Optimal Control and Reinforcement Learning

Heng Yang

2025-03-08

Contents

Preface **5**

Feedback 5

Offerings 5

1 The Optimal Control Formulation **7**

1.1 The Basic Problem 7

1.2 Dynamic Programming and Principle of Optimality 9

1.3 Infinite-horizon Formulation 12

2 Exact Dynamic Programming **15**

2.1 Linear Quadratic Regulator 15

2.2 Markov Decision Process 21

3 Approximate Optimal Control **33**

3.1 Fitted Value Iteration 34

3.2 Trajectory Optimization 40

3.3 Model Predictive Control 60

3.4 Policy Gradient 87

4 Stability Analysis **89**

4.1 Autonomous Systems 90

4.2 Controlled Systems 109

4.3 Non-autonomous Systems 110

5 Problem Sets **113**

Acknowledgement **131**

A Linear Algebra and Differential Equations **133**

A.1 Linear Algebra 133

A.2 Solving an Ordinary Differential Equation 135

B Convex Analysis and Optimization **137**

B.1 Theory 137

B.2 Practice 143

C Linear System Theory **159**

C.1 Stability 159

C.2 Controllability and Observability 163

C.3 Stabilizability And Detectability 175

D Algebraic Techniques and Sum-of-Squares **179**

D.1 Algebra 179

E The Kalman-Yakubovich Lemma **181**

F Feedback Linearization **183**

G Sliding Control **185**

Preface

This is the textbook for Harvard ES/AM 158: Introduction to Optimal Control and Reinforcement Learning.

Feedback

I would like to invite you to provide feedback to the textbook via inline comments with Hypothesis:

- Go to Hypothesis and create an account
- Install the Chrome extension of Hypothesis
- Provide public comments to textbook contents and I will try to address them

Offerings

2025 Fall

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

Location: TBD

Instructor: Heng Yang

Teaching Fellow: TBD

Syllabus

2023 Fall

The course was previously offered as Introduction to Optimal Control and Estimation.

Starting Fall 2025, contents about reinforcement learning have been added to the course.

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.

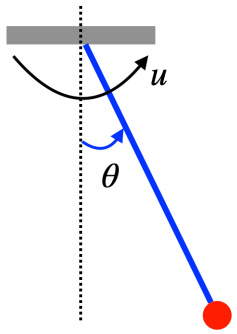


Figure 2.1: A Simple Pendulum.

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.

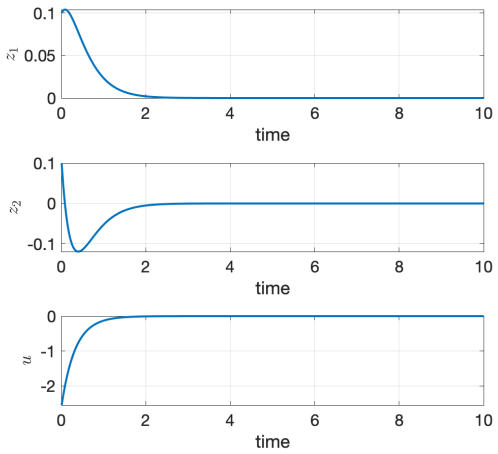


Figure 2.2: LQR stabilization of a simple pendulum.

2.1.2 LQR with Constraints

Let's explore LQR with constraints in Exercise 5.2

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:

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

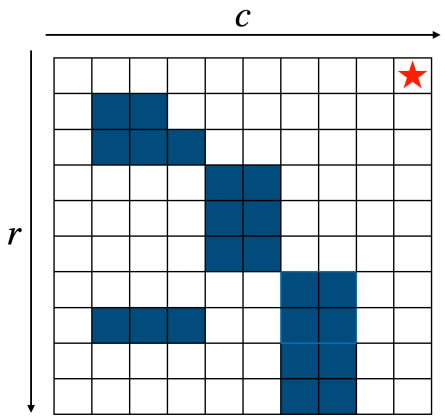


Figure 2.3: Grid World with Obstacles.

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?

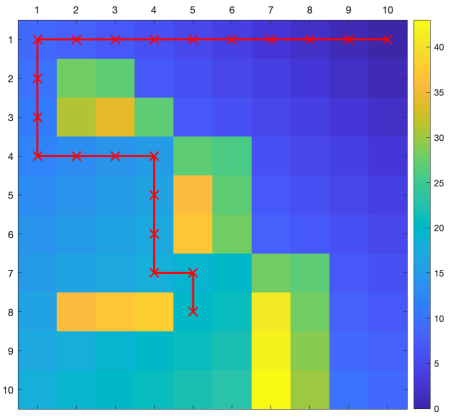


Figure 2.4: Optimal cost-to-go and an optimal trajectory.

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.

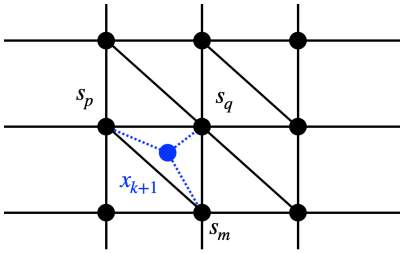


Figure 2.5: Barycentric Interpolation.

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.

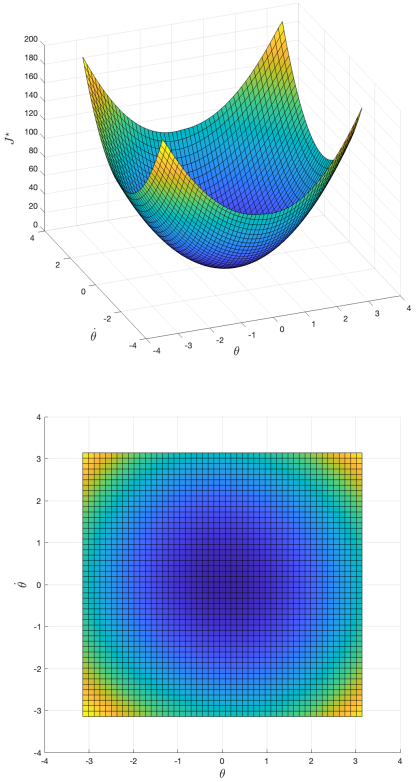


Figure 2.6: Optimal cost-to-go with discount factor 0.9.

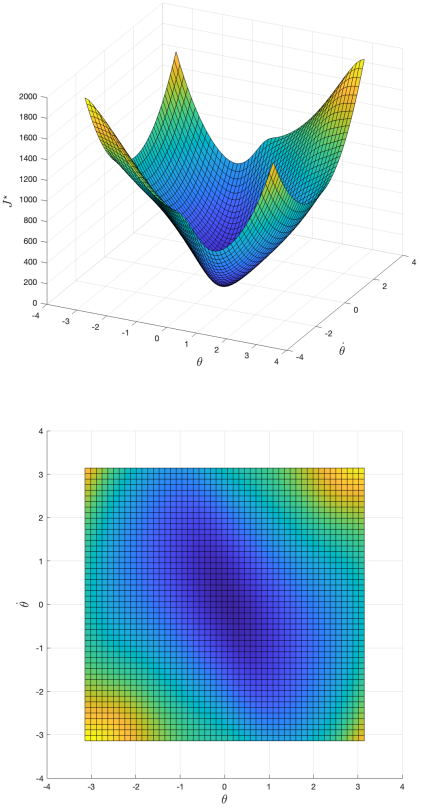


Figure 2.7: Optimal cost-to-go with discount factor 0.99.

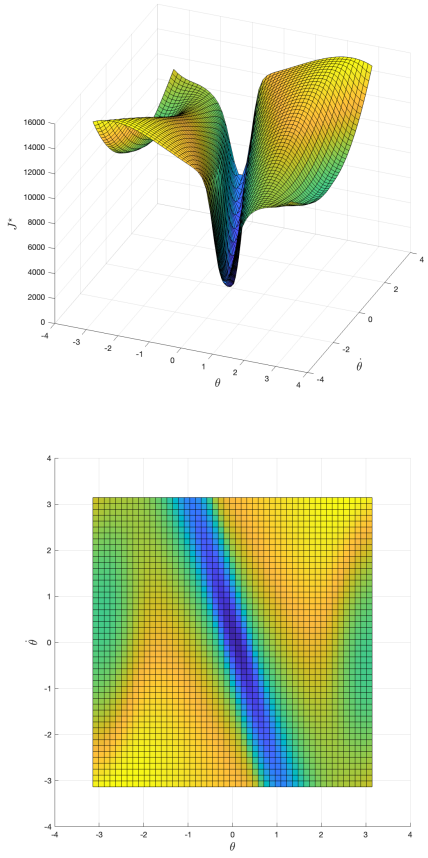


Figure 2.8: Optimal cost-to-go with discount factor 0.999.

Chapter 3

Approximate Optimal Control

Thanks to Jiarui Li for contributing to this Chapter.

In Chapter 2, we have studied two cases where dynamic programming (DP) can be executed exactly. Both cases are interesting yet limiting.

In the linear quadratic regulator (LQR) case, the optimal controller is a linear controller and it does not require discretization of the state and control space. However, LQR only handles systems with linear dynamics.

In the Markov Decision Process (MDP) case, value iteration can obtain the optimal cost-to-go (or the optimal Q-value function) usually in a finite number of iterations. With barycentric interpolation, value iteration also leads to a practical controller for swinging up the pendulum (see Example 2.3), at least starting from some initial states. However, value iteration suffers from the *curse of dimensionality*, i.e., the amount of memory and computation needed to compute the optimal cost-to-go grows exponentially with the number of grids used to discretize the state space and control space.

In this Chapter, we will study approximate dynamic programming, which aims to not find the optimal controller (simply because it is too demanding), but only a suboptimal controller that is perhaps good enough. It is worth noting that there exists a large amount of algorithms and literature for approximate DP, most of which are closely related to reinforcement learning, and studying all of them is beyond the scope of this course. In the following we will only go through several algorithmic frameworks that are representative.

3.1 Fitted Value Iteration

Let us consider the infinite-horizon optimal control problem introduced in Chapter 1.3

$$\min_{\pi} \mathbb{E}\left\{\sum_{k=0}^{\infty} g(x_k, \pi(x_k))\right\}.$$

Under some technical conditions, we know that the optimal policy is a deterministic and stationary policy and the optimal cost-to-go satisfies the following Bellman Optimality Equation

$$J^\star(x) = \min_{u\in\mathbb{U}} \mathbb{E}_w \left\{g(x,u) + J^\star(f(x,u,w))\right\}, \quad \forall x \in \mathbb{X}. \tag{3.1}$$

From the plots in Example 2.3, we observe that even for a "simple" problem like pendulum swing-up, the optimal cost-to-go $J^\star(x)$ does not look simple at all.

So here comes a very natural idea. What if we parametrize a cost-to-go function $\tilde{J}(x,r)$ by a vector of unknown coefficients $r \in \mathbb{R}^d$ and ask $\tilde{J}(x,r)$ to satisfy (3.1) as closely as possible?

This indeed leads to a valid algoithm called fitted value iteration (FVI).

In FVI, we initialize the parameters r as r_0 in the first step. Then, at the k-th iteration, we perform two subroutines.

Value update. Firstly, we sample a large amount of points in the state space $\{x_k^s\}_{s=1}^q$, and for each x_k^s, we solve the minimization problem on the right-hand side of (3.1) using $J(x, r^{(k)})$ as the cost-to-go. Formally, this is

$$\beta_k^s \leftarrow \min_{u\in\mathbb{U}} \mathbb{E}_w \left\{g(x_k^s, u) + \tilde{J}(f(x_k^s, u, w), r^{(k)})\right\}, \quad \forall x_k^s, s = 1, \dots, q. \tag{3.2}$$

This step gives us a list of scalars $\{\beta_k^s\}_{s=1}^q$. If $\tilde{J}(x,r)$ indeed satisfies the Bellman optimality equation, then we should have

$$\tilde{J}(x_k^s, r^{(k)}) = \beta_k^s, \quad \forall s = 1, \dots, q.$$

This is certainly not true when the parameter r is imperfect.

Parameter update. Therefore, we will see the best parameter that minimizes the violation of the above equation

$$r^{(k+1)} \leftarrow \arg\min_{r\in\mathbb{R}^d} \sum_{s=1}^{q} \left(\tilde{J}(x_k^s, r) - \beta_k^s\right)^2. \tag{3.3}$$

FVI essentially carries out (3.2) and (3.3) until some convergence metric is met.

Two challenges immediately show up:

- How to perform the minimization over u in the first step (3.2)?
- How to find the best parameter update in the second step (3.3)?

To solve the first challenge, we will assume that $\mathbb{U}$ is a finite set so that minimization over u can be solved exactly. Note that this assumption is not strictly necessary. In fact, as long as $\mathbb{U}$ is a convex set and the objective in (3.2) is also convex, then the minimization can also be solved exactly (Yang and Boyd, 2023). However, we assume $\mathbb{U}$ is finite to simplify our presentation.

To solve the second challenge, we will assume $\tilde{J}(x, r)$ is *linear* in the parameters r, as we will study further in the following.

3.1.1 Linear Features

We parameterize $\tilde{J}(x, r)$ as follows:

$$\tilde{J}(x, r) = \sum_{i=1}^{d} r_i \cdot \phi_i(x) = \begin{bmatrix} \phi_1(x) & \cdots & \phi_d(x) \end{bmatrix} r = \phi(x)^T r, \tag{3.4}$$

where $\phi_1(x), \ldots, \phi_d(x)$ are a set of known basis functions, or *features*. Common examples of features include polynomials and radial basis functions. With this parametrization, the optimization in (3.3) becomes

$$\min_{r \in \mathbb{R}^d} \sum_{s=1}^{q} \left(\phi(x_k^s)^T r - \beta_k^s \right)^2, \tag{3.5}$$

which is a least-squares problem that can be solved in closed form. In particular, we can compute the gradient of its objective with respect to r and set it to zero, leading to

$$0 = \sum_{s=1}^{q} 2(\phi(x_k^s)^T r - \beta_k^s)\phi(x_k^s) \implies \tag{3.6}$$

$$r = \left(\sum_{s=1}^{q} \phi(x_k^s)\phi(x_k^s)^T \right)^{-1} \left(\sum_{s=1}^{q} \beta_k^s \phi(x_k^s) \right). \tag{3.7}$$

Let us apply FVI with linear features to a linear system for which we know the optimal cost-to-go.

Example 3.1 (Fitted Value Iteration for Double Integrator)**.** Consider the double integrator

$$\ddot{q} = u.$$

Take $x = [q; \dot{q}]$, we can write the dynamics in state-space form

$$\dot{x} = \begin{bmatrix} \dot{q} \\ u \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{3.8}$$

We use a constant time $h = 0.01$ differentiation to convert the continuous-time dynamics into discrete-time:

$$x_{k+1} = h \cdot \dot{x}_k + x_k = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ h \end{bmatrix} u_k.$$

The goal is to regulate the system at $(0, 0)$. To do so, we use a quadratic cost

$$J(x) = \min \sum_{k=0}^{\infty} x_k^T Q x_k + u_k^T R u_k, \quad x_0 = x.$$

with $Q = 0.1 I_2$ and $R = 1$.

In Chapter 2, we learned how to use LQR to precisely calculate the cost-to-go function of these systems using the *Algebraic Recatti Equation*. Solving the ARE, we obtain

$$S_{\text{LQR}} = \begin{bmatrix} 27.1640 & 31.7584 \\ 31.7584 & 85.9510 \end{bmatrix}.$$

Now we want to investigate if FVI can find the same matrix S. To do so, we parametrize

$$\tilde{J}(x) = x^T S x = x^T \begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} x = \begin{bmatrix} x_1^2 & 2x_1 x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix},$$

where S only has three independent variables due to being symmetric. Do note that $\tilde{J}(x)$ is *linear* in S, so the parameter update step can be solved in closed form.

We now show that the value update step can also be solved in closed form. Suppose we choose x_k as a sample, then the right-hand side of (3.2) reads:

$$\min_u x_k^T Q x_k + u^2 + (A x_k + B u)^T S^{(k)} (A x_k + B u),$$

where $S^{(k)}$ is the value of S at the k-th iteration. Clearly, the optimization problem above is a convex quadratic optimization and can be solved in closed form:

$$u = -(1 + B^T S^{(k)} B)^{-1} B^T S^{(k)} A x_k.$$

Applying fitted value iteration on randomly sampled points mentioned above, we obtain the fitted S

$$S_{\text{FVI}} = \begin{bmatrix} 27.1640 & 31.7583 \\ 31.7584 & 85.9509 \end{bmatrix}.$$

We can see that S_{FVI} almost exactly matches the groundtruth LQR solution S_{LQR}.

You can play with the code here.

3.1.2 Neural Network Features

We have seen that simple quadratic features work for the LQR problem. For more complicated problems, we will need more powerful function approximators like neural networks.

When we parameterize $\tilde{J}(x, r)$ as a neural network, r will be the weights of the neural network. We may still be able to solve the value update step if u lives in a finite space. However, the parameter update step usually cannot be solved exactly and will need to rely on numerical algoithms such as gradient descent.

Let us try neural FVI on the same double integrator problem.

Example 3.2 (Neural Fitted Value Iteration for the Double Integrator)**.** 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 positive definite network to model the cost-to-go function

$$J(x) = N(x)^T N(x), \tag{3.9}$$

where $N(x)$ is a 3-layer Multi-Layer-Perceptron with ReLU activation.

Using mini-batch learning plus Adam optimizer, we obtain the cost-to-go in Fig. 3.1 after 300 epochs of training.

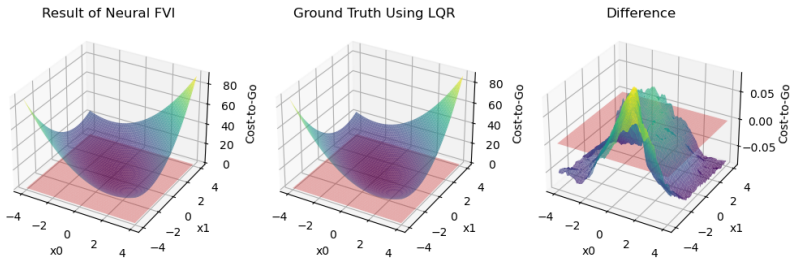


Figure 3.1: Comparison between the result of NN-based FVI and ground truth

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 $(0, 0)$.

You can see the code here.

3.1.3 Fitted Q-value Iteration

From the MDP Chapter we know there is an equivalent representation of the Bellman Optimality Equation by replacing the J value function in (3.1) with

the Q-value function $Q(x,u)$. In particular, with

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

substituted into (3.1), we obtain the Bellman Optimality Equation in $Q^\star(x,u)$:

$$Q^\star(x,u) = g(x,u) + \mathbb{E}_w\left\{\min_{u'\in\mathbb{U}} Q^\star(f(x,u,w),u')\right\}. \tag{3.10}$$

We can then use the same idea to approximate $Q^\star(x,u)$ as

$$\tilde{Q}(x,u,r)$$

with $r\in\mathbb{R}^d$ a parameter vector. By iteratively evaluating the right-hand side of (3.10), we obtain the algoithm known as *fitted Q-value iteration* (FQI).

For example, we can similarly adopt the linear feature parameterization and set

$$\tilde{Q}(x,u,r) = \phi(x,u)^T r,$$

where $\phi(x,u)$ is a known pre-selected feature vector. Then at the k-th iteration of FQI, we perform two subroutines.

Value update. We sample a number of state-control pairs $\{(x_k^s,u_k^s)\}_{s=1}^q$ and evaluate the right-hand side of the Bellman optimality equation (3.10):

$$\beta_k^s \leftarrow g(x_k^s,u_k^s) + \mathbb{E}_w\left\{\min_{u'\in\mathbb{U}} \tilde{Q}(f(x_k^s,u_k^s,w),u',r^{(k)})\right\}, \quad \forall s=1,\dots,q. \tag{3.11}$$

Again, if u lives in a finite space, or the minimization is convex, the above value update can be solved exactly.

Parameter update. We update the parameter vector using the updated values $\{\beta_k^s\}_{s=1}^q$:

$$r^{(k+1)} \leftarrow \arg\min_{r\in\mathbb{R}^d} \sum_{s=1}^q \left(\tilde{Q}(x_k^s,u_k^s,r) - \beta_k^s\right)^2, \tag{3.12}$$

which is a least squares problem that can be solved in closed form.

Let us apply FQI to the same double integrator example.

Example 3.3 (Fitted Q-value Iteration for Double Integrator)**.** Consider the same double integrator dynamics in Example 3.1.

From the LQR solution we know the optimal Q-value function is

$$Q^\star(x,u) = x^TQx + u^TRu + (Ax+Bu)^TS(Ax+Bu) \tag{3.13}$$

$$= \begin{bmatrix}x\\u\end{bmatrix}^T \underbrace{\begin{bmatrix}A^TSA+Q & A^TSB\\ B^TSA & B^TSB+R\end{bmatrix}}_{M_{\text{LQR}}} \begin{bmatrix}x\\u\end{bmatrix}, \tag{3.14}$$

where M_{LQR} is

$$M_{\text{LQR}} = \begin{bmatrix} 27.2640 & 32.0300 & 0.3176 \\ 32.0300 & 86.6889 & 0.8627 \\ 0.3176 & 0.8627 & 1.0086 \end{bmatrix}.$$

Let us apply FQI to see if we get the same solution. We parametrize

$$\tilde{Q}(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} M_1 & M_2 & M_3 \\ M_2 & M_4 & M_5 \\ M_3 & M_5 & M_6 \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{3.15}$$

$$= \begin{bmatrix} x_1^2 & 2x_1x_2 & 2x_1u & x_2^2 & 2x_2u & u^2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{bmatrix}. \tag{3.16}$$

At the k-th FQI iteration, we are given $M^{(k)}$. Suppose we sample (x_k, u_k), then the value update step needs to solve (3.11), which reads:

$$\beta_k = g(x_k,u_k) + \min_{u'} \underbrace{\begin{bmatrix} Ax_k + Bu_k \\ u' \end{bmatrix}^T M^{(k)} \begin{bmatrix} Ax_k + Bu_k \\ u' \end{bmatrix}}_{\psi(u')}.$$

The objective function $\psi(u')$ can be shown to be quadratic:

$$\psi(u') = (Lu' + a_k)^T M^{(k)} (Lu' + a_k), \quad L = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, a_k = \begin{bmatrix} Ax_k + Bu_k \\ 0 \end{bmatrix}.$$

Therefore, we can solve u' in closed-form

$$u' = -(L^T M^{(k)} L)^{-1} L^T M^{(k)} a_k.$$

Applying FQI with the closed-form update above, we get

$$M_{\text{FQI}} = \begin{bmatrix} 27.2640 & 32.0300 & 0.3176 \\ 32.0300 & 86.6889 & 0.8627 \\ 0.3176 & 0.8627 & 1.0086 \end{bmatrix},$$

which is exactly the same as the solution obtained from LQR.

You can play with the code here.

3.1.4 Deep Q Network

Although we have only tested FVI and FQI on the simple double integrator linear system, these algoithms are really not so different from the state-of-the-art reinforcement learning algoithms. For example, the core of the seminal work

Deep Q-Network (DQN) (Mnih et al., 2015) is to use a deep neural network to parameterize the Q-value function. Of course the DQN work has used other clever ideas to make it work in practice, such as experience replay, but the essence is fitted Q-value iteration.

You can find a good explanation of DQN in this tutorial, and a practical step-by-step Pytorch implementation that applies DQN on the cart-pole system.

3.1.5 Deep + Shallow

Combine the rich features of DQN with the stable learning of FQI (Levine et al., 2017).

3.2 Trajectory Optimization

Consider a continuous-time optimal control problem (OCP) in full generality

$$\begin{aligned} \min_{u(t), t\in[0,T]} \quad & g_T(x(T)) + \int_{t=0}^{T} g(x(t), u(t))dt \\ \text{subject to} \quad & \dot{x} = f(x(t), u(t)) \\ & x(0) = x_0 \\ & (x(t), u(t)) \in \mathcal{X} \times \mathcal{U} \\ & \phi_i(x(t), u(t)) \geq 0, i = 1, \dots, q. \end{aligned} \tag{3.17}$$

where g_T the terminal cost, g the running cost, x_0 the initial condition, $\mathcal{X}$ the state constraint set, $\mathcal{U}$ the control constraint set, and $\{\phi_i\}_{i=1}^q$ are general state-control constraints. We assume that the state constraint set $\mathcal{X}$ and control constraint set $\mathcal{U}$ can be described by a finite set of inequalities, i.e.,

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid c_i^x(x) \geq 0, i = 1, \dots, q_x\}, \quad \mathcal{U} = \{u \in \mathbb{R}^m \mid c_i^u(u) \geq 0, i = 1, \dots, q_u\}.$$

Assume that all functions are differentiable.

Our goal is to "transcribe" the infinite-dimensional continuous-time optimization (3.17) into a *nonlinear programming problem* (NLP) of the following form

$$\begin{aligned} \min_{v\in\mathbb{R}^{n_v}} \quad & c(v) \\ \text{subject to} \quad & c_{\text{eq},i}(v) = 0, i = 1, \dots, n_{\text{eq}} \\ & c_{\text{ineq},i}(v) \geq 0, i = 1, \dots, n_{\text{ineq}}, \end{aligned} \tag{3.18}$$

where v is a finite-dimensional vector variable to be optimized, $c, c_{\text{eq},i}, c_{\text{ineq},i}$ are (continuously differentiable) objective and constraint functions. Once we have done the transcription from (3.17) to (3.18), then we have a good number of

numerical optimization algorithms at our disposal. For example, the Matlab `fmincon` provides a nice interface to many such algorithms, e.g., interior-point method and sequential quadratic programming. You have already played with a simple example of `fmincon` in Exercise 5.1. Other well-implemented NLP algorithms include IPOPT and SNOPT. However, usually the Matlab `fmincon` gives a decent start point for trying out different algorithms before moving to commercial solvers such as SNOPT. For a more comprehensive understanding about how NLP algoithms work under the hood, I suggest reading (Nocedal and Wright, 1999).

Before we talk about how to transcribe the OCP into an NLP, let us use a simple example to get a taste of `fmincon`.

Example 3.4 (Fire a Canon with fmincon)**.** On a 2D plane, suppose we have a canon at the origin $(0, 0)$, and we want to fire the canon, with control over the initial velocity (v_1, v_2), so that the canon ball hits the target location $(10, 10)$.

From our basic physics knowledge, we know the trajectory of the canon ball is described by

$$\begin{cases} x_1(t) = v_1 t \\ x_2(t) = v_2 t - \frac{1}{2} g t^2, \end{cases}$$

where g is the gravity constant.

With the trajectory of the canon ball written above, we can formulate our nonlinear programming problem (NLP) as:

$$\begin{aligned} \min_{v_1, v_2, T} \quad & \frac{1}{2}(v_1^2 + v_2^2) \\ \text{subject to} \quad & v_1, v_2, T \geq 0 \\ & x_1(T) = v_1 T = 10 \\ & x_2(T) = v_2 T - \frac{1}{2} g T^2 = 10 \end{aligned} \tag{3.19}$$

where our unknown variables are the initial velocities and the time T at which the canon ball hits the target. In the NLP (3.19), the first constraint asks all our varaibles to be nonnegative, the second and third constraints enforce the canon ball to hit the target.

The following script formulates the NLP (3.19) in matlab.

```
clc; clear; close all;
g = 9.8;
x0 = [0;0;1]; % initial guess of the solution
% define objective function
obj = @(x) objective(x);
% define nonlinear constraints
```

```
nonlincon = @(x) nonlinear_con(x,g);
% define options for fmincon
options = optimoptions('fmincon','Algorithm','interior-point',...
    'SpecifyObjectiveGradient',true,'SpecifyConstraintGradient',true,...
    'checkGradients',false);
% call fmincon
[xopt,fopt,~,out] = fmincon(obj,x0,... % objective and initial guess
    -eye(3),zeros(3,1),... % linear inequality constraints
    [],[],... % no linear equality constraints
    [],[],... % no upper and lower bounds
    nonlincon,... % nonlinear constraints
    options);
fprintf("Maximum constraint violation: %3.2f.\n",out.constrviolation);
fprintf("Objective: %3.2f.\n",fopt);
% plot the solution
T = xopt(3);
t = 0:0.01:T;
xt = xopt(1)*t;
yt = xopt(2)*t - 0.5*g*(t.^2);
figure;
plot(xt,yt,'LineWidth',2);
hold on
scatter(10,10,100,"red",'filled','diamond');
axis equal; grid on;
xlabel('$x_1(t)$','FontSize',24,'Interpreter','latex');
ylabel('$x_2(t)$','FontSize',24,'Interpreter','latex');
ax = gca; ax.FontSize = 20;
%% helper functions
function [f,df] = objective(x)
% x is our decision variable: [v1, v2, T]
% objective function
f = 0.5 * (x(1)^2 + x(2)^2);
% gradient of the objective function (optional)
df = [x(1); x(2); 0]; % column vector
end

function [c,ceq,dc,dceq] = nonlinear_con(x,g)
% no inequality constraints
c = []; dc = [];
% two equality constraints
ceq = [x(1)*x(3)-10;
       x(2)*x(3)-0.5*g*x(3)^2-10];
% explicit gradient of ceq
dceq = [x(3), 0;
```

```
        0, x(3);
        x(1), x(2)-g*x(3)];
end
```

Running the code above, we get the following trajectory

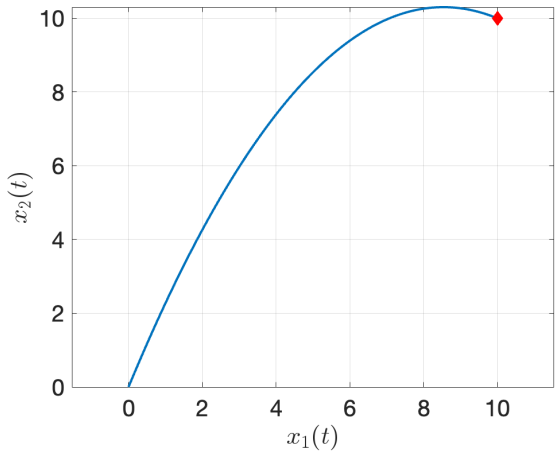


Figure 3.2: Trajectory of the canon ball obtained from fmincon.

What if you change the initial guess to `fmincon`, or add more constraints? Feel free to play with the code here.

It turns out there are multiple different ways to transcribe the optimal control problem (3.17) as the nonlinear programming problem (3.18). In the following, we will introduce three different formulations, direct single shooting, direct multiple shooting, and direct collocation.

3.2.1 Direct Single Shooting

In direct single shooting, we transcribe the OCP to the NLP by only optimizing a sequence of control values, with the intuition being that the state values are functions of the controls by invoking the system dynamics. In particular, we follow the procedure below.

Time discretization. We first discretize the total time window $[0, T]$ into a set of N intervals:

$$0 = t_0 \leq t_1 \leq \cdots \leq t_k \leq t_{k+1} \leq \cdots \leq t_N = T.$$

We denote

$$h_k = t_{k+1} - t_k$$

as the length of the k-th time interval.

Piece-wise constant control. We will approximate the continuous-time control signal $u(t)$ as a piece-wise constant function, i.e.,

$$u(t) = u_k, \forall t \in [t_k, t_{k+1}).$$

This is shown in Fig. 3.3. Let us collect all the constant control values as our decision variable to be optimized

$$v = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \in \mathbb{R}^{Nm}. \tag{3.20}$$

This v will be our variable in the NLP (3.18). Note that v has dimension Nm.

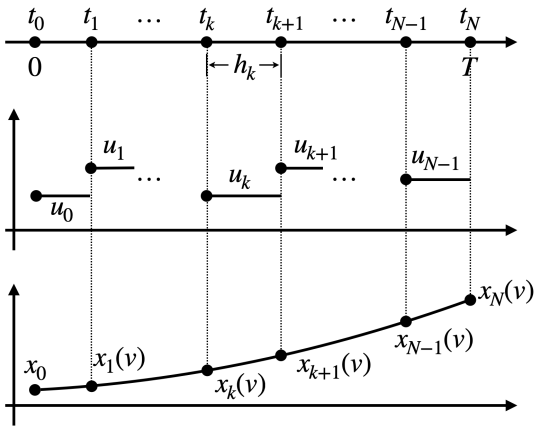


Figure 3.3: Direct single shooting.

Dynamics integration. It is clear that once v in (3.20), i.e., the set of controls, is determined, then the state trajectory $x(t)$ is also uniquely determined by the initial condition $x(0) = x_0$ and the dynamics

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

In order to enforce the state constraint $x(t) \in \mathcal{X}$, we will enforce the values of x at $t_0, t_1, \ldots, t_N$, i.e., $x_k = x(t_k)$, to lie in the constraint set $\mathcal{X}$. To do so, we need to integrate the dynamics. When the dynamics is linear, this integration can be done in closed form:

$$x_{k+1} = x_k + \int_{\tau=t_k}^{t_k+h_k} Ax(\tau) + Bu(\tau)d\tau = e^{Ah_k}x_k + A^{-1}(e^{Ah_k} - I)Bu_k.$$

By running the above equation from $k = 0$ to $k = N - 1$, we obtain the values of $x_k, k = 1, \dots, N$ as functions of the control vectors v, shown in Fig. 3.3. When the dynamics is nonlinear, we will need to perform the dynamics integration using numerical integration. One of the most well known family of integrators is called "Runge-Kutta methods". Among the family of methods, RK45 is perhaps the most popular one. We now briefly describe how a fourth-order RK integrator, i.e., RK4, works. Suppose we have already computed x_k, and we want to integrate the dynamics to obtain x_{k+1}. The RK4 integrator first computes the time derivatives at a sequence of four points:

$$\alpha_1 = f(x_k, u_k) \tag{3.21}$$

$$\alpha_2 = f\left(x_k + \frac{1}{2}h_k\alpha_1, u_k\right) \tag{3.22}$$

$$\alpha_3 = f\left(x_k + \frac{1}{2}h_k\alpha_2, u_k\right) \tag{3.23}$$

$$\alpha_4 = f\left(x_k + h_k\alpha_3, u_k\right). \tag{3.24}$$

Then we can obtain x_{k+1} via a weighted sum of the α_i's

$$x_{k+1} = x_k + \frac{1}{6}h_k\left(\alpha_1 + 2\alpha_2 + 2\alpha_3 + \alpha_4\right) = \text{RK4}(x_k, u_k). \tag{3.25}$$

Notice that a naive integrator would just perform $x_{k+1} = x_k + h_k\alpha_1$ without querying the gradients at three other points. The nice property of RK4 is that it leads to better accuracy than the naive integration. Writing the RK4 integrator recursively, we obtain

$$x_{k+1} = \text{RK4}(x_k, u_k) = \text{RK4}(\text{RK4}(x_{k-1}, u_{k-1}), u_k) = \text{RK4}(\text{RK4}(...(x_0, u_0))...u_k),$$

where RK4 is invoked for $k+1$ times. Clearly, each x_k is a complicated function of the sequence of controls v and the initial state x_0. We will write $x_k(v)$ to make this explict, as shown in Fig. 3.3.

Objective approximation. We can approximate the objective of the OCP (3.17) using trapezoidal integration:

$$g_T(x(T)) + \int_{t=0}^{T} g(x(t), u(t))dt \approx g_T(x_N(v)) + \sum_{k=0}^{N-1} \frac{h_k}{2}(g(x_k(v), u_k) + g(x_{k+1}(v), u_{k+1})).$$

Summary. In summary, the transcribed NLP for the optimal control problem using direct single shooting would be

$$\begin{aligned} \min_{v=(u_0,\dots,u_{N-1})} \quad & g_T(x_N(v)) + \sum_{k=0}^{N-1} \frac{g(x_k(v), u_k) + g(x_{k+1}(v), u_{k+1})}{2} h_k \\ \text{subject to} \quad & c_i^u(u_k) \geq 0, i = 1, \dots, q_u, \quad k = 0, \dots, N-1 \\ & c_i^x(x_k(v)) \geq 0, i = 1, \dots, q_x, \quad k = 1, \dots, N \\ & \phi_i(x_k(v), u_k) \geq 0, i = 1, \dots, q, \quad k = 0, \dots, N. \end{aligned} \tag{3.26}$$

Pros and Cons. The advantage of using direct single shooting to transcribe the OCP is clear: it leads to fewer variables because the only decision variables in (3.26) are the sequence of controls. In other transcription methods, the decision variable of the NLP typically involves the states as well. The disadvantage of direct single shooting is the complication of the function $x_k(v)$ caused by numerical integrators such as RK4. Even though the original continuous-time dynamics $f(x, u)$ may be simple, the RK4 function can be highly complicated due to evaluating $\dot{x}$ at multiple other locations. Moreover, the recursion of the RK4 operator makes this complication even worse.

Let us try the RK4 simulator for a simple example.

Example 3.5 (Propagation of Nonlinearities)**.** Consider the following dynamics

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \dot{x} = \begin{bmatrix} 10(x_2 - x_1) \\ x_1(u - x_3) - x_2 \\ x_1 x_2 - 3x_3 \end{bmatrix}$$

where u is a single scalar control. We are interested in running the RK4 simulator for 100 seconds (with $h_k = 0.01$ for all k) at $x_0 = [1, 0, 0]^T$ with u varying from 0 to 100, and see how the nonlinearities propagate.

Fig. 3.4 plots x as a function of u. We can clearly see the nonlinearities getting worse due to RK4.

You can play with the code here.

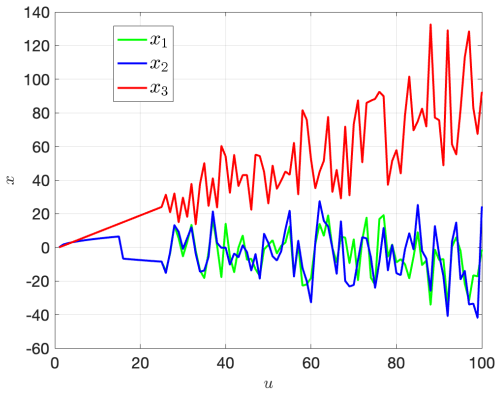


Figure 3.4: RK4 simulation.

3.2.2 Direct Multiple Shooting

Propagation of the nonlinearities through numerical integrators motivates using *direct multiple shooting* to transcribe the OCP problem into a nonlinear programming problem.

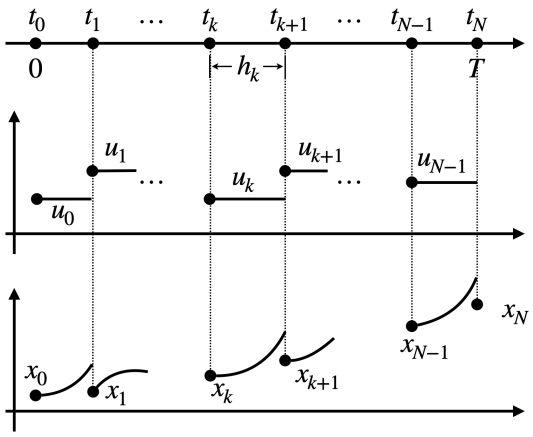


Figure 3.5: Direct multiple shooting.

Instead of only optimizing the controls, direct multiple shooting optimizes both the controls and states. The decision variable in the NLP becomes

$$v = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^{N(n+m)}.$$

With the state variables also introduced in the optimization, we no longer need to recursively run the RK4 integrators. Instead, we just need to run it once and enforce

$$x_{k+1} = \text{RK4}(x_k, u_k),$$

i.e., the current state and the next state satisfy the dynamics constraint. This is shown in Fig. 3.5.

In summary, the transcribed NLP for the OCP problem using direct multiple shooting becomes

$$\begin{aligned} \min_{v=(u_0,\dots,u_{N-1},x_1,\dots,x_N)} \quad & g_T(x_N) + \sum_{k=0}^{N-1} \frac{g(x_k,u_k)+g(x_{k+1},u_{k+1})}{2} h_k \\ \text{subject to} \quad & x_{k+1} = \text{RK4}(x_k,u_k), \quad k = 0,\dots,N-1 \\ & c_i^u(u_k) \geq 0, i = 1,\dots,q_u, \quad k = 0,\dots,N-1 \\ & c_i^x(x_k) \geq 0, i = 1,\dots,q_x, \quad k = 1,\dots,N \\ & \phi_i(x_k,u_k) \geq 0, i = 1,\dots,q, \quad k = 0,\dots,N. \end{aligned} \tag{3.27}$$

Direct multiple shooting avoids the recursion of numerical integrators such as RK4, but at the expense of introducing additional state variables in the NLP.

Let us try direct multiple shooting on the double integrator.

Example 3.6 (Direct Multiple Shooting for Minimum-Time Double Integrator)**.** The double integrator has continuous-time dynamics (which you have already seen in Exercise 5.2):

$$\ddot{q} = u.$$

In standard state-space form, the dynamics is linear

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}, \quad \dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u.$$

Let us consider the minimum-time optimal control problem

$$\begin{aligned} \min_{u(t), t \in [0,T]} \quad & T \\ \text{subject to} \quad & \dot{x} = Ax + Bu, \quad x(0) = x_0 \\ & x(T) = 0 \\ & u(t) \in [-1, 1], \forall t \in [0, T]. \end{aligned} \tag{3.28}$$

which seeks to get from the initial condition x_0 to the origin as fast as possible.

You should know from our physics intuition that the optimal controller is bang-bang, i.e., the optimal control first accelerates (deaccelerates) using the maximum control and then reverses using the opposite maximum control.

Let's see if we can obtain the optimal controller using direct multiple shooting.

In direct multiple shooting, we will optimize the control trajectory, the state trajectory, and the final time T. We fix the number of intervals N, and discretize $[0, T]$ evenly into N intervals, which leads to variables for the NLP as

$$v = (T, x_0, \dots, x_N, u_0, \dots, u_N).$$

We can easily enforce the control constraints:

$$u_k \in [-1, 1], k = 0, \dots, N,$$

and the initial / terminal constraints

$$x_0 = x_0, \quad x_N = 0.$$

The only nontrivial constraint is the dynamics constraint:

$$x_{k+1} = \text{RK4}(x_k, u_k), k = 0, \dots, N-1.$$

The following script shows how to use the Matlab integrator `ode45` to enforce the dynamics constraint.

```
function dx = double_integrator(t,states,v)
% return xdot at the selected times t and states, using information from v
% assume the controls in v define piece-wise constant control signal
T = v(1); % final time
N = (length(v) - 1) / 3; % number of knot points
u_grid = v(2+2*N:3*N+1); % N controls
t_grid = linspace(0,1,N)*T;
u_t = interp1(t_grid,u_grid,t,'previous'); % piece-wise constant
A = [0 1; 0 0];
B = [0; 1];
dx = A * states + B * u_t;
end

function [c,ceq] = double_integrator_nonlincon(v,initial_state)
% enforce x_k+1 = RK45(x_k, u_k); integration done using ode45
T = v(1); % final time
N = (length(v) - 1) / 3; % number of knot points
t_grid = linspace(0,1,N)*T;
x1 = v(2:N+1); % position
x2 = v(2+N:2*N+1); % velocity
% u = v(2+2*N:3*N+1); % controls
% no inequality constraints
c = [];
% equality constraints
ceq = [];
for i = 1:(N-1)
    ti = t_grid(i);
    tip1 = t_grid(i+1);
    xi = [x1(i);x2(i)];
    xip1 = [x1(i+1);x2(i+1)];
    % integrate system dynamics starting from xi in [ti,tip1]
    tt = ti:(tip1-ti)/20:tip1; % fine-grained time discretization
    [~,sol_int] = ode45(@(t,y) double_integrator(t,y,v),tt,xi);
    xip1_int = sol_int(end,:);
    % enforce them to be the same
    ceq = [ceq;
           xip1_int(1) - xip1(1);
           xip1_int(2) - xip1(2)];
end
% add initial state constraint
ceq = [ceq;
       x1(1) - initial_state(1);
       x2(1) - initial_state(2)];
```

```
% add terminal state constraint: land at origin
ceq = [ceq;
       x1(end);
       x2(end)];
end
```

I first define a function `double_integrator` that returns the continuous-time dynaimcs, and then in the nonlinear constraints function `double_integrator_nonlincon` I use `ode45` to simulate the dynamics starting at x_k from t_k to t_{k+1}:

```
[~,sol_int] = ode45(@(t,y) double_integrator(t,y,v),tt,xi);
```

The solution I got from `ode45` should be equal to my decision state variable.

Running the complete code with $x_0 = [-10; 0]$ and $N = 51$, I obtain the control signal in Fig. 3.6, which is bang-bang.

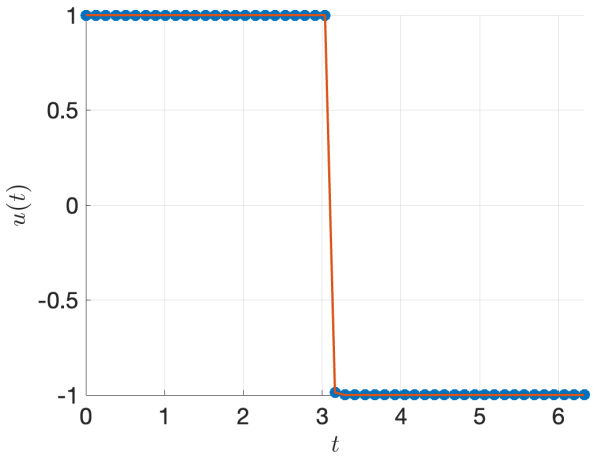


Figure 3.6: Optimal control signal by direct multiple shooting.

Using `ode45` to integrate the double integrator dynamics from x_0 with the controller in Fig. 3.6, we obtain the following state trajectory. Notice that the final state does not exactly land at the origin. This is expected due to our time discretization and imperfect dynamics integration using `ode45`. Make sure you play with the code, e.g., by changing the initial state x_0 and see what happens.

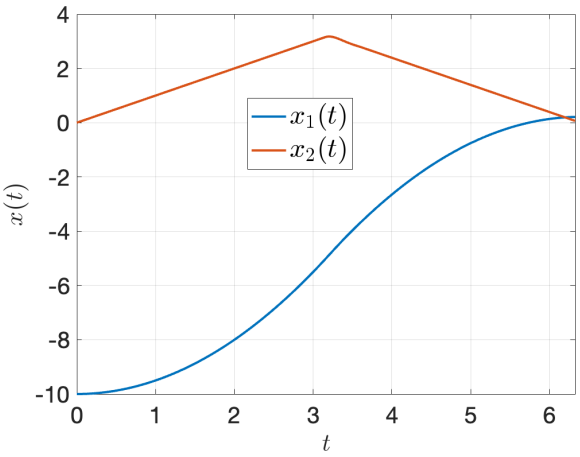


Figure 3.7: ODE45 integration with the optimal control signal found by direct multiple shooting.

3.2.3 Direct Collocation

In direct multiple shooting, we still need to rely on the numerical integrator RK4, which complicates the original nonlinear dynamics. In direct collocation, we will remove our dependency of RK4.

The key idea of direct collocation is to approximate the state trajectory $x(t)$ and the control trajectory $u(t)$ as piece-wise polynomial functions. In the following, we will describe the Hermite-Simpson collocation method.

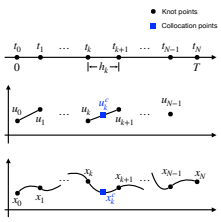


Figure 3.8: Direct collocation.

Time discretization. We first discretize the total time window $[0, T]$ into a set of N intervals:

$$0 = t_0 \leq t_1 \leq \cdots \leq t_k \leq t_{k+1} \leq \cdots \leq t_N = T.$$

We denote

$$h_k = t_{k+1} - t_k$$

as the length of the k-th time interval. As we will see, the length of the time interval does not need to be fixed, and instead they can themselves be unknown variables to be optimized (in which case the final time T also becomes flexible).

Knot variables. At each of the timestamps $t_0, \dots, t_N$, we assign *knot variables*, which are unknown state and control varaibles that need to be optimized. In particular, we have state knot variables

$$x_k = x(t_k), k = 1, \dots, N,$$

and control knot variables

$$u_k = u(t_k), k = 0, \dots, N-1.$$

As a result, the entire set of knot varaibles to be optimized is

$$v = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}. \tag{3.29}$$

If the time-discretization is also optimized, then v includes the time intervals as well.

Transcribe dynamics. The most important step is to transcribe the nonlinear dynamics $\dot{x} = f(x(t), u(t))$ as constraints on the knot varaibles. In direct collocation, the way this is done is to enforce the dynamics equation at the set of *collocation points* that are the mid-points between each consecutive pair of knot variables (x_k, x_{k+1}).

Specifically, in each subinterval $[t_k, t_{k+1}]$, we approximate the state trajectory as a cubic polynomial

$$x(t) = p_{k,0} + p_{k,1}(t - t_k) + p_{k,2}(t - t_k)^2 + p_{k,3}(t - t_k)^3, \quad t \in [t_k, t_{k+1}], \tag{3.30}$$

where $p_{k,0}, p_{k,1}, p_{k,2}, p_{k,3} \in \mathbb{R}^n$ are the coefficients of the polynomial. You would think that we would need to optimize the coefficients as well, but actually we won't need to, as will be shown soon. With this parameterization, we can obtain the time derivative of $x(t)$ as

$$\dot{x}(t) = p_{k,1} + 2p_{k,2}(t - t_k) + 3p_{k,3}(t - t_k)^2, \quad t \in [t_k, t_{k+1}]. \tag{3.31}$$

Now the key step is to write the coefficients $p_{k,0}, p_{k,1}, p_{k,2}, p_{k,3}$ using our knot variables (3.29). To do so, we can invoke (3.30) and (3.31) to obtain

$$\begin{bmatrix} x_k \\ \dot{x}_k = f(x_k, u_k) \\ x_{k+1} \\ \dot{x}_{k+1} = f(x_{k+1}, u_{k+1}) \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ I & h_k I & h_k^2 I & h_k^3 I \\ 0 & I & 2h_k I & 3h_k^2 I \end{bmatrix} \begin{bmatrix} p_{k,0} \\ p_{k,1} \\ p_{k,2} \\ p_{k,3} \end{bmatrix}.$$

Solving the above equation, we get

$$\begin{bmatrix} p_{k,0} \\ p_{k,1} \\ p_{k,2} \\ p_{k,3} \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ -\frac{3}{h_k^2} I & -\frac{2}{h_k} I & \frac{3}{h_k^2} I & -\frac{1}{h_k} I \\ \frac{2}{h_k^3} I & \frac{1}{h_k^2} I & -\frac{2}{h_k^3} I & \frac{1}{h_k^2} I \end{bmatrix} \begin{bmatrix} x_k \\ f(x_k, u_k) \\ x_{k+1} \\ f(x_{k+1}, u_{k+1}) \end{bmatrix}. \tag{3.32}$$

Equation (3.32) implies, using the knot variables v in (3.29), we can query the value of $x(t)$ and $\dot{x}(t)$ at any time $t \in [0, T]$. In particular, we will query the values of $x(t)$ and $\dot{x}(t)$ at the midpoints to obtain

$$x_k^c = x\left(t_k + \frac{h_k}{2}\right) = \frac{1}{2}(x_k + x_{k+1}) + \frac{h_k}{8}(f(x_k, u_k) - f(x_{k+1}, u_{k+1})),$$

and

$$\dot{x}_k^c = \dot{x}\left(t_k + \frac{h_k}{2}\right) = -\frac{3}{2h_k}(x_k - x_{k+1}) - \frac{1}{4}\left(f(x_k, u_k) + f(x_{k+1}, u_{k+1})\right).$$

At the midpoint, we assume the control is

$$u_k^c = \frac{1}{2}(u_k + u_{k+1}).$$

Therefore, we can enforce the dynamics constraint at the midpoint as

$$\dot{x}_k^c = f(x_k^c, u_k^c). \tag{3.33}$$

Transcribe other constraints. The other constraints in the continuous-time formulation (3.17) can be transcribed to the knot variables in a straightforward way:

$$\begin{aligned} x_k \in \mathcal{X} \Rightarrow c_i^x(x_k) \geq 0, i = 1, \dots, q_x, \quad k = 1, \dots, N \qquad & (3.34) \\ u_k \in \mathcal{X} \Rightarrow c_i^u(u_k) \geq 0, i = 1, \dots, q_u, \quad k = 0, \dots, N-1 \qquad & (3.35) \\ \phi_i(x_k, u_k) \geq 0, i = 1, \dots, q, \quad k = 0, \dots, N. \qquad & (3.36) \end{aligned}$$

Transcribe the objective. We can write the objective as

$$g_T(x_N) + \sum_{k=0}^{N-1} \frac{g(x_k, u_k) + g(x_{k+1}, u_{k+1})}{2} h_k.$$

Summary. In summary, the final optimization problem becomes

$$\begin{aligned} \min_{u_0,\dots,u_{N-1},x_1,\dots,x_N} \quad & g_T(x_N) + \sum_{k=0}^{N-1} \frac{g(x_k,u_k)+g(x_{k+1},u_{k+1})}{2} h_k \\ \text{subject to} \quad & \dot{x}_k^c = f(x_k^c, u_k^c), \quad k = 0,\dots,N-1 \\ & c_i^x(x_k) \geq 0, i = 1,\dots,q_x, \quad k = 1,\dots,N \\ & c_i^u(u_k) \geq 0, i = 1,\dots,q_u, \quad k = 0,\dots,N-1 \\ & \phi_i(x_k,u_k) \geq 0, i = 1,\dots,q, \quad k = 0,\dots,N. \end{aligned} \tag{3.37}$$

Let us apply direct collocation to the same double integrator Example 3.6.

Example 3.7 (Direct Collocation for Minimum-Time Double Integrator)**.** Consider the same minimum-time optimal control problem in Example 3.6.

To apply direct collocation, we have our NLP variable

$$v = (T, x_0, \dots, x_N, u_0, \dots, u_N).$$

Similarly we can enforce the control constraints and the initial / terminal constraints.

The following script shows how to enforce the collocation constraint.

```
function [c,ceq] = collocation(v,N,initial_state)
T = v(1);
h = T/(N-1);
x = reshape(v(2:2*N+1),2,N);
u = v(2*N+2:end);
c = [];
ceq = [];
for k=1:N-1
    uk = u(k);
    ukp1 = u(k+1);
    xk = x(:,k);
    xkp1 = x(:,k+1);
    fk = double_integrator(xk,uk);
    fkp1 = double_integrator(xkp1,ukp1);
    % collocation points
    xkc = 0.5*(xk+xkp1) + h/8 * (fk - fkp1);
    ukc = 0.5*(uk + ukp1);
    dxkc = -3/(2*h) * (xk-xkp1) - 0.25*(fk + fkp1);
    % collocation constraint
    ceq = [ceq;
           dxkc - double_integrator(xkc,ukc)];
```

```
end
ceq = [ceq;
       x(:,1) - initial_state; % initial condition
       x(:,end)]; % land at zero
end
```

As you can see, we do not need any numerical integrator such as `ode45`. The only thing we need is the continuous-time double integrator dynamics.

Running the complete code with $x_0 = (-10, 0)$ and $N = 51$, we get the control signal in Fig. 3.9 that is bang-bang.

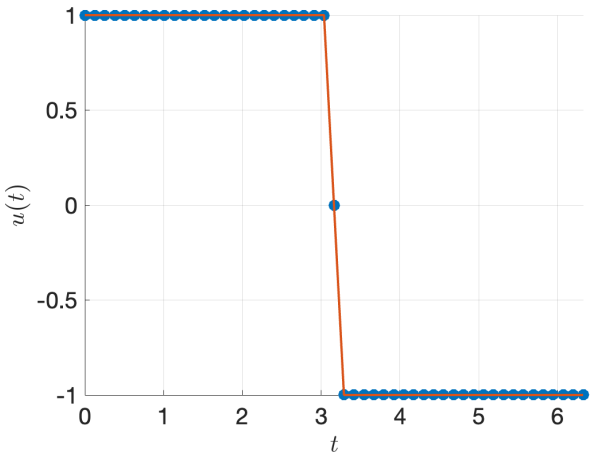


Figure 3.9: Optimal control signal by direct collocation.

Using `ode45` to integrate the double integrator dynamics from x_0 with the controller in Fig. 3.9, we obtain the following state trajectory. Comparing Fig. 3.10 with Fig. 3.7, we can observe that the terminal state of the trajectory obtained from direct collocation is more accurate than that obtained from direct multiple shooting.

Not only is direct collocation more accurate for this example, it is also faster. This is evident because in direct multiple shooting, evaluating the nonlinear constraints requires runing `ode45`, while in direct collocation, evaluating the nonlinear constraints simply requires calling the original continuous-time dynamics.

We can easily optimize with a larger $N = 101$ and get the following results.

If you are interested in direct collocation, there is a nice tutorial with Matlab examples in (Kelly, 2017).

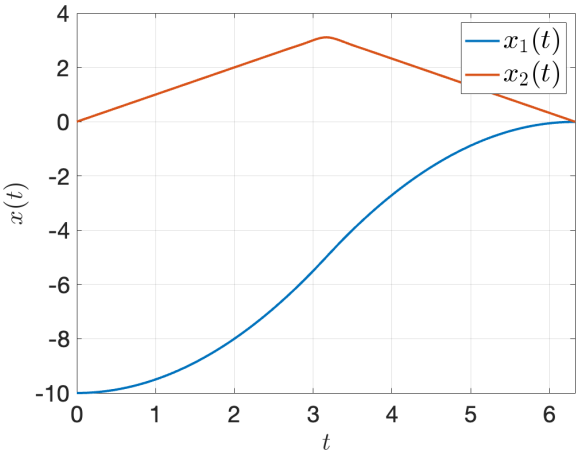


Figure 3.10: ODE45 integration with the optimal control signal found by direct collocation.

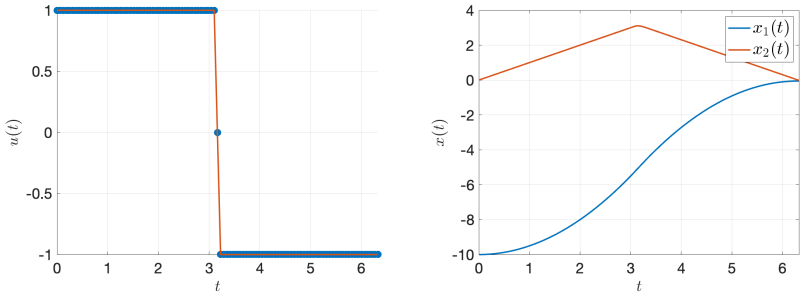


Figure 3.11: Direct collocation with N=101.

3.2.4 Direct Orthogonal Collocation

3.2.5 Failure of Open-Loop Control

Trajectory optimization produces very satisfying trajectories when the nonlinear programming algorithms work as expected, as shown by the previous Examples 3.7 and 3.6 on the double integrator system.

However, does this imply that if we simply run the planned controls on a real system, i.e., doing *open-loop control*, the trajectory will behave as planned?

Unfortunately the answer is No, for two major reasons.

1. Trajectory optimization only produces an *approximate* solution to the optimal control problem (OCP) (3.17), regardless of what transcription method is used (such as multiple shooting and direct collocation). This means every u_k we obtained is an approximation to the true optimal controller $u(t_k)$, and the approximation errors will accumulate as the dynamical system is evolving.

2. Even the OCP (3.17) is an imperfect approximation of the true control task. This is due to we assumed we have perfect knowledge of the system dynamics
$$\dot{x} = f(x(t), u(t)),$$
which rarely holds in practice. For example, in the double integrator example, there is always friction between the mass and the ground. A more realistic assumption is that the system dynamics is
$$\dot{x} = f(x(t), u(t)) + w_t,$$
where w_t is some unknown disturbance, or modeling error.

Let us observe the failure of open-loop control on our favorite pendulum example.

Example 3.8 (Failure of Open-Loop Control on A Simple Pendulum)**.** Consider an "ideal" pendulum dynamics model
$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}, \quad \dot{x} = f(x, u) = \begin{bmatrix} \dot{\theta} \\ -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta) + \frac{1}{ml^2}u \end{bmatrix},$$
with $m = 1, l = 1, g = 9.8, b = 0.1$. We enforce control saturation
$$u \in [-u_{\max}, u_{\max}]$$
with $u_{\max} = 4.9$.

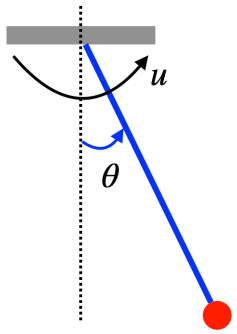


Figure 3.12: Simple pendulum.

Consider the optimal control problem with $T = 10$

$$\begin{aligned} \min_{u(t),t\in[0,T]} \quad & \int_{t=0}^{T} (\cos\theta(t)+1)^2 + \sin^2\theta(t) + \dot{\theta}^2(t) + u^2 dt \\ \text{subject to} \quad & \dot{x} = f(x(t),u(t)) \\ & x(0) = [0,0]^T, \quad x(T) = [\pi,0]^T, \\ & u(t) \in [-u_{\max}, u_{\max}], \end{aligned} \tag{3.38}$$

where we start from the initial state $[0,0]^T$, i.e., the bottomright position, and we want to swing up the pendulum to the final state $[\pi,0]^T$, i.e., the upright position. The cost function in (3.38) penalizes the deviation of the state trajectory from the target state (the target state has $\cos\theta = -1$, $\sin\theta = 0$ and $\dot{\theta} = 0$), together with the magnitude of control.

Trajectory optimization with direct collocation. We perform trajectory optimization for the OCP (3.38) with direct collocation. We choose $N = 101$ break points with $h = 0.1$ equal interval to discretize the time.

The result of trajectory optimization is shown in Fig. 3.13. We can see that the pendulum is perfectly swung up to $[\pi,0]^T$ and stabilized there.

Deploy the optimized plan. We then deploy the optimized controls in Fig. 3.13 on the "ideal" pendulum. We deploy the control every 0.1 seconds with zero-order hold, and use Matlab `ode89` to integrate the pendulum dynamics.

Fig. 3.14 shows the true trajectory of the pendulum. Unfortunately, the pendulum swing-up and stabilization is unsuccessful.

Adding noise. What if the true pendulum dynamics is

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}, \quad \dot{x} = f(x,u) = \begin{bmatrix} \dot{\theta} \\ -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta) + \frac{1}{ml^2}u \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

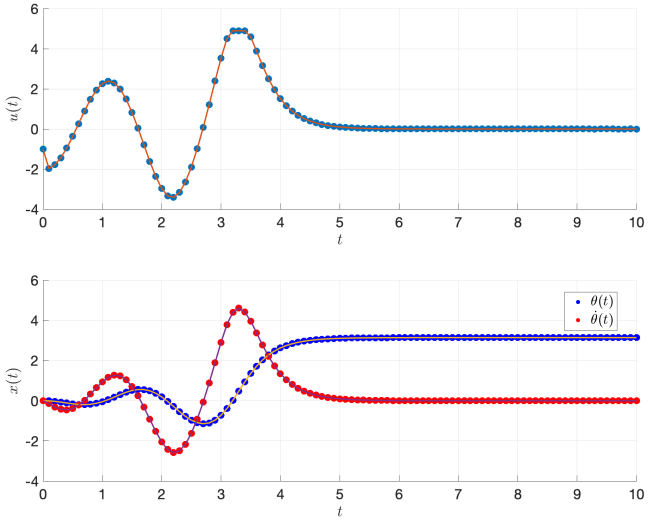


Figure 3.13: Pendulum swing-up with direct collocation

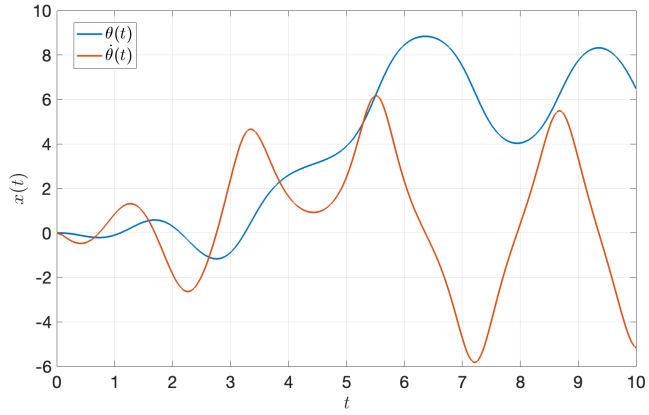


Figure 3.14: Deploy optimized controls

That is, there is a constant unmodelled angular acceleration of 1, which maybe come from some unknown external force.

Fig. 3.15 shows the trajectory of the pendulum with the optimized controlls. We can clearly see the unmodelled dynamics makes the performance much worse.

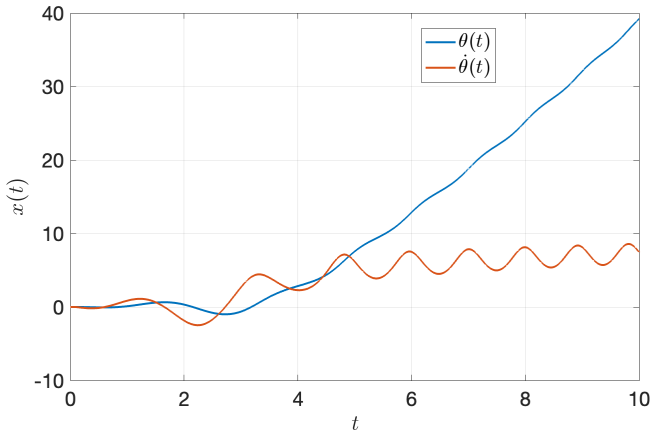


Figure 3.15: Deploy optimized controls on a system with disturbance

You can play with the code here.

The failure of open-loop control motivates feedback control.

3.2.6 LQR Trajectory Tracking

3.3 Model Predictive Control

The model predictive control framework leverages the idea of *receding horizon control* (RHC) to turn an open-loop controller into a closed-loop controller, i.e., a feedback controller.

For example, suppose we have an optimal control problem with horizon T:

$$\begin{aligned} \min_{u(t),t\in[0,T]} \quad & g_T(x(T)) + \int_{t=0}^{T} g(x(t),u(t))dt \\ \text{subject to} \quad & \dot{x} = f(x(t),u(t)), \quad x(0) = x_0 \\ & (x(t),u(t)) \in \mathcal{X}\times\mathcal{U}, \quad x(T) \in \mathcal{X}_T \end{aligned} \tag{3.39}$$

where $\dot{x} = f(x(t),u(t))$ is the best "ideal" model we have about our system.

The idea of RHC is as follows. At time t, we obtain a measurement of the current state of the system, denoted as x_t, and we solve the following OCP with horizon $H < T$:

$$\begin{aligned} \min_{u(\tau),\tau\in[0,H]} \quad & g_t(x(H)) + \int_{\tau=0}^{H} g(x(\tau),u(\tau))d\tau \\ \text{subject to} \quad & \dot{x} = f(x(\tau),u(\tau)), \quad x(0) = x_t \\ & (x(t),u(t)) \in \mathcal{X}\times\mathcal{U}, \quad x(H) \in \mathcal{X}_t \end{aligned} \tag{3.40}$$

where notice that I used a different notation for the terminal cost g_t (as supposed to g_T in (3.39)), and the terminal constraint $\mathcal{X}_t$ (as supposed to $\mathcal{X}_T$ in (3.40)). Using a different terminal cost and terminal constraint is meant to make the problem (3.40) always feasible.

Suppose we solve the open-loop control problem (3.40) using trajectory optimization with N time intervals and obtained the following solution

$$u_{t,0}, u_{t,1}, \dots, u_{t,N}.$$

Then RHS will only take the first control $u_{t,0}$ and apply it to the system. Therefore, the closed-loop dynamics is effectively

$$\dot{x} = f(x(t), u_{t,0}), \tag{3.41}$$

where $u_{t,0}$ is the first control solution to the OCP (3.40).

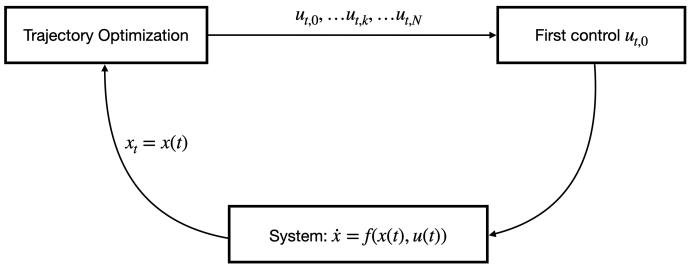


Figure 3.16: Receding horizon control

It is worth noting that RHS effectively introduces feedback control, because the controller $u_{t,0}$ in (3.41) depends on $x(t)$ via the open-loop optimal control problem (3.40).

Trajectory optimization and model predictive control are the workhorse for control of highly dynamic robots. For example, here is a talk by Scott Kuindersma on how trajectory optimization and model predictive control enable a diverse set of behaviors of Boston Dynamics's humanoid robot.

3.3.1 Turn Trajectory Optimization into Feedback Control

Let us turn our trajectory optimization algorithm in Example 3.8 into a feedback controller through receding horizon control.

Example 3.9 (Pendulum Swingup with Model Predictive Control)**.** Consider again the pendulum swingup task in Example 3.8.

This time, we will apply MPC to this task.

Again, we discretize the time windown $[0, T]$ into $N - 1 = 100$ equal intervals with $h = 0.1$. At each timestep $t_k = kh$, $k = 0, \dots, N-1$, we first measure the current state of the pendulum as x_k, then solve the following OCP with planning horizon $H = 5$

$$\begin{aligned} \min_{u(t), t\in[0,H]} \quad & \int_{t=0}^{H} (\cos\theta(t) + 1)^2 + \sin^2\theta(t) + \dot{\theta}^2(t) + u^2 dt \\ \text{subject to} \quad & \dot{x} = f(x(t), u(t)) \\ & x(0) = x_k, \quad x(H) = [\pi, 0]^T, \\ & u(t) \in [-u_{\max}, u_{\max}]. \end{aligned} \tag{3.42}$$

We solve the OCP (3.42) using trajectory optimization with direct collocation (using M break points), which gives us a sequence of controls

$$u_{k,1}, \dots, u_{k,M},$$

we deploy the first control $u_{k,1}$ to the pendulum system.

MPC results. Fig. 3.17 shows the control and state trajectories when using MPC to swing up the pendulum. We can see that it works very well.

Robustness to noise. MPC is naturally robust to modelling errors and noises. When the true system dynamics is

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}, \quad \dot{x} = f(x, u) = \begin{bmatrix} \dot{\theta} \\ -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta) + \frac{1}{ml^2}u \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The MPC controller still works very well, as shown in Fig. 3.18!

Feel free to play with the code here.

Software tools. Matlab implements a nice package for nonlinear MPC, you can refer to the documentation and examples. In Python, the do-mpc package is quite popular.

However, it is important to recognize that trajectory optimization for the non-convex open-loop optimal control problen (3.40) (a) only guarantees a locally optimal solution and hence can be brittle, (b) can be time-consuming when the problem is high-dimensional.

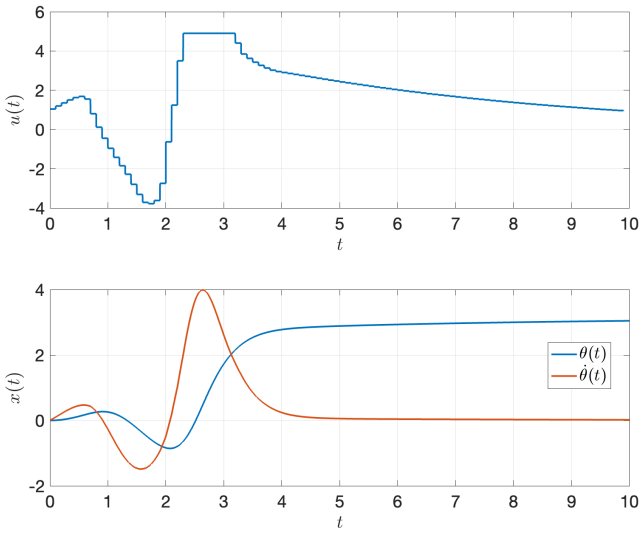


Figure 3.17: MPC for pendulum swing-up

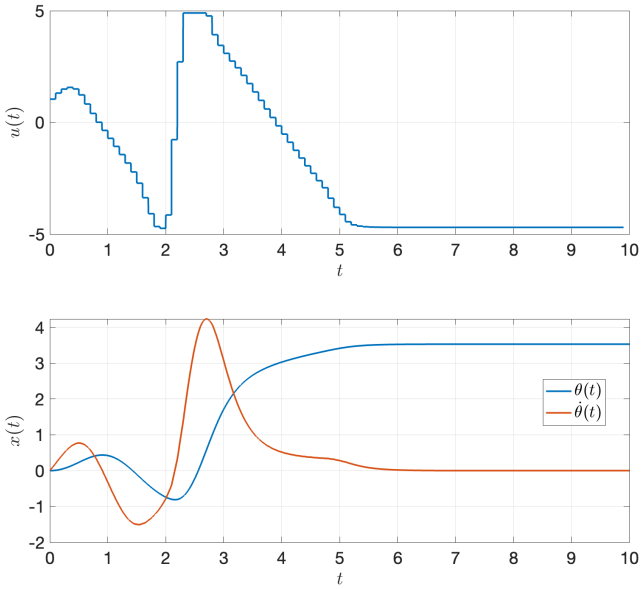


Figure 3.18: MPC for pendulum swing-up with noise

In fact, MPC was originally developed in the context of chemical plant control (Borrelli et al., 2017), which has linear system dyanmics, as we will introduce soon. Before we introduce the MPC formulation and understand its theoretical properties, we need to take a detour and introduce various important notions of sets.

3.3.2 Controllability, Reachability, and Invariance

Consider the autonomous system

$$x_{t+1} = f_a(x_t) \tag{3.43}$$

and the controlled system

$$x_{t+1} = f(x_t, u_t) \tag{3.44}$$

for $t \geq 0$. Both systems are subject to state and input constraints

$$x_t \in \mathcal{X}, \quad u_t \in \mathcal{U}, \forall t \geq 0, \tag{3.45}$$

with $\mathcal{X}$ and $\mathcal{U}$ both being polyhedral sets (see Definition B.9)

$$\begin{aligned} x_t \in \mathcal{X} = \{x \in \mathbb{R}^n \mid Cx \leq c\}, \quad C \in \mathbb{R}^{l_x \times n}, c \in \mathbb{R}^{l_x}, \\ u_t \in \mathcal{U} = \{u \in \mathbb{R}^m \mid Du \leq d\}, \quad D \in \mathbb{R}^{l_u \times m}, d \in \mathbb{R}^{l_u}. \end{aligned} \tag{3.46}$$

We are going to make a few definitions.

3.3.2.1 Controllable and Reachable Sets

Definition 3.1 (Precursor Set)**.** For the autonomous system (3.43), we define the precursor set to a set $\mathcal{S}$ as

$$\text{Pre}(\mathcal{S}) = \{x \in \mathbb{R}^n \mid f_a(x) \in \mathcal{S}\}.$$

In words, $\text{Pre}(\mathcal{S})$ is the set of states which evolve into the target set $\mathcal{S}$ in one time step.

For the controlled system (3.44), we define the precursor set to the set $\mathcal{S}$ as

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

In words, $\text{Pre}(\mathcal{S})$ is the set of states that can be driven into the target set $\mathcal{S}$ while satisfying control and state constraints.

A related concept is the successor set.

Definition 3.2 (Successor Set)**.** For the autonomous system (3.43), we define the successor set to a set $\mathcal{S}$ as

$$\text{Suc}(\mathcal{S}) = \{x \in \mathbb{R}^n \mid \exists x' \in \mathcal{S} \text{ s.t. } x = f_a(x')\}.$$

In words, the states in $\mathcal{S}$ get mapped into the states in $\text{Suc}(\mathcal{S})$ after one time step.

For the controlled system (3.44), we define the successor set to the set $\mathcal{S}$ as

$$\text{Suc}(\mathcal{S}) = \{x \in \mathbb{R}^n \mid \exists x' \in \mathcal{S}, \exists u' \in \mathcal{U} \text{ s.t. } x = f(x', u').\}$$

In words, the states in $\mathcal{S}$ get mapped into the states in $\text{Suc}(\mathcal{S})$ after one time step while satisfying the control constraints.

The N-step controllable set is defined by iterating $\text{Pre}(\cdot)$ computations.

Definition 3.3 (N-Step Controllable Set)**.** Let $\mathcal{S} \subseteq \mathcal{X}$ be a target set. The N-step controllable set $\mathcal{K}_N(\mathcal{S})$ of the system (3.43) or (3.44) subject to the constraints (3.45) is defined recursively as:

$$\mathcal{K}_0(\mathcal{S}) = \mathcal{S}, \quad \mathcal{K}_i(\mathcal{S}) = \text{Pre}(\mathcal{K}_{i-1}(\mathcal{S})) \cap \mathcal{X}, i = 1, \dots, N.$$

According to this definition, given any $x_0 \in \mathcal{K}_N(\mathcal{S})$:

- For the autonomous system (3.43), its state will evolve into the target set $\mathcal{S}$ in N steps while satisfying state constraints
- For the controlled system (3.44), there exists a sequence of admissible controls (i.e., satisfying control constraints) such that its state will evolve into the target set $\mathcal{S}$ in N steps while satisfying state constraints.

Similarly, the N-step reachable set is defined by iterating $\text{Suc}(\cdot)$ computations.

Definition 3.4 (N-Step Reachable Set)**.** Let $\mathcal{X}_0 \subseteq \mathcal{X}$ be an initial set. The N-step reachable set $\mathcal{R}_N(\mathcal{X}_0)$ of the system (3.43) or (3.44) subject to the constraints (3.45) is defined recursively as:

$$\mathcal{R}_0(\mathcal{X}_0) = \mathcal{X}_0, \quad \mathcal{R}_i(\mathcal{X}_0) = \text{Suc}(\mathcal{R}_{i-1}(\mathcal{X}_0)) \cap \mathcal{X}, i = 1, \dots, N.$$

According to this definition, given any $x_0 \in \mathcal{X}_0$:

- For the autonomous system (3.43), its state will evolve into $\mathcal{R}_N(\mathcal{X}_0)$ in N steps while satisfying state constraints
- For the controlled system (3.44), there exists a sequence of admissible controls (i.e., satisfying control constraints) such that its state will evolve into $\mathcal{R}_N(\mathcal{X}_0)$ in N steps while satisfying state constraints.

In the literature, the controllable set is often referred to as the *backwards reachable set*.

3.3.2.2 Computation of Controllable and Reachable Sets

For a linear time-invariant system, when the state constraint set $\mathcal{X}$, control constraint set $\mathcal{U}$, the target set $\mathcal{S}$, and the initial set $\mathcal{X}_0$ are all polytopes, then the N-step controllable set $\mathcal{K}_N(\mathcal{S})$ and the N-step reachable set $\mathcal{R}_N(\mathcal{X}_0)$ are both polytopes and can be computed exactly and efficiently.

We will not describe the underlying algorithm for computing $\mathcal{K}_N(\mathcal{S})$ and $\mathcal{R}_N(\mathcal{X}_0)$ (these details can be found in (Borrelli et al., 2017) and they are not very difficult to understand, so I suggest you to read them), but we now show you how to use the Multi-Parametric Toolbox (MPT) to compute them.

Example 3.10 (Compute Controllable and Reachable Sets)**.** Consider a linear system

$$x_{t+1} = \begin{bmatrix} 1.5 & 0 \\ 1 & -1.5 \end{bmatrix} x_t + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_t$$

with state and control constraints

$$\mathcal{X} = [-10, 10]^2, \quad \mathcal{U} = [-5, 5].$$

Given $\mathcal{S} = \mathcal{X}_0 = [-1, 1]^2$, I want to compute $\mathcal{K}_i(\mathcal{S})$ and $\mathcal{R}_i(\mathcal{X}_0)$, for $i = 0, 1, 2, 3, 4$.

Dynamical system. We first define the linear time-invariant system as follows.

```
A = [1.5, 0; 1, -1.5]; B = [1; 0];
sys = LTISystem('A',A,'B',B);
```

We then define the state and control constraints.

```
calX = Polyhedron('A',...
    [1,0;0,1;-1,0;0,-1], ...
    'b',[10;10;10;10]);
calU = Polyhedron('A',[1;-1],'b',[5;5]);
```

Note that in the MPT toolbox, to define a polytope $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, we just need to call the `Polyhedron` function with inputs `A` and `b`.

Controllable sets. We have from Definition 3.3 the recursion

$$\mathcal{K}_0(\mathcal{S}) = \mathcal{S}, \quad \mathcal{K}_i(\mathcal{S}) = \text{Pre}(\mathcal{K}_{i-1}(\mathcal{S})) \cap \mathcal{X}.$$

To implement the above set operation, we need two functions, one for the "$\cap$" intersection operation, and the other for the "Pre$(\cdot)$" precursor set operation. These two functions are both available through MPT.

- `intersect(P,S)` computes the intersection of two sets `P` and `S`, both defined as polytopes.
- `system.reachableSet('X', X, 'U', U, 'N', 1, 'direction', 'backwards')` computes the one-step backwards reachable set (which is just the precursor set) of the `system` with target set `X` and control constraints `U`.

Using these two functions, the following code snippet computes recursively the controllable sets for N steps.

```
% target set
S = Polyhedron('A',[1,0;0,1;-1,0;0,-1],'b',[1;1;1;1]);
K = [S];
N = 4;
for i = 1:N
    Si = sys.reachableSet('X',K(i),'U',calU,'N',1,'direction','backwards');
    K = [K; intersect(Si,calX)];
end
```

Fig. 3.19 plots the controllable sets computed by running the above code.

Reachable sets. We have Definition 3.4 the recursion

$$\mathcal{R}_0(\mathcal{X}_0) = \mathcal{X}_0, \quad \mathcal{R}_i(\mathcal{X}_0) = \mathrm{Suc}(\mathcal{R}_{i-1}(\mathcal{X}_0)) \cap \mathcal{X}.$$

To implement the recursion above, we need "$\cap$" set intersection, which is available via `intersect(P,S)`, and the "Suc$(\cdot)$" successor set operation, which is available as

- `system.reachableSet('X', X, 'U', U, 'N', 1, 'direction', 'forward')` computes the one-step forward reachable set (which is just the successor set) of the `system` with initial set `X` and control constraints `U`.

Therefore, we can compute the reachable sets recursively using the following code snippet

```
% initial set
X0 = Polyhedron('A',[1,0;0,1;-1,0;0,-1],'b',[1;1;1;1]);
R = [X0];
N = 4;
for i = 1:N
    Ri = sys.reachableSet('X',R(i),'U',calU,'N',1,'direction','forward');
    R = [R; intersect(Ri,calX)];
end
```

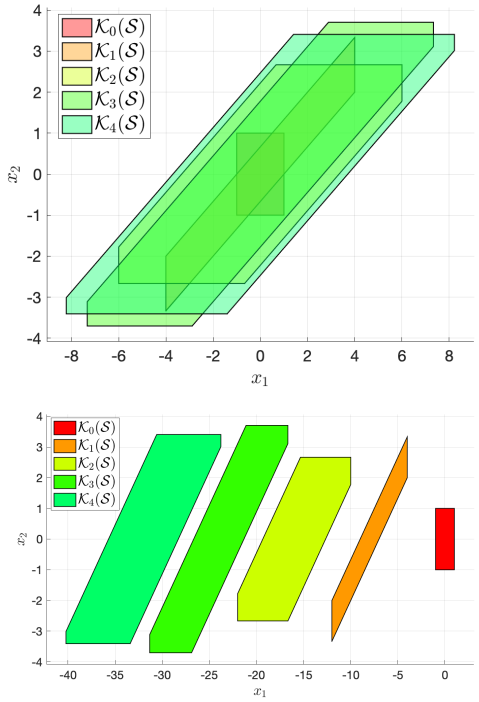


Figure 3.19: Computation of controllable sets using MPT. The bottom plot shifts the sets horizontally for better visualization.

Fig. 3.20 plots the reachable sets computed by running the above code.

Feel free to play with the code here. Note that you need to install the MPT toolbox before running the code.

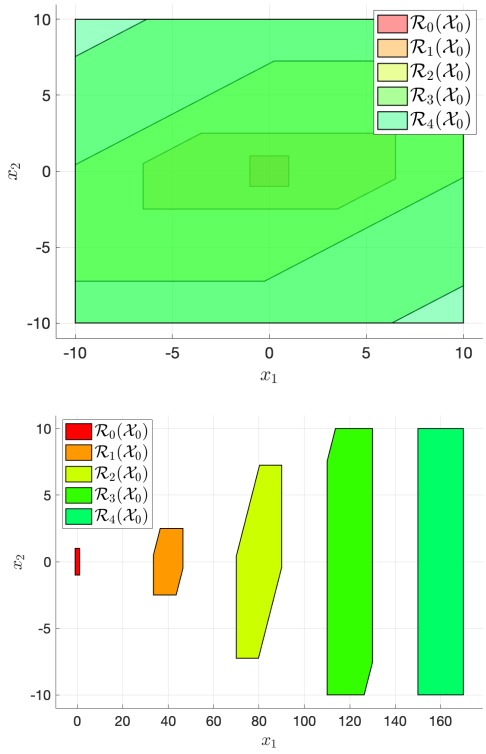


Figure 3.20: Computation of reachable sets using MPT. The bottom plot shifts the sets horizontally for better visualization.

3.3.2.3 Invariant Sets

Definition 3.5 (Positive Invariant Set)**.** A set $\mathcal{O} \subseteq \mathcal{X}$ is said to be a positive invariant set for the autonomous system (3.43) subject to the constraints in (3.45) if

$$x_0 \in \mathcal{O} \Longrightarrow x_t \in \mathcal{O}, \forall t \geq 0.$$

That is, if the system starts in $\mathcal{O}$, it stays in $\mathcal{O}$ for all future timesteps.

The maximal positive invariant set is the largest positive invariant set.

Definition 3.6 (Maximal Positive Invariant Set)**.** A set $\mathcal{O}_\infty \subseteq \mathcal{X}$ is said to be the maximal positive invariant set for the autonomous system (3.43) subject to

the constraints in (3.45) if (a) $\mathcal{O}_\infty$ is positive invariant and (b) $\mathcal{O}_\infty$ contains all the invariant sets contained in $\mathcal{X}$.

Essentially, the (maximal) positive invariant set is the set of initial conditions under which the system does not blow up.

For the controlled system (3.44), we have the similar notion of a control invariant set.

Definition 3.7 (Control Invariant Set)**.** A set $\mathcal{C} \subseteq \mathcal{X}$ is said to be a control invariant set for the controlled system (3.44) subject to the constraints in (3.45) if

$$x_0 \in \mathcal{C} \Longrightarrow \exists u_t \in \mathcal{U} \text{ s.t. } f(x_t, u_t) \in \mathcal{C}, \forall t \geq 0$$

That is, if the system starts in $\mathcal{C}$, it can be controlled to stay in $\mathcal{C}$ for all future time steps.

The maximal control invariant set is the largest control invariant set.

Definition 3.8 (Maximal Control Invariant Set)**.** A set $\mathcal{C}_\infty \subseteq \mathcal{X}$ is said to be the maximal control invariant set for the controlled system (3.44) subject to the constraints in (3.45) if (a) $\mathcal{C}_\infty$ is control invariant, and (b) $\mathcal{C}_\infty$ contains all control invariant sets contained in $\mathcal{X}$.

Essentially, the (maximal) control invariant set is the set of initial conditions under which the system can be controlled to not blow up.

We now state a necessary and sufficient condition for a set to be positive invariant and control invariant.

Theorem 3.1 (Geometric Condition for Invariance)**.** *For the autonomous system* (3.43) *subject to the constraint* (3.45)*, a set $\mathcal{O} \subseteq \mathcal{X}$ is positive invariant if and only if*

$$\mathcal{O} \subseteq \mathit{Pre}(\mathcal{O}). \tag{3.47}$$

Similarly, for the controlled system (3.44) *subject to the constraint* (3.45)*, a set $\mathcal{C}$ is control invariant if and only if*

$$\mathcal{C} \subseteq \mathit{Pre}(\mathcal{C}). \tag{3.48}$$

Proof. We only prove (3.47) since (3.48) can be proved using similar arguments.

- "$\Leftarrow$": we want to show $\mathcal{O}$ is positive invariant if (3.47) holds. We prove this by contradiction. If $\mathcal{O}$ is not positive invariant, the $\exists \hat{x} \in \mathcal{O}$ such that $f_a(\hat{x}) \notin \mathcal{O}$. This implies we have found $\hat{x} \in \mathcal{O}$ but $\hat{x} \notin \text{Pre}(\mathcal{O})$, contradicting (3.47).

- "⇒": we want to show (3.47) holds if $\mathcal{O}$ is positive invariant. We prove this by contradiction. Suppose (3.47) does not hold, then $\exists \hat{x}$ such that $\hat{x} \in \mathcal{O}$ but $\hat{x} \notin \text{Pre}(\mathcal{O})$. This implies we have found $\hat{x} \in \mathcal{O}$ that does not remain in $\mathcal{O}$ in the next step, contradicting $\mathcal{O}$ being positive invariant.

This shows that (3.47) is a sufficient and necessary condition. □

Theorem 3.1 immediately suggests an algoithm for computing (control) invariant sets, as we will describe in the next section.

3.3.2.4 Computation of Invariant Sets

Observe that the geometric conditions (3.47) and (3.48) are equivalent to the following conditions

$$\mathcal{O} = \text{Pre}(\mathcal{O}) \cap \mathcal{O}, \quad \mathcal{C} = \text{Pre}(\mathcal{C}) \cap \mathcal{C}.$$

Based on the equation above, we can design an algoithm that iteratively evaluates $\text{Pre}(\Omega) \cap \Omega$ until it converges.

Algorithm: Computation of $\mathcal{O}_\infty$

Input: f_a, $\mathcal{X}$

Output: $\mathcal{O}_\infty$

$\Omega_0 \leftarrow \mathcal{X}$, $k = 0$

Repeat

$\Omega_{k+1} \leftarrow \text{Pre}(\Omega_k) \cap \Omega_k$

$k \leftarrow k + 1$

Until $\Omega_{k+1} = \Omega_k$.

$\mathcal{O}_\infty \leftarrow \Omega_k$

The algorithm above generates a sequence of sets $\{\Omega_k\}$ satisfying $\Omega_{k+1} \subseteq \Omega_k$ for any k, and it terminates when $\Omega_{k+1} = \Omega_k$. If it terminates, then Ω_k is the maximal positive invariant set $\mathcal{O}_\infty$. If $\Omega_k = \emptyset$ for some intege k then $\mathcal{O}_\infty = \emptyset$.

In general, the algoithm above may never terminate. If the algoithm does not terminate in a finite number of iterations, then it can be proven that (Kolmanovsky et al., 1998)

$$\mathcal{O}_\infty = \lim_{k \to \infty} \Omega_k.$$

Conditions for finite time termination of the algoithm can be found in (Gilbert and Tan, 1991). A simple sufficient condition requires the dynamics f_a to be linear and stable, and the constraint set $\mathcal{X}$ to be bounded and contain the origin.

The same algoithm can be used to compute the maximal control invariant set $\mathcal{C}_\infty$.

Algorithm: Computation of $\mathcal{C}_\infty$

Input: f, $\mathcal{X}$, $\mathcal{U}$

Output: $\mathcal{C}_\infty$

$\Omega_0 \leftarrow \mathcal{X}$, $k = 0$

Repeat

$\Omega_{k+1} \leftarrow \text{Pre}(\Omega_k) \cap \Omega_k$

$k \leftarrow k + 1$

Until $\Omega_{k+1} = \Omega_k$.

$\mathcal{C}_\infty \leftarrow \Omega_k$

Similarly, the above algoithm generates $\{\Omega_k\}$ such that $\Omega_{k+1} \subseteq \Omega_k$ for any k. If the algoithm terminates, then $\Omega_k = \mathcal{C}_\infty$.

In general, the algoithm may never terminate. If the algoithm does not terminate, then in general convergence is not guaranteed

$$\mathcal{C}_\infty \neq \lim_{k \to \infty} \Omega_k.$$

The work in (Bertsekas, 1972) reports examples of nonlinear systems for which the above equation can be observed. A sufficient condition for the convergence

of Ω_k to $\mathcal{C}_\infty$ as $k \to \infty$ requires the polyhedral sets $\mathcal{X}$ and $\mathcal{U}$ to be bounded and the system $f(x, u)$ to be continuous (Bertsekas, 1972).

Let us apply the algoithm to compute the maximal control invariant set for the linear system in Example 3.10.

Example 3.11 (Computation of the Maximal Control Invariant Set)**.** Consider the linear system in Example 3.10 with same state constraint and control constraint.

The following code snippet shows how to apply the iterative algorithm introduced above to compute the maximal control invariant set.

```
Omega = [calX];
while true
    last_Omega = Omega(end);
    pre_omega = sys.reachableSet('X',last_Omega,'U',calU,'N',1,...
        'direction','backwards');
    new_Omega = intersect(pre_omega,last_Omega);
    Omega = [Omega;new_Omega];
    if new_Omega == last_Omega
        fprintf("Converged to maximal control invariant set.\n");
        break;
    end
end
```

The algoithm converges in 37 iterations and Fig. 3.21 plots the sequence of sets generated by the algorithm.

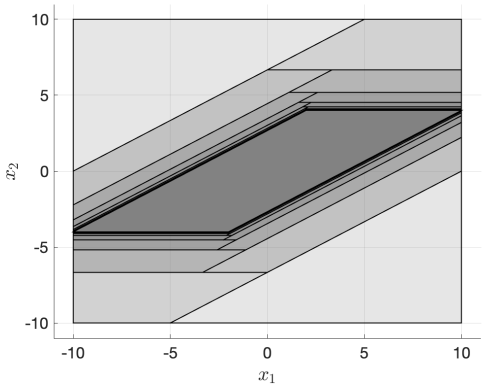


Figure 3.21: Maximal control invariant set.

The MPT toolbox actually implements this algoithm for us to use directly. If we use

```
C = sys.invariantSet('X', calX, 'U', calU);
```

we get the same result as before.

You can play with the code here.

3.3.3 Basic Formulation for Linear Systems

We are now ready to introduce the basic formulation of MPC for linear systems and study its theoretical properties.

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

$$x_{t+1} = Ax_t + Bu_t, \tag{3.49}$$

where the state x_t and control u_t are constrained to lie in polyhedral sets $\mathcal{X}$ and $\mathcal{U}$, respectively. We assume $\mathcal{X}$ contains the origin 0.

We can formulate the following optimal control problem to regulate the system to the origin

$$\begin{aligned} J^\star(x_0) = \min_{u_t, t=0,\dots} \quad & \sum_{t=0}^{\infty} x_t^T Q x_t + u_t^T R u_t \\ \text{subject to} \quad & x_{t+1} = Ax_t + Bu_t, \\ & (u_t, x_t) \in \mathcal{U} \times \mathcal{X}, \forall t = 0, \dots \end{aligned} \tag{3.50}$$

with $Q \succeq 0, R \succ 0$. Had we not included the constraints $(u_t, x_t) \in \mathcal{U} \times \mathcal{X}$, then problem (3.50) is exactly the infinite-horizon LQR problem in Section 2.1.1, for which we know the optimal controller is $u_t = -Kx_t$ with K computed in closed-form as (2.13).

However, in the presence of constraints, problem (3.50) does not admit a simple closed-form solution. In fact, problem (3.50) is commonly referred to as the *contrained LQR* (CLQR) problem and it is known that the optimal controller is *piece-wise affine*, see Theorem 11.4 in (Borrelli et al., 2017). We have asked you to numerically play with a toy example of CLQR in Exercise 5.2.

Receding horizon control. Leveraging the receding horizon control framework, we can approach problem (3.50) by online solving convex optimization problems.

At time t, suppose we can measure the current state of the system x_t, then we

solve the following optimal control problem with a finite horizon N

$$\begin{aligned} J_t^\star(x_t) = \min_{u(0),\dots,u(N-1)} \quad & p(x(N)) + \sum_{k=0}^{N-1} q(x(k),u(k)) \\ \text{subject to} \quad & x(k+1) = Ax(k) + Bu(k), k = 0,\dots,N-1, \quad x(0) = x_t \\ & (x(k),u(k)) \in \mathcal{X} \times \mathcal{U}, k = 0,\dots,N-1 \\ & x(N) \in \mathcal{X}_f, \end{aligned} \tag{3.51}$$

and $p(x)$, $q(x,u)$ convex functions. For example, a simple choice is $p(x) = x^T P x$ and $q = x^T Q x + u^T R u$ with $P, Q \succeq 0$ and $R \succ 0$. I hope you could pay attention to the notation in (3.51). I used x_t, u_t to denote the state and control for the original linear system (3.49) at time t, as well as in the CLQR problem (3.50). However, in every subproblem (3.51) of RHS at time t, I used $x(k), u(k)$, with k as the time step, to denote the state and control in the finite-horizon optimal control problem starting at x_t, with $x_t = x(0)$ (k is the shifted time horizon in RHS that always starts at zero). In addition to the difference in notation between (3.51) and (3.50), problem (3.51) is also different in the following two ways:

1. Terminal cost $p(x)$: in the objective of problem (3.51), there is an additional terminal cost $p(x(N))$.
2. Terminal constraint set $\mathcal{X}_f$: problem (3.51) has an additional constraint that the final state $x(N)$ must belong to the set $\mathcal{X}_f$. We assume $\mathcal{X}_f$ is also polyheral (and convex).

Feasible sets. We denote as $\mathcal{X}_0 \subseteq \mathcal{X}$ the set of initial states x_t such that the RHC subproblem (3.51) is feasible, i.e.,

$$\begin{aligned} \mathcal{X}_0 = \{x(0) \in \mathbb{R}^n \mid \exists (u(0),\dots,u(N-1)) \text{ such that } & x(k) \in \mathcal{X}, u(k) \in \mathcal{U}, k = 0,\dots,N-1, \\ & x(N) \in \mathcal{X}_f \text{ with } x(k+1) = Ax(k) + Bu(k), k = 0,\dots,N-1\}. \end{aligned} \tag{3.52}$$

Similarly, we denote as $\mathcal{X}_i$ the set of states such that the RHC subproblem is feasible from step $k = i$:

$$\begin{aligned} \mathcal{X}_i = \{x(i) \in \mathbb{R}^n \mid \exists (u(i),\dots,u(N-1)) \text{ such that } & x(k) \in \mathcal{X}, u(k) \in \mathcal{U}, k = i,\dots,N-1, \\ & x(N) \in \mathcal{X}_f \text{ with } x(k+1) = Ax(k) + Bu(k), k = i,\dots,N-1\}. \end{aligned} \tag{3.53}$$

Clearly, by definition we have

$$\mathcal{X}_N = \mathcal{X}_f,$$

and

$$\mathcal{X}_i = \{x \in \mathcal{X} \mid \exists u \in \mathcal{U} \text{ such that } Ax + Bu \in \mathcal{X}_{i+1}\}, i = 0,\dots,N-1,$$

or written in a compact way as

$$\mathcal{X}_i = \text{Pre}(\mathcal{X}_{i+1}) \cap \mathcal{X}. \tag{3.54}$$

Note that from equation (3.54) we have that, if we pick $x_i \in \mathcal{X}_i$ and let $U(x_i)$ be the set of feasible controls at x_i, then pick any $(u(0), \dots, u(N-1)) \in U(x_i)$ and apply $u(0)$ to the system to get $x_{i+1} = Ax_i + Bu(0)$, we have $x_{i+1} \in \mathcal{X}_{i+1}$.

Since the definitions (3.52) and (3.53) are rather abstract, let us visualize them using the double integrator example.

Example 3.12 (Double Integrator RHC Feasible Sets)**.** Consider the discrete-time double integrator dynamics

$$x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_t,$$

subject to control constraint

$$u \in \mathcal{U} = [-0.5, 0.5],$$

and state constraint

$$x \in \mathcal{X} = [-5, 5] \times [-5, 5].$$

We use $N = 3$ and visualize the feasible sets (3.52) and (3.53) for two choices of $\mathcal{X}_f$.

Choice 1. $\mathcal{X}_f = \mathcal{X}$ is the full state space. We use the recursion (3.54) to compute the feasible sets $\mathcal{X}_i$ for $i = 0, 1, 2, 3$. Fig. 3.22 plots the feasible sets. Observe that in this case $\mathcal{X}_0 \subset \mathcal{X}_1 \subset \mathcal{X}_2 \subset \mathcal{X}_3$. This creates a concern: suppose the RHC starts at $x_0 \in \mathcal{X}_0$ that is feasible, in the next iteration we have $x_1 \in \mathcal{X}_1$. However, since $\mathcal{X}_1$ is larger than $\mathcal{X}_0$, x_1 is not guaranteed to be feasible.

Choice 2. $\mathcal{X}_f = \{(0, 0)\}$ is the origin. Fig. 3.23 plots the feasible sets. Observe that in this case $\mathcal{X}_3 \subset \mathcal{X}_2 \subset \mathcal{X}_1 \subset \mathcal{X}_0$. This is a nice case, because if RHC starts at $x_0 \in \mathcal{X}_0$ that is feasible, in the next iteration we have $x_1 \in \mathcal{X}_1 \subset \mathcal{X}_0$, which implies that x_1 is guaranteed to remain feasible!

In fact, as we will soon show, in the first choice the RHC does suffer from infeasibility.

The code for this example can be found here.

Algorithm. Above all, problem (3.51) is a convex optimization problem that we know how to solve efficiently (you have solved such convex optimization problems in Exercise 5.2).

Let $u^\star(0), \dots, u^\star(N-1)$ be the optimal solution of problem (3.51) when it is feasible, the RHS framework will only apply the first control $u^\star(0)$ to the system, and hence the closed-loop system is

$$x_{t+1} = Ax_t + u^\star_{x_t}(0) = f_{\text{cl}}(x_t), \tag{3.55}$$

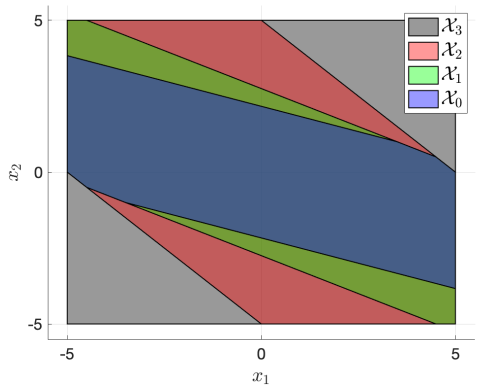


Figure 3.22: Feasible sets of the double integrator receding horizon controller without terminal constraint.

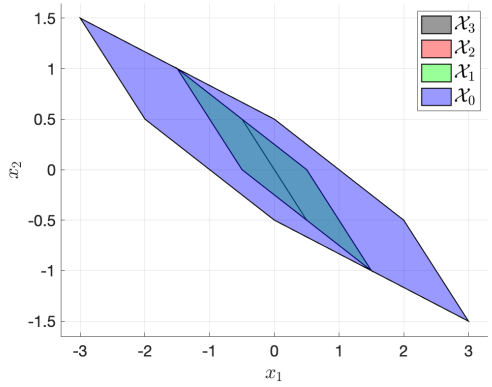


Figure 3.23: Feasible sets of the double integrator receding horizon controller with terminal constraint.

where I have used $u^\star_{x_t}(0)$ to make it explicit that the control is the first optimal control of solving (3.51) with an initial state x_t. The following algorithm summarizes the receding horizon control algorithm.

Algorithm: Online Receding Horizon Control

Input: State x_t at time t

Output: Control $u^\star_{x_t}(0)$

1. Solve problem (3.51) to get the optimal controls $u^\star(0), \dots, u^\star(N-1)$
2. **if** the problem is infeasible, **then** stop
3. **else return** $u^\star_{x_t}(0) = u^\star(0)$

RHC main questions. Two main questions arise regarding the RHC controller.

1. **Persistent feasibility**. If the RHC algorithm starts at a state x_0 for which the convex optimization (3.51) is feasible, i.e., $x_0 \in \mathcal{X}_0$, will the convex optimization (3.51) remain feasible for all future time steps?
2. **Stability**. Assuming the convex optimization is always feasible. Will the closed-loop system (3.55) (induced by the RHC controller) converge to the desired origin $x = 0$?

Let us use a couple of examples to illustrate that, in general, the answers to the above two questions are both NO.

Example 3.13 (Receding Horizon Control for Double Integrator)**.** Consider the discrete-time double integrator dynamics

$$x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_t,$$

subject to control constraint

$$u \in \mathcal{U} = [-0.5, 0.5],$$

and state constraint

$$x \in \mathcal{X} = [-5, 5] \times [-5, 5].$$

In the RHS subproblem (3.51), we use $N = 3$, $p(x) = x^T Px$, $q(x, u) = x^T Qx + u^T Ru$ with $P = Q = I$, $R = 10$, and $\mathcal{X}_f = \mathbb{R}^2$ (i.e., there is not terminal constraint). The subproblem is implemented using CVX in Matlab.

Fig. 3.24 shows the state trajectory of executing RHC starting at $x_0 = [-4.5; 2]$ and $x_0 = [-4.5; 3]$, respectively.

We can see that when the initial state is $x_0 = [-4.5; 2]$, the trajectory successfully converges to the origin (the blue line). However, when the initial state is $x_0 = [-4.5; 3]$, RHC fails in the third iteration because the subproblem becomes infeasible (the red line).

You can find code for this example here.

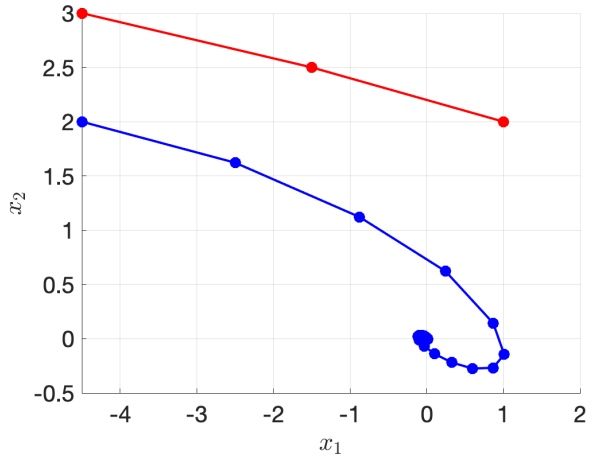


Figure 3.24: Receding horizon control for the double integrator with two initial states.

We now show another example, adapted from (Borrelli et al., 2017) where the design of N, $p(x)$ and $q(x, u)$ affects the closed-loop performance.

Example 3.14 (RHC Performance Affected by Parameters)**.** Consider the system

$$x_{t+1} = \begin{bmatrix} 2 & 1 \\ 0 & 0.5 \end{bmatrix} x_t + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_t,$$

with control constraint

$$u \in \mathcal{U} = [-1, 1],$$

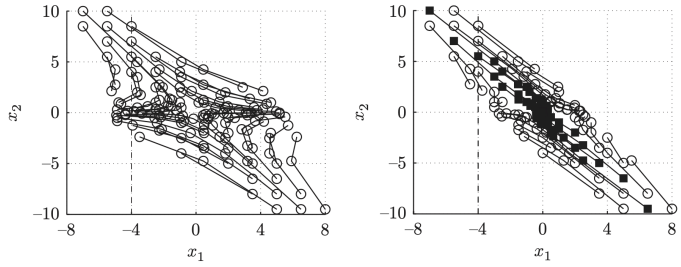
and state constraint

$$x \in \mathcal{X} = [-10, 10] \times [-10, 10].$$

In the RHC problem (3.51), we choose $p(x) = x^T Px$, $q(x, u) = x^T Qx + u^T Ru$. We fix $P = 0$, $Q = I$, and $\mathcal{X}_f = \mathbb{R}^2$, but vary N and R:

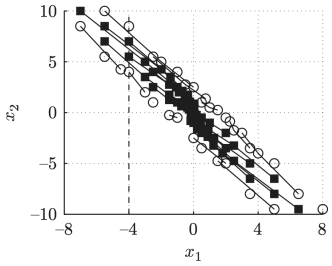
- Setting 1: $N = 2$, $R = 10$;
- Setting 2: $N = 3$, $R = 2$;
- Setting 3: $N = 4$, $R = 1$.

Fig. 3.25 shows the closed-loop trajectories of three different settings. As we can see, the closed-loop performance depends on the parameters in a very complicated manner.



(a) *Setting 1*: $N = 2, R = 10$.

(b) *Setting 2*: $N = 3, R = 2$.



(c) *Setting 3*: $N = 4, R = 1$.

Figure 3.25: Closed-loop trajectories for different settings of horizon N and weight R. Boxes (circles) are initial points leading to feasible (infeasible) closed-loop trajectories.

3.3.4 Persistent Feasibility

Under what conditions can we guarantee the RHC subproblem (3.51) is always feasible?

Intuitively, we will show that, by designing the terminal constraint set $\mathcal{X}_f$, we can guarantee the configuration of the feasible sets will look like choice 2 in Example 3.12.

There are various sets here of interest for answering this question.

- $\mathcal{C}_\infty$: The maximal control invariant set $\mathcal{C}_\infty$ is only affected by the system dynamics (3.49) and the constraint sets $\mathcal{X} \times \mathcal{U}$. It is the largest set over which we can expect *any* controller to work, because otherwise the system trajectory will blow up.
- $\mathcal{X}_0$: The set of states at which the RHC subproblem (3.51) is feasible. The set $\mathcal{X}_0$ depends on the system dynamics, the constraint sets $\mathcal{X} \times \mathcal{U}$, as well as the RHC horizon N and the terminal constraint set $\mathcal{X}_f$. It is worth noting that it does not depend on the objective function in (3.51) (i.e., P, Q, R).
- $\mathcal{O}_\infty$: The maximal positive invariant set for the closed-loop system (3.55) induced by the RHC control law. This depends on the RHC controller and hence it depends on the system dynamics, the constraint set $\mathcal{X} \times \mathcal{U}$, N, $\mathcal{X}_f$ and the objective function of (3.51) P, Q, R.

A few relationships between these sets are easy to observe.

- $\mathcal{O}_\infty \subseteq \mathcal{X}_0$. Any state $x \in \mathcal{O}_\infty$ needs to stay in $\mathcal{O}_\infty$ for all future time steps. Thus, the subproblem (3.51) must be feasible for any $x \in \mathcal{O}_\infty$, otherwise the closed-loop system (3.55) is not even well defined.
- $\mathcal{O}_\infty \subseteq \mathcal{C}_\infty$. The set $\mathcal{O}_\infty$ is control invariant because there exists a controller (specifically, the RHC controller) that makes it positive invariant. Therefore, $\mathcal{O}_\infty$ must belong to the maximal control invariant set $\mathcal{C}_\infty$.

We can now state necessary and sufficient conditions guaranteeing persistent feasibility.

Lemma 3.1 (Sufficient and Necessary Condition for Persistent Feasibility)**.** *The RHC subproblem* (3.51) *is persistently feasible if and only if* $\mathcal{X}_0 = \mathcal{O}_\infty$.

Proof. We have already argued that $\mathcal{O}_\infty \subseteq \mathcal{X}_0$. It remains to show $\mathcal{X}_0 \subseteq \mathcal{O}_\infty$. By definition, $\mathcal{X}_0$ is persistently feasible implies that

$$x \in \mathcal{X}_0 \Longrightarrow f_{\text{cl}}^t(x) \in \mathcal{X}_0, \forall t$$

where f_{cl}^t means applying the closed-loop dynamics f_{cl} in (3.55) t times. This shows that $\mathcal{X}_0$ is a positive invariant set for the closed-loop system, and hence $\mathcal{X}_0 \subseteq \mathcal{O}_\infty$. □

We argued that $\mathcal{X}_0$ does not depend on the RHC parameters P, Q, R but $\mathcal{O}_\infty$ does. Therefore, in general only some P, Q, R are allowed for persistent feasibility to hold. Due to the complicated relationship between P, Q, R and $\mathcal{O}_\infty$, it is generally difficult to design P, Q, R such that RHC has persistent feasibility.

We now state a sufficient condition for persistent feasibility to hold.

Lemma 3.2 (Sufficient Condition for Persistent Feasibility)**.** *Consider the RHC subproblem* (3.51) *with* $N \geq 1$*. If* $\mathcal{X}_1$ *is a control invariant set for the linear system* (3.49)*, then the RHC controller is persistently feasible.*

Proof. If $\mathcal{X}_1$ is control invariant, then by definition $\mathcal{X}_1 \subseteq \text{Pre}(\mathcal{X}_1)$. Since $\mathcal{X}_1 \subseteq \mathcal{X}$, we have

$$\mathcal{X}_1 \subseteq \text{Pre}(\mathcal{X}_1) \cap \mathcal{X} = \mathcal{X}_0.$$

where the last equality in the equation above is due to (3.54). Now pick any $x_0 \in \mathcal{X}_0$, the set of controls that make x_0 feasible is denoted as

$$U = \{u(0), \dots, u(N-1) \mid x(0) = x_0, x(k) \in \mathcal{X}, k = 0, \dots, N-1, x(N) \in \mathcal{X}_f\}.$$

The RHC will pick some control sequence from U, say $\hat{u}(0), \dots, \hat{u}(N-1)$ and apply the first control $\hat{u}(0)$ to the system, which will bring the system to a new state

$$x_1 = Ax_0 + B\hat{u}(0).$$

Observe that $x_1 \in \mathcal{X}_1$ by definition. Since $\mathcal{X}_1 \subseteq \mathcal{X}_0$, we have $x_1 \in \mathcal{X}_0$. This proves that the RHC is persistently feasible, i.e., starting with any $x_0 \in \mathcal{X}_0$, the RHC subproblem (3.51) is always feasible. □

Lemma 3.2 states that if $\mathcal{X}_1$ is control invariant, then the RHC subproblem is persistently feasible. An immediate result of this Lemma is that when $N = 1$, then $\mathcal{X}_1 = \mathcal{X}_f$. Therefore, if we choose the terminal constraint set $\mathcal{X}_f$ to be control invariant, then RHC is has persistent feasibility.

The next theorem states that when $N \geq 1$, as long as we choose the terminal constraint set $\mathcal{X}_f$ to be control invariant, then persistent feasibility also holds.

Theorem 3.2 (Control Invariant Terminal Constraint Set Guarantees Persistent Feasibility)**.** *Consider the RHC subproblem* (3.51) *with* $N \geq 1$*. If* $\mathcal{X}_f$ *is a control invariant set for the linear system* (3.49)*, then the RHC controller is persistently feasible.*

Proof. We will prove that $\mathcal{X}_f$ being control invariant implies

$$\mathcal{X}_{N-1}, \mathcal{X}_{N-2}, \dots, \mathcal{X}_1$$

are all control invariant, and then by Lemma 3.2, we can guarantee persistent feasibility. Fig. 3.26 shows the nested control invariant sets when $\mathcal{X}_f$ is control invariant.

It suffices to show $\mathcal{X}_{i+1}$ being control invariant leads to $\mathcal{X}_i$ being control invariant. First, by $\mathcal{X}_{i+1}$ control invariant, we have $\mathcal{X}_{i+1} \subseteq \text{Pre}(\mathcal{X}_i) \cap \mathcal{X} = \mathcal{X}_i$. Now pick any $x_i \in \mathcal{X}_i$, for any feasible $\hat{u}(0), \dots, \hat{u}(N-1)$, applying the first control $\hat{u}(0)$ brings the system to a new state

$$x_{i+1} = Ax_i + B\hat{u}(0) \in \mathcal{X}_{i+1} \subseteq \mathcal{X}_i.$$

This shows $\mathcal{X}_i$ is control invariant. □

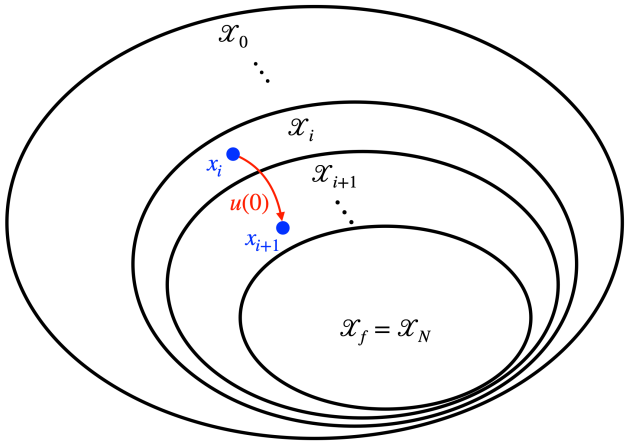


Figure 3.26: Nested control invariant sets.

Persistent feasibility does not guarantee the closed-loop system will converge to the origin. In fact, from Theorem 3.2, it is clear that if the system starts at $x_0 \in \mathcal{X}_0$, then we can only guarantee $x_t \in \mathcal{X}_1$ for all $t \geq 0$.

3.3.5 Stability

We now answer the question of how can we guarantee the RHC controller will drive the system to the desired origin.

Theorem 3.3 (Sufficient Condition for Stability)**.** *Consider the linear system* (3.49)*, and the RHC algorithm* (3.51)*. Assume that*

1. *The stage cost $q(x,u)$ and terminal cost $p(x)$ are continuous and positive definite functions.*
2. *The sets $\mathcal{X}$, $\mathcal{X}_f$ and $\mathcal{U}$ contain the origin in their interior and are closed.*
3. *$\mathcal{X}_f \subseteq \mathcal{X}$ is control invariant.*
4. *For any $x \in \mathcal{X}_f$, the following inequality holds*

$$\min_{u \in \mathcal{U}, Ax+Bu \in \mathcal{X}_f} (-p(x) + q(x,u) + p(Ax+Bu)) \leq 0. \tag{3.56}$$

Then, the origin of the closed-loop system (3.55) *is asymptotically stable with domain of attraction $\mathcal{X}_0$. In words, for any $x_0 \in \mathcal{X}_0$, if the closed-loop system* (3.55) *starts at x_0, then the system trajectory converges to the origin as t tends to infinity.*

Let us interpret Theorem 3.3 before proving it. It should be clear that the first three assumptions are easy to satisfy. For example, if we choose $p(x) = x^T P x$ and $q(x,u) = x^T Q x + u^T R u$ with $P, Q, R \succ 0$, then assumption 1 is satisfied. Usually $\mathcal{X}$ and $\mathcal{U}$ are both polyhedral sets containing the origin in the interior, so assumption 2 also holds naturally. Finding a control invariant set $\mathcal{X}_f$ is not a trivial task but there exists numerical algorithms for this task (e.g., using the algorithms introduced in Section 3.3.2.4). After we find a control invariant $\mathcal{X}_f$, by Theorem 3.2, we know persistent feasibility will hold.

We now prove the theorem by showing that Assumption 4 guarantees stability. Before we show the proof, we need the concept of a *Lyapunov function*. Below we introduce Lyapunov function for a discrete-time dynamical system, but we will study more details of Lyapunov function for continuous-time dynamical systems in Chapter 4.

Lemma 3.3 (Discrete-time Lyapunov Function)**.** *Consider the discrete-time autonomous system* (3.43)*, restated below for convenience:*

$$x_{t+1} = f_a(x_t),$$

and assume $x = 0$ is an equilibrium point of the system, i.e., $0 = f_a(0)$ (if the system starts at the origin, it stays at the origin). Let $\Omega \subset \mathbb{R}^n$ be a closed and bounded set containing the origin. If there exists a function $V : \mathbb{R}^n \to \mathbb{R}$ that is (a) continuous at the origin, (b) finite for every $x \in \Omega$, and (c) satisfies

1. *V is positive definite: $V(0) = 0$ and $V(x) > 0, \forall x \in \Omega \backslash \{0\}$,*
2. *$V(x_{t+1}) - V(x_t) \leq -\alpha(x_t) < 0$ for any $x_t \in \Omega \backslash \{0\}$,*

where $\alpha : \mathbb{R}^n \to \mathbb{R}$ is a continuous positive definite function. Then $x = 0$ is asymptotically stable in Ω, i.e, if the system starts within Ω, then its trajectory tends to 0 as $t \to \infty$.

A function V satisfying the conditions above is called a Lyapunov function for the system.

One can think of the Lyapunov function V as an energy function for the system (3.43) that maps a system state to a single scalar. Condition 1 in Lemma 3.3 states that V (the energy function) is strictly positive except at the origin. Condition 2 in Lemma 3.3 states that, the system energy V is strictly decreasing along any system trajectory. Therefore, we conclude that V must converge to $V = 0$, and hence any state trajectory must converge to the origin.

We will now use Lemma 3.3 to prove RHC stability under Theorem 3.3.

Proof. Our goal is to prove that the optimal cost of the RHC problem (3.51), $J_t^\star(x_t)$, is a Lyapunov function for the closed-loop system (3.55) on the domain

$\mathcal{X}_0$. Then using Lemma 3.3 we can conclude that the closed-loop system (3.55) will converge to the origin.

Positive definite. Clearly, we have $J_t^\star(0) = 0$ by the positive definiteness of p and q and the fact that $x = 0$ is an equilibrium point of the linear system (3.49). For $x_t \neq 0$, it is also clear that $J_t^\star(x) > 0$. Therefore, $J_t^\star(x)$ is positive definite on $\mathcal{X}_0$.

Strictly decrease. It suffices to show $J_1^\star(x_1) - J_0^\star(x_0) < 0$ because the constraints of the RHC problem (3.51) is time-invariant. Pick any $x_0 \in \mathcal{X}_0$, and let

$$(u^\star(0), u^\star(1), \dots, u^\star(N-1)) \tag{3.57}$$

be an optimal control trajectory to problem (3.51). Denote

$$(x_0, x(1), \dots, x(N)) \tag{3.58}$$

as the associated optimal state trajectory for problem (3.51). The RHC controller will apply $u^\star(0)$ to the system, leading to the next state

$$x_1 = x(1) = Ax_0 + Bu^\star(0).$$

Then we will solve problem (3.51) to get $J_1^\star(x_1)$, and we want to show $J_1^\star(x_1) < J_0^\star(x_0)$. Towards this, we will construct a feasible solution to (3.51) starting at x_1, and hence an upper bound on $J_1^\star(x_1)$. Consider the control sequence

$$(u^\star(1), \dots, u^\star(N-1), v) \tag{3.59}$$

which is different from the control sequence in (3.57) by removing $u^\star(0)$ and appending v. The corresponding state trajectory to (3.59) is

$$(x_1 = x(1), x(2), \dots, x(N), Ax(N) + Bv) \tag{3.60}$$

which is different from the state trajectory in (3.58) by removing x_0 and appending $Ax(N)+Bv$. Since $x(N) \in \mathcal{X}_f$ and $\mathcal{X}_f$ is control invariant, there exists v such that $Ax(N) + Bv \in \mathcal{X}_f$ and the corresponding control trajectory (3.59) is feasible. Applying the control trajectory (3.59) to the optimization problem (3.51) will lead to the total cost

$$J_1(x_1) = J_0^\star(x_0) - q(x_0, u^\star(0)) \underbrace{-p(x(N)) + q(x(N), v) + p(Ax(N) + Bv)}_{s(x(N),v)}.$$

Now by assumption 4 in Theorem 3.3, we can choose v such that the sum $s(x(N), v) \leq 0$. Consequently, we have

$$J_1^\star(x_1) \leq J_1(x_1) \leq J_0^\star(x_0) - q(x_0, u^\star(0)),$$

which leads to

$$J_1^\star(x_1) - J_0^\star(x_0) \leq -q(x_0, u^\star(0)).$$

Because the choice of x_0 was arbitrary, we conclude that $J_t^\star(x_t)$ strictly decreases along any system trajectory that starts within $\mathcal{X}_0$.

Continuity at the origin. We will show that $J_0^\star(x) \leq p(x)$ for any $x \in \mathcal{X}_f$. With this argument, since $p(x)$ is positive definite and continuous, then $J_0^\star(x)$ must be continuous at the origin. We now prove $J_0^\star(x) \leq p(x)$ for any $x \in \mathcal{X}_f$. Since $\mathcal{X}_f$ is control invariant, pick any $x \in \mathcal{X}_f$, there exists a sequence of controls $(u(0), u(1), \dots, u(N-1))$ such that the state trajectory $(x(0) = x, x(1), \dots, x(N))$ stays in $\mathcal{X}_f$. Such a control sequence lead to an upper bound on $J_0^\star(x)$:

$$J_0^\star(x(0)) \leq p(x(N)) + \sum_{i=0}^{N-1} q(x(i), u(i)) = p(x(0)) + \sum_{i=0}^{N-1} \left(q(x(i), u(i)) + p(x(i+1)) - p(x(i)) \right),$$

since each $x(i) \in \mathcal{X}_f$, according to Assumption 4, we can choose $u(i)$ such that

$$J_0^\star(x(0)) \leq p(x(0))$$

for any $x(0) \in \mathcal{X}_f$.

In conclusion, we have shown that $J_t^\star(x_t)$ is a Lyapunov function, and by Lemma 3.3, the closed-loop system is asymptotically stable. □

A function $p(x)$ that satisfies Assumption 4 in Theorem 3.3 is typicalled known as a *control Lyapunov function*.

Now two natural problems arise:

1. Given $p(x)$, how to verify if assumption 3 holds in Theorem 3.3?
2. How to synthesize a $p(x)$ that satisfies assumption 3 in Theorem 3.3?

Unfortunately, both problems are hard. To see this, suppose we are given a candidate function $p(x) = x^T P x$ with $P \succ 0$ that is clearly positive definite. Assume $q(x, u) = x^T Q x + u^T R u$ with $Q, R \succ 0$ and $\mathcal{U}, \mathcal{X}_f$ are given polyhedral sets. Then verifying if $p(x)$ satisifies Assumption 4 boils down to checking if

$$\min_{u \in \mathcal{U}, Ax+Bu \in \mathcal{X}_f} x^T Q x + u^T R u + (Ax + Bu)^T P (Ax + Bu) - x^T P x$$

is non-positive for any possible $x \in \mathcal{X}_f$. Although for each possible x, the above problem is a convex optimization problem, there are an infinite number of points in the set $\mathcal{X}_f$ and enumerating over all points is a daunting task. We will see in Chapter 4 that convex relaxations, in particular semidefinite relaxations, can help us partially solve these hard problems.

A simple control lyapunov function. One can solve the infinite-horizon unconstrained LQR problem

$$\min_{u_t} \sum_{t=0}^{\infty} x_t^T Q x_t + u_t^T R u_t,$$

for which the optimal cost-to-go is

$$J_\infty(x) = x^T S x,$$

with S the solution to the algebraic Riccati equation (2.14). Denote

$$u_t = \Pi_{\mathcal{U}}(-K x_t)$$

as the optimal controller ($\Pi_{\mathcal{U}}$ is the projection of the controller to the feasible set $\mathcal{U}$), and use $\mathcal{X}_f$ as the maximal positive invariant set of the closed-loop system

$$x_{t+1} = A x_t + B u_t.$$

Then $J_\infty(x) = x^T S x$ is a control Lyapunov function over the set $\mathcal{X}_f$.

3.3.6 Explicit MPC

See the original paper (Bemporad et al., 2002), and check out Matlab's explict MPC design.

3.4 Policy Gradient

Chapter 4

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

$$\min_{u(t)\in\mathbb{U}} \int_0^\infty \|x(t) - x_d\|^2 dt, \quad \text{subject to} \quad \dot{x} = f(x,u), x(0) = x_0, \tag{4.1}$$

where x_d is the desired upright position for the pendulum. However, does the solution of problem (4.1), 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

$$\min_{u(t)\in\mathbb{U}} \int_0^T \|u(t)\|^2 dt, \quad \text{subject to} \quad x(0) = x_0, x(T) \in \Omega, \dot{x} = f(x,u), \tag{4.2}$$

where now it is very clear, if a solution exists to problem (4.2), 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 (4.2) 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 (4.2) 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).

4.1 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.

4.1.1 Concepts of Stability

Consider the autonomous system

$$\dot{x} = f(x) \tag{4.3}$$

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 4.1 (Equilibrium Point)**.** A state $x^\star$ is called an equilibrium point of system (4.3) 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 (4.3) near the equilibrium point $x^\star$, we can create a new variable

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

so that

$$\dot{z} = \dot{x} = f(x) = f(z + x^\star). \tag{4.4}$$

Clearly, $z^\star = 0$ is an equilibrium point for the shifted system (4.4).

Let us find the equilibrium points of a simple pendulum.

Example 4.1 (Equilibrium Points of A Simple Pendulum)**.** Consider the dynamics of an uncontrolled pendulum

$$\begin{cases} \dot{\theta} = \dot{\theta} \\ \ddot{\theta} = -\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta) \end{cases} \tag{4.5}$$

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 (4.5) 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 (4.3) with $f(0) = 0$, i.e., $x^\star = 0$ is an equilibrium point. We now formally define the different concepts of stability.

Definition 4.2 (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 4.1 (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 4.3 (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 4.2 (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.)

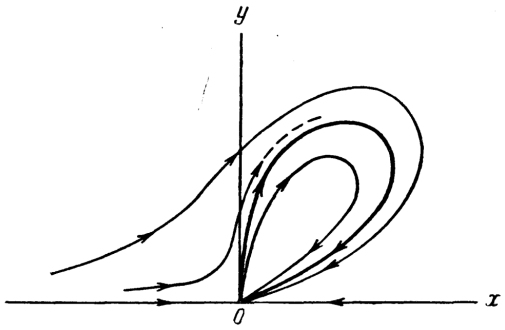


Figure 4.1: Trajectories of the Vinograd system. Copied from the original article of Vinograd.

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 4.4 (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 **??**, 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 4.5 (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 4.2-4.5). 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 (4.3), 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.

4.1.2 Stability by Linearization

A natural idea is to linearize, if possible, the nonlinear system (4.3) 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 4.1 (Stability by Linearization)**.** *Assume $x = 0$ is an equilibrium point of system* (4.3) *and f is continuously differentiable. Let*

$$\dot{x} = Ax, \quad A = \frac{\partial f}{\partial x}\Big|_{x=0} \tag{4.6}$$

be the linearized system at $x = 0$. The following statements are true about the stability relationship between (4.3) *and* (4.6)*.*

- *If the linearized system* (4.6) *is strictly stable (i.e., all eigenvalues of A have strictly negative real parts), then the original system* (4.3) *is asymptotically stable at $x = 0$.*
- *If the linearized system* (4.6) *is unstable (i.e., at least one eigenvalue of A has strictly positive real part), then the original system* (4.3) *is unstable at $x = 0$.*
- *If the linearized system* (4.6) *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* (4.3) *at $x = 0$ is indeterminate.*

Theorem 4.1 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 4.2 (Stability of A Simple Pendulum by Linearization)**.** Consider the simple pendulum dynamics (4.5) in Example 4.1. 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.

4.1.3 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 4.6 (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 4.7 (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 (4.3). 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 4.2 (Lyapunov Local Stability)**.** *Consider the nonlinear system* (4.3) *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 4.2).

Moreover,

- *if $\dot{V}(x)$ is negative definite in B_R, then the equilibrium point is asymptotically stable (cf. Definition 4.3).*
- *if $\dot{V}(x) \leq -\alpha V(x)$ for any $x \in B_R$, then the equilibrium point is exponentially stable (cf. Definition 4.4).*

Let us apply Theorem 4.2 to the simple pendulum.

Example 4.3 (Lyapunov Local Stability for A Simple Pendulum)**.** Consider the pendulum dynamics (4.5). The total energy of a pendulum is

$$V(x) = \frac{1}{2}ml^2\dot{\theta}^2 + mgl(1-\cos\theta). \tag{4.7}$$

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 4.2 we conclude that the equilibrium point is stable in the sense of Lyapunov.

Note that with this choice of $V(x)$ as in (4.7), 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

$$\tilde{V}(x) = \frac{1}{2}ml^2\dot{\theta}^2 + \frac{1}{2}ml^2\left(\frac{b}{ml^2}\theta + \dot{\theta}\right)^2 + 2mgl(1-\cos\theta), \tag{4.8}$$

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

$$\tilde{V}(x) = ml^2\dot{\theta}^2 + 2mgl(1-\cos\theta) + \frac{1}{2}ml^2\left(\frac{b^2}{m^2l^4}\theta^2 + \frac{2b}{ml^2}\theta\dot{\theta}\right) \tag{4.9}$$

$$= 2V(x) + \frac{1}{2}ml^2\left(\frac{b^2}{m^2l^4}\theta^2 + \frac{2b}{ml^2}\theta\dot{\theta}\right). \tag{4.10}$$

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

$$\dot{\tilde{V}}(x) = 2\dot{V}(x) + \frac{ml^2}{2}\left(\frac{2b^2}{m^2l^4}\theta\dot{\theta} + \frac{2b}{ml^2}(\dot{\theta}^2 + \theta\ddot{\theta})\right) \tag{4.11}$$

$$= 2\dot{V}(x) + b\dot{\theta}^2 + \left(\frac{b^2}{ml^2}\theta\dot{\theta} + b\theta\left(-\frac{1}{ml^2}(b\dot{\theta} + mgl\sin\theta)\right)\right) \tag{4.12}$$

$$= -b\left(\dot{\theta}^2 + \frac{g}{l}\theta\sin\theta\right). \tag{4.13}$$

$\dot{\tilde{V}}(x)$ 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 4.2). $\tilde{V}(x)$ does not have any physical intuition, but it successfully certifies local asymptotic stability.

In Section 4.1.4, 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 4.3 (Lyapunov Global Stability)**.** *For the autonomous system* (4.3)*, 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 4.5).

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.*

4.1.4 Invariant Set Theorem

Through Theorem 4.2, Theorem 4.3, and Example 4.3, 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 4.3 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 4.8 (Invariant Set)**.** A set G is an invariant set for a dynamical system (4.3) 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 4.3).

We now state the local invariant set theorem.

Theorem 4.4 (Local Invariant Set)**.** *Consider the autonomous system* (4.3)*, 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 4.3.

Example 4.4 (Revisiting the Local Stability of A Simple Pendulum)**.** In Example 4.3, 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 4.4 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 $\rho = \frac{1}{4}mgl$,

$$V(x) < \frac{1}{4}mgl \Rightarrow \frac{1}{2}ml^2\dot{\theta}^2 < \frac{1}{4}mgl \Rightarrow \dot{\theta}^2 < \frac{1}{2}\frac{g}{l} \tag{4.14}$$

$$V(x) < \frac{1}{4}mgl \Rightarrow mgl(1 - \cos\theta) < \frac{1}{4}mgl \Rightarrow \cos\theta > \frac{3}{4} \Rightarrow \theta \in (-\arccos\frac{3}{4}, \arccos\frac{3}{4}). \tag{4.15}$$

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 4.4, 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 4.5 (Global Invariant Set)**.** *For the autonomous system* (4.3)*, 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* $t \to \infty$*.*

4.1.5 Computing Lyapunov Certificates

All the Theorems we have stated so far (Theorems 4.2, 4.3, 4.4, and 4.5) 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 4.3, we have seen that physical intuition can help us find a good Lyapunov function (4.7). Nevertheless, it did not quite give us what we want in terms of asymptotic stability. Instead, a hand-crafted function (4.8) 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 4.2, 4.3, 4.4, and 4.5 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 (4.3) 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 4.5 (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

$$p(x) = -x^4 + 2x^2 + 2x + 1 = (x+1)^2 + x^2(1 - x^2), \tag{4.16}$$

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 (4.16)), 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 (4.16)), 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

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid p_i(x) = 0, i = 1, \dots, l_{\text{eq}}; p_i(x) \geq 0, i = l_{\text{eq}} + 1, \dots, l_{\text{eq}} + l_{\text{ineq}}\} \tag{4.17}$$

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 4.5. 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 4.5, you will say if someone can produce a decomposition of $p_0(x)$ as

$$p_0(x) = \sigma_0(x) + \sum_{i=1}^{l_{\text{ineq}}} \sigma_i(x) p_{i+l_{\text{eq}}}(x) + \sum_{i=1}^{l_{\text{eq}}} \lambda_i(x) p_i(x), \tag{4.18}$$

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 (4.17),
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 (4.17).

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 4.5, it is natural to come up with the choice of a "sum-of-squares" (SOS) polynomial.

Definition 4.9 (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 4.1 (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 (4.18):

$$\begin{aligned} \text{find} \quad & \{\sigma_i\}_{i=0}^{l_{\text{ineq}}}, \{\lambda_i\}_{i=1}^{l_{\text{eq}}} \\ \text{subject to} \quad & p_0(x) = \sigma_0(x) + \sum_{i=1}^{l_{\text{ineq}}} \sigma_i(x) p_{i+l_{\text{eq}}}(x) + \sum_{i=1}^{l_{\text{eq}}} \lambda_i(x) p_i(x), \\ & \sigma_i \text{ is SOS}, i = 0, \dots, l_{\text{ineq}}, \\ & \lambda_i \text{ is polynomial}, i = 1, \dots, l_{\text{eq}} \\ & \deg(\sigma_0) \le 2\kappa, \deg(\sigma_i p_{i+l_{\text{eq}}}) \le 2\kappa, i = 1, \dots, l_{\text{ineq}}, \\ & \deg(\lambda_i p_i) \le 2\kappa, i = 1, \dots, l_{\text{eq}}. \end{aligned} \tag{4.19}$$

Bounding the Degree. The careful reader realizes that in (4.19) 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 (4.19) 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 (4.19) 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 4.1.

[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 (4.19) 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 (4.19) 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 (4.19), 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 (4.19) 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 (4.19)? 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 (4.19) 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

$$\begin{aligned} \max \quad & \gamma \\ \text{subject to} \quad & p_0(x) - \gamma \geq 0, \quad \forall x \in \mathcal{X}. \end{aligned} \tag{4.20}$$

Basically I want to push the lower bound γ as high as possible. The constraint in (4.20) asks $p_0(x)-\gamma$ to be nonnegative on $\mathcal{X}$. With the SOS framework introduced above, we can naturally relax it to

$$\begin{aligned} \max \quad & \gamma \\ \text{subject to} \quad & p_0(x) - \gamma \quad \text{is SOS on} \quad \mathcal{X}, \end{aligned} \tag{4.21}$$

where the "SOS on $\mathcal{X}$" constraint is exactly the problem (4.19). Therefore, we have relaxed the nonconvex optimization (4.20) into a convex problem (4.21)! 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 (4.20). 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 (4.21) 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 (4.21) at a finite relaxation order κ, an exact global optimizer of the original nonconvex problem (4.20) 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 (4.19) and (4.21), 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 (4.18). 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 4.2, 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

$$\begin{aligned} \text{find} \quad & V(x) && (4.22)\\ \text{subject to} \quad & V(x) - \epsilon_1 \|x - x^\star\|^2 \quad \text{is SOS on} \quad \mathcal{X} && (4.23)\\ & -\epsilon_2 \|x - x^\star\|^2 - \frac{\partial V(x)}{\partial x} f(x) \quad \text{is SOS on} \quad \mathcal{X} && (4.24)\\ & V(x^\star) = 0, && (4.25) \end{aligned}$$

where $\epsilon_1, \epsilon_2 > 0$ are (small) positive constants. This is a convex optimization problem, just like (4.19) (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 4.6 (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

$$\theta \in \left[-\arccos\frac{3}{4}, \arccos\frac{3}{4}\right], \quad \dot{\theta} \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]. \tag{4.26}$$

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

$$\begin{aligned} V(x) &\ge \epsilon_1 (x - x_e)^T (x - x_e), \quad \forall x \in \mathcal{X} && (4.27)\\ \dot{V}(x) = \frac{\partial V}{\partial x}\dot{x} &\le -\epsilon_2 (x - x_e)^T (x - x_e), \quad \forall x \in \mathcal{X} && (4.28)\\ V(x_e) &= 0, \quad \dot{V}(x_e) = 0 && (4.29) \end{aligned}$$

where (4.27) ensures $V(x)$ is positive definite, (4.28) ensures $\dot{V}(x)$ is negative definite, and (4.29) 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

$$V(x) - \epsilon_1 (x - x_e)^T (x - x_e) \quad \text{is SOS on} \quad \mathcal{X} \tag{4.30}$$

$$-\epsilon_2 (x - x_e)^T (x - x_e) - \frac{\partial V}{\partial x} \dot{x} \quad \text{is SOS on} \quad \mathcal{X} \tag{4.31}$$

$$V(x_e) = 0, \quad \dot{V}(x_e) = 0. \tag{4.32}$$

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 (4.26) using $(\theta, \dot{\theta})$ coordinates, we get

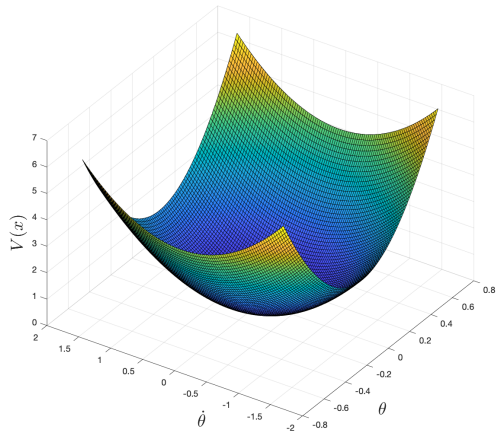


Figure 4.2: Lyapunov local stability certificate computed via convex optimization.

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

Plotting $\dot{V}(x)$ in the constraint set (4.26) 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.

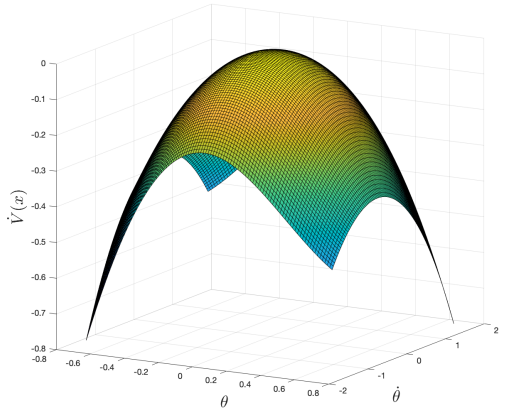


Figure 4.3: Derivative of the Lyapunov local stability certificate computed via convex optimization.

4.2 Controlled Systems

We can generalize the notion of a Lyapunov function for an autonomous system to the notion of a *Control Lyapunov Function* (CLF) for a controlled system (Sontag, 1983).

Consider a controlled system

$$\dot{x} = f(x, u) \tag{4.33}$$

where $x \in \mathbb{X} \subseteq \mathbb{R}^n$ the state, $u \in \mathbb{U} \subseteq \mathbb{R}^m$ the control, and f is continuously differentiable. Let $x = 0, u = 0$ be an equilibrium point, i.e., $f(0, 0) = 0$. The existence of a CLF guarantees that we can design a controller to stabilize the equilibrium point.

Theorem 4.6 (Control Lyapunov Function)**.** *Let $V(x) : \mathbb{X} \to \mathbb{R}$ be a continuously differentiable positive definite function, i.e., $V(0) = 0$ and $V(x) > 0, \forall x \in \mathbb{X}\backslash\{0\}$. If there exists $\rho > 0$ such that $\Omega := \{x \in \mathbb{X} \mid V(x) \leq \rho\}$ is bounded and*

$$\min_{u \in \mathbb{U}} \dot{V}(x) = \frac{\partial V}{\partial x} f(x, u) < 0, \quad \forall x \in \Omega\backslash\{0\}, \tag{4.34}$$

then $V(x)$ is called a control Lyapunov function.

In this case, if the system starts within the set Ω, then there exists a controller $u(t)$ that drives the system towards the equilibrium point 0.

In Theorem 4.6, it is clear that *Omega* is a control invariant set. In fact, *Omega* is an inner approximation of the *region of attraction* to the equilibrium point 0.

Deploying a CLF. Now we observe that, when a CLF $V(x)$ is given, it can help us design an asymptotically stablizing controller. Consider a special instance of the controlled system (4.33) that is *control-affine*

$$\dot{x} = f_1(x) + f_2(x)u. \tag{4.35}$$

Clearly, the dynamics (4.35) is affine in the control u. Almost all robotics systems are control affine, so (4.35) is quite general. Suppose we are given an arbitrary controller $u_d(t)$, which may or may not stabilize the system towards the equilibrium point (e.g., $u_d(t)$ may be obtained from neural networks). We can use the CLF $V(x)$ to correct $u_d(t)$ so that we always obtain a stabilizing feedback controller

$$\begin{aligned} u(x(t)) = \arg\min_{u\in\mathbb{U}} \quad & \|u - u_d(t)\|^2 \\ \text{subject to} \quad & \dot{V}(x) = \frac{\partial V}{\partial x}(f_1(x) + f_2(x)u) < 0. \end{aligned} \tag{4.36}$$

Note that the optimization (4.36) is actually a convex optimization problem! Since we are minimizing over u (the state $x(t)$ is given), the objective in (4.36) is quadratic in u and the constraint in (4.36) is linear in u, implying (4.36) is a quadratic optimization problem. The CLF condition (4.34) ensures that (4.36) is always feasible. Intuitively, optimization (4.36) seeks to find the feedback controller that (a) minimally modifies the given arbitrary controller $u_d(t)$, and (b) guarantees stabilization towards the equilibrium.

Verification and Synthesis of a CLF. When a CLF is given, deploying it to compute a stablizing controller is easy. However, verifying if a candidate function is indeed a control Lyapunov function is a highly nontrivial task. In fact, it is an active area of research. The interested reader can refer to (Kang et al., 2023) and (Dai and Permenter, 2023).

4.3 Non-autonomous Systems

Lemma 4.2 (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 4.7 (Barbalat's Stability Certificate)**.** *If a scalar function $V(x,t)$ satisfies*

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

- $V(x,t)$ *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* $t \to \infty$.

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 4.2. □

Chapter 5

Problem Sets

Exercise 5.1 (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. 5.1 with $N = 3, 4, 5$.

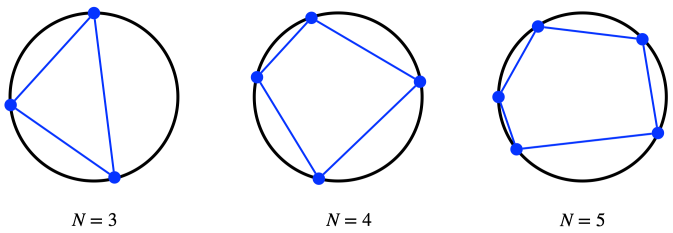


Figure 5.1: Polygons inscribed inside a circle

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.

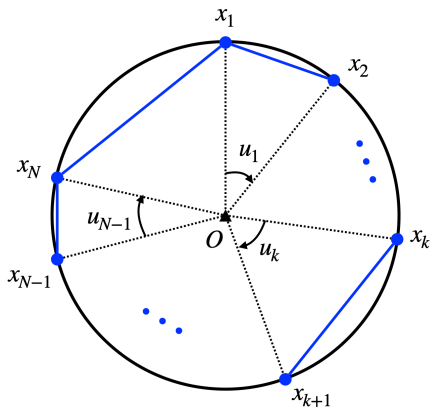


Figure 5.2: Sequential placement of N points on the circle.

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. 5.2, 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. 5.2. 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. 5.2), 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

$$J_{N-1}(x_{N-1}) = \max_{0<u_{N-1}<2\pi-x_{N-1}} \left\{ \underbrace{2\sin\left(\frac{u_{N-1}}{2}\right) + J_N(x_{N-1}+u_{N-1})}_{Q_{N-1}(x_{N-1},u_{N-1})} \right\}, \quad (5.1)$$

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 (5.1) 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. 5.2, 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

$$\begin{aligned} \max_{u_1,\ldots,u_N} \quad & \sum_{k=1}^{N} 2\sin\left(\frac{u_k}{2}\right) \\ \text{subject to} \quad & u_k > 0, k = 1, \ldots, N \\ & u_1 + \cdots + u_N = 2\pi \end{aligned} \tag{5.2}$$

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 (5.2) 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 (5.2) 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 (5.2) so we can numerically prove our claim.

g. I have provided most of the code necessary for solving (5.2) 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 5.2 (LQR with Constraints)**.** In class we worked on the LQR problem where the states and controls are unbounded. This is rarely the case in real life – you only have a limited amount of control power, and you want your states to be bounded (e.g., not entering some dangerous zones).

For linear systems with convex constraints on the control and states, the seminal

paper (Bemporad et al., 2002) investigates the landscape of the optimal cost-to-go and controller.

In this exercise, let us use convex optimization to numerically study a toy problem.

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:

$$\begin{aligned} J(x_0) = \min_{u_0,\dots,u_{N-1}\in[-u_{\max},u_{\max}]} \quad & x_N^T Q_N x_N + \sum_{k=0}^{N-1} (x_k^T Q_k x_k + u_k^T R_k u_k) \\ \text{subject to} \quad & x_{k+1} = Ax_k + Bu_k, k = 0, \dots, N-1 \end{aligned} \tag{5.3}$$

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 (5.3), 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 (5.3) at a dense grid of x_0 (e.g., using CVX or cvxpy). Plot the optimal cost-to-go $J(x_0)$, and the optimal controls $u_0(x_0), \dots, u_{N-1}(x_0)$. For the optimal controls, you can just plot one of the controls such as $u_0(x_0)$. You may want to use the Matlab function `surf`. (Hint: you will most likely benefit from Appendix B.2.)

d. Increase N and decrease dt, repeat (c). Do you get more fine-grained plots of the optimal cost-to-go and controls? (When you increase N, the convex optimization has more variables to optimize, so there is a limit at which the solver takes too much time.)

e. **(Bonus)** We only have constraints on the control so far. What if you add constraints to the states as well? For example, you can try limiting the velocity $\dot{q}$ to be at least 0.1 by adding $\dot{q}_k \geq 0.1$ for some k. How will J and u change?

f. **(Bonus)** Can you write down the KKT optimality conditions of (5.3) and explain what you have observed from the numerical experiments? (Hint: KKT optimality conditions can be found in Theorem B.2.)

Exercise 5.3 (Cart Pole System)**.** In this exercse, let us study the cart-pole system (we saw the video of human-controlled version in our first lecture), another interesting nonlinear control problem, and reinforce our knowledge about LQR.

Our task is to balance a pendulum on a cart by horizontally moving the cart. Fig. 5.3 gives an illustration of the system. See this video for an actual robotic implementation.

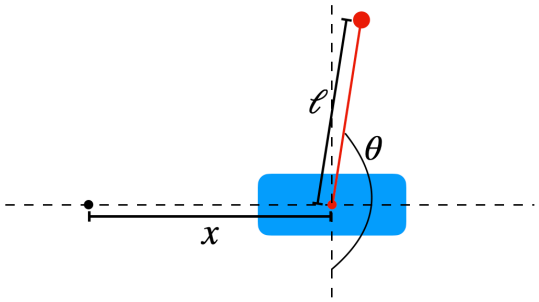


Figure 5.3: Illustration of cart-pole problem

With the above illustration, we parameterize the system with two scalars: 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. **(Bonus)** For those of you who have background in rigid-body dynamics, this is an opportunity for you to apply your knowledge. However, feel free to skip this subproblem and it won't affect the rest of the exercise. Denote the mass of cart and pole as m_c and m_p, respectively. Derive the equations of motion:

$$\left(m_c + m_p\right)\ddot{x} + m_p l\ddot{\theta}\cos\theta - m_p l\dot{\theta}^2\sin\theta = f, \tag{5.4}$$

$$m_p l\ddot{x}\cos\theta + m_p l^2\ddot{\theta} + m_p gl\sin\theta = 0. \tag{5.5}$$

(Hints: compute the Lagrangian of the system and the corresponding Lagrangian equations. Analyzing the two objects separately also works.)

b. Translate the equations in (a) into the basic state-space dynamics form

$$\dot{\mathbf{x}} = F(\mathbf{x}, \mathbf{u}). \tag{5.6}$$

What are $\mathbf{x}, \mathbf{u}, F$ here? (Hint: try $\mathbf{x} = [x, \theta, \dot{x}, \dot{\theta}]^\top$.)

c. Linearize the dynamics in (b) around the unstable equilibrium where $\theta^* = \pi$ and $x^* = \dot{x}^* = \dot{\theta}^* = 0$. (i.e., the pole is in the upright position and the cart stay at zero.) The result should be in the form of

$$\dot{\Delta\mathbf{x}} = A\Delta\mathbf{x} + B\Delta\mathbf{u}, \tag{5.7}$$

where $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}^*$ and $\Delta\mathbf{u} = \mathbf{u} - \mathbf{u}^*$.

d. 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 (5.6) and the linearized system (5.7) 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?)

e. Convert the continuous-time dynamics in (5.7) to discrete-time with a fixed time-discretization. Then design an LQR controller to stabilize the cart-pole at the unstable equilibrium. Does the LQR controller succeed for all initial conditions you tested? (Hint: try several initial conditions where the end point of the pole is above or below the horizontal line.) You may want to take a look at the LQR example for the simple pendulum in Example 2.1.

Exercise 5.4 (Trajectory Optimization)**.** Let us use this exercise to practice your skills in implementing trajectory optimization.

Consider a dynamical system

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \dot{x} = f(x, u) = \begin{bmatrix} (1 - x_2^2)x_1 - x_2 + u \\ x_1 \end{bmatrix}, \quad x(0) = x_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

With $T = 10$, consider the following optimal control problem

$$\begin{aligned} \min_{u(t), t\in[0,T]} \quad & \int_{t=0}^{T} \|x(t)\|^2 + u(t)^2 dt \\ \text{subject to} \quad & \dot{x} = f(x, u), \quad x(0) = x_0 \\ & u(t) \in [-1, 1], \forall t \in [0, T]. \end{aligned} \tag{5.8}$$

a. Solve the problem using direct multiple shooting with $N = 50$ time intervals.

b. Solve the problem using direct collocation with $N = 50$.

Plot the optimized control signal and resulting state trajectory for both a and b. You probably want to refer to the source codes of Example 3.6 and 3.7.

Exercise 5.5 (Policy Iteration of Shortest Path)**.** In Example 2.2, we used value iteration to solve the shortest path problem with obstacles. In this exercise, we will implement policy iteration. Instead of value iteration that updates the Q-function by $Q^{(k+1)} = \mathcal{T}Q^{(k)}$ where $\mathcal{T}$ is the Bellman optimality operator defined in (2.32), we use a different update rule:

1. Initialize the policy π_0.
2. For $k = 0, 1, 2, ...$, compute Q_{π_k} via (2.29). Then update policy by greedy method $\pi_{k+1} = \pi_{Q^{\pi_k}}$ (see (2.31) for the definition of π_Q).

Now try to solve the following problems.

a. With the same obstacles in Fig. 2.3, implement the above policy iteration algorithm. Can you recover the cost-to-go and shortest path found in the example?

b. We designed the cost function g to be 20 when there is an obstacle. Is there a lower bound on this value, such that the shortest path with any starting point will still avoid obstacles? If yes, how will the lower bound change with different obstacle structures? (Bonus) With the assumption that the grids are connected, give a conjecture on the maximum of the lower bound.

c. If it is allowed to go diagonally (for example, from (4,4) to (3,5)), how will the results change?

d. (Bonus) Beyond iterative algorithms, there is another formulation of the problem in linear programming. Try to implement the following problem:

$$\max_J \sum_x J(x), \quad s.t. J(x) \leq g(x,u) + \gamma \sum_{x'} P(x'|x,u)J(x'), \ \forall x, u.$$

What can you find? (Hint: formulate the problem as a linear programming problem.)

Exercise 5.6 (Verifying Control Lyapunov Function for the Double Integrator)**.** Consider the following discrete-time double integrator dynamics

$$x_{t+1} = \underbrace{\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}}_{A} x_t + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B} u_t,$$

with control constraints

$$u \in \mathcal{U} = [-1, 1].$$

We do not enforce any state constraint on x, that is $\mathcal{X} = \mathbb{R}^2$.

Receding horizon controller. We aim to regulate the system at the origin, and design the following receding horizon controller (RHC) with horizon $N = 3$

$$\begin{aligned} \min_{u(0),\dots,u(N-1)} \quad & x(N)^T P x(N) + \sum_{k=0}^{N-1} x(k)^T Q x(k) + u(k)^T R u(k) \\ \text{subject to} \quad & u(k) \in \mathcal{U}, \forall k = 0, \dots, N-1 \\ & x(k+1) = Ax(k) + Bu(k), \forall k = 0, \dots, N-1 \\ & x(0) = x_t, \end{aligned} \tag{5.9}$$

where x_t is the system state at time t, and P, Q, R matrices are designed as follows

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = 1, \quad P = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}.$$

a. Implement the RHC in (5.9), and run the RHC with initial system state $x_0 = [2; 1]$. Plot the system trajectory on a 2D plane like Fig. 3.24.

b. From Theorem 3.3, we know that a sufficient condition for the stability of RHC is that the terminal cost, in our example is $p(x) = x^T P x$, needs to satisfy (3.56). When $p(x)$ satisfies (3.56), we say $p(x)$ is a control Lyapunov function (CLF). Instantiating (3.56) for this exercise, it becomes (recall that we do not have terminal constraint, i.e., $\mathcal{X}_f = \mathbb{R}^2$):

$$\rho = \min_{u \in \mathcal{U}} \ (Ax+Bu)^T P (Ax+Bu) - x^T P x + x^T Q x + u^T R u \leq 0, \quad \forall x \in \mathbb{R}^2. \tag{5.10}$$

Computing ρ for all x in $\mathbb{R}^2$ is difficult, so I will ask you to compute ρ along the state trajectory generated by RHC in (a). Compute and plot the trajectory of ρ.

c. From Proposition 2.2, we know the optimal cost-to-go of the unconstrained infinite-horizon LQR problem with $x(0) = x$

$$J_\infty(x) = \min_{u(0),\dots} \sum_{k=0}^{\infty} u(k)^T R u(k) + x(k)^T Q x(k), \quad \text{subject to} \quad x(k+1) = Ax(k) + Bu(k),$$

is a quadratic function

$$J_\infty(x) = x^T S x,$$

where S can be computed by solving the algebraic Riccati equation. Now use Matlab or Python to compute S (e.g., using `dlqr` in Matlab) for the double integrator.

d. Redo (a) and (b) by using S in (c) as the terminal cost, i.e., setting $P = S$. Plot the state trajectory, as well as the trajectory of ρ.

e. Redo (a) and (b) by using $0.1S$ and $10S$ as the terminal cost, i.e., setting $P = 0.1S$ and $P = 10S$. Plot the state trajectory, as well as the trajectory of ρ.

Exercise 5.7 (Polyhedral Controllable and Reachable Sets)**.** In this exercise, we aim to compute controllable and reachable sets explicitly, rather than implementing MPT.

a. A bounded polyhedron in $\mathbb{R}^n$ can be expressed as the convex hull of a finite set of points (vertices) $V = \{V_1, \dots, V_k\} \subset \mathbb{R}^n$. Show that a linear transform of such a polyhedron is simply tranforming its vertices, namely,

$$A\text{conv}(V) + b = \text{conv}(AV) + b,$$

where $A \in \mathbb{R}^{m\times n},\ b \in \mathbb{R}^m$.

b. The exact definition of a (closed) polyhedron $\mathcal{P}$ is a set defined by finite number of linear inequalities $\mathcal{P} = \{x \in \mathbb{R}^n \mid Hx \le h\}$ where $H \in \mathbb{R}^{m\times n},\ h \in \mathbb{R}^m$. Let $A \in \mathbb{R}^{n\times n}$ be an invertible matrix, $b \in \mathbb{R}^n$. Suppose the linear transformed polyhedron $A\mathcal{P} + b$ has representation

$$\{y \in \mathbb{R}^n \mid H'y \le h'\}.$$

What are H' and h'?

c. With the same setting as Example 3.10, consider linear system

$$x_{t+1} = \begin{bmatrix} 1.5 & 0 \\ 1 & -1.5 \end{bmatrix} x_t + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_t$$

with state and control constraints

$$\mathcal{X} = [-10, 10]^2, \quad \mathcal{U} = [-5, 5].$$

Compute $\text{Pre}(\mathcal{X})$ and $\text{Suc}(\mathcal{X})$ in the form of $\{x \in \mathbb{R}^n \mid Hx \le h\}$. What are the vertices of two sets respectively? (Hints: Denote the dynamics as $x_{t+1} = Ax_t + Bu_t$. For the successor set,

$$\text{Suc}(\mathcal{X}) = \cup_{u\in\mathcal{U}}\{x : \exists x' \in \mathcal{X},\ \text{s.t.},\ Ax' + Bu = x\} = \cup_{u\in\mathcal{U}}(A\mathcal{X} + Bu).$$

What is $A\mathcal{X} + Bu$ for each u? Will the shape of $A\mathcal{X} + Bu$ change when changing u? The conclusion in (b) might help. Similarly, for the precursor set,

$$\text{Pre}(\mathcal{X}) = \cup_{u\in\mathcal{U}}\{x : Ax + Bu \in \mathcal{X}\} = \cup_{u\in\mathcal{U}}A^{-1}(\mathcal{X} - Bu).$$

Mimic the computation of the successor set.)

d. Continued with the results in (c), compute one-step controllable set $\mathcal{K}_1(\mathcal{X}) = \text{Pre}(\mathcal{X}) \cap \mathcal{X}$. (Hints: There are 4 inequalities for each of the two sets. Any redundancy in those 8 inequalities when combining them together?)

e. From the above questions, we get the intuition of how the MPT compute controllable set:

(i) Obtain the representation of precursor set;
(ii) Intersect the precursor set with the feasible domain $\mathcal{X}$;
(iii) Remove redundant (useless) inequalities.

Provided the following algorithm that removes redundancy of a polyhedral representation, write the codes to find $\mathcal{K}_1(\mathcal{X})$ without using MPT. Does the outcome match your results in (d)?

Algorithm: Find redundacy-free representation of a polyhedron

---

Input: $H \in \mathbb{R}^{m\times n}$ and $h \in \mathbb{R}^m$ that represents $\mathcal{P} = \{x \in \mathbb{R}^n \mid Hx \leq h\}$

Output: $H_0 \in \mathbb{R}^{m_0\times n}$ and $h_0 \in \mathbb{R}^{m_0}$ that represents $\mathcal{P}$ without redundancy

$\mathcal{I} \leftarrow \{1, \dots, m\}$

For $i = 1$ **to** m

$\mathcal{I} \leftarrow \mathcal{I} \setminus \{i\}$

$f^* \leftarrow \max_x H_i x,\ s.t. H_{\mathcal{I}} x \leq h_{\mathcal{I}}, H_i x \leq h_i + 1$

%% H_i is the i-th row of H, $H_{\mathcal{I}}$ concatenates rows of H with index in $\mathcal{I}$

If $f^* > h_i$ **Then** $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$

$H_0 \leftarrow H_{\mathcal{I}}, h_0 \rightarrow h_{\mathcal{I}}$

Exercise 5.8 (Certified Region of Attraction of Clipped LQR Controller)**.** In this exercise, let us use the simple pendulum as an example to (i) recognize the difficulty of nonlinear control in the presence of control limits, and (ii) appreciate the power of Lyapunov analysis and Sums-of-Squares (SOS) programming.

A starting script for this exercise can be found here, where you need to fill out some specific details to finish this exercise.

Let us consider the continuous-time pendulum dynamics

$$\dot{x} = \begin{bmatrix} x_2 \\ \frac{1}{ml^2}(u - bx_2 + mgl\sin x_1) \end{bmatrix} \tag{5.11}$$

where x_1 is the angular position of the pendulum, and x_2 is the angular velocity. Note that the dynamics above is written such that $x = 0$ represents the upright position.

In the case of no control saturation, i.e., $u \in \mathbb{R}$, stabilizing the pendulum at $x = 0$ is easy, we can linearize the dynamics at $x = 0$ and obtain

$$\dot{x} \approx \underbrace{\begin{bmatrix} 0 & 1 \\ \frac{g}{l} & -\frac{b}{ml^2} \end{bmatrix}}_{A} x + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix}}_{B} u.$$

We then solve an infinite-horizon LQR problem

$$\min \int_{t=0}^{\infty} x(t)^T Q x(t) + u(t)^T R u(t) dt, \quad \text{subject to} \quad x(0) = x_0.$$

The solution to this LQR problem can be written in closed form, see Section **??**. Calling the Matlab `lqr` function (which is for continuous-time LQR), we can get a feedback controller

$$u(x) = -Kx, \tag{5.12}$$

with the optimal cost-to-go

$$J(x_0) = x_0^T S x_0, \tag{5.13}$$

with K and S constant matrices.

Control saturation. Assume now our control has hard bounds, i.e.,

$$u \in [-u_{\max}, u_{\max}].$$

We wish to keep using the LQR controller (5.12). Therefore, our saturated controller will be

$$\bar{u}(x) = \text{clip}(u(x), -u_{\max}, u_{\max}) = \text{clip}(-Kx, -u_{\max}, u_{\max}), \tag{5.14}$$

where the clip function is defined as

$$\text{clip}(u, -u_{\max}, u_{\max}) = \begin{cases} u_{\max} & \text{if} \quad u \geq u_{\max} \\ u & \text{if} \quad -u_{\max} \leq u \leq u_{\max} \\ -u_{\max} & \text{if} \quad u \leq -u_{\max} \end{cases}.$$

A natural question is: can the saturated controller $\bar{u}(t)$ in (5.14) still stabilize the pendulum?

Let's simulate the pendulum and the controller to investigate this. We will use the parameter $m = 1, g = 9.8, l = 1, b = 0.1$, and $u_{\max} = 2$. For the LQR cost matrix, let us use $Q = I$ and $R = 1$.

a. Compute the LQR gain K and implement the saturated controller (5.14). In a region $\mathcal{X} = [-0.2\pi, 0.2\pi] \times [-0.2\pi, 0.2\pi]$, sample $N = 1000$ initial states, and for each initial state, simulate the system under the saturated controller using `ode45` or `ode89`. There will be some initial states from which the pendulum gets stabilized and others from which the pendulum fails to be stabilized. Plot the stabilized initial states as "circles", and the non-stabilized initial states as "squares" on a 2D plot. (Hint: you should get something like Fig. 5.4.)

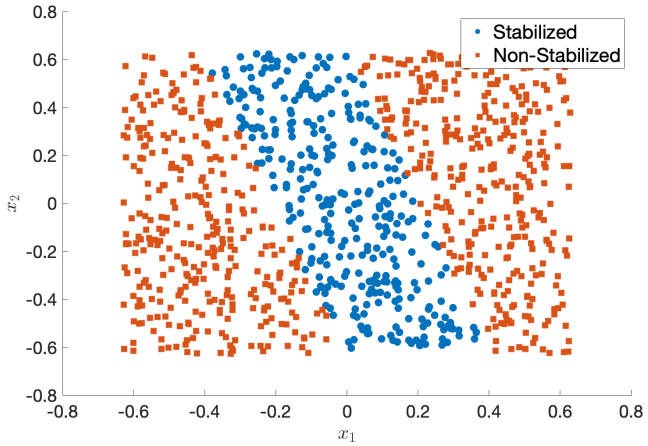


Figure 5.4: Example of stabilized and non-stabilized initial states.

The plot in Fig. 5.4 intuitively makes sense: when the initial state is very close to 0, the saturated controller can still stabilize the pendulum.

Now our goal is to use Sums-of-Squares and Lyapunov analysis to get a **certified region of stabilization**, also known as the region of attraction (ROA), under the saturated control.

Approximate polynomial dynamics. The SOS tool requires the system dynamics to be polynomial. The dynamics in (5.11) is polynomial except the term $\sin x_1$. Therefore, we perform a Taylor expansion of $\sin x_1$

$$\sin x_1 = x_1 - \frac{x_1^3}{3!} + \frac{x_1^5}{5!} + \dots,$$

leading to an approximate polynomial dynamics

$$\dot{x} = \bar{f}(x,u) = \begin{bmatrix} x_2 \\ \frac{1}{ml^2}\left(u - bx_2 + mgl\left(x_1 - \frac{x_1^3}{3!} + \frac{x_1^5}{5!}\right)\right) \end{bmatrix}. \tag{5.15}$$

Candidate ROA. We want to certify the following candidate ROA

$$\Omega_\rho = \{x \in \mathbb{R}^2 \mid J(x) := x^T S x \le \rho\}$$

for some $\rho > 0$, and S is exactly the LQR cost-to-go in (5.13). Since $J(x)$ is already positive definite (because $S \succ 0$), Ω_ρ will look like an elliptical region around the origin. To certify all initial states inside Ω_ρ can be stabilized, we need $\dot{J}(x)$ to be negative definite on Ω, under the saturated LQR controller (5.14) (according to Theorem 4.2), that is to say

$$\dot{J}(x) = \frac{\partial J}{\partial x}\bar{f}(x, \bar{u}(x)) < 0, \quad \forall x \in \Omega_\rho \backslash \{0\}.$$

A sufficient condition for the above equation to hold is

$$-\frac{\partial J}{\partial x}\bar{f}(x, \bar{u}(x)) - \epsilon\|x\|^2 \quad \text{is SOS on } \Omega_\rho. \tag{5.16}$$

for some $\epsilon > 0$ (do you see this?).

The condition (5.16) is almost ready for us to implement in SOSTOOLS. However, there is one last issue. The saturated controller $\bar{u}(x)$ is not a polynomial!

The last trick. Fortunately, we can use a trick here to save us. A closer look at (5.16) and the saturated controller $\bar{u}(x)$ shows that it is equivalent to asking

$$\begin{aligned}
-\frac{\partial J}{\partial x}\bar{f}(x, u_{\max}) - \epsilon\|x\|^2 \quad &\text{is SOS on} \quad \Omega_\rho \cap \{x \mid u(x) \geq u_{\max}\} \\
-\frac{\partial J}{\partial x}\bar{f}(x, u(x)) - \epsilon\|x\|^2 \quad &\text{is SOS on} \quad \Omega_\rho \cap \{x \mid -u_{\max} \leq u(x) \leq u_{\max}\} \\
-\frac{\partial J}{\partial x}\bar{f}(x, -u_{\max}) - \epsilon\|x\|^2 \quad &\text{is SOS on} \quad \Omega_\rho \cap \{x \mid u(x) \leq -u_{\max}\}.
\end{aligned} \tag{5.17}$$

Essentially, (5.17) breaks the SOS condition into three cases, each corresponding to one case in the clip function. (1) If $u(x) \geq u_{\max}$, then the first equation takes $u_{\max}$ to be the controller, (2) If $u(x) \leq -u_{\max}$, then the third equation takes $-u_{\max}$ to be the controller, (3) otherwise, the second equation takes $u(x) = -Kx$ to be the controller. Condition (5.17) has three SOS constraints and can be readily implemented using SOSTOOLS. We will use $\epsilon = 0.01$.

b. Choose $\rho = 1$, and implement the SOS conditions in (5.17) with a chosen relaxation order κ (in the code I choose $\kappa = 4$). Is the SOS program feasible (i.e., does the SOS program produce a certificate)? If so, plot the boundary of Ω_ρ on top of the samples plot in Fig. 5.4. Does Ω_ρ agree with the samples? (Hint: you should see a plot similar to Fig. 5.5.)

c. Now try $\rho = 2, 3, 4, 5$, for which values of ρ the SOS program is feasible, and for which values of ρ the SOS program becomes infeasible?

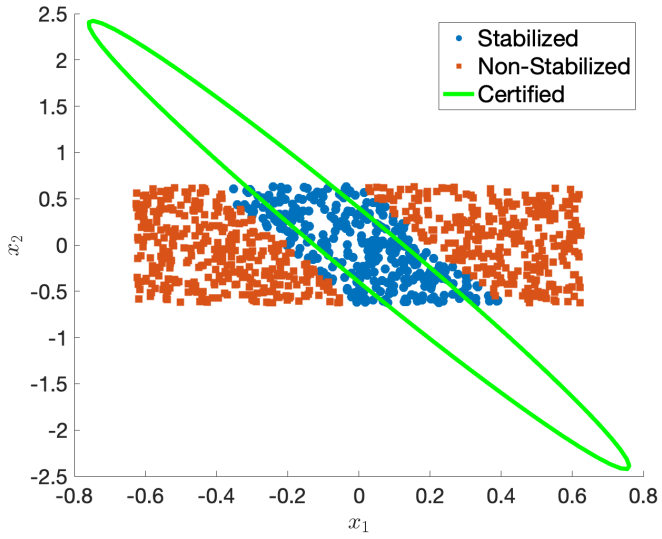


Figure 5.5: Certified Region of Attraction.

Exercise 5.9 (Energy pumping of Simple Pendulum)**.** In Exercise 5.8, we have studied the region of attraction of the Clipped LQR Controller. The conclusion we got there is, once we enter the region of attraction Ω_ρ (the green elliptical region in Fig. 5.5), we can switch to the clipped LQR controller and it guarantees stabilization towards the upright position.

In this exercise, we are going to design a controller that can swing up the pendulum to enter the region of attraction.

The total energy of the pendulum is given by

$$E = \frac{1}{2}ml^2x_2^2 + mgl\cos x_1,$$

where x_1 and x_2 are notations in (5.11).

a. With the dynamics in (5.11), one can obtain the change of energy over time:

$$\dot{E} = h(x,u)\cdot x_2.$$

Find the expression of $h(x,u)$.

b. Our desired energy is the one with the upright equilibruim: $E^d = mgl$. Define the energy of difference

$$\tilde{E} = E - E^d.$$

Find the expression of $\dot{\tilde{E}}$.

c. Now consider the feedback controller of the form

$$u = -kx_2\tilde{E} + bx_2.$$

Find the expression of $\dot{\tilde{E}}$. (Bonus) Explain heuristically why the controller works.

d. However, our controller has saturation as before:

$$\bar{u}(x) = \text{clip}(u(x), -u_{\max}, u_{\max}).$$

Set the parameters the same as the previous exercise and $k = 1$. Simulate the system under the saturated controller with initial state $(x_1, x_2) = (\pi, 1)$. Will the state hit the candidate ROA Ω_ρ with $\rho = 2$? If yes, stop the system once it hits the region, and draw the trajectory on the Figure similar to Fig. 5.5. Try $k = 0.01, 0.1, 1, 10$, which one hits the region the fastest?

Congrats! We have designed a full nonlinear controller (energy pumping + clipped LQR) that guarantees swing-up of the pendulum from any initial state to the upright position. Try the controller for yourself.

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.

$$\exists x \in \textbf{\textit{relint}}\mathcal{D} : \quad f_i(x) < 0, \quad i = 1...m, \quad Ax = b$$

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 $c_1 x_1 + c_2 x_2$ 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:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T P x + q^T x \tag{B.6}$$

$$\text{subject to } Gx \leq h \tag{B.7}$$

$$Ax = b \tag{B.8}$$

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,

$$\mathbb{P}(a_i^T x \leq b_i) = \mathbb{P}(a_i^T x - b_i \leq 0) \tag{B.18}$$

$$= \mathbb{P}(t \leq -||\Sigma_i^{\frac{1}{2}} x||_2^{-1} (\bar{a}_i^T x - b_i)) \tag{B.19}$$

$$= \Phi(-||\Sigma_i^{\frac{1}{2}} x||_2^{-1} (\bar{a}_i^T x - b_i)) \tag{B.20}$$

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,

$$\mathbb{P}(a_i^T x \leq b_i) \geq p \tag{B.21}$$

$$\Longleftrightarrow \Phi(-||\Sigma_i^{\frac{1}{2}} x||_2^{-1} (\bar{a}_i^T x - b_i)) \geq p \tag{B.22}$$

$$\Longleftrightarrow -||\Sigma_i^{\frac{1}{2}} x||_2^{-1} (\bar{a}_i^T x - b_i) \geq \Phi^{-1}(p) \tag{B.23}$$

$$\Longleftrightarrow \Phi^{-1}(p) ||\Sigma_i^{\frac{1}{2}} x||_2 \leq b_i - \bar{a}_i^T x \tag{B.24}$$

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 4.3, 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 $B, AB, A^2B, \ldots, A^{n-1}B$.*

Proof. Directly from Cayley Hamilton Theorem, A^n can be expressed as a linear combination of $I, A, A^2, \ldots, A^{n-1}$. By recursion, it is easy to show that for all $m > n$, A^m is also a linear combination of $I, A, A^2, \ldots, A^{n-1}$. 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:*

$$\mathit{rank}(\begin{bmatrix} B & AB & \ldots & A^{n-1}B \end{bmatrix}) = \mathit{rank}(\begin{bmatrix} B & AB & \ldots & A^{k-1}B \end{bmatrix})$$

Proof. First prove LHS $\leq$ RHS. $\forall v \in \mathbb{C}^n$ such that

$$v^* \begin{bmatrix} B & AB & \ldots & A^{k-1}B \end{bmatrix} = v^* \begin{bmatrix} B & AB & \ldots & A^{n-1}B & \ldots A^{k-1}B \end{bmatrix} = 0$$

$v^* \begin{bmatrix} B & AB & \ldots & A^{n-1}B \end{bmatrix} = 0$ must hold.

Second prove LHS $\geq$ RHS. For any $v \in \mathbb{C}^n$ such that $v^* \begin{bmatrix} B & AB & \ldots & A^{n-1}B \end{bmatrix} = 0$ and any $k > n$, by Corollary C.2, there exists a sequence $c_i, i = 0 \ldots n-1$ satisfy the following:

$$v^* A^k B = v^* \sum_{i=0}^{n-1} c_i A^i B = 0$$

Therefore, $v^* \begin{bmatrix} B & AB & \ldots & A^{k-1}B \end{bmatrix} = 0$. □

Corollary C.4. *For any $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times m}$, define*

$$\mathcal{C} = \begin{bmatrix} B & AB & \ldots & A^{n-1}B \end{bmatrix}$$

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

$$\bar{\mathcal{C}} := \begin{bmatrix} \bar{B}_c & \bar{A}_c\bar{B}_c & \bar{A}_c^2\bar{B}_c & \ldots & \bar{A}_c^{k_1-1}\bar{B}_c \end{bmatrix}$$

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 $q_1 \ldots q_{k_1}, q_i \in \mathbb{C}^n$. Then, we arbitrarily set other $n-k_1$ vectors $q_{k_1+1} \ldots q_n$ as long as

$$Q = \begin{bmatrix} q_1 & \ldots & q_{k_1} & q_{k_1+1} & \ldots & q_n \end{bmatrix}$$

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:

$$\det(A - \lambda I) = \lambda^n + a_1\lambda^{n-1} + \cdots + a_n\lambda$$

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

$$\left.\frac{d^i F}{dt^i}\right|_{t=0} = v^* A^i e^{At} B\big|_{t=0} = v^* A^i B = 0, \quad i = 0 \dots n-1$$

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.

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.

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

Bertsekas, D. (1972). Infinite time reachability of state-space regions by using feedback control. *IEEE Transactions on Automatic Control*, 17(5):604–613.

Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific.

Blekherman, G., Parrilo, P. A., and Thomas, R. R. (2012). *Semidefinite optimization and convex algebraic geometry*. SIAM.

Borrelli, F., Bemporad, A., and Morari, M. (2017). *Predictive control for linear and hybrid systems*. Cambridge University Press.

Chen, C.-T. (1984). *Linear system theory and design*. Saunders college publishing.

Dai, H. and Permenter, F. (2023). Convex synthesis and verification of control-lyapunov and barrier functions with input constraints. In *2023 American Control Conference (ACC)*, pages 4116–4123. IEEE.

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*.

Gilbert, E. G. and Tan, K. T. (1991). Linear systems with state and control constraints: The theory and application of maximal output admissible sets. *IEEE Transactions on Automatic control*, 36(9):1008–1020.

Kang, S., Chen, Y., Yang, H., and Pavone, M. (2023). Verification and synthesis of robust control barrier functions: Multilevel polynomial optimization and semidefinite relaxation. In *2023 62nd IEEE Conference on Decision and Control (CDC)*.

Kelly, M. (2017). An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904.

Kolmanovsky, I., Gilbert, E. G., et al. (1998). Theory and computation of disturbance invariant sets for discrete-time linear systems. *Mathematical problems in engineering*, 4:317–367.

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.

Levine, N., Zahavy, T., Mankowitz, D. J., Tamar, A., and Mannor, S. (2017). Shallow updates for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 30.

Magron, V. and Wang, J. (2023). *Sparse polynomial optimization: theory and practice*. World Scientific.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Murray, R., Chandrasekaran, V., and Wierman, A. (2021). Signomial and polynomial optimization via relative entropy and partial dualization. *Mathematical Programming Computation*, 13:257–295.

Nie, J. (2023). *Moment and Polynomial Optimization*. SIAM.

Nocedal, J. and Wright, S. J. (1999). *Numerical optimization*. Springer.

Sontag, E. D. (1983). A lyapunov-like characterization of asymptotic controllability. *SIAM journal on control and optimization*, 21(3):462–471.

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, A. and Boyd, S. (2023). Value-gradient iteration with quadratic approximate value functions. *arXiv preprint arXiv:2307.07086*.

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.