}B^T)$

Proof. See [PP], section 8. □

Given this result, note that

$$y^* = \boldsymbol{\theta}^T \boldsymbol{x}^* + \epsilon \tag{8.33}$$

where $\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \mid \mathcal{D})$ with $\boldsymbol{\theta}^T$ and ϵ independent. Thus, applying the above lemma, we have

$$p(y \mid \boldsymbol{x}, \mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}_d^T \boldsymbol{x}^*, \boldsymbol{x}^{*,T} \Sigma_d^{-1} \boldsymbol{x}^* + \sigma^2). \tag{8.34}$$

Therefore, given the Gaussian model assumption, we can exactly compute the posterior predictive distribution.

Maximum a Posteriori (MAP) Estimation

A common approach to incorporate prior information into parameter estimation without performing fully Bayesian inference is to compute the *maximum a Posteriori* (MAP) parameter estimate, defined as

$$\hat{\boldsymbol{\theta}}_{MAP} = \text{argmax}_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \mathcal{D}) = \text{argmax}_{\boldsymbol{\theta}} p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \tag{8.35}$$

where we drop the marginal likelihood from the denominator because it does not effect the maximization (and because we do not need to compute a probability density and thus we do not have the requirement that the output integrates to one). Examining (8.35), we can see that it corresponds to the maximum likelihood estimate, with an added term representing the prior belief. As such, it allows a practitioner to include prior information without the computational challenges associated with general Bayesian inference. Indeed, as the prior becomes flatter—as all values of the parameter become equally likely under the prior—the MAP estimator approaches the max likelihood estimator.

Examining the MAP estimator for the Bayesian least squares problem, we have

$$\hat{\boldsymbol{\theta}}_{MAP} = \boldsymbol{\mu}_d \tag{8.36}$$

as the max of a Gaussian is the mean. Let us choose a prior with mean $\boldsymbol{\mu}_0 = 0$ and variance $\Sigma_0 = I$ and fix the noise variance as $\sigma^2 = \lambda$. Then, the MAP estimator is

$$\hat{\boldsymbol{\theta}}_{MAP} = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y} \tag{8.37}$$

which is exactly the ridge regression (or L_2 regularized) least squares estimator. This gives us some insight into why a positive choice of λ typically outperforms the least norm solution: the ridge regression approach imposes an implicit prior on the value of $\boldsymbol{\theta}$.

8.2 Gaussian Processes

We have discussed an approach to nonlinear regression using fixed basis functions. In this section, we will discuss Gaussian process regression, as well as the (strong) connections to Bayesian linear regression. Gaussian process regression, instead of performing inference over a finite collection of weights as in the previous section, performs inference directly over *functions.* We previously addressed the regression problem via a *parametric* approach in which we fixed our model to be a linear function of some (possibly nonlinearly transformed) features. In contrast, Gaussian process regression instead performs *nonparametric* inference, in which the class of models is not specified via a finite set of parameters. We will begin by defining a Gaussian process

Definition 8.2.1 ([Ras03])**.** *A Gaussian process is a time-continuous stochastic process such that for every finite collection of time indices $t_1, \dots, t_k$, the collection if random variables $y_{t_1}, \dots, y_{t_k}$ is jointly Gaussian.*

A Gaussian process $f(\boldsymbol{x})$ is fully specified by its mean

$$\mu(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \tag{8.38}$$

and covariance

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}[(f(\boldsymbol{x}) - \mu(\boldsymbol{x}))(f(\boldsymbol{x}') - \mu(\boldsymbol{x}'))]. \tag{8.39}$$

Thus, the behavior of the GP regression model is governed entirely by these quantities. The mean is typically assumed to be zero, although this is not necessary. We refer to $k(\cdot, \cdot)$ as the *kernel function.* We require the kernel function to be *symmetric*:

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x}), \tag{8.40}$$

as well as *positive definite*, or

$$K = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_k) \\ \vdots & & \vdots \\ k(\boldsymbol{x}_k, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_k, \boldsymbol{x}_k) \end{bmatrix} \succ 0 \tag{8.41}$$

for any $\boldsymbol{x}_1, \dots, \boldsymbol{x}_k$. These conditions are intuitive: they guarantee that the variance matrix associated with the GP for an arbitrary set of inputs is well-defined. A kernel satisfying these properties is a *Mercer* or positive definite kernel.

As an example of a simple Gaussian process model, let us consider Bayesian linear regression on features $\phi(\boldsymbol{x})$. Fixing a prior $\boldsymbol{\theta} \sim \mathcal{N}(0, \Sigma_0)$, we have

$$\mu(\boldsymbol{x}) = \mathbb{E}[\boldsymbol{\theta}^T \phi(\boldsymbol{x})] = 0 \tag{8.42}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{\theta}\boldsymbol{\theta}^T] \phi(\boldsymbol{x}) = \phi(\boldsymbol{x})^T \Sigma_0 \phi(\boldsymbol{x}) \tag{8.43}$$

As we will see later, using the machinery of kernel GP regression to perform Bayesian linear regression with a pre-specified set of features is inefficient. However, let us consider the squared exponential kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{1}{2\ell} \|\boldsymbol{x} - \boldsymbol{x}'\|_2^2) \tag{8.44}$$

where ℓ is a length scale hyperparameter. In this case, the set of basis functions resulting in this kernel is actually infinite dimensional. Indeed, every Mercer kernel corresponds to a (potentially) infinite dimensional set of features, and the choice of kernel allows intuitive control of how the regression model behaves, and implies a distribution over functions.

To see how we can use a GP regression model to make predictions, we will rely on the following result.

Lemma 8.2.2 ([Ras03])**.** *Let $\boldsymbol{y}_1, \boldsymbol{y}_2$ be jointly Gaussian such that*

$$\begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}). \tag{8.45}$$

Then,

$$p(\boldsymbol{y}_1 \mid \boldsymbol{y}_2) = \mathcal{N}(\boldsymbol{\mu}_1 + \Sigma_{21}\Sigma_{22}^{-1}(\boldsymbol{y}_2 - \boldsymbol{\mu}_2), \Sigma_{11} - \Sigma_{21}\Sigma_{22}^{-1}\Sigma_{12}) \tag{8.46}$$

We will assume a model of the form $y = f(\boldsymbol{x}) + \epsilon$, and we will write a test point as $\boldsymbol{x}^*$. Then, given training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^d$, we note that

$$\begin{bmatrix} f(\boldsymbol{x}^*) \\ \boldsymbol{y} \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x}^*, \boldsymbol{x}^*) & K(X, \boldsymbol{x}^*) \\ K(\boldsymbol{x}^*, X) & K(X, X) + \sigma^2 I \end{bmatrix}). \tag{8.47}$$

where

$$K(\boldsymbol{x}^*, X) = \begin{bmatrix} k(\boldsymbol{x}^*, \boldsymbol{x}_1) \\ \vdots \\ k(\boldsymbol{x}^*, \boldsymbol{x}_d) \end{bmatrix} \tag{8.48}$$

and

$$K(X, X) = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_d) \\ \vdots & & \vdots \\ k(\boldsymbol{x}_d, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_d, \boldsymbol{x}_d) \end{bmatrix}. \tag{8.49}$$

Thus, applying Lemma 8.2.2, we can see that $p(f(\boldsymbol{x}^*) \mid \mathcal{D})$ is Gaussian with mean

$$\mathbb{E}[f(\boldsymbol{x}^*)] = K(\boldsymbol{x}^*, X)(K(X, X) + \sigma^2 I)^{-1}\boldsymbol{y} \tag{8.50}$$

and variance

$$\text{var}(f(\boldsymbol{x}^*)) = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - K(\boldsymbol{x}^*, X)(K(X, X) + \sigma^2 I)^{-1}K(X, \boldsymbol{x}^*). \tag{8.51}$$

From this, using the fact that the sum of independent Gaussians is Gaussian, the predictive variance is

$$\text{var}(y^*) = \text{var}(f(\boldsymbol{x}^*)) + \sigma^2. \tag{8.52}$$

and the predictive mean is $\mathbb{E}[y^*] = \mathbb{E}[f(\boldsymbol{x}^*)]$. Given these update rules, we have the following procedure for performing regression with Gaussian process models. First, we select a kernel function (there are many to choose from; see [Ras03] or [Mur12] for a discussion of possible choices). Then, we compute the kernel matrices $k(\boldsymbol{x}^*, \boldsymbol{x}^*)$, $K(\boldsymbol{x}^*, X)$, and $K(X, X)$. Finally, we use the conditioning equations above to compute our posterior predictive distribution.

The choice of kernel can result in much more expressive models than are easily designed with standard Bayesian linear regression. Moreover, whereas the expressivity of a standard

Bayesian linear regression model is limited by the finite set of parameters, the function representation of the GP can be infinitely improved—indeed, kernel GP regression can asymptotically approximate any continuous function. So, are GP regression models always better than standard Bayesian linear regression? For small datasets, they perform quite well and are a strong choice. However, for larger datasets, the computational performance suffers. Note that the posterior inference procedure requires inverting the matrix $K(X, X)$ (plus an identity term), which is $d \times d$. Matrix inversion has (practically) complexity $O(d^3)$, and so this inversion step can be prohibitively slow for large training datasets.

8.3 Neural Networks

Our discussion has so far focused on models that are linear in a pre-specified set of features. In the case of standard or Bayesian linear regression, these features were chosen explicitly. In the case of Gaussian process regression, the system designer chose a kernel function which induced a set of features. The previously presented models are useful for a reason: they are simply, effective, and training the model (or computing the posterior) corresponds to either a closed-form solution or to a convex optimization problem. However, these methods come with the limitation that we must explicitly choose our features, and this is a difficult task in itself.

In this section, we will discuss neural network models. These models are composed of the composition of simple nonlinear transformations of the inputs. As such, neural network models can be viewed as nonlinear function parameterizations. Indeed, neural networks are an expressive and general function parameterization. However, the optimization problem associated with choosing the parameters of these nonlinear models results in a highly non-convex optimization problem. As a consequence, neural network models largely fell out of favor compared to convex models. In the early 2010s, extremely strong empirical performance on image recognition benchmarks brought these models back to the forefront of both research and commercial application [KSH12]. Currently, they are the predominant regression model for nonlinear system identification and reinforcement learning. While neural networks are an extremely broad topic, and one that is growing rapidly, we will focus in this section on a simple, minimal discussion of neural network models. For a deeper discussion, we refer the reader to [GBC16].

8.3.1 The Perceptron

Before introducing a full neural network model, we will begin with introducing the minimal unit of the network, the *perceptron*. Note that while we use the term perceptron to a general hidden unit, it is often frequently used to discuss the perceptron learning algorithm which we will not discuss. The perceptron is defined as

$$h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) \tag{8.53}$$

where $\sigma(\cdot)$ is a nonlinear function, typically referred to as a threshold function, and $\boldsymbol{w}$ is a vector of weights. Classically, the threshold function is one that (approximately) maps to 1 if $\boldsymbol{w}^T\boldsymbol{x} > 0$, and 0 otherwise. This hard thresholding is typically relaxed to a soft threshold to result in a smooth gradient. A widespread threshold function is the logistic function

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{8.54}$$

This nonlinearity leads to a simple binary classifier, where the weights $\boldsymbol{w}$ result in a linear decision boundary. While nonlinearities corresponding to binary classifiers were originally dominant, the ReLU (rectified linear unit) nonlinearity

$$\sigma(x) = \begin{cases} x \text{ for } x > 0 \\ 0 \text{ otherwise} \end{cases} \tag{8.55}$$

has gained substantial popularity recently due to good empirical performance.

8.3.2 Feed-Forward Neural Networks

By combining perceptrons (or hidden units), we arrive at the feed-forward neural network. The term feed-forward refers to the fact that the hidden units will be stacked in a graph without loops or temporal dependency, as is typical in recurrent networks, and will not include features other than the previously described simple nonlinear hidden units, as in for example convolutional networks. We will refer to a *hidden layer* as a nonlinear function of the form

$$\boldsymbol{h} = \sigma(W\boldsymbol{x}) \tag{8.56}$$

where σ is applied element-wise. The simplest feed-forward neural network is one with a single hidden layer

$$\boldsymbol{h} = \sigma(W_1\boldsymbol{x}) \tag{8.57}$$

$$\boldsymbol{y} = W_2\boldsymbol{h} \tag{8.58}$$

where W_1, W_2 are weight matrices. This provides a nonlinear function parameterization. This can be combined with any loss function—for regression, the most common choice is the mean squared error loss function that we have discussed previously. This neural network is non-convex, and thus simple gradient descent algorithms can not provide guarantees on finding the global minima. However, in practice and combined with several effective training techniques including regularization schemes, these models perform well despite the non-convexity of the associated optimization problem. While we've considered only single hidden layer networks so far, we can also consider *deep* neural network models with multiple hidden layers; for example of the form

$$\boldsymbol{h}_1 = \sigma(W_1\boldsymbol{x}) \tag{8.59}$$

$$\boldsymbol{h}_2 = \sigma(W_2\boldsymbol{h}_1) \tag{8.60}$$

$$\boldsymbol{y} = W_3\boldsymbol{h}_2. \tag{8.61}$$

This increased depth allows representation of increasingly complex features.

Chapter 9

System Identification

9.1 Linear System Identification

9.1.1 Persistent Excitation

9.1.2 Linear Systems with Observations

9.2 Nonlinear System Identification

9.3 Further Reading

Chapter 10

Adaptive Control and Adaptive Optimal Control

10.1 Adaptive Control

We will now discuss adaptive control, which (broadly speaking) aims to perform online adaptation of the control policy to improve performance. The formulation of the adaptive control problem is typically not in terms of optimal adaptive control. Typically, the designer of an adaptive control system will place more emphasis on proving the combined stability of the controller and the adaptive process than on minimizing a cost function. While in the system identification setting we were concerned with model identification in particular, work on adaptive control investigates both model adaptation as well as controller adaptation, and combinations. Indeed, adaptive control encompasses a wide variety of techniques. In [ÅW13], the authors define an adaptive controller as "a controller with adjustable parameters and a mechanism for adjusting the parameters". Examples of adaptive control strategies include

- Adaptive pole placement or policy adaptation
- Iterative learning control (ILC)
- Gain scheduling
- Model reference adaptive control (MRAC)
- Model identification adaptive control (MIAC)
- Dual control strategies

though we will focus primarily on the last three.

10.1.1 Model Reference Adaptive Control (MRAC)

We will now introduce model reference adaptive control, which may be interpreted as a combined model-based and model-free adaptive control scheme. While it leverages a model, this model is responsible only for generating a reference output to track, and control adaptation is done directly via updating the policy. Despite stating our desired adaptive control problem in discrete time, we will describe the MRAC approach in continuous time, which is the more standard setting. A model reference adaptive controller is composed of four parts:

1. A plant containing unknown parameters
2. A *reference model* for compactly specifying the desired output
3. A feedback control law for containing adjustable parameters
4. An adaptation mechanism for updating the adjustable parameters

The reference model is used to provide the ideal plant response which the adaptation mechanism should aim to achieve. Choice of reference model is therefore an art, similar to choice of a cost function. However, practically, it is more difficult than choice of a cost function. A cost function designer for an optimal control or reinforcement learning system aims to reflect their desired performance characteristics is a cost function that yields a tractable optimization problem. The designer of an MRAC system, on the other hand, must choose a model that achieves good performance. As such, an MRAC system designer is implicitly solving a policy optimization problem to choose a reference model. Practically, choice of these models is simplified by considering linear systems, and thus one may specify performance in terms of relatively simple characteristics such as damping.

One of the simplest approaches to MRAC is the so-called MIT rule. Let $\hat{\boldsymbol{y}}(t)$ denote the reference signal we aim to track, and $\boldsymbol{y}(t)$ denote the output of the system as a result of controller π, parameterized by $\boldsymbol{\theta}$. We will write specify the error $e(t) = \|\hat{\boldsymbol{y}}(t) - \boldsymbol{y}(t)\|_2$, as well as $J(t) = \frac{1}{2}e(t)^2$. Then, the MIT rule consists of taking gradient updates of the form

$$\dot{\boldsymbol{\theta}} = -\gamma \frac{\partial J}{\partial \boldsymbol{\theta}} = -\gamma e(t) \frac{\partial e}{\partial \boldsymbol{\theta}}(t) \tag{10.1}$$

where γ is a gain parameter governing update rate. This rule may be applied in discrete or continuous time (via an update difference equation or differential equation), and the joint dynamics of the system and the control parameter $\boldsymbol{\theta}$ may be analyzed for stability, typically with tools from Lyapunov stability theory. For a more complete discussion on MRAC and design of MRAC systems using Lyapunov theory, we refer the reader to [ÅW13].

10.1.2 Model Identification Adaptive Control (MIAC)

Model identification adaptive control is the adaptive control scheme that logically follows from system identification: we will concurrently perform parameter estimation while controlling the system by using our estimate of the parameters. We will refer to our estimate of the model parameters as $\hat{\boldsymbol{\theta}}$. MIAC designs a control scheme that takes $\hat{\boldsymbol{\theta}}$ as an input in addition to the state $\boldsymbol{x}$.

An important distinction within MIAC schemes is between *certainty-equivalent* controllers and so-called *cautious* controllers. Certainty-equivalent approaches, like certainty-equivalence in the LQG setting, have the policy as a function only of a point estimate of the estimated parameters. In the LQG setting, in which the state is estimated by Kalman filter, it can be shown that certainty-equivalent control performs equivalently to a control scheme incorporating other statistics of the state estimate. However, this principle does not in general hold, and thus certainty-equivalence in adaptive control is often a design choice to make stability analysis tractable. Cautious controllers, on the other hand, incorporate other information about the estimate of the parameters. For example, in the Bayesian setting, the posterior density of the parameters may be passed to the controller. This approach is known as "cautious" as it explicitly factors the uncertainty of the parameter estimate into the control decision, and thus will be more robust when uncertainty is high. However, because we are operating within the passively adaptive control setting, cautious methods will not include the expected uncertainty reduction due to future information, and thus may be overly conservative.

10.1.3 Linear Quadratic Adaptive Control

One of the earliest works on adaptive control in the linear dynamics, quadratic cost setting was Simon [Sim56], who proposed using certainty-equivalence in the model estimation. His approach, however—to utilize the mean parameter estimates combined with an LQR controller—can converge to incorrect values with non-zero probability [BKW85, AYS11]. Thus, it is necessary to augment the control strategy with heuristic approaches to actively explore the environment. A simple example exploration strategy is so-called ϵ-greedy exploration, in which a random action is taken with probability ϵ, and the system otherwise follows the certainty-equivalent optimal controller. While this performs reasonably well in the linear setting (where the definition of "reasonably well" may perhaps be debated), it perfoms poorly in nonlinear MDPs and discrete MDPs due to the highly "local" nature of the exploration [MLJA15, OVRW16]. We expand on the problem of exploration in the next section.

10.2 Probing, Planning for Information Gain, and Dual Control

Thus, we can augment our state with some statistics of our estimate (which we will write $\hat{\boldsymbol{\theta}}$) to yield hyperstate $\boldsymbol{y}$, and augment our dynamics with the dynamics of the estimation process. To solve the Bellman dynamic programming problem for this hyperstate is dual control.

The difference between these two approaches may seem subtle, but that former approach will result in a control scheme that optimally identifies the parameters for the purposes of control, whereas the latter approach will only passively adaptive the parameter estimate, and act with respect to that passive estimate.

10.3 Further Reading

Chapter 11

Model-free Reinforcement Learning

11.1 Temporal Difference Learning

11.1.1 Q-Learning

11.1.2 SARSA

11.1.3 Q-Learning with Function Approximation

11.2 Policy Gradient Methods

11.2.1 The Likelihood Ratio Trick and REINFORCE

11.2.2 Policy Gradient with Deep Representations

11.3 Actor-Critic Methods

11.4 Further Reading

Chapter 12

Model-based Reinforcement Learning

12.1 Adaptive and Learning MPC

12.2 Combining Model and Policy Learning

12.3 Further Reading

Bibliography

[Alt99] Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.

[AM07] Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

[ÅW13] Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013.

[AYS11] Yasin Abbasi-Yadkori and Csaba Szepesvári. Regret bounds for the adaptive control of linear quadratic systems. *Conference on Learning Theory (COLT)*, 2011.

[BBM17] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[Ber12] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Number 1. 4 edition, 2012.

[Ber16] Dimitri P Bertsekas. *Nonlinear programming*. Athena Scientific, 2016.

[BH75] Arthur Bryson and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation and Control*. CRC Press, 1975.

[BKW85] Arthur Becker, P Kumar, and Ching-Zong Wei. Adaptive control with the stochastic approximation algorithm: Geometry and convergence. *IEEE Transactions on Automatic Control*, 1985.

[Bre10] Alberto Bressan. Noncooperative differential games. a tutorial. 2010.

[BT97] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*. Athena Scientific, 1997.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[FHT08] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 2. 2008.

[GB17] Markus Giftthaler and Jonas Buchli. A projection approach to equality constrained iterative linear quadratic optimal control. In *IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, 2017.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[IS12] Petros A Ioannou and Jing Sun. *Robust adaptive control*. Courier Corporation, 2012.

[JM70] David Jacobson and David Mayne. *Differential Dynamic Programming*. Elsevier, 1970.

[Kel17a] Matthew Kelly. An introduction to trajectory optimization: how to do your own direct collocation. *SIAM Review*, 2017.

[Kel17b] Matthew Kelly. Transcription methods for trajectory optimization: a beginners tutorial. *arXiv:1707.00284*, 2017.

[Kir12] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2012.

[Kol08] Zico Kolter. Convex optimization overview. *CS 229 Lecture Notes*, 2008.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, 2012.

[LaV06] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[LK14] Sergey Levine and Vladlen Koltun. Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning (ICML)*, 2014.

[LM67] Ernest Bruce Lee and Lawrence Markus. Foundations of optimal control theory. 1967.

[LS92] Li-zhi Liao and Christine A Shoemaker. Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems. Technical report, Cornell University, 1992.

[LT04] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *International Conference on Informatics in Control, Automation, and Robotics*, 2004.

[MBT05] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 2005.

[MLJA15] Teodor Mihai Moldovan, Sergey Levine, Michael I Jordan, and Pieter Abbeel. Optimism-driven exploration for nonlinear systems. *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[OVRW16] Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. *International Conference on Machine Learning (ICML)*, 2016.

[Pow12] Warren B Powell. AI, OR and control theory: A rosetta stone for stochastic optimization. 2012.

[PP] Kaare Brandt Petersen and Michael Syskind Pedersen. The matrix cookbook.

[Ras03] Carl Edward Rasmussen. Gaussian processes in machine learning. Springer, 2003.

[RMD17] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model Predictive Control: Theory, Computation, and Design.* Nob Hill Publishing, 2017.

[Sim56] Herbert A Simon. Dynamic programming under uncertainty with a quadratic criterion function. *Econometrica*, 1956.

[TET12] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[TL05] Emanuel Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference (ACC)*, 2005.

[TMT14] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[XLH17] Zhaoming Xie, C Karen Liu, and Kris Hauser. Differential dynamic programming with nonlinear constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.