Optimal Control and Estimation

Heng Yang

2023-09-12

Contents

Preface

This is the textbook for Harvard ES/AM 158: Introduction to Optimal Control and Estimation. Information about the offerings of the class is listed below.

2023 Fall

Time: Mon/Wed 2:15 - 3:30pm

Location: Science and Engineering Complex, 1.413

Instructor: Heng Yang

Teaching Fellow: Weiyu Li

Syllabus

Chapter 1

The Optimal Control Formulation

1.1 The Basic Problem

Consider a discrete-time dynamical system

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \dots, N-1 \tag{1.1}$$

where

- $x_k \in \mathbb{X} \subseteq \mathbb{R}^n$ is the *state* of the system,
- $u_k \in \mathbb{U} \subseteq \mathbb{R}^m$ is the *control* we wish to design,
- $w_k \in \mathbb{W} \subseteq \mathbb{R}^p$ a random *disturbance* or noise (e.g., due to unmodelled dynamics) which is described by a probability distribution $P_k(\cdot \mid x_k, u_k)$ that may depend on x_k and u_k but not on prior disturbances $w_0, \dots, w_{k-1}$,
- k indexes the discrete time,
- N denotes the horizon,
- f_k models the transition function of the system (typically $f_k \equiv f$ is time-invariant, especially for robotics systems; we use f_k here to keep full generality).

Remark (Deterministic v.s. Stochastic). When $w_k \equiv 0$ for all k, we say the system (1.1) is *deterministic*; otherwise we say the system is *stochastic*. In the following we will deal with the stochastic case, but most of the methodology should carry over to the deterministic setup.

We consider the class of *controllers* (also called *policies*) that consist of a sequence of functions

$$\pi = \{\mu_0, \dots, \mu_{N-1}\},$$

where $\mu_k(x_k) \in \mathbb{U}$ for all x_k, i.e., μ_k is a *feedback* controller that maps the state to an admissible control. Given an initial state x_0 and an admissible policy π, the state *trajectory* of the system is a sequence of random variables that evolve according to

$$x_{k+1} = f_k(x_k, \mu_k(x_k), w_k), \quad k = 0, \dots, N-1 \tag{1.2}$$

where the randomness comes from the disturbance w_k.

We assume the state-control trajectory $\{u_k\}_{k=0}^{N-1}$ and $\{x_k\}_{k=0}^{N}$ induce an *additive cost*

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \tag{1.3}$$

where $g_k, k = 0, \dots, N$ are some user-designed functions.

With (1.2) and (1.3), for any admissible policy π, we denote its induced *expected cost* with initial state x_0 as

$$J_\pi(x_0) = \mathbb{E}\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k)) \right\}, \tag{1.4}$$

where the expectation is taken over the randomness of w_k.

Definition 1.1 (Discrete-time, Finite-horizon Optimal Control)**.** Find the best admissible controller that minimizes the expected cost in (1.4)

$$\pi^\star \in \arg\min_{\pi \in \Pi} J_\pi(x_0), \tag{1.5}$$

where Π is the set of all admissible controllers. The cost attained by the optimal controller, i.e., $J^\star = J_{\pi^\star}(x_0)$ is called the optimal *cost-to-go*, or the optimal *value function*.

Remark (Open-loop v.s. Closed-loop). An important feature of the basic problem in Definition 1.1 is that the problem seeks *feedback policies*, instead of numerical values of the controls, i.e., $u_k = \mu_k(x_k)$ is in general a function of the state x_k. In other words, the controls are executed sequentially, one at a time after observing the state at each time. This is called closed-loop control, and is in general better than open-loop control

$$\min_{u_0, \dots, u_{N-1}} \mathbb{E}\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \right\}$$

where all the controls are planned at $k = 0$. Intuitively, a closed-loop policy is able to utilize the extra information received at each timestep (i.e., it observes x_{k+1} and hence also observes the disturbance w_k) to obtain a lower cost than

an open-loop controller. Example 1.2.1 in (Bertsekas, 2012) gives a concrete application where a closed-loop policy attains a lower cost than an open-loop policy.

In deterministic control (i.e., when $w_k \equiv 0, \forall k$), however, a closed-loop policy has no advantage over an open-loop controller. This is obvious because at $k = 0$, even the open-loop controller predicts perfectly the consequences of all its actions and there is no extra information to be observed at later time steps. In fact, even in stochastic problems, a closed-loop policy may not be advantageous, see Exercise 1.27 in (Bertsekas, 2012).

1.2 Dynamic Programming and Principle of Optimality

We now introduce a general and powerful algorithm, namely *dynamic programming* (DP), for solving the optimal control problem 1.1. The DP algorithm builds upon a quite simple intuition called the *Bellman principle of optimality.*

Theorem 1.1 (Bellman Principle of Optimality)**.** *Let $\pi^\star = \{\mu_0^\star, \mu_1^\star, \dots, \mu_{N-1}^\star\}$ be an optimal policy for the optimal control problem 1.1. Assume that when using $\pi^\star$, a given state x_i occurs at timestep i with positive probability (i.e., x_i is reachable at time i).*

Now consider the following subproblem where we are at x_i at time i and wish to minimize the cost-to-go from time i to time N

$$\min_{\mu_i,\dots,\mu_{N-1}} \mathbb{E}\left\{ g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k(x_k)) \right\}.$$

Then the truncated policy $\{\mu_i^\star, \mu_{i+1}^\star, \dots, \mu_{N-1}^\star\}$ must be optimal for the subproblem.

Theorem 1.1 can be proved intuitively by contradiction: if the truncated policy $\{\mu_i^\star, \mu_{i+1}^\star, \dots, \mu_{N-1}^\star\}$ is not optimal for the subproblem, say there exists a different policy $\{\mu_i', \mu_{i+1}', \dots, \mu_{N-1}'\}$ that attains a lower cost for the subproblem starting at x_i at time i. Then the combined policy $\{\mu_0^\star, \dots, \mu_{i-1}^\star, \mu_i', \dots, \mu_{N-1}'\}$ must attain a lower cost for the original optimal control problem 1.1 due to the additive cost structure, contradicting the optimality of $\pi^\star$.

The Bellman principle of optimality is more than just a principle, it is also an algorithm. It suggests that, to build an optimal policy, one can start by solving the last-stage subproblem to obtain $\{\mu_{N-1}^\star\}$, and then proceed to solve the subproblem containing the last two stages to obtain $\{\mu_{N-2}^\star, \mu_{N-1}^\star\}$. The recursion continues until optimal policies at all stages are computed. The following theorem formalizes this concept.

Theorem 1.2 (Dynamic Programming)**.** *The optimal value function $J^\star(x_0)$ of the optimal control problem 1.1 (starting from any given initial condition x_0) is equal to $J_0(x_0)$, which can be computed backwards and recursively as*

$$J_N(x_N) = g_N(x_N) \tag{1.6}$$

$$J_k(x_k) = \min_{u_k \in \mathbb{U}} \mathbb{E}_{w_k \sim P_k(\cdot|x_k,u_k)} \left\{ g_k(x_k,u_k) + J_{k+1}(f_k(x_k,u_k,w_k)) \right\}, \ k = N-1, \dots, 1, 0. \tag{1.7}$$

Moreover, if $u_k^\star = \mu_k^\star(x_k)$ is a minimizer of (1.7) *for every x_k, then the policy $\pi^\star = \{\mu_0^\star, \dots, \mu_{N-1}^\star\}$ is optimal.*

Proof. For any admissible policy $\pi = \{\mu_0, \dots, \mu_{N-1}\}$, denote $\pi^k = \{\mu_k, \dots, \mu_{N-1}\}$ the last-$(N-k)$-stage truncated policy. Consider the subproblem consisting of the last $N-k$ stages starting from x_k, and let $J_k^\star(x_k)$ be its optimal cost-to-go. Mathematically, this is

$$J_k^\star(x_k) = \min_{\pi^k} \mathbb{E}_{w_k,\dots,w_{N-1}} \left\{ g_N(x_N) + \sum_{i=k}^{N-1} g_i(x_i, \mu_i(x_i)) \right\}, \quad k = 0, 1, \dots, N-1. \tag{1.8}$$

We define $J_N^\star(x_N) = g(x_N)$ for $k = N$.

Our goal is to prove the $J_k(x_k)$ computed by dynamic programming from (1.7) is equal to $J_k^\star(x_k)$ for all $k = 0, \dots, N$. We will prove this by induction.

Firstly, we already have $J_N^\star(x_N) = J_N(x_N) = g(x_N)$, so $k = N$ holds automatically.

Now we assume $J_{k+1}^\star(x_{k+1}) = J_{k+1}(x_{k+1})$ for all x_{k+1}, and we wish to induce

$J_k^\star(x_k) = J_k(x_k)$. To show this, we write

$$J_k^\star(x_k) = \min_{\pi^k} \mathbb{E}_{w_k,\dots,w_{N-1}} \left\{ g_N(x_N) + \sum_{i=k}^{N-1} g_i(x_i, \mu_i(x_i)) \right\} \tag{1.9}$$

$$= \min_{\mu_k, \pi^{k+1}} \mathbb{E}_{w_k,\dots,w_{N-1}} \left\{ g_k(x_k, \mu_k(x_k)) + g_N(x_N) + \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i)) \right\} \tag{1.10}$$

$$= \min_{\mu_k} \left[\min_{\pi^{k+1}} \mathbb{E}_{w_k,\dots,w_{N-1}} \left\{ g_k(x_k, \mu_k(x_k)) + g_N(x_N) + \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i)) \right\} \right] \tag{1.11}$$

$$= \min_{\mu_k} \mathbb{E}_{w_k} \left\{ g_k(x_k, \mu_k(x_k)) + \min_{\pi^{k+1}} \left[\mathbb{E}_{w_{k+1},\dots,w_{N-1}} \left\{ g_N(x_N) + \sum_{i=k+1}^{N-1} g_i(x_i, \mu_i(x_i)) \right\} \right] \right\} \tag{1.12}$$

$$= \min_{\mu_k} \mathbb{E}_{w_k} \left\{ g_k(x_k, \mu_k(x_k)) + J_{k+1}^\star(f_k(x_k, \mu_k(x_k), w_k)) \right\} \tag{1.13}$$

$$= \min_{\mu_k} \mathbb{E}_{w_k} \left\{ g_k(x_k, \mu_k(x_k)) + J_{k+1}(f_k(x_k, \mu_k(x_k), w_k)) \right\} \tag{1.14}$$

$$= \min_{u_k \in \mathbb{U}} \mathbb{E}_{w_k} \left\{ g_k(x_k, \mu_k(x_k)) + J_{k+1}(f_k(x_k, \mu_k(x_k), w_k)) \right\} \tag{1.15}$$

$$= J_k(x_k), \tag{1.16}$$

where (1.9) follows from definition (1.8); (1.10) expands $\pi^k = \{\mu_k, \pi^{k+1}\}$ and $\sum_{i=k}^{N-1} g_i = g_k + \sum_{i=k+1}^{N-1}$; (1.11) writes the joint minimization over μ_k and π^{k+1} as equivalently first minimizing over π^{k+1} and then minimizing over μ_k; (1.12) is the key step and holds because g_k and w_k depend only on μ_k but not on π^{k+1}; (1.13) follows again from definition (1.8) with k replaced by $k+1$; (1.14) results from the induction assumption; (1.15) clearly holds because any $\mu_k(x_k)$ belongs to $\mathbb{U}$ and any element in $\mathbb{U}$ can be chosen by a feedback controller μ_k; and lastly (1.16) follows from the dynamic programming algorithm (1.7).

By induction, this shows that $J_k^\star(x_k) = J_k(x_k)$ for all $k = 0, \dots, N$. □

The careful reader, especially from a robotics background, may soon become disappointed when seeing the DP algorithm (1.7) because it is rather conceptual than practical. To see this, we only need to run DP for $k = N-1$:

$$J_{N-1}(x_{N-1}) = \min_{u_{N-1} \in \mathbb{U}} \mathbb{E}_{w_{N-1}} \left\{ g_{N-1}(x_{N-1}, u_{N-1}) + J_N(f_{N-1}(x_{N-1}, u_{N-1}, w_{N-1})) \right\}. \tag{1.17}$$

Two challenges immediately show up:

- How to perform the minimization over u_{N-1} when $\mathbb{U}$ is a continuous constraint set? Even if we assume g_{N-1} is convex[1] in u_{N-1}, J_N is convex in

[1] You may want to read Appendix B if this is your first time seeing "convex" things.

x_N, and the dynamics f_{N-1} is also convex in u_{N-1} (so that the optimization (1.17) is convex), we may be able to solve the minimization *numerically* for each x_{N-1} using a convex optimization solver, but rarely will we be able to find an analytical policy $\mu^\star_{N-1}$ such that $u^\star_{N-1} = \mu^\star_{N-1}(x_{N-1})$ for every x_{N-1} (i.e., the optimal policy $\mu^\star_{N-1}$ is implict but not explict).

- Suppose we can find an anlytical optimal policy $\mu^\star_{N-1}$, say $\mu^\star_{N-1} = Kx_{N-1}$ a linear policy, how will plugging $\mu^\star_{N-1}$ into (1.17) affect the complexity of $J_{N-1}(x_{N-1})$? One can see that even if $\mu^\star_{N-1}$ is linear in x_{N-1}, J_{N-1} may be highly nonlinear in x_{N-1} due to the composition with g_{N-1}, f_{N-1} and J_N. If $J_{N-1}(x_{N-1})$ becomes too complex, then clearly it becomes more challenging to perform (1.17) for the next step $k = N-2$.

Due to these challenges, only in a very limited amount of cases will we be able to perform *exact dynamic programming*. For example, when the state space $\mathbb{X}$ and control space $\mathbb{U}$ are discrete, we can design efficient algorithms for exact DP. For another example, when the dynamics f_k is linear and the cost g_k is quadratic, we will also be able to compute $J_k(x_k)$ in closed form (though this sounds a bit surprising!). We will study these problems in more details in Chapter 2.

For general optimal control problems with continuous state space and control space (and most problems we care about in robotics), unfortunately, we will have to resort to *approximate dynamic programming*, basically variations of the DP algorithm (1.7) where approximate value functions $J_k(x_k)$ and/or control policies $\mu_k(x_k)$ are used (e.g., with neural networks and machine learning).[2] We will introduce several popular approximation schemes in Chapter 3. We will see that, although exact DP is not possible anymore, the Bellman principle of optimality still remains one of the most important guidelines for designing approximation algorithms. Efficient algorithms for approximate dynamic programming, preferrably with performance guarantees, still remain an active area of research.

1.3 Infinite-horizon Formulation

So far we are focusing on problems with a finite horizon N, what if the horizon N tends to infinity?

In particular, consider the controller π now contains an infinite sequence of functions

$$\pi = \{\mu_0, \dots\}$$

and let us try to find the best policy that minimizes the cost-to-go starting from x_0 subject to the same dynamics as in (1.1) (with N tends to infinity and

[2] Another possible solution is to discretize continuous states and controls. However, when the dimension of state and control is high, discretization becomes too expensive in terms of memory and computational complexity.

$f_k \equiv f$)

$$J_\pi(x_0) = \mathbb{E}\left\{\sum_{k=0}^{\infty} g(x_k, \mu_k(x_k))\right\}, \tag{1.18}$$

where the expectation is taken over the (infinite number of) disturbances $\{w_0, \dots\}$.

We can write (1.18) equivalently as

$$J_\pi(x_0) = \lim_{N\to\infty} J_\pi^N(x_0),$$

where, with a slight abuse of notation, $J_\pi^N(x_0)$ is (1.4) with $g_N(x_N)$ set to zero.

Now we invoke the dynamic programming algorithm in Theorem 1.2. We will first set $J_N(x_N) = g_N(x_N) = 0$, and then compute backwards in time

$$J_k(x_k) = \min_{u_k\in\mathbb{U}} \mathbb{E}_{w_k}\left\{g(x_k, u_k) + J_{k+1}(f(x_k, u_k, w_k))\right\}, \quad k = N-1, \dots, 0.$$

To make our presentation easier later, the above DP iterations are equivalent to

$$J_0(x_0) = 0 \tag{1.19}$$

$$J_{k+1}(x_{k+1}) = \min_{u_k\in\mathbb{U}} \mathbb{E}_{w_k}\left\{g(x_k, u_k) + J_k(f(x_k, u_k, w_k))\right\}, \quad k = 0, \dots, N, \tag{1.20}$$

where I have done nothing but reversed the time indexing.

Observe that when $N \to \infty$, (1.20) performs the recursion an infinite number of times.

We may want to conjecture three natural consequences of the infinite-horizon solution:

1. The optimal infinite-horizon cost is the limit of the corresponding N-stage optimal cost as $N \to \infty$, i.e.,

   $$J^\star(x) = \lim_{N\to\infty} J_N(x_N),$$

   where $J_N(x_N)$ is computed from DP (1.20).

2. Bacause $J^\star$ is the result of DP (1.20) when N tends to infinity, if the DP algorithm converges to $J^\star$, then $J^\star$ should satisfy

   $$J^\star(x) = \min_{u\in\mathbb{U}} \mathbb{E}_w\left\{g(x, u) + J^\star(f(x, u, w))\right\}, \quad \forall x \tag{1.21}$$

   Note that (1.21) is an *equation* that $J^\star(x)$ should satisfy for all x. In fact, this is called the *Bellman Optimality Equation*.

3. If $\mu(x)$ satisfies the Bellman equation (1.21), i.e., $u = \mu(x)$ minimizes the right-hand side of (1.21) for any x, then the policy $\pi = \{\mu, \mu, ...\}$ should be optimal. This is saying, the optimal policy is time-invariant.

In fact, all of our conjectures above are true, for most infinite-horizon problems. For example, in Chapter 2.2, we will investigate the Markov Decision Process (MDP) formulation, under which the above conjectures all hold. However, one should know that there also exist many infinite-horizon problems where our conjectures will fail, and there are many mathematical subtleties in rigorously proving the conjectures.

The reader should see why it can be more convenient to study the infinite-horizon formulation: (i) the optimal cost-to-go is only a function of the state x, but not a function of timestep k; (ii) the optimal policy is time-invariant and easier to implement.

Value Iteration. The Bellman optimality equation (1.21) also suggests a natural algorithm for computing $J^\star(x)$. We start with $J(x)$ being all zero, and then iteratively update $J(x)$ by performing the right-hand side of (1.21). This is the famous *value iteration* algorithm. We will study it in Chapter 2.2.

As practitioners, we may simply execute the dynamic programming (value iteration) algoithm without carefully checking if our problem satisfies the assumptions. If the algorithm converges, oftentimes the problem indeed satisfies the assumptions. Otherwise, the algorithm may fail to converge, as we will see in Example 2.3.

Chapter 2

Exact Dynamic Programming

In Chapter 1, we introduced the basic formulation of the finite-horizon and discrete-time optimal control problem, presented the Bellman principle of optimality, and derived the dynamic programming (DP) algorithm. We mentioned that, despite being a general-purpose algorithm, it can be difficult to implement DP exactly in practical applications.

In this Chapter, we will introduce two problem setups where DP can in fact be implemented exactly.

2.1 Linear Quadratic Regulator

Consider a linear discrete-time dynamical system

$$x_{k+1} = A_k x_k + B_k u_k + w_k, \quad k = 0, 1, \dots, N-1, \tag{2.1}$$

where $x_k \in \mathbb{R}^n$ the state, $u_k \in \mathbb{R}^m$ the control, $w_k \in \mathbb{R}^n$ the independent, zero-mean disturbance with given probability distribution that does not depend on x_k, u_k, and $A_k \in \mathbb{R}^{n \times n}, B_k \in \mathbb{R}^{n \times m}$ are known matrices determining the transition dynamics.

We want to solve the following optimal control problem

$$\min_{\mu_0, \dots, \mu_{N-1}} \mathbb{E}\left\{ x_N^T Q_N x_N + \sum_{k=0}^{N-1} \left(x_k^T Q_k x_k + u_k^T R_k u_k \right) \right\}, \tag{2.2}$$

where the expectation is taken over the randomness in $w_0, \dots, w_{N-1}$. In (2.2), $\{Q_k\}_{k=0}^N$ are positive semidefinite matrices, and $\{R_k\}_{k=0}^{N-1}$ are positive definite

matrices. The formulation (2.2) is typically known as the linear quadratic regulator (LQR) problem because the dynamics is linear, the cost is quadratic, and the formulation can be considered to "regulate" the system around the origin $x = 0$.

We will now show that the DP algorithm in Theorem 1.2 can be exactly implemented for LQR.

The DP algorithm computes the optimal cost-to-go backwards in time. The terminal cost is

$$J_N(x_N) = x_N^T Q_N x_N$$

by definition.

The optimal cost-to-go at time $N-1$ is equal to

$$\begin{aligned} J_{N-1}(x_{N-1}) = \min_{u_{N-1}} \mathbb{E}_{w_{N-1}} \{ & x_{N-1}^T Q_{N-1} x_{N-1} + u_{N-1}^T R_{N-1} u_{N-1} + \\ & \| \underbrace{A_{N-1} x_{N-1} + B_{N-1} u_{N-1} + w_{N-1}}_{x_N} \|_{Q_N}^2 \} \end{aligned} \tag{2.3}$$

where $\|v\|_Q^2 = v^T Q v$ for $Q \succeq 0$. Now observe that the objective in (2.3) is

$$\begin{aligned} x_{N-1}^T Q_{N-1} x_{N-1} + u_{N-1}^T R_{N-1} u_{N-1} + \|A_{N-1} x_{N-1} + B_{N-1} u_{N-1}\|_{Q_N}^2 + \\ \mathbb{E}_{w_{N-1}} \left[2(A_{N-1} x_{N-1} + B_{N-1} u_{N-1})^T Q_{N-1} w_{N-1} \right] + \\ \mathbb{E}_{w_{N-1}} \left[w_{N-1}^T Q_N w_{N-1} \right] \end{aligned} \tag{2.4}$$

where the second line is zero due to $\mathbb{E}(w_{N-1}) = 0$ and the third line is a constant with respect to u_{N-1}. Consequently, the optimal control $u_{N-1}^\star$ can be computed by setting the derivative of the objective with respect to u_{N-1} equal to zero

$$u_{N-1}^\star = -\left[\left(R_{N-1} + B_{N-1}^T Q_N B_{N-1} \right)^{-1} B_{N-1}^T Q_N A_{N-1} \right] x_{N-1}. \tag{2.5}$$

Plugging the optimal controller $u_{N-1}^\star$ back to the objective of (2.3) leads to

$$J_{N-1}(x_{N-1}) = x_{N-1}^T S_{N-1} x_{N-1} + \mathbb{E}\left[w_{N-1}^T Q_N w_{N-1} \right], \tag{2.6}$$

with

$$S_{N-1} = Q_{N-1} + A_{N-1}^T \left[Q_N - Q_N B_{N-1} \left(R_{N-1} + B_{N-1}^T Q_N B_{N-1} \right)^{-1} B_{N-1}^T Q_N \right] A_{N-1}.$$

We note that S_{N-1} is positive semidefinite (this is an exercise for you to convince yourself).

Now we realize that something surprising and nice has happened.

1. The optimal controller $u_{N-1}^\star$ in (2.5) is a linear feedback policy of the state x_{N-1}, and

2. The optimal cost-to-go $J_{N-1}(x_{N-1})$ in (2.6) is quadratic in x_{N-1}, just the same as $J_N(x_N)$.

This implies that, if we continue to compute the optimal cost-to-go at time $N-2$, we will again compute a linear optimal controller and a quadratic optimal cost-to-go. This is the rare nice property for the LQR problem, that is,

> The (representation) complexity of the optimal controller and cost-to-go does not grow as we run the DP recursion backwards in time.

We summarize the solution for the LQR problem (2.2) as follows.

Proposition 2.1 (Solution of Discrete-Time Finite-Horizon LQR)**.** *The optimal controller for the LQR problem* (2.2) *is a linear state-feedback policy*

$$\mu_k^\star(x_k) = -K_k x_k, \quad k = 0, \dots, N-1. \tag{2.7}$$

The gain matrix K_k can be computed as

$$K_k = \left(R_k + B_k^T S_{k+1} B_k\right)^{-1} B_k^T S_{k+1} A_k,$$

where the matrix S_k satisfies the following backwards recursion

$$\begin{aligned} S_N &= Q_N \\ S_k &= Q_k + A_k^T \left[S_{k+1} - S_{k+1} B_k \left(R_k + B_k^T S_{k+1} B_k\right)^{-1} B_k^T S_{k+1}\right] A_k, k = N-1, \dots, 0. \end{aligned} \tag{2.8}$$

The optimal cost-to-go is given by

$$J_0(x_0) = x_0^T S_0 x_0 + \sum_{k=0}^{N-1} \mathbb{E}\left[w_k^T S_{k+1} w_k\right].$$

The recursion (2.8) *is called the* discrete-time Riccati equation.

Proposition 2.1 states that, to evaluate the optimal policy (2.7), one can first run the backwards Riccati equation (2.8) to compute all the positive definite matrices S_k, and then compute the gain matrices K_k. For systems of reasonable dimensions, evalutating the matrix inversion in (2.8) should be fairly efficient.

2.1.1 Infinite-Horizon LQR

In many robotics applications, it is often more useful to study the infinite-horizon LQR problem

$$\min_{u_k} \quad \sum_{k=0}^{\infty} \left(x_k^T Q x_k + u_k^T R u_k\right) \tag{2.9}$$

$$\text{subject to} \quad x_{k+1} = A x_k + B u_k, \quad k = 0, \dots, \infty, \tag{2.10}$$

where $Q \succeq 0$, $R \succ 0$, and A, B are constant matrices. The reason for studying the formulation (2.9) is twofold. First, for nonlinear systems, we often linearize the nonlinear dynamics around an (equilibrium) point we care about, leading to constant A and B matrices. Second, we care more about the *asymptotic* effect of our controller than its behavior in a fixed number of steps. We will soon see an example of this formulation for balancing a simple pendulum.

The infinite-horizon formulation is essentially the finite-horizon formulation (2.2) with $N \to \infty$. Based on our intuition in deriving the finite-horizon LQR solution, we may want to hypothesize that the optimal cost-to-go is a quadratic function

$$J_k(x_k) = x_k^T S x_k, k = 0, \dots, \infty \tag{2.11}$$

for some positive definite matrix S, and proceed to invoke the DP algorithm. Notice that we hypothesize the matrix S is in fact *stationary*, i.e., it does not change with respect to time. This hypothesis makes sense because the A, B, Q, R matrices are stationary in the formulation (2.9). Invoking the DP algorithm we have

$$x_k^T S x_k = J_k(x_k) = \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \| \underbrace{A x_k + B u_k}_{x_{k+1}} \|_S^2 \right\}. \tag{2.12}$$

The minimization over u_k in (2.12) can again be solved in closed-form by setting the gradient of the objective with respect to u_k to be zero

$$u_k^\star = - \underbrace{\left[\left(R + B^T S B \right)^{-1} B^T S A \right]}_{K} x_k. \tag{2.13}$$

Plugging the optimal $u_k^\star$ back into (2.12), we see that the matrix S has to satisfy the following equation

$$S = Q + A^T \left[S - S B \left(R + B^T S B \right)^{-1} B^T S \right] A. \tag{2.14}$$

Equation (2.14) is the famous *algebraic Riccati equation*.

Let's zoom out to see what we have done. We started with a hypothetical optimal cost-to-go (2.11) that is stationary, and invoked the DP algorithm in (2.12), which led us to the algebraic Riccati equation (2.14). Therefore, if there actually exists a solution to the algebraic Riccati equation (2.14), then the linear controller (2.13) is indeed optimal (by the optimality of DP)!

So the question boils down to if the algebraic Riccati equation has a solution S that is positive definite? The following proposition gives an answer.

Proposition 2.2 (Solution of Discrete-Time Infinite-Horizon LQR)**.** *Consider a linear system*

$$x_{k+1} = A x_k + B u_k,$$

with (A, B) controllable (see Appendix C.2). Let $Q \succeq 0$ in (2.9) be such that Q can be written as $Q = C^T C$ with (A, C) observable.

Then the optimal controller for the infinite-horizon LQR problem (2.9) is a stationary linear policy

$$\mu^\star(x) = -Kx,$$

with

$$K = \left(R + B^T S B\right)^{-1} B^T S A.$$

The matrix S is the unique positive definite matrix that satisfies the algebraic Riccati equation

$$S = Q + A^T \left[S - SB\left(R + B^T S B\right)^{-1} B^T S\right] A.$$

Moreover, the closed-loop system

$$x_{k+1} = Ax_k + B(-Kx_k) = (A - BK)x_k$$

is stable, i.e., the eigenvalues of the matrix $A - BK$ are strictly within the unit circle (see Appendix C.1.2).

A rigorous proof of Proposition 2.2 is available in Proposition 3.1.1 of (Bertsekas, 2012). The proof basically studies the limit of the discrete-time Riccati equation (2.8) when $N \to \infty$. Indeed, the algebraic Riccati equation (2.14) is the limit of the discrete-time Riccati equation (2.8) when $N \to \infty$. The assumptions of (A, B) being controllable and (A, C) being observable can be relaxted to (A, B) being stabilizable and (A, C) being detectable (for definitions of stabilizability and detectability, see Appendix C).

We have not discussed how to solve the algebraic Riccati equation (2.8). It is clear that (2.8) is not a linear system of equations in S. In fact, the numerical algorithms for solving the algebraic Riccati equation can be highly nontrivial, for example see (Arnold and Laub, 1984). Fortunately, such algorithms are often readily available, and as practitioners we do not need to worry about solving the algebraic Riccati equation by ourselves. For example, the Matlab `dlqr` function computes the K and S matrices from A, B, Q, R.

Let us now apply the infinite-horizon LQR solution to stabilizing a simple pendulum.

Example 2.1 (Pendulum Stabilization by LQR)**.** Consider the simple pendulum in Fig. 2.1 with dynamics

$$x = \begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix}, \quad \dot{x} = f(x, u) = \begin{bmatrix}\dot{\theta} \\ -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta) + \frac{1}{ml^2}u\end{bmatrix} \tag{2.15}$$

where m is the mass of the pendulum, l is the length of the pole, g is the gravitational constant, b is the damping ratio, and u is the torque applied to the pendulum.

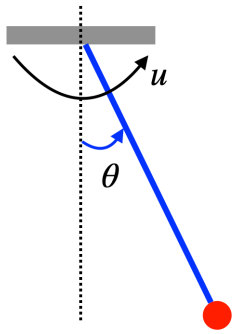

Figure 2.1: A Simple Pendulum.

We are interested in applying the LQR controller to balance the pendulum in the upright position $x_d = [\pi, 0]^T$ with a zero velocity.

Let us first shift the dynamics so that "0" is the upright position. This can be done by defining a new variable $z = x - x_d = [\theta - \pi, \dot{\theta}]^T$, which leads to

$$\dot{z} = \dot{x} = f(x,u) = f(z + x_d, u) = \begin{bmatrix} z_2 \\ \frac{1}{ml^2}\left(u - bz_2 + mgl\sin z_1\right) \end{bmatrix} = f'(z,u). \tag{2.16}$$

We then linearize the nonlinear dynamics $\dot{z} = f'(z,u)$ at the point $z^\star = 0, u^\star = 0$:

$$\dot{z} \approx f'(z^\star, u^\star) + \left(\frac{\partial f'}{\partial z}\right)_{z^\star,u^\star}(z - z^\star) + \left(\frac{\partial f'}{\partial u}\right)_{z^\star,u^\star}(u - u^\star) \tag{2.17}$$

$$= \begin{bmatrix} 0 & 1 \\ \frac{g}{l}\cos z_1 & -\frac{b}{ml^2} \end{bmatrix}_{z^\star,u^\star} z + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u \tag{2.18}$$

$$= \underbrace{\begin{bmatrix} 0 & 1 \\ \frac{g}{l} & -\frac{b}{ml^2} \end{bmatrix}}_{A_c} z + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix}}_{B_c} u. \tag{2.19}$$

Finally, we convert the continuous-time dynamics to discrete time with a fixed discretization h

$$z_{k+1} = \dot{z}_k \cdot h + z_k = \underbrace{(h \cdot A_c + I)}_{A} z_k + \underbrace{(h \cdot B_c)}_{B} u_k.$$

We are now ready to implement the LQR controller. In the formulation (2.9), we choose $Q = I$, $R = I$, and solve the gain matrix K using the Matlab `dlqr` function.

Fig. 2.2 shows the simulation result for $m = 1, l = 1, b = 0.1$, $g = 9.8$, and $h = 0.01$, with an initial condition $z^0 = [0.1, 0.1]^T$. We can see that the LQR controller successfully stabilizes the pendulum at $z^\star$, the upright position.

You can play with the Matlab code here.

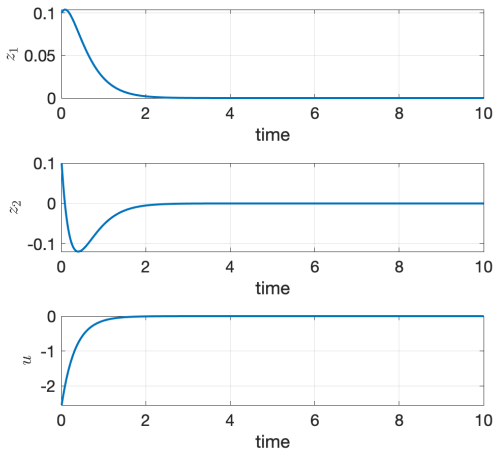

Figure 2.2: LQR stabilization of a simple pendulum.

2.1.2 LQR with Constraints

Let's explore LQR with constraints in Exercise 9.2

2.2 Markov Decision Process

In Section 2.1, we see that linear dynamics and quadratic costs leads to exact dynamic programming. We now introduce another setup where the number of states and controls is finite (as opposed to the LQR case where x_k and u_k live in continuous spaces). We will see that we can execute DP exactly in this setup as well.

Optimal control in the case of finite states and controls is typically introduced in the framework of a *Markov Decision Process* (MDP, which is common in Reinforcement Learning). There are many variations of a MDP, and here we only focus on the discounted infinite-horizon MDP. For a more complete treatment of MDPs, I suggest checking out this course at Harvard.

Formally, a discounted infinite-horizon MDP $\mathcal{M} = (\mathbb{X}, \mathbb{U}, P, g, \gamma, \sigma)$ is specified by

- a state space $\mathbb{X}$ that is finite with size $|\mathbb{X}|$
- a control space $\mathbb{U}$ that is finite with size $|\mathbb{U}|$

- a transition function $P : \mathbb{X} \times \mathbb{U} \to \Delta(\mathbb{X})$, where $\Delta(\mathbb{X})$ is the space of probability distributions over $\mathbb{X}$; specifically, $P(x' \mid x, u)$ is the probability of transitioning into state x' from state x using control u. If the system is deterministic, then $P(x' \mid x, u)$ is nonzero only for a single next state x'
- a cost function $g : \mathbb{X} \times \mathbb{U} \to [0, 1]$; $g(x, u)$ is the cost of taking the control u at state x
- a discount factor $\gamma \in [0, 1)$
- an initial state distribution $\sigma \in \Delta(\mathbb{X})$ that specifies how the initial state x_0 is generated; in many cases we will assume x_0 is fixed and σ is a distribution supported only on x_0.

In an MDP, the system starts at some state $x_0 \sim \sigma$. At each step $k = 0, 1, 2, ...$, the system decides a control $u_k \in \mathbb{U}$ and incurs a cost $g(s_k, u_k)$. The control u_k brings the system into a new state $x_{k+1} \sim P(\cdot \mid x_k, u_k)$, at which the controller decides a new control u_{k+1}. This process continues forever.

Controller (policy). In general, a time-varying controller $\pi = (\pi_0, \dots, \pi_k, \dots)$ is a mapping from all previous states and controls to a distribution over current controls. The mapping π_k at timestep k is

$$\pi_k : (x_0, u_0, x_1, u_1, \dots, x_k) \mapsto u_k \sim q_k \in \Delta(\mathbb{U}).$$

Note that u_k can be randomized and it is drawn from a distribution q_k supported on the set of controls $\mathbb{U}$. A *stationary* controller (policy) $\pi : \mathbb{X} \to \Delta(\mathbb{U})$ specifies a decison-making strategy that is purely based on the current state x_k. A *deterministic* and stationary controller $\pi : \mathbb{X} \to \mathbb{U}$ excutes a deterministic control u_k at each step.

Cost-to-go and Q-value. Given a controller π and an initial state x_0, we associate with it the following discounted infinite-horizon cost

$$J_\pi(x_0) = \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k g(x_k, u_k^\pi)\right\}, \tag{2.20}$$

where the expectation is taken over the randomness of the transition P and the controller π. Note that we have used u_k^π to denote the control at step k by following the controller π. Similarly, we define the Q-value function as

$$Q_\pi(x_0, u_0) = \mathbb{E}\left\{g(x_0, u_0) + \sum_{k=1}^{\infty} \gamma^k g(x_k, u_k^\pi)\right\}. \tag{2.21}$$

The difference between $Q_\pi(x_0, u_0)$ and $J_\pi(x_0)$ is that at step zero, $J_\pi(x_0)$ follows the controller π while $Q_\pi(x_0, u_0)$ assumes the control u_0 is given. By the assumption that $g(x_k, u_k) \in [0, 1]$, we have

$$0 \le J_\pi(x_0), Q_\pi(x_0, u_0) \le \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}, \quad \forall \pi.$$

Our goal is to find the best controller that minimizes the cost function

$$\pi^\star \in \arg\min_{\pi\in\Pi} J_\pi(x_0) \tag{2.22}$$

for a given initial state x_0, where Π is the space of all non-stationary and randomized controllers.

A remarkable property of MDPs is that there exists an optimal controller that is stationary and deterministic.

Theorem 2.1 (Deterministic and Stationary Optimal Policy)**.** *Let Π be the space of all non-stationary and randomized policies. Define*

$$J_\pi^\star(x) = \min_{\pi\in\Pi} J_\pi(x), \quad Q_\pi^\star(x,u) = \min_{\pi\in\Pi} Q_\pi(x,u).$$

There exists a deterministic and stationary policy $\pi^\star$ such that for all $x \in \mathbb{X}$ and $u \in \mathbb{U}$,

$$J_{\pi^\star}(x) = J^\star(x), \quad Q_{\pi^\star}(x,u) = Q^\star(x,u).$$

We call such a policy π an optimal policy.

Proof. See Theorem 1.7 in (Agarwal et al., 2022). □

This Theorem shows that we can restrict ourselves to stationary and deterministic policies without losing performance.

In the next, we show how to characterize the optimal policy and value function.

2.2.1 Bellman Optimality Equations

We now restrict ourselves to stationary policies. We first introduce the Bellman Consistency Equations for stationary policies.

Lemma 2.1 (Bellman Consistency Equations)**.** *Let π be a stationary policy. Then J_π and Q_π satisfy the following Bellman consistency equations*

$$J_\pi(x) = \mathbb{E}_{u\sim\pi(\cdot|x)} Q_\pi(x,u), \tag{2.23}$$

$$Q_\pi(x,u) = g(x,u) + \gamma \mathbb{E}_{x'\sim P(\cdot|x,u)} J_\pi(x'). \tag{2.24}$$

Proof. By the definition of the cost-to-go function in (2.20), we have

$$J_\pi(x_0) = \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k g(x_k, \pi(x_k))\right\} = \mathbb{E}_{u_0\sim\pi(\cdot|x_0)} \underbrace{\mathbb{E}\left\{g(x_0,u_0) + \sum_{k=1}^{\infty} \gamma^k g(x_k,\pi(x_k))\right\}}_{Q_\pi(x_0,u_0)}.$$

The above equation holds for any x_0, proving (2.23).

To show (2.24), we recall the definition of the Q-value function (2.21)

$$Q_\pi(x_0, u_0) = \mathbb{E}\left\{ g(x_0, u_0) + \sum_{k=1}^{\infty} \gamma^k g(x_k, \pi(x_k)) \right\} \tag{2.25}$$

$$= g(x_0, u_0) + \gamma \mathbb{E}\left\{ \sum_{k=1}^{\infty} \gamma^{k-1} g(x_k, \pi(x_k)) \right\} \tag{2.26}$$

Now observe that the expectation of the second term in (2.26) is taken over both the randomness of x_1 and the randomness of the policy after x_1 is reached. Therefore,

$$\mathbb{E}\left\{ \sum_{k=1}^{\infty} \gamma^{k-1} g(x_k, \pi(x_k)) \right\} = \mathbb{E}_{x_1 \sim P(\cdot|x_0,u_0)} \underbrace{\left\{ \mathbb{E}\left\{ \sum_{k=1}^{\infty} \gamma^{k-1} g(x_1, \pi(x_1)) \right\} \right\}}_{J_\pi(x_1)}.$$

Plugging the above equation back to (2.26), we obtain the desired result in (2.24). □

Matrix Representation. It is useful to think of P, g, J_π, Q_π as matrices. In particular, the transition function P can be considered as a matrix of dimension $|\mathbb{X}||\mathbb{U}| \times \mathbb{X}$, where

$$P_{(x,u),x'} = P(x' \mid x, u)$$

is the entry of P at the row (x, u) (there are $|\mathbb{X}||\mathbb{U}|$ such rows) and column x' (there are $|\mathbb{X}|$ such columns). The running cost g is vector of $|\mathbb{X}||\mathbb{U}|$ entries. The cost-to-go $J_\pi(x)$ is a vector of $|\mathbb{X}|$ entries. The Q-value function $Q_\pi(x, u)$ is a vector of $|\mathbb{X}||\mathbb{U}|$ entries. We also introduce P^π with dimension $|\mathbb{X}||\mathbb{U}| \times |\mathbb{X}||\mathbb{U}|$ as the transition matrix induced by a stationary policy π. In particular,

$$P^\pi_{(x,u),(x',u')} = P(x' \mid x, u)\pi(u' \mid x').$$

In words, $P^\pi_{(x,u),(x',u')}$ is the probability that (x', u') follows (x, u).

With the matrix representation, we can compactly write the Bellman consistency equation (2.24) as

$$Q_\pi = g + \gamma P J_\pi. \tag{2.27}$$

We can also combine (2.23) and (2.24) together and write

$$Q_\pi(x, u) = g(x, u) + \gamma \mathbb{E}_{x' \sim P(\cdot|x,u)} \left\{ \mathbb{E}_{u' \sim \pi(\cdot|x')} Q_\pi(x', u') \right\} = g(x, u) + \gamma \mathbb{E}_{(x',u') \sim P^\pi(\cdot|(x,u))} Q_\pi(x', u'),$$

which, using matrix representation, becomes

$$Q_\pi = g + \gamma P^\pi Q_\pi. \tag{2.28}$$

Equation (2.28) immediately yields

$$Q_\pi = (I - \gamma P^\pi)^{-1} g, \tag{2.29}$$

that is, the Q-value function associated with a stationary policy π can be simply computed from solving a linear system as in (2.29).[1]

Lemma 2.1, together with the equivalent matrix equations (2.27) and (2.28), provide the conditions that J_π and Q_π, induced by any stationary policy π, need to satisfy. In the next, we describe the conditions that characterize the optimal policy.

Theorem 2.2 (Bellman Optimality Equations)**.** *A vector $Q \in \mathbb{R}^{|\mathbb{X}||\mathbb{U}|}$ is said to satisfy the Bellman optimality equation if*

$$Q(x,u) = g(x,u) + \gamma \mathbb{E}_{x' \sim P(\cdot|x,u)} \left\{ \min_{u' \in \mathbb{U}} Q(x',u') \right\}, \quad \forall (x,u) \in \mathbb{X} \times \mathbb{U}. \tag{2.30}$$

A vector $Q^\star$ is the optimal Q-value function if and only if it satisfies (2.30)*. Moreover, the deterministic policy defined by*

$$\pi^\star(x) \in \arg\min_{u \in \mathbb{U}} Q^\star(x,u)$$

with ties broken arbitrarily is an optimal policy.

Proof. See Theorem 1.8 in (Agarwal et al., 2022). □

We now make a few definitions to interpret Theorem 2.2. For any vector $Q \in \mathbb{R}^{|\mathbb{X}||\mathbb{U}|}$, define the greedy policy as

$$\pi_Q(x) \in \arg\min_{u \in \mathbb{U}} Q(x,u) \tag{2.31}$$

with ties broken arbitrarily. With this notation, by Theorem 2.2, the optimal policy is

$$\pi^\star = \pi_{Q^\star},$$

where $Q^\star$ is the optimal Q-value function. Similarly, let us define

$$J_Q(x) = \min_{u \in \mathbb{U}} Q(x,u).$$

Note that J_Q has dimension $|\mathbb{X}|$. With these notations, the *Bellman optimality operator* is defined as

$$\mathcal{T}Q = g + \gamma P J_Q, \tag{2.32}$$

which is nothing but a matrix representation of the right-hand side of (2.30). This allows us to concisely write the Bellman optimality equation (2.30) as

$$Q = \mathcal{T}Q. \tag{2.33}$$

Therefore, an equivalent way to interpret Theorem 2.2 is that $Q = Q^\star$ if and only if Q is a fixed point to the Bellman optimality operator $\mathcal{T}$.

[1] One can show that the matrix $I - \gamma P^\pi$ is indeed invertible, see Corollary 1.5 in (Agarwal et al., 2022).

2.2.2 Value Iteration

Intepreting the optimal Q-value function as the fixed point to the Bellman optimality operator (2.33) leads us to a natural algorithm for solving the optimal control problem.

We start with $Q^{(0)}$ being an all-zero vector and then at iteration t, we perform

$$Q^{(t+1)} \leftarrow \mathcal{T} Q^{(t)},$$

with $\mathcal{T}$ defined in (2.32). Let us observe the simplicity of this algorithm: at each iteration, one only needs to perform $\min_{u \in \mathbb{U}} Q^{(t)}(x, u)$, which is very efficient when $|\mathbb{U}|$ is not too large.

The next theorem states this simple algorithm converges to the optimal value function.

Theorem 2.3 (Value Iteration)**.** *Set $Q^{(0)} = 0$. For $t = 0, \ldots,$ perform*

$$Q^{(t+1)} \leftarrow \mathcal{T} Q^{(t)}.$$

Let $\pi^{(k)} = \pi_{Q^{(k)}}$ (see the definition in (2.31)). For $t \geq \frac{\log \frac{2}{(1-\gamma)^2 \epsilon}}{1-\gamma}$, we have

$$J^{\pi^{(t)}} \leq J^\star + \epsilon \mathbb{1},$$

where $\mathbb{1}$ is the all-ones vector.

Essentially, the value function obtained from value iteration converges to the optimal cost-to-go.

Let us use an example to appreciate this algorithm.

Example 2.2 (Shortest Path in Grid World)**.** Consider the following 10×10 grid world, where the top-right cell is the goal location, and the dark blue colored cells are obstacles.

We want to find the shortest path from a given cell to the target cell, while not hitting obstacles.

To do so, we set the state space of the system as

$$\mathbb{X} = \left\{ \begin{bmatrix} r \\ c \end{bmatrix} \middle| r, c \in \{1, \ldots, 10\} \right\}$$

where r is the row index (from top to bottom) and c is the column index (from left to right). The control space is moving left, right, up, down, or do nothing:

$$\mathbb{U} = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\}.$$

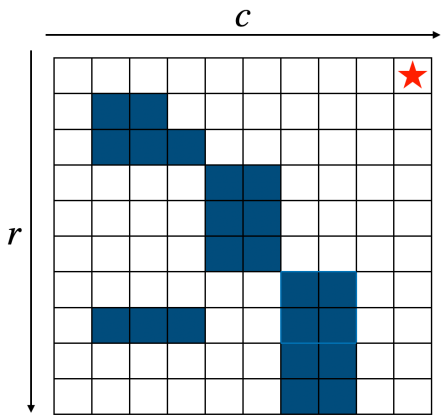

Figure 2.3: Grid World with Obstacles.

The system dynamics is deterministic

$$x' = \begin{cases} x+u & \text{if } x+u \text{ is inside the grid} \\ x & \text{otherwise} \end{cases}.$$

We then design the following running cost function g

$$g(x,u) = \begin{cases} 0 & \text{if } x = [1,10]^T \text{ is the target} \\ 20 & \text{if } x \text{ is an obstacle} \\ 1 & \text{otherwise} \end{cases}.$$

Note that $g(x,u)$ defined above does not even satisfy $g \in [0,1]$. We then use value iteration to solve the optimal control problem with $\gamma = 1$

$$J(x_0) = \min_{\pi} \sum_{k=0}^{\infty} g(x_k, \pi(x_k)).$$

The Matlab script of value iteration converges in 27 iterations, and we obtain the optimal cost-to-go in Fig. 2.4.

Starting from the cell $[8,5]^T$, the red line in Fig. 2.4 plots the optimal trajectory that clearly avoids the obstacles.

Feel free to play with the size of the grid and the number of obstacles.

Example 2.2 shows the simplicity and power of value iteration. However, the states and controls in the grid world are naturally discrete and finite. Is it possible to apply value iteration to optimal control problems where the states and controls live in continuous spaces?

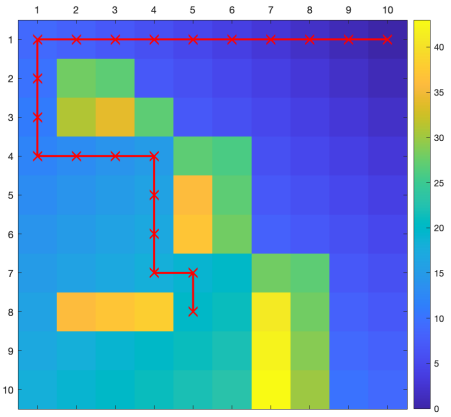

Figure 2.4: Optimal cost-to-go and an optimal trajectory.

2.2.3 Value Iteration with Barycentric Interpolation

Let us consider the discrete-time dynamics

$$x_{k+1} = f(x_k, u_k)$$

where both x_k and u_k live in a continuous space, say $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively.

A natural idea to apply value iteration is to discretize the state space and control space. For example, suppose $x \in \mathbb{R}^2$ and we have discretized $\mathbb{R}^2$ using N points

$$\mathcal{S} = \{s_1, \dots, s_N\}$$

that lie on a 2D grid, as shown in Fig. 2.5. Assume $x_k \in \mathcal{S}$ lies on the mesh grid, the next state $x_{k+1} = f(x_k, u_k)$ will, however, most likely not lie exactly on one of the grid points.

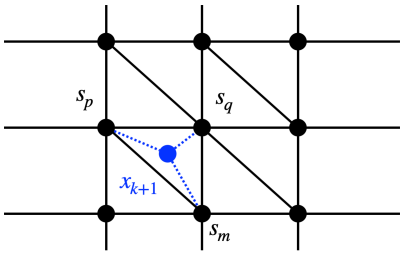

Figure 2.5: Barycentric Interpolation.

Nevertheless, x_{k+1} will lie inside a triangle with vertices s_p, s_q, s_m. We will now try to write x_{k+1} using the vertices, that is, to find three numbers $\lambda_p, \lambda_q, \lambda_m$

such that

$$\lambda_p, \lambda_q, \lambda_m \geq 0, \quad \lambda_p + \lambda_q + \lambda_m = 1, \quad x_{k+1} = \lambda_p s_p + \lambda_q s_q + \lambda_m s_m.$$

$\lambda_p, \lambda_q, \lambda_m$ are called the barycentric coordinates of x_{k+1} in the triangle formed by s_p, s_q, s_m. With the barycentric coordinates, we will assign the transition matrix

$$P(x_{k+1} = s_p \mid x_k, u_k) = \lambda_p, \quad P(x_{k+1} = s_q \mid x_k, u_k) = \lambda_q, \quad P(x_{k+1} = s_m \mid x_k, u_k) = \lambda_m.$$

Let us apply value iteration with barycentric interpolation to the simple pendulum.

Example 2.3 (Value Iteration with Barycentric Interpolation on A Simple Pendulum)**.** Consider the continuous-time pendulum dynamics in (2.16) that is already shifted such that $z = 0$ corresponds to the upright position. With time discretization h, we can write the discrete-time dynamics as

$$z_{k+1} = \dot{z}_k \cdot h + z_k = f'(z_k, u_k) \cdot h + z_k.$$

We are interested in solving the optimal control problem

$$J(z_0) = \min_{u_k} \left\{ \sum_{k=0}^{\infty} \gamma^k g(x_k, u_k) \right\},$$

where the running cost is simply

$$g(x_k, u_k) = x_k^T x_k + u_k^2.$$

We will use the parameters $m = 1, g = 9.8, l = 1, b = 0.1$, and assume the control is bounded in $[-4.9, 4.9]$.

We want to compute the optimal cost-to-go in the range $z_1 \in [-\pi, \pi]$ and $z_2 \in [-\pi, \pi]$. We discretize both z_1 and z_2 using N points, leading to N^2 points in the state space. We also discretize u using N points.

Applying value iteration with $\gamma = 0.9$ and $N = 50$, we obtain the optimal cost-to-go in Fig. 2.6. The value iteration converges in 277 iterations.

Applying value iteration with $\gamma = 0.99$ and $N = 50$, we obtain the optimal cost-to-go in Fig. 2.7. The value iteration converges in 2910 iterations.

Applying value iteration with $\gamma = 0.999$ and $N = 50$, we obtain the optimal cost-to-go in Fig. 2.8. The value iteration converges in 28850 iterations.

You can find the Matlab code here.

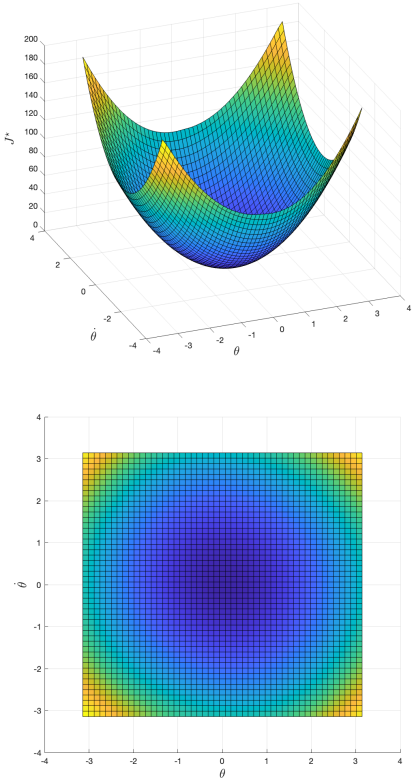

Figure 2.6: Optimal cost-to-go with discount factor 0.9.

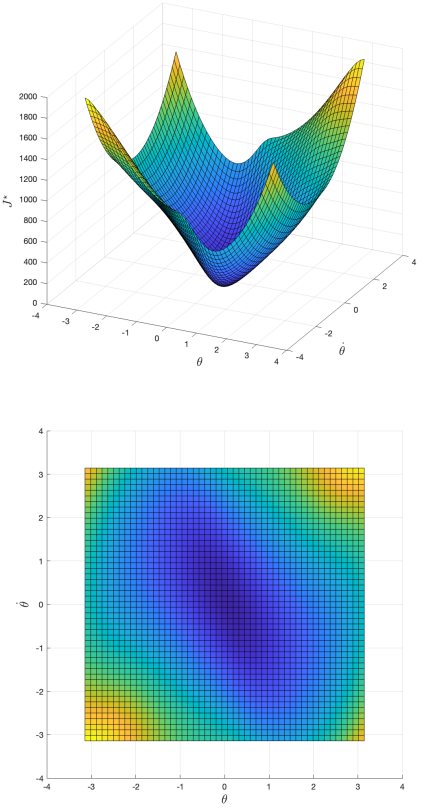

Figure 2.7: Optimal cost-to-go with discount factor 0.99.

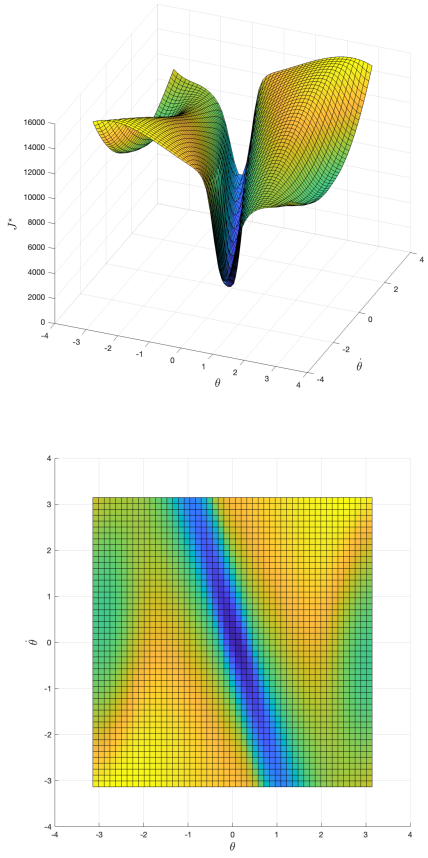

Figure 2.8: Optimal cost-to-go with discount factor 0.999.

Chapter 3

Approximate Dynamic Programming

Thanks to Jiarui Li for writing this Chapter.

3.1 Introduction

The limitations of classical deterministic dynamic programming (DP) were mentioned in Chapter 1, particularly its inefficiency in fields such as robotics where both the state and control spaces are typically large and continuous. The process of discretization in such contexts is not only challenging but also costly. Even when discretization is achievable, the resultant state and control spaces tend to be extraordinarily vast and often high-dimensional, leading to prohibitive computational demands. This issue, commonly called the *curse of dimensionality*, renders the use of classical DP unfeasible. Add time complexity analysis here. To circumvent the constraints of traditional DP algorithms in such contexts, a pragmatic approach involves the adoption of a suboptimal control scheme. This compromises between the ease of implementation and adequate performance. The principal objective of this chapter is to find such suboptimal control. In this chapter, we will spend most of the time discussing finite horizon problems with discrete state and control space, which is the classical scenario. We will also mention the infinite horizon problem and continuous state and control spaces scenario later.

Broadly, two categories of approximation are used in the context of DP-based suboptimal control. The first is *approximation in value space*, where we aim to approximate the optimal cost function or the cost function of a given policy. The second is *approximation in policy space*, where we select the policy by using optimization over a suitable class of policies.

3.2 Approximation in value space

Let us first recap the iteration process of the generic form of DP as mentioned in theorem 1.2. We can obtain the cost-to-go function J_k, which means the cost-to-go value at time k, thereby defining corresponding control u_k or policy μ_k.

$$J_k(x_k) = \min_{u_k \in \mathbb{U}} \mathbb{E}\left\{g_k(x_k, u_k) + J_{k+1}(f_k(x_k, u_k, w_k))\right\}, \ k = N-1, \dots, 1, 0. \quad (3.1)$$

By using the *approximation in value space* methods, we could replace the optimal cost-to-go function J_k with some other functions $\tilde{J}_k$. In other words, the suboptimal policy $\tilde{\mu}_k(x_k)$ (and the corresponding control) is obtained from the one-step lookahead minimization

$$\tilde{J}_k(x_k) = \min_{u_k \in \mathbb{U}_k(x_k)} \mathbb{E}\left\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\right\} \quad (3.2)$$

$$\tilde{\mu}_k(x_k) = \arg\min_{u_k \in \mathbb{U}_k(x_k)} \mathbb{E}\left\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\right\} \quad (3.3)$$

The major issue in value space approximation is how to compute the approximate cost-to-go functions $\tilde{J}_{k+1}$ in (3.3). We will consider three types of methods:

1. *Problem approximation*
2. *Parametric cost approximation*
3. *Online approximate optimization*

In approximation in value space, we may also distinguish between *online* and *offline* methods.

1. *Offline* methods, where the entire function $\tilde{J}_{k+1}$ in (3.3) is computed for every k before the control process begins. The advantage of this is that most of the computation is done offline. Once the control process starts, the only thing we have to do is one-step lookahead minimization. These methods are well-suited for settings where there are strict time constraints for the online computation of the control, and where there is no need for online replanning.
2. *Online* methods, where most of the computation is performed just after the current state x_k becomes known, the values $\tilde{J}_{k+1}(x_{k+1})$ are computed only at the relevant next states x_{k+1} and are used to compute u_k via (3.3). These methods require the computation of control only for the N states actually encountered in the control process. These methods are well-suited for online replanning.

3.2.1 Problem Approximation

The functions $\tilde{J}_{k+1}$ are obtained (by exact DP, or other methods) as the optimal or nearly optimal cost functions of a simplified version of the original problem. The problem is how to simplify the problem, which is more convenient for computation. There are three widely-used approaches to simplify the initial problem:

1. *Simplifying the structure of the problem through enforced decomposition.*
2. *Simplifying the probabilistic structure of the problem*, such as replacing the stochastic problem with a deterministic one by *certainty equivalence*. To be more specific, the original stochastic system contains the disturbance term $w_k(x_k, u_k)$. To simplify the probabilistic structure of the problem, we could fix the disturbances at some "typical" values and transform the stochastic problem into a deterministic one.
3. *Aggregation*, where the original problem is approximated with a new problem with fewer states, makes it easier to obtain the cost-to-go function. The state in this new problem is the "combination" of the states in the initial problem. It is worth noting that the discretization of continuous state space and action space could be viewed as a kind of aggregation.

3.2.2 Parametric cost approximation

For discrete problems, it is natural to consider using the tabular method to represent the $\tilde{J}_k$ functions. However, if the number of the state space is large this method's memory cost will be overwhelming. On the other hand, for tabular representation, it is not convenient to optimize the function, while we can only update the value of *one* state at a time, but in many circumstances, there is a cluster of states that have similar attributes, which means their corresponding $\tilde{J}_k$ are also similar. It is inconvenient to update them one by one. In this part, we will discuss an alternative approach to represent $\tilde{J}_k$ function, whereby $\tilde{J}_k$ are chosen to be members of a parametric class of functions, with the parameters "optimized" or "trained" by using some algorithms.

To be more specific, the $\tilde{J}_k$ functions could be described as $\tilde{J}_k(x_k, r_k)$ that for each k, depend on the current state x_k and a vector $r_k = (r_{1,k}, ..., r_{m,k})$ of m "tunable" scalar parameters, also called *weights*. By adjusting the weights, one can change the "shape" of $\tilde{J}_k$ so that it is a reasonably close approximation to the true cost-to-go function J_k. In order to train those weights, we can use some cost functions to measure the accuracy of the approximation. The most common cost function is *least squares*. Training the parameters r_k using least squares as the cost function is sometimes referred to as *fitted value iteration*. Value iteration could be viewed as a special form of dynamic programming, where the parameter vectors r_k are determined sequentially, starting from the

end of the horizon and proceeding backward. The algorithm samples the state space for each stage k and generates a large number of states x_k^s, $s = 1, ..., q$. It then determines sequentially the parameter vectors r_k to obtain a good "least square fit" to the DP algorithm.

$$\beta_k^s = \min_{u \in \mathbb{U}_k(x_k^s)} E\left\{ g(x_k^s, u, w_k) + \tilde{J}_{k+1}(f_k(x_k^s, u, w_k), r_{k+1}) \right\} \tag{3.4}$$

$$r_k = \arg\min_r \sum_{s=1}^{q} (\tilde{J}_k(x_k^s, r) - \beta_k^s)^2 \tag{3.5}$$

The next question is how to choose the most suitable class of functions, which is called *approximation architecture*. It is obvious that approximation architecture can greatly affect the performance of the approximation and the difficulty of training. The most popular architecture is *neural networks*, which are widely used in *reinforcement learning*, but the optimization process is difficult and the optimality is not guaranteed. We will start with a simpler linear feature-based approximation architecture.

3.2.2.1 Linear feature-based architecture

In this architecture, the $\tilde{J}_k$ function could be parameterized as follows:

$$\tilde{J}_k(x_k, r_k) = r_k^T \phi_k(x_k) \tag{3.6}$$

Here T means the transpose of the matrix, $\phi_k(x_k)$ is pre-selected and called the (non-linear) feature vector associated with x_k at time k. The scalar components of the feature vector are called *features*. Common examples of features include polynomials and radial basis functions. The notion of feature is commonly used in the theory of computer vision, where x_k could be interpreted as an image, and the ϕ_k function extracts the critical features such as angles and points that could be used for object recognition or image alignment. By using this architecture, the fitted value iteration (3.4), (3.5) greatly simplifies and admits a closed-form solution.

Example 3.1 (Optimal Control for Linear System Using Feature-based Method)**.** Consider a linear system whose dynamics is:

$$x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, \quad \dot{x} = f(x, u) = \begin{bmatrix} x_1 \\ u \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{3.7}$$

The goal is to regulate the system at $(0, 0)$. This system is a typical linear system, named as **double integrator** on many occasions. In Chapter 2, we learned how to use LQR to precisely calculate the cost-to-go function of these

systems using *Recatti Equation*. Now we can show that using linear feature-based fitted value iteration, we can also get the cost-to-go function of double integrator and the result is the same as the ground truth.

In this example, we use second-order monomials as the features and use the linear combination of them to model the cost-to-go function of the double integrator, while the combination parameter w is tuned using fitted value iteration.

$$\text{features vector: } F = \begin{bmatrix} x_0^2 \\ x_0 x_1 \\ x_1^2 \\ x_0 \\ x_1 \\ 1 \end{bmatrix}, \quad J(x) = w^T F \tag{3.8}$$

Applying fitted value iteration on randomly sampled points mentioned above, we obtain the cost-to-go in Fig. 3.1. The value iteration converges in 3000 iterations.

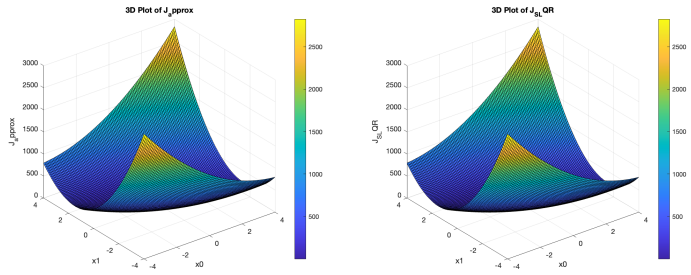

Figure 3.1: Comparison between the result of feature-based FVI and ground truth

The S matrix calculated by the result of FVI is S_1, and S_2 is the solution to the Recatti Equation. We have

$$S_1 = \begin{bmatrix} 27.164022 & 31.758386 \\ 31.758386 & 85.950968 \end{bmatrix}, \quad S_2 = \begin{bmatrix} 27.164022 & 31.758386 \\ 31.758386 & 85.950968 \end{bmatrix} \tag{3.9}$$

S_1 and S_2 are almost the same, which means that we obtained the optimal cost-to-go function using fitted value iteration. It is worth noting that the cost-to-go function in Linear Quadratic Regulator is also using the linear architecture mentioned above.

$$J(x, S) = x^T S x = \text{tr}(S x^T x) \tag{3.10}$$

It is quadratic in x but linear in S.

3.2.2.2 Neural networks

The selection of features is frequently manually crafted, relying on human intellect, intuition, or experience, and can pose considerable challenges. The utilization of a neural network as the approximation architecture has emerged as a popular approach in recent years. In this context, the parameter r_k may correspond to the weights of the neural networks. A diverse range of machine learning techniques can then be implemented to manage the training problem, steering the approximation toward the optimal value.

However, the optimization process of the weights is more difficult due to the non-convexity. According to the equation (3.3), the J_{k+1} is non-convex which makes the entire equation hard to optimize.

Example 3.2 (Optimal Control for Linear System Using NN-based Method)**.**

In this example, we will use neural network as the approximation of the cost-to-go function and conduct neural FVI on a double integrator. The dynamics of the double integrator has been introduced in example 3.1. We use a semi-positive-definite network to model the cost-to-go function.

$$J(x) = N(x)^T N(x) \tag{3.11}$$

$N(x)$ is a 3-layer Multi-Layer-Perceptron with ReLU activation functions. Using mini-batch learning plus Adam optimizer, we obtain the cost-to-go in Fig. 3.2 after 300 epochs of training.

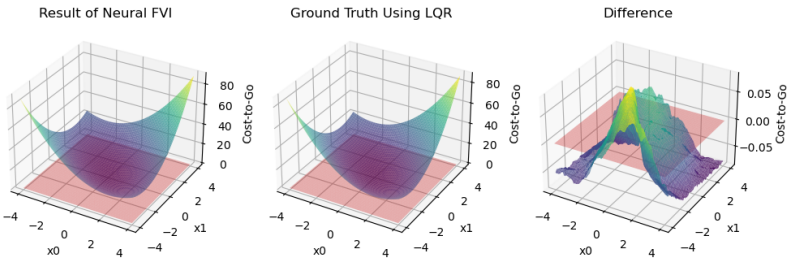

Figure 3.2: Comparison between the result of NN-based FVI and ground truth

The figure shows that the approximation performance of NN is pretty good. Simulation experiments also shows that the corresponding controller could successfully regulate the system at $(0, 0)$.

3.2.3 Online approximate optimization

Different from previous sections, in this section, we will discuss *online* approaches for computing the one-step lookahead control u_k just after the current

state x_k becomes known. Here, to compute u_k, the values $\tilde{J}_{k+1}(x_{k+1})$ need only be computed at the relevant next states x_{k+1} (the ones that can occur following application of u_k).

A particularly effective online approach is *rollout*. In rollout algorithm, $\tilde{J}_{k+1}(x_{k+1})$ is calculated by a *suboptimal policy*, or *base policy*. $\tilde{J}_{k+1}$ could be calculated either analytically or by Monte Carlo simulation. This part is interconnected with *model predictive control (MPC)*, which we will also discuss at the end of this section.

3.2.3.1 Rollout algorithm

The essence of the rollout is policy improvement, which generates a better policy on top of the base policy. In the rollout algorithm, $\tilde{J}_{k+1}$ is the cost-to-go of some known suboptimal policy $\pi = \{\mu_0, ..., \mu_{N-1}\}$, referred to as *base policy*. The policy $\bar{\pi} = \{\bar{\mu}_0, ..., \bar{\mu}_{N-1}\}$ thus obtained is called the *rollout policy* based on π. In short, *the rollout* policy is the one-step lookahead policy, with the optimal cost-to-go approximated by the cost-to-go of the base policy_.

Definition 3.1 (One-step Rollout Algorithm) We can get an improved policy from the base policy π

$$\bar{\mu}_k(x_k) = \arg \min_{u_k \in \mathbb{U}_k(x_k)} E\left\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\right\} \tag{3.12}$$

where $\tilde{J}_{k+1}$ is the corresponding cost-to-go function of the base policy π. If we use H_{k+1} to represent the cost-to-go function of the base policy π, the rollout algorithm will be:

$$\bar{\mu}_k(x_k) = \arg \min_{u_k \in \mathbb{U}_k(x_k)} E\left\{g_k(x_k, u_k, w_k) + H_{k+1}(f_k(x_k, u_k, w_k))\right\} \tag{3.13}$$

In the control system, after the current state x_k is revealed, we calculate the cost-to-go function H_{k+1} of the known base policy π and conduct one-step lookahead minimization to find $\bar{\mu}_k(x_k)$ and feed it into the system immediately.

Note that it is also possible to define the rollout policy that makes use of multistep lookahead. While such multistep lookahead involves much more online computation, it will likely yield better performance than its one-step counterpart. In what follows, we concentrate on rollout policy with one-step lookahead.

Theorem 3.1 (Cost improvement property of rollout algorithm)**.** *It is possible to show that the rollout policy's performance is no worse than the one of the base policy, while some special conditions must hold to guarantee this cost improvement property. Here we introduce the* sequential improvement *condition. We say that the base policy has sequential improvement property if, for all x_k and k, we have*

$$\min_{u_k \in \mathbb{U}_k(x_k)} \{g_k(x_k, u_k) + H_{k+1}(f_k(x_k, u_k))\} \leq H_k(x_k) \tag{3.14}$$

where $H_k(x_k)$ denotes the cost of the base policy starting from x_k. Here we use deterministic problems to make our proof concise. Sometimes people also use the Q factor *mentioned below:*

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}(f_k(x_k, u_k)) \tag{3.15}$$

so now the sequential improvement property could also be written as:

$$\min_{u_k \in \mathbb{U}_k(x_k)} \tilde{Q}_k(x_k, u_k) \leq H_k(x_k) \tag{3.16}$$

We will now show that the rollout algorithm obtained with a base policy with sequential improvement property yields no worse cost than the base policy. In particular, consider the rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, ..., \tilde{\mu}_{N-1}\}$, and let $J_{k,\tilde{\pi}}(x_k)$ denote the cost obtained with $\tilde{\pi}$ starting from x_k. We claim that

$$J_{k,\tilde{\pi}}(x_k) \leq H_k(x_k), for\ all\ x_k\ and\ k \tag{3.17}$$

Proof. We prove this inequality by induction. Clearly it holds for $k = N$, since $J_{N,\tilde{\pi}} = H_N = g_N$. Assume it holds for index $k + 1$. We have:

$$\begin{aligned}
\tilde{J}_{k,\tilde{\pi}}(x_k) &= g_k(x_k, \tilde{\mu}_k(x_k)) + J_{k+1,\tilde{\pi}}(f_k(x_k, \tilde{\mu}_k(x_k))) && (3.18)\\
&\leq g_k(x_k, \tilde{\mu}_k(x_k)) + H_{k+1}(f_k(x_k, \tilde{\mu}_k(x_k))) && (3.19)\\
&= \min_{u_k \in \mathbb{U}_k(x_k)} [g_k(x_k, u_k) + H_{k+1}(f_k(x_k, u_k))] && (3.20)\\
&= \min_{u_k \in \mathbb{U}_k(x_k)} \tilde{Q}_k(x_k, u_k) && (3.21)\\
&\leq H_k(x_k) && (3.22)
\end{aligned}$$

where:

a. The first equality is the DP equation for the rollout policy $\tilde{\pi}$.

b. The first inequality holds by the induction hypothesis.

c. The second equality holds by the definition of the rollout algorithm.

d. The second inequality holds by the sequential improvement property.

This completes the induction proof of the cost improvement property (3.17). □

Computational issues in rollout algorithms. In the rollout algorithm, the cost-to-go function H_{k+1} of the base policy is required to be computed online at all possible next states $f_k(x_k, u_k, w_k)$. However, the real-time constraint will be a critical problem, for the corresponding cost-to-go function of a given base policy is not easy to calculate in real-time. In most cases, we will use the approximate version of the cost-to-go $\tilde{H}_{k+1}$ to simplify the calculation. So the rollout algorithm will be:

$$\bar{\mu}_k(x_k) = \arg \min_{u_k \in \mathbb{U}_k(x_k)} E\left\{g_k(x_k, u_k, w_k) + \tilde{H}_{k+1}(f_k(x_k, u_k, w_k))\right\} \tag{3.23}$$

There are two variants to handle the computational difficulties, deterministic case, and stochastic case.

1. *Deterministic case*. If the problem is deterministic, the calculation is greatly simplified.
2. *Stochastic case*. In these cases, the $\tilde{H}_{k+1}$ are evaluated online by Monte Carlo simulation for all $u_k \in \mathbb{U}_k(x_k)$.

Truncated rollout algorithm with multistep lookahead and terminal cost approximation. We may incorporate multistep lookahead into the rollout framework. Let us start with a two-step lookahead for deterministic problems. Suppose that after k steps we have reached state x_k. We then consider the set of all two-step-ahead states x_{k+2}, run the base policy starting from each of them, and compute the two-stage cost to get from x_k to x_{k+2}, plus the cost of the base policy from x_{k+2}. We select the state, say $\tilde{x}_{k+2}$, that is associated with minimum cost, compute the controls $\tilde{u}_k$ and $\tilde{u}_{k+1}$ that lead from x_k to $\tilde{x}_{k+2}$, and choose $\tilde{u}_k$ as the next rollout control and $x_{k+1} = f_k(x_k, \tilde{u}_k)$ as the next state.

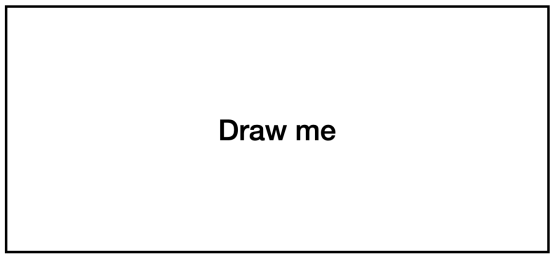

Figure 3.3: Illustration of truncated rollout with two-step lookahead

The extension of the algorithm to lookahead of more than two steps is straightforward: instead of the two-step-ahead states x_{k+2} we run the base policy starting from all the possible l-step ahead states x_{k+l}, etc.

An important variation for problems with a large number of stages is *truncated rollout with terminal cost approximation*. Here the rollout trajectories are obtained by running the base policy from the leaf nodes of the lookahead tree, and they are truncated after a given number of steps, while a terminal cost approximation is added to the policy cost to compensate for the resulting error. One possibility that works well for many problems is to simply set the terminal cost approximation to zero. Alternatively, the terminal cost function approximation may be obtained by problem approximation or by using some sophisticated offline training process that may involve an approximation architecture such as a neural network.

3.2.3.2 Model predictive control (MPC)

In this section, we will discuss a popular control algorithm called *model predictive control (MPC)*. We will start by considering the case where the objective is to keep the state close to the origin (or more generally some point of interest, called the *set point*, or *fixed point*); this is called the *regulation problem*. Similar approaches have been developed for the problem of maintaining the state of a non-stationary system along a given state trajectory, and also, with appropriate modifications, to control problems involving disturbances. In particular, in some cases, the trajectory is treated like a sequence of set points, and the subsequently described algorithm is applied repeatedly.

We will consider a deterministic system

$$x_{k+1} = f_k(x_k, u_k) \tag{3.24}$$

whose state x_k and control u_k are vectors that consist of a finite number of scalar components. The cost per stage is assumed nonnegative

$$g_k(x_k, u_k) \geq 0, for\ all\ (x_k, u_k) \tag{3.25}$$

(e.g., a quadratic cost). We impose state and control constraints

$$x_k \in \mathbb{X}_k, u_k \in \mathbb{U}_k(x_k), k = 0, 1, ... \tag{3.26}$$

We also assume that the system can be kept at the origin at zero cost, i.e.,

$$f_k(0, \bar{u}_k) = 0, g_k(0, \bar{u}_k) = 0 \tag{3.27}$$

for some control $\bar{u}_k \in \mathbb{U}_k(0)$. This is a characteristic that all fixed points possess.

For a given initial state $x_0 \in \mathbb{X}_0$, we want to obtain a sequence $\{u_0, u_1, ...\}$ such that the states and controls of the system satisfy the state and control constraints with a small total cost.

The MPC algorithm. Let us describe the MPC algorithm for the deterministic problem just described. At the current state x_k:

1. MPC solves an l-step lookahead version of the problem, which requires that $x_{k+l} = 0$.
2. If $\{\tilde{u}_k, ..., \tilde{u}_{k+l-1}\}$ is the optimal control sequence of this problem, MPC applies $\tilde{u}_k$ and discards the other controls $\tilde{u}_{k+1}, ..., \tilde{u}_{k+l-1}$.
3. At the next stage, MPC repeats this process, once the next state x_{k+1} is revealed.

In some literature, this MPC algorithm is also called *Receding Horizon Control* algorithm, or RHC for short. One obvious drawback of this method is the online computation time limit. The MPC algorithm needs to solve an optimization problem online, which is time-consuming and does not guarantee a solution.

To make the connection between MPC and rollout, we first recap the case of the truncated rollout algorithm. In a truncated rollout algorithm with multistep lookahead and terminal cost approximation,

$$\min_{u_k \in \mathbb{U}_k(x_k), ..., u_{k+l} \in \mathbb{U}_{k+l}(x_{k+l})} \left\{ \sum_{i=k}^{k+l} g_i(x_i, u_i) + \sum_{i=k+l+1}^{k+l+m} g_i(x_i, \mu_i(x_i)) + \tilde{J}(x_{k+l+m+1}) \right\} \tag{3.28}$$

such that

$$x_{i+1} = f_i(x_i, u_i) \tag{3.29}$$

The control u_k will be used as the control at step k (online current step). All the x_i are admissible states. Here $\tilde{J}$ means the terminal cost approximation, which can be obtained through offline computation or sometimes be set to zero. l means the number of lookahead steps, and m means the number of steps that the base policy runs to evaluate the cost-to-go function H_{k+l+1}. Let us discuss a special case, where the $\tilde{J}$ is set to zero while m is also zero. So now the rollout algorithm becomes:

$$\min_{u_k \in \mathbb{U}_k(x_k), ..., u_{k+l} \in \mathbb{U}_{k+l}(x_{k+l})} \sum_{i=k}^{k+l} g_i(x_i, u_i) \tag{3.30}$$

while u_k still be used as the current online control, and all other optimized controls are discarded. We can see that now it is *almost* the case of model predictive control, without the terminal state constraint (in this case the terminal state constraint is $x_{k+l+1} = 0$). This constraint is also called *recursive feasibility*, for it guarantees the optimization will not suddenly encounter a situation where the solver returns "infeasible".

3.3 Approximation in policy space

A major alternative to approximation in value space is *approximation in policy space*, whereby we select the policy from a suitably restricted class of policies, usually a parametric class of some form. In particular, we can introduce a parametric family of policies

$$\mu_k(x_k, r_k), k = 0, ..., N-1 \tag{3.31}$$

where r_k is a parameter, such as a family represented by a neural network, and then estimate the parameters r_k using some type of optimization.

An important advantage of approximation in policy space is that the computation of controls during the online operation of the system is often much easier compared with the lookahead minimization (3.3). In this section, we will present two distinct approaches for computing r: *training by cost optimization* and *training by using an expert*.

3.3.1 Training by using an expert

This approach is pretty similar to *supervised learning* in machine learning. We r_k by "training" on a large number of sample state-control pairs $(x_k^s, u_k^s), s = 1, ..., q$, such that for each s, u_k^s is a "good" control at state x_k^s. This can be done for example by solving for each k the least squares problem

$$\min_{r_k} \sum_{s=1}^{q} \left\| u_k^s - \tilde{\mu}_k(x_k^s, r_k) \right\|^2 \tag{3.32}$$

(possibly with added regularization). In particular, we may determine u_k^s by a human or a software "expert" that can choose "near-optimal" controls at the given states x_k^s, so $\tilde{\mu}_k$ is trained to match the behavior of the expert. Of course, in the expert training approach, we cannot expect to obtain a controller that performs better than the expert with which it is trained.

The "near-optimal" controls of sampled states $x_k^s, s = 1, ..., q$ could also be calculated from one-step lookahead minimization with a suitable approximation $\tilde{J}_{k+1}$.

$$u_k^s = \arg \min_{u_k \in \mathbb{U}_k(x_k)} \mathbb{E}\left\{ g_k(x_k^s, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k^s, u_k, w_k)) \right\} \tag{3.33}$$

3.3.2 Training by cost optimization

POLICY GRADIENT

3.4 Extension

It is possible for a suboptimal control scheme to employ both types of approximation: in policy space and in value space, with a distinct architecture for each case. This is known as the simultaneous use of a "policy network" (or "actor network") and a "value network" (or "critic network"), each with its own set of parameters. Simultaneous approximation in policy space and value space through the use of deep neural networks are central in AlphaGo and AlphaZero, DeepMind's Go and chess playing programs.

Chapter 4

Continuous-time Optimal Control

So far we have been focusing on stochastic and discrete-time optimal control problems. In this Chapter, we will switch gear to deterministic and continuous-time optimal control (still with continuous state and action space).

The goal of a continuous-time introduction is threefold. (1) Real-world systems are natively continuous-time. (2) We will see the continuous-time analog of the Bellman principle of optimality in discrete-time (cf. Theorem 1.1). (3) The continuous-time setup is more natural and popular for stability analysis to be introduced in Chapter 5.

4.1 The Basic Problem

Consider a continuous-time dynamical system

$$\dot{x}(t) = f(x(t), u(t)),\ t \in [0, T], \quad x(0) = x_0, \tag{4.1}$$

where

- $x(t) \in \mathbb{R}^n$ is the state of the system,
- $u(t) \in \mathbb{U} \subseteq \mathbb{R}^m$ is the control we wish to design,
- $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ models the system dynamics, and
- $x_0 \in \mathbb{R}^n$ is the initial state of the system.

We assume the admissible control functions $\{u(t) \mid u(t) \in \mathbb{U}, t \in [0,T]\}$, also called control trajectories, must be *piecewise continuous*.[1] For any control trajectory, we assume the system (4.1) has a unique solution $\{x(t) \mid t \in [0,T]\}$, called the state trajectory.

We now state the continuous-time optimal control problem.

Definition 4.1 (Continuous-time, Finite-horizon Optimal Control)**.** Find the best admissible control trajectory $\{u(t) \mid t \in [0,T]\}$ that minimizes the cost function

$$J(0,x_0) = \min_{u(t)\in\mathbb{U}} h(x(T)) + \int_0^T g(x(t),u(t))dt, \tag{4.2}$$

subject to (4.1), where the functions g and h are continuously differentiable with respect to x, and g is continuous with respect to u.

The function J in (4.2) is called the *optimal cost-to-go*, or the *optimal value function*. Notice that the optimal cost-to-go is a function of both the state x and the time t, just as in the discrete-time case we used J_k with a subscript k to denote the optimal cost-to-go for the tail problem starting at timestep k (cf. Theorem 1.2). Specifically, we should interpret

$$J(t,x_0) = \min_{u(t)\in\mathbb{U}} h(x(T)) + \int_t^T g(x(\tau),u(\tau))d\tau, \quad x(t) = x_0,$$

as the optimal cost-to-go of the system starting from x_0 at time t (i.e., the tail problem). We assume $J(0,x_0)$ is finite when x_0 is in some set $\mathcal{X}_0$.

4.2 The Hamilton-Jacobi-Bellman Equation

Recall that in discrete-time, the dynamic programming (DP) algorithm in Theorem 1.2 states that the optimal cost-to-go has to satisfy a recursive equation (1.7), i.e., the optimal cost-to-go at time k can be calculated by choosing the best action that minimizes the stage cost at time k plus the optimal cost-to-go at time $k+1$. In the next, we will show a result of similar flavor to (1.7), but in the form of a partial differential equation (PDE), known as the Hamilton-Jacobi-Bellman (HJB) equation.

Let us informally derive the HJB equation by applying the DP algorithm to a discrete-time approximation of the continuous-time optimal control problem. We divide the time horizon $[0,T]$ into N pieces of equal length $\delta = T/N$, and denote

$$x_k = x(k\delta), \quad u_k = u(k\delta), \quad k = 0,1,\dots,N.$$

[1] Even though we write $dx_i(t)/dt$ in the system (4.1), we allow $x(t)$ to be only directionally differentiable at a finite number of points to account for the possible discontinuity of $u(t)$.

We then approximate the continuous-time dynamics (4.1) as

$$x_{k+1} = x_k + \dot{x}_k \cdot \delta = x_k + f(x_k, u_k) \cdot \delta,$$

and the cost function in (4.2) as

$$h(x_N) + \sum_{k=0}^{N-1} g(x_k, u_k) \cdot \delta.$$

This problem now is in the form of a discrete-time, finite-horizon optimal control 1.1, for which we can apply dynamic programming.

Let us use $\tilde{J}(t, x)$ (as opposed to $J(t, x)$) to denote the optimal cost-to-go at time t and state x for the discrete-time approximation. According to (1.7), the DP backward recursion is

$$\tilde{J}(N\delta, x) = h(x), \tag{4.3}$$

$$\tilde{J}(k\delta, x) = \min_{u \in \mathbb{U}} \left[g(x, u) \cdot \delta + \tilde{J}((k+1)\delta, x + f(x, u) \cdot \delta) \right], \quad k = N-1, \dots, 0. \tag{4.4}$$

Suppose $\tilde{J}(t, x)$ is differentiable, we can perform a Taylor-series expansion of $\tilde{J}((k+1)\delta, x + f(x, u)\delta)$ in (4.4) as follows

$$\tilde{J}((k+1)\delta, x+f(x,u)\delta) = \tilde{J}(k\delta, x) + \nabla_t \tilde{J}(k\delta, x) \cdot \delta + \nabla_x \tilde{J}(k\delta, x)^T f(x, u) \cdot \delta + o(\delta),$$

where $o(\delta)$ includes high-order terms that approach zero when δ tends to zero, $\nabla_t \tilde{J}$ and $\nabla_x \tilde{J}$ (a column vector) denote the partial derivates of $\tilde{J}$ with respect to t and x, respectively. Plugging the first-order Taylor expansion into the DP recursion (4.4), we obtain

$$\tilde{J}(k\delta, x) = \min_{u \in \mathbb{U}} \left[g(x, u) \cdot \delta + \tilde{J}(k\delta, x) + \nabla_t \tilde{J}(k\delta, x)\delta + \nabla_x \tilde{J}(k\delta, x)^T f(x, u)\delta + o(\delta) \right]. \tag{4.5}$$

Cancelling $\tilde{J}(k\delta, x)$ from both sides, dividing both sides by δ, and assuming $\tilde{J}$ converges to J uniformly in time and state, i.e.,

$$\lim_{\delta \to 0, k\delta = t} \tilde{J}(k\delta, x) = J(t, x), \quad \forall t, x,$$

we obtain from (4.5) the following partial differential equation

$$0 = \min_{u \in \mathbb{U}} \left[g(x, u) + \nabla_t J(t, x) + \nabla_x J(t, x)^T f(x, u) \right], \quad \forall t, x, \tag{4.6}$$

with the boundary condition $J(T, x) = h(x)$. Equation (4.6) is called the Hamilton-Jacobi-Bellman equation.

Our derivation above is informal, let us formally state the HJB equation.

Theorem 4.1 (Hamilton-Jacobi-Bellman Equation as A Sufficient Condition for Optimality)**.** *Consider the optimal control problem 4.1 for system* (4.1)*. Suppose $V(t,x)$ is a solution to the Hamilton-Jacobi-Bellman equation, i.e., $V(t,x)$ is continuously differentiable and satisfies*

$$0 = \min_{u \in \mathbb{U}} \left[g(x,u) + \nabla_t V(t,x) + \nabla_x V(t,x)^T f(x,u) \right], \quad \forall t, x, \tag{4.7}$$

$$V(T,x) = h(x), \quad \forall x. \tag{4.8}$$

Suppose $\mu^\star(t,x)$ attains the minimum in (4.7) *for all t and x. Let $\{x^\star(t) \mid t \in [0,T]\}$ be the state trajectory obtained from the given initial condition $x(0)$ when the control trajectory $u^\star(t) = \mu^\star(t, x^\star(t))$ is applied, i.e., $x^\star(0) = x(0)$ and for any $t \in [0,T]$, $\dot{x}^\star(t) = f(x^\star(t), \mu^\star(t, x^\star(t)))$ and we assume this differential equation has a unique solution starting at any (t,x) and that the control trajectory $\{\mu^\star(t, x^\star(t)) \mid t \in [0,T]\}$ is piecewise continuous as a function of t. Then $V(t,x)$ is equal to the optimal cost-to-go $J(t,x)$ for all t and x. Moreover, the control trajectory $u^\star(t)$ is optimal.*

Proof. Let $\{\hat{u}(t) \mid t \in [0,T]\}$ be any admissible control trajectory and let $\hat{x}(t)$ be the resulting state trajectory. From the "min" in (4.7), we know

$$0 \le g(\hat{x},\hat{u}) + \nabla_t V(t,\hat{x}) + \nabla_x V(t,\hat{x})^T f(\hat{x},\hat{u}) = g(\hat{x},\hat{u}) + \frac{d}{dt} V(t,\hat{x}).$$

Integrating the above inequality over $t \in [0,T]$, we obtain

$$0 \le \left(\int_0^T g(\hat{x}(t),\hat{u}(t)) dt \right) + V(T,\hat{x}(T)) - V(0,\hat{x}(0)).$$

Using the terminal constraint $V(T,x) = h(x)$ for any x and the initial condition $\hat{x}(0) = x(0)$, we have

$$V(0,x(0)) \le h(\hat{x}(T)) + \int_0^T g(\hat{x}(t),\hat{u}(t)) dt.$$

This shows that $V(0,x(0))$ is a lower bound to the optimal cost-to-go, because any admissible control trajectory $\hat{u}(t)$ leads to a cost no smaller than $V(0,x(0))$.

It remains to show that $V(0,x(0))$ is attainable. This is done by plugging the optimal control trajectory $u^\star(t)$ and state trajectory $x^\star(t)$ to the derivation above, leading to

$$V(0,x(0)) = h(x^\star(T)) + \int_0^T g(x^\star(t),u^\star(t)) dt.$$

This shows that $V(0,x(0)) = J(0,x(0))$.

The argument above is generic and holds for any initial time $t \in [0,T]$ and initial state x. Therefore, $V(t,x) = J(t,x)$ is the optimal cost-to-go. □

Theorem 4.1 effectively turns the optimal control problem (4.2) into finding a solution for the partial differential equation (4.7). Let us illustrate the theorem using a simple example.

Example 4.1 (A Scalar System)**.** Consider the following dynamical system

$$\dot{x}(t) = u(t), \quad t \in [0, T]$$

where $x \in \mathbb{R}$ is the state, and $u \in \mathbb{U} = [-1, 1]$ is the control.

We are interested in the following optimal control problem

$$\min_{u(t)} \frac{1}{2} \left(x(T)\right)^2,$$

where the goal is to move the initial state as close as possible to the origin 0 at the terminal time T.

There is a simple optimal controller for this scalar system. We move the state as quickly as possible to the origin 0, using maximum control, and then maintain the state at the origin using zero control. Formally, this is

$$\mu^\star(t, x) = -\text{sgn}(x) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x > 0 \end{cases}.$$

With this controller, we know that if the system starts at x at time t, the terminal state will satisfy

$$|x(T)| = \begin{cases} |x| - (T - t) & \text{if } T - t < |x| \\ 0 & \text{otherwise} \end{cases}.$$

As a result, the optimal cost-to-go is

$$J(t, x) = \frac{1}{2} \left(\max\{0, |x| - (T - t)\}\right)^2. \tag{4.9}$$

Let us verify if this function satisfies the HJB equation.

Boundary condition. Clearly,

$$J(T, x) = \frac{1}{2} x^2$$

satisfies the boundary condition.

Differentiability. When viewed as a function of t, $J(t, x)$ in (4.9) can be plotted as in Fig. 4.1. We can see that $J(t, x)$ is differentiable in t and

$$\nabla_t J(t, x) = \max\{0, |x| - (T - t)\}. \tag{4.10}$$

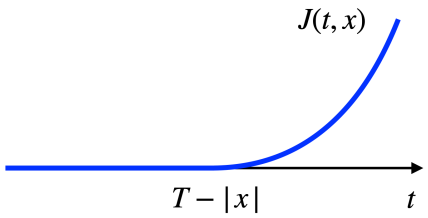

Figure 4.1: Optimal cost-to-go as a function of time.

Figure 4.2: Optimal cost-to-go as a function of state.

When viewed as a function of x, $J(t,x)$ can be plotted as in Fig. 4.2. We can see $J(t,x)$ is differentiable in x and

$$\nabla_x J(t,x) = \text{sgn}(x) \cdot \max\{0, |x| - (T-t)\}. \tag{4.11}$$

PDE. Substituting (4.10) and (4.11) into the HJB equation (4.7), we need to verify that the followng equation holds for all t and x

$$0 = \min_{u \in \mathbb{U}} (1 + \text{sgn}(x) \cdot u) \max\{0, |x| - (T-t)\}. \tag{4.12}$$

This is easy to verify as $u = -\text{sgn}(x)$ attains the minimum and sets the right-hand side equal to zero for any (t,x).

However, we can observe that the optimal controller need not be unique. For example, when $|x| \leq T - t$, we have

$$\max\{0, |x| - (T-t)\} = 0,$$

and any $u \in \mathbb{U}$ would attain the minimum in (4.12) and hence be optimal.

4.3 Linear Quadratic Regulator

4.4 The Pontryagin Minimum Principle

The HJB equation in Theorem 4.1 provides a *sufficient* condition for the optimal cost-to-go. However, since the HJB equation is a sufficient condition, there

do exist cases where the optimal cost-to-go does not satisfy the HJB equation but is still optimal (e.g., when the optimal cost-to-go is not continuously differentiable).

We now introduce a *necessary* condition that any optimal control trajectory and state trajectory must satisfy. This condition is the celebrated Pontryagin minimum principle.

A rigorous derivation of the Pontryagin minimum principle can be mathematically involving and is beyond the scope of this lecture notes (see Section 7.3.2 in (Bertsekas, 2012) for a more rigorous treatment). In the following, we provide an informal derivation of the Pontryagin minimum principle.

Recall the HJB equation in Theorem 4.1 states that, if a control trajectory $u^\star(t)$ and the associated state trajectory $x^\star(t)$ is optimal, then for all $t \in [0, T]$, the following condition must hold

$$u^\star(t) = \arg\min_{u \in \mathbb{U}} \left[g(x^\star(t), u) + \nabla_x J(t, x^\star(t))^T f(x^\star(t), u) \right]. \tag{4.13}$$

The above equation says, in order to compute the optimal control, we do not need to know the value of $\nabla_x J$ at *all* possible values of x and t (which is what the HJB equation tries to do), and we only need to know the value of $\nabla_x J$ along the *optimal* trajectory, i.e., to know only $\nabla_x J(t, x^\star(t))$.

The Pontryagin minimum principle builds upon this key observation, and it points out that $\nabla_x J(t, x^\star(t))$ (but not $\nabla_x J(t, x)$ for any x) satisfies a certain differential equation called the *adjoint equation*.

We now provide an informal derivation of the adjoint equation that is based on differentiating the HJB equation. Towards this goal, we first present the following lemma which is itself quite useful.

Lemma 4.1 (Differentiating Functions Involving Minimization)**.** *Let $F(t, x, u)$ be a continuously differentiable function of $t \in \mathbb{R}$, $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, and let $\mathbb{U}$ be a convex subset of $\mathbb{R}^m$. Suppose $\mu^\star(t, x)$ is a continuously differentiable function such that*

$$\mu^\star(t, x) = \arg\min_{u \in \mathbb{U}} F(t, x, u), \quad \forall t, x.$$

Then

$$\nabla_t \left\{ \min_{u \in \mathbb{U}} F(t, x, u) \right\} = \nabla_t F(t, x, \mu^\star(t, x)), \quad \forall t, x, \tag{4.14}$$

$$\nabla_x \left\{ \min_{u \in \mathbb{U}} F(t, x, u) \right\} = \nabla_x F(t, x, \mu^\star(t, x)), \quad \forall t, x. \tag{4.15}$$

In words, the partial derivates (with respect to t and x) of "the minimum of $F(t, x, u)$ over u" (commonly known in optimization as the value function $\psi(t, x)$ of $F(t, x, u)$) are equal to the partial derivates of $F(t, x, u)$ (with respect to t and x) after plugging in the optimizer $\mu^\star(t, x)$.

We now start with the HJB equation in (4.7), restated below with $V(t,x)$ replaced by the optimal $J(t,x)$ for the reader's convenience

$$0 = \min_{u \in \mathbb{U}} \left[g(x,u) + \nabla_t J(t,x) + \nabla_x J(t,x)^T f(x,u) \right]. \tag{4.16}$$

Assume that $\mu^\star(t,x)$ attains the minimum in the equation above and it is also continuously differentiable. Note that we have made the restrictive assumption that $\mathbb{U}$ is convex and $\mu^\star(t,x)$ is continuously differentiable, which are not necessary in a more rigorous derivation of Pontryagin's principle (cf. Section 7.3.2 in (Bertsekas, 2012)).

We differentiate both sides of (4.16) with respect to t and x. In particular, let

$$F(t,x,u) = g(x,u) + \nabla_t J(t,x) + \nabla_x J(t,x)^T f(x,u)$$

and invoke Lemma 4.1, we can write

$$0 = \nabla_x g(x, \mu^\star(t,x)) + \nabla^2_{xt} J(t,x) + \nabla^2_{xx} J(t,x) f(x, \mu^\star(t,x)) + \nabla_x f(x, \mu^\star(t,x)) \nabla_x J(t,x). \tag{4.17}$$

$$0 = \nabla^2_{tt} J(t,x) + \nabla^2_{xt} J(t,x)^T f(x, \mu^\star(t,x)) \tag{4.18}$$

where the first equation results from differentiation of (4.16) with respect to x, and the second equation results from differentiation of (4.16) with respect to t. In (4.17), $\nabla_x f(x, \mu^\star(t,x))$ is

$$\nabla_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_n}{x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

evaluated at $(x, \mu^\star(t,x))$. Equations (4.17) and (4.18) hold for any (t,x) (under the restrictive assumptions we have made).

We then evaluate the equations (4.17) and (4.18) only along the optimal control and state trajectory $(u^\star(t), x^\star(t))$ that satisfies

$$\dot{x}^\star(t) = f(x^\star(t), u^\star(t)), \quad u^\star(t) = \mu^\star(t, x^\star(t)), \quad t \in [0,T]. \tag{4.19}$$

Specifically, along the optimal trajectory, we have

$$\frac{d}{dt} \left(\nabla_x J(t, x^\star(t)) \right) = \nabla^2_{xt} J(t, x^\star(t)) + \nabla^2_{xx} J(t, x^\star(t)) f(x^\star(t), u^\star(t)),$$

where the right-hand side contains terms in the right-hand side of (4.17) (when evaluted along the optimal trajectory). Similarly,

$$\frac{d}{dt} \left(\nabla_t J(t, x^\star(t)) \right) = \nabla^2_{tt} J(t, x^\star(t)) + \nabla^2_{xt} J(t, x^\star(t))^T f(x^\star(t), u^\star(t)),$$

where the right-hand side is exactly the right-hand side of (4.18) (when evaluted along the optimal trajectory). As a result, equations (4.17) and (4.18), when evaluated along the optimal trajectory, are equivalent to

$$0 = \nabla_x g(x^\star(t), u^\star(t)) + \frac{d}{dt}\left(\nabla_x J(t, x^\star(t))\right) + \nabla_x f(x^\star(t), u^\star(t)) \nabla_x J(t, x^\star(t)). \tag{4.20}$$

$$0 = \frac{d}{dt}\left(\nabla_t J(t, x^\star(t))\right). \tag{4.21}$$

Therefore, if we denote

$$p(t) = \nabla_x J(t, x^\star(t)), \quad p_0(t) = \nabla_t J(t, x^\star(t)),$$

then equations (4.20) and (4.21) become

$$\dot{p}(t) = -\nabla_x f(x^\star(t), u^\star(t)) p(t) - \nabla_x g(x^\star(t), u^\star(t)), \tag{4.22}$$

$$\dot{p}_0(t) = 0. \tag{4.23}$$

Equation (4.22), which is a system of n first-order differential equations, is known as the adjoint equation and it describes the evolution of $p(t)$, known as the *costate*, along the optimal trajectory. To obtain a boundary condition for the adjoint equation (4.22), we note that the boundary condition of the HJB equation

$$J(T, x) = h(x), \quad \forall x$$

implies

$$p(T) = \nabla h(x^\star(T)).$$

This is basically the Pontryagin Minimum Principle.

The Hamiltonian formulation. It is usually more convenient to state the Pontryagin Principle using the concept of a *Hamiltonian*. Formally, we define the Hamiltonian function that maps the triplet $(x, u, p) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n$ to real numbers given by

$$H(x, u, p) = g(x, u) + p^T f(x, u).$$

Note that the dynamics along the optimal trajectory (4.19) can be conveniently written as

$$\dot{x}^\star(t) = \nabla_p H(x^\star(t), u^\star(t), p(t)),$$

and the adjoint equation (4.22) can be written as

$$\dot{p}(t) = -\nabla_x H(x^\star(t), u^\star(t), p(t)).$$

We are now ready to state the Pontryagin Minimum Principle.

Theorem 4.2 (Pontryagin Minimum Principle as A Necessary Condition for Optimality)**.** *Let $(u^\star(t), x^\star(t)), t \in [0, T]$ be a pair of optimal control and state trajectories satisfying*

$$\dot{x}^\star(t) = f(x^\star(t), u^\star(t)), \quad x^\star(0) = x_0 \text{ given.}$$

Let $p(t)$ be the solution of the adjoint equation

$$\dot{p}(t) = -\nabla_x H(x^\star(t), u^\star(t), p(t)),$$

with the boundary condition

$$p(T) = \nabla h(x^\star(T)),$$

where h is the terminal cost function. Then, for all $t \in [0, T]$, we have

$$u^\star(t) = \arg\min_{u \in \mathbb{U}} H(x^\star(t), u, p(t)).$$

Moreover, there is a constant C such that

$$H(x^\star(t), u^\star(t), p(t)) = C, \quad \forall t \in [0, T].$$

To see why $H(x^\star(t), u^\star(t), p(t))$ is a constant along the optimal trajectory, we observe that from the HJB equation (4.16), we obtain

$$g(x^\star, u^\star) + \nabla_t J(t, x^\star) + \nabla_x J(t, x^\star)^T f(x^\star, u^\star) = 0 \tag{4.24}$$

$$\implies \underbrace{g(x^\star, u^\star) + \nabla_x J(t, x^\star)^T f(x^\star, u^\star)}_{H(x^\star(t), u^\star(t), p(t))} = -\underbrace{\nabla_t J(t, x^\star)}_{p_0(t)}. \tag{4.25}$$

From (4.23), we know $p_0(t)$ is a constant.

A necessary condition. It is important to recognize that the Pontryagin Minimum Principle in Theorem 4.2 is a necessary condition for optimality, i.e., all optimal control and state trajectories must satisfy this condition, but *not all* trajectories satisfying the condition are optimal. Extra arguments are needed to guarantee optimality. One common strategy is to show that an optimal control trajectory exists, and then verify that there is only one control trajectory satisfying the conditions of the Minimum Principle (or that all trajectories verifying the Minimum Principle have equal costs). A setup where the Minimum Principle is both necessary is sufficient is when $f(x, u)$ is linear in (x, u), the the constraint set U is convex, and the cost functions h and g are convex.

Two-point boundary problem. The Pontryagin Minimum Principle is particularly useful when

$$u^\star = \arg\min_{u \in \mathbb{U}} H(x^\star, u, p) = \arg\min_{u \in \mathbb{U}} g(x^\star, u) + p^T f(x^\star, u) \tag{4.26}$$

can be solved analytically so that $u^\star$ becomes a function of $x^\star$ and p. For example, this is possible when problem (4.26) is a convex problem, for which

one can invoke the KKT optimality conditions (cf. Appendix B.1.4). Once $u^\star(t)$ is expressed as a function of $x^\star(t)$ and $p(t)$, we can merge the system equation (4.19) and the adjoint equation (4.22) together and arrive at

$$\begin{cases} \dot{x}^\star(t) = f(x^\star(t), u^\star(t)) \\ \dot{p}(t) = -\nabla_x f(x^\star(t), u^\star(t))p(t) - \nabla_x g(x^\star(t), u^\star(t)) \end{cases}, \tag{4.27}$$

which is a set of $2n$ first-order differential equations in $x^\star(t)$ and $p(t)$. The boundary conditions are

$$x^\star(0) = x_0, \quad p(T) = \nabla h(x^\star(T)). \tag{4.28}$$

The number of boundary conditions is also $2n$, so generally we expect to be able to solve these differential equations numerically.

4.5 Infinite-Horizon Problems

4.6 Viscosity Solution

Chapter 5

Stability Analysis

Optimal control formulates a control problem via the language of mathematical optimization. However, there are control problems, and sometimes even the very basic control problems, that cannot be easily stated in the optimal control formulation.

For example, suppose our goal is to *swing up a pendulum to the upright position and stabilize it there*. You may want to formalize the problem as

$$\min_{u(t)\in\mathbb{U}} \int_0^\infty \|x(t) - x_d\|^2 dt, \quad \text{subject to} \quad \dot{x} = f(x,u), x(0) = x_0, \tag{5.1}$$

where x_d is the desired upright position for the pendulum. However, does the solution of problem (5.1), if exists, guarantee the stabilization of the pendulum at the upright position? The answer is unclear without a rigorous proof.

However, after a slight change of perspective, the optimal control problem may be formulated to better match the goal. Suppose there exists a region, Ω, in the state space such that as long as the pendulum enters Ω, there always exists a sequence of control to bring the pendulum to the goal state x_d, then we can simply formulate a different optimal control problem

$$\min_{u(t)\in\mathbb{U}} \int_0^T \|u(t)\|^2 dt, \quad \text{subject to} \quad x(0) = x_0, x(T) \in \Omega, \dot{x} = f(x,u), \tag{5.2}$$

where now it is very clear, if a solution exists to problem (5.2), then we will definitely achieve our goal. This is because the constraint $x(T) \in \Omega$ guarantees that we will be able to stabilize the pendulum, and the cost function of (5.2) simply encourages minimum control effort along the way.

This highlights that, sometimes the formulation of a problem may deserve more thoughts than the actual solution. Of course the formulation (5.2) may be much

more difficult to solve. In fact, does the set Ω exist, and if so, how to describe it?

This is the main focus of this chapter: to introduce tools that can help us analyze the *stability* of uncontrolled and controlled nonlinear systems. Specifically, we will introduce the notion of *stability certificates*, which are conditions that, if hold, certify the stability of the system (e.g., in the set Ω). Interestingly, you will see that the notion of stability certificates is intuitive and easy, but what is really challenging is to *find* and *compute* the stability certificates. We will highlight the power and also limitation of computational tools, especially those that are based on convex optimization (see Appendix B for a review of convex optimization).

5.1 Autonomous Systems

Let us first focus on autonomous systems, i.e., systems whose dynamics do not depent on time (and control). We introduce different concepts of stability and ways to certify them.

5.1.1 Concepts of Stability

Consider the autonomous system

$$\dot{x} = f(x) \tag{5.3}$$

where $x \in \mathbb{X} \subseteq \mathbb{R}^n$ is the state and $f : \mathbb{R}^n \to \mathbb{R}^n$ is the (potentially nonlinear) dynamics.

Before talking about concepts of stability, we need to define an *equilibrium point*.

Definition 5.1 (Equilibrium Point)**.** A state $x^\star$ is called an equilibrium point of system (5.3) if $f(x^\star) = 0$, i.e., once the system reaches $x^\star$, it stays at $x^\star$.

For example, a linear system

$$\dot{x} = Ax$$

has a single equilibrium point $x^\star = 0$ when A is nonsingular, and an infinite number of equilibrium points when A is singular (those equilibrium points lie in the kernel of matrix A).

When analyzing the behavior of a dynamical system around the equilibrium point, it is often helpful to "shift" the dynamics equation so that 0 is the equilibrium point. For example, if we are interested in the behavior of system (5.3) near the equilibrium point $x^\star$, we can create a new variable

$$z = x - x^\star,$$

so that

$$\dot{z} = \dot{x} = f(x) = f(z + x^\star). \tag{5.4}$$

Clearly, $z^\star = 0$ is an equilibrium point for the shifted system (5.4).

Let us find the equilibrium points of a simple pendulum.

Example 5.1 (Equilibrium Points of A Simple Pendulum)**.** Consider the dynamics of an uncontrolled pendulum

$$\begin{cases} \dot{\theta} = \dot{\theta} \\ \ddot{\theta} = -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta) \end{cases} \tag{5.5}$$

where θ is the angle between the pendulum and the vertical line, and $x = [\theta, \dot{\theta}]^T$ is the state of the pendulum (m, g, l, b denote the mass, gravity constant, length, and damping constant, respectively).

To find the equilibrium points of the pendulum, we need the right hand sides of (5.5) to be equal to zero:

$$\dot{\theta} = 0, \quad -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta) = 0.$$

The solutions are easy to find

$$x^\star = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \pi \\ 0 \end{bmatrix},$$

corresponding to the bottomright and upright positions of the pendulum, respectively.

The pendulum dynamics has two equilibrium points, but our physics intuition tells us these two equilibrium points are dramatically different. Specifically, the bottomright equilibrium $x^\star = [0, 0]^T$ is such that if you perturb the pendulum around the equilibrium, the pendulum will go back to that equilibrium; the upright equilibrium $x^\star = [\pi, 0]^T$ is such that if you perturb the pendulum (even just a little bit) around the equilibrium, it will diverge from that equilibrium.

This physical intuition is exactly what we want to formalize as the concepts of stability.

In the following, we focus on the nonlinear autonomous system (5.3) with $f(0) = 0$, i.e., $x^\star = 0$ is an equilibrium point. We now formally define the different concepts of stability.

Definition 5.2 (Lyapunov Stability)**.** The equilibrium point $x = 0$ is said to be *stable in the sense of Lyapunov* if, for any $R > 0$, there exists $r > 0$ such that if $\|x(0)\| < r$, then $\|x(t)\| < R$ for all $t \geq 0$. Otherwise, the equilibrium point is unstable.

For a system that is Lyapunov stable around $x = 0$, the definition says that, if we want to constrain the trajectory of the system to be within the ball $B_R = \{x \mid \|x\| < R\}$, then we can always find a smaller ball $B_r = \{x \mid \|x\| < r\}$ such that if the system starts within B_r, it will remain in the larger ball B_R.

On the other hand, if the system is not Lyapunov stable at $x = 0$, then there exists at least one ball B_R, such that no matter how close the system's initial condition is to the origin, it will eventually exit the ball B_R. The following exercise is left for you to verify the instability of the Van der Pol oscillator.

Exercise 5.1 (Instability of the Van der Pol oscillator)**.** Show that the Van der Pol oscillator

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -x_1 + (1 - x_1^2)x_2 \end{cases}$$

is unstable at the equilibrium point $x = 0$.

Lyapunov stability does not guarantee the system trajectory will actually converge to $x = 0$. Instead, asymptotic stability will ask the system trajectory to converge to $x = 0$.

Definition 5.3 (Asymptotic Stability and Domain of Attraction)**.** The equilibrium point $x = 0$ is said to be *asymptotically stable* if (i) it is Lyapunov stable, and (ii) there exists some $r > 0$ such that $x(0) \in B_r$ implies $x(t) \to 0$ as $t \to \infty$.

The domain of attraction (for the equilibrium $x = 0$) is the largest set of points in the state space such that trajectories initiated at those points will converge to the equilibrium point. That is,

$$\Omega(x^\star) = \{x \in \mathbb{X} \mid x(0) = x \Longrightarrow \lim_{t\to\infty} x(t) = x^\star\}.$$

The ball B_r is *a* domain of attraction for the equilibrium point $x = 0$, but not necessarily the largest domain of attraction.

You may immediately realize that in the definition of asymptotic stability, we require Lyapunov stability to hold first. Is this necessary? i.e., does there exist a system where trajectories eventually converge to zero, but is not stable in the sense of Lyapunov? You should work out the following exercise.

Exercise 5.2 (Vinograd System)**.** Show that for the Vinograd dynamical system (Vinograd, 1957)

$$\begin{cases} \dot{x} = \frac{x^2(y-x)+y^5}{(x^2+y^2)(1+(x^2+y^2)^2)} \\ \dot{y} = \frac{y^2(y-2x)}{(x^2+y^2)(1+(x^2+y^2)^2)} \end{cases},$$

all system trajectories converge to the equilibrium point $(x, y) = 0$, but the equilibrium point is not stable in the sense of Lyapunov.

(Hint: the system trajectories will behave like the following plot.)

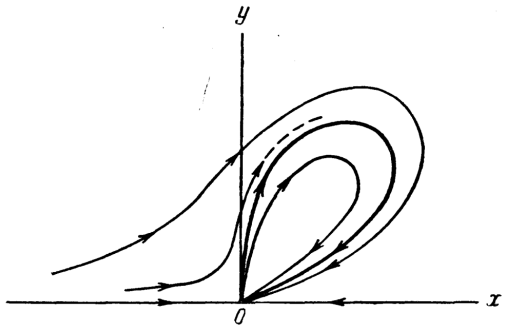

Figure 5.1: Trajectories of the Vinograd system. Copied from the original article of Vinograd.

In many cases, we want the convergence of the system trajectory towards $x = 0$ to be fast, thus bringing in the notion of exponential stability.

Definition 5.4 (Exponential Stability)**.** An equilibrium point $x = 0$ is said to be exponentially stable, if there exists a ball B_r such that as long as as $x(0) \in B_r$, then

$$\|x(t)\| \leq \alpha \|x(0)\| e^{-\lambda t}, \quad \forall t,$$

for some $\alpha > 0$ and $\lambda > 0$ (λ is called the rate of exponential convergence).

Exponential stability implies asymptotic stability (and certainly also Lyapunov stability). What is nice about exponential stability is that we can quantify the distance of the system trajectory to the equilibrium point as a function of time (as long as we know the constants $\alpha, \|x(0)\|, \lambda$). In many safety-critical applications, we need such performance guarantees. For example, in Chapter 6.1, we will see the application of exponential stability in observer-feedback control.

All the concepts of stability we have mentioned so far only talk about the stability of the system *locally* around the equilibrium point $x = 0$ (via arguments like B_r and B_R). It would be much nicer if we can guarantee stability of the system *globally*, i.e., no matter where the system starts in the state space $\mathbb{X}$, its trajectoy will converge to $x = 0$.

Definition 5.5 (Global Asymptotic and Exponential Stability)**.** The equilibrium point $x = 0$ is said to be globally asymptotically (exponentially) stable if asymptotic (exponential) stability holds for any initial states. That is,

$$\forall x \in \mathbb{X}, \quad x(0) = x \Longrightarrow \begin{cases} \lim_{t\to\infty} x(t) = 0 & \text{global asymptotic stability} \\ \exists \alpha, \lambda > 0, \text{ s.t. } \|x(t)\| \leq \alpha \|x(0)\| e^{-\lambda t} & \text{global exponential stability} \end{cases}$$

This concludes our definitions of stability for nonlinear systems (Definition 5.2-5.5). It is worth mentioning that the concepts of stability are complicated

(refined) here due to our focus on nonlinear systems. For linear systems, the concepts of stability are simpler. Specifically, all local stability properties of linear systems are also global and asymptotic stability is equal to exponential stability. In fact, for a linear time-invariant system $\dot{x} = Ax$, it is either asymptotically (exponentially) stable, or marginally stable, or unstable. Moreover, we can fully characterize the stability property by inspecting the eigenvalues of A (you can find a refreshment of this in Appendix C.1).

How do we characterize the stability property of a nonlinear system? If someone gave me a nonlinear system (5.3), how can I provide a certificate to her that the system is stable or unstable (I cannot use eigenvalues anymore in this case)? Let us describe some of these certificates below.

5.1.2 Stability by Linearization

A natural idea is to linearize, if possible, the nonlinear system (5.3) at a given equilibrium point $x^\star$ and inspect the stability of the linearized system (for which we can compute eigenvalues). Therefore, the key question here is how does the stability and instability of the linearized system relate to the stability and instability of the original nonlinear system.

Theorem 5.1 (Stability by Linearization)**.** *Assume $x = 0$ is an equilibrium point of system* (5.3) *and f is continuously differentiable. Let*

$$\dot{x} = Ax, \quad A = \frac{\partial f}{\partial x}\Big|_{x=0} \tag{5.6}$$

be the linearized system at $x = 0$. The following statements are true about the stability relationship between (5.3) *and* (5.6)*.*

- *If the linearized system* (5.6) *is strictly stable (i.e., all eigenvalues of A have strictly negative real parts), then the original system* (5.3) *is asymptotically stable at $x = 0$.*
- *If the linearized system* (5.6) *is unstable (i.e., at least one eigenvalue of A has strictly positive real part), then the original system* (5.3) *is unstable at $x = 0$.*
- *If the linearized system* (5.6) *is marginally stable (i.e., all eigenvalues of A have nonpositive real parts, and at least one eigenvalue has zero real part), then the stability of the original system* (5.3) *at $x = 0$ is indeterminate.*

Theorem 5.1 is actually quite useful when we want to quickly examine the local stability of a nonlinear system around a given equilibrium point, as we will show in the next example.

Example 5.2 (Stability of A Simple Pendulum by Linearization)**.** Consider the simple pendulum dynamics (5.5) in Example 5.1. Without loss of generality, let $m = 1, l = 1, b = 0.1$. The Jacobian of the nonlinear dynamics reads

$$A = \frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l}\cos\theta & -\frac{b}{ml^2} \end{bmatrix}.$$

At the bottomright equilibrium point $\theta = 0, \dot{\theta} = 0$, the matrix A has two eigenvalues

$$-0.0500 \pm 3.13i,$$

and hence the pendulum is asymptotically stable at the bottomright equilibrium point.

At the upright equilibrium point $\theta = \pi, \dot{\theta} = 0$, the matrix A has two eigenvalues

$$3.08, \quad -3.18,$$

and hence the pendulum is unstable at the upright equilibrium point.

The linearization method is easy to carry out. However, it tells us nothing about global stability or exponential stability. Moreover, when the linearized system is marginally stable, the stability of the orignal system is inconclusive. In the next, we will introduce a more general, and perhaps the most popular framework for analyzing the stability of nonliear systems.

5.1.3 Lyapunov Analysis

The basic idea of Lyapunov analysis is quite intuitive: if can find an "energy-like" scalar function for a system such that the scalar function is zero at an equilibrium point and positive everywhere else, and the time-derivative of the scalar function is zero at the equilibrium point but negative otherwise, then we know that the energy of the system will eventually converge to zero, and hence the state trajectory will converge to the equilibrium point. Lyapunov analysis was originally inspired by the energy function of a mechanical system: the total energy of a mechanical system (potental energy plus kinetic energy) will settle down to its minimum value if it is constantly dissipated (e.g., due to damping). However, the concept of a Lyapunov function is much broader than the energy function, i.e., it can be an arbitrary abstract function without any physical meaning.

Let us now introduce the concept of a Lyapunov function.

Definition 5.6 (Positive Definite Function)**.** A scalar function $V(x)$ is said to be locally positive definite in a ball B_R if

$$V(0) = 0 \quad \text{and} \quad V(x) > 0, \forall x \in B_R \backslash \{0\},$$

and globally positive definite if

$$V(0) = 0 \quad \text{and} \quad V(x) > 0, \forall x \in \mathbb{X}\backslash\{0\},$$

where $\mathbb{X}$ is the entire state space.

A function $V(x)$ is said to be negative definite if $-V(x)$ is positive definite.

A function $V(x)$ is said to be positive semidefinite if the "$>$" sign is replaced by the "$\geq$" sign in the above equations.

A function $V(x)$ is said to be negative semidefinite if $-V(x)$ is positive semidefinite.

For example, when $\mathbb{X} = \mathbb{R}^2$, the function $V(x) = x_1^2 + x_2^2$ is positive definite, but the function $V(x) = x_1^2$ is only positive semidefinite.

Definition 5.7 (Lyapunov Function)**.** In the ball B_R, if a function $V(x)$ is positive definite, and its time derivative along any system trajectory

$$\dot{V}(x) = \frac{\partial V}{\partial x} f(x)$$

is negative semidefinite (we assume the partial derivative $\frac{\partial f}{\partial x}$ exists and is continuous), then $V(x)$ is said to be a Lyapunov function for system (5.3). Note that $\dot{V}(x^\star) = 0$ at any equilibrium point $x^\star$ by definition.

With the introduction of positive definite and Lyapunov functions, we are now ready to use them to certify different concepts of stability.

Theorem 5.2 (Lyapunov Local Stability)**.** *Consider the nonlinear system* (5.3) *in a ball B_R with equilibrium point $x = 0$, if there exists a scalar function $V(x)$ (with continuous partial derivatives) such that*

- *$V(x)$ is positive definite (in B_R)*
- *$\dot{V}(x)$ is negative semidefinite (in B_R)*

then the equilibrium point $x = 0$ is stable in the sense of Lyapunov (cf. Definition 5.2).

Moreover,

- *if $\dot{V}(x)$ is negative definite in B_R, then the equilibrium point is asymptotically stable (cf. Definition 5.3).*
- *if $\dot{V}(x) \leq -\alpha V(x)$ for any $x \in B_R$, then the equilibrium point is exponentially stable (cf. Definition 5.4).*

Let us apply Theorem 5.2 to the simple pendulum.

Example 5.3 (Lyapunov Local Stability for A Simple Pendulum)**.** Consider the pendulum dynamics (5.5). The total energy of a pendulum is

$$V(x) = \frac{1}{2}ml^2\dot{\theta}^2 + mgl(1-\cos\theta). \tag{5.7}$$

Clearly, $V(x)$ is positive definite on the entire state space, and the only point where $V(x) = 0$ is the equilibrium point $\theta = 0, \dot{\theta} = 0$.

Let us compute the time derivative of $V(x)$:

$$\dot{V}(x) = ml^2\dot{\theta}\ddot{\theta} + mgl\sin\theta\dot{\theta} = ml^2\dot{\theta}\left(-\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta)\right) + mgl\sin\theta\dot{\theta} = -b\dot{\theta}^2,$$

which is clearly negative semidefinite. In fact, $\dot{V}(x)$ is precisely the energy dissipation rate due to damping. By Theorem 5.2 we conclude that the equilibrium point is stable in the sense of Lyapunov.

Note that with this choice of $V(x)$ as in (5.7), we actually cannot certify asymptotic local stability of the bottomright equilibrium point. So a natural question is, can we find a better Lyapunov function that indeed certifies asymptotic stability?

The answer is yes. Consider a different Lyapunov function

$$\tilde{V}(x) = \frac{1}{2}ml^2\dot{\theta}^2 + \frac{1}{2}ml^2\left(\frac{b}{ml^2}\theta + \dot{\theta}\right)^2 + 2mgl(1-\cos\theta), \tag{5.8}$$

which is positive definite and admits a single zero-value point $\theta = 0, \dot{\theta} = 0$ that is also the bottomright equilibrium point. Simplifying $\tilde{V}(x)$ we can get

$$\tilde{V}(x) = ml^2\dot{\theta}^2 + 2mgl(1-\cos\theta) + \frac{1}{2}ml^2\left(\frac{b^2}{m^2l^4}\theta^2 + \frac{2b}{ml^2}\theta\dot{\theta}\right) \tag{5.9}$$

$$= 2V(x) + \frac{1}{2}ml^2\left(\frac{b^2}{m^2l^4}\theta^2 + \frac{2b}{ml^2}\theta\dot{\theta}\right). \tag{5.10}$$

The time derivative of the new function $\tilde{V}(x)$ is

$$\dot{\tilde{V}}(x) = 2\dot{V}(x) + \frac{ml^2}{2}\left(\frac{2b^2}{m^2l^4}\theta\dot{\theta} + \frac{2b}{ml^2}(\dot{\theta}^2 + \theta\ddot{\theta})\right) \tag{5.11}$$

$$= 2\dot{V}(x) + b\dot{\theta}^2 + \left(\frac{b^2}{ml^2}\theta\dot{\theta} + b\theta\left(-\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta)\right)\right) \tag{5.12}$$

$$= -b\left(\dot{\theta}^2 + \frac{g}{l}\theta\sin\theta\right). \tag{5.13}$$

$\dot{\tilde{V}}(x)$ is negative definite locally around the equilibrium point (locally $\sin\theta \approx \theta$). Therefore, with the new Lyapunov function $\tilde{V}(x)$ we can certify asymptotic stability.

Interestingly, $V(x)$ is intuitive (the total energy of the pendulum system), but it fails to certify asymptotic local stability (as least by just using Theorem 5.2). $\tilde{V}(x)$ does not have any physical intuition, but it successfully certifies local asymptotic stability.

In Section 5.1.4, we will see that when using $V(x)$ with the invariant set theorem, we can actually still certify the asymptotic stability of the pendulum around the bottomright equilibrium.

In many applications, we desire to certify the global stability of an equilibrium point. The following theorem states that if in addition the scalar function $V(x)$ is *radially unbounded*, then global stability can be certified.

Theorem 5.3 (Lyapunov Global Stability)**.** *For the autonomous system* (5.3)*, suppose there exists a scalar function $V(x)$ with (continuous partial derivatives) such that*

- *$V(x)$ is positive definite;*
- *$\dot{V}(x)$ is negative definite;*
- *$V(x) \to \infty$ as $\|x\| \to \infty$,*

then the equilibrium point $x = 0$ is globally asymptotically stable (cf. Definition 5.5).

Moreover, if in addition to the three conditions above

- *$\dot{V}(x) \leq -\alpha V(x)$ for some $\alpha > 0$, then the equilibrium point is globally exponentially stable.*

5.1.4 Invariant Set Theorem

Through Theorem 5.2, Theorem 5.3, and Example 5.3, we see that in order to certify asymptotic stability, the time derivative $\dot{V}(x)$ is required to be positive definite. However, in many cases, with Example 5.3 being a typical one, $\dot{V}(x)$ is only negative semidefinite, which makes it difficult to certify asymptotic stability.

In this section, we will introduce the invariant set theorem that can help us reason about asymptotic stability even when $\dot{V}(x)$ is only negative semidefinite.

Let us first introduce the notion of an invariant set.

Definition 5.8 (Invariant Set)**.** A set G is an invariant set for a dynamical system (5.3) if every system trajectory that starts within G remains in G for all future time. Formally,

$$x(0) \in G \Longrightarrow x(t) \in G, \forall t.$$

A trivial invariant set is the entire state space $\mathbb{X}$. Another example of an invariant set is the singleton $\{x^\star\}$ with $x^\star$ being an equilibrium point. A nontrivial invariant set is the domain of attraction of an equilibrium point (cf. Definition 5.3).

We now state the local invariant set theorem.

Theorem 5.4 (Local Invariant Set)**.** *Consider the autonomous system* (5.3)*, and let $V(x)$ be a scalar function with continuous partial derivatives. Assume that*

- *the sublevel set $\Omega_\rho = \{x \in \mathbb{X} \mid V(x) < \rho\}$ is bounded for some $\rho > 0$, and*
- *$\dot{V}(x) \leq 0$ for all $x \in \Omega_\rho$.*

Let $\mathcal{R}$ be the set of all points within Ω_ρ such that $\dot{V}(x) = 0$, and $\mathcal{M}$ be the largest invariant set in $\mathcal{R}$. Then, every trajectory that starts in Ω_ρ will converge to $\mathcal{M}$ as $t \to \infty$.

With this theorem, we can now revisit the pendulum example 5.3.

Example 5.4 (Revisiting the Local Stability of A Simple Pendulum)**.** In Example 5.3, using the Lyapunov function

$$V(x) = \frac{1}{2}ml^2\dot{\theta}^2 + mgl(1 - \cos\theta),$$

with time derivative

$$\dot{V}(x) = -b\dot{\theta}^2,$$

we were only able to verify the stability of the bottomright equilibrium point in the sense of Lyapunov.

Now let us use the invariant set theorem 5.4 to show the asymptotic stability of the bottomright equilibrium point.

First it is easy to see that the sublevel set of $V(x)$ is bounded. For example, with $\rho = \frac{1}{4}mgl$,

$$V(x) < \frac{1}{4}mgl \Rightarrow \frac{1}{2}ml^2\dot{\theta}^2 < \frac{1}{4}mgl \Rightarrow \dot{\theta}^2 < \frac{1}{2}\frac{g}{l} \tag{5.14}$$

$$V(x) < \frac{1}{4}mgl \Rightarrow mgl(1 - \cos\theta) < \frac{1}{4}mgl \Rightarrow \cos\theta > \frac{3}{4} \Rightarrow \theta \in (-\arccos\frac{3}{4}, \arccos\frac{3}{4}). \tag{5.15}$$

The set $\mathcal{R}$, including all the points in Ω_ρ such that $\dot{V}(x) = 0$ is

$$\mathcal{R} = \{x \in \Omega_\rho \mid \dot{\theta} = 0\}.$$

We now claim that the largest invariant set $\mathcal{M}$ in $\mathcal{R}$ is just the single equilibrium point $x = [0, 0]^T$. We can prove this by contradiction. Suppose there is a different point $x' = [\theta, 0]^T$ with $\theta \neq 0$ also belonging to the invariant set $\mathcal{M}$, then

$$\ddot{\theta} = -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta) = -\frac{g}{l}\sin\theta \neq 0,$$

which means $\dot{\theta}$ will immediately become nonzero, and hence the trajectory will exit $\mathcal{R}$ and also $\mathcal{M}$. So that point cannot belong to the invariant set.

Now by Theorem 5.4, we conclude the bottomright equilibrium point is asymptotically stable.

Note that through this analysis we also obtain Ω_ρ as a domain of attraction for the bottomright equilibrium point.

Similarly, with the addition of the radial unboundedness of $V(x)$, we have a global version of the invariant set theorem.

Theorem 5.5 (Global Invariant Set)**.** *For the autonomous system* (5.3)*, let* $V(x)$ *be a scalar function with continuous partial derivatives that satisfies*

- $V(x) \to \infty$ *as* $\|x\| \to \infty$*, and*
- $\dot{V}(x) \leq 0$ *over the entire state space.*

Let $\mathcal{R} = \{x \in \mathbb{X} \mid \dot{V}(x) = 0\}$*, and* $\mathcal{M}$ *be the largest invariant set in* $\mathcal{R}$*. Then all system trajectories asymptotically converge to* $\mathcal{M}$ *as* $t \to \infty$*.*

5.1.5 Computing Lyapunov Certificates

All the Theorems we have stated so far (Theorems 5.2, 5.3, 5.4, and 5.5) are very general and powerful tools for certifying stability of nonlinear systems. However, the key requirement for applying the results is a Lyapunov function $V(x)$ that verifies different types of nonnegativity constraints.

How to find these functions?

In Example 5.3, we have seen that physical intuition can help us find a good Lyapunov function (5.7). Nevertheless, it did not quite give us what we want in terms of asymptotic stability. Instead, a hand-crafted function (5.8) helped us certify local asymptotic stability.

Wouldn't it be cool that we can design an algorithm to find the Lyapunov certificates for us?

A closer look at the Theorems 5.2, 5.3, 5.4, and 5.5 tells us the key property of a Lyapunov certificate is that it needs to satisfy the positivity (or negativity)

constraint for all states inside a set. This is a nontrivial and difficult requirement, because even if we were given a function $V(x)$, naively evaluting if $V(x)$ is nonnegative inside a set requires enumeration over all the states in the set, which is impractical given that the set is continuous and has infinite number of states.[1] When the dynamics (5.3) is linear, searching for Lyapunov functions is well understood and presented in Appendix C.1. However, when the dynamics is nonlinear, things can get very complicated.

In the next, I want to introduce a general framework for searching Lyapunov certificates for nonlinear systems that is based on convex optimization.

This framework, although having deep connections with many other disciplines such as algebraic geometry, theoretical computer science, and mathematical optimization, is based on a very simple intution that we all have since high school.

Example 5.5 (A Simple Example for Certifying Nonnegativity)**.** Suppose I give you a polynomial of a single variable $x \in \mathbb{R}$

$$p(x) = x^2 + 2x + 1$$

and ask you if $p(x) \geq 0$ for all x. You would not hesitate to answer "yes", because you know

$$p(x) = (x+1)^2$$

is the square of $x+1$ and hence must be nonnegative.

Let me make it more challenging. Suppose I give you a different polynomial

$$p(x) = -x^4 + 2x^2 + x + 1$$

and ask you if $p(x)$ is nonnegative for any $x \in [-1, 1]$ (instead of any $x \in \mathbb{R}$). At first glance, it seems much harder to answer this question because (i) we have a constraint set $x \in [-1, 1]$, and (ii) the polynomial $p(x)$ has a higher degree and it is not a polynomial that we are very famliar with (compared to $p(x) = x^2 + 2x + 1$).

However, if I show you that $p(x)$ can be written as

$$p(x) = -x^4 + 2x^2 + 2x + 1 = (x+1)^2 + x^2(1 - x^2), \tag{5.16}$$

it becomes easy again to certify that $p(x)$ is nonnegative for any $x \in [-1, 1]$. Why?

1. First notice that $(x+1)^2 \geq 0$ for any $x \in \mathbb{R}$,
2. Then notice that $1 - x^2 \geq 0$ for any $x \in [-1, 1]$, and $x^2 \geq 0$ for any x. Therefore, $x^2(1 - x^2) \geq 0$ for any $x \in [-1, 1]$.

[1] In fact, many of the recent works verify "neural" Lyapunov certificates (and other types of certificates) using this idea, see for example (Dawson et al., 2023).

Combining the above two reasonings, it becomes clear $p(x)$ is nonnegative for any $x \in [-1, 1]$.

What we have learned from this simple example is that

> Given a polynomial $p(x)$ and a constraint set $x \in \mathcal{X} \subseteq \mathbb{R}^n$, if we can write $p(x)$ as a sum of a finite number of products
>
> $$p(x) = \sum_{i=1}^{K} \sigma_i(x) g_i(x)$$
>
> where $\sigma_i(x)$ is a polynomial that we know is always nonnegative for any $x \in \mathbb{R}^n$ (just like $(x+1)^2$ and x^2 in (5.16)), and $g_i(x)$ is a polynomial that we know is always nonnegative for any x in the constraint set $\mathcal{X}$ (just like $1 - x^2$ for the set $[-1, 1]$ in (5.16)), then we have a certificate that $p(x) \geq 0$ for any $x \in \mathcal{X}$.

With this simple intuition, let me now formalize the framework of sum of squares (SOS) certificates for proving nonnegativity (also known as *positivstellensatz*, or in short P-satz).

Positivstellensatz, Sum of Squares, and Convex Optimization

Basic Semialgebraic Set. Let $x = [x_1, \dots, x_n] \in \mathbb{R}^n$ be a list of variables, we define a *basic semialgebraic set* as

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid p_i(x) = 0, i = 1, \dots, l_{\text{eq}}; p_i(x) \geq 0, i = l_{\text{eq}} + 1, \dots, l_{\text{eq}} + l_{\text{ineq}}\} \tag{5.17}$$

where $p_i(x), i = 1, \dots, l_{\text{eq}} + l_{\text{ineq}}$ are polynomial functions in x. In other words, the set $\mathcal{X}$ is a subset of $\mathbb{R}^n$ that is defined by l_{eq} equality constraints and l_{ineq} inequality constraints.

Observe that a basic semialgebraic set can capture a lot of the common constraint sets, such as a unit sphere, a unit ball, and a box (try this for yourself).

Positivstellensatz. We are now given the same question as in Example 5.5. Suppose I give you another polynomial function $p_0(x)$, how can you tell me if $p_0(x)$ is nonnegative for any x in the basic semialgebraic set $\mathcal{X}$? That is, to verify if

$$p_0(x) \geq 0, \quad \forall x \in \mathcal{X}.$$

Formalizing the intution obtained from Example 5.5, you will say if someone can produce a decomposition of $p_0(x)$ as

$$p_0(x) = \sigma_0(x) + \sum_{i=1}^{l_{\text{ineq}}} \sigma_i(x) p_{i+l_{\text{eq}}}(x) + \sum_{i=1}^{l_{\text{eq}}} \lambda_i(x) p_i(x), \tag{5.18}$$

where $\sigma_0, \sigma_1, \dots, \sigma_{l_{\text{ineq}}}$ are "some type of" polynomials that we know are always nonnegative (for any $x \in \mathbb{R}^n$), and $\lambda_1, \dots, \lambda_{l_{\text{eq}}}$ are arbitrary polynomials. Then I have a "certificate" that $p_0(x) \geq 0$ for any $x \in \mathcal{X}$.

Why? The reasoning is exactly the same as before.

1. $\sigma_0(x) \geq 0$ for any x,
2. $\sigma_i(x) p_{i+l_{\text{eq}}}(x) \geq 0, i = 1, \dots, l_{\text{ineq}}$ for any $x \in \mathcal{X}$, because (a) $\sigma_i(x) \geq 0$ for any x, and (b) $p_{i+l_{\text{eq}}}(x) \geq 0$ for any $x \in \mathcal{X}$ by definition of the basic semialgebraic set (5.17),
3. $\lambda_i(x) p_i(x) = 0, i = 1, \dots, l_{\text{eq}}$ for any $x \in \mathcal{X}$ by definition of the basic semialgebraic set (5.17).

We call σ_i's "nonnegative polynomial multipliers", and λ_i's "polynomial multipliers".

Sum-of-Squares. Now it comes the key question: what type of polynomials should we choose as the nonnegative polynomial multipliers? Ideally, this type of polynomials should

a. be always (trivially) nonnegative, and

b. have a nice representation for its unknown parameters (coefficients).

Looking back at our choice of multipliers, i.e., $(x+1)^2$ and x^2 in Example 5.5, it is natural to come up with the choice of a "sum-of-squares" (SOS) polynomial.

Definition 5.9 (Sum-of-Squares Polynomial)**.** A polynomial $\sigma(x)$ is called an SOS polynomial if

$$\sigma(x) = \sum_{i=1}^{k} q_i^2(x),$$

i.e., $\sigma(x)$ can be written as a sum of k squared polynomials.

OK, an SOS polynomial is trivially nonnegative (satisfying requirement (a) above), but does it have a nice representation for its parameters? The following Lemma gives us an affirmative answer.

Lemma 5.1 (SOS Polynomial and Positive Semidefinite Matrix)**.** *A polynomial $\sigma(x)$ is SOS if and only if*

$$\sigma(x) = [x]_d^T Q [x]_d$$

for some $Q \succeq 0$, where $[x]_d$ is the vector of monomials in x of degree up to d. For example, if $x \in \mathbb{R}^2$ and $d = 2$, then

$$[x]_2 = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]^T.$$

With the choice of $\sigma(x)$ as SOS polynomials, we are now ready to explicitly search for a nonnegativity certificate in the form of (5.18):

$$\begin{aligned} \text{find} \quad & \{\sigma_i\}_{i=0}^{l_{\text{ineq}}}, \{\lambda_i\}_{i=1}^{l_{\text{eq}}} \\ \text{subject to} \quad & p_0(x) = \sigma_0(x) + \sum_{i=1}^{l_{\text{ineq}}} \sigma_i(x) p_{i+l_{\text{eq}}}(x) + \sum_{i=1}^{l_{\text{eq}}} \lambda_i(x) p_i(x), \\ & \sigma_i \text{ is SOS}, i = 0, \dots, l_{\text{ineq}}, \\ & \lambda_i \text{ is polynomial}, i = 1, \dots, l_{\text{eq}} \\ & \deg(\sigma_0) \le 2\kappa, \deg(\sigma_i p_{i+l_{\text{eq}}}) \le 2\kappa, i = 1, \dots, l_{\text{ineq}}, \\ & \deg(\lambda_i p_i) \le 2\kappa, i = 1, \dots, l_{\text{eq}}. \end{aligned} \tag{5.19}$$

Bounding the Degree. The careful reader realizes that in (5.19) we have added constraints on the degrees of the polynomial multipliers σ_i's and λ_i's.[2] Precisely, we choose an integer κ, which we call the relaxation order, such that

$$2\kappa \ge \max\{\deg(p_i(x))\}_{i=0}^{l_{\text{eq}}+l_{\text{ineq}}},$$

and restrict the products $\sigma_i p_{i+l_{\text{eq}}}$'s and $\lambda_i p_i$'s to have degrees at most 2κ. With this, we are explicitly limiting the degrees of the multipliers σ_i's and λ_i's, and hence asking the formulation (5.19) to search for a finite number of parameters (otherwise, if the degree of the multipliers is unbounded, then the number of parameters to be searched is infinite).

Convex Optimization. The last crucial (and surprising) observation is that the problem (5.19) is a convex optimization! This is due to the following three reasons

a. The polynomial multipliers λ_i's can be fully parametrized by their coefficients, and these coefficients can be arbitrary vectors. Precisely, if $\lambda(x)$ is a polynomial with degree up to d, then

$$\lambda(x) = c^T [x]_d,$$

where $[x]_d$ is the vector of monomials in x of degree up to d, and c is the vector of coefficients.

b. The SOS multipliers σ_i's can be fully parametrized by their coefficients, and these coefficients are positive semidefinite matrices, according to Lemma 5.1.

[2]The degree of a monomial is the sum of its exponents. For example, $\deg(x_1 x_2^4 x_3^2) = 1 + 4 + 2 = 7$. The degree of a polynomial is the maximum degree of its monomials. For example, the polynomial $p(x) = 1 + x_2 + x_1^2 x_2^3$ has three monomials with degrees 0, 1, and 5, respectively. Therefore, $\deg(p) = 5$.

c. The equality constraint of decomposing $p_0(x)$ as a sum of products in (5.19) therefore becomes a set of affine equality constraints on the parameters of λ_i's and σ_i's, by matching coefficients of the monomials on the left-hand size and the right-hand side.

Therefore, the problem (5.19) is a convex semidefinite program (SDP). There are multiple software packages, e.g., SOSTOOLS, YALMIP, SumOfSquares.py, that allow us to model our problem in the form of (5.19), convert the formulation into SDPs, and pass them to SDP solvers (such as MOSEK). We will see an example of this soon.

Extensions. I want to congratulate, and welcome you to enter the world of SOS relaxations! Like I said before, this is an active area of research and the framework I just introduced is just a tip of the iceberg. Therefore, before I end this tutorial, I want to point out several extensions of the SOS framework.

- **Necessary Condition**. We have seen that a decomposition in the form of (5.19) is a *sufficient* condition to prove the nonnegativity of $p_0(x)$. Is it also a *necessary* condition? That is, for any $p_0(x)$ that is nonnegative on the set $\mathcal{X}$, does it admit a decomposition in the form of (5.19)? In general, the answer is no, and there exist nonnegative polynomials that cannot be written in the form of SOS decompositions (e.g., the Motzkin's polynomial). However, with certain assumptions on the set $\mathcal{X}$, the decomposition (5.19) is also necessary for nonnegativity! A well-known assumption is called the Archimedean condition (which, roughly speaking, requires the set $\mathcal{X}$ to be compact)). I suggest you to read (Blekherman et al., 2012) for more details.

- **Global Polynomial Optimization**. The SOS framework can be used for global optimization of polynomials in a straightforward way. Consider the polynomial optimization problem (POP)

$$\min_{x\in\mathcal{X}} p_0(x),$$

where one seeks the global minimum of the polynomial $p_0(x)$ on the set $\mathcal{X}$. A POP is generally a nonconvex optimization problem, and it is difficult to obtain a globally optimal solution. However, with a slight change of perspective, we can write the problem above equivalently as

$$\begin{aligned} \max \quad & \gamma \\ \text{subject to} \quad & p_0(x) - \gamma \geq 0, \quad \forall x \in \mathcal{X}. \end{aligned} \tag{5.20}$$

Basically I want to push the lower bound γ as high as possible. The constraint in (5.20) asks $p_0(x)-\gamma$ to be nonnegative on $\mathcal{X}$. With the SOS framework introduced above, we can naturally relax it to

$$\begin{aligned} \max \quad & \gamma \\ \text{subject to} \quad & p_0(x) - \gamma \quad \text{is SOS on} \quad \mathcal{X}, \end{aligned} \tag{5.21}$$

where the "SOS on $\mathcal{X}$" constraint is exactly the problem (5.19). Therefore, we have relaxed the nonconvex optimization (5.20) into a convex problem (5.21)! Moreover, by increasing the relaxation order κ, we obtain a sequence of lower bounds that asymptotically converge to the true global optimum of the nonconvex problem (5.20). This is called Lasserre's hierarchy of moment-SOS relaxations, originally proposed by Lasserre in the seminal work (Lasserre, 2001). As this name suggests, the dual problem to the SOS relaxation (5.21) is called the moment relaxation. Lasserre's hierarchy has recently gained a lot of attention due to the empirical observation in many engineering disciplines that the convergence to global optimum is finite, i.e., by solving the convex problem (5.21) at a finite relaxation order κ, an exact global optimizer of the original nonconvex problem (5.20) can be extracted. For a pragmatic introduction to the moment relaxation, I suggest to read Section 2.2 of (Yang and Carlone, 2022). For more applications of Lasserre's hierarchy, please refer to (Lasserre, 2009).

- **Scalability**. I have to warn you that there is no free lunch. The fact that so many challenging problems can be relaxed or restated as convex optimization problems should send you an alert. Does this mean that we can use convex optimization to solve all the challenging problems? Well, although we hope this is the case, in practice we are limited by the computational resources. The caveat is that the problem (5.19) and (5.21), despite being convex, grows very large as the dimension n and relaxation order κ increases. Another way of saying this is that, we seek to solve small-to-medium scale nonconvex problems with large-scale convex problems. Unfortunately, today's SDP solver cannot solve all the problems we formulate, and hence a major research direction in the mathematical optimization community is to develop SDP solvers that are more scalable. You can read (Yang et al., 2022) and references therein for more details.

- **Non-SOS Certificates**. Nobody is preventing us to use a different choice of nonnegative polynomial multipliers (other than SOS multipliers) in (5.18). For example, one can use a decomposition as the sum of nonnegative circuit polynomials (Wang, 2022) or signomials (Murray et al., 2021). However, to the best of my knowledge, non-SOS certificates are far less popular than SOS certificates.

There are many other extensions to the SOS framework, and a complete enumeration is beyond the scope of this lecture notes. For the connection between SOS and theoretical computer science, you can see the lecture notes by Boaz Barak and David Steurer. There are also more recent monographs about SOS, for example (Magron and Wang, 2023) and (Nie, 2023). I plan to introduce these in more details in an upcoming graduate-level class at Harvard.

That was a long detour from Lyapunov analysis! The SOS machinery will come back later when we study multiple other topics in optimal control and estima-

tion. But now let us show how to tackle the problem of computing Lyapunov certificates using the SOS machinery.

According to Theorem 5.2, given a set $\mathcal{X}$ that contains an equilibrium point $x^\star$, if we can find a Lyapunov function $V(x)$ such that $V(x)$ is positive definite on $\mathcal{X}$ and $\dot{V}(x)$ is negative definite on $\mathcal{X}$, then the equilibrium point $x^\star$ is locally asymptotically stable. With the SOS machinery, we can search for a $V(x)$ that is a polynomial as

$$\begin{aligned} \text{find} \quad & V(x) && (5.22)\\ \text{subject to} \quad & V(x) - \epsilon_1 \|x - x^\star\|^2 \quad \text{is SOS on} \quad \mathcal{X} && (5.23)\\ & -\epsilon_2 \|x - x^\star\|^2 - \frac{\partial V(x)}{\partial x} f(x) \quad \text{is SOS on} \quad \mathcal{X} && (5.24)\\ & V(x^\star) = 0, && (5.25) \end{aligned}$$

where $\epsilon_1, \epsilon_2 > 0$ are (small) positive constants. This is a convex optimization problem, just like (5.19) (try to convince yourself my claim is true). Similarly, we can choose a relaxation order κ and solve the above problem. If a solution exists, then we find a valid Lyapunov certificate.

Let us apply it to the simple pendulum to synthesize local stability certificates.

Example 5.6 (Computing Lyapunov Local Stability Certificate for the Simple Pendulum with Convex Optimization)**.** The SOS framework works with polynomials, so let us first write the pendulum dynamics in polynomial form via a change of coordinate $x = [\mathfrak{s}, \mathfrak{c}, \dot{\theta}]^T$ with $\mathfrak{s} = \sin\theta$, $\mathfrak{c} = \cos\theta$:

$$\begin{cases} \dot{\mathfrak{s}} = \mathfrak{c}\dot{\theta} \\ \dot{\mathfrak{c}} = -\mathfrak{s}\dot{\theta} \\ \ddot{\theta} = -\frac{1}{ml^2}(b\dot{\theta} + mgl\mathfrak{s}) \end{cases}.$$

We will use $m = 1, l = 1, b = 0.1$ for our numerical experiment.

We want to find a local Lyapunov certificate in the compact set

$$\theta \in \left[-\arccos\frac{3}{4}, \arccos\frac{3}{4}\right], \quad \dot{\theta} \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]. \tag{5.26}$$

In the new coordinates x, this is equivalent to the semialgebraic set

$$\mathcal{X} = \left\{ x \in \mathbb{R}^3 \mid \mathfrak{s}^2 + \mathfrak{c}^2 = 1, \dot{\theta}^2 \le \frac{\pi^2}{4}, \mathfrak{c} \ge \frac{3}{4} \right\}.$$

Denoting the bottomright equilibrium point as $x_e = [0, 1, 0]^T$, and with $\epsilon_1, \epsilon_2 > 0$ two positive constants, we can seek a Lyapunov function $V(x)$ that satisfies the following conditions

$$\begin{aligned} V(x) &\ge \epsilon_1 (x - x_e)^T (x - x_e), \quad \forall x \in \mathcal{X} && (5.27)\\ \dot{V}(x) = \frac{\partial V}{\partial x}\dot{x} &\le -\epsilon_2 (x - x_e)^T (x - x_e), \quad \forall x \in \mathcal{X} && (5.28)\\ V(x_e) &= 0, \quad \dot{V}(x_e) = 0 && (5.29) \end{aligned}$$

where (5.27) ensures $V(x)$ is positive definite, (5.28) ensures $\dot{V}(x)$ is negative definite, and (5.29) ensures $V(x), \dot{V}(x)$ vanish at the equilibrium point.

To leverage the power of convex optimization, we can relax the positivity constraints as SOS constraints

$$V(x) - \epsilon_1 (x - x_e)^T (x - x_e) \quad \text{is SOS on} \quad \mathcal{X} \tag{5.30}$$

$$-\epsilon_2 (x - x_e)^T (x - x_e) - \frac{\partial V}{\partial x} \dot{x} \quad \text{is SOS on} \quad \mathcal{X} \tag{5.31}$$

$$V(x_e) = 0, \quad \dot{V}(x_e) = 0. \tag{5.32}$$

If we limit the degree of V to 2, choose the relaxation order $\kappa = 2$, and $\epsilon_1 = \epsilon_2 = 0.01$, we obtain a solution

$$V(x) = 2.7982\mathfrak{s}^2 + 0.086248\mathfrak{s}\dot{\theta} + 2.4548\mathfrak{c}^2 + 0.88117\dot{\theta}^2 - 16.6277\mathfrak{c} + 14.1728$$

with the time derivative

$$\dot{V}(x) = 0.68675\mathfrak{s}\mathfrak{c}\dot{\theta} + 0.086248 * \mathfrak{c}\dot{\theta}^2 - 0.84523\mathfrak{s}^2 - 0.65191\mathfrak{s}\dot{\theta} - 0.17623\dot{\theta}^2.$$

Plotting $V(x)$ in the constraint set (5.26) using $(\theta, \dot{\theta})$ coordinates, we get

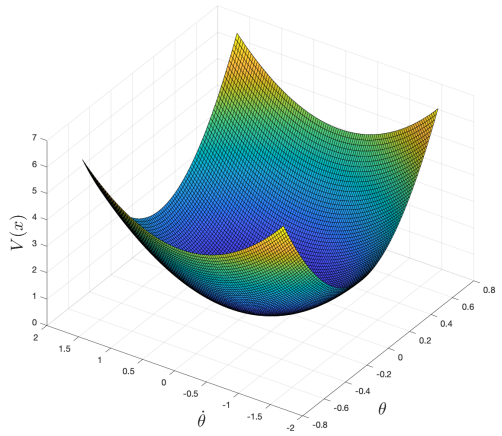

Figure 5.2: Lyapunov local stability certificate computed via convex optimization.

and verify that $V(x)$ is locally positive definite.

Plotting $\dot{V}(x)$ in the constraint set (5.26) using $(\theta, \dot{\theta})$ coordinates, we get

and verify that $\dot{V}(x)$ is locally negative definite.

You should try the code for this example here.

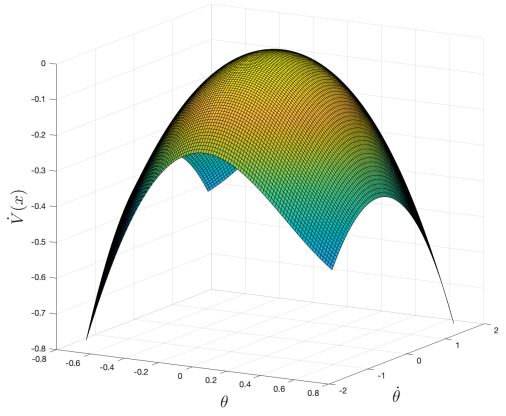

Figure 5.3: Derivative of the Lyapunov local stability certificate computed via convex optimization.

5.2 Controlled Systems

5.3 Non-autonomous Systems

Lemma 5.2 (Barbalat's Lemma)**.** *Let $f(t)$ be differentiable, if*

- $\lim_{t\to\infty} f(t)$ *is finite, and*
- $\dot{f}(t)$ *is uniformly continuous,*[3]

then

$$\lim_{t\to\infty} \dot{f}(t) = 0.$$

Theorem 5.6 (Barbalat's Stability Certificate)**.** *If a scalar function $V(x,t)$ satisfies*

- $V(x,t)$ *is lower bounded,*
- $\dot{V}(x,t)$ *is negative semidefinite*
- $\dot{V}(x,t)$ *is uniformly continuous*

then $\dot{V}(x,t) \to 0$ as $t \to \infty$.

[3]A sufficient condition for this to hold is that $\ddot{f}$ exists and is bounded.

Proof. $V(x,t)$ is lower bounded and $\dot{V}$ is negative semidefinite implies the limit of V as $t \to \infty$ is finite (note that $V(x,t) \leq V(x(0),0)$). Then the theorem clearly follows from Barbalat's Lemma 5.2. □

Chapter 6

Output Feedback

Consider a continuous-time dynamical system

$$\begin{aligned} \dot{x} &= f(x,u) \\ y &= h(x,u) \end{aligned} \tag{6.1}$$

where $x(t) \in \mathbb{X} \subseteq \mathbb{R}^n$ the state of the system, $u(t) \in \mathbb{U} \subseteq \mathbb{R}^m$ the control (or input), $y(t) \in \mathbb{Y} \subseteq \mathbb{R}^d$ the output (i.e., measurement) of the state and control, and f, g the evolution and measurement functions (which are sufficiently smooth).

6.1 State Observer

For the system (6.1), let us denote

- $X(x_0, t_0; t; u)$ the solution at time t with input u and initial condition x_0 at time t_0; when $t_0 = 0$, we write $X(x_0; t; u)$
- $Y(x_0, t_0; t; u)$ the output at time t with input u and initial condition x_0 at time t_0, i.e., $Y(x_0, t_0; t; u) = h(X(x_0, t_0; t; u), u(t))$; when $t_0 = 0$, we write $y_{x_0,u}(t)$;
- $\mathcal{X}_0$ a subset of $\mathbb{X}$ containing the initial conditions we consider; for any $x_0 \in \mathcal{X}_0$, we write $\sigma^+_{\mathcal{X}}(x_0; u)$ the maximal time of existence of $X(x_0, \cdot; t; u)$ in a set $\mathcal{X}$
- $\mathcal{U}$ the set of all sufficiently many times differentiable inputs $u : [0, +\infty) \to \mathbb{U}$.

The problem of state observation is to produce an estimated state $\hat{x}(t)$ of the true state $X(x_0, t_0; t; u)$ based on knowledge about the system (6.1) and information about the history of inputs $u_{[0,t]}$ and outputs $y_{[0,t]}$, so that $\hat{x}(t)$ asymptotically converges to $X(x_0, t_0; t; u)$, for any initial condition $x_0 \in \mathcal{X}_0$ and any input $u \in \mathcal{U}$.

There are multiple ways for solving the problem of state observation (see e.g., (Bernard, 2019), (Bernard et al., 2022)). Here we are particularly interested in the approach using a *state observer*, i.e., a dynamical system whose *internal state* evolves according to the history of inputs and outputs, from which a state estimation can be reconstructed that guarantees asymptotic convergence to the true state. We formalize this concept below.

Definition 6.1 (State Observer)**.** A state observer for system (6.1) is a couple $(\mathcal{F}, \mathcal{T})$ such that

1. $\mathcal{F} : \mathbb{R}^l \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^l$ is continuous

2. $\mathcal{T}$ is a family of continuous functions indexed by $u \in \mathcal{U}$ where each $\mathcal{T}_u : \mathbb{R}^l \times [0, +\infty) \to \mathbb{R}^n$ respects the causality condition

$$\forall \tilde{u} : [0, +\infty) \to \mathbb{R}^m, \forall t \in [0, +\infty), u_{[0,t]} = \tilde{u}_{[0,t]} \Rightarrow \mathcal{F}_u(\cdot, t) = \mathcal{F}_{\tilde{u}}(\cdot, t).$$

3. For any $u \in \mathcal{U}$, any $z_0 \in \mathbb{R}^l$, and any $x_0 \in \mathcal{X}_0$ such that $\sigma^+_{\mathbb{X}}(x_0; u) = +\infty$, any solution $Z(z_0; t; u, y_{x_0,u})$[1] to

$$\dot{z} = \mathcal{F}(z, u, y_{x_0,u}) \tag{6.2}$$

initialized at z_0 at time 0 with input u and $y_{x_0,u}$ exists on $[0, +\infty)$ and satisfies

$$\lim_{t \to \infty} \|\hat{X}(x_0, z_0; t; u) - X(x_0; t; u)\| = 0, \tag{6.3}$$

with

$$\hat{X}(x_0, z_0; t; u) = \mathcal{T}_u(Z(z_0; t; u, y_{x_0,u}), t). \tag{6.4}$$

In words, (i) the state observer maintains an internal state (or latent state) $z \in \mathbb{R}^l$ that evolves according to the latent dynamics $\mathcal{F}$ in (6.2), where u and $y_{x_0,u}$ are inputs; (ii) an estimated state can be reconstructed from the internal state using $\mathcal{T}_u$ as in (6.4); and (iii) the error between the estimated state and the true state (defined by a proper distance function $\|\cdot\|$ on $\mathbb{X}$) converges to zero.

[1] We say "any solution" because there may be several solutions to the observer (6.2) due to $\mathcal{F}$ only being continuous. This is not a problem as long as any such solution satisfies the required convergence property.

If $\mathcal{T}_u$ is the same for any $u \in \mathcal{U}$ and is also time independent, then we say $\mathcal{T}$ is *stationary*.[2] In this case, we can simply write the observer (6.2) and (6.4) as

$$\begin{aligned} \dot{z} &= \mathcal{F}(z, u, y) \\ \hat{x} &= \mathcal{T}(z). \end{aligned} \tag{6.5}$$

If $\hat{x}$ can be read off directly from z, then we say the observer (6.5) is *in the given coordinates*. A special case of this is when $\hat{x} = z$, i.e., the internal state of the observer is the same as the system state.

6.1.1 General Design Strategy

Theorem 6.1 (Meta Observer)**.** *Let $F : \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^p$, $H : \mathbb{R}^p \times \mathbb{R}^m \to \mathbb{R}^d$ and $\mathcal{F} : \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^p$ be continuous functions such that*

$$\dot{\hat{\xi}} = \mathcal{F}(\hat{\xi}, u, \tilde{y}) \tag{6.6}$$

is an observer for

$$\dot{\xi} = F(\xi, u, H(\xi, u)), \quad \tilde{y} = H(\xi, u), \tag{6.7}$$

i.e., for any $\xi_0, \hat{\xi}_0 \in \mathbb{R}^p$ and any $u \in \mathcal{U}$, the solution of the observer (6.6), denoted by $\hat{\Xi}(\hat{\xi}_0; t; u; \tilde{y}_{\xi_0,u})$, and the solution of the true system (6.7), denoted by $\Xi(\xi_0; t; u)$, satisfy

$$\lim_{t\to\infty} \|\hat{\Xi}(\hat{\xi}_0; t; u; \tilde{y}_{\xi_0,u}) - \Xi(\xi_0; t; u)\| = 0. \tag{6.8}$$

Note that the observer (6.6) is stationary and in the given coordinates for system (6.7). Indeed the internal state of the observer is the same as the system state.

Now suppose for any $u \in \mathcal{U}$, there exists a continuous function (i.e., coordinate transformation) $T_u : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^p$ and a subset $\mathcal{X}$ of $\mathbb{X}$ such that

1. *For any $x_0 \in \mathcal{X}_0$ such that $\sigma^+_{\mathbb{X}}(x_0; u) = +\infty$, $X(x_0; \cdot; u)$ remains in $\mathcal{X}$*
2. *There exists a concave $\mathcal{K}$[3] function ρ and a positive number $\bar{t}$ such that*

$$\|x_a - x_b\| \le \rho(|T_u(x_a, t) - T_u(x_b, t)|), \quad \forall x_a, x_b \in \mathcal{X}, t \ge \bar{t},$$

i.e., $x \mapsto T_u(x, t)$ becomes injective on $\mathcal{X}$,[4] uniformly in time and space, after a certain time $\bar{t}$.

[2] The time dependence of $\mathcal{T}_u$ enables us to cover the case where the knowledge of the u and $y_{x_0,u}$ is used to construct the estimate from the observer state. In particular, using the output sometimes can reduce the dimension of the observer state (and thus alleviate the computations), thus obtaining a reduced-order observer. For example, see (Karagiannis and Astolfi, 2005) and (Astolfi and Ortega, 2003).

[3] A function $\rho : \mathbb{R}_+ \to \mathbb{R}_+$ is a $\mathcal{K}$ function if $\rho(0) = 0$, ρ is continuous, and ρ is increasing.

[4] An injective function is a function f that maps distinct elements of its domain to distinct elements. That is, $f(x_a) = f(x_b)$ implies $x_a = x_b$, or equivalently, $x_a \neq x_b$ implies $f(x_a) \neq f(x_b)$.

3. T_u *transforms the system* (6.1) *into the system* (6.7)*, i.e., for all* $x \in \mathcal{X}$ *and all* $t \geq 0$*, we have*

$$L_{(f,1)}T_u(x,t) = F(T_u(x,t), u, h(x,u)), \quad h(x,u) = H(T_u(x,t), u), \quad (6.9)$$

where $L_{(f,1)}T_u(x,t)$ *is the Lie derivative of* T_u *along the vector field* $(f,1)$

$$L_{(f,1)}T_u(x,t) = \lim_{\tau \to 0} \frac{T_u(X(x,t;t+\tau;u),t+\tau) - T_u(x,t)}{\tau}.$$

4. T_u *respects the causality condition*

$$\forall \tilde{u} : [0,+\infty) \to \mathbb{R}^m, \forall t \in [0,+\infty), u_{[0,t]} = \tilde{u}_{[0,t]} \Rightarrow T_u(\cdot,t) = T_{\tilde{u}}(\cdot,t).$$

Then, for any $u \in \mathcal{U}$*, there exists a function* $\mathcal{T}_u : \mathbb{R}^p \times [0,+\infty) \to \mathcal{X}$ *(satisfying the causality condition) such that for any* $t \geq \bar{t}$*,* $\xi \mapsto \mathcal{T}_u(\xi,t)$ *is uniformly continuous on* $\mathbb{R}^p$ *and satisfies*

$$\mathcal{T}_u(T_u(x,t),t) = x, \forall x \in \mathcal{X}.$$

Moreover, denoting $\mathcal{T}$ *the family of functions* $\mathcal{T}_u$ *for* $u \in \mathcal{U}$*, the couple* $(\mathcal{F},\mathcal{T})$ *is an observer for the system* (6.1) *initialized in* $\mathcal{X}_0$*.*

Proof. See Theorem 1.1 in (Bernard, 2019). □

A simpler version of Theorem 6.1 where the coordinate transformation T_u is stationary and fixed for all u is stated below as a corollary.

Corollary 6.1 (Meta Observer with Fixed Transformation)**.** *Let* $F : \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^p$*,* $H : \mathbb{R}^p \times \mathbb{R}^m \to \mathbb{R}^d$ *and* $\mathcal{F} : \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^p$ *be continuous functions such that* (6.6) *is an observer for* (6.7)*.*

Suppose there exists a continuous coordinate transformation $T : \mathbb{R}^p \to \mathbb{R}^n$ *and a compact subset* Ω *of* $\mathbb{R}^n$ *such that*

1. *For any* $x_0 \in \mathcal{X}_0$ *such that* $\sigma^+_{\mathbb{X}}(x_0;u) = +\infty$*,* $X(x_0;\cdot;u)$ *remains in* Ω
2. $x \mapsto T(x)$ *is injective on* Ω
3. T *transforms the system* (6.1) *into system* (6.7)

$$L_f T(x) = F(T(x), u, h(x,u)), \quad h(x,u) = H(T(x), u),$$

where $L_f T(x)$ *is the Lie derivative of* $T(x)$ *along* f

$$L_f T(x) = \lim_{\tau \to 0} \frac{T(X(x,t;t+\tau;u)) - T(x)}{\tau}.$$

Then, there exists a uniformly continuous function $\mathcal{T} : \mathbb{R}^p \to \mathbb{R}^n$ *such that*

$$\mathcal{T}(T(x)) = x, \quad \forall x \in \Omega,$$

and $(\mathcal{F}, \mathcal{T})$ *is an observer for system* (6.1) *initialized in* $\mathcal{X}_0$.

Theorem 6.1 and Corollary 6.1 suggest the following general observer design strategy:

1. Find an injective coordinate transformation T_u (that may be time-varying and also dependent on u) that transforms the original system (6.1) with coordinate x into a new system (6.7) with coordinate ξ
2. Design an observer (6.6), $\hat{\xi}$, for the new system
3. Compute a left inverse, $\mathcal{T}_u$, of the transformation T_u to recover a state estimation $\hat{x}$ of the original system.

The transformed systems (6.7) are typically referred to as *normal forms*, or in my opinion, *templates*.

Of course, the general design strategy is rather conceptual, and in order for it to be practical, we have to answer three questions.

- What templates do we have, what are their associated observers, and what are the conditions for the observers to be asymptotically converging?
- What kinds of (nonlinear) systems can be transformed into the templates, and how to perform the transformation?
- How to invert the coordinate transformation? Is it analytical or does it require numerical approximation?

In the following sections, we will study several representative normal forms and answer the above questions. Before presenting the results, let us first introduce several notions of observability.

Definition 6.2 (Observability)**.** Consider an open subset $\mathcal{L}$ of the state space $\mathbb{X} \subseteq \mathbb{R}^n$ of system (6.1). The system (6.1) is said to be

- **Distinguishable** on $\mathcal{L}$ for some input $u(t)$, if for all $(x_a, x_b) \in \mathcal{L} \times \mathcal{L}$,

$$y_{x_a,u}(t) = y_{x_b,u}(t), \forall t \in [0, \min\{\sigma^+_{\mathbb{X}}(x_a; u), \sigma^+_{\mathbb{X}}(x_b; u)\}] \Longrightarrow x_a = x_b$$

- **Instantaneously distinguishable** on $\mathcal{L}$ for some input $u(t)$, if for all $(x_a, x_b) \in \mathcal{L} \times \mathcal{L}$, and for all $\bar{t} \in (0, \min\{\sigma^+_{\mathbb{X}}(x_a; u), \sigma^+_{\mathbb{X}}(x_b; u)\})$,

$$y_{x_a,u}(t) = y_{x_b,u}(t), \forall t \in [0, \bar{t}) \Longrightarrow x_a = x_b$$

- **Uniformly observable** on $\mathcal{L}$ if it is distinguishable on $\mathcal{L}$ for any input $u(t)$ (not only for $u \in \mathcal{U}$)
- **Uniformly instantaneously observable** on $\mathcal{L}$ if it is instantaneously observable on $\mathcal{L}$ for any input $u(t)$ (not only for $u \in \mathcal{U}$).

Moreover, let $\mathcal{X}$ be a subset of $\mathbb{X}$ such that $\mathrm{cl}(\mathcal{X})$, i.e., the closure of $\mathcal{X}$, is contained in $\mathcal{L}$. Then the system (6.1) is said to be

- **Backward $\mathcal{L}$-distinguishable on $\mathcal{X}$** for some input $u(t)$, if for any $(x_a, x_b) \in \mathcal{X} \times \mathcal{X}$ such that $x_a \neq x_b$, there exists $t \in (\max\{\sigma^-_{\mathcal{L}}(x_a; u), \sigma^-_{\mathcal{L}}(x_b; u)\}, 0]$ such that $y_{x_a,u}(t) \neq y_{x_b,u}(t)$, or in words similar to the definition of distinguishable on $\mathcal{L}$, for all $(x_a, x_b) \in \mathcal{X} \times \mathcal{X}$

$$y_{x_a,u}(t) = y_{x_b,u}(t), \forall t \in (\max\{\sigma^-_{\mathcal{L}}(x_a; u), \sigma^-_{\mathcal{L}}(x_b; u)\}, 0] \Longrightarrow x_a = x_b.$$

6.1.2 Luenberger Template

Consider an instance of the normal form (6.7) as follows:

$$\dot{\xi} = A\xi + B(u, y), \quad y = C\xi, \tag{6.10}$$

where A, C are constant matrices, and $B(u, y)$ can depend nonlinearly on u and y.

For this template, we have the well-known Luenberger observer.

Theorem 6.2 (Luenberger Observer)**.** *If the pair (A, C) is detectable (see Theorem C.9), then there exists a matrix K such that $A - KC$ is Hurwitz and the system*

$$\dot{\hat{\xi}} = A\hat{\xi} + B(u, y) + K(y - C\hat{\xi}) \tag{6.11}$$

is an observer for (6.10)*.*

Proof. Define $e(t) = \xi(t) - \hat{\xi}(t)$. In that case,

$$\dot{e}(t) = [A - KC]e(t) \tag{6.12}$$

Solving (6.12), we obtain

$$e(t) = \exp[(A - KC)t]e(0) \tag{6.13}$$

Then, if the real components of the eigenvalues of $A - KC$ are strictly negative (i.e., $A - KC$ is Hurwitz), then $e(t) \to 0$ as $t \to \infty$, independent of the initial error $e(0) = \xi(0) - \hat{\xi}(0)$. From Theorem C.9, we know that (A, C) being detectable implies the existence of K such that $A - KC$ is Hurwitz.

If one is further interested in estimating the convergence rate of the Luenberger observer, then one can use the result from Corollary C.1. Particularly, one can solve the Lyapunov equation

$$(A-KC)^T P + P(A-KC) = -I$$

to obtain P. Then the convergence rate of $\|e\|$ towards zero is $\frac{0.5}{\lambda_{\max}(P)}$. □

The Luenberger observer is an elegant result in observer design (and even in control theory) that has far-reaching impact. In my opinion, the essence of observer design is twofold: (i) to simulate the dynamics when the state estimation is correct, and (ii) to correct the state estimation from observation when it is off. These two pieces of ideas are evident in (6.11): the observer is a copy of the original dynamics ($A\hat{\xi} + B(u,y)$) plus a feedback correction from the difference between the "imagined" observation $C\hat{\xi}$ and the true observation y.

You may think the Luenberger template is restricting because it requires the system to be linear (up to the only nonlinearly in $B(u,y)$). However, it turns out the Luenberger template is already quite useful, as I will show in the following pendulum example.

Example 6.1 (Luenberger Observer for A Simple Pendulum)**.** Consider a simple pendulum dynamics model

$$x = \begin{bmatrix}\theta \\ \dot{\theta}\end{bmatrix}, \quad \dot{x} = \begin{bmatrix}\dot{\theta} \\ -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta)\end{bmatrix} + \begin{bmatrix}0 \\ \frac{1}{ml^2}\end{bmatrix} u, \quad y = \theta, \tag{6.14}$$

where θ the angular position of the pendulum from the vertical line, $m > 0$ the mass of the pendulum, $l > 0$ the length, g the gravitational constant, $b > 0$ the dampling coefficient, and u the control input (torque).

We assume we can only observe the angular position of the pendulum in (6.14), e.g., using a camera, but not the angular velocity. Our goal is to construct an observer that can provide a full state estimation.

We first note that the pendulum dynamics (6.14) can actually be written in the (linear) Luenberger template (6.10) as[5]

$$\begin{aligned}\dot{x} &= \underbrace{\begin{bmatrix}0 & 1 \\ 0 & -\frac{b}{ml^2}\end{bmatrix}}_{=:A} x + \underbrace{\begin{bmatrix}0 \\ \frac{u - mgl\sin\theta}{ml^2}\end{bmatrix}}_{=:B(u,y)}. \\ y &= \underbrace{\begin{bmatrix}1 & 0\end{bmatrix}}_{=:C} x\end{aligned} \tag{6.15}$$

In order for us to use the Luenberger observer, we need to check if the pair (A, C) is detectable. We can easily find the eigenvalues and eigenvectors of A:

$$A\begin{bmatrix}1 \\ 0\end{bmatrix} = 0, \quad A\begin{bmatrix}-\frac{ml^2}{b} \\ 1\end{bmatrix} = \begin{bmatrix}1 \\ -\frac{b}{ml^2}\end{bmatrix} = -\frac{b}{ml^2}\begin{bmatrix}-\frac{ml^2}{b} \\ 1\end{bmatrix}.$$

[5] I have to say I was a bit surprised when I arrived at this formulation.

The first eigenvalue has real part equal to 0. However,

$$C\begin{bmatrix}1\\0\end{bmatrix} = 1 \neq 0.$$

According to Theorem C.9, we conclude (A, C) is detectable. In fact, the pair (A, C) is more than just detectable, it is indeed observable (according to Theorem C.7). Therefore, the poles of $A - KC$ can be arbitrarily placed.

Finding K. Now we need to find K. An easy choice of K is

$$K = \begin{bmatrix}k\\0\end{bmatrix}, \quad A - KC = \begin{bmatrix}-k & 1\\ 0 & -\frac{b}{ml^2}\end{bmatrix}.$$

With $k > 0$, we know $A - KC$ is guaranteed to be Hurwitz (the two eigenvalues of $A - KC$ are $-k$ and $-b/ml^2$), and we have obtained an observer!

We can also estimate the convergence rate of this observer. Let us use $m = 1, g = 9.8, l = 1, b = 0.1$ as parameters of the pendulm dynamics. According to Theorem 6.2, we solve the Lyapunov equation

$$(A - KC)^T P + P(A - KC) = -I$$

and $\gamma = \frac{0.5}{\lambda_{\max}(P)}$ will be our best estimate of the convergence rate (of $\|e\| = \|\hat{x} - x\|$ towards zero).

Table 6.1 below shows the convergence rates computed for different values of k. We can see that as k is increased, the convergence rate estimation is also increased. However, it appears that 0.1 is the best convergence rate we can achieve, regardless of how large k is.

Table 6.1: Convergence rate estimation of the Luenberger observer for a simple pendulm.

k	0.1	1	10	100	1000	10000
γ	0.0019	0.0523	0.0990	0.1000	0.1000	0.1000

Optimal K. Is it true that 0.1 is the best convergence rate, or in other words, what is the best K that maximizes the convergence rate γ?

A natural way (and my favorite way) to answer this question is to formulate an optimization problem.

$$\begin{aligned} \min_{P,K} \quad & \lambda_{\max}(P) \\ \text{subject to} \quad & (A - KC)^T P + P(A - KC) = -I \\ & P \succeq 0 \end{aligned} \tag{6.16}$$

The above formulation seeks the best possible K that minimizes $\lambda_{\max}(P)$ which, according to $\gamma = 0.5/\lambda_{\max}(P)$, also maximizes γ.

However, problem (6.16) is not a convex formulation due to the bilinear term PK. Nevertheless, via a simple change of variable $H = PK$, we arrive at the following convex formulation

$$\begin{aligned} \min_{P,H} \quad & \lambda_{\max}(P) \\ \text{subject to} \quad & A^T P - C^T H^T + PA - HC = -I \\ & P \succeq 0 \end{aligned} \tag{6.17}$$

Problem (6.17) is a semidefinite programming problem (SDP), that can be modeled and solved by off-the-shelf tools. We can recover $K = P^{-1}H$ from (6.17) after it is solved.

Interestingly, solving problem (6.17) verifies that the minimum $\lambda_{\max}(P)$ is 5 and the maximum converge rate is 0.1. An optimal solution of (6.17) is

$$P^\star = \begin{bmatrix} 2.4923 & 0 \\ 0 & 5 \end{bmatrix}, \quad K^\star = \begin{bmatrix} 0.2006 \\ 0.4985 \end{bmatrix}.$$

You should check out the Matlab code of this example here.

6.1.3 State-affine Template

Consider an instance of the normal form (6.7) where the dynamics is linear in ξ, but the coefficients are time-varying and dependent on the input and output

$$\dot{\xi} = A(u, y)\xi + B(u, y), \quad y = C(u)\xi. \tag{6.18}$$

The difference between the state-affine template (6.18) and the Luenberger template (6.10) is that the linear matrices A, C are allowed to depend nonlinearly on the input (u, y).

Kalman and Bucy originally proposed an observer for linear time-varying systems (Kalman and Bucy, 1961). The result is later extened by (Besançon et al., 1996) and (Hammouri and de Leon Morales, 1990) to deal with coefficient matrices dependent on the control. The following theorem is a direct extension of the result from (Besançon et al., 1996) and (Hammouri and de Leon Morales, 1990) by considering (u, y) as an augmented control input.

Before presenting the theorem, we need to introduce the following terminology.

Definition 6.3 (Linear Time-Varying System)**.** For a linear time-varying system of the form

$$\dot{\chi} = A(\nu)\chi, \quad y = C(\nu)\chi, \tag{6.19}$$

with input ν and output y, we define

- the *transition matrix* Ψ_ν as the unique solution to

$$\Psi_\nu(t,t) = I, \quad \frac{\partial \Psi_\nu}{\partial \tau}(\tau,t) = A(\nu(\tau))\Psi_\nu(\tau,t).$$

  Note that the transition matrix is used to express the solution to (6.19) because it satisfies

$$\chi(\chi_0, t_0; t; \nu) = \Psi_\nu(t, t_0)\chi_0.$$

- the *observability grammian* as

$$\Gamma_\nu(t_0,t_1) = \int_{t_0}^{t_1} \Psi_\nu(\tau,t_0)^T C(\nu(\tau))^T C(\nu(\tau))\Psi_\nu(\tau,t_0)d\tau.$$

- the *backward observability grammian* as

$$\Gamma^b_\nu(t_0,t_1) = \int_{t_0}^{t_1} \Psi_\nu(\tau,t_1)^T C(\nu(\tau))^T C(\nu(\tau))\Psi_\nu(\tau,t_1)d\tau.$$

We now introduce the Kalman-Bucy Observer for the state-affine template (6.18).

Theorem 6.3 (Kalman-Bucy Observer)**.** *Let $y_{\xi_0,u}(t) = C(u(t))\Xi(\xi_0;t;u)$ be the output of system* (6.18) *at time t with initialization ξ_0 and control u. Suppose the control u satisfies*

- *For any ξ_0, $t \mapsto A(u(t), y_{\xi_0,u}(t))$ is bounded by $A_{\max}$*
- *For any ξ_0, the augmented input $\nu = (u, y_{\xi_0,u})$ is* regularly persistent *for the dynamics*

$$\dot{\chi} = A(\nu)\chi, \quad y = C(\nu)\chi \tag{6.20}$$

  uniformly with respect to ξ_0. That is, there exist strictly positive numbers $t_0, \bar{t}$, and α such that for any ξ_0 and any time $t \geq t_0 \geq \bar{t}$,

$$\Gamma^b_v(t-\bar{t},t) \succeq \alpha I,$$

  where Γ^b_v is the backward observability grammian *associated with system* (6.20)*.*

Then, given any positive definite matrix P_0, there exist $\alpha_1, \alpha_2 > 0$ such that for any $\lambda \geq 2A_{\max}$ and any $\xi_0 \in \mathbb{R}^p$, the matrix differential equation

$$\dot{P} = -\lambda P - A(u,y)^T P - PA(u,y) + C(u)^T C(u) \tag{6.21}$$

initialized at $P(0) = P_0$ admits a unique solution satisfying $P(t) = P(t)^T$ and

$$\alpha_2 I \succeq P(t) \succeq \alpha_1 I.$$

Moreover, the system

$$\dot{\hat{\xi}} = A(u,y)\hat{\xi} + B(u,y) + K(y - C(u)\hat{\xi}) \tag{6.22}$$

with a time-varying gain matrix

$$K = P^{-1}C(u)^T \tag{6.23}$$

is an observer for the state-affine system (6.18).

Let us work out an example of the Kalman-Bucy Observer for nonlinear systems.

Example 6.2 (Kalman-Bucy Observer for A Simple Pendulum)**.** Let us reconsider the pendulum dynamics (6.14) but this time try to design a Kalman-Bucy observer.

We first write the pendulum dynamics in a new coordinate system so that it is in the state-affine normal form (6.18). We choose $\xi = [\mathfrak{s}, \mathfrak{c}, \dot{\theta}]^T$ with $\mathfrak{s} = \sin\theta$ and $\mathfrak{c} = \cos\theta$. Clearly, we will be able to observe $y = [\mathfrak{s}, \mathfrak{c}]^T$ in this new coordinate. The state-affine normal form of the pendulum dynamics reads

$$\begin{aligned} \dot{\xi} = \begin{bmatrix} \mathfrak{c}\dot{\theta} \\ -\mathfrak{s}\dot{\theta} \\ -\frac{1}{ml^2}(b\dot{\theta} + mgl\mathfrak{s}) + \frac{1}{ml^2}u \end{bmatrix} &= \underbrace{\begin{bmatrix} 0 & 0 & \mathfrak{c} \\ 0 & 0 & -\mathfrak{s} \\ 0 & 0 & -\frac{b}{ml^2} \end{bmatrix}}_{=:A(u,y)} \xi + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \frac{u - mgl\mathfrak{s}}{ml^2} \end{bmatrix}}_{=:B(u,y)}. \\ y &= \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{=:C(u)} \xi \end{aligned} \tag{6.24}$$

Note that $C(u)$ is in fact time-invariant, and $B(u,y)$ only depends on u; but we adopt the same notation as the general state-affine template (6.18).

In order to use the Kalman-Bucy observer in Theorem 6.3, we need to verify the boundedness of $A(u,y)$, and the regular persistence of (6.20).

Boundedness of $A(u,y)$. We can easily show the boundedness of $A(u,y)$ by writing

$$\|A(u,y)\xi\| = \|\xi_3(\mathfrak{c} - \mathfrak{s} - b/ml^2)\| \le |\xi_3|\sqrt{3}\sqrt{\mathfrak{c}^2 + \mathfrak{s}^2 + b^2/m^2l^4} \le \|\xi\|\sqrt{3 + 3b^2/m^2l^4}.$$

Therefore, we can take $A_{\max} = \sqrt{3 + 3b^2/m^2l^4}$.

Regular persistence. We write the backward observability grammian of system (6.20)

$$\Gamma^b_\nu(t - \bar{t}, t) = \int_{t-\bar{t}}^{t} \Psi_\nu(\tau,t)^T C^T C \Psi_\nu(\tau,t) d\tau = \int_{t-\bar{t}}^{t} \Psi_\nu(\tau,t)^T \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{=:\tilde{C}} \Psi_\nu(\tau,t) d\tau.$$

$\Gamma_\nu^b(t-\bar{t},t) \succeq \alpha I$ if and only if

$$w^T\Gamma_\nu^b(t-\bar{t},t)w \geq \alpha, \quad \forall w \in \mathbb{R}^3, \|w\| = 1.$$

With this, we develop $w^T\Gamma_\nu^b(t-\bar{t},t)w$

$$\begin{aligned} w^T\Gamma_\nu^b(t-\bar{t},t)w &= \int_{t-\bar{t}}^{t} s^T\tilde{C}s d\tau, \\ &= \int_{t-\bar{t}}^{t} (s_1^2+s_2^2)\, d\tau, \quad s = \Psi_\nu(\tau,t)w \end{aligned} \tag{6.25}$$

and observe that $s = \Psi_\nu(\tau,t)w$ is equivalent to

$$w = (\Psi_\nu(\tau,t))^{-1}s = \Psi_\nu(t,\tau)s,$$

that is, w is the solution of $\dot{\xi} = A(\nu)\xi$ at time t with initial condition s at time $\tau \leq t$. Equivalently, this is saying s is the initial condition of $\dot{\xi} = A(\nu)\xi$ at time $\tau \leq t$ such that its solution at time t is w. Note that from (6.25) it is clearly that $\int_{t-\bar{t}}^{t} (s_1^2+s_2^2)\, d\tau \geq 0$, and $\int_{t-\bar{t}}^{t} (s_1^2+s_2^2)\, d\tau = 0$ if and only if $s_1^2+s_2^2 = 0$, or equivalently $s_1 = s_2 = 0$ for any $\tau \in [t-\bar{t},t]$.

We then take a closer look at the system $\dot{\xi} = A(\nu)\xi$:

$$\begin{aligned} \dot{\xi}_1 &= \mathfrak{c}\xi_3 \\ \dot{\xi}_2 &= -\mathfrak{s}\xi_3 \\ \dot{\xi}_3 &= -\frac{b}{ml^2}\xi_3. \end{aligned} \tag{6.26}$$

If the solution of ξ_3 at time t is w_3, then

$$\xi_3(\tau) = w_3 e^{\frac{b}{ml^2}(t-\tau)}, \quad \tau \leq t.$$

We can now claim it is impossible that $s_1 = s_2 = 0$ at any time $\tau \in [t-\bar{t},t]$.

We can show this by contradiction. First of all, $s_1 = s_2 = 0$ at $\tau = t$ implies $w_1 = w_2 = 0$ and hence $w_3 = \pm 1$. This implies $\xi_3 \neq 0$ for any $\tau \in [t-\bar{t},t]$. Then, $s_1 = 0, \forall \tau \in [t-\bar{t},t]$ implies $\dot{\xi}_1 = 0$ which, due to $\xi_3 \neq 0$, implies $\mathfrak{c} = 0$ for all τ. Similarly, $s_2 = 0, \forall \tau \in [t-\bar{t},t]$ implies $\dot{\xi}_2 = 0$ and $\mathfrak{s} = 0$. This creates a contradiction because $\mathfrak{c}^2 + \mathfrak{s}^2 = 1$ and $\mathfrak{c}, \mathfrak{s}$ cannot be simultaneously zero.

The above reasoning proves that the backward observability Grammian is positive definite, which is, however, still insufficient for the Kalman-Bucy observer. We need a stronger uniformly positive definite condition on Γ_ν^b, i.e., to find $t_0, \bar{t}$ and $\alpha > 0$ so that $\Gamma_\nu^b(t-\bar{t},t) \succeq \alpha I$ for all $t \geq t_0$.

If the control u is unbounded, then sadly, one can show that the uniform positive definite condition fails to hold, as left by you to show in the following exercise.

Exercise 6.1 (Counterexample for Kalman-Bucy Observer)**.** Show that, if the control u is unbounded, then for any $\alpha > 0$, $t_0 \geq \bar{t} > 0$, there exists $t \geq t_0$ such that $\Gamma_\nu^b(t - \bar{t}, t) \prec \alpha I$. (Hint: consider a controller that spins the pendulum faster and faster such that in time $\bar{t}$ it has rotated $2k\pi$, in this case the angular velocity becomes unobservable because we are not sure how many rounds the pendulum has rotated.)

Fortunately, if the control u is bounded, then we can prove the uniform positive define condition holds for $\Gamma_\nu^b(t-\bar{t}, t)$. The following proof is given by Weiyu Li.

Without loss of generality, let $\frac{b}{ml^2} = 1$. Assume u is bounded such that the third entry of $B(u, y)$ in (6.24) is bounded by $\beta > 0$

$$\left|\frac{u - mgl\mathfrak{s}}{ml^2}\right| \leq \beta.$$

Assuming the initial velocity of the pendulum is $\dot{\theta}(0) = \dot{\theta}_0$, we know $\dot{\theta}(t)$ is bounded by

$$\dot{\theta}(t) \in [c_1(1-\beta)e^{-t} - \beta, c_2(1-\beta)e^{-t} + \beta],$$

where c_1, c_2 are constants chosen to satisfy the initial condition. Clearly, for all $t > 0$, we see $\dot{\theta}(t)$ is bounded, and hence we know $\dot{\mathfrak{c}}$ and $\dot{\mathfrak{s}}$ are bounded (due to $\mathfrak{c}$ and $\mathfrak{s}$ are bounded). Intuitively, what we have just shown says that when the control is bounded, the measurements $\mathfrak{c}$ and $\mathfrak{s}$ will have bounded time derivatives. (This will help us analyze the auxiliary system (6.26).)

Now back to checking regular persistence of the auxilary system (6.26). We will discuss two cases: (1) $w_3^2 > 1-\delta$, and (2) $w_3^2 \leq 1-\delta$, for some constant $\delta < 0.5$ determined later.

1. $w_3^2 > 1 - \delta > 0.5$. In this case we have $w_1^2 + w_2^2 = 1 - w_3^2 < \delta$, and hence $w_1^2 < \delta$, $w_2^2 < \delta$. On the other hand, from (6.26) we have

   $$\dot{\xi}_1^2(\tau) + \dot{\xi}_2^2(\tau) = \xi_3^2 = w_3^2 e^{2(t-\tau)} > w_3^2 > 1 - \delta, \quad \forall \tau < t.$$

   Without loss of generality assume $\dot{\xi}_1(t)^2 > (1-\delta)/2$. As $\dot{\xi}_1 = \mathfrak{c}\xi_3$ and both $\mathfrak{c}$ and ξ_3 have bounded derivatives, we know $\dot{\xi}_1$ will not change sign for some duration T that is independent from the choice of δ (because the time derivatives of $\mathfrak{c}$ and ξ_3 do not depend on δ). That is $|\dot{\xi}_1| > \sqrt{(1-\delta)/2} > 1/2$ for $\tau \in [t-T, t]$. Consequently,

   $$|\xi_1(t-\tau)| > \frac{1}{2}\tau - |w_1| > \frac{1}{2}\tau - \sqrt{\delta}, \quad \tau \in [0, T].$$

   Choosing δ small enough, we have $|\xi_1(t-\tau)| > 0.25\tau$ for $\tau \in [0.5T, T]$. Then we have

   $$\Gamma_\nu^b(t-T, t) \succ [(0.25 \times 0.5T)^2 \times 0.5T]I.$$

2. $w_3^2 \leq 1 - \delta$. In this case $w_1^2 + w_2^2 = 1 - w_3^2 \geq \delta$, and at least one of w_1 and w_2 has absolute value larger than $\sqrt{\delta/2}$. Because the derivatives of ξ_1 and ξ_2 are both bounded, we know ξ_1 and ξ_2 will remain large for some constant time. Thus there is a uniform lower bound.

The intuition of the above proof is simple: when ξ_1 and/or ξ_2 already have large absolute value (case 2), we can find a time window such that ξ_1 and/or ξ_2 remain large in that time window; when ξ_1 and/or ξ_2 are small (case 1), using the observation that their time derivatives are large (because w_3 is large), together with the fact that these derivatives remain large (because the derivative of these derivatives are bounded), we can also find a time window that ξ_1 and/or ξ_2 are large (back in time). Therefore, the backward observability Grammian is uniformly positive definite.

6.1.4 Kazantzis-Kravaris-Luenberger (KKL) Template

In Luenberger's original paper about observer design for linear systems (Luenberger, 1964), the goal was to transform a linear system

$$\dot{x} = Fx, \quad y = Cx$$

into a Hurwitz form

$$\dot{\xi} = A\xi + By \tag{6.27}$$

with A a Hurwitz (stable) matrix. If such a transformation is available, then the following system

$$\dot{\hat{\xi}} = A\hat{\xi} + By,$$

which is nothing but a copy of the dynamics (6.27), is in fact an observer. This is because the error $e = \hat{\xi} - \xi$ evolves as

$$\dot{e} = Ae,$$

which implies that e tends to zero regardless of the initial error $e(0)$. Luenberger proved that when (F, C) is observable, a stationary transformation $\xi = Tx$ with $p = n$, i.e., $T \in \mathbb{R}^{n\times n}$, always exists and is unique, for any matrix A that is Hurwitz and (A, B) that is controllable. This is based on the fact that

$$(AT + BC)x = A\xi + By = \dot{\xi} = T\dot{x} = TFx, \forall x \tag{6.28}$$

$$\iff AT + BC = TF, \tag{6.29}$$

known as the Sylvester equation, admits a unique and invertible solution T.

A natural extension of Luenberger's original idea is to find a transformation that converts the nonlinear system (6.1) into the following form

$$\dot{\xi} = A\xi + B(u, y), \quad y = H(\xi, u), \tag{6.30}$$

with A a Hurwitz matrix (but H can be nonlinear, as opposed to the Luenberger template in Theorem 6.2). If such a transformation can be found, then we can design a similar observer that copies the dynamics (6.30)

$$\dot{\hat{\xi}} = A\hat{\xi} + B(u, y). \tag{6.31}$$

We refer to such a nonlinear Luenberger template the Kazantzis-Kravaris-Luenberger (KKL) template, due to the seminal work (Kazantzis and Kravaris, 1998).

The KKL template, once found, is nice in the sense that (i) the observer (6.31) is a simple copy of the dynamics and also very easy to implement (as opposed to the Kalman-Bucy observer); and (ii) checking if the matrix A is Hurwitz is easy, at least when A has reasonable size, (e.g., compared to checking the regular persistence condition in the state-affine template in Theorem 6.3).

However, the KKL template is difficult to realize in the sense that (i) what kind of nonlinear systems can be converted to (6.30), and (ii) for those systems, how do we find the coordinate transformation?

Recent works have leveraged deep learning to learn the coordinate transformation, for example in (Janny et al., 2021), (Niazi et al., 2023), (Miao and Gatsis, 2023). Before hammering the problem with deep learning, let us look at the fundamentals of the KKL observer.

6.1.4.1 Autonomous Systems

Consider the autonomous version of system (6.1) without control

$$\dot{x} = f(x), \quad y = h(x), \tag{6.32}$$

where $x \in \mathbb{X} \subseteq \mathbb{R}^n, y \in \mathbb{Y} \subseteq \mathbb{R}^d$.

The following result, established by (Andrieu and Praly, 2006), states that the KKL observer exists under mild conditions.

Theorem 6.4 (KKL Observer for Autonomous Systems)**.** *Assum $\mathcal{X}$ and $\mathcal{L}$ are open bounded sets in $\mathbb{X}$ (the state space) such that* $\mathrm{cl}(\mathcal{X})$ *is contained in $\mathcal{L}$ and the system* (6.32) *is backward $\mathcal{L}$-distinguishable on $\mathcal{X}$ (cf. Definition 6.2). Then there exists a strictly positive number γ and a set $\mathcal{S}$ of zero Lebesgue measure in $\mathbb{C}^{n+1}$ such that denoting $\Omega = \{\lambda \in \mathbb{C} \mid \mathrm{Re}(\lambda) < -\gamma\}$, for any $(\lambda_1, \dots, \lambda_{n+1}) \in \Omega^{n+1} \backslash \mathcal{S}$, there exists a function $T : \mathbb{R}^n \to \mathbb{R}^{(n+1)d}$ uniformly injective on $\mathcal{X}$ satisfying*

$$L_f T(x) = AT(x) + B(h(x))$$

with

$$A = \tilde{A} \otimes I_d, \quad B(y) = (\tilde{B} \otimes I_d)y \tag{6.33}$$

$$\tilde{A} = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_{n+1} \end{bmatrix} \quad \tilde{B} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{6.34}$$

Moreover, if $\mathcal{X}$ is backward invariant, then T is unique and defined by

$$T(x) = \int_{-\infty}^{0} e^{-A\tau} B(h(X(x,\tau)))d\tau. \tag{6.35}$$

Remark. The function T in Theorem 6.4 takes complex numbers. To simulate the observer

$$\dot{\hat{\xi}} = A\hat{\xi} + B(y),$$

one needs to implement in real numbers, for each λ_i and $j \in [d]$

$$\dot{\hat{\xi}}_{\lambda_i,j} = \begin{bmatrix} -\mathrm{Re}(\lambda_i) & -\mathrm{Im}(\lambda_i) \\ \mathrm{Im}(\lambda_i) & -\mathrm{Re}(\lambda_i) \end{bmatrix} \hat{\xi}_{\lambda_i,j} + \begin{bmatrix} y_j \\ 0 \end{bmatrix}.$$

Therefore, the dimension of the observer is $2 \times d(n+1)$.

Theorem 6.4 states that as long as the system (6.32) is backward distinguishable, then there exists a stationary transformation T that can transform the system to a new coordinate system ξ such that the dynamics in ξ is Hurwitz. A closer look at the structure of A and B reveals that the coordinate transformation needs to satisfy $n+1$ differential equations of the form

$$\frac{\partial T_\lambda}{\partial x}(x)\dot{x} = \lambda T_\lambda(x) + y$$

where each T_λ transforms the state x into a new coordinate having the same dimension of y. Clearly, if $T = (T_\lambda)$, i.e., there is a single λ, then T is not uniformly injective (as the dimension of ξ is $d < n$). Consequently, by choosing

$$T = (T_{\lambda_1}, \dots, T_{\lambda_{n+1}}),$$

the uniform injectivity of T is ensured.

However, the difficulty lies in the computation of T (and T_λ), let alone its inverse (that recovers x from ξ). Even though $\mathcal{X}$ is backward invariant, the formulation (6.35) is difficult to compute. I tried very hard to find a coordinate transformation T that can convert the non-controlled pendulum dynamics into the KKL form but did not succeed. **You should let me know if you were able to find one!** Nevertheless, the following example shows you the flavor of how such a transformation may look like for a different system.

Example 6.3 (KKL Observer for an Oscillator with Unknown Frequency)**.** Consider a harmonic oscillator with unknown frequency

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -x_1 x_3 \\ \dot{x}_3 = 0 \end{cases}, \quad y = x_1$$

Consider the coordinate transformation

$$T_{\lambda_i}(x) = \frac{\lambda_i x_1 - x_2}{\lambda_i^2 + x_3}, \quad \lambda_i > 0, i = 1, \dots, p.$$

We have

$$\frac{\partial T_{\lambda_i}(x)}{\partial x}\dot{x} = \left\langle \begin{bmatrix} \frac{\lambda_i}{\lambda_i^2+x_3} \\ \frac{-1}{\lambda_i^2+x_3} \\ \frac{x_2-\lambda_i x_1}{(\lambda_i^2+x_3)^2} \end{bmatrix}, \begin{bmatrix} x_2 \\ -x_1 x_3 \\ 0 \end{bmatrix} \right\rangle = \frac{\lambda_i x_2 + x_1 x_3}{\lambda_i^2 + x_3} \tag{6.36}$$

$$-\lambda_i T_{\lambda_i}(x) + y = \frac{-\lambda_i^2 x_1 + \lambda_i x_2 + x_1 \lambda_i^2 + x_1 x_3}{\lambda_i^2 + x_3} = \frac{\lambda_i x_2 + x_1 x_3}{\lambda_i^2 + x_3} \tag{6.37}$$

Therefore, with

$$\xi = T(x) = [T_{\lambda_1}(x), T_{\lambda_2}(x), \dots, T_{\lambda_p}(x)]^T,$$

we have

$$\dot{\xi} = \underbrace{\begin{bmatrix} -\lambda_1 & & \\ & \ddots & \\ & & -\lambda_p \end{bmatrix}}_{A} \xi + \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} y$$

with A clearly Hurwitz.

With some extra arguments (cf. Section 8.1.1 in (Bernard, 2019)), one can see that the transformation T is injective with $p \geq 4$ distinct λ_i's. Therefore, this is a valid KKL observer.

The final issue that one needs to think about is, since the observer is estimating $\hat{\xi}$, how to recover $\hat{x}$? In this example, there is actually no analytical formula for recovering $\hat{x}$ from $\hat{\xi}$. In this case, one approach is to solve the following optimization problem

$$\hat{x} = \arg\min_x \|\hat{\xi} - T(x)\|^2,$$

which may be quite expensive.

A more general treatment is given in Section 8.2.2 in (Bernard, 2019).

6.1.4.2 Controlled Systems

6.1.5 Triangular Template

6.1.6 Design with Convex Optimization

Consider a nonlinear system

$$\dot{x} = f(x) + \psi(u, y), \quad y = Cx \tag{6.38}$$

where $x \in \mathbb{X} \subseteq \mathbb{R}^n$, $y \in \mathbb{R}^d$, C a constant matrix, and $\psi(u, y)$ a nonlinear function. We assume that $f(x)$ is a polynomial vector map (i.e., each entry of f is a polynomial function in x). Certainly the formulation in (6.38) is not as general as (6.1), but it is general enough to include many examples in robotics.

Recall that I said the essence of observer design is to (i) simulate the dynamics when the state estimation is correct, and (ii) to correct the state estimation from observation when it is off. Therefore, we wish to design an observer for (6.38) in the following form

$$\dot{\hat{x}} = \underbrace{f(\hat{x}) + \psi(u, y)}_{\text{dynamics simulation}} + \underbrace{K(y - \hat{y}, y)(C\hat{x} - y)}_{\text{feedback correction}}, \tag{6.39}$$

where, compared to the Luenberger observer (6.11), we allow the gain matrix K to be nonlinear functions of the true observation y and the estimated observation $\hat{y}$.

With the observer (6.39), the dynamics on the estimation error $e = \hat{x} - x$ becomes

$$\dot{e} = f(x + e) - f(x) + K(Ce, Cx)Ce.$$

If we can find a Lyapunov-like function $V(e)$ so that $V(e)$ is positive definite and $\dot{V}(e)$ is negative definite, then Lyapunov stability theorem 5.3 tells us that $e = 0$ is asymptotically stable. Because we do not know the gain matrix K either, we need to jointly search for V and K (that are polynomials). Mathematically, this is

$$\begin{aligned} \text{find} \quad & V, K \\ \text{subject to} \quad & V(0) = 0, \quad V(e) > 0, \forall e \neq 0 \\ & \dot{V}(e) = \frac{\partial V}{\partial e}\left(f(x + e) - f(x) + K(Ce, Cx)Ce\right) < 0, \forall e \neq 0, \forall x \in \mathbb{X} \\ & V(e) \geq \epsilon \|e\|^2, \forall e \end{aligned} \tag{6.40}$$

where the last constraint is added to make sure $V(e)$ is radially unbounded. Furthermore, if we replace the second constraint by $\dot{V}(e) \leq -\lambda V(e)$, then we can guarantee $V(e)$ converges to zero exponentially.

Problem (6.40), however, is not a convex optimization problem, due to the term $\frac{\partial V}{\partial e}K$ being bilinear in the coefficients of V and K. Nevertheless, as shown in (Ebenbauer et al., 2005), we can use a reparameterization trick to formulate a stronger version of (6.40) as follows.

$$\begin{aligned} \text{find} \quad & V, Q(Ce), M(Ce, Cx) \\ \text{subject to} \quad & V(0) = 0, \quad V(e) > 0, \forall e \neq 0 \\ & \frac{\partial V}{\partial e} = e^T Q(Ce), \quad Q(Ce) \succ 0 \\ & e^T Q(Ce)\left(f(x+e) - f(x)\right) + e^T M(Ce, Cx)Ce < 0, \forall e \neq 0, \forall x \in \mathbb{X} \\ & V(e) \geq \epsilon \|e\|^2, \forall e \end{aligned} \tag{6.41}$$

Clearly, if we can solve problem (6.41), then

$$K = Q(Ce)^{-1} M(Ce, Cx)$$

is the right gain matrix for the formulation (6.40).

Let us bring this idea to action in our pendulum example.

Example 6.4 (Pendulum Observer with Convex Optimization)**.** With $x = [\mathfrak{s}, \mathfrak{c}, \dot{\theta}]^T$ ($\mathfrak{s} = \sin\theta, \mathfrak{c} = \cos\theta$), we can write the pendulum dynamics as

$$\dot{x} = \underbrace{\begin{bmatrix} \mathfrak{c}\dot{\theta} \\ -\mathfrak{s}\dot{\theta} \\ -\frac{b}{ml^2}\dot{\theta} \end{bmatrix}}_{=:f(x)} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \frac{u - mgl\mathfrak{s}}{ml^2} \end{bmatrix}}_{=:\psi(u,y)}, \quad y = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{=:C} x$$

Clearly $f(x)$ is a polynomial. Solving the convex optimization problem (6.41), we obtain a solution

$$V(e) = 0.5954e_1^2 + 0.5954e_2^2 + 0.9431e_3^2$$

$$Q(Ce) = \begin{bmatrix} 0.4603e_2^2 + 1.1909 & -0.4603e_1e_2 & 0 \\ -0.4603e_1e_2 & 0.4603e_1^2 + 1.1909 & 0 \\ 0 & 0 & 1.8863 \end{bmatrix}$$

$$M(Ce, Cx) = \begin{bmatrix} \begin{matrix} -2.0878e_1^2 - 0.8667e_2^2 - \\ 0.4588(y_1^2 + y_2^2) - 0.4885 \end{matrix} & -0.8667e_1e_2 \\ -0.8667e_1e_2 & \begin{matrix} -0.8667e_1^2 - 2.0878e_2^2 - \\ 0.4588(y_1^2 + y_2^2) - 0.4885 \end{matrix} \\ -1.1909y_2 & 1.1909y_1 \end{bmatrix}$$

Simulating this observer, we verify that the observer is in fact exponentially converging, as shown in Fig. 6.1.

The Matlab code for formulating and solving the convex optimization (6.41) can be found here. The code for simulating the observer can be found here.

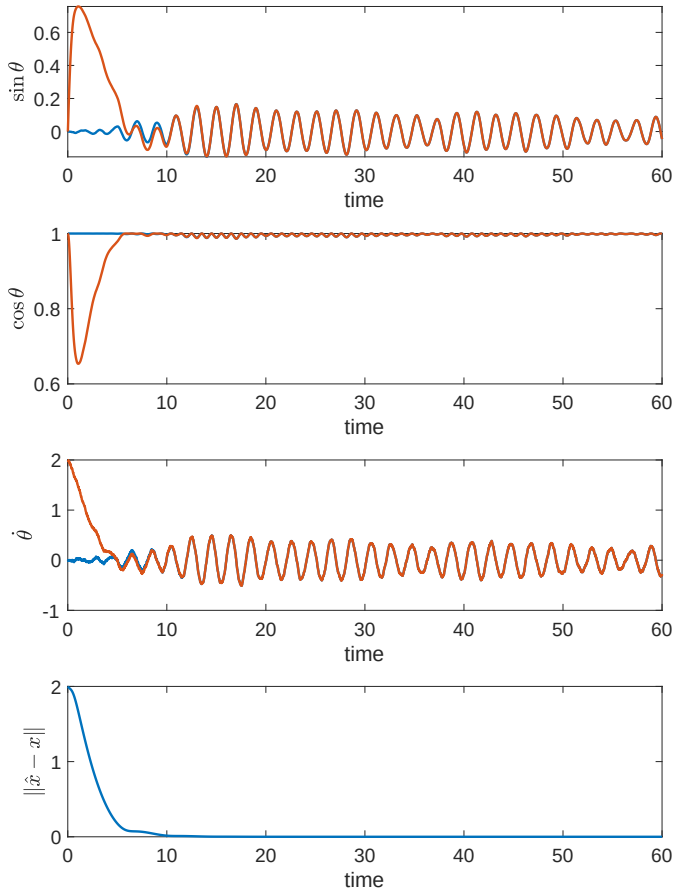

Figure 6.1: Simulation of the pendulum observer design from convex optimization

6.2 Observer Feedback

Now that we have good ways to design a state observer, we will see how we can use the observer for feedback control.

Example 6.5 (Pendulum Stabilization with A Luenberger Observer)**.** In Example 6.1, we have written the dynamics of a pendulum, and the dynamics of a Luenberger observer as

$$\dot{x} = Ax + B(u, y) \tag{6.42}$$

$$\dot{\hat{x}} = A\hat{x} + B(u, y) + KC(x - \hat{x}) \tag{6.43}$$

We wish to understand (so we can optimize) the behavior of this system under certain control input u. To do so, let us denote $e = \hat{x} - x$, and write the above dynamics as

$$\dot{x} = Ax + B(u, Cx) \tag{6.44}$$

$$\dot{e} = (A - KC)e \tag{6.45}$$

Denoting $z = [x, e]^T$, we have the augmented dynamics

$$\dot{z} = \underbrace{\begin{bmatrix} A & 0 \\ 0 & A - KC \end{bmatrix}}_{=:F} z + \underbrace{\begin{bmatrix} B(u, Dz) \\ 0 \end{bmatrix}}_{=:G(z,u)}$$

We want to stabilize the system at $z_0 = [\pi, 0, 0, 0]^T$ (the upright position) subject to control bounds $u \in \mathbb{U} = [-u_{\max}, u_{\max}]$.

We need to find a control Lyapunov function (CLF), $V(z)$, that satisfies the following constraints:

$$V(z_0) = 0$$

$$V(z) > 0 \quad \forall z \in \{z : V(z) < \rho, z \neq z_0\}$$

$$\inf_{u \in \mathbb{U}} [L_F V(z) + L_G V(z)] \leq 0 \quad \forall z \in \mathcal{Z}$$

where $L_F V$ and $L_G V$ are the Lie derivatives of V along F and $G(z, u)$, respectively. $\mathcal{Z}$ is the set of all possible augmented states. The CLF will define the set of admissible control inputs U.

$$U = \{u : L_f V(z) + L_g V(z) u \leq 0\}$$

To find the smallest-magnitude control input such that $u \in K$, we may use a quadratic program:

$$\min_{u \in \mathcal{U}} ||u||^2$$

$$\text{s.t.} \quad L_f V(z) + L_g V(z) u \leq -cV(z)$$

where c is some positive constant. The challenge now is in choosing a suitable $V(z)$.

Chapter 7

Geometric Vision

In this Chapter, we introduce the fundamentals of *geometric vision*, a (classical) branch of computer vision that seeks to estimate geometric models (e.g., 3D rotations, translations, and points) from sensor measurements (e.g., images and point clouds). There are two goals for introducing geometric vision.

1. In the output feedback Chapter 6, we see that the full state x of a dynamical system is often not available, and needs to be estimated from the measurement signal y that satisfies

   $$y(t) = h(x(t), u(t))$$

   potentially plus some noise. In Chapter 6, we studied the case where y is part of the state x, often the position q of a second-order system $x = [q; \dot{q}]$. For example, in the pendulm swing-up example, we assume the angular position θ is observed, but not the angular velocity $\dot{\theta}$. However, in many practical applications, the measurement signal y does not directly tell us the position q, and we need to estimate q from y. For instance, a quadcopter needs to estimate its position from its onboard cameras. Once we obtain an estimated q from y, we can use the observer synthesis methods in Chapter 6 to obtain the full state estimation.

2. The estimation community and the control community are a bit separated (at least in my opinion), despite that they share a lot of common tools, especially *optimization*. We will see that estimating q from y, where y could be a high-dimensional image, is often formulated as an optimization problem that is difficult to solve. However, using the SOS tool we developed in Chapter 5, we can actually solve the optimization problem to global optimality.

7.1 3D Rotations and Poses

7.1.1 Rotation matrices

The first part is a quick recap of the basics in linear algebra.

Definition 7.1 (Orthogonal Matrix)**.** We call a $n \times n$ square matrix A orthogonal if the column of A is orthogonal to each other and all the column vectors have unit length. The set of all $n \times n$ orthogonal matrices is denoted as $O(n)$.

Below are some basic properties of orthogonal matrices:

Proposition 7.1 (Property of Orthogonal Matrix)**.** *Let A be a $n \times n$ orthogonal matrix. Then:*

1. *$A^T = A^{-1}$ and A^T is also a orthogonal matrix.*
2. *For every orthogonal matrices A, B, AB is also a orthogonal matrix.*
3. $\det(A) = \pm 1$.
4. *A preserves dot product, i.e. $< x, y >=< Ax, Ay >$, thus preserves the length of a vector, i.e. $\|Ax\|_2 = \|x\|_2$.*
5. *All the eigenvalues of A have modulus one.*

Proof. We only offer the proof of the last property. Consider any eigenvalue λ of A, and x be its eigenvector. We have $< x, x >=< A^T Ax, x >=< Ax, Ax >= |\lambda|^2 < x, x >$, thus $|\lambda| = 1$. □

There are two types of orthogonal matrices, categorized by determinant 1 and -1. Those with determinant 1 are called rotation matrices. The set of rotation matrices is denoted as $SO(n)$ (Special Orthogonal). In the world of robotics and most engineering fields, we care about $SO(3)$ the most. Below are some basic properties of 3×3 rotation matrices:

Proposition 7.2 (Property of Rotation Matrix)**.** *Let A be a 3×3 orthogonal matrix. Then:*

1. $\det(A) = \pm 1$
2. *For every rotation matrices A, B, A^T,AB are also rotation matrices.*
3. *A always has an eigenvalue 1. If A is not identity, A either has two conjugate complex eigenvalues not equal to 1, or has two eigenvalues -1.*

Proof. We can only prove the last property.

From the property of orthogonal matrix, we know that all eigenvalues have modulus one. First, note that there must exist at least one real eigenvalue. Because eigenvalues with nonzero imaginary parts always come in pair and 3 is an odd number.

Then there are two possible cases: (1) All the eigenvalues are real; (2) There is only one eigenvalue.

For case one, note that the determinant of a matrix is the product of all eigenvalues, then rotation matrix can't have all the eigenvalues -1.

For case two. The product of the pairing complex eigenvalues is the square of the modulus of the eigenvalue, which is 1. So the real eigenlvalue left must be 1.

Finally if all the eigenvalues of A are 1, then A must be identity. □

7.1.2 Coordinate Frame

Coordinate frames are a set of orthogonal basis (containing three axes) attached to a certain body at a point. It serves as the tool to describe the position of points relative to that body. Conventionally, coordinate frames are right-handed. We will encounter different frames in applications, including: (1) Robot (robot frame "r"), (2) Sensor on the robot (e.g. camera frame 'c'), (3) A fixed location in the world (world frame "w")

It's worth metion that, denote $\vec{x}, \vec{y}, \vec{z}$ as the three axes of the coordinate frame, then the right-handed property can be expressed as: $\vec{x} \cdot (\vec{y} \times \vec{z}) = 1$, which is the same as $\det([\vec{x}, \vec{y}, \vec{z}]) = 1$. So the matrix $[\vec{x}, \vec{y}, \vec{z}]$ is a rotation matrix.

It's natural for us to ask three questions: (1) How to express a point in a given frame? (2) How to represent a frame r with respect to a frame w? (3) How to translate the coordinate of a point in different frames?

(1) How to express a point in a given frame?

Let's consider a reference frame r and denote the the three axes attached to it as $\vec{x^r}, \vec{y^r}, \vec{z^r}$. Then for any point p, we care about the vector pointing from the origin of the frame to p. We slightly abuse the notation and let that vector called p.(Since we will only care about the vector) Then, we can express p as the combination of the basis, i.e.

$$p = p_x^r \vec{x^r} + p_y^r \vec{y^r} + p_z^r \vec{z^r} = [\vec{x^r}, \vec{y^r}, \vec{z^r}] \begin{bmatrix} p_x^r \\ p_y^r \\ p_z^r \end{bmatrix}$$

Thus we can fully describe point p with three scalars p_x^r, p_y^r, p_z^r, which is called the coordinates of p with respect to the frame r.

(2) How to represent a frame r with respect to a frame w?

Now we consider how to describe frame w in frame r. Let's focus on the simple case where the origin of the two coordinate systems coincide. Then we can express the axes $\vec{x^w}, \vec{y^w}, \vec{z^w}$ in frame r directly. For example, thanks to the orthogonality, we can get:

$$\vec{x^w} = < \vec{x^w}, \vec{x^r} > \vec{x^r} + < \vec{x^w}, \vec{y^r} > \vec{y^r} + < \vec{x^w}, \vec{z^r} > \vec{z^r} = [\vec{x^r}, \vec{y^r}, \vec{z^r}] \begin{bmatrix} < \vec{x^w}, \vec{x^r} > \\ < \vec{x^w}, \vec{y^r} > \\ < \vec{x^w}, \vec{z^r} > \end{bmatrix}$$

So we can get:

$$[\vec{x^w}, \vec{y^w}, \vec{z^w}] = [\vec{x^r}, \vec{y^r}, \vec{z^r}] \begin{bmatrix} < \vec{x^w}, \vec{x^r} > & < \vec{y^w}, \vec{x^r} > & < \vec{z^w}, \vec{x^r} > \\ < \vec{x^w}, \vec{y^r} > & < \vec{y^w}, \vec{y^r} > & < \vec{z^w}, \vec{y^r} > \\ < \vec{x^w}, \vec{z^r} > & < \vec{y^w}, \vec{z^r} > & < \vec{z^w}, \vec{z^r} > \end{bmatrix} = [\vec{x^r}, \vec{y^r}, \vec{z^r}] \, R_w^r$$

Note that $R_w^r = ([\vec{x^r}, \vec{y^r}, \vec{z^r}])^T \, [\vec{x^w}, \vec{y^w}, \vec{z^w}]$ is a rotation matrix.

Example 7.1 (A simple example of translation between frames)**.** If we for frame w we have

$$[\vec{x^w}, \vec{y^w}, \vec{z^w}] = I_3$$

and for frame r we have

$$[\vec{x^r}, \vec{y^r}, \vec{z^r}] = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then we can have:

$$[\vec{x^w}, \vec{y^w}, \vec{z^w}] = [\vec{x^r}, \vec{y^r}, \vec{z^r}] \begin{bmatrix} < \vec{x^w}, \vec{x^r} > & < \vec{y^w}, \vec{x^r} > & < \vec{z^w}, \vec{x^r} > \\ < \vec{x^w}, \vec{y^r} > & < \vec{y^w}, \vec{y^r} > & < \vec{z^w}, \vec{y^r} > \\ < \vec{x^w}, \vec{z^r} > & < \vec{y^w}, \vec{z^r} > & < \vec{z^w}, \vec{z^r} > \end{bmatrix} \tag{7.1}$$

$$= [\vec{x^r}, \vec{y^r}, \vec{z^r}] \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.2}$$

So we can get $R_w^r = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

(3) How to translate the coordinate of a point in different frames?

First let us consider frame w and r with the same origin. Then for any point $\vec{p}$, we will have:

$$\vec{p} = [\vec{x^r}, \vec{y^r}, \vec{z^r}] \begin{bmatrix} p_x^r \\ p_y^r \\ p_z^r \end{bmatrix} = [\vec{x^w}, \vec{y^w}, \vec{z^w}] \begin{bmatrix} p_x^w \\ p_y^w \\ p_z^w \end{bmatrix} \tag{7.3}$$

$$\Rightarrow \vec{p^r} = \begin{bmatrix} p_x^r \\ p_y^r \\ p_z^r \end{bmatrix} = \begin{bmatrix} \vec{x^r}^T \\ \vec{y^r}^T \\ \vec{z^r}^T \end{bmatrix} [\vec{x^w}, \vec{y^w}, \vec{z^w}] \begin{bmatrix} p_x^w \\ p_y^w \\ p_z^w \end{bmatrix} = R_w^r \begin{bmatrix} p_x^w \\ p_y^w \\ p_z^w \end{bmatrix} = R_w^r \vec{p^w} \tag{7.4}$$

We can find that, we only need to multiply the matrix R_w^r to translate the coordinate in w frame to r frame.

Example 7.2 (A simple example of translation between frames (Cont.))**.** If we for frame w we have

$$[\vec{x^w}, \vec{y^w}, \vec{z^w}] = I_3$$

and for frame r we have

$$[\vec{x^r}, \vec{y^r}, \vec{z^r}] = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Assume point p has coordinates $\vec{p^w} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix}$ in frame w. Then we can have:

$$\vec{p^r} = R_w^r \vec{p^w} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

What if the origin of the two frames are not the same? We may think of two ways to do so: (1) First do the translation, then do the rotation, (2) First do the rotation, then do the translation. The most natural way is (1). Here is a quick example:

Example 7.3 (A simple example with different origins)**.**

Suppose we have two frames: Frame 1 and Frame 2 as depicted below. All the coordinates are in frame 1.

$$R_1^2 = \begin{bmatrix} < \vec{x^1}, \vec{x^2} > & < \vec{y^1}, \vec{x^2} > \\ < \vec{x^1}, \vec{y^2} > & < \vec{y^1}, \vec{y^2} > \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$$

and $t_1^2 = \begin{bmatrix} 0 \\ -2 \end{bmatrix}$.

We can see that, if we want to use method 1, we can just consecutively do the translation and the rotation. However, if we want to do it reversely, we must

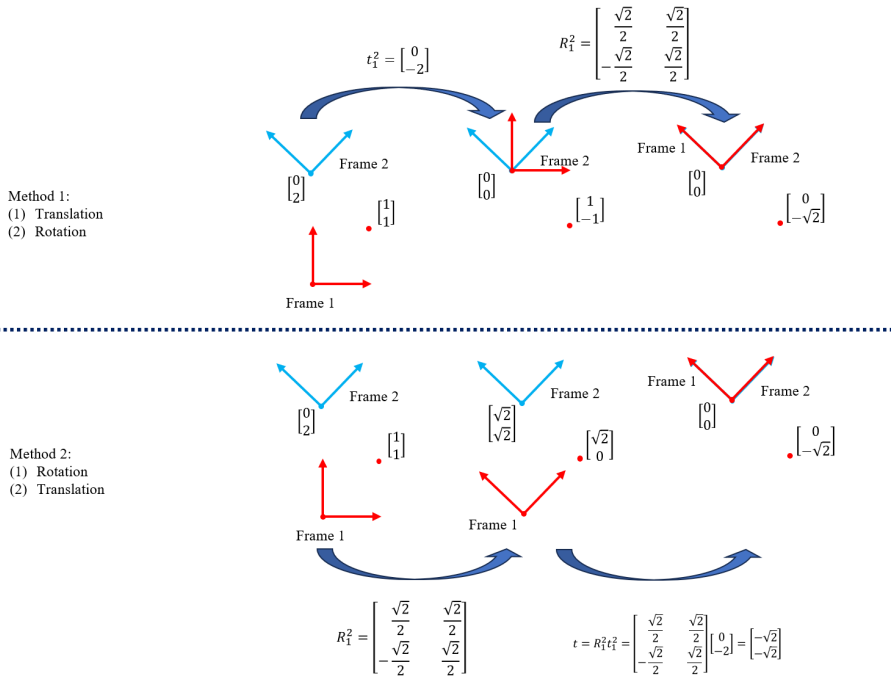

Figure 7.1: A simple example with different origins

multiply the translation by the rotation matrix. Actually it's easy to show this two methods are equivalent.

In conclusion, the formula is as follows:

$$p^2 = R_1^2(p^1 + t_1^2) = R_1^2 p^1 + R_1^2 t_1^2$$

7.1.3 Representations of the rotations

Although rotation matrix is enough to characterize a rotation. But it's not simple enough and not intuitive enough. For example, we can't explicitly know the rotation angles from the rotation matrix. The rotation matrix has 9 elements, but there are many constraints. So is it possible for us to find a simpler representation for rotation?

7.1.3.1 Euler angles representation

Intuitively, we can achieve any rotation by rotating along the x, y, z axes in turn. First let's introduce some basic rotations along x,y,z axes.

Proposition 7.3 (Basic Rotations)**.** *Below are the basic rotation matrices along the x, y, z axes, all the rotations are counterclockwise.*

1. *Rotation along z axes, with angle γ is:*

$$\begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. *Rotation along y axes, with angle β is:*

$$\begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

3. *Rotation along x axes, with angle α is:*

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

Actually every rotation can be written into combination of no more than three basic rotations, with no two consecutive rotations along the same axis. A popular choice of the sequence is roll-pitch-yaw.

Proposition 7.4 (Roll-pitch-yaw angle representation)**.** *Any rotation matrix R_r^w can be written into combination of basic rotations in the following order:*

$$R_r^w = R_z(\gamma)R_y(\beta)R_x(\alpha)$$

where γ is called the yaw angle, β is called the pitch angle, and α is called the roll angle.

This representation is very intuitive, because it directly tells us the rotation angles. However, the calculation will include trigonometric functions. So it can be hard to calculate and analyze. Moreover, if you want to recover the Euler angles from rotation matrices, there may be problems in certain point. The example below shows the singularities.

Example 7.4 (Singularities for Euler angles)**.** In formula

$$R = R_z(\gamma)R_y(\beta)R_x(\alpha)$$

consider $\beta = \frac{\pi}{2}$. Then we will have:

$$R = R_z(\gamma)R_y(\frac{\pi}{2})R_x(\alpha) = \begin{bmatrix} 0 & \sin(\alpha+\gamma) & -\cos(\alpha+\gamma) \\ 0 & \cos(\alpha+\gamma) & \sin(\alpha+\gamma) \\ 1 & 0 & 0 \end{bmatrix}$$

So there are ambiguities in choosing α, γ for the same rotation matrices.

7.1.3.2 Axis-angle representation

Another intuitive representation is the axis-angle representation. Imagine a rotation in 3D space, it seems that all the rotations are rotation with respect to an axis (not necessary to be aligned with the x,y,z axes) for some angle. How to find the axis and the corresponding angle?

Given rotation angle and axis, how can we get the rotation matrix? Next theorem will give us an explicit formula.

Theorem 7.1 (Rodrigues' rotation formula)**.** *Given a rotation angle θ and a rotation axis u (expressed by a unit vector). The rotation matrix R can be computed as:*

$$R = \cos(\theta)I_3 + \sin(\theta)[u]_\times + (1-\cos(\theta))uu^T$$

where $[u]_\times = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}$

How can we do it reversely? Intuitively, axis of the rotation is the direction that the rotation preserves. Mathematically speaking, axis of rotation is in the direction of the eigenvector with respect to the eigenvalue of 1. From the

discussion above, if rotation is not equal to identity, then there is a unique direction of rotation axis. So we can solve for the rotation axis by calculating u satisfying:

$$Ru = u$$

To get the rotation angle, we notice that if we take trace of both of the sides of Rodrigues' rotation formula, we can get:

$$\mathrm{Tr}(R) = 2\cos(\theta) + 1$$

Note that if we treat θ as the minimal angle of the rotation, we can always restrict the $\theta \in [0, \pi]$ and thus $\theta = \arccos(\frac{\mathrm{Tr}(R)-1}{2})$. However, then we will leave the rotation direction to the sign of axis u. That is to say: $R(u, \theta)^{-1} = R(-u, \theta)$. So we need to double check the two potential solutions to make sure which rotation is the one we want.

7.1.3.3 Quaternion representation

W.R. Hamilton first introduced the definition of quaternion representation.

Definition 7.2 (Quaternion)**.** A quaternion is represented in the form $\mathbf{q} = \mathbf{i}q_1 + \mathbf{j}q_2 + \mathbf{k}q_3 + q_4$, where q_1, q_2, q_3, q_4 are real numbers and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ satisfying:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1 \tag{7.5}$$

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k} \quad \mathbf{jk} = -\mathbf{kj} = \mathbf{i} \quad \mathbf{ki} = -\mathbf{ik} = \mathbf{j} \tag{7.6}$$

We can also write the quaternion in the column vector form: $\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$. We are particularly interested in **unit quaternions**, which means the column vector has unit length. Unit quaternions can represent rotations, actually we can get quaternions immediately from the axis-angle representation. The main idea is to use the axis-angle representation but in a more compact format. Concretely speaking, given rotation angle θ and rotation axis u(in unit length), the corresponding unit quaternion is as follows:

$$q = \begin{bmatrix} u\sin(\frac{\theta}{2}) \\ \cos(\frac{\theta}{2}) \end{bmatrix}$$

The corresponding rotation matrix is:

$$R(q) = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\ 2(q_1q_2 + q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 - q_1q_4) \\ 2(q_1q_3 - q_2q_4) & 2(q_2q_3 + q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix}$$

Quaternion representation doesn't have singularities. But there is still a little ambiguity that q and $-q$ always represent the same rotation, which means the quaternion is a double cover of the 3D rotation. Actually, quaternions also give us great convenience in calculation, because of its compact representation. For computational use, we care about: **(1)How to compose rotations? (2) How to take inverse for quaternions (3) How to rotate a 3D vector using quaternions?**

(1)How to compose rotations?

Consider two quaternions $q_a = q_{a,1}i + q_{a,2}j + q_{a,3}k + q_{a,4}$, $q_b = q_{b,1}i + q_{b,2}j + q_{b,3}k + q_{b,4}$. Then the composition of the corresponding rotation is just the product of two quaternions. Explicitly,

$$\begin{aligned} q_c = q_a \otimes q_b =& (q_{a,4}q_{b,1} - q_{a,3}q_{b,2} + q_{a,2}q_{b,3} + q_{a,1}q_{b,4})i && (7.7)\\ &+(q_{a,3}q_{b,1} + q_{a,4}q_{b,2} - q_{a,1}q_{b,3} + q_{a,2}q_{b,4})j && (7.8)\\ &+(-q_{a,2}q_{b,1} + q_{a,1}q_{b,2} + q_{a,4}q_{b,3} + q_{a,3}q_{b,4})k && (7.9)\\ &+(-q_{a,1}q_{b,1} - q_{a,2}q_{b,2} - q_{a,3}q_{b,3} + q_{a,4}q_{b,4}) && (7.10) \end{aligned}$$

If we use vector to represent quaternion, we can claim the following formula:

$$q_c = \begin{bmatrix} q_{a,4} & -q_{a,3} & q_{a,2} & q_{a,1} \\ q_{a,3} & q_{a,4} & -q_{a,1} & q_{a,2} \\ -q_{a,2} & q_{a,1} & q_{a,4} & q_{a,3} \\ -q_{a,1} & -q_{a,2} & -q_{a,3} & q_{a,4} \end{bmatrix} \begin{bmatrix} q_{b,1} \\ q_{b,2} \\ q_{b,3} \\ q_{b,4} \end{bmatrix}$$

Similar formula can be derived for q_a.

(2) How to take inverse for quaternions?

Assume that now we have the pose of frame r with respect to a frame w, i.e. we know q_r^w. How can we get the opposite, i.e. the pose of frame w with respect to frame r? The answer is to take the inverse of the rotation. It's easy to carry out using rotation matrix, but how shall we proceed using quaternions? Let's remind ourselves that quaternion is nothing but rearranged axis-angle representation. So naturally the inverse is as follows:

$$q_w^r = \begin{bmatrix} -u_r^w \sin(\frac{\theta}{2}) \\ \cos(\frac{\theta}{2}) \end{bmatrix} = (q_r^w)^{-1}$$

This process is also compatible with the quaternion product.

(3) How to rotate a 3D vector using quaternions?

One key question is, given coordinates in one frame, and the pose of that frame in another frame, how can we translate the coordinates? Assume there is a point p, and we know its coordinates in frame r, denoted as p^r. Also we know the pose of frame r in frame w, denoted as q_r^w. For simplicity, we assume that

the two frame share the same origin. How can we get the coordinates of p with respect to the frame w, i.e. p^w? We have the following formula:

$$p^w = (q_r^w) \otimes \begin{bmatrix} p^r \\ 1 \end{bmatrix} \otimes (q_r^w)^{-1} = \begin{bmatrix} R_r^w p^r \\ 1 \end{bmatrix}$$

i.e. we can compute the rotation by first stack an extra entry 1 at the end of p^r, then let q_r^w act 'conjugately' on it.

7.1.4 Miscellaneous topics on rotations

7.1.4.1 Lie group structure of rotations

Actually, on one hand, the set of rotation matrices ($SO(3)$) is closed under matrix multiplication (and some other properties), which gives rise to its algebraic structure (which is called 'group'). On the other hand, the set of rotation matrices has its own topological structure (which is called 'manifold'). Lie group is both group and smooth manifold, but we'll not introduce Lie group formally.

Geometrically, thanks to the quaternion representation, we can treat the group $SO(3)$ as a sphere in 4D, but due to $R(q) = R(-q)$, we must image there are portals connecting the antipodal points. Locally we can just think it as a sphere. For a sphere, we can imagine that at each point there is a tangent space. The tangent space at the identity is called Lie algebra. Why identity? There is nothing special with identity, just because: (1) Every group has identity. (2) The tangent space at other points can be obtained from identity. Lie algebra is deeply connected with Lie group. Imagine the sphere case, the sphere itself is curved, but we can use coordinates to translate the sphere into a flat map. Lie algebra is a flat space which we can make use of to study about the complicated curved Lie group. We will have some intuition on it by examining $SO(3)$.

Lie group $SO(3)$ is related to its Lie algebra $\mathfrak{so}(3)$. We will state without proof that the Lie algebra $\mathfrak{so}(3)$ is the set of 3×3 skew symmetric matrices.

Exponential Map and Logarithm Map:

Looking at the sphere in the picture, it seems natural for us to bridge the endpoint of the green curved line, which is still on the sphere, with the endpoint of the red line, which is in the tangent space. This retraction is called exponential map. For matrix Lie group (including $SO(3)$), the exponential map coinsides with the matrix exponential (See A.1.)

We will check this on the $SO(3)$ case:

Theorem 7.2 (Matrix exponential for rotations)**.** *The exponential map* $exp{:}\mathfrak{so}(3) \to SO(3)$ *is well-defined.*

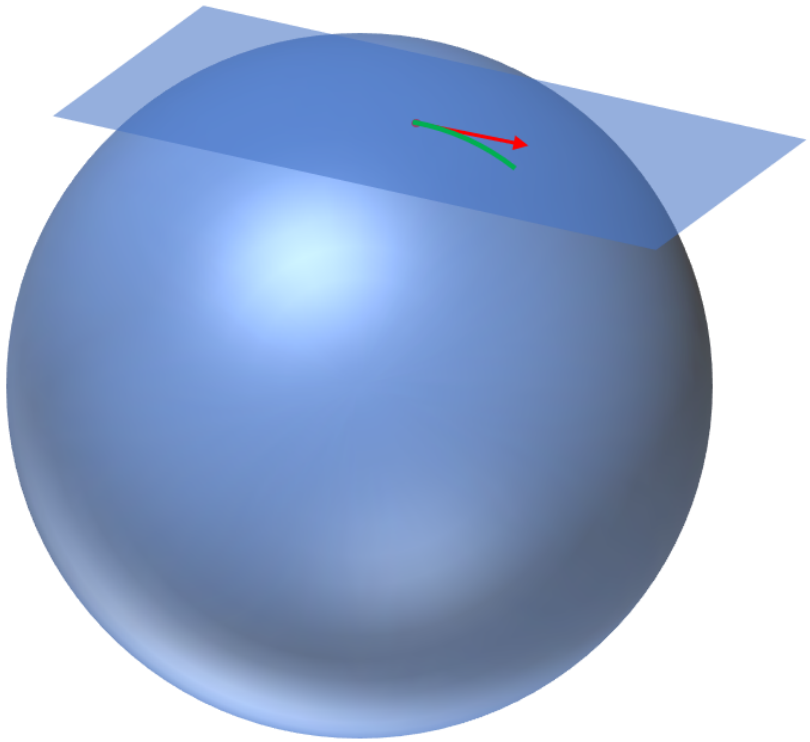

Figure 7.2: Sphere and tangent space

Proof. Consider a 3×3 skew symmetric matrix $A = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix}$, define $w = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \hat{w} = w/\|w\|, \theta = \|w\|$. Then $A = w_\times$. Consider $\hat{w}_\times$, using the properties of the cross product, we have:$\hat{w}_\times^3 = -\hat{w}_\times$. Thus we can know that:

$$\exp(A) = I + (1 - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} + ...)\hat{w}_\times + (\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!})\hat{w}_\times^2 \tag{7.11}$$

$$= I + \frac{\sin\theta}{\theta}A + \frac{1 - \cos\theta}{\theta^2}A^2 \tag{7.12}$$

which actually coincide with the Rodrigues' formula. Thus $\exp(A)$ is a rotation matrix. □

Actually the exponential map for $SO(3)$ is surjective. From the derivation above, I believe we can see the similarity of this with the axis-angle representation. In order to map backwards, we want to recover the angle and the axis from the rotation matrix. There are infinitely many possible points due to the periodic property of trigonometric functions. So we may restrict the rotation angle $\theta \in [0, \pi)$ Then from the derivation of the axis-angle representation, we can get:

$$\theta = \arccos(\frac{\mathrm{Tr}(R) - 1}{2}) \tag{7.13}$$

Next is to recover the axis. Note that the whole space of $n \times n$ matrices is the direct sum of $n \times n$ symmetric matrices and $n \times n$ skew symmetric matrices. So we can decompose the rotation matrix into symmetric and skew symmetric parts. From the formula above, we can decompose as:

$$\exp(A) = \underbrace{\frac{\sin\theta}{\theta}A}_{\text{skew symmetric}} + \underbrace{I + \frac{1 - \cos\theta}{\theta^2}A^2}_{\text{symmetric}}$$

Thus we can recover the skew symmetric matrix A (the axis) by taking the skew symmetric part of $\exp(A)$. Concretely, we can get: $A = \frac{\theta}{2\sin\theta}(R - R^T)$.

Next we will give an example of how to use exponential map and logarithm map to do interpolation of rotations. Suppose we have two rotations R_1, R_2, and we want to find a rotation R that is in between. We can use the following formula:

$$R = R_1 \exp(\lambda \log(R_1^T R_2))$$

where $\lambda \in [0, 1]$.

7.2 The Pinhole Camera Model

Next let's take a look at the pinhole camera model, which is the simplest one. In this model, the picture in 2D is the projection of the corresponding 3D points with respect to the optical center.

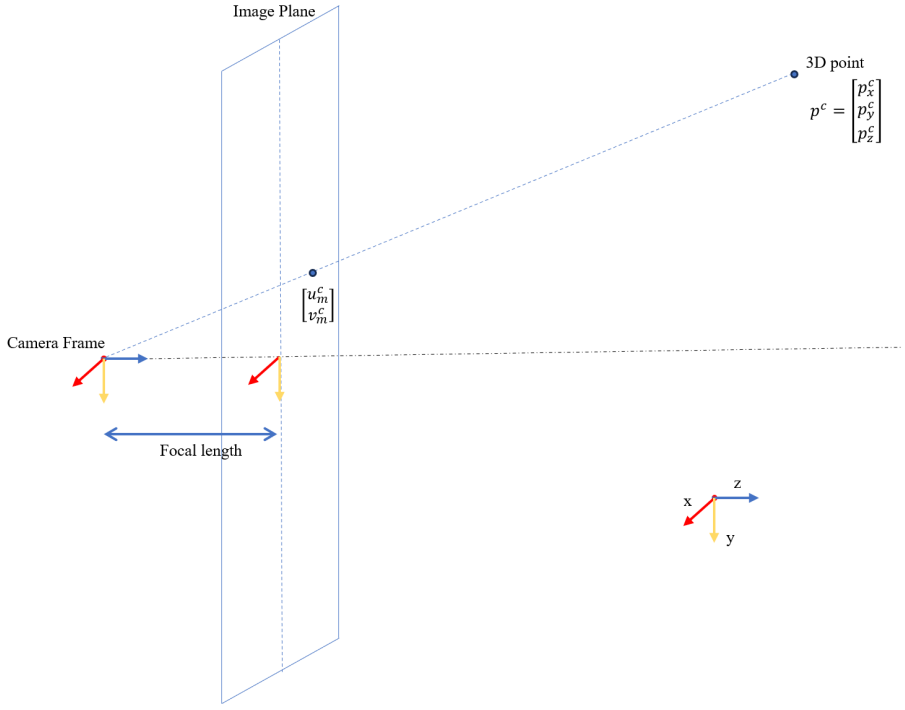

Figure 7.3: Pinhole Camera Model

Consider point p in the camera frame. Suppose the focal length(i.e. the distance from the optical center to the image plane) is f. Then the coordinates of the projection of p is:

$$p_m^c = \begin{bmatrix} u_m^c \\ v_m^c \end{bmatrix} = \begin{bmatrix} f\frac{p_x^c}{p_z^c} \\ f\frac{p_y^c}{p_z^c} \end{bmatrix}$$

The unit of these coordinates are meter. We would prefer to write the formula into a matrix form, but the barrier here is the quotient. In order to overcome this, we introduce **homogeneous coordinates**. In homogeneous coordinates system, if coordinates are multiplied by a non-zero scalar then the resulting coordinates represent the same point. We will add an extra entry to better

represent this equivalence. Denote

$$\tilde{p^c} = \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \\ 1 \end{bmatrix}$$

Then the equivalence can be expressed as

$$\forall k \neq 0, \quad \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \\ 1 \end{bmatrix} \sim \begin{bmatrix} kp_x^c \\ kp_y^c \\ kp_z^c \\ k \end{bmatrix}$$

With this notation, the projection equation can be written as:

$$p_z^c \begin{bmatrix} u_m^c \\ v_m^c \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \\ 1 \end{bmatrix}$$

Normally when we deal with points in a 2D picture, we will prefer the unit is in pixels instead of meters. Next we will show how to convert the coordinates into pixels. In convention, the origin of the pixel coordinates is at the top-left of the image. So that we can express the coordinates in pixel as follows:

$$\begin{bmatrix} u^I \\ v^I \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_m^c \\ v_m^c \\ 1 \end{bmatrix}$$

where s_x is the number of horizontal pixels per meter, s_y is the number of vertical pixels per meter, and $\begin{bmatrix} o_x \\ o_y \end{bmatrix}$ is the coordinates of the optical center.

Combing the previous result, we can get:

$$p_z^c \begin{bmatrix} u^I \\ v^I \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \\ 1 \end{bmatrix} \tag{7.14}$$

$$= \underbrace{\begin{bmatrix} s_x f & 0 & o_x \\ 0 & s_y f & o_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_x^c \\ p_y^c \\ p_z^c \\ 1 \end{bmatrix} \tag{7.15}$$

where K is called intrinsic matrix.

Notice that, in our derivation above, the homogeneous coordinates of p is in the camera frame. However, it's not always easy to know the coordinates in

the camera frame. For example, sometimes we may have many different camera views, or the camera itself is moving. What we usually have is the coordinate of the 3D points in the world frame. Therefore we will add a procedure to transform the coordinates from world fram to the camera frame. From the previous section, we can know that we just need to do some rotations and translations. Therefore we can obtain the formula:

$$p_z^c \begin{bmatrix} u^I \\ v^I \\ 1 \end{bmatrix} = \begin{bmatrix} s_x f & 0 & o_x \\ 0 & s_y f & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_w^c & t_w^c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_x^w \\ p_y^w \\ p_z^w \\ 1 \end{bmatrix} \tag{7.16}$$

$$= \begin{bmatrix} s_x f & 0 & o_x \\ 0 & s_y f & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_w^c & t_w^c \end{bmatrix} \begin{bmatrix} p_x^w \\ p_y^w \\ p_z^w \\ 1 \end{bmatrix} \tag{7.17}$$

where the matrix $\begin{bmatrix} R_w^c & t_w^c \end{bmatrix}$ is called the extrinsic matrix.

Some pinhole cameras will introduce significant distortion to images. For cameras with wide field of view (FOV), they often suffers from radial distortion. Radial distortion makes straight lines look curved in the image. The farther points are from the center of the image, the larger the radial distortion will be. Radial distortion is primarily dominated by low-order components. An easy model of the distortion (in camera frame) is:

$$u^c = (1 + K_1 r^2 + K_2 r^4) u^c_{\text{distort}} \qquad v^c = (1 + K_1 r^2 + K_2 r^4) v^c_{\text{distort}}$$

where (u^c, v^c) is the undistorted image point, $(u^c_{\text{distort}}, v^c_{\text{distort}})$ is the distorted image point, $r^2 = (u^c_{\text{distort}})^2 + (v^c_{\text{distort}})^2$, and the K_n is the n-th distortion coefficient.

If we want to express in the image frame, we can obtain the following model:

$$u^I = (1+K_1 r^2+K_2 r^4)(u^I_{\text{distort}} - o_x) + o_x \qquad v^I = (1+K_1 r^2+K_2 r^4)(v^I_{\text{distort}} - o_y) + o_y$$

where $r^2 = (u^I_{\text{distort}} - o_x)^2 + (v^I_{\text{distort}} - o_y)^2$.

It's worth to mention that despite the radial distortion, there is another distortion called tangential distortion, which will not be elaborated here.

We often assume that the intrinsics of the camera and he distortion coefficients are already known from the previous camera calibration process, i.e. we always assume that the camera is calibrated.

There are many existing codes and toolbox by hand if we want to undistort the pictures. Such as OpenCV has undistort function, and Matlab has undistortImage function. They can take in distorted images and return the undistorted ones.

7.3 Camera Pose Estimation

Suppose now we have a calibrated camera, but we don't know the rotation and translation, i.e. the pose of the camera. We wish to find the extrinsics. Now what we have is a set of n 3D points and their corresponding 2D image projections. We wish to estimate the pose of the camera from these corresponding points.

7.3.1 The P3P Problem

Let's start from something simple. Consider there are three points P_1^w, P_2^w, P_3^w in the world frame, and their corresponding projection points on the image plane $P_{m,1}^c, P_{m,2}^c, P_{m,3}^c$. The problem is: Can we found the coordinates of the points P_1, P_2, P_3 in the camera frame? We will introduce a direct way to tackle the problem, called Grunert's method.

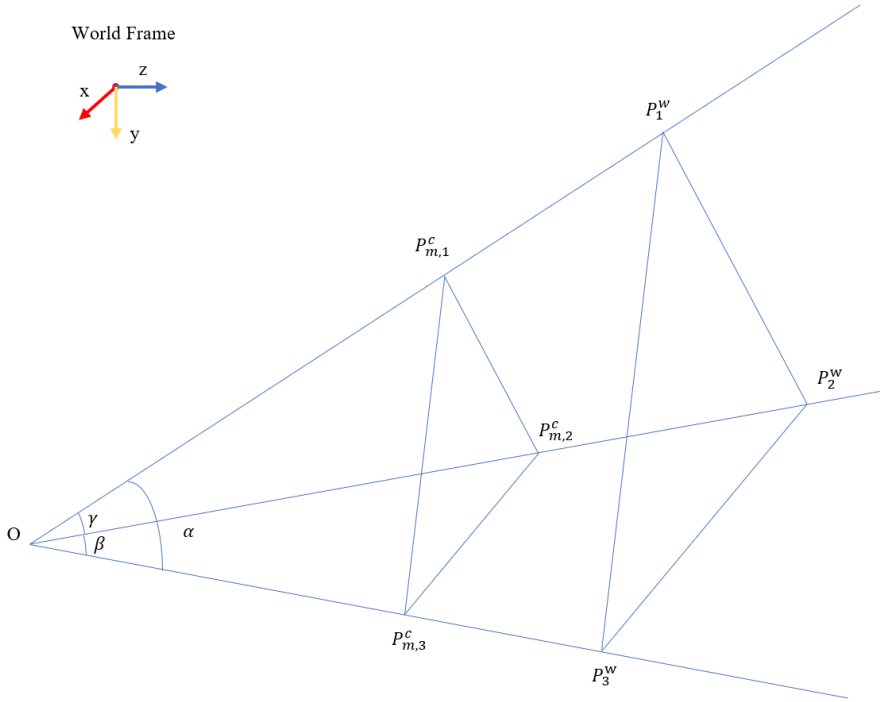

Figure 7.4: P3P problem

From the coordinates of $P_{m,1}^c, P_{m,2}^c, P_{m,3}^c$ in the image plane, we can calculate the angles between P_1, P_2, P_3. Denote the angle between P_1, P_2 as γ, the angle be-

tween P_2, P_3 as β, and the angle between P_1, P_3 as α. Remember that if $P^c_{m,i} = \begin{bmatrix} u^c_{m,i} \\ v^c_{m,i} \\ 1 \end{bmatrix}$, then the camera frame coordinates $P^c_i = \frac{s_i}{\sqrt{1+(u^c_{m,i})^2+(v^c_{m,i})^2}} \begin{bmatrix} u^c_{m,i} \\ v^c_{m,i} \\ 1 \end{bmatrix}$. So what we have to do is to solve for s_i, which is the distance from the origin to the point P_i.

Using law of cosines, we can obtain:

$$s_1^2 + s_2^2 - 2s_1 s_2 \cos\gamma = c^2 \tag{7.18}$$
$$s_1^2 + s_3^2 - 2s_1 s_3 \cos\beta = b^2 \tag{7.19}$$
$$s_2^2 + s_3^2 - 2s_2 s_3 \cos\alpha = a^2 \tag{7.20}$$
$$\tag{7.21}$$

Then if we let $s_2 = us_1, s_3 = vs_1$, we can obtain:

$$s_1^2 = \frac{c^2}{1 + u^2 - 2u\cos\gamma} \tag{7.22}$$
$$s_1^2 = \frac{b^2}{1 + v^2 - 2v\cos\beta} \tag{7.23}$$
$$s_1^2 = \frac{a^2}{v^2 + u^2 - 2uv\cos\alpha} \tag{7.24}$$
$$\tag{7.25}$$

Then we can get:

$$u^2 - \frac{c^2}{b^2}v^2 + 2v\frac{c^2}{b^2}\cos\beta - 2u\cos\gamma + \frac{b^2 - c^2}{b^2} = 0$$

and

$$u^2 + \frac{b^2 - a^2}{b^2}v^2 - 2uv\cos\alpha + \frac{2a^2}{b^2}v\cos\beta - \frac{a^2}{b^2} = 0$$

Then substituting the later equation back to the former one, we can express u in terms of v. Then plug this into the latter equality, we can finally get an equation:

$$A_4 v^4 + A_3 v^3 + A_2 v^2 + A_1 v + A_0 = 0$$

where

$$A_4 = \left(\frac{a^2-c^2}{b^2} - 1\right)^2 - \frac{4c^2}{b^2}\cos^2\alpha \tag{7.26}$$

$$A_3 = 4\left[\frac{a^2-c^2}{b^2}\left(1 - \frac{a^2-c^2}{b^2}\right)\cos\beta - \left(1 - \frac{a^2+c^2}{b^2}\right)\cos\alpha\cos\gamma + 2\frac{c^2}{b^2}\cos^2\alpha\cos\beta\right] \tag{7.27}$$

$$A_2 = 2[\left(\frac{a^2-c^2}{b^2}\right)^2 - 1 + 2\left(\frac{a^2-c^2}{b^2}\right)^2\cos^2\beta + 2\left(\frac{b^2-c^2}{b^2}\right)\cos^2\alpha \tag{7.28}$$

$$-4\left(\frac{a^2+c^2}{b^2}\right)\cos\alpha\cos\beta\cos\gamma + 2\left(\frac{b^2-a^2}{b^2}\right)\cos^2\gamma] \tag{7.29}$$

$$A_1 = 4\left[-\left(\frac{a^2-c^2}{b^2}\right)\left(1 + \frac{a^2-c^2}{b^2}\right)\cos\beta + \frac{2a^2}{b^2}\cos^2\gamma\cos\beta - \left(1 - \left(\frac{a^2+c^2}{b^2}\right)\right)\cos\alpha\cos\gamma\right] \tag{7.30}$$

$$A_0 = \left(1 + \frac{a^2-c^2}{b^2}\right)^2 - \frac{4a^2}{b^2}\cos^2\gamma \tag{7.31}$$

From this equation, we can obtain up to four solutions. With each v we can obtain the corresponding u and thus all the other parameters. To eliminate the ambiguity, we can use a 4th point to confirm which one is the right solution.

7.3.2 The PnP Problem

7.3.3 Global Optimality

7.3.4 Handling Outliers

7.4 Point Cloud Registration

Chapter 8

Adaptive Control

8.1 Model-Reference Adaptive Control

Basic flow for designing an adaptive controller

1. Design a control law with variable parameters
2. Design an adaptation law for adjusting the control parameters
3. Analyze the convergence of the closed-loop system

The control law design at the first step typically requires the designer to know what a good controller is if the true parameters were actually known, e.g., from feedback linearization (Appendix F), sliding control (Appendix G) etc.

The design of the adaptation law typically comes from analyzing the dynamics of the tracking error, which as we will see often appears in the form of Lemma 8.1.

The convergence of the closed-loop system is usually analyzed with the help of a Lyapunov-like function introduced in Chapter 5.

Lemma 8.1 (Basic Lemma)**.** *Let two signals $e(t)$ and $\phi(t)$ be related by*

$$e(t) = H(p)[k\phi(t)^T v(t)] \tag{8.1}$$

where $e(t)$ a scalar output signal, $H(p)$ a strictly positive real (SPR) transfer function, k an unknown real number with known sign, $\phi(t) \in \mathbb{R}^m$ a control signal, and $v(t) \in \mathbb{R}^m$ a measurable input signal.

If the control signal $\phi(t)$ satisfies

$$\dot{\phi}(t) = -\text{sgn}(k)\gamma e(t)v(t) \tag{8.2}$$

with $\gamma > 0$ a positive constant, then $e(t)$ and $\phi(t)$ are globally bounded. Moreover, if $v(t)$ is bounded, then

$$\lim_{t\to\infty} e(t) = 0.$$

Proof. Let the state-space representation of (8.1) be

$$\dot{x} = Ax + b[k\phi^T v], \quad e = c^T x. \tag{8.3}$$

Since $H(p)$ is SPR, it follows from the Kalman-Yakubovich Lemma E.1 that there exist $P, Q \succ 0$ such that

$$A^T P + PA = -Q, \quad Pb = c.$$

Let

$$V(x, \phi) = x^T P x + \frac{|k|}{\gamma}\phi^T\phi,$$

clearly V is positive definite (i.e., $V(0,0) = 0$, and $V(x,\phi) > 0$ for all $x \neq 0, \phi \neq 0$). The time derivative of V along the trajectory defined by (8.3) with ϕ chosen as in (8.2) is

$$\begin{aligned}
\dot{V} &= \frac{\partial V}{\partial x}\dot{x} + \frac{\partial V}{\partial \phi}\dot{\phi} && (8.4)\\
&= x^T(PA + A^T P)x + 2x^T Pb(k\phi^T v) + \frac{2|k|}{\gamma}\phi^T(-\mathrm{sgn}(k)\gamma e v) && (8.5)\\
&= -x^T Q x + 2(x^T c)(k\phi^T v) - 2\phi^T(ev) && (8.6)\\
&= -x^T Q x \leq 0. && (8.7)
\end{aligned}$$

As a result, we know x and ϕ must be bounded ($V(x(t), \phi(t)) \leq V(x(0), \phi(0))$ is bounded). Since $e = c^T x$, we know e must be bounded as well.

If the input signal v is also bounded, then $\dot{x}$ is bounded as seen from (8.3). Because $\ddot{V} = -2x^T Q\dot{x}$ is now bounded, we know $\dot{V}$ is uniformly continuous. Therefore, by Barbalat's stability certificate (Theorem 5.6), we know $\dot{V}$ tends to zero as t tends to infinity, which implies $\lim_{t\to\infty} x(t) = 0$ and hence $\lim_{t\to\infty} e(t) = 0$. □

8.1.1 First-Order Systems

Consider the first-order single-input single-output (SISO) system

$$\dot{x} = -ax + bu \tag{8.8}$$

where a and b are unknown groundtruth parameters. However, we do assume that the sign of b is known. What if the sign of b is unknown too?

Let $r(t)$ be a reference trajectory, e.g., a step function or a sinusoidal function, and $x_d(t)$ be a desired system trajectory that tracks the reference

$$\dot{x}_d = -a_d x_d + b_d r(t), \tag{8.9}$$

where $a_d, b_d > 0$ are user-defined constants. Note that the transfer function from r to x_d is

$$x_d = \frac{b_d}{p + a_d} r$$

and the system is stable. Review basics of transfer function.

The goal of adaptive control is to design a control law and an adaptation law such that the tracking error of the system $x(t) - x_d(t)$ converges to zero.

Control law. We design the control law as

$$u = \hat{a}_r(t) r + \hat{a}_x(t) x \tag{8.10}$$

where $\hat{a}_r(t)$ and $\hat{a}_x(t)$ are time-varying feedback gains that we wish to adapt. The closed-loop dynamics of system (8.8) with the controller (8.10) is

$$\dot{x} = -ax + b(\hat{a}_r r + \hat{a}_x x) = -(a - b\hat{a}_x)x + b\hat{a}_r r.$$

With the equation above, the reason for choosing the control law (8.10) is clear: if the system parameters (a, b) were known, then choosing

$$a_r^\star = \frac{b_d}{b}, \quad a_x^\star = \frac{a - a_d}{b} \tag{8.11}$$

leads to the closed-loop dynamics $\dot{x} = -a_d x + b_d r$ that is exactly what we want in (8.9).

However, in adaptive control, since the true parameters (a, b) are not revealed to the control designer, an adaptation law is needed to dynamically adjust the gains $\hat{a}_r$ and $\hat{a}_x$ based on the tracking error $x(t) - x_d(t)$.

Adaptation law. Let $e(t) = x(t) - x_d(t)$ be the tracking error, and we develop its time derivative

$$\begin{aligned}
\dot{e} &= \dot{x} - \dot{x}_d && (8.12)\\
&= -a_d(x - x_d) + (a_d - a + b\hat{a}_x)x + (b\hat{a}_r - b_d)r && (8.13)\\
&= -a_d e + b\underbrace{(\hat{a}_x - \hat{a}_x^\star)}_{=:\tilde{a}_x} x + b\underbrace{(\hat{a}_r - \hat{a}_r^\star)}_{=:\tilde{a}_r} r && (8.14)\\
&= -a_d e + b(\tilde{a}_x x + \tilde{a}_r r) && (8.15)
\end{aligned}$$

where $\tilde{a}_x$ and $\tilde{a}_r$ are the gain errors w.r.t. the optimal gains in (8.11) if the true parameters were known. The error dynamics (8.15) is equivalent to the following transfer function

$$e = \frac{1}{p + a_d} b(\tilde{a}_x x + \tilde{a}_r r) = \frac{1}{p + a_d} \left(b \begin{bmatrix} \tilde{a}_x \\ \tilde{a}_r \end{bmatrix}^T \begin{bmatrix} x \\ r \end{bmatrix} \right), \tag{8.16}$$

which is in the form of (8.1). Therefore, we choose the adaptation law

$$\begin{bmatrix} \dot{\tilde{a}}_x \\ \dot{\tilde{a}}_r \end{bmatrix} = -\text{sgn}(b)\gamma e \begin{bmatrix} x \\ r \end{bmatrix}. \tag{8.17}$$

Tracking convergence. With the control law (8.10) and the adaptation law (8.17), we can prove that the tracking error converges to zero, using Lemma 8.1. With $\tilde{a} = [\tilde{a}_x, \tilde{a}_r]^T$, let

$$V(e, \tilde{a}) = e^2 + \frac{|b|}{\gamma}\tilde{a}^T\tilde{a} \tag{8.18}$$

be a positive definite Lyapunov function candidate with time derivative

$$\dot{V} = -2a_d e^2 \leq 0.$$

Clearly, e and $\tilde{a}$ are both bounded. Assuming the reference trajectory r is bounded, we know x_d is bounded (due to (8.9)) and hence x is bounded (due to $e = x - x_d$ being bounded). Consequently, from the error dynamics (8.15) we know $\dot{e}$ is bounded, which implies $\ddot{V} = -4a_d e\dot{e}$ is bounded and $\dot{V}$ is uniformly continuous. By Barbalat's stability certificate 5.6, we conlude $e(t) \to 0$ as $t \to \infty$.

It is always better to combine mathematical analysis with intuitive understanding. Can you explain intuitively why the adaptation law (8.17) makes sense? (Hint: think about how the control should react to a negative/positive tracking error.)

Parameter convergence. We have shown the control law (8.10) and the adaptation law (8.17) guarantee to track the reference trajectory. However, is it guaranteed that the gains of the controller (8.10) also converge to the optimal gains in (8.11)?

We will now show that the answer is indefinite and it depends on the reference trajectory $r(t)$. Because the tracking error e converges to zero, and e is the output of a stable filter (8.16), we know the input $b(\tilde{a}_x x + \tilde{a}_r r)$ must also converge to zero. On the other hand, the adaptation law (8.17) shows that both $\dot{\tilde{a}}_x$ and $\dot{\tilde{a}}_r$ converge to zero (due to e converging to zero and x, r being bounded). As a result, we know $\tilde{a} = [\tilde{a}_x, \tilde{a}_r]^T$ converges to a constant that satisfies

$$v^T\tilde{a} = 0, \quad v = \begin{bmatrix} x \\ r \end{bmatrix}, \tag{8.19}$$

which is a single linear equation of $\tilde{a}$ with time-varying coeffients.

- **Constant reference: no guaranteed convergence**. Suppose $r(t) \equiv r_0 \neq 0$ for all t. From (8.9) we know $x = x_d = \alpha r_0$ when $t \to \infty$, where α is the constant DC gain of the stable filter. Therefore, the linear equation (8.19) reduces to

$$\alpha\tilde{a}_x + \tilde{a}_r = 0.$$

This implies that $\tilde{a}$ does not necessarily converge to zero. In fact, it converges to a straight line in the parameter space.

- **Persistent excitation: guaranteed convergence**. However, when the signal v satisfies the so-called *persistent excitation* condition, which states that for any t, there exists $T, \beta > 0$ such that

$$\int_t^{t+T} vv^T d\tau \geq \beta I, \tag{8.20}$$

then $\tilde{a}$ is guaranteed to converge to zero. To see this, we multiply (8.19) by v and integrate it from t to $t+T$, which gives rise to

$$\left(\int_t^{t+T} vv^T d\tau \right) \tilde{a} = 0.$$

By the persistent excitation condition (8.20), we infer that $\tilde{a} = 0$ is the only solution.

It remains to understand under what conditions of the reference trajectory $r(t)$ can we guarantee the persistent excitation of v. We leave it as an exercise for the reader to show, if $r(t)$ contains at least one sinusoidal component, then the persistent excitation condition of v is guaranteed.

Exercise 8.1 (Extension to Nonlinear Systems)**.** Design a control law and an adaptation law for the following system

$$\dot{x} = -ax - cf(x) + bu$$

with unknown true parameters (a, b, c) (assume the sign of b is known) and known nonlinearity $f(x)$ to track a reference trajectory $r(t)$. Analyze the convergence of tracking error and parameter estimation error.

8.1.2 High-Order Systems

Consider an n-th order nonlinear system

$$q^{(n)} + \sum_{i=1}^{n} \alpha_i f_i(x, t) = bu \tag{8.21}$$

where $x = [q, \dot{q}, \ddot{q}, \dots, q^{(n-1)}]^T$ is the state of the system, f_i's are known nonlinearities, $(\alpha_1, \dots, \alpha_n, b)$ are unknown parameters of the system (with $\text{sgn}(b)$ known).

The goal of adaptive control is to control the system (8.21) trajectory to follow a desired trajectory $q_d(t)$ despite no knowing the true parameters.

To facilitate the derivation of the adaptive controller, let us divide both sides of (8.21) by b

$$hq^{(n)} + \sum_{i=1}^{n} a_i f_i(x,t) = u \tag{8.22}$$

where $h = 1/b$ and $a_i = \alpha_i/b$.

Control law. Recall that the choice of the control law is tyically inspired by the control design if the true system parameters were known. We will borrow ideas from sliding control (Appendix G).

- **Known parameters**. Let $e = q(t) - q_d(t)$ be the tracking error, and define the following combined error

$$s = e^{(n-1)} + \lambda_{n-2} e^{(n-2)} + \cdots + \lambda_0 e = \Delta(p) e$$

where $\Delta(p) = p^{n-1} + \lambda_{n-2} p^{(n-2)} + \cdots + \lambda_0$ is a stable polynomial with user-chosen coeffients $\lambda_0, \ldots, \lambda_{n-2}$. The rationale for defining the combined error s is that the convergence of e to zero can be guaranteed by the convergence of s to zero (when $\Delta(p)$ is stable). Note that s can be equivalently written as

$$s = (q^{(n-1)} - q_d^{(n-1)}) + \lambda_{n-2} e^{(n-2)} + \cdots + \lambda_0 e \tag{8.23}$$

$$= q^{(n-1)} - \underbrace{\left(q_d^{(n-1)} - \lambda_{n-2} e^{(n-2)} - \cdots - \lambda_0 e\right)}_{q_r^{(n-1)}}. \tag{8.24}$$

Now consider the control law

$$u = h q_r^{(n)} - ks + \sum_{i=1}^{n} a_i f_i(x,t) \tag{8.25}$$

where

$$q_r^{(n)} = q_d^{(n)} - \lambda_{n-2} e^{(n-1)} - \cdots - \lambda_0 \dot{e}$$

and k is a design constant that has the same sign as h. This choice of control, plugged into the system dynamics (8.22), leads to

$$hq^{(n)} + \sum_{i=1}^{n} a_i f_i(x,t) = h q_r^{(n)} - ks + \sum_{i=1}^{n} a_i f_i(x,t) \iff \tag{8.26}$$

$$h\left(q^{(n)} - q_r^{(n)}\right) + ks = 0 \iff \tag{8.27}$$

$$h\dot{s} + ks = 0, \tag{8.28}$$

which guarantees the exponential convergence of s to zero (note that h and k have the same sign), and hence the convergence of e to zero.

- **Unknown parameters**. Inspired by the control law with known parameters in (8.25), we design the adapative control law as

$$u = \hat{h} q_r^{(n)} - ks + \sum_{i=1}^{n} \hat{a}_i f_i(x,t), \tag{8.29}$$

where the time-varying gains $\hat{h}, \hat{a}_1, \dots, \hat{a}_n$ will be adjusted by an adaptation law.

Adaptation law. Inserting the adapative control law (8.29) into the system dynamics (8.22), we obtain

$$h\dot{s} + ks = \tilde{h} q_r^{(n)} + \sum_{i=1}^{n} \tilde{a}_i f_i(x,t) \Longleftrightarrow \tag{8.30}$$

$$s = \frac{1}{p + k/h} \frac{1}{h} \underbrace{\left(\begin{bmatrix} \tilde{h} \\ \tilde{a}_1 \\ \vdots \\ \tilde{a}_n \end{bmatrix}^T \begin{bmatrix} q_r^{(n)} \\ f_1(x,t) \\ \vdots \\ f_n(x,t) \end{bmatrix} \right)}_{=:\phi^T v} \tag{8.31}$$

where $\tilde{h} = \hat{h} - h$ and $\tilde{a}_i = \hat{a}_i - a_i, i = 1, \dots, n$. Again, (8.31) is in the familiar form of (8.1), which naturally leads to the following adaptation law with $\gamma > 0$ a chosen constant

$$\dot{\phi} = \begin{bmatrix} \dot{\tilde{h}} \\ \dot{\tilde{a}}_1 \\ \vdots \\ \dot{\tilde{a}}_n \end{bmatrix} = -\mathrm{sgn}(h) \gamma s \begin{bmatrix} q_r^{(n)} \\ f_1(x,t) \\ \vdots \\ f_n(x,t) \end{bmatrix}. \tag{8.32}$$

Tracking and parameter convergence. With the following Lyapunov function

$$V(s,\phi) = |h| s^2 + \frac{1}{\gamma} \phi^T \phi, \quad \dot{V}(s,\phi) - 2|k| s^2, \tag{8.33}$$

the global convergence of s to zero can be easily shown. For parameter convergence, it is easy to see that when v satisfies the persistent excitation condition, we have that ϕ converges to zero. (However, the relationship between the reference trajectory $q_d(t)$ and the persistent excitation of v becomes nontrivial due to the nonlinearities f_i.)

8.1.3 Robotic Manipulator

So far our focus has been on systems with a single input ($u \in \mathbb{R}$). In the following, we will show that similar techniques can be applied to adapative

control of systems with multiple inputs, particularly, trajectory control of a robotic manipulator.

Let $q \in \mathbb{R}^n$ be the joint angles of a multi-link robotic arm, and $\dot{q} \in \mathbb{R}^n$ be the joint velocities. The dynamics of a robotic manipulator reads

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau, \tag{8.34}$$

where $H(q) \in \mathbb{S}^n_{++}$ is the manipulator inertia matrix (that is positive definite), $C(q,\dot{q})\dot{q}$ is a vector of centripetal and Coriolis torques (with $C(q,\dot{q}) \in \mathbb{R}^{n\times n}$), and $g(q)$ denotes gravitational torques.

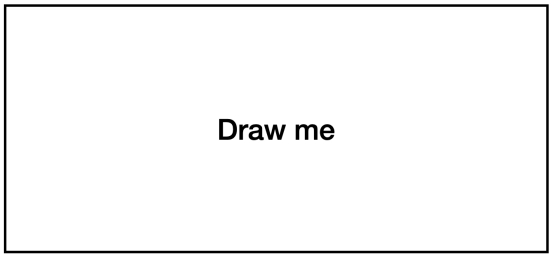

Figure 8.1: Planar two-link manipulator

Example 8.1 (Planar Two-link Manipulator)**.** The dynamics of a planar two-link manipulator in Fig. 8.1 is

$$\begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} -h\dot{q}_2 & -h(\dot{q}_1+\dot{q}_2) \\ h\dot{q}_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}, \tag{8.35}$$

where

$$H_{11} = a_1 + 2a_3\cos q_2 + 2a_4 \sin q_2 \tag{8.36}$$
$$H_{12} = H_{21} = a_2 + a_3 \cos q_2 + a_4 \sin q_2 \tag{8.37}$$
$$H_{22} = a_2 \tag{8.38}$$
$$h = a_3 \sin q_2 - a_4 \cos q_2 \tag{8.39}$$

with

$$a_1 = I_1 + m_1 l_{c1}^2 + I_e + m_e l_{ce}^2 + m_e l_1^2 \tag{8.40}$$
$$a_2 = I_e + m_e l_{ce}^2 \tag{8.41}$$
$$a_3 = m_e l_1 l_{ce} \cos\delta_e \tag{8.42}$$
$$a_4 = m_e l_1 l_{ce} \sin\delta_e. \tag{8.43}$$

As seen from the above example, the parameters a (which are nonlinear functions of the physical parameters such as mass and length) enter linearly in H and C ($g(q)$ is ignored because the manipulator is on a horizontal plane).

The goal of the control design is to have the manipulator track a desired trajectory $q_d(t)$.

Known parameters. When the parameters are known, we follow the sliding control design framework. Let $\tilde{q} = q(t) - q_d(t)$ be the tracking error, and define the combined error

$$s = \dot{\tilde{q}} + \Lambda\tilde{q} = \dot{q} - \underbrace{(\dot{q}_d - \Lambda\tilde{q})}_{\dot{q}_r}$$

where $\Lambda \in \mathbb{S}^n_{++}$ is a user-chosen positive definite matrix (in general we want $-\Lambda$ to be Hurwitz). In this case, $s \to 0$ implies $\tilde{q} \to 0$ as $t \to \infty$. Choosing the control law (coming from feedback linearization Appendix F)

$$\tau = H\ddot{q}_r - K_D s + C\dot{q} + g(q) \tag{8.44}$$

with $K_D \in \mathbb{S}^n_{++}$ positive definite leads to the closed-loop dynamics

$$H\dot{s} + K_D s = 0 \iff \dot{s} = -H^{-1}K_D s.$$

Because the matrix $H^{-1}K_D$ is the product of two positive definite matrices (recall H is positive definite and so is H^{-1}), it has strictly positive real eigenvalues.[1] Hence, $-H^{-1}K_D$ is Hurwitz and s is guaranteed to converge to zero.

Control law. A closer look at the controller (8.44) allows us to write it in the following form

$$\begin{aligned}
\tau &= H\ddot{q}_r + C(s + \dot{q}_r) + g(q) - K_D s && (8.45)\\
&= H\ddot{q}_r + C\dot{q}_r + g(q) + (C - K_D)s && (8.46)\\
&= Y(q, \dot{q}, \dot{q}_r, \ddot{q}_r)a + (C - K_D)s && (8.47)
\end{aligned}$$

where $a \in \mathbb{R}^m$ contains all the parameters and $Y \in \mathbb{R}^{n\times m}$ is the matrix that collects all the coeffients of a in $H\ddot{q}_r + C\dot{q}_r + g(q)$. As a result, we design the adapative control law to be

$$\tau = Y\hat{a} - K_D s, \tag{8.48}$$

with $\hat{a}$ the time-varying parameter that we wish to adapt. Note that here we have done something strange: the adapative control law does not exactly follow the controller (8.44) in the known-parameter case.[2] We first separated s from $\dot{q}$ and wrote $Ya = H\ddot{q}_r + C\dot{q}_r + g$ instead of $Ya = H\ddot{q}_r + C\dot{q} + g$; then we dropped the "$C$" matrix in front of s in the adapative control law. The reason for doing this will soon become clear when we analyze the tracking convergence.

[1] Consider two positive definite matrices A and B, let $B = B^{1/2}B^{1/2}$. The product AB can be written as $AB = AB^{1/2}B^{1/2} = B^{-1/2}(B^{1/2}AB^{1/2})B^{1/2}$. Therefore AB is similar to $B^{1/2}AB^{1/2}$ and is positive definite.

[2] In fact, one can show that the controller (8.48) with known parameters, i.e., $\tau = Ya - K_D s$, also guarantees the convergence of s towards zero, though it is different from the feedback linearization controller (8.44). Try proving the convergence with a Lyapunov candidate $V = \frac{1}{2}s^T Hs$.

Adaptation law and tracking convergence. Recall that the key of adapative control is to design a control law and an adaptation law such that global converge of the tracking error s can be guaranteed by a Lyapunov function. Looking at the previous Lyapunov functions in (8.18) and (8.33), we see that they both contain a positive definite term in the tracking error s (or e if in first-order systems) and another positive definite term in the parameter error $\tilde{a}$. This hints us that we may try a Lyapunov candidate function of the following form

$$V = \frac{1}{2}\left(s^T H s + \tilde{a}\Gamma^{-1}\tilde{a}\right), \tag{8.49}$$

where $\Gamma \in \mathbb{S}^m_{++}$ is a constant positive definite matrix, and $\tilde{a} = \hat{a} - a$ is the parameter error.

The next step would be to derive the time derivative of V, which, as we can expect, will contain a term that involves $\dot{H}$ and complicates our analysis. Fortunately, the following lemma will help us.

Lemma 8.2. *For the manipulator dynamics* (8.34)*, there exists a way to define* C *such that* $\dot{H} - 2C$ *is skew-symmetric.*

Proof. See Section 9.1, page 399-402 in (Slotine et al., 1991). You should also check if this is true for the planar two-link manipulator dynamics in Example 8.1. □

With Lemma 8.2, the time derivative of V in (8.49) reads

$$\begin{aligned}
\dot{V} &= s^T H\dot{s} + \frac{1}{2}s^T\dot{H}s + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}} && (8.50)\\
&= s^T(H\ddot{q} - H\ddot{q}_r) + \frac{1}{2}s^T\dot{H}s + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}} && (8.51)\\
&= s^T(\tau - C\dot{q} - g - H\ddot{q}_r) + \frac{1}{2}s^T\dot{H}s + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}} && (8.52)\\
&= s^T(\tau - H\ddot{q}_r - C(s + \dot{q}_r) - g) + \frac{1}{2}s^T\dot{H}s + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}} && (8.53)\\
&= s^T(\tau - H\ddot{q}_r - C\dot{q}_r - g) + \frac{1}{2}s^T(\dot{H} - 2C)s + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}} && (8.54)\\
&= s^T(\tau - H\ddot{q}_r - C\dot{q}_r - g) + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}} && (8.55)\\
&= s^T(Y\hat{a} - K_D s - Ya) + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}} && (8.56)\\
&= s^T Y\tilde{a} + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}} - s^T K_D s, && (8.57)
\end{aligned}$$

where we used the manipulator dynamics (8.34) to rewrite $H\ddot{q}$ in (8.52), used $\dot{H} - 2C$ is skew-symmetric in (8.54), invoked the adapative control law (8.48) and reused $Ya = H\ddot{q}_r + C\dot{q}_r + g(q)$ in (8.56). The derivation above explains why the choice of the control law in (8.48) did not exactly follow its counterpart when the parameters are known: we need to use $s^T Cs$ to cancel $\frac{1}{2}s^T\dot{H}s$ in (8.54).

We then wonder if we can design $\dot{\tilde{a}}$ such that $\dot{V}$ in (8.57) is negative semidefinie? This turns out to be straightforward with the adaptation law

$$\dot{\tilde{a}} = -\Gamma Y^T s, \tag{8.58}$$

to make $s^T Y\tilde{a} + \tilde{a}^T\Gamma^{-1}\dot{\tilde{a}}$ vanish and so

$$\dot{V} = -s^T K_D s \leq 0.$$

We are not done yet. To show s converges to zero (which is implied by $\dot{V}$ converges to zero), by Barbalat's stability certificate 5.6, it suffices to show

$$\ddot{V} = -2s^T K_D \dot{s}$$

is bounded. We already know s and $\tilde{a}$ are bounded, due to the fact that V in (8.49) is bounded. Therefore, we only need to show $\dot{s}$ is bounded. To do so, we plug the adapative control law (8.48) into the manipulator dynamics (8.34) and obtain

$$H\dot{s} + (C + K_D)s = Y\tilde{a},$$

which implies the boundedness of $\dot{s}$ (note that H is uniformly positive definite, i.e., $H \succeq \alpha I$ for some $\alpha > 0$). This concludes the analysis of the tracking convergence $s \to 0$ as $t \to \infty$.

8.2 Certainty-Equivalent Adaptive Control

Chapter 9

Problem Sets

Exercise 9.1 (Inscribed Polygon of Maximal Perimeter)**.** In this exercise, we will use dynamic programming to solve a geometry problem, i.e., to find the N-side polygon inscribed inside a circle with maximum perimeter. We will walk you through the key steps of formulating and solving the problem, while leaving a few mathematical details for you to fill in.

Given a circle with radius 1, we can randomly choose N distinct points on the circle to form a polygon with N vertices and sides, as shown in Fig. 9.1 with $N = 3, 4, 5$.

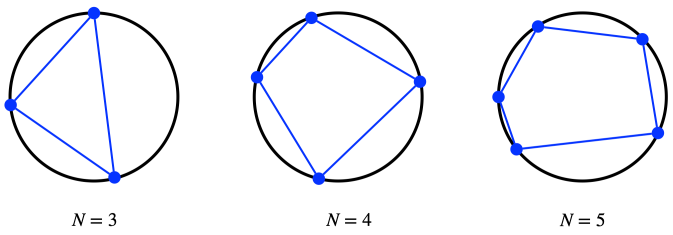

Figure 9.1: Polygons inscribed inside a circle

Once the N points are chosen, the N-polygon will have a perimeter, i.e., the sum of the lengths of its edges.

What is the configuration of the N points such that the resulting N-polygon has the maximum perimeter? I claim that the answer is when the N-polygon has edges of equal lengths, or in other words, when the N points are placed on the circle evenly.

Let us use dynamic programming to prove the claim.

To use dynamic programming, we need to definie a dynamical system and an objective function.

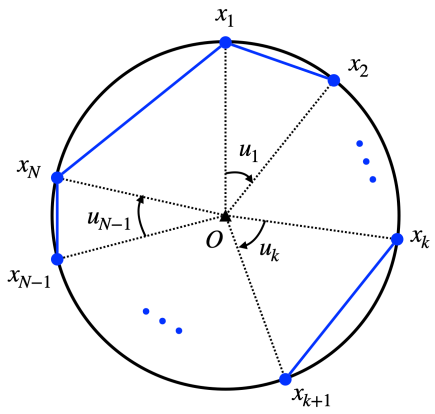

Figure 9.2: Sequential placement of N points on the circle.

Dynamical system. We will use $\{x_1, \dots, x_N\}$ to denote the angular positions of the N points to be placed on the circle (with slight abuse of notation, we will call each of those points x_k as well). In particular, as shown in Fig. 9.2, let us use x_k to denote the angle between the line $O - x_k$ and the vertical line (O is the center of the circle), with zero angle starting at 12 O'clock and clockwise being positive. Without loss of generality, we assume $x_1 = 0$. (if x_1 is nonzero, we can always rotate the entire circle so that $x_1 = 0$).

After the k-th point is placed, we can "control" where the next point x_{k+1} will be, by deciding the incremental angle between x_{k+1} and x_k, denoted as $u_k > 0$ in Fig. 9.2. This is simply saying the dynamics is

$$x_{k+1} = x_k + u_k, \quad k = 1, \dots, N-1, \quad x_1 = 0.$$

Cost-to-go. The perimeter of the N-polygon is therefore

$$g_N(x_N) + \sum_{k=1}^{N-1} g_k(x_k, u_k),$$

with the terminal cost

$$g_N(x_N) = 2 \sin\left(\frac{2\pi - x_N}{2}\right)$$

the distance between x_N and x_1 (see Fig. 9.2), and the running cost

$$g_k(x_k, u_k) = 2 \sin\left(\frac{u_k}{2}\right)$$

the distance between x_{k+1} and x_k.

Dynamic programming. We are now ready to invoke dynamic programming. We start by setting

$$J_N(x_N) = g_N(x_N) = 2\sin\left(\frac{2\pi - x_N}{2}\right).$$

We then compute $J_{N-1}(x_{N-1})$ as

$$J_{N-1}(x_{N-1}) = \max_{0<u_{N-1}<2\pi-x_{N-1}} \left\{ \underbrace{2\sin\left(\frac{u_{N-1}}{2}\right) + J_N(x_{N-1}+u_{N-1})}_{Q_{N-1}(x_{N-1},u_{N-1})} \right\}, \quad (9.1)$$

where $u_{N-1} < 2\pi - x_{N-1}$ because we do not want x_N to cross 2π.

a. Show that

$$Q_{N-1}(x_{N-1},u_{N-1}) = 2\sin\left(\frac{u_{N-1}}{2}\right) + 2\sin\left(\frac{2\pi - x_{N-1} - u_{N-1}}{2}\right),$$

and

$$\frac{\partial Q_{N-1}(x_{N-1},u_{N-1})}{\partial u_{N-1}} = \cos\left(\frac{u_{N-1}}{2}\right) - \cos\left(\frac{2\pi - x_{N-1} - u_{N-1}}{2}\right).$$

b. Show that $Q_{N-1}(x_{N-1},u_{N-1})$ is concave (i.e., $-Q_{N-1}(x_{N-1},u_{N-1})$ is convex) in u_{N-1} for every $x_{N-1} \in (0,\pi)$ and $u_{N-1} \in (0, 2\pi - x_{N-1})$. (Hint: compute the second derivative of $Q_{N-1}(x_{N-1},u_{N-1})$ with respect to u_{N-1} and use Proposition B.2).

c. With a and b, show that the optimal u_{N-1} that solves (9.1) is

$$u^\star_{N-1} = \frac{2\pi - x_{N-1}}{2},$$

and therefore

$$J_{N-1}(x_{N-1}) = 4\sin\left(\frac{2\pi - x_{N-1}}{4}\right).$$

(Hint: the point at which a concave function's gradient vanishes must be the unique maximizer of that function)

d. Now use induction to show that the k-th step dynamic programming

$$J_k(x_k) = \max_{0<u_k<2\pi-x_k} \left\{ 2\sin\left(\frac{u_k}{2}\right) + J_{k+1}(x_k+u_k) \right\}$$

admits an optimal control

$$u^\star_k = \frac{2\pi - x_k}{N-k+1},$$

and optimal cost-to-go

$$J_k(x_k) = 2(N-k+1)\sin\left(\frac{2\pi - x_k}{2(N-k+1)}\right).$$

e. Starting from $x_1 = 0$, what is the optimal sequence of controls?

Hopefully now you see why my original claim is true!

(Bonus) We are not yet done for this exercise. Since you have probably already spent quite some time on this exercise, I will leave the rest of the exercise a bonus. In case you found this simple geometric problem interesting, you should keep reading as we will use numerical techniques to prove the same claim.

In Fig. 9.2, by denoting

$$u_N = 2\pi - x_N = 2\pi - (u_1 + \cdots + u_{N-1})$$

as the angle between the line $O - x_N$ and the line $O - x_1$, it is not hard to observe that the perimeter of the N-polygon is

$$\sum_{k=1}^{N} 2\sin\left(\frac{u_k}{2}\right).$$

Consequently, to maximize the perimeter, we can formulate the following optimization

$$\begin{aligned} \max_{u_1,\ldots,u_N} \quad & \sum_{k=1}^{N} 2\sin\left(\frac{u_k}{2}\right) \\ \text{subject to} \quad & u_k > 0, k = 1, \ldots, N \\ & u_1 + \cdots + u_N = 2\pi \end{aligned} \tag{9.2}$$

where u_k can be seen as the angle spanned by the line $x_k - x_{k+1}$ with respect to the center O so that they are positive and sum up to 2π.

f. Show that the optimization (9.2) is convex. (Hint: first show the feasible set is convex, and then show the objective function is concave over the feasible set.)

Now that we have shown (9.2) is a convex optimization problem, we know that pretty much any numerical algorithm will guarantee convergence to the globally optimal solution.

It is too much to ask you to implement a numerical algorithm on your own, as that can be a one-semester graduate-level course (Nocedal and Wright, 1999). However, Matlab provides a nice interface, `fmincon`, to many such numerical algorithms, and let me show you how to use `fmincon` to solve (9.2) so we can numerically prove our claim.

g. I have provided most of the code necessary for solving (9.2) below. Please fill in the definition of the function `perimeter(u)`, and then run the code in Matlab. Show your results for $N = 3, 10, 100$. Do the solutions obtained from `fmincon` verify our claim?

```
clc; clear; close all;
% number of points to be placed
N = 10;
% define the objective function
% fmincon assumes minimization
% We minimize the negative perimeter so as to maximize the perimeter
objective = @(u) -1*perimeter(u);
% choose which algorithm to use for solving
options = optimoptions('fmincon', 'Algorithm', 'interior-point');
% supply an initial guess
% since this is a convex problem, we can use any initial guess
u0 = rand(N,1);
% solve
uopt = fmincon(objective,u0,... % objective and initial guess
    -eye(N),zeros(N,1),... % linear inequality constraints
    ones(1,N),2*pi,... % linear equality constraints
    [],[],[],... % we do not have lower/upper bounds and nonlinear constraints
    options);

% plot the solution
x = zeros(N,1);
for k = 2:N
    x(k) = x(k-1) + uopt(k-1);
end
figure;
% plot a circle
viscircles([0,0],1);
hold on
% scatter the placed points
scatter(cos(x),sin(x),'blue','filled');
axis equal;

%% helper functions
% The objective function
function f = perimeter(u)
% TODO: define the perimeter function here.
end
```

Exercise 9.2 (LQR with Constraints)**.** Consider a variant of the LQR problem (2.2) where the controls are bounded between $[-u_{\max}, u_{\max}]$, the system

matrices A_k, B_k are constant, and the dynamics is deterministic:

$$\begin{aligned} J(x_0) = \min_{u_0,\dots,u_{N-1}\in[-u_{\max},u_{\max}]} \quad & x_N^T Q_N x_N + \sum_{k=0}^{N-1} (x_k^T Q_k x_k + u_k^T R_k u_k) \\ \text{subject to} \quad & x_{k+1} = Ax_k + Bu_k, k = 0, \dots, N-1 \end{aligned} \tag{9.3}$$

We assume $Q_k \succeq 0$ for $k = 0, \dots, N$ and $R_k \succ 0$ for all $k = 0, \dots, N-1$.

a. Show that Problem (9.3), when x_0 is given, is a convex optimization problem.

b. Discretize the continuous-time double integrator dynamics

$$\ddot{q} = u, \quad u \in [-1, 1]$$

in the form of $x_{k+1} = Ax_k + Bu_k$ with a constant dt time discretization. (Hint: take $x = [q, \dot{q}]$ as the state vector.)

c. Fix $N = 50$, $dt = 0.1$ and choose your favorite Q_k and R_k. Solve the convex optimization (9.3) at a dense grid of x_0. Plot $J(x_0)$. (Hint: you will most likely benefit from Appendix B.2.)

d. Increase N and decrease dt, plot $J(x_0)$ for different pairs of (N, dt). What do you observe?

e. (Bonus) Can you write down the KKT optimality conditions of (9.3) and explain what you have observed from the numerical experiment? (Hint: KKT optimality conditions can be found in Theorem B.2.)

Exercise 9.3 (Cart Pole System)**.** In this question, we will study the cart-pole system that we saw the video of human-controlled version in our first lecture. The task is to balance a pendulum on a cart by horizontally moving the cart. See this video (https://www.youtube.com/watch?v=Bzq96V1yN5k) for an actual robotic implementation.

With the above illustration, we parameterize the system with two scalers: x represents the current location of the cart, while θ is the angle between current pole and the stable equilibrium. Therefore, our goal is to study the motion of the cart-pole system with a horizonal control f. We assume hereafter the system is ideal such that there is no friction, and the mass of the pole concentrates at the free end point.

a. Denote the mass of cart and pole as m_c and m_p, respectively. (i). Derive the equations of motion:

$$\left(m_c + m_p\right)\ddot{x} + m_p l\ddot{\theta}\cos\theta - m_p l\dot{\theta}^2 \sin\theta = f, \tag{9.4}$$

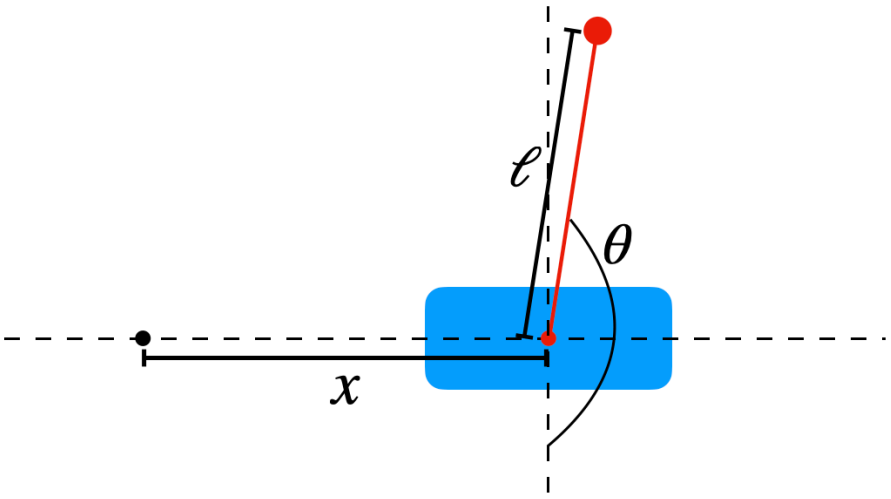

Figure 9.3: Illustration of cart-pole problem

$$m_p l\ddot{x}\cos\theta + m_p l^2\ddot{\theta} + m_p gl\sin\theta = 0. \tag{9.5}$$

(Hints: compute the Lagrangian of the system and the corresponding Lagrangian equations. Analyzing the two objects separately also works.) (ii). Translate the equations (9.4) and (9.5) into the standard form

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_g(\mathbf{q}) + \mathbf{B}\mathbf{u}, \tag{9.6}$$

where $\mathbf{q} = \begin{bmatrix} x \\ \theta \end{bmatrix}$, $\mathbf{u} = [f]$. What are $\mathbf{M}, \mathbf{C}, \tau_g, \mathbf{B}$ here? (iii). Translate the equations into the basic state-space dynamics form

$$\dot{\mathbf{x}} = f(\mathbf{x},\mathbf{u}). \tag{9.7}$$

What are $\mathbf{x}, \mathbf{u}, f$ here? *(Hint: try* $\mathbf{x} = [x, \theta, \dot{x}, \dot{\theta}]^\top$*.)*

b. Linearize the dynamics in 1.(c) around the unstable equilibrium where $\theta^* = \pi$ and $x^* = \dot{x}^* = \dot{\theta}^* = 0$. The result should be in the form of

$$\dot{\Delta\mathbf{x}} = A\Delta\mathbf{x} + B\Delta\mathbf{u}, \tag{9.8}$$

where $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}^*$ and $\Delta\mathbf{u} = \mathbf{u} - \mathbf{u}^*$.

c. Define the linearization error $e(\mathbf{x},\mathbf{u}) := \|f(\mathbf{x},\mathbf{u}) - (A\Delta\mathbf{x} + B\Delta\mathbf{u})\|^2$. Simulate the original system (9.7) and the linearized system (9.8) with the same initial condition. How does the linearization error change over time? Provide at least three different initialization results. *(Hints: (i) Sanity check: intuitively the error should not depend on the initial location* x*, and*

it should have symmetry. Is that true in your simulation? (ii) In the same unstable position, how does push/pull (positive/negative) force change the results?)

d. (Bonus) Will a linear controller (i.e., f is linear in $\mathbf{x}$) be a good controller? Why or why not? The answer might depend on whether the end point of the pole is above the horizonal line.

Acknowledgement

Appendix A

Linear Algebra and Differential Equations

In this Chapter, we provide basic concepts in linear algebra and ordinary differential equations (ODE), which can be a cheatsheet for readers.

A.1 Linear Algebra

Most of the linear algebraic concepts used in this textbook are provided in this section.

A.1.1 Matrix Exponential

Definition A.1 (Matrix exponential)**.** Given a $n \times n$ matrix A, the matrix exponential of A, denoted as e^A, is defined as:

$$e^A = \sum_{p=0}^{\infty} \frac{A^p}{p!}.$$

Note that the matrix exponential is well defined, and every entry converges absolutely. We show some special cases of matrix exponential.

1. Diagonal matrix. Note that if $A = \text{diag}(a_1, a_2, \dots, a_n)$, then $A^p = \text{diag}(a_1^p, a_2^p, \dots, a_n^p)$, and $e^A = \text{diag}(e^{a_1}, e^{a_2}, \dots, e^{a_n})$.
2. Diagonalizable matrix:

Definition A.2 (Diagonablizable matrix)**.** A square matrix A is said to be **diagonalizable** or **non-defective**, if there exists an invertible matrix P, such that $P^{-1}AP$ is a diagonal matrix. In words, after change of coordination, the matrix becomes diagonal. If a matrix is not diagonalizable, it is **defective**.

With diagonalization $A = PDP^{-1}$ where D is a diagonal matrix (e.g. symmetric matrix is diagonalizable), we have $e^A = Pe^DP^{-1}$. Specifically if U consists of eigenvectors of A and D is the spectrum, then matrix exponential is the same as taking exponents of eigenvalues of A while keeping the eigenbasis invariant.

3. Exchangable matrices. If A_1 and A_2 are exchangeable, that is, $A_1A_2 = A_2A_1$, then $e^{A_1+A_2} = e^{A_1}e^{A_2}$ (exercise!).

4. Nilpotent matrix. If N is a nilpotent matrix, which means that $N^K = 0$ for some integer K, then $e^N = I + N + \frac{1}{2}N^2 + ... + \frac{1}{(K-1)!}N^{K-1}$.

5. Any matrix with Jordan canonical form (*This is out of scope of this textbook. We leave it for readers with interest). Any matrix A can be decomposed as $P(D + N)P^{-1}$, where D is diagonal and exchangeable with the nilpotent matrix N. Then based on the above discussions, $e^A = P(e^De^N)P^{-1}$.

6. Projection matrix $A^2 = A$. Then $e^A = I + (e - 1)A$ (exercise!).

A.1.2 Gradients

In matrix calculus, index may not be consistent in different references. It should be noted that in neural network literature, gradients may have a transpose on our results here.

For any function $f(X) : \mathbb{R}^{m\times n} \to \mathbb{R}$ of an m-by-n matrix X (if $n = 1$ then it is a vector), the gradient $\nabla f(X)$ is another m-by-n matrix with the (i, j)-th entry $\frac{\partial f(X)}{\partial X_{ij}}$.

For any function $f(x) : \mathbb{R}^m \to \mathbb{R}^n$ that maps an m-dimensional vector x to n-dimensional space, $\nabla f(x)$ is an m-by-n matrix with the (i, j)-th entry $\frac{\partial (f(x))_j}{\partial x_i}$.

Some important examples include:

- Vector inner-product. $\nabla(a^\top x) = a$.
- Quadratic form $\nabla(x^\top Ax) = (A + A^\top)x$.
- Composition. Suppose $f(y) : \mathbb{R}^n \to \mathbb{R}^k$ and $g(x) : \mathbb{R}^m \to \mathbb{R}^n$, then $\nabla f(g(x))$ is an m-by-k matrix $\nabla g(x)\nabla f(y = g(x))$.

A.2 Solving an Ordinary Differential Equation

A **differential equation** is an equation with a function and its derivatives. Compared to **partial differential equation (PDE)** that consists of partial derivatives, throughout this textbook we will mainly focus on **ordinary differential equations (ODEs)**, including but not limited to, the solutions, convergence and stability analysis.

An example of ODE looks like this:

$$x + \frac{dx}{dt} = 5t, \quad t \in [0, T]. \tag{A.1}$$

Definition A.3 (Ordinary Differential Equation)**.** In general, an ODE of **order** k, or, a k-th order ODE, is in the form of

$$F(t, x, x', ..., x^{(k)}) = 0.$$

Further, if F is linear in $x, x', ..., x^{(k)}$, we call it a **linear ODE**. If a linear ODE with $F = x^{(k)} + \sum_{i=0}^{k-1} a_i(t)x^{(i)}$ that does not independently relate to t, we call it **homogeneous**. Note that $x(t) \equiv 0$ will always be a trivial solution for homogeneous ODE.

For example, the equation (A.1) is a linear ODE but not homogeneous.

The solution, a function $x = x(t)$ is not unique in general, and additional conditions are required. For example, we can have one of the following conditions for the above ODE:

- Initial condition, e.g., $x(0) = 0$ gives the constraint that the function at initial time $t = 0$ starts at zero point.
- End-point condition, e.g., $x(T) = 0$ implies that the dynamics should end at zero.
- (Initial) velocity condition, e.g., $\frac{dx}{dt}|_{t=0} = 0$ suggests that at the start time the "slope"/"velocity" of x is zero.

For higher-order ODEs, the conditions may be much more complicated to guarantee uniqueness.

A.2.1 Separation of Variables

A.2.2 First-order Linear ODE

A.2.3 Gronwall Inequality

A.2.4 Matlab

Appendix B

Convex Analysis and Optimization

B.1 Theory

B.1.1 Sets

Convex set is one of the most important concepts in convex optimization. Checking convexity of sets is crucial to determining whether a problem is a convex problem. Here we will present some definitions of some set notations in convex optimization.

Definition B.1 (Affine set)**.** A set $C \subset \mathbb{R}^n$ is affine if the line through any two distinct points in C lies in C, i.e., if for any $x_1, x_2 \in C$ and any $\theta \in \mathbb{R}$, we have $\theta x_1 + (1-\theta)x_2 \in C$.

Definition B.2 (Convex set)**.** A set $C \subset \mathbb{R}^n$ is convex if the line segment between any two distinct points in C lies in C, i.e., if for any $x_1, x_2 \in C$ and any $\theta \in [0,1]$, we have $\theta x_1 + (1-\theta)x_2 \in C$.

Definition B.3 (Cone)**.** A set $C \subset \mathbb{R}^n$ is a cone if for any $x \in C$ and any $\theta \geq 0$, we have $\theta x \in C$.

Definition B.4 (Convex Cone)**.** A set $C \subset \mathbb{R}^n$ is a convex cone if C is convex and a cone.

Below are some important examples of convex sets:

Definition B.5 (Hyperplane)**.** A hyperplane is a set of the form

$$\{x | a^T x = b\}$$

Definition B.6 (Halfspaces)**.** A (closed) halfspace is a set of the form

$$\{x|a^T x \leq b\}$$

Definition B.7 (Balls)**.** A ball is a set of the form

$$B(x,r) = \{y|\|y-x\|_2 \leq r\} = \{x+ru|\|u\|_2 \leq 1\}$$

where $r > 0$.

Definition B.8 (Ellipsoids)**.** A ellipsoid is a set of the form

$$\mathcal{E} = \{y|(y-x)^T P^{-1}(y-x) \leq 1\}$$

where P is symmetric and positive definite.

Definition B.9 (Polyhedra)**.** A polyhedra is defined as the solution set of a finite number of linear equalities and inequalities:

$$\mathcal{P} = \{x|a_j^T x \leq b_j, j = 1,...,m, c_k^T x = d_k, k = 1,...,p\}$$

Definition B.10 (Norm ball)**.** A norm ball B of radius r and a center x_c associated with the norm $\|\cdot\|$ is defined as:

$$B = \{x|\|x - x_c\| \leq r\}$$

Definition B.11 (Norm cone)**.** A norm cone C associated with the norm $\|\cdot\|$ is defined as:

$$C = \{(x,t)|\|x\| \leq t\} \subset \mathbb{R}^{n+1}$$

Simplexes are important family of polyhedra. Suppose the $k+1$ points $v_0,...,v_k \in \mathbb{R}^n$ are affinely independent, which means $v_1 - v_0,...,v_k - v_0$ are linearly independent.

Definition B.12 (Simplex)**.** A simplex C defined by points $v_0,...,v_k$ is:

$$C = \mathbf{conv}\{v_0,...,v_k\} = \{\theta_0 v_0 + ...\theta_k v_k|\theta \succeq 0, \mathbf{1}^T\theta = 1\}$$

Extremely important examples of convex sets are positive semidefinite cones:

Definition B.13 (Symmetric,positive semidefinite,positive definite matrices)**.**

1. Symmetric matrices: $\mathbf{S}^n = \{X \in \mathbb{R}^{n\times n}|X = X^T\}$
2. Symmetric Positive Semidefinite matrices: $\mathbf{S}_+^n = \{X \in \mathbf{S}^n|X \succeq 0\}$
3. Symmetric Positive definite matrices: $\mathbf{S}_{++}^n = \{X \in \mathbf{S}^n|X \succ 0\}$

In most scenarios, the set we encounter is more complicated. In general it is extermely hard to determine whether a set in convex or not. But if the set is 'generated' by some convex sets, we can easily determine its convexity. So let's focus on operations that preserve convexity:

Proposition B.1. *Assume S is convex, $S_\alpha, \alpha \in \mathcal{A}$ is a family of convex sets. Following operations on convex sets will preserve convexity:*

1. *Intersection: $\bigcap_{\alpha\in\mathcal{A}} S_\alpha$ is convex.*
2. *Image under affine function: A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is affine if it has the form $f(x) = Ax+b$. The image of S under affine function f is convex. I.e. $f(S) = \{f(x)|x \in S\}$ is convex*
3. *Image under perspective function: We define the perspective function $P : \mathbb{R}^{n+1}$, with domain $\mathbf{dom}P = \mathbb{R}^n \times \mathbb{R}_{++}$(where $\mathbb{R}_{++} = \{x \in \mathbb{R}|x > 0\}$) as $P(z,t) = z/t$. The image of S under perspective function is convex.*
4. *Image under linear-fractional function: We define linear fractional function $f : \mathbb{R}^n \to \mathbb{R}^m$ as:$f(x) = (Ax+b)/(c^Tx+d)$ with $\mathbf{dom}f = \{x|c^Tx+d > 0\|$. The image of S under linear fractional functions is convex.*

In some cases, the restrictions of **interior** is too strict. For example, imagine a plane in $\mathbb{R}^3$. The interior of the plane is $\emptyset$. But intuitively many property should be extended to this kind of situation. Because the points in the plane also lies 'inside' the convex set. Thus, we will define **relative interior**. First we will define **affine hull**.

Definition B.14 (Affine hull)**.** The affine hull of a set S is the smallest affine set that contains S, which can be written as:

$$\text{aff}(S) = \{\sum_{i=1}^{k} \alpha_i x_i | k > 0, x_i \in S, \alpha_i \in \mathbb{R}, \sum_{i=1}^{k} \alpha_i = 1\}$$

Definition B.15 (Relative Interior)**.** The relative interior of a set S (denoted relint(S)) is defined as its interior within the affine hull of S. I.e.

$$\text{relint}(S) := \{x \in S : \text{there exists } \epsilon > 0 \text{ such that } N_\epsilon \cap \text{aff}(S) \subset S\}$$

where $N_\epsilon(x)$ is a ball of radius ϵ centered on x.

B.1.2 Convex function

In this section, let's define convex functions:

Definition B.16 (Convex function)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is **convex** if **dom** f is convex and $\forall x, y \in$ **dom** f and with $\theta \in [0,1]$, we have:

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$$

The function is **strictly convex** if the inequality holds whenever $x \neq y$ and $\theta \in (0,1)$.

If a function is differentiable, it will be easier for us to check its convexity:

Proposition B.2 (Conditions for Convex function)**.** *1.(First order condition) Suppose f is differentiable, then f is convex if and only if $\boldsymbol{domf}$ is convex and $\forall x, y \in \boldsymbol{domf}$,*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$

2.(Second order conditions) Suppose f is twice differentiable, then f is convex if and only if $\boldsymbol{domf}$ is convex and $\forall x \in \boldsymbol{domf}$,

$$\nabla^2 f(x) \succeq \boldsymbol{0}$$

For the same purpose, some operations that preserve the convexity of the convex functions are presented here:

Proposition B.3 (Operations that preserve convexity)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function and $g_1, ..., g_n$ be convex functions. The following operations will preserve convexity of the function:*

1.(Nonnegative weighted sum): A nonnegative weighted sum of convex functions:

$$f = \omega_1 f_1 + ... + \omega_m f_m$$

2.(Composition with an affine mapping) Suppose $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$, then $g(x) = f(Ax + b)$ is convex.

3.(Pointwise maximum and supremum) $g(x) = \max\{g_1(x), ..., g_n(x)\}$ is convex. If $h(x, y)$ is convex in x for each $y \in \mathcal{A}$, then $\sup_{y \in \mathcal{A}} h(x, y)$ is also convex in x.

4.(Minimization) If $h(x, y)$ is convex in (x, y), and C is a convex nonempty set, then $\inf_{x \in C} h(x, y)$ is convex in x.

5.(Perspective of a function) The perspective of f is the function $h : \mathbb{R}^{n+1} \to \mathbb{R}$ defined by: $h(x, t) = tf(x/t)$ with domain $\boldsymbol{dom}\ h = \{(x, t) | x/t \in \boldsymbol{domf}, t > 0\}$. And h is convex.

B.1.3 Lagrange dual

We consider an optimization problem in the standard form (without assuming convexity of anything):

$$\begin{aligned} p^* = \quad & \min_x \quad f_0(x) \\ & \text{s.t.} \quad f_i(x) \leq 0 \quad i = 1..., m \\ & \qquad h_i(x) = 0 \quad i = 1, ..., p \end{aligned} \tag{B.1}$$

Definition B.17 (Lagrange dual function)**.** The Lagrangian related to the problem above is defined as:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$$

The Lagrange dual function is defined as:

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu)$$

When the Lagrangian is unbounded below in x, the dual function takes on the value $-\infty$. Note that since the Lagrange dual function is a pointwise infimum of a family of affine functions of (λ, ν), so it's concave. The Lagrange dual function will give us lower bounds of the optimal value of the original problem:

$$g(\lambda, \nu) \leq p^*$$

. We can see that, the dual function can give a nontrivial lower bound only when $\lambda \succeq 0$. Thus we can solve the following dual problem to get the best lower bound.

Definition B.18 (Lagrange dual problem)**.** The lagrangian dual problem is defined as follows:

$$\begin{aligned} d^* = \quad & \max_{\lambda, \nu} \quad g(\lambda, \nu) \\ & \text{s.t.} \quad \lambda \succeq 0 \end{aligned} \tag{B.2}$$

This is a convex optimization problem.

We can easily see that

$$d^* \leq p^*$$

always hold. This property is called **weak duality**. If

$$d^* = p^*$$

, it's called **strong duality**. Strong duality does not hold in general, but it usually holfs for convex problems. We can find conditions that guarantee strong duality in convex problems, which are called constrained qualifications. Slater's constraint qualification is a useful one.

Theorem B.1 (Slater's constraint qualification)**.** *Strong duality holds for a convex problem*

$$\begin{aligned} p^* = \quad & \min_{x} \quad f_0(x) \\ & s.t. \quad f_i(x) \leq 0 \quad i = 1..., m \\ & \qquad Ax = b \end{aligned} \tag{B.3}$$

if it is strictly feasible, i.e.

$$\exists x \in \boldsymbol{relint}\mathcal{D} : \quad f_i(x) < 0, \quad i = 1...m, \quad Ax = b$$

And the linear inequalities do not need to hold with strict inequality.

B.1.4 KKT condition

Note that if strong duality holds, denote x^* to be primal optimal, and (λ^*, ν^*) to be dual optimal. Then:

$$\begin{aligned} f_0(x^*) = g(\lambda^*, \nu^*) =& \inf_x (f_0(x) + \sum_{i=1}^m \lambda_i^* f_i(x) + \sum_{i=1}^p \nu_i^* h_i(x)) \\ \leq& f_0(x^*) + \sum_{i=1}^m \lambda_i^* f_i(x) + \sum_{i=1}^p \nu_i^* h_i(x) \\ \leq& f_0(x^*) \end{aligned} \tag{B.4}$$

from this, combining $\lambda^* \geq 0$ and $f_i(x^*) \leq 0$, we can know that: $\lambda_i^* f_i(x^*) = 0 \quad i = 1 \cdots m$. This means for λ_i^* and $f_i(x^*)$, one of them must be zero, which is known as complementary slackness).

Thus we arrived at the following four conditions, which are called KKT conditions.

Theorem B.2 (Karush-Kuhn-Tucker(KKT) Conditions)**.** *The following four conditions are called KKT conditions (for a problem with differentiable f_i, h_i)*

1. *Primal feasible:* $f_i(x) \leq 0, i, \cdots, m,\ h_i(x) = 0, i = 1, \cdots, p$
2. *Dual feasible:* $\lambda \succeq 0$
3. *Complementary slackness:* $\lambda_i f_i(x) = 0, i = 1, \cdots, m$
4. *Gradient of Lagrangian with respect to x vanishes:* $\nabla f_0(x) + \sum_{i=1}^m \lambda_i \nabla f_i(x) + \sum_{i=1}^p \nu_i \nabla h_i(x) = 0$

From the discussion above, we know that if strong duality holds and x, λ, ν are optimal, then they must satisfy the KKT conditions.

Also if x, λ, ν satisfy KKT for a convex problem, then they are optimal. However, the converse is not generally true, since KKT condition implies strong duality. If Slater's condition is satisfied, then x is optimal if and only if there exist λ, ν that satisfy KKT conditions. Sometimes, by solving the KKT system, we can derive the closed-form solution of a optimization directly. Also, sometimes we will use the residual of the KKT system as the termination condition.

In general, f_i, h_i may not be differentiable. There are also KKT conditions for them, which will include knowledge of subdifferential and will not be included here.

B.2 Practice

B.2.1 CVX Introduction

In the last section, we have learned basic concepts and theorems in convex optimization. In this section, on the other hand, we will introduce you how to model basic convex optimization problems with CVX, an easy-to-use MATLAB package. To install CVX, please refer to this page. Note that every time you what to use the CVX package, you should add it to your MATLAB path. For example, if I install CVX package in the parent directory of my current directory with default directory name `cvx`, the following line should be added before your CVX codes:

```
addpath(genpath("../cvx/"));
```

With CVX, it is incredibly easy for us to define and solve a convex optimization problem. You just need to:

1. define the variables.
2. define the objective function you want to minimize or maximize.
3. define the constraints.

After running your codes, the optimal objective value is stored in the variable `cvx_optval`, and the problem status is stored in the variable `cvx_status` (when your problem is well-defined, this variable's value will be `Solved`). The optimal solutions will be stored in the variables you define.

Throughout this section, we will study five types of convex optimization problems: linear programming (LP), quadratic programming (QP), (convex) quadratically constrained quadratic programming (QCQP), second-order cone programming (SOCP), and semidefinite programming (SDP). Given two types of optimization problems A and B, we say $A < B$ if A can always be converted to B while the inverse is not true. Under this notation, we have

$$\text{LP} < \text{QP} < \text{QCQP} < \text{SOCP} < \text{SDP}$$

B.2.2 Linear Programming (LP)

Definition. An LP has the following form:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \ c^T x \\ &\text{subject to } Ax \leq b \end{aligned} \tag{B.5}$$

where x is the variable, $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$ are the parameters. Note that the constraint $Ax \leq b$ already incorporates linear equality constraints. To see this, consider the constraint $A'x = b'$, we can reformulate it as $Ax \leq b$ by

$$\begin{bmatrix} A' \\ -A' \end{bmatrix} x \leq \begin{bmatrix} b' \\ -b' \end{bmatrix}$$

Example. Consider the problem of minimizing a linear function $c_1x_1 + c_2x_2$ over a rectangle $[-l_1, l_1] \times [-l_2, l_2]$. We can convert it to the standard LP form in (B.5) by simply setting c as $[c_1,\ c_2]^T$ and the linear inequality constraint as

$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} l_1 \\ l_1 \\ l_2 \\ l_2 \end{bmatrix}$$

Corresponding CVX codes are shown below:

```
%% Define the LP example setting
c1 = 2;
c2 = -5;
l1 = 3;
l2 = 7;
% parameters: c, A, b
c = [c1; c2];
A = [1, 0; -1, 0; 0, 1; 0, -1];
b = [l1; l1; l2; l2];

%% solve LP
cvx_begin
    variable x(2); % define variables [x1, x2]
    minimize(c' * x); % define the objective
    subject to
        A * x <= b; % define the linear constraint
cvx_end
```

B.2.3 Quadratic Programming (QP)

Definition. A QP has the following form:

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}x^TPx + q^Tx \tag{B.6}$$

$$\text{subject to } Gx \leq h \tag{B.7}$$

$$Ax = b \tag{B.8}$$

where $P \in \mathcal{S}_+^n, q \in \mathbb{R}^n, G \in \mathbb{R}^{m\times n}, h \in \mathbb{R}^m, A \in \mathbb{R}^{p\times n}, b \in \mathbb{R}^p$. Here $\mathcal{S}_+^n$ denotes the set of positive semidefinite matrices of size $n \times n$. Obviously, if we set P as zero, QP will degenerate to LP.

Example. Consider the problem of minimizing a quadratic function

$$f(x_1, x_2) = p_1 x_1^2 + 2p_2 x_1 x_2 + p_3 x_2^2 + q_1 x_1 + q_2 x_2$$

over a rectangle $[-l_1, l_1] \times [-l_2, l_2]$. Since $P = 2\begin{bmatrix} p_1 & p_2 \\ p_2 & p_3 \end{bmatrix} \succeq 0$, the following two conditions must hold:

$$\begin{cases} p_1 \geq 0 \\ p_1 p_3 - 4p_2^2 \geq 0 \end{cases}$$

Same as in the LP example, G and h can be expressed as:

$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} l_1 \\ l_1 \\ l_2 \\ l_2 \end{bmatrix}$$

Corresponding CVX codes are shown below:

```
%% Define the QP example setting
p1 = 2;
p2 = 0.5;
p3 = 4;
q1 = -3;
q2 = -6.5;
l1 = 2;
l2 = 2.5;
% check if the generated P is positive semidefinite
tmp1 = (p1 >= 0);
tmp2 = (p1*p3 - 4*p2^2 >= 0);
if ~(tmp1 && tmp2)
    error("P is not positve semidefinite!");
end
% parameters: P, q, G, h
P = 2 * [p1, p2; p2, p3];
q = [q1; q2];
G = [1, 0; -1, 0; 0, 1; 0, -1];
h = [l1; l1; l2; l2];

%% Solve the QP problem
cvx_begin
    variable x(2); % define variables [x1; x2]
```

```
    % define the objective, where quad_form(x, P) = x'*P*x
    obj = 0.5 * quad_form(x, P) + q' * x;
    minimize(obj);
    subject to
        G * x <= h; % define the linear constraint
cvx_end
```

B.2.4 Quadratically Constrained Quadratic Programming (QCQP)

Definition. An (convex) QCQP has the following form:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T P_0 x + q_0^T x \tag{B.9}$$

$$\text{subject to } \frac{1}{2} x^T P_i x + q_i^T x + r_i \leq 0, \ i = 1 \dots m \tag{B.10}$$

$$Ax = b \tag{B.11}$$

where $P_i \in \mathcal{S}_+^n, i = 0 \dots m$, $q_i \in \mathbb{R}^n, i = 0 \dots m$, $A \in \mathbb{R}^{p \times n}$, and $b \in \mathbb{R}^p$. Note that in other literature, you may find a more general form of QCQP: they don't require P_i's to be positive semidefinite. Yet in this case, the problem is non-convex and beyond our scope.

Example. We study the problem of getting the minimum distance between two ellipses. By convention, when the ellipses overlap, we set the minimum distance as 0. This problem can be exactly solved by (convex) QCQP. Consider two ellipses of the following form:

$$\begin{cases} \frac{1}{2} \begin{bmatrix} y_1 \\ z_1 \end{bmatrix}^T K_1 \begin{bmatrix} y_1 \\ z_1 \end{bmatrix} + k_1^T \begin{bmatrix} y_1 \\ z_1 \end{bmatrix} + c_1 \leq 0 \\ \frac{1}{2} \begin{bmatrix} y_2 \\ z_2 \end{bmatrix}^T K_2 \begin{bmatrix} y_2 \\ z_2 \end{bmatrix} + k_2^T \begin{bmatrix} y_2 \\ z_2 \end{bmatrix} + c_2 \leq 0 \end{cases}$$

where $[y_1, z_1]^T$ and $[y_2, z_2]^T$ are arbitrary points inside the two ellipses respectively. Also, two ensure the ellipses are well defined, we should enforce the following properties in $(K_i, k_i, c_i), i = 1, 2$: (1) $K_i \succ 0$; (2) Let $K_i = L_i L_i^T$ be the Cholesky decomposition of K_i. Then, ellipse i can be rewritten as:

$$\frac{1}{2} \parallel L_i^T \begin{bmatrix} y_i \\ z_i \end{bmatrix} - L_i^{-1} k_i \parallel^2 \leq \frac{1}{2} \parallel L_i^{-1} k_i \parallel^2 - c_i$$

Thus,

$$\frac{1}{2} \parallel L_i^{-1} k_i \parallel^2 - c_i > 0$$

With these two assumptions, we want to minimize:

$$\frac{1}{2}(y_1 - y_2)^2 + (z_1 - z_2)^2$$

Now, we construct P, q, r's in QCQP with the above parameters. Define the variable x as $[y_1, z_1, y_2, z_2]$.

(1) P_0 can be obtained from:

$$\frac{1}{2}(y_1 - y_2)^2 + (z_1 - z_2)^2 = \frac{1}{2}\begin{bmatrix} y_1 \\ z_1 \\ y_2 \\ z_2 \end{bmatrix}^T \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ z_1 \\ y_2 \\ z_2 \end{bmatrix}$$

(2) P_1, q_1, r_1 can be obtained from:

$$\frac{1}{2}\begin{bmatrix} y_1 \\ z_1 \end{bmatrix}^T K_1 \begin{bmatrix} y_1 \\ z_1 \end{bmatrix} + k_1^T \begin{bmatrix} y_1 \\ z_1 \end{bmatrix} + c_1 = \frac{1}{2}x^T \begin{bmatrix} K_1 & O \\ O & O \end{bmatrix} + \begin{bmatrix} k_1 \\ O \end{bmatrix}^T x + c_1 \leq 0$$

(3) P_2, q_2, r_2 can be obtained from:

$$\frac{1}{2}\begin{bmatrix} y_2 \\ z_2 \end{bmatrix}^T K_2 \begin{bmatrix} y_2 \\ z_2 \end{bmatrix} + k_2^T \begin{bmatrix} y_2 \\ z_2 \end{bmatrix} + c_2 = \frac{1}{2}x^T \begin{bmatrix} O & O \\ O & K_2 \end{bmatrix} + \begin{bmatrix} O \\ k_2 \end{bmatrix}^T x + c_2 \leq 0$$

The corresponding codes are shown below. In this example, we test the minimum distance between a circle $y_1^2 + z_1^2 \leq 1$ and another circle $(y_2 - 2)^2 + (z_2 - 2)^2 \leq 1$. You can check whether the result from QCQP aligns with your manual calculation.

```
%% Define the QCQP example setting
K1 = eye(2);
k1 = zeros(2, 1);
c1 = -0.5;
K2 = eye(2);
k2 = [2; 2];
c2 = 3.5;
if ~(if_ellipse(K1, k1, c1) && if_ellipse(K2, k2, c2))
    error("The example setting is not correct");
end
% define parameters P0, P1, P2, q1, q2, r1, r2
P0 = [1,0,-1,0; 0,1,0,-1; -1,0,1,0; 0,-1,0,1];
P1 = zeros(4, 4);
P1(1:2, 1:2) = K1;
```

```
P2 = zeros(4, 4);
P2(3:4, 3:4) = K2;
q1 = [k1; zeros(2, 1)];
q2 = [zeros(2, 1); k2];
r1 = c1;
r2 = c2;

%% Solve the QCQP problem
cvx_begin
    variable x(4); % define variables [y1; z1; y2; z2]
    % define the objective, where quad_form(x, P) = x'*P*x
    obj = 0.5 * quad_form(x, P0);
    minimize(obj);
    subject to
        0.5 * quad_form(x, P1) + q1' * x + r1 <= 0;
        0.5 * quad_form(x, P2) + q2' * x + r2 <= 0;
cvx_end

%% detect whether (K, k, c) generates a ellipse
function flag = if_ellipse(K, k, c)
    L = chol(K);
    radius_square = 0.5 * norm(L \ k)^2 - c; % L \ k = inv(L) * k
    flag = (radius_square > 0);
end
```

B.2.5 Second-Order Cone Programming (SOCP)

Definition. An SOCP has the following form:

$$\min_{x \in \mathbb{R}^n} f^T x \tag{B.12}$$

$$\text{subject to } ||A_i x + b_i||_2 \leq c_i^T x + d_i, \ i = 1 \dots m \tag{B.13}$$

$$Fx = g \tag{B.14}$$

where $f \in \mathbb{R}^n, A_i \in \mathbb{R}^{n_i \times n}, b_i \in \mathbb{R}^{n_i}, c_i \in \mathbb{R}^n, d_i \in \mathbb{R}, F \in \mathbb{R}^{p \times n}$, and $g \in \mathbb{R}^p$.

Example. We consider the problem of stochastic linear programming:

$$\min_x c^T x \tag{B.15}$$

$$\text{subject to } \mathbb{P}(a_i^T x \leq b_i) \geq p, \ i = 1 \dots m \tag{B.16}$$

$$a_i \sim \mathcal{N}(\bar{a}_i, \Sigma_i), \ i = 1 \dots m \tag{B.17}$$

Here p should be more than 0.5. We show that this problem can be converted to a SOCP:

Since $a_i \sim \mathcal{N}(\bar{a}_i, \Sigma_i)$, then $(a_i^T x - b_i) \sim \mathcal{N}(\bar{a}_i^T x - b_i, x^T \Sigma_i x)$. Standardize it:

$$t := ||\Sigma_i^{\frac{1}{2}} x||_2^{-1} \left\{(a_i^T x - b_i) - (\bar{a}_i^T x - b_i)\right\} \sim \mathcal{N}(0, 1)$$

Then,

$$\begin{aligned} \mathbb{P}(a_i^T x \leq b_i) &= \mathbb{P}(a_i^T x - b_i \leq 0) && \text{(B.18)} \\ &= \mathbb{P}(t \leq -||\Sigma_i^{\frac{1}{2}} x||_2^{-1}(\bar{a}_i^T x - b_i)) && \text{(B.19)} \\ &= \Phi(-||\Sigma_i^{\frac{1}{2}} x||_2^{-1}(\bar{a}_i^T x - b_i)) && \text{(B.20)} \end{aligned}$$

Here $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution:

$$\Phi(\xi) = \int_{-\infty}^{\xi} e^{-\frac{1}{2}t^2}\, dt$$

Thus,

$$\begin{aligned} & \mathbb{P}(a_i^T x \leq b_i) \geq p && \text{(B.21)} \\ \Longleftrightarrow & \Phi(-||\Sigma_i^{\frac{1}{2}} x||_2^{-1}(\bar{a}_i^T x - b_i)) \geq p && \text{(B.22)} \\ \Longleftrightarrow & -||\Sigma_i^{\frac{1}{2}} x||_2^{-1}(\bar{a}_i^T x - b_i) \geq \Phi^{-1}(p) && \text{(B.23)} \\ \Longleftrightarrow & \Phi^{-1}(p)||\Sigma_i^{\frac{1}{2}} x||_2 \leq b_i - \bar{a}_i^T x && \text{(B.24)} \end{aligned}$$

which is exactly the same as inequality constraints in SOCP formulation. (You can see why we enforce $p > 0.5$ here: otherwise $\Phi^{-1}(p)$ will be negative and the constraint with not be an second-order cone.)

In the following code example, we set up four inequality constraints and let $\bar{a}_i^T x \leq b_i,\ i = 1 \ldots 4$ form an square located at the origin of size 2. Then, for convenience, we set $\Sigma_i \equiv \sigma^2 I$.

```
%% Define the SOCP example setting
bar_a1 = [1; 0];
b1 = 1;
bar_a2 = [0; 1];
b2 = 1;
bar_a3 = [-1; 0];
b3 = 1;
bar_a4 = [0; -1];
b4 = 1;
sigma = 0.1;
c = [2; 3];
p = 0.9; % p should be more than 0.5
Phi_inv = norminv(p); % get Phi^{-1}(p)
```

```
%% Solve the SOCP problem
cvx_begin
    variable x(2); % define variables [x1; x2]
    minimize(c' * x);
    subject to
        sigma*Phi_inv * norm(x) <= b1 - bar_a1' * x;
        sigma*Phi_inv * norm(x) <= b2 - bar_a2' * x;
        sigma*Phi_inv * norm(x) <= b3 - bar_a3' * x;
        sigma*Phi_inv * norm(x) <= b4 - bar_a4' * x;
cvx_end
```

B.2.6 Semidefinite Programming (SDP)

Definition. An SDP has the following form:

$$\min_{X_i, x_i} \sum_{i=1}^{n_s} C_i \cdot X_i + \sum_{i=1}^{n_u} c_i \cdot x_i \tag{B.25}$$

$$\text{subject to } \sum_{i=1}^{n_s} A_{i,j} \cdot X_i + \sum_{i=1}^{n_u} a_{i,j} \cdot x_i = b_j, \quad j = 1 \dots m \tag{B.26}$$

$$X_i \in \mathcal{S}_+^{D_i}, \quad i = 1 \dots n_s \tag{B.27}$$

$$x_i \in \mathbb{R}^{d_i}, \quad i = 1 \dots n_u \tag{B.28}$$

where $C_i, A_{i,j} \in \mathbb{R}^{D_i \times D_i}$, $c_i, a_{i,j} \in \mathbb{R}^{d_i}$, and $\cdot$ means element-wise product. For two square matrices A, B, the dot product $A \cdot B$ is equal to $\text{tr}(AB)$; for two vectors a, b, the dot product $a \cdot b$ is the same as inner product $a^T b$.

Note that actually there are many "standard" forms of SDP. For example, in the convex optimization theory part, you may find an SDP that looks like:

$$\min_X C \cdot X \tag{B.29}$$

$$\text{subject to } A \cdot X = b \tag{B.30}$$

$$X \succeq 0 \tag{B.31}$$

It is convenient for us to analyze the theoretical properties of SDP with this form. Also, in SDP solvers' User Guide, you may see more complex SDP forms which involve more general convex cones. For example, see MOSEK's MATLAB API docs. Here we turn to use the form of (B.25) for two reasons: (1) it is general enough: our SDP example below can be converted to this form (also, SDPs from sum-of-squares programming in this book are exactly of the form (B.25)); (2) it is more readable than more complex forms.

Example. We consider the problem of finding the minimum eigenvalue for a positive semidefinite matrix S. We will show that this problem can be converted

to (B.25). Since S is positive semidefinite, the finding procedure can be cast as

$$\max_{\lambda} \lambda \tag{B.32}$$

$$\text{subject to } S - \lambda I \succeq 0 \tag{B.33}$$

Now define an auxiliary matrix $X := S - \lambda I$. We have

$$\min_{\lambda, X} \ -\lambda \tag{B.34}$$

$$\text{subject to } X + \lambda I = S \tag{B.35}$$

$$X \succeq 0 \tag{B.36}$$

It is obvious that the linear matrix equality constraint $X+\lambda I = S$ can be divided into several linear scalar equality constraints in (B.25). For example, we consider $S \in \mathbb{S}^3_+$. Thereby $X + \lambda I = S$ will lead to 6 linear equality constraints (We don't consider X is a symmetric matrix here, since most solvers will implicitly consider this. Thus, only the upper-triangular part of X and S are actually used in the equality construction.):

$$\begin{bmatrix}1&0&0\\0&0&0\\0&0&0\end{bmatrix} \cdot X + \lambda = S[0,0], \begin{bmatrix}0&1&0\\0&0&0\\0&0&0\end{bmatrix} \cdot X = S[0,1], \begin{bmatrix}0&0&1\\0&0&0\\0&0&0\end{bmatrix} \cdot X = S[0,2] \tag{B.37}$$

$$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} \cdot X + \lambda = S[1,1], \begin{bmatrix}0&0&0\\0&0&1\\0&0&0\end{bmatrix} \cdot X = S[1,2], \begin{bmatrix}0&0&0\\0&0&0\\0&0&1\end{bmatrix} \cdot X + \lambda = S[2,2] \tag{B.38}$$

Seems tedious? Fortunately, CVX provides a high-level API to handle these linear equality constraints: you just need to write down

```
X + lam * eye(3) == S; % linear equality constraints: X + lam *I = S
```

CVX will autometically convert this high-level constraint to (B.25) and pass them to the underlying solver.

To generate a ramdom $S \in \mathcal{S}^3_+$, you just need to assign three nonnegative eigenvalues to the program. After that, an random S will be generated by $S = Q\ \mathrm{diag}(\lambda_1, \lambda_2, \lambda_3)\ Q^T$, where Q is random orthonormal matrix.

```
%% Define the SDP example setting
lam_list = [0.7; 2.4; 3.7];
S = generate_random_PD_matrix(lam_list); % get a PD matrix S
```

```
%% Solve the SDP problem
cvx_begin
    variable X(3, 3) symmetric;
    variable lam;
    maximize(lam);
    subject to
        % here "==" should be read as "is in"
        X == semidefinite(3);
        X + lam * eye(3) == S;
cvx_end

% this function help to generate PD matrix of size 3*3
% if you provide the eigenvalues [lam_1, lam_2, lam_3]
function S = generate_random_PD_matrix(lam_list)
    if ~all(lam_list >= 0) % all eigenvalues >= 0
        error("All eigenvalues must be nonnegative.");
    end
    D = diag(lam_list);
    % use QR factorization to generate a random orthonormal matrix Q
    [Q, ~] = qr(rand(3, 3));
    S =  Q * D * Q';
end
```

B.2.7 CVXPY Introduction and Examples

Apart from CVX MATLAB, we also have a Python package called CVXPY, which functions almost the same as CVX MATLAB. To define and solve a convex optimization problem CVXPY, basically, there are three steps (apart from importing necessary packages):

- Step 1: Define parameters and variables in a certain type of convex problem. Here variables are what you are trying to optimize or "learn". Parameters are the "coefficients" of variables in the objective and constraints.
- Step 2: Define the objective function and constraints.
- Step 3: Solve the problem and get the results.

Here we provide the CVXPY codes for the above five convex optimization examples.

B.2.7.1 LP

```
import cvxpy as cp
import numpy as np

## Define the LP example setting
c1 = 2
c2 = -5
l1 = 3
l2 = 7

## Step 1: define variables and parameters
x = cp.Variable(2) # variable: x = [x1, x2]^T
# parameters: c, A, b
c = np.array([c1, c2])
A = np.array([[1, 0], [-1, 0], [0, 1], [0, -1]])
b = np.array([l1, l1, l2, l2])

## Step 2: define objective and constraints
obj = cp.Minimize(c.T @ x)
constraints = [A @ x <= b]
prob = cp.Problem(obj, constraints) # form the problem

## Step 3: solve problem and get results
prob.solve()
print("status: ", prob.status) # check whether the status is "optimal"
print("optimal value: ", prob.value) # optimal objective
print("optimal solution: ", x.value) # optimal x
```

B.2.7.2 QP

```
import cvxpy as cp
import numpy as np

## Define the LP example setting
p1 = 2
p2 = 0.5
p3 = 4
q1 = -3
q2 = -6.5
l1 = 2
```

```
l2 = 2.5
# check if the generated P is positive semidefinite
tmp1 = (p1 >= 0)
tmp2 = (p1*p3 - 4*p2**2 >= 0)
assert(tmp1 and tmp2, "P is not positve semidefinite!")

## Step 1: define variables and parameters
x = cp.Variable(2) # variable: x = [x1, x2]^T
# parameters: P, q, G, h
P = 2*np.array([[p1, p2], [p2, p3]])
q = np.array([q1, q2])
G = np.array([[1, 0], [-1, 0], [0, 1], [0, -1]])
h = np.array([l1, l1, l2, l2])

## Step 2: define the objective and constraints
fx = 0.5 * cp.quad_form(x, P) + q.T @ x
obj = cp.Minimize(fx)
constraints = [G @ x <= h]
prob = cp.Problem(obj, constraints) # form the problem

## Step 3: solve the problem and get results
prob.solve()
print("status: ", prob.status) # check whether the status is "optimal"
print("optimal value: ", prob.value) # optimal objective
print("optimal solution: ", x.value) # optimal x
```

B.2.7.3 QCQP

```
import cvxpy as cp
import numpy as np
from numpy.linalg import cholesky, inv, norm

## Define the QCQP example setting
def if_ellipse(K, k, c):
    # examine whether 0.5*x^T K x + k^T x + c <= 0 is a ellipse
    # if K is not positive semidefinite, Cholesky will raise an error
    L = cholesky(K)
    radius_square = 0.5 * norm(inv(L) @ k)**2 - c
    return radius_square > 0
K1 = np.eye(2)
k1 = np.zeros(2)
c1 = -0.5
```

```
K2 = np.array([[1, 0], [0, 1]])
k2 = np.array([2, 2])
c2 = 3.5
if not (if_ellipse(K1, k1, c1) and if_ellipse(K2, k2, c2)):
    raise ValueError("The example setting is not correct")

## Step 1: define variables and parameters
P0 = np.array([[1,0,-1,0], [0,1,0,-1], [-1,0,1,0], [0,-1,0,1]])
P1 = np.zeros((4,4))
P1[:2, :2] = K1
P2 = np.zeros((4,4))
P2[2:, 2:] = K2
q1 = np.concatenate([k1, np.zeros(2)])
q2 = np.concatenate([np.zeros(2), k2])
r1 = c1
r2 = c2

## Step 2: define objective and constraints
x = cp.Variable(4) # variable: x = [y1, z1, y2, z2]^T
fx = 0.5 * cp.quad_form(x, P0)
obj = cp.Minimize(fx)
con1 = (0.5 * cp.quad_form(x, P1) + q1.T @ x + r1 <= 0) # ellipse 1
con2 = (0.5 * cp.quad_form(x, P2) + q2.T @ x + r2 <= 0) # ellipse 2
constraints = [con1, con2]
prob = cp.Problem(obj, constraints) # form the problem

## Step 3: solve problem and get results
prob.solve()
print("status: ", prob.status) # check whether the status is "optimal"
print("optimal value: ", prob.value) # optimal objective
print("optimal solution: ", x.value) # optimal x
```

B.2.7.4 SOCP

```
import cvxpy as cp
import numpy as np
from scipy.stats import norm

## Define the SOCP example setting
# define bar_ai, bi (i = 1, 2, 3, 4)
bar_a1 = np.array([1, 0])
b1 = 1
```

```
bar_a2 = np.array([0, 1])
b2 = 1
bar_a3 = np.array([-1, 0])
b3 = 1
bar_a4 = np.array([0, -1])
b4 = 1
sigma = 0.1
c = np.array([2, 3])
p = 0.9 # p should be more than 0.5

## Step 1: define variables and parameters
Phi_inv = norm.ppf(p) # get Phi^{-1}(p)

## Step 2: define objective and constraints
x = cp.Variable(2) # variable: x = [x1, x2]^T
obj = cp.Minimize(c.T @ x)
# use cp.SOC(t, x) to create the SOC constraint ||x||_2 <= t
constraints = [
    cp.SOC(b1 - bar_a1.T @ x, sigma*Phi_inv*x),
    cp.SOC(b2 - bar_a2.T @ x, sigma*Phi_inv*x),
    cp.SOC(b3 - bar_a3.T @ x, sigma*Phi_inv*x),
    cp.SOC(b4 - bar_a4.T @ x, sigma*Phi_inv*x),
]
prob = cp.Problem(obj, constraints) # form the problem

## Step 3: solve problem and get results
prob.solve()
print("status: ", prob.status) # check whether the status is "optimal"
print("optimal value: ", prob.value) # optimal objective
print("optimal solution: ", x.value) # optimal x
```

B.2.7.5 SDP

```
import cvxpy as cp
import numpy as np
from scipy.stats import ortho_group

## Define the SDP example setting
# this function help to generate PD matrix of size 3*3
# if you provide the eigenvalues [lam_1, lam_2, lam_3]
def generate_random_PD_matrix(lam_list):
    assert np.all(lam_list >= 0) # all eigenvalues >= 0
```

```
    # S = Q @ D @ Q.T
    D = np.diag(lam_list)
    Q = ortho_group.rvs(3)
    return Q @ D @ Q.T
lam_list = np.array([0.5, 2.4, 3.7])
S = generate_random_PD_matrix(lam_list) # get a PD matrix S

## Step 1: define variables and parameters
# get coefficients for equality constraints
A_00 = np.array([[1, 0, 0], [0, 0, 0], [0, 0, 0]]) # tr(A_00 @ X) + lam = S_00
A_01 = np.array([[0, 1, 0], [0, 0, 0], [0, 0, 0]]) # tr(A_01 @ X) = S_01
A_02 = np.array([[0, 0, 1], [0, 0, 0], [0, 0, 0]]) # tr(A_02 @ X) = S_02
A_11 = np.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]]) # tr(A_11 @ X) + lam = S_11
A_12 = np.array([[0, 0, 0], [0, 0, 1], [0, 0, 0]]) # tr(A_12 @ X) = S_12
A_22 = np.array([[0, 0, 0], [0, 0, 0], [0, 0, 1]]) # tr(A_22 @ X) + lam = S_22

## Step 2: define objective and constraints
# define a PD matrix variable X of size 3*3
X = cp.Variable((3, 3), symmetric=True)
constraints = [X >> 0] # the operator >> denotes matrix inequality
lam = cp.Variable(1)
constraints += [
    cp.trace(A_00 @ X) + lam == S[0,0],
    cp.trace(A_01 @ X) == S[0,1],
    cp.trace(A_02 @ X) == S[0,2],
    cp.trace(A_11 @ X) + lam == S[1,1],
    cp.trace(A_12 @ X) == S[1,2],
    cp.trace(A_22 @ X) + lam == S[2,2],
]
obj = cp.Minimize(-lam)
prob = cp.Problem(obj, constraints) # form the problem

## Step 3: solve problem and get results
prob.solve()
print("status: ", prob.status) # check whether the status is "optimal"
print("optimal value: ", prob.value) # optimal objective
print("optimal solution: ", lam.value) # optimal lam
```

Appendix C

Linear System Theory

Thanks to Shucheng Kang for writing this Appendix.

C.1 Stability

C.1.1 Continuous-Time Stability

Consider the continuous-time linear time-invariant (LTI) system

$$\dot{x} = Ax. \tag{C.1}$$

the system is said to be "diagonalizable" if A is diagonalizable.

Definition C.1 (Asymptotic and Marginal Stability)**.** The diagonalizable, LTI system (C.1) is

1. "asymptotically stable" if $x(t) \to 0$ as $t \to \infty$ for every initial condition x_0
2. "marginally stable" if $x(t) \nrightarrow 0$ but remains bounded as $t \to \infty$ for every initial condition x_0
3. "stable" if it is either asymptotically or marginally stable
4. "unstable" if it is not stable

One can show that A's eigenvalues determine the LTI system's stability, as the following Theorem states:

Theorem C.1 (Stability of Continuous-Time LTI System)**.** *The diagonalizable*[1]*, LTI system* (C.1) *is*

1. *asymptotically stable if* $Re(\lambda_i) < 0$ *for all* i
2. *marginally stable if* $Re(\lambda_i) \leq 0$ *for all* i *and there exists at least one* i *for which* $Re(\lambda_i) = 0$
3. *stable if* $Re(\lambda_i) \leq 0$ *for all* i
4. *unstable if* $Re(\lambda_i) > 0$ *for at least one* i

Proof. Here we only represent the proof of (1). Similar procedure can be adopted for the proof of (2) - (4).

Since A is diagonalizable, there exists an similarity transformation matrix T, s.t. $A = T\Lambda T^{-1}$, where $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$. Then, under the coordinate transformation $z = T^{-1}x$, $\dot{x} = Ax$ can be restated as $\dot{z} = \Lambda z$. Consider the i's component of z:

$$\dot{z}_i = \lambda_i z_i \Longrightarrow z_i(t) = e^{\lambda_i t} z_i(0)$$

Since $\text{Re}(\lambda_i) < 0$, $z_i(t)$ will go to 0 as $t \to 0$ regardless how we choose $z_i(0)$.

□

C.1.2 Discrete-Time Stability

Now consider the diagonalizable, discrete-time linear time-invariant (LTI) system

$$x_{t+1} = Ax_t. \tag{C.2}$$

Theorem C.2 (Stability of Discrete-Time LTI System)**.** *The diagonalizable, discrete-time LTI system* (C.2) *is*

1. *asymptotically stable if* $|\lambda_i| < 1$ *for all* i
2. *marginally stable if* $|\lambda_i| \leq 1$ *for all* i *and there exists at least one* i *for which* $|\lambda_i| = 1$
3. *stable if* $|\lambda_i| \leq 1$ *for all* i
4. *unstable if* $|\lambda_i| > 1$ *for at least one* i*.*

Note that $|\lambda_i| < 1$ *means the eigenvalue lies strictly inside the unit circle in the complex plane.*

[1] when A is not diagonalizable, similar results can be derived via Jordan decomposition.

Proof. Here we only represent the proof of (1). Similar procedure can be adopted for the proof of (2) - (4).

Since A is diagonalizable, there exists an similarity transformation matrix T, s.t. $A = T\Lambda T^{-1}$, where $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$. Then, under the coordinate transformation $z = T^{-1}x$, $x_{t+1} = Ax$ can be restated as $z_{t+1} = \Lambda z_t$. Expanding the recursion, we have

$$z_t = \Lambda^{t-1} z_0 \Longrightarrow z_{t,i} = \lambda_i^{t-1} z_{0,i}$$

Since $|\lambda_i| < 1$, $z_{t,i}$ will go to 0 as $t \to 0$ regardless how we choose $z_{0,i}$. □

C.1.3 Lyapunov Analysis

Theorem C.3 (Lyapunov Equation)**.** *The following is equivalent for a linear time-invariant system $\dot{x} = Ax$*

1. *The system is globally asymptotically stable, i.e., A is Hurwitz and $\lim_{t\to\infty} x(t) = 0$ regardless of the initial condition;*
2. *For any positive definite matrix Q, the unique solution P to the Lyapunov equation*
$$A^T P + PA = -Q \tag{C.3}$$
is positive definite.

Proof. (a): $2 \Rightarrow 1$. Suppose we are given two positive definite matrices $P, Q \succ 0$ that satisfies the Lyapunov equation (C.3). Define a scalar function

$$V(x) = x^T P x.$$

It is clear that $V > 0$ for any $x \neq 0$ and $V(x) = 0$ (i.e., $V(x)$ is positive definite). We also see $V(x)$ is radially unbounded because:

$$V(x) \geq \lambda_{\min}(P)\|x\|^2 \Rightarrow \lim_{x\to\infty} V(x) \to \infty.$$

The time derivative of V reads

$$\dot{V} = 2x^T P\dot{x} = x^T(A^T P + PA)x = -x^T Q x.$$

Clearly, $\dot{V} < 0$ for any $x \neq 0$ and $\dot{V}(0) = 0$. According to Lyapunov's global stability theorem 5.3, we conclude the linear system $\dot{x} = Ax$ is globally asymptotically stable at $x = 0$.

(b): $1 \Rightarrow 2$. Suppose A is Hurwitz, we want to show that, for any $Q \succ 0$, there exists a unique $P \succ 0$ satisfying the Lyapunov equation (C.3). In fact, consider the matrix

$$P = \int_{t=0}^{\infty} e^{A^T t} Q e^{At} dt.$$

Because A is Hurwitz, the integral exists, and clearly $P \succ 0$ due to $Q \succ 0$. To show this choice of P satisfies the Lyapunov equation, we write

$$A^T P + PA = \int_{t=0}^{\infty} \left(A^T e^{A^T t} Q e^{At} + e^{A^T t} Q e^{At} A \right) dt \tag{C.4}$$

$$= \int_{t=0}^{\infty} d\left(e^{A^T t} Q e^{At} \right) \tag{C.5}$$

$$= e^{A^T t} Q e^{At}|_{t=\infty} - e^{A^T t} Q e^{At}|_{t=0} = -Q, \tag{C.6}$$

where the last equality holds because $e^{A\infty} = 0$ (recall A is Hurwitz).

To show the uniqueness of P, we assume that there exists another matrix P' that also satisfies the Lyapunov equation. Therefore,

$$P' = e^{A^T t} P' e^{At}|_{t=0} - e^{A^T t} P' e^{At}|_{t=\infty} \tag{C.7}$$

$$= -\int_{t=0}^{\infty} d\left(e^{A^T t} P' e^{At} \right) \tag{C.8}$$

$$= -\int_{t=0}^{\infty} e^{A^T t} \left(A^T P' + P' A \right) e^{At} dt \tag{C.9}$$

$$= \int_{t=0}^{\infty} e^{A^T t} Q e^{At} dt = P, \tag{C.10}$$

leading to $P' = P$. Hence, the solution is unique. □

Convergence rate estimation. We now show that Theorem C.3 can allow us to quantify the convergence rate of a (stable) linear system towards zero.

For a Hurwitz linear system $\dot{x} = Ax$, let us pick a positive definite matrix Q. Theorem C.3 tells us we can find a unique $P \succ 0$ satisfying the Lyapunov equation (C.3). In this case, we can upper bound the scalar function $V = x^T P x$ as

$$V \leq \lambda_{\max}(P) \|x\|^2.$$

The time derivative of V is $\dot{V} = -x^T Q x$, which can be upper bounded by

$$\dot{V} \leq -\lambda_{\min}(Q) \|x\|^2 \tag{C.11}$$

$$= -\frac{\lambda_{\min}(Q)}{\lambda_{\max}(P)} \underbrace{\left(\lambda_{\max}(P) \|x\|^2 \right)}_{\geq V} \tag{C.12}$$

$$\leq -\frac{\lambda_{\min}(Q)}{\lambda_{\max}(P)} V. \tag{C.13}$$

Denoting $\gamma(Q) = \frac{\lambda_{\min}(Q)}{\lambda_{\max}(P)}$, the above inequality implies

$$V(0) e^{-\gamma(Q) t} \geq V(t) = x^T P x \geq \lambda_{\min}(P) \|x\|^2.$$

As a result, $\|x\|^2$ converges to zero exponentially with a rate at least $\gamma(Q)$, and $\|x\|$ converges to zero exponentially with a rate at least $\gamma(Q)/2$.

Best convergence rate estimation. I have used $\gamma(Q)$ to make it explict that the rate γ depends on the choice of Q, because P is computed from the Lyapunov equation as an implicit function of Q. Naturally, choosing different Q will lead to different $\gamma(Q)$. So what is the choice of Q that maximizes the convergence rate estimation?

Corollary C.1 (Maximum Convergence Rate Estimation)**.** *$Q = I$ maximizes the convergence rate estimation.*

Proof. let us denote P_0 as the solution to the Lyapunov equation with $Q = I$

$$A^T P_0 + P_0 A = -I.$$

Let P be the solution corresponding to a different choice of Q

$$A^T P + PA = -Q.$$

Without loss of generality, we can assume $\lambda_{\min}(Q) = 1$, because rescaling Q will recale P by the same factor, which does not affect $\gamma(Q)$. Subtracting the two Lyapunov equations above we get

$$A^T(P - P_0) + (P - P_0)A = -(Q - I).$$

Since $Q - I \succeq 0$ (due to $\lambda_{\min}(Q) = 1$), we know $P - P_0 \succeq 0$ and $\lambda_{\max}(P) \geq \lambda_{\max}(P_0)$. As a result,

$$\gamma(Q) = \frac{\lambda_{\min}(Q)}{\lambda_{\max}(P)} = \frac{\lambda_{\min}(I)}{\lambda_{\max}(P)} \leq \frac{\lambda_{\min}(I)}{\lambda_{\max}(P_0)} = \gamma(I),$$

and $Q = I$ maximizes the convergence rate estimation. □

C.2 Controllability and Observability

Consider the following linear time-invariant (LTI) system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{C.14}$$

where $x \in \mathbb{R}^n$ the state, $u \in \mathbb{R}^m$ the control input, $y \in \mathbb{R}^p$ the output, and A, B, C, D are constant matrices with proper sizes. If we know the initial state

$x(0)$ and the control inputs $u(t)$ over a period of time $t \in [0, t_1]$, the system trajectory $(x(t), y(t))$ can be determined as

$$\begin{aligned} x(t) &= e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{C.15}$$

To study the internal structure of linear systems, two important properties should be considered: controllability and observability. In the following analysis, we will see that they are actually dual concepts. Their definitions (Chen, 1984) are given below.

Definition C.2 (Controllability)**.** The LTI system (C.14), or the pair (A, B), is controllable, if for any initial state $x(0) = x_0$ and final state x_f, there exists a sequence of control inputs that transfer the system from x_0 to x_f in finite time.

Definition C.3 (Observability)**.** The LTI system (C.14), or the pair (C, A), is observable, if for any unknown initial state $x(0)$, there exists a finite time $t_1 > 0$, such that knowing y and u over $[0, t_1]$ suffices to determine $x(0)$.

Sometimes it will become more convenient for us to analyze the system (C.14) under another coordinate basis, i.e., $z = Tx$, where the coordinate transformation T is nonsingular (i.e., full-rank). Define $A' = TAT^{-1}, B' = PB, C' = CT^{-1}, D' = D$, we get

$$\begin{aligned} \dot{z} &= A'z + B'u \\ y &= C'z + D'u \end{aligned}$$

Since the coordinate transformation only changes the system's coordinate basis, physical properties like controllability and observability will not change.

C.2.1 Cayley-Hamilton Theorem

In the analysis of controllability and observability, Cayley Hamilton Theorem lays the foundation. The statement of the theory and its (elegant) proof are given blow. Some useful corollaries are also presented.

Theorem C.4 (Cayley-Hamilton)**.** *Let $A \in \mathbb{C}^{n\times n}$ and denote the characteristic polynomial of A as*

$$det(\lambda I - A) = \lambda^n + a_1\lambda^{n-1} + \cdots + a_n \in \mathbb{C}[\lambda],$$

which is a polynomial in a single variable λ with coefficients $a_1, \ldots, a_n$. Then

$$A^n + a_1A^{n-1} + \cdots + a_nI = 0$$

Proof. Define the adjugate of $\lambda I - A$ as

$$B = \text{adj}(\lambda I - A)$$

From B's definition, we have

$$(\lambda I - A)B = \det(\lambda I - A)I = (\lambda^n + a_1\lambda^{n-1} + \cdots + a_n)I \tag{C.16}$$

Also, B is a polynomial matrix over λ, whose maximum degree is no more than $n-1$. Therefore, we write B as follows:

$$B = \sum_{i=0}^{n-1} \lambda^i B_i$$

where B_i's are constant matrices. In this way, we unfold $(\lambda I - A)B$:

$$\begin{aligned}(\lambda I - A)B &= (\lambda I - A)\sum_{i=0}^{n-1} \lambda^i B_i \\ &= \lambda^n B_{n-1} + \sum_{i=1}^{n-1} \lambda^i(-AB_i + B_{i-1}) - AB_0\end{aligned} \tag{C.17}$$

Since λ can be arbitrarily set, matching the coefficients of (C.16) and (C.17), we have

$$\begin{aligned} B_{n-1} &= I \\ -AB_i + B_{i-1} &= a_{n-i}I, \quad i = 1 \dots n-1 \\ -AB_0 &= a_n I \end{aligned}$$

Thus, we have

$$\begin{aligned} &B_{n-1} \cdot A^n + \sum_{i=1}^{n-1}(-AB_i + B_{i-1}) \cdot A^i + (-AB_0) \cdot I \\ =&I \cdot A^n + \sum_{i=1}^{n-1}(a_{n-i}I) \cdot A^i + (a_n I) \cdot I \\ =&A^n + a_1 A^{n-1} + a_2 A^{n-2} + \cdots + a_n I \end{aligned}$$

On the other hand, one can easily check that

$$B_{n-1} \cdot A^n + \sum_{i=1}^{n-1}(-AB_i + B_{i-1}) \cdot A^i + (-AB_0) \cdot I = 0$$

since each term offsets completely. Therefore,

$$A^n + a_1 A^{n-1} + a_2 A^{n-2} + \cdots + a_n I = 0,$$

concluding the proof. □

Here are some corollaries of the Cayley-Hamilton Theorem.

Corollary C.2. *For any $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times m}, k \geq n$, $A^k B$ is a linear combination of $B, AB, A^2B, \dots, A^{n-1}B$.*

Proof. Directly from Cayley Hamilton Theorem, A^n can be expressed as a linear combination of $I, A, A^2, \dots, A^{n-1}$. By recursion, it is easy to show that for all $m > n$, A^m is also a linear combination of $I, A, A^2, \dots, A^{n-1}$. Post-multiply both sides with B, we get what we want. □

Corollary C.3. *For any $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times m}, k > n$, the following equality always holds:*

$$\mathit{rank}(\begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix}) = \mathit{rank}(\begin{bmatrix} B & AB & \dots & A^{k-1}B \end{bmatrix})$$

Proof. First prove LHS $\leq$ RHS. $\forall v \in \mathbb{C}^n$ such that

$$v^* \begin{bmatrix} B & AB & \dots & A^{k-1}B \end{bmatrix} = v^* \begin{bmatrix} B & AB & \dots & A^{n-1}B & \dots A^{k-1}B \end{bmatrix} = 0$$

$v^* \begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix} = 0$ must hold.

Second prove LHS $\geq$ RHS. For any $v \in \mathbb{C}^n$ such that $v^* \begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix} = 0$ and any $k > n$, by Corollary C.2, there exists a sequence $c_i, i = 0 \dots n-1$ satisfy the following:

$$v^* A^k B = v^* \sum_{i=0}^{n-1} c_i A^i B = 0$$

Therefore, $v^* \begin{bmatrix} B & AB & \dots & A^{k-1}B \end{bmatrix} = 0$. □

Corollary C.4. *For any $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times m}$, define*

$$\mathcal{C} = \begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix}$$

If $\mathit{rank}(\mathcal{C}) = k_1 < n$, there exist a similarity transformation T such that

$$TAT^{-1} = \begin{bmatrix} \bar{A}_c & \bar{A}_{12} \\ 0 & \bar{A}_{\bar{c}} \end{bmatrix}, TB = \begin{bmatrix} \bar{B}_c \\ 0 \end{bmatrix}$$

where $\bar{A}_c \in \mathbb{C}^{k_1\times k_1}, \bar{B}_c \in \mathbb{C}^{k_1\times m}$. Moreover, the matrix

$$\bar{\mathcal{C}} := \begin{bmatrix} \bar{B}_c & \bar{A}_c\bar{B}_c & \bar{A}_c^2\bar{B}_c & \dots & \bar{A}_c^{k_1-1}\bar{B}_c \end{bmatrix}$$

has full row rank.

Proof. Since $\mathcal{C}$ is not full row rank, we pick k_1 linearly independent columns from $\mathcal{C}$. Denote them as $q_1 \dots q_{k_1}, q_i \in \mathbb{C}^n$. Then, we arbitrarily set other $n-k_1$ vectors $q_{k_1+1} \dots q_n$ as long as

$$Q = \begin{bmatrix} q_1 & \dots & q_{k_1} & q_{k_1+1} & \dots & q_n \end{bmatrix}$$

is invertible. Define the similarity transformation matrix by $T = Q^{-1}$. Note that Aq_i can be seen as a column picked from $A^k B, k \in \{1 \dots n\}$, which is guaranteed to be a linear combination of $B, AB, \dots, A^{n-1}B$ from Cayley Hamilton Theorem. Thus, Aq_i is bound to be a linear transformation of columns from $\begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix} = \mathcal{C}$. Since $q_1 \dots q_{k_1}$ is the largest linearly independent column vector set from $\mathcal{C}$, this implies Aq_i can be expressed as a linear combination of $q_1 \dots q_{k_1}$:

$$\begin{aligned} AQ = AT^{-1} &= A \begin{bmatrix} q_1 & \dots & q_{k_1} & q_{k_1+1} & \dots & q_n \end{bmatrix} \\ &= \begin{bmatrix} q_1 & \dots & q_{k_1} & q_{k_1+1} & \dots & q_n \end{bmatrix} \begin{bmatrix} \bar{A}_c & \bar{A}_{12} \\ 0 & \bar{A}_{\bar{c}} \end{bmatrix} = T^{-1} \begin{bmatrix} \bar{A}_c & \bar{A}_{12} \\ 0 & \bar{A}_{\bar{c}} \end{bmatrix} \end{aligned}$$

Similarly, B itself is part of $\mathcal{C}$. Therefore, each column of B is naturally a linear combination of $q_1 \dots q_{k_1}$:

$$B = \begin{bmatrix} q_1 & \dots & q_{k_1} & q_{k_1+1} & \dots & q_n \end{bmatrix} \begin{bmatrix} \bar{B}_c \\ 0 \end{bmatrix} = T^{-1} \begin{bmatrix} \bar{B}_c \\ 0 \end{bmatrix}$$

To see $\bar{\mathcal{C}}$ has full row rank, note that $\text{rank}\mathcal{C} = k_1$ and

$$\mathcal{C} = T^{-1} \begin{bmatrix} \bar{B}_c & \bar{A}_c\bar{B}_c & \bar{A}_c^2\bar{B}_c & \dots & \bar{A}_c^{k_1-1}\bar{B}_c & \dots & \bar{A}_c^{n-1}\bar{B}_c \\ 0 & 0 & 0 & \dots & 0 & \dots & 0 \end{bmatrix}$$

Thus,

$$\text{rank} \begin{bmatrix} \bar{B}_c & \bar{A}_c\bar{B}_c & \bar{A}_c^2\bar{B}_c & \dots & \bar{A}_c^{k_1-1}\bar{B}_c & \dots & \bar{A}_c^{n-1}\bar{B}_c \end{bmatrix} = k_1.$$

By Corollary C.3, $\text{rank}\bar{\mathcal{C}} = k_1$. □

The following Corollary is especially useful in the study of pole assignment in the single-input-multiple-output (SIMO) LTI system.

Corollary C.5. *For any $A \in \mathbb{C}^{n\times n}, b \in \mathbb{C}^n$, if*

$$\mathcal{C} = \begin{bmatrix} b & Ab & \dots & A^{n-1}b \end{bmatrix} \in \mathbb{C}^{n\times n}$$

has full rank, then there exists a similarity transformation T such that

$$TAT^{-1} = A_1 := \begin{bmatrix} -a_1 & -a_2 & \dots & -a_{n-1} & -a_n \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}, \quad Tb = b_1 := \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $a_1, \dots, a_n$ are the coefficients of A's characteristic polynomial:

$$\det(A - \lambda I) = \lambda^n + a_1\lambda^{n-1} + \dots + a_n\lambda$$

Proof. Since $\mathcal{C}$ is invertible, define its inverse

$$\mathcal{C}^{-1} = \begin{bmatrix} M_1 \\ M_2 \\ ... \\ M_n \end{bmatrix}$$

where $M_i \in \mathbb{C}^{1\times n}$. Then,

$$I = \mathcal{C}^{-1}\mathcal{C} = \begin{bmatrix} M_1 b & M_1 Ab & ... & M_1 A^{n-1}b \\ M_2 b & M_2 Ab & ... & M_2 A^{n-1}b \\ \vdots & \vdots & & \vdots \\ M_n b & M_n Ab & ... & M_n A^{n-1}b \end{bmatrix} \Longrightarrow \begin{cases} M_n A^{n-1}b = 1 \\ M_n A^i b = 0,\ i = 0, ..., n-2 \end{cases}$$

Now we claim that the transformation matrix T can be constructed as follows:

$$T = \begin{bmatrix} M_n A^{n-1} \\ M_n A^{n-2} \\ ... \\ M_n \end{bmatrix}$$

We first show T is invertible by calculating $T\mathcal{C}$:

$$T\mathcal{C} = \begin{bmatrix} M_n A^{n-1}b & \star & ... & \star \\ M_n A^{n-2}b & M_n A^{n-1}b & ... & \star \\ \vdots & \vdots & & \vdots \\ M_n b & M_n Ab & ... & M_n A^{n-1}b \end{bmatrix} = \begin{bmatrix} 1 & \star & ... & \star \\ 0 & 1 & ... & \star \\ \vdots & \vdots & & \vdots \\ 0 & 0 & ... & 1 \end{bmatrix}$$

Then we calculate Tb and TA:

$$Tb = \begin{bmatrix} M_n A^{n-1}b \\ M_n A^{n-2}b \\ \vdots \\ M_n b \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$TA = \begin{bmatrix} M_n A^n \\ M_n A^{n-1} \\ \vdots \\ M_n A \end{bmatrix} = \begin{bmatrix} -M_n \cdot \sum_{i=0}^{n-1} a_{n-i} A^i \\ M_n A^{n-1} \\ \vdots \\ M_n A \end{bmatrix}$$

$$= \begin{bmatrix} -a_1 & -a_2 & ... & -a_{n-1} & -a_n \\ 1 & 0 & ... & 0 & 0 \\ 0 & 1 & ... & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & ... & 1 & 0 \end{bmatrix} \begin{bmatrix} M_n A^{n-1} \\ M_n A^{n-2} \\ \vdots \\ M_n A \\ M_n \end{bmatrix} = A_1 T$$

where the penultimate equality uses Cayley Hamilton Theorem. □

C.2.2 Equivalent Statements for Controllability

There are a few equivalent statements to express an LTI system's controllability that one should be familiar with:

Theorem C.5 (Equivalent Statements for Controllability)**.** *The following statements are equivalent (Chen, 1984), (Zhou et al., 1996):*

1. (A, B) *is controllable.*
2. *The matrix*
$$W_c(t) := \int_0^t e^{A\tau} BB^* e^{A^*\tau} d\tau$$
is positive definite for any $t > 0$.
3. *The controllability matrix*
$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}$$
has full row rank.
4. *The matrix* $[A - \lambda I, B]$ *has full row rank for all* $\lambda \in \mathbb{C}$.
5. *Let* λ *and* x *be any eigenvalue and any corresponding left eigenvector* A*, i.e.,* $x^*A = x^*\lambda$*, then* $x^*B \neq 0$.
6. *The eigenvalues of* $A + BF$ *can be freely assigned (with the restriction that complex eigenvalues are in conjugate pairs) by a suitable choice of* F.
7. *If, in addition, all eigenvalues of* A *have negative real parts, then the unique solution of*
$$AW_c + W_cA^* = -BB^*$$
is positive definite. The solution is called the controllability Gramian *and can be expressed as*
$$W_c = \int_0^\infty e^{A\tau} BB^* e^{A^*\tau} d\tau$$

Proof. (1. $\Rightarrow$ 2.) Prove by contradiction. Assume that (A, B) is controllable but $W_c(t_1)$ is singular for some $t_1 > 0$. This implies there exists a real vector $v \neq 0 \in \mathbb{R}^n$, s.t.

$$v^* W_c(t_1) v = v^* \left(\int_0^{t_1} e^{At} BB^* e^{A^* t} dt \right) v = \int_0^{t_1} v^* (e^{At} BB^* e^{A^* t}) v \, dt = 0$$

Since $e^{At} BB^* e^{A^* t} \succeq 0$ for all t, we must have

$$\begin{aligned} & v^* (e^{At} BB^* e^{A^* t}) v = \| \, v^* B e^{At} \, \|^2 = 0, \quad \forall t \in [0, t_1] \\ \Longrightarrow & v^* B e^{At} = 0, \quad \forall t \in [0, t_1] \end{aligned}$$

Setting $x(t_1) = 0$, from (C.15), we have

$$0 = e^{At_1}x(0) + \int_0^{t_1} e^{A(t_1-\tau)}Bu(\tau)d\tau = 0$$

Pre-multiply the above equation by v^*, then

$$0 = v^* e^{At_1}x(0)$$

Since $x(0)$ can be chosen arbitrarily, we set $x(0) = ve^{-At_1}$, which results in $v = 0$. Contradiction!

(2. $\Rightarrow$ 1.) For any $x(0) = x_0, t_1 > 0, x(t_1) = x_1$, since $W_c(t_1) \succ 0$, we set the control inputs as

$$u(t) = -B^* e^{A^*(t_1-t)}W_c^{-1}(t_1)[e^{At_1}x_0 - x_1]$$

We claim that the picked $u(t)$ satisfies (C.15) by

$$\begin{aligned}
&e^{At}x_0 + \int_0^{t_1} e^{A(t_1-t)}Bu(t)dt \\
&= e^{At}x_0 - \int_0^{t_1} e^{A(t_1-t)}BB^*e^{A^*(t_1-t)}dt \cdot W_c^{-1}(t_1)[e^{At_1}x_0 - x_1] \\
&\overset{\tau=t_1-t}{=} e^{At}x_0 - \underbrace{\int_0^{t_1} e^{A\tau}BB^*e^{A^*\tau}d\tau}_{W_c(t_1)} \cdot W_c^{-1}(t_1)[e^{At_1}x_0 - x_1] \\
&= e^{At}x_0 - [e^{At_1}x_0 - x_1] = x_1
\end{aligned}$$

(2. $\Rightarrow$ 3.) Prove by contradiction. Suppose $W_c(t) \succ 0, \forall t > 0$ but $\mathcal{C}$ is not of full row rank. Then there exists $v \neq 0 \in \mathbb{C}^n$, s.t.

$$v^*A^kB = 0, \quad k = 0 \dots n-1$$

By Corollary C.2, we have

$$v^*A^kB = 0,\ \forall k \in \mathbb{N} \Longrightarrow v^*e^{At}B = 0,\ \forall t > 0$$

which implies

$$v^*W_c(t)v = v^*(\int_0^t e^{A\tau}BB^*e^{A^*\tau}d\tau)v = 0, \quad \forall t > 0$$

Contradiction!

(3. $\Rightarrow$ 2.) Prove by contradiction. Suppose $\mathcal{C}$ has full row rank but $W_c(t_1)$ is singular at some $t_1 > 0$. Then, similar to the proof in (1. $\Rightarrow$ 2.), there exists $v \neq 0 \in \mathbb{C}^n$, s.t. $F(t) := v^*e^{At}B \equiv 0, \forall t \in [0, t_1]$. Since $F(t)$ is infinitely

differentiable, we get its i's derivative at $t = 0$, where $i = 0, 1, ...\, n-1$. This results in

$$\left.\frac{d^i F}{dt^i}\right|_{t=0} = v^* A^i e^{At} B\big|_{t=0} = v^* A^i B = 0, \quad i = 0 \ldots n-1$$

Thus, $v^* \begin{bmatrix} B & AB & ... & A^{n-1}B \end{bmatrix} = 0$. Contradiction!

(3. $\Rightarrow$ 4.) Proof by contradiction. Suppose $[A - \lambda I, B]$ does not have full row rank for some $\lambda \in \mathbb{C}$. Then, there exists $v \neq 0 \in \mathbb{C}^n$, s.t. $v^*[A - \lambda I, B] = 0$. This implies $v^* A = v^* \lambda$ and $v^* B = 0$. On the other hand,

$$v^* \begin{bmatrix} B & AB & ... & A^{n-1}B \end{bmatrix} = v^* \begin{bmatrix} B & \lambda B & ... & \lambda^{n-1}B \end{bmatrix} = 0$$

Contradiction!

(4. $\Rightarrow$ 5.) Proof by contradiction. If there exists a left eigenvector and eigenvalue pair (x, λ), s.t. $x^* A = \lambda x^*$ while $x^* B = 0$, then $x^*[A - \lambda I, B] = 0$. Contradiction!

(5. $\Rightarrow$ 3.) Proof by contradiction. If the controllability matrix $\mathcal{C}$ does not have full row rank, i.e., $\text{rank}(\mathcal{C}) = k < n$. Then, from Corollary C.4, there exists a similarity transformation T, s.t.

$$TAT^{-1} = \begin{bmatrix} \bar{A}_c & \bar{A}_{12} \\ 0 & \bar{A}_{\bar{c}} \end{bmatrix}, \quad TB = \begin{bmatrix} \bar{B}_c \\ 0 \end{bmatrix}$$

where $\bar{A}_c \in \mathbb{R}^{k \times k}, \bar{A}_{\bar{c}} \in \mathbb{R}^{(n-k)\times(n-k)}$. Now arbitrarily pick one of $\bar{A}_{\bar{c}}$'s left eigenvector $x_{\bar{c}}$ and its corresponding eigenvalue λ_1. Define the vector $x = \begin{bmatrix} 0 \\ x_{\bar{c}} \end{bmatrix}$. Then,

$$x^*(TAT^{-1}) = \begin{bmatrix} 0 & x_{\bar{c}}^* \end{bmatrix} \begin{bmatrix} \bar{A}_c & \bar{A}_{12} \\ 0 & \bar{A}_{\bar{c}} \end{bmatrix} = \begin{bmatrix} 0 & x_{\bar{c}}^* \bar{A}_{\bar{c}} \end{bmatrix} = \begin{bmatrix} 0 & \lambda_1 x_{\bar{c}}^* \end{bmatrix} = \lambda_1 x^*$$

$$x^*(TB) = \begin{bmatrix} 0 & x_{\bar{x}} \end{bmatrix} \begin{bmatrix} B_{\bar{c}} \\ 0 \end{bmatrix} = 0$$

which implies (TAT^{-1}, TB) is not controllable. However, similarity transformation does not change controllability. Contradiction!

(6. $\Rightarrow$ 1.) Prove by contradiction. If (A, B) is not controllable, i.e., $\text{rank}(\mathcal{C}) = k < n$. Then from Corollary C.4, there exists a similarity transformation T s.t.

$$TAT^{-1} = \begin{bmatrix} \bar{A}_c & \bar{A}_{12} \\ 0 & \bar{A}_{\bar{c}} \end{bmatrix}, \quad TB = \begin{bmatrix} \bar{B}_c \\ 0 \end{bmatrix}$$

Now arbitrarily pick $F \in \mathbb{R}^{m \times n}$ and define $FT^{-1} = [F_1, F_2]$, where $F_1 \in$

$\mathbb{R}^{m\times k}, F_2 \in \mathbb{R}^{m\times(n-k)}$. Thus,

$$\begin{aligned}
\det(A+BF-\lambda I) &= \det\left(T^{-1}\begin{bmatrix}\bar{A}_c & \bar{A}_{12}\\ 0 & \bar{A}_{\bar{c}}\end{bmatrix}T + T^{-1}\begin{bmatrix}\bar{B}_c\\ 0\end{bmatrix}F - \lambda\begin{bmatrix}I_1 & 0\\ 0 & I_2\end{bmatrix}\right)\\
&= \det\left(T^{-1}\left\{\begin{bmatrix}\bar{A}_c & \bar{A}_{12}\\ 0 & \bar{A}_{\bar{c}}\end{bmatrix} + \begin{bmatrix}\bar{B}_c\\ 0\end{bmatrix}FT^{-1} - \lambda\begin{bmatrix}I_1 & 0\\ 0 & I_2\end{bmatrix}\right\}T\right)\\
&= \det\left(\begin{bmatrix}\bar{A}_c & \bar{A}_{12}\\ 0 & \bar{A}_{\bar{c}}\end{bmatrix} + \begin{bmatrix}\bar{B}_c\\ 0\end{bmatrix}\begin{bmatrix}F_1 & F_2\end{bmatrix} - \lambda\begin{bmatrix}I_1 & 0\\ 0 & I_2\end{bmatrix}\right)\\
&= \det\begin{bmatrix}\bar{A}_c + \bar{B}_cF_1 - \lambda I_1 & \bar{A}_{12} + \bar{B}_cF_2\\ 0 & \bar{A}_{\bar{c}} - \lambda I_2\end{bmatrix}\\
&= \det(\bar{A}_c + \bar{B}_cF_1 - \lambda I_1)\cdot\det(\bar{A}_{\bar{c}} - \lambda I_2)
\end{aligned}$$

where I_1 is the identity matrix of size k. Similarly, I_2 of size $n-k$. Thus, at least $n-k$ eigenvalues of $A+BF$ cannot be freely assigned by choosing F. Contradiction!

(1. $\Rightarrow$ 6.) Here we only represent the SIMO case. For the MIMO case, the proof is far more complex. Interesting readers can refer to (Davison and Wonham, 1968) (the shortest proof I can find). Since there is only one input, the matrix B degenerate to vector b. From Corollary C.5, there exist a similarity transformation matrix T, s.t.

$$TAT^{-1} = A_1 := \begin{bmatrix}-a_1 & -a_2 & \dots & -a_{n-1} & -a_n\\ 1 & 0 & \dots & 0 & 0\\ 0 & 1 & \dots & 0 & 0\\ \vdots & \vdots & & \vdots & \vdots\\ 0 & 0 & \dots & 1 & 0\end{bmatrix}, \quad Tb = b_1 := \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}$$

For any $F \in \mathbb{C}^{1\times n}$, denote FT^{-1} as $[f_1, f_2, \dots, f_n]$. Calculating the characteristic polynomial of $A+bF$:

$$\begin{aligned}
\det(\lambda I - A - bF) &= \det(\lambda I - T^{-1}A_1T - T^{-1}b_1F)\\
&= \det(\lambda I - A_1 - b_1FT^{-1})\\
&= \det\begin{bmatrix}\lambda + a_1 - f_1 & \lambda + a_2 - f_2 & \dots & \lambda + a_{n-1} - f_{n-1} & \lambda + a_n - f_n\\ -1 & \lambda & \dots & 0 & 0\\ 0 & -1 & \dots & 0 & 0\\ \vdots & \vdots & & \vdots & \vdots\\ 0 & 0 & \dots & -1 & \lambda\end{bmatrix}\\
&= \lambda^n + (a_1 - f_1)\lambda^{n-1} + \cdots + (a_n - f_n)
\end{aligned}$$

By choosing $[f_1, f_2, \dots, f_n]$, $A+bF$'s eigenvalues can be arbitrarily set.

(7. $\Rightarrow$ 1.) Prove by contradiction. Assume that (A, B) is not controllable. Then from 2., there exists $v \neq 0 \in \mathbb{C}^n$ and $t_1 > 0$,

$$F(t) = v^*e^{At}B = 0, \quad \forall t \in [0, t_1]$$

Now consider $F(z) = v^* e^{Az} B, z \in \mathcal{C}$, which is a vector of analytic function in complex analysis. For a arbitrary $t_2 \in (0, t_1)$, we have $F^{(i)}(t_2) = 0, \forall i \in \mathbb{N}$. Then, by invoking the fact from complex analysis: "Let G a connected open set and $f : G \to \mathbb{C}$ be analytic, then $f \equiv 0$ on G, if and only if there is a point $a \in G$ such that $f^{(i)}(a) = 0, \forall n \in \mathbb{N}$", we have $f(z) \equiv 0, \forall z \in \mathbb{C}$.

On the other hand, however, $W_c \succ 0$ implies there exists $t_3 > 0$, such that for the above v, we have $v^* e^{At_3} B \neq 0$. Contradiction!

(1. $\Rightarrow$ 7.) Since (A, B) is controllable, from 2., $W_c(t) \succ 0, \forall t$. Therefore, $W_c \succ 0$. The existence and uniqueness of the solution for $AW_c + W_c A^* = -BB^*$ can be obtained directly from the proof of Theorem C.3, by setting Q there to be positive semidefinite. □

C.2.3 Duality

Although controllability and observability seemingly have no direct connections from their definitions C.2 and C.3, the following theorem (Chen, 1984) states their tight relations.

Theorem C.6 (Theorem of Duality)**.** *The pair (C, A) is observable if and only if (A^*, C^*) is controllable.*

Proof.

(1) We first show that (C, A) is observable if and only if the $n \times n$ matrix $W_o(t) = \int_0^t e^{A^*\tau} C^* C e^{A\tau}$ is positive definite (nonsingular) for any $t > 0$:

"$\Longleftarrow$": From (C.15), given initial state $x(0)$ and the inputs $u(t)$, $y(t)$ can be expressed as

$$y(t) = Ce^{At}x(0) + C\int_0^t e^{A(t-\tau)}Bu(\tau)d\tau + Du(t)$$

Define a known function $\bar{y}(t)$ as $y(t) - C\int_0^t e^{A(t-\tau)}Bu(\tau)d\tau - Du(t)$ and we will get

$$Ce^{At}x(0) = \bar{y}(t)$$

Pre-multiply the above equation by $e^{A^*t}C^*$ and integrate it over $[0, t_1]$ to yield

$$(\int_0^{t_1} e^{A^*t}C^*Ce^{At}dt)x(0) = W_o(t_1)x(0) = \int_0^{t_1} e^{A^*t}C^*\bar{y}(t)dt$$

Since $W_o(t_1) \succ 0$,

$$x(0) = W_o(t_1)^{-1}\int_0^{t_1} e^{A^*t}C^*\bar{y}(t)dt$$

can be observed.

"$\Longrightarrow$": Prove by contradiction. Suppose (C, A) is observable but there exists $t_1 > 0$, s.t. $W_o(t_1)$ is singular. This implies there exists $v \neq 0 \in \mathbb{C}^n$, s.t.

$$v^* W_o(t_1) v = 0 \Longrightarrow Ce^{At}v \equiv 0, \ \forall t \in [0, t_1]$$

Similar to the proof of Theorem C.5 (7. $\Rightarrow$ 1.), we can use conclusions from complex analysis to claim that $Ce^{At}v \equiv 0, \forall t > 0$. On the other hand, we set $u(t) \equiv 0$, which results in $y(t) = Ce^{At}x(0)$. In this case $x(0) = 0$ and $x(0) = v \neq 0$ will lead to the same output responses $y(t)$ over $t > 0$, which implies (C, A) is not observable. Contradiction!

(2) Next we show the duality of controllability and observability:

From (1) we know (C, A) is controllable if and only of

$$\int_0^t e^{A^*\tau} C^* C e^{A\tau} d\tau = \int_0^t e^{(A^*)\tau} (C^*)^* (C^*) e^{(A^*)^*\tau} d\tau$$

is nonsingular for all $t > 0$. The latter is exactly the definition of (A^*, C^*)'s controllability Gramian $W_c(t)$.

□

C.2.4 Equivalent Statements for Observability

With the Theorem of Duality C.6, we can directly write down the equivalent statements of observability without any additional proofs:

Theorem C.7 (Equivalent Statements for Observability)**.** *The following statements are equivalent (Chen, 1984), (Zhou et al., 1996):*

1. (C, A) *is observable.*
2. *The matrix*
$$W_o(t) := \int_0^t e^{A^*\tau} C^* C e^{A\tau} d\tau$$
is positive definite for any $t > 0$.
3. *The observability matrix*
$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \cdots \\ CA^{n-1} \end{bmatrix}$$
has full column rank.

4. *The matrix* $\begin{bmatrix} A-\lambda I \\ C \end{bmatrix}$ *has full column rank for all* $\lambda \in \mathbb{C}$.

5. *Let* λ *and* y *be any eigenvalue and any corresponding right eigenvector of* A*, i.e.,* $Ay = \lambda y$*, then* $Cy \neq 0$.

6. *The eigenvalues of* $A + LC$ *can be freely assigned (with the restriction that complex eigenvalues are in conjugate pairs) by a suitable choice of* L.

7. (A^*, C^*) *is controllable.*

8. *If, in addition, all eigenvalues of* A *have negative parts, then the unique solution of*

$$A^* W_o + W_o A = -C^* C$$

is positive definite. The solution is called the observability Gramian *and can be expressed as*

$$W_o = \int_0^\infty e^{A^*\tau} C^* C e^{A\tau} d\tau$$

C.3 Stabilizability And Detectability

To define stabilizability and detectability of an LTI system, we first introduce the concept of *system mode*, which can be naturally derived from the fifth definition of controllability C.5 (observability C.7).

Definition C.4 (System Mode)**.** λ is a mode of an LTI system, if it is an eigenvalue of A. The mode λ is said to be:

- stable, if $\mathrm{Re}\lambda < 0$,
- controllable, if $x^* B \neq 0$ for all left eigenvectors of A associated with λ,
- observable, if $Cx \neq 0$ for all right eigenvectors of A associated with λ.

Otherwise, the mode is said to be uncontrollable (unobservable).

With the concept of system mode, the fifth definition of controllability C.5 (observability C.7) can be restated as

> An LTI system is controllable (observable) if and only if all modes are controllable (observable).

Stabilizability (detectability) is defined similarly via loosening part of controllability (observability) conditions.

Definition C.5 (Stabilizability)**.** An LTI system is said to be stabilizable if all of its unstable modes are controllable.

Definition C.6 (Detectability)**.** An LTI system is said to be detectable if all of its unstable modes are observable.

Like in the case of controllability and observability, duality also holds in stabilizability and detectability. Moreover, similarity transformation will not influence an LTI system's stabilizability and detectability.

C.3.1 Equivalent Statements for Stabilizability

Theorem C.8 (Equivalent Statements for Stabilizability)**.** *The following statements are equivalent (Zhou et al., 1996):*

1. (A, B) *is stabilizable.*
2. *For all* λ *and* x *such that* $x^*A = \lambda x^*$ *and* $Re\lambda \geq 0$, $x^*B \neq 0$.
3. *The matrix* $[A - \lambda I, B]$ *has full rank for all* $Re\lambda \geq 0$.
4. *There exists a matrix* F *such that* $A + BF$ *are Hurwitz.*

Proof. (1. $\Leftrightarrow$ 2.) Directly from stabilizability's definition.

(2. $\Leftrightarrow$ 3.) If 2. holds but 3. not hold, then there exists $v \neq 0 \in \mathbb{C}^n$, s.t.

$$v^*[A - \lambda I, B] = 0 \Leftrightarrow v^*A = \lambda v^*, v^*B = 0, \mathrm{Re}\lambda \geq 0$$

Contradiction! Vice versa.

(4. $\Rightarrow$ 2.) Prove by contradiction. Suppose there $x \neq 0 \in \mathbb{C}^n$, s.t.

$$x^*[A - \lambda I, B] = 0 \Leftrightarrow x^*A = \lambda x^*, x^*B = 0, \mathrm{Re}\lambda \geq 0$$

Thus, for any F,

$$x^*(A + BF) = \lambda x^*, \mathrm{Re}\lambda \geq 0$$

On the other hand, suppose $A + BF$ has I Jordon blocks, with each equipped with an eigenvalue $\eta_i, i = 1 \dots I$ (note that η_α may be equal to η_β, i.e., they are equivalent eigenvalues with different Jordon blocks). Since $A + BF$'s eigenvalues all have negative real parts, $\mathrm{Re}(\eta_i) < 0, i = 1 \dots I$. For each $\eta_i, i \in \{1 \dots i\}$, denote its K_i generalized left eigenvectors as $v_{i,1}, v_{i,2}, \dots v_{i,K_i}$. By definition, $\sum_{i=1}^{I} K_i = n$ and

$$\begin{aligned} v_{i,1}^*(A + BF) &= v_{i,1}^* \cdot \eta_i \\ v_{i,2}^*(A + BF) &= v_{i,1}^* + v_{i,2}^* \cdot \eta_i \\ &\vdots \\ v_{i,K_i}^*(A + BF) &= v_{i,K_i-1}^* + v_{i,K_i}^* \cdot \eta_i \end{aligned}$$

for all $i \in \{1 \dots i\}$. Also, $v_{i,k}, i = 1 \dots I, k = 1 \dots K_i$ are linearly independent and spans $\mathbb{C}^n$. Therefore,

$$x^* = \sum_{i=1}^{I} \sum_{k=1}^{K_i} \xi_{i,k} \cdot v_{i,k}^*$$

which leads to

$$\sum_{i=1}^{I} \sum_{k=1}^{K_i} \xi_{i,k} \cdot v_{i,k}^* (A + BF) = \sum_{i=1}^{I} \sum_{k=1}^{K_i} \xi_{i,k} \cdot \lambda \cdot v_{i,k}^*$$

Since $v_{i,k}$'s are $A + BF$'s generalized eigenvectors, we have

$$\begin{aligned}
&\sum_{i=1}^{I} \sum_{k=1}^{K_i} \xi_{i,k} \cdot v_{i,k}^* \cdot (A + BF) \\
=&\sum_{i=1}^{I} \left\{ \xi_{i,1} \cdot \eta_i \cdot v_{i,1}^* + \sum_{k=2}^{K_i} \xi_{i,k} (v_{i,k-1}^* + \eta_i \cdot v_{i,k}^*) \right\} \\
=&\sum_{i=1}^{I} \left\{ \sum_{k=1}^{K_i - 1} (\xi_{i,k} \cdot \eta_i + \xi_{i,k+1}) v_{i,k}^* + \xi_{i,K_i} \cdot \eta_i \cdot v_{i,K_i}^* \right\}
\end{aligned}$$

Combining the above two equations:

$$\sum_{i=1}^{I} \left\{ \sum_{k=1}^{K_i - 1} \left[\xi_{i,k} \cdot (\eta_i - \lambda) + \xi_{i,k+1} \right] v_{i,k}^* + \xi_{i,K_i} \cdot (\eta_i - \lambda) \cdot v_{i,K_i}^* = 0 \right\}$$

Since $v_{i,k}$'s are linearly independent, for any $i \in \{i \dots I\}$:

$$\begin{aligned}
\xi_{i,1} \cdot (\eta_i - \lambda) + \xi_{i,2} = 0 &\Rightarrow \xi_{i,2} = (-1) \cdot \xi_{i,1} \cdot (\eta_i - \lambda) \\
\xi_{i,2} \cdot (\eta_i - \lambda) + \xi_{i,3} = 0 &\Rightarrow \xi_{i,3} = (-1)^2 \cdot \xi_{i,1} \cdot (\eta_i - \lambda)^2 \\
&\vdots \\
\xi_{i,K_i - 1} \cdot (\eta_i - \lambda) + \xi_{i,K_i} = 0 &\Rightarrow \xi_{i,K_i} = (-1)^{K_i - 1} \cdot \xi_{i,1} \cdot (\eta_i - \lambda)^{K_i - 1} \\
\xi_{i,K_i} \cdot (\eta_i - \lambda) = 0 &
\end{aligned}$$

Thus,

$$(-1)^{K_i - 1} \cdot \xi_{i,1} \cdot (\eta_i - \lambda)^{K_i} = 0$$

Denote $\xi_{i,1}$ as $r_1 e^{\theta_1}$, $(\eta_i - \lambda)$ as $r_2 e^{\theta_2}$. Since $\mathrm{Re}\lambda \geq 0, \mathrm{Re}(\eta_i) < 0$, $r_2 > 0$. On the other hand, the following equation suggests

$$r_1 r_2^{K_i - 1} e^{j[\theta_1 + \theta_2 (K_i - 1)]} = 0$$

Thus, r_1 has to be 0, which implies $\xi_{i,1} = 0$. By recursion, $\xi_{i,k} = 0, \forall k = 1 \dots K_i$. Contradiction!

(1. $\Rightarrow$ 4.) If (A, B) is controllable, then from Theorem **??**(thm:lticontrollable)'s sixth definition, we can freely assign the poles of $A+BF$ via choosing F properly.

Otherwise, if (A, B) is uncontrollable, then from Corollary C.4 and proof of Theorem C.5 (6. $\Rightarrow$ 1.), there exists a similarity transformation T, s.t.

$$TAT^{-1} = \begin{bmatrix} \bar{A}_c & \bar{A}_{12} \\ 0 & \bar{A}_{\bar{c}} \end{bmatrix}, \quad TB = \begin{bmatrix} \bar{B}_c \\ 0 \end{bmatrix}$$

and

$$\det(A + BF - \lambda I) = \underbrace{\det(\bar{A}_c + \bar{B}_c F_1 - \lambda I_1)}_{\chi_c(\lambda)} \cdot \underbrace{\det(\bar{A}_{\bar{c}} - \lambda I_2)}_{\chi_{\bar{c}}(\lambda)}$$

where $\bar{A}_c \in \mathbb{C}^{k_1 \times k_1}$, I_1 identity matrix of size k_1, $[F_1, F_2] = FT^{-1}$, and $k_1 = \text{rank}\mathcal{C}$. Additionally, $(\bar{A}_c, \bar{B}_c)$ is controllable. Thus, $\chi_c(\lambda)$'s zeros can be freely assigned by choosing proper F, i.e., system modes with $\chi_c(\lambda)$ is controllable, regardless of its stability. On the other hand, system modes with $\chi_{\bar{c}}(\lambda)$ must be stable. Otherwise, we cannot affect it by assigning F, which is a contradiction to statement (1). Therefore, (TAT^{-1}, TB) is stabilizable. Since similarity transformation does not change stabilizability, (A, B) is stabilizable. □

C.3.2 Equivalent Statements for Detectability

Thanks to duality, we can directly write down the equivalent statements of observability without any additional proofs:

Theorem C.9 (Equivalent Statements for Detectability)**.** *The following statements are equivalent (Zhou et al., 1996):*

1. *(C, A) is detectable.*
2. *For all λ and x such that $Ax = \lambda x$ and $Re\lambda \geq 0$, $Cx \neq 0$.*
3. *The matrix $\begin{bmatrix} A - \lambda I \\ C \end{bmatrix}$ has full rank for all $Re\lambda \geq 0$.*
4. *There exists a matrix L such that $A + LC$ are Hurwitz.*
5. *(A^*, C^*) is stabilizable.*

Appendix D

Algebraic Techniques and Sum-of-Squares

D.1 Algebra

D.1.1 Polynomials

Definition D.1 (Monomial,Polynomial)**.** A **monomial** in $x_1, \cdots, x_n$ is a product of the form $x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}$. The **total degree** of the monomial is $\alpha_1 + \cdots + \alpha_n$.

A **polynomial** f in $x_1, \cdots, x_n$ with coefficients in $\mathbb{R}$ is a finite linear combination (with coefficients in $\mathbb{R}$) of monomials. We will write a polynomial f in the form: $\Sigma_\alpha a_\alpha x^\alpha$. where the sum is over a finite number of n-tuples $\alpha = (\alpha_1, \cdots, \alpha_n)$. The set of all polynomials in $x_1, \cdots, x_n$ with coefficients in $\mathbb{R}$ is denoted $\mathbb{R}[x_1, \cdots, x_n]$

Definition D.2 (Affine Variety)**.** Let $f_1, \cdots, f_s \in \mathbb{R}[x_1, \cdots, x_n]$, we set

$$V(f_1, \cdots, f_s) = \{(a_1, \cdots, a_n) \in \mathbb{R}^n | f_i(a_1, \cdots a_n) = 0 \quad \forall i \leq i \leq s\}$$

We call $V(f_1, \cdots, f_s)$ the **affine variety** defined bt $f_1, \cdots, f_s$

Definition D.3 (Ideal)**.** A subset $I \subset \mathbb{R}[x_1, \cdots, x_n]$ is an ideal if it satisfies: (i) Contains additive identity: $0 \in I$ (ii) Closed under addition: For all $f, g \in I$, $f + g \in I$ (iii) Absorption of multiplication: If $f \in I$ and $h \in \mathbb{R}[x_1, \cdots, x_n]$, then $hf \in I$

Definition D.4 (Sum of squares,Quadratic Module and Preordering)**.**

`Sum of squares`

D.1.2 Representation of nonnegative polynomial: Univariate case

Theorem D.1 (Global version)**.** *A polynomial $p \in \mathbb{R}[x]$ of even degree is nonnegative if and only if it can be written as a sum of squares of other polynomials, i.e., $p(x) = \Sigma_{i=1}^{k}[h_i(x)]^2$, with $h_i \in R[x], i = 1, \cdots, k$.*

Theorem D.2 (Compact interval version)**.** *A polynomial $p \in \mathbb{R}[x]$ of even degree is nonnegative if and only if it can be written as a sum of squares of other polynomials, i.e., $p(x) = \Sigma_{i=1}^{k}[h_i(x)]^2$, with $h_i \in R[x], i = 1, \cdots, k$.*

Appendix E

The Kalman-Yakubovich Lemma

Lemma E.1 (Kalman-Yakubovich)**.** *Consider a controllable linear time-invariant system*

$$\dot{x} = Ax + buy = c^T x.$$

The transfer function

$$h(p) = c^T(pI - A)^{-1}b$$

is strictly positive real (SPR) if and only if there exist positive definite matrices P and Q such that

$$A^T P + PA = -QPb = c.$$

Appendix F

Feedback Linearization

Appendix G

Sliding Control

Bibliography

Agarwal, A., Jiang, N., Kakade, S. M., and Sun, W. (2022). Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 32.

Andrieu, V. and Praly, L. (2006). On the existence of a kazantzis–kravaris/luenberger observer. *SIAM Journal on Control and Optimization*, 45(2):432–456.

Arnold, W. F. and Laub, A. J. (1984). Generalized eigenproblem algorithms and software for algebraic riccati equations. *Proceedings of the IEEE*, 72(12):1746–1754.

Astolfi, A. and Ortega, R. (2003). Immersion and invariance: A new tool for stabilization and adaptive control of nonlinear systems. *IEEE Transactions on Automatic control*, 48(4):590–606.

Bernard, P. (2019). *Observer design for nonlinear systems*, volume 479. Springer.

Bernard, P., Andrieu, V., and Astolfi, D. (2022). Observer design for continuous-time dynamical systems. *Annual Reviews in Control*.

Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific.

Besançon, G., Bornard, G., and Hammouri, H. (1996). Observer synthesis for a class of nonlinear control systems. *European Journal of control*, 2(3):176–192.

Blekherman, G., Parrilo, P. A., and Thomas, R. R. (2012). *Semidefinite optimization and convex algebraic geometry*. SIAM.

Chen, C.-T. (1984). *Linear system theory and design*. Saunders college publishing.

Davison, E. and Wonham, W. (1968). On pole assignment in multivariable linear systems. *IEEE Transactions on Automatic Control*, 13(6):747–748.

Dawson, C., Gao, S., and Fan, C. (2023). Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics*.

Ebenbauer, C., Renz, J., and Allgower, F. (2005). Polynomial feedback and observer design using nonquadratic lyapunov functions. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 7587–7592. IEEE.

Hammouri, H. and de Leon Morales, J. (1990). Observer synthesis for state-affine systems. In *29th IEEE Conference on Decision and Control*, pages 784–785. IEEE.

Janny, S., Andrieu, V., Nadri, M., and Wolf, C. (2021). Deep kkl: Data-driven output prediction for non-linear systems. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 4376–4381. IEEE.

Kalman, R. E. and Bucy, R. S. (1961). New results in linear filtering and prediction theory.

Karagiannis, D. and Astolfi, A. (2005). Nonlinear observer design using invariant manifolds and applications. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 7775–7780. IEEE.

Kazantzis, N. and Kravaris, C. (1998). Nonlinear observer design using lyapunov's auxiliary theorem. *Systems & Control Letters*, 34(5):241–247.

Lasserre, J. B. (2001). Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817.

Lasserre, J. B. (2009). *Moments, positive polynomials and their applications*, volume 1. World Scientific.

Luenberger, D. G. (1964). Observing the state of a linear system. *IEEE transactions on military electronics*, 8(2):74–80.

Magron, V. and Wang, J. (2023). *Sparse polynomial optimization: theory and practice*. World Scientific.

Miao, K. and Gatsis, K. (2023). Learning robust state observers using neural odes. In *Learning for Dynamics and Control Conference*, pages 208–219. PMLR.

Murray, R., Chandrasekaran, V., and Wierman, A. (2021). Signomial and polynomial optimization via relative entropy and partial dualization. *Mathematical Programming Computation*, 13:257–295.

Niazi, M. U. B., Cao, J., Sun, X., Das, A., and Johansson, K. H. (2023). Learning-based design of luenberger observers for autonomous nonlinear systems. In *2023 American Control Conference (ACC)*, pages 3048–3055. IEEE.

Nie, J. (2023). *Moment and Polynomial Optimization*. SIAM.

Nocedal, J. and Wright, S. J. (1999). *Numerical optimization*. Springer.

Slotine, J.-J. E., Li, W., et al. (1991). *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ.

Vinograd, R. È. (1957). Inapplicability of the method of characteristic exponents to the study of non-linear differential equations. *Matematicheskii Sbornik*, 83(4):431–438.

Wang, J. (2022). Nonnegative polynomials and circuit polynomials. *SIAM Journal on Applied Algebra and Geometry*, 6(2):111–133.

Yang, H. and Carlone, L. (2022). Certifiably optimal outlier-robust geometric perception: Semidefinite relaxations and scalable global optimization. *IEEE transactions on pattern analysis and machine intelligence*, 45(3):2816–2834.

Yang, H., Liang, L., Carlone, L., and Toh, K.-C. (2022). An inexact projected gradient method with rounding and lifting by nonlinear programming for solving rank-one semidefinite relaxation of polynomial optimization. *Mathematical Programming*, pages 1–64.

Zhou, K., Doyle, J., and Glover, K. (1996). Robust and optimal control. *Control Engineering Practice*, 4(8):1189–1190.