

# Contents

<b>3</b>	<b>LQR</b>	<b>2</b>
3.1	Motivation . . . . .	2
3.2	Optimal control . . . . .	3
3.2.1	Discretization . . . . .	4
3.3	The Linear Quadratic Regulator Problem . . . . .	5
3.4	Optimality and the Riccati Equation . . . . .	7
3.4.1	Expected state at time $h$ . . . . .	10
3.5	Extensions . . . . .	11
3.5.1	Time-dependency . . . . .	12
3.5.2	General quadratic cost . . . . .	12
3.5.3	Tracking a predefined trajectory . . . . .	13
3.6	The infinite-horizon setting . . . . .	13
3.7	Approximating nonlinear dynamics . . . . .	14
3.7.1	Local linearization . . . . .	14
3.7.2	Iterative LQR . . . . .	16
3.8	Programming and Implementation . . . . .	17
3.9	Exercises . . . . .	17

# Chapter 3

## Linear Quadratic Regulators

### 3.1 Motivation

Have you ever tried balancing a pen upright on your palm? If not, try it! It's a lot harder than seems. Unlike the settings we studied in the previous chapter, the state space and action space in this example aren't *finite*, or even *discrete*. Instead, they are *continuous* (real-valued) and therefore *uncountably infinite*. In addition, the state transitions governing the system – that is, the laws of physics – are highly complex.

This task is an example of a classic *control problem*:

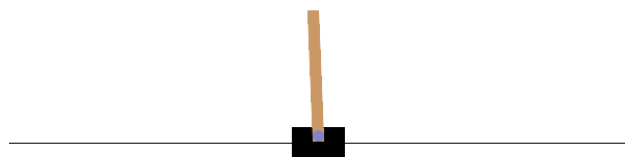


Figure 3.1: CartPole: A pole balanced on a cart.

#### Example 3.1.1: CartPole

Consider a pole balanced on a cart. The state  $x$  consists of just four continuous values:

1. The position of the cart;
2. The velocity of the cart;

3. The angle of the pole;
4. The angular velocity of the pole.

We can *control* the cart by applying a horizontal force  $u$ .

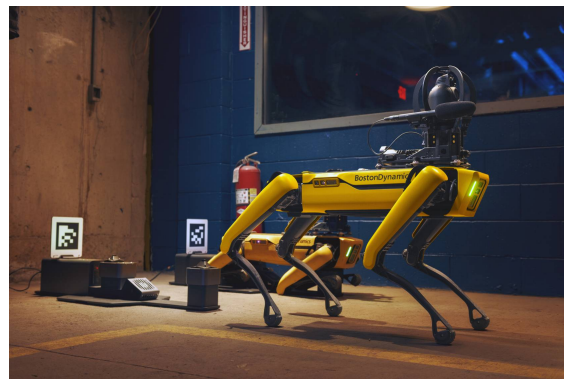
**Goal:** Stabilize the cart around an ideal state and action  $(x^*, u^*)$ .

*Controls* are the analogue to *actions* in MDPs. In control theory, the state and controls are typically denoted as  $x$  and  $u$ , but we'll stick with the  $x$  and  $u$  notation to highlight the similarity with MDPs.

Beyond this simple scenario, there are many real-world examples that involve continuous control. Here are just a few: Autonomous driving, controlling a robot's joints, and automated manufacturing. How can we teach computers to solve these kinds of problems?



(a) Solving a Rubik's Cube with a robot hand.



(b) Boston Dynamics's Spot robot.

Figure 3.2: Examples of control tasks.

In the last chapter, we developed efficient algorithms (*value iteration* and *policy iteration*) for calculating the optimal value function  $V^*$  and optimal policy  $\pi^*$  in an MDP. In this chapter, we'll develop similar algorithms for the continuous control setting.

Note that we're still assuming that the entire environment is *known* (i.e. the state transitions, rewards, etc). We'll explore the unknown case in the next chapter.

## 3.2 Optimal control

Recall that an MDP is defined by its state space  $\mathcal{S}$ , action space  $\mathcal{A}$ , state transitions  $P$ , reward function  $r$ , and discount factor  $\gamma$  or time horizon  $H$ . What are the equivalents in the control setting?

- The state and action spaces are *continuous* rather than finite. That is,  $\mathcal{S} = \mathbb{R}^{n_x}$  and  $\mathcal{A} = \mathbb{R}^{n_u}$ , where  $n_x$  and  $n_u$  are the number of coordinates to specify a single state or action respectively.

- We call the state transitions the **dynamics** of the system. In the most general case, these might change across timesteps and also include some stochastic **noise**  $w_h$ . We denote these dynamics as the function  $f_h$ , such that  $x_{h+1} = f_h(x_h, u_h, w_h)$ . Of course, we can simplify to cases where the dynamics are *deterministic/noise-free* (no  $w_h$  term) or are *stationary/time-homogeneous* (the same function  $f$  across timesteps).
- Instead of a reward function, it's more common to consider a **cost function**  $c_h : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  that describes *how far away* we are from our **goal state-action pair**  $(x^*, u^*)$ . An important special case is when the cost is *time-homogeneous*; that is, it remains the same function  $c$  at each timestep.
- We seek to minimize the *undiscounted* cost within a *time horizon*  $H$ . Note that we end an episode at  $x_H$  – there is no  $u_H$ , and so we denote the cost for the final state as  $c_H(x_H)$ .

With all of these components, we can now formulate the **optimal control problem**: *find a time-dependent policy to minimize the expected undiscounted cost over  $H$  timesteps.*

**Definition 3.2.1: Optimal control problem**

$$\begin{aligned}
 \min_{\pi_0, \dots, \pi_{H-1} : \mathcal{S} \rightarrow \mathcal{A}} \quad & \mathbb{E}_{x_0, w_h} \left[ \left( \sum_{h=0}^{H-1} c_h(x_h, u_h) \right) + c_H(x_H) \right] \\
 \text{where} \quad & x_{h+1} = f_h(x_h, u_h, w_h), \\
 & u_h = \pi_h(x_h) \\
 & x_0 \sim \mu_0 \\
 & w_h \sim \text{noise}
 \end{aligned} \tag{3.1}$$

### 3.2.1 Discretization

How does this relate to the finite horizon case? If  $x_h$  and  $u_h$  were discrete, then we'd be able to work backwards using the DP algorithms we saw before (??). As a matter of fact, let's consider what happens if we *discretize* the problem. For intuition, suppose  $n_x = n_u = 1$  (that is, states and actions are real numbers). To make  $\mathcal{S}$  and  $\mathcal{A}$  discrete, let's choose some small positive  $\epsilon$ , and simply round states and actions to the nearest multiple of  $\epsilon$ . For example, if  $\epsilon = 0.01$ , then we're just rounding  $x$  and  $u$  to two decimal spaces.

How well does this work? Even if our  $\mathcal{S}$  and  $\mathcal{A}$  are finite, the existing algorithms might take unfeasibly long to complete. Suppose our state and action spaces are bounded by some constants  $\max_{x \in \mathcal{S}} \|x\| \leq B_x$  and  $\max_{u \in \mathcal{A}} \|u\| \leq B_u$ . Then using our rounding method, we must divide *each dimension* into intervals of length  $\epsilon$ , resulting in  $(B_x/\epsilon)^{n_x}$  and  $(B_u/\epsilon)^{n_u}$  total points. To get a sense of how quickly this grows, let's consider  $\epsilon = 0.01, n_x = n_u = 10$ . Then the number of elements in our transition matrix is  $|\mathcal{S}|^2 |\mathcal{A}| = (100^{10})^2 (100^{10}) = 10^{60}$ ! Try finding a computer that'll fit that in memory! (For reference, 32 GB of memory can store  $10^9$  32-bit floating point numbers.)

So as we've seen, discretizing the problem becomes impractical as soon as the action and state spaces are even moderately high-dimensional. How can we do better?

Note that by discretizing the state and action spaces, we implicitly assumed that rounding each state or action vector by some tiny amount  $\varepsilon$  wouldn't change the behavior much; namely, that the functions involved were relatively *continuous*. Can we use this continuous structure in other ways? This brings us to the topic of **Linear Quadratic Regulators**, a widely used and studied tool in control theory.

### 3.3 The Linear Quadratic Regulator Problem

The optimal control problem stated above seems very difficult to solve. The cost function might not be convex, making optimization difficult, and the state transitions might be very complex, making it difficult to satisfy the constraints. Is there a relevant simplification that we can analyze?

We'll show that a natural structure to impose is *linear dynamics* and a *quadratic cost function* (in both arguments). This is called the **linear quadratic regulator** (LQR) model, and is a popular tool in control theory. In fact, some people even design systems to be linear in order to use results from LQR!

Why are these assumptions useful? As we'll see later in the chapter, it lets us *locally approximate* nonlinear dynamics and cost functions using their *Taylor approximations* (up to first and second order respectively). We'll also find that even for more complex setups, we can generalize the algorithms for LQR to get surprisingly good solutions.

#### Definition 3.3.1: The linear quadratic regulator

**Linear, time-homogeneous dynamics:**

$$x_{h+1} = f(x_h, u_h, w_h) = Ax_h + Bu_h + w_h$$

**Quadratic, time-homogeneous cost function:**

$$c(x_h, u_h) = \begin{cases} x_h^\top Q x_h + u_h^\top R u_h & h < H \\ x_H^\top Q x_H & h = H \end{cases}$$

We want  $c$  to be a convex function (easy to optimize) in both  $x_h$  and  $u_h$ , so we'll set  $Q$  and  $R$  to both be positive definite matrices. We can furthermore assume without loss of generality that they are both symmetric (see exercise below).

Intuitively, the cost function punishes states and actions that are far away from the origin (i.e. both the state and action are zero vectors). More generally, we'll want to replace the origin with a *goal* state and action  $(x^*, u^*)$ . This can easily be done by replacing  $x_h$  with  $(x_h - x^*)$  and  $u_h$  with  $(u_h - u^*)$  in the expression above.

**Spherical Gaussian noise:**

$$w_h \sim \mathcal{N}(0, \sigma^2 I)$$

Putting everything together, the optimization problem we want to solve is:

$$\begin{aligned} \min_{\pi_0, \dots, \pi_{H-1}: \mathcal{S} \rightarrow \mathcal{A}} \quad & \mathbb{E} \left[ \left( \sum_{h=0}^{H-1} x_h^\top Q x_h + u_h^\top R u_h \right) + x_H^\top Q x_H \right] \\ \text{where} \quad & x_{h+1} = A x_h + B u_h + w_h \\ & u_h = \pi_h(x_h) \\ & w_h \sim \mathcal{N}(0, \sigma^2 I) \\ & x_0 \sim \mu_0. \end{aligned}$$

**Exercise:** We've set  $Q$  and  $R$  to be positive definite and symmetric. Here we'll show that the symmetry condition can be imposed without loss of generality. Show that replacing  $Q$  with  $(Q + Q^\top)/2$  (which is symmetric) yields the same cost function.

So how do we go about analyzing this system? A good first step might be to introduce *value functions*, analogous to those in the previous chapter, to reason about the behavior of the system over the time horizon.

### Definition 3.3.2: Value functions for LQR

Given a policy  $\pi = (\pi_0, \dots, \pi_{H-1})$ , we can define the value function  $V_h^\pi : \mathcal{S} \rightarrow \mathbb{R}$  as

$$\begin{aligned} V_h^\pi(x) &= \mathbb{E} \left[ \left( \sum_{i=h}^{H-1} c(x_i, u_i) \right) + c(x_H) \mid x_h = x, \forall i \geq h. u_i = \pi_i(x_i) \right] \\ &= \mathbb{E} \left[ \left( \sum_{i=h}^{H-1} x_i^\top Q x_i + u_i^\top R u_i \right) + x_H^\top Q x_H \mid x_h = x, \forall i \geq h. u_i = \pi_i(x_i) \right] \end{aligned}$$

The expression inside the expectation is called the **cost-to-go**, since it's just the total cost starting from timestep  $h$ .

The  $Q$  function additionally conditions on the first action we take:

$$\begin{aligned} Q_h^\pi(x, u) &= \mathbb{E} \left[ \left( \sum_{i=h}^{H-1} c(x_i, u_i) \right) + c(x_H) \mid (x_h, u_h) = (x, u), \forall i \geq h. u_i = \pi_i(x_i) \right] \\ &= \mathbb{E} \left[ \left( \sum_{i=h}^{H-1} x_i^\top Q x_i + u_i^\top R u_i \right) + x_H^\top Q x_H \mid (x_h, u_h) = (x, u), \forall i \geq h. u_i = \pi_i(x_i) \right] \end{aligned}$$

As in the previous chapter, these will be instrumental in constructing optimal policy  $\pi$  via dynamic programming.

## 3.4 Optimality and the Riccati Equation

In this section, we'll compute the optimal value function and policy in the LQR setting using **dynamic programming**, in a very similar way to the DP algorithms we saw in the previous chapter.

1. We'll compute  $V_H^*(x)$  as our base case.
2. Then we'll work backwards, using  $V_{h+1}^*(x)$  to compute  $Q_h^*(x, u)$ ,  $\pi_{h+1}^*(x)$ , and  $V_h^*(x)$ .

Along the way, we will prove that  $V_h^*$  and  $\pi_h^*$  have very simple structure:  $V_h^*(x)$  is a *convex quadratic* and  $\pi_h^*(x)$  is *linear*.

### Definition 3.4.1: Optimal value functions for LQR

The **optimal value function** is the one that, at any time, in any state, achieves *minimum cost across all policies*:

$$\begin{aligned} V_h^*(x) &= \min_{\pi_h, \dots, \pi_{H-1}} V_h^\pi(x) \\ &= \min_{\pi_h, \dots, \pi_{H-1}} \mathbb{E} \left[ \left( \sum_{i=h}^{H-1} x_i^\top Q x_i + u_i^\top R u_i \right) + x_H^\top Q x_H \mid x_h = x, u_i = \pi_i(x_i) \quad \forall i \geq h \right] \end{aligned}$$

### Theorem 3.4.1: Optimal value function in LQR is a convex quadratic

$$V_h^*(x) = x^\top P_h x + p_h$$

for some time-dependent  $P_h \in \mathbb{R}^{n_x \times n_x}$  and  $p_h \in \mathbb{R}^{n_x}$  where  $P_h$  is symmetric and positive definite. Note that there is no linear term.

### Theorem 3.4.2: Optimal policy in LQR is linear

$$\pi_h^*(x) = -K_h x$$

for some  $K_h \in \mathbb{R}^{k \times d}$ . (The negative is due to convention.)

**Base case:**  $V_H^*(x) = x^\top P_H x + p_H$ . At the final timestep, there are no possible actions to take, and so  $V_H^*(x) = c(x) = x^\top Q x$ . Thus  $P_H = Q$  and  $p_H$  is the zero vector.

**Inductive hypothesis:** We seek to show that the inductive step holds for both theorems: If  $V_{h+1}^*(x)$  is a convex quadratic, then  $V_h^*(x)$  must also be a convex quadratic, and  $\pi_h^*(x)$  must be linear. We'll break this down into the following steps:

- Step 1.** Show that  $Q_h^*(x, u)$  is a convex quadratic (in both  $x$  and  $u$ ).
- Step 2.** Derive the optimal policy  $\pi_h^*(x) = \arg \min_u Q_h^*(x, u)$  and show that it's linear.
- Step 3.** Show that  $V_h^*(x)$  is a convex quadratic.

This is essentially the same proof that we wrote in the finite-horizon MDP setting, except

now the state and action are *continuous* instead of finite.

We first assume our theorems are true at time  $h + 1$ . That is,

$$V_{h+1}^*(x) = x^\top P_{h+1}x + p_{h+1} \quad \text{for all states } x \in \mathcal{S}.$$

**Step 1.** We'll start off by demonstrating that  $Q_h^*(x)$  is a convex quadratic. Recall that the definition of  $Q_h^* : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is

$$Q_h^*(x, u) = c(x, u) + \mathbb{E}_{x' \sim f(x, a, w_{h+1})} V_{h+1}^*(x').$$

We know  $c(x, u) := x^\top Qx + u^\top Ru$ . Let's consider the average value over the next timestep. The only randomness in the dynamics comes from the noise  $w_{h+1}$ , so we can write out this expected value as:

$$\begin{aligned} & \mathbb{E}_{x' \sim f(x, a, w_{h+1})} V_{h+1}^*(x') \\ &= \mathbb{E}_{w_{h+1} \sim \mathcal{N}(0, \sigma^2 I)} V_{h+1}^*(Ax + Bu + w_{h+1}) && \text{definition of } f \\ &= \mathbb{E}_{w_{h+1}} [(Ax + Bu + w_{h+1})^\top P_{h+1} (Ax + Bu + w_{h+1}) + p_{h+1}]. && \text{inductive hypothesis} \end{aligned}$$

Summing and combining like terms, we get

$$\begin{aligned} Q_h^*(x, u) &= x^\top Qx + u^\top Ru + \mathbb{E}_{w_{h+1}} [(Ax + Bu + w_{h+1})^\top P_{h+1} (Ax + Bu + w_{h+1}) + p_{h+1}] \\ &= x^\top (Q + u^\top P_{h+1} A)x + u^\top (R + B^\top P_{h+1} B)u + 2x^\top u^\top P_{h+1} Ba \\ &\quad + \mathbb{E}_{w_{h+1}} [w_{h+1}^\top P_{h+1} w_{h+1}] + p_{h+1}. \end{aligned}$$

Note that the terms that are linear in  $w_h$  have mean zero and vanish. Now consider the remaining expectation over the noise. By expanding out the product and using linearity of expectation, we can write this out as

$$\mathbb{E}_{w_{h+1}} [w_{h+1}^\top P_{h+1} w_{h+1}] = \sum_{i=1}^d \sum_{j=1}^d (P_{h+1})_{ij} \mathbb{E}_{w_{h+1}} [(w_{h+1})_i (w_{h+1})_j].$$

When dealing with these *quadratic forms*, it's often helpful to consider the terms on the diagonal ( $i = j$ ) separately from those off the diagonal. On the diagonal, the expectation becomes

$$(P_{h+1})_{ii} \mathbb{E}(w_{h+1})_i^2 = (P_{h+1})_{ii} \text{Var}((w_{h+1})_i) = \sigma^2 (P_{h+1})_{ii}.$$

Off the diagonal, since the elements of  $w_{h+1}$  are independent, the expectation factors, and since each element has mean zero, the term disappears:

$$(P_{h+1})_{ij} \mathbb{E}(w_{h+1})_i \mathbb{E}(w_{h+1})_j = 0.$$

Thus, the only terms left are the ones on the diagonal, so the sum of these can be expressed as the trace of  $\sigma^2 P_{h+1}$ :

$$\mathbb{E}_{w_{h+1}} [w_{h+1}^\top P_{h+1} w_{h+1}] = \text{Tr}(\sigma^2 P_{h+1}).$$



Substituting this back into the expression for  $Q_h^*$ , we have:

$$Q_h^*(x, u) = x^\top (Q + u^\top P_{h+1} A) x + u^\top (R + B^\top P_{h+1} B) a + 2x^\top u^\top P_{h+1} B a + \text{Tr}(\sigma^2 P_{h+1}) + p_{h+1}. \quad (3.2)$$

As we hoped, this expression is quadratic in  $x$  and  $u$ . Furthermore, we'd like to show that it also has positive curvature with respect to  $u$  so that its minimum with respect to  $u$  is well-defined. This is fairly straightforward:

**Theorem 3.4.3:  $Q_h^*$  is a convex quadratic**

Consider the part of Equation 3.2 that is quadratic in  $u$ , namely  $u^\top (R + B^\top P_{h+1} B) u$ . Then  $Q_h^*$  is a convex quadratic with respect to  $u$  if  $R + B^\top P_{h+1} B$  is symmetric and positive definite (SPD).

To see why this is the case, recall that in our definition of LQR, we assumed that  $R$  is SPD (see Definition 3.3.1). Also note that since  $P_{h+1}$  is SPD (by the inductive hypothesis), so too must be  $B^\top P_{h+1} B$ . (If this isn't clear, try proving it as an exercise!) Since the sum of two SPD matrices is also SPD, we have that  $R + B^\top P_{h+1} B$  is SPD, and so  $Q_h^*$  is convex with respect to  $u$ . A similar proof shows that  $Q + u^\top P_{h+1} A$  is SPD, and so  $Q_h^*$  is also a convex quadratic.

**Step 2.** Now let's move on to the next part of proving the inductive hypothesis: showing that  $\pi_h^*(x) = \arg \min_u Q_h^*(x, u)$  is linear. Since  $Q_h^*$  is a convex quadratic, finding its minimum over  $u$  is easy: we can just take the gradient with respect to  $u$  and set it to zero. First, we calculate the gradient:

$$\begin{aligned} \nabla_u Q_h^*(x, u) &= \nabla_u [u^\top (R + B^\top P_{h+1} B) a + 2x^\top u^\top P_{h+1} B a] \\ &= 2(R + B^\top P_{h+1} B) a + (2x^\top u^\top P_{h+1} B)^\top \end{aligned}$$

Setting this to zero, we get

$$\begin{aligned} 0 &= (R + B^\top P_{h+1} B) a + B^\top P_{h+1} A x \\ \pi_h^*(x) &:= a = -(R + B^\top P_{h+1} B)^{-1} B^\top P_{h+1} A x \\ &= -K_h x, \end{aligned} \quad (3.3)$$

where  $K_h = (R + B^\top P_{h+1} B)^{-1} B^\top P_{h+1} A$ .

Note that this optimal policy has an interesting property: in addition to being independent of the starting distribution  $\mu_0$  (which also happened for our finite-horizon MDP solution), it's also fully deterministic and isn't affected by noise! (Compare this with the discrete MDP case, where calculating our optimal policy required taking an expectation over the state transitions.)

**Step 3.** To complete our inductive proof, we must show that the inductive hypothesis is true at time  $h$ ; that is, we must prove that  $V_h^*(x)$  is quadratic. Using the identity  $V_h^*(x) = Q_h^*(x, \pi^*(x))$ , we have:

$$V_h^*(x) = Q_h^*(x, \pi^*(x))$$

$$\begin{aligned}
&= x^\top (Q + u^\top P_{h+1} A) x + (-K_h x)^\top (R + B^\top P_{h+1} B) (-K_h x) + 2x^\top u^\top P_{h+1} B (-K_h x) \\
&\quad + \text{Tr}(\sigma^2 P_{h+1}) + p_{h+1}
\end{aligned}$$

Note that with respect to  $x$ , this is the sum of a quadratic term and a constant, which is exactly what we were aiming for!

To conclude our proof, let's concretely specify the values of  $P_h$  and  $p_h$ . The constant term is clearly  $p_h = \text{Tr}(\sigma^2 P_{h+1}) + p_{h+1}$ . We can simplify the quadratic term by substituting in  $K_h$ . Notice that when we do this, the  $(R + B^\top P_{h+1} B)$  term in the expression is cancelled out by its inverse, and the remaining terms combine to give what is known as the *Riccati equation*:

**Definition 3.4.2: Riccati equation**

$$P_h = Q + u^\top P_{h+1} A - u^\top P_{h+1} B (R + B^\top P_{h+1} B)^{-1} B^\top P_{h+1} A.$$

There are several nice things to note about the Riccati equation:

1. It's defined **recursively**. Given the dynamics defined by  $u$  and  $B$ , and the state coefficients  $Q$ , we can recursively calculate  $P_h$  across all timesteps starting from  $P_H = Q$ .
2.  $P_h$  often appears in calculations surrounding optimality, such as  $V_h^*$ ,  $Q_h^*$ , and  $\pi_h^*$ .
3. Together with  $A$ ,  $B$ , and the action coefficients  $R$ , it fully defines the optimal policy.

Now we've shown that  $V_h^*(x) = x^\top P_h x + p_h$ , which is a convex quadratic, and this concludes our proof. ■

In summary, we just demonstrated that:

- The optimal value function  $V_h^*$  is convex at all  $h$ .
- The optimal  $Q$ -function  $Q_h^*$  is convex (in both arguments) at all  $h$ .
- The optimal policy  $\pi_h^*$  is linear at all  $h$ .
- All of these quantities can be calculated using a symmetric matrix  $P_h$  for each timestep, which can be defined recursively using the Riccati equation.

Before we move on to some extensions of LQR, let's consider how the state at time  $h$  behaves when we act according to this optimal policy.

### 3.4.1 Expected state at time $h$

Suppose you're about to go to bed, and the thermostat in your room is controlled by an optimal policy under LQR. A very reasonable question to ask would be: what's the expected state (i.e. temperature) of the room at a given time  $h$ ? Certainly you don't want to freeze or boil overnight!

To answer this question, let's first express the state at time  $h$  in a cleaner way in terms of the history. Note that having linear dynamics makes it easy to expand terms backwards in time:

$$x_h = Ax_{h-1} + Bu_{h-1} + w_{h-1}$$

$$\begin{aligned}
&= A(Ax_{h-2} + Bu_{h-2} + w_{h-2}) + Bu_{h-1} + w_{h-1} \\
&= \dots \\
&= u^h x_0 + \sum_{i=0}^{h-1} u^i (Bu_{h-i-1} + w_{h-i-1}).
\end{aligned}$$

Let's consider the *average state* at this time, given all the past states and actions. Since we assume that  $\mathbb{E} w_h = 0$  (this is the zero vector in  $d$  dimensions), when we take an expectation, the  $w_h$  term vanishes due to linearity, and so we're left with

$$\mathbb{E}[x_h \mid x_{0:(h-1)}, u_{0:(h-1)}] = u^h x_0 + \sum_{i=0}^{h-1} u^i Bu_{h-i-1}.$$

If we choose actions according to our optimal policy, this becomes

$$\mathbb{E}[x_h \mid x_0, \forall i \leq h. u_i = -K_i x_i] = \left( \prod_{i=0}^{h-1} (A - BK_i) \right) x_0.$$

This introduces the quantity  $A - BK_i$ , which shows up frequently in control theory. For example, one important question is: will  $x_h$  remain bounded, or will it go to infinity as time goes on? To answer this, let's imagine that these  $K_i$  are equal (call this matrix  $K$ ). Then the expression above becomes  $(A - BK)^h x_0$ . Now consider the maximum eigenvalue  $\lambda_{\max}$  of  $A - BK$ . If  $|\lambda_{\max}| > 1$ , then there's some nonzero initial state  $\bar{x}_0$ , the corresponding eigenvector, for which

$$\lim_{h \rightarrow \infty} (A - BK)^h \bar{x}_0 = \lambda_{\max}^h \bar{x}_0 = \infty.$$

By then, your room is *definitely* on fire! Otherwise, if  $|\lambda_{\max}| < 1$ , then it's impossible for your original state to explode as dramatically (assuming it's properly normalized).

We've now formulated an optimal solution for the typical, time-homogeneous case of LQR, and considered the expected state under the optimal policy. However, this simple case is insufficient for more complex tasks. In the following sections, we'll consider some motivating examples, and extensions of LQR where some assumptions are relaxed.

## 3.5 Extensions

In this section, we'll consider settings where some of the assumptions we made above are relaxed. Specifically, we'll consider:

1. **Time-dependency**, where the dynamics and cost function might change depending on the timestep.
2. **General quadratic cost**, where we allow for linear terms and a constant term.
3. **Tracking a goal trajectory** rather than aiming for a single goal state-action pair.

### 3.5.1 Time-dependent dynamics and cost function

So far, we've considered the *time-homogeneous* case, where the dynamics and cost function stay the same at every timestep. However, this might not always be the case. For example, if we want to preserve the temperature in a greenhouse, the outside forces are going to change depending on the time of day. A xanother example, in many sports or video games, the rules and scoring system might change during overtime. To address these sorts of problems, we can loosen the time-homogeneous restriction, and consider the case where the dynamics and cost function are *time-dependent*. Our analysis remains almost identical; in fact, we can simply add a time index to the matrices  $u$  and  $B$  that determine the dynamics and the matrices  $Q$  and  $R$  that determine the cost. (A xan exercise, walk through the derivation and verify this claim!)

The modified problem is now defined as follows:

$$\begin{aligned} \arg \min_{\pi_0, \dots, \pi_{H-1}: \mathcal{S} \rightarrow \mathcal{A}} \quad & \mathbb{E} \left[ \left( \sum_{h=0}^{H-1} (x_h^\top Q_h x_h) + u_h^\top R_h u_h \right) + x_H^\top Q_H x_H \right] \\ \text{where} \quad & x_{h+1} = f_h(x_h, u_h, w_h) = u_h x_h + B_h u_h + w_h \\ & x_0 \sim \mu_0 \\ & u_h = \pi_h(x_h) \\ & w_h \sim \mathcal{N}(0, \sigma^2 I). \end{aligned}$$

The derivation of the optimal value functions and the optimal policy remains almost exactly the same, and we can modify the Riccati equation accordingly:

**Definition 3.5.1: Time-dependent Riccati Equation**

$$P_h = Q_h + u_h^\top P_{h+1} u_h - u_h^\top P_{h+1} B_h (R_h + B_h^\top P_{h+1} B_h)^{-1} B_h^\top P_{h+1} u_h.$$

Note that this is just the time-homogeneous Riccati equation (??), but with the time index added to each of the relevant matrices.

Additionally, by allowing the dynamics to vary across time, we gain the ability to *locally approximate* nonlinear dynamics at each timestep. We'll discuss this later in the chapter.

### 3.5.2 More general quadratic cost functions

Our original cost function had only second-order terms with respect to the state and action. We can also consider more general quadratic cost functions that also have first-order terms and a constant term. Combining this with time-dependent dynamics results in the following expression, where we introduce a new matrix  $M_h$  for the cross term, linear coefficients  $q_h$  and  $r_h$  for the state and action respectively, and a constant term  $c_h$ :

$$c_h(x_h, u_h) = (x_h^\top Q_h x_h + x_h^\top M_h u_h + u_h^\top R_h u_h) + (x_h^\top q_h + u_h^\top r_h) + c_h. \quad (3.4)$$

Similarly, we can also include a constant term  $v_h \in \mathbb{R}^{n_x}$  in the dynamics (note that this is *fixed* at each timestep, unlike the noise  $w_h$ ):

$$x_{h+1} = f_h(x_h, u_h, w_h) = u_h x_h + B_h u_h + v_h + w_h.$$

The derivation of the optimal solution in this case will be left as a homework exercise.

### 3.5.3 Tracking a predefined trajectory

So far, we've been trying to get the robot to stay as close as possible to the origin, or more generally a goal state-action pair  $(x^*, u^*)$ . However, consider applying LQR to autonomous driving. Now, we want the desired state to change over time, instead of remaining in one location. Otherwise, it wouldn't be a very useful vehicle! In these cases, we want the robot to follow a predefined *trajectory* of states and actions  $(x_h^*, u_h^*)_{h=0}^{H-1}$ . To do this, we'll modify the cost function accordingly:

$$c_h(x_h, u_h) = (x_h - x_h^*)^\top Q(x_h - x_h^*) + (u_h - u_h^*)^\top R(u_h - u_h^*).$$

Note that this punishes states and actions that are far from the intended trajectory. By expanding out these multiplications, we can see that this is actually a special case of the more general quadratic cost function we discussed above (Equation 3.4):

$$M_h = 0, \quad q_h = -2Qx_h^*, \quad r_h = -2Ru_h^*, \quad c_h = (x_h^*)^\top Q(x_h^*) + (u_h^*)^\top R(u_h^*).$$

## 3.6 The infinite-horizon setting

Another assumption we've made is that the task has a *finite horizon*  $H$ . How about tasks that might matter indefinitely, where we want to minimize the expected cost over *all future timesteps*? Consider, for example, controlling the long-term value of a portfolio, or managing acidity levels in a lake.

In the previous chapter, we dealt with such **infinite-horizon** cases by *discounting* future rewards. This time, we'll take a different approach by considering the limit of a *finite-horizon* task as  $H \rightarrow \infty$ . A bit turns out, our derivation is exactly analogous to value iteration from the previous chapter! However, here the structure is nice enough that we don't need a discount factor  $\gamma$  to deal with limits analytically. (Note that we must normalize by  $1/H$  to keep the total cost bounded.)

In the discounted case, analogously to taking the limit as  $H \rightarrow \infty$ , we consider the limit as the discount factor  $\gamma$  approaches 1. This is because as  $\gamma \rightarrow 1$ , time discounting becomes less and less important – just like the horizon  $H$  vanishing into the distance – and we're left with the undiscounted case. (Note that for the total reward to remain bounded, we must normalize the sum by  $1 - \gamma$ .) Let's consider value iteration in this setting, which uses the Bellman operator  $\mathcal{J}$  to update  $V$ :

$$V_{h+1}(x) = \lim_{\gamma \rightarrow 1} (\mathcal{J}V_h)(x) = \lim_{\gamma \rightarrow 1} \max_a \left( r(x, a) + \gamma \mathbb{E}_{x' \sim P(x, a)} V_h(x') \right).$$

This is exactly analogous to the *Riccati equation* (??)! Instead of thinking of  $P_{h+1}$  as defining the value function for the *next timestep*, though, we think of it as the *next version* of the value function in an iterative algorithm. Then both of these algorithms are doing the same thing: iteratively refining the value function by acting greedily with respect to the current iteration.

By going through the same derivation in section 3.4, we'll see that  $P_h$  (defined recursively by the Riccati equation) converges to a fixed value  $P$ . For an intuitive perspective on why this happens, let's suppose you have an upcoming project deadline. When it's still a few months away, you might not pay much attention to it, and behave as you normally do. But as the deadline gets closer and closer, suddenly the horizon becomes more and more relevant, and you'll spend more time thinking about it. The infinite-horizon case is just where the deadline is infinitely far away, so you can just behave "like normal" all the time!

## 3.7 Approximating nonlinear dynamics

The LQR algorithm solves for the optimal policy when the dynamics are *linear* and the cost function is a *convex quadratic*. However, real settings are rarely this simple! Let's return to the CartPole example from the start of the chapter (Example 3.1.1). The dynamics (physics) aren't linear, and we might also want to specify a cost function that's more complex.

Concretely, let's consider a *noise-free* problem since, as we saw, the noise doesn't factor into the optimal policy. Let's assume the dynamics and cost function are stationary, and ignore the terminal state for simplicity:

### Definition 3.7.1: Nonlinear control problem

$$\begin{aligned} \min_{\pi_0, \dots, \pi_{H-1}: \mathcal{S} \rightarrow \mathcal{A}} \quad & \mathbb{E}_{x_0} \left[ \sum_{h=0}^{H-1} c(x_h, u_h) \right] \\ \text{where} \quad & x_{h+1} = f(x_h, u_h) \\ & u_h = \pi_h(x_h) \\ & x_0 \sim \mu_0 \\ & c(x, u) = d(x, x^*) + d(a, u^*). \end{aligned}$$

Here,  $d$  denotes some general measure of distance to the goal state and action  $(x^*, u^*)$ .

This is now only slightly simplified from the general optimal control problem (see 3.2.1). Here, we don't know an analytical form for the dynamics  $f$  or the cost function  $c$ , but we assume that we're able to *query/sample/simulate* them to get their values at a given state and action. How can we adapt LQR to this more general nonlinear case?

### 3.7.1 Local linearization

As we briefly mentioned in the introduction, part of the reason we designed the LQR problem the way we did was because we can take any *locally continuous* function, and approximate it using a Taylor expansion of low-order polynomials. By taking a linear approximation of  $f$  and a quadratic approximation of  $c$ , we're back to the regime of LQR with these derived matrices in terms of their gradients and Hessians accordingly.

**Assumptions.** This approach assumes that  $f$  is differentiable and that  $c$  is twice-differentiable, both around  $(x^*, u^*)$ . Additionally, since a Taylor expansion generally gets less and less accurate the further you stray from the point of expansion, this means that we also assume that all states are close to the optimal state  $x^*$ , and that we can stay close using actions that are close to  $u^*$ . If this seems like a strong set of restrictions, it is! But we'll save this discussion for the next section.

If you're unfamiliar with Taylor expansions, we recommend taking a calculus course; we'll use them here without much further introduction. Linearizing the dynamics around  $(x^*, u^*)$  gives:

$$f(x, u) \approx f(x^*, u^*) + \nabla_x f(x^*, u^*)(x - x^*) + \nabla_u f(x^*, u^*)(u - u^*)$$

$$(\nabla_x f(x, u))_{ij} = \frac{df_i(x, u)}{dx_j}, \quad i, j \leq n_x \quad (\nabla_u f(x, u))_{ij} = \frac{df_i(x, u)}{du_j}, \quad i \leq n_x, j \leq n_u$$

and quadratizing the cost function around  $(x^*, u^*)$  gives:

$$\begin{aligned} c(x, u) &\approx c(x^*, u^*) && \text{constant} \\ &+ \nabla_x c(x^*, u^*)(x - x^*) + \nabla_u c(x^*, u^*)(u - u^*) && \text{linear} \\ &+ \frac{1}{2}(x - x^*)^\top \nabla_{xx} c(x^*, u^*)(x - x^*) \\ &+ \frac{1}{2}(u - u^*)^\top \nabla_{uu} c(x^*, u^*)(u - u^*) && \text{quadratic} \\ &+ (x - x^*)^\top \nabla_{xu} c(x^*, u^*)(u - u^*) \end{aligned}$$

where the gradients and Hessians are defined as

$$\begin{aligned} (\nabla_x c(x, u))_i &= \frac{dc(x, u)}{dx_i}, \quad i \leq n_x & (\nabla_u c(x, u))_i &= \frac{dc(x, u)}{du_i}, \quad i \leq n_u \\ (\nabla_{xx} c(x, u))_{ij} &= \frac{d^2 c(x, u)}{dx_i dx_j}, \quad i, j \leq n_x & (\nabla_{uu} c(x, u))_{ij} &= \frac{d^2 c(x, u)}{du_i du_j}, \quad i, j \leq n_u \\ (\nabla_{xu} c(x, u))_{ij} &= \frac{d^2 c(x, u)}{dx_i du_j}, \quad i \leq n_x, j \leq n_u \end{aligned}$$

We note that this cost can be expressed in the general quadratic form seen in Equation 3.4. We leave it as an exercise to derive the corresponding matrices and vectors  $Q, R, M, q, r, c$ . To calculate these gradients and Hessians in practice, we use a method known as **finite differencing** for numerically computing derivatives. Namely, we can simply use the definition of derivative, and see how the function changes as we add or subtract a tiny  $\delta$  to the input.

However, simply taking the second-order approximation of the cost function is insufficient; we also need to ensure that it is locally convex, that is, the Hessians  $\nabla_{xx} c(x^*, u^*)$  and  $\nabla_{uu} c(x^*, u^*)$  are positive definite.

**Local convexification.** Recall that an equivalent definition of positive definite matrices is that all of their eigenvalues are positive. One way to naively *force*  $\nabla_{xx} c(x^*, u^*)$  and  $\nabla_{uu} c(x^*, u^*)$  to be positive definite is to set any negative eigenvalues to some small positive value. Note that

both of these Hessians are symmetric, and so they can be decomposed in terms of their eigenbasis. Thus by removing their eigenvalues and adding a small ‘lower bound’ to the eigenvalues (so that the surface will have some minimum amount of curvature), we obtain

$$\begin{aligned}\nabla_{xx}c(x^*, u^*) &= \sum_{i=1}^{n_x} \sigma_i u_i u_i^\top \\ \nabla_{xx}\tilde{c}(x^*, u^*) &= \sum_{\substack{i=1 \\ \sigma_i > 0}}^{n_x} \sigma_i u_i u_i^\top + \lambda I.\end{aligned}$$

We can use a similar approach to convexify  $\nabla_{uu}\tilde{c}(x^*, u^*)$ . Now that we have a convex quadratic approximation to the cost function, and a linear approximation to the state transitions, we can simply apply the time-homogenous LQR methods from section 3.4.

But what happens when our assumptions break down, namely when we enter states far away from  $x^*$  and want to use actions far from  $u^*$ ? As we mentioned above, our Taylor approximation will typically become less accurate. To address this, we’ll need to do a Taylor approximation around *different points at each time step*.

### 3.7.2 Iterative LQR

**Iterative LQR** is a way to resolve the issues with local linearization. The key idea is to linearize around different points at each timestep, creating a time-dependent approximation of the dynamics. We’ll break it into a few steps:

- Step 1.** Form an LQR around the current candidate trajectory  $(\bar{x}_h^i, \bar{u}_h^i)_{h=0}^{H-1}$  using local approximation.
- Step 2.** Apply the solution to time-dependent LQR from subsection 3.5.1 to obtain an optimal policy  $\pi^i$ .
- Step 3.** Generate a new trajectory  $(\tilde{u}_h)_{h=0}^{H-1}$  using  $\pi^i$ .
- Step 4.** Compute a better candidate trajectory  $\mathbf{a}^{i+1}$  by interpolating between  $\mathbf{a}^i$  and  $\tilde{\mathbf{a}}$ .

Now the question becomes: How do we choose the best *waypoints*  $(\bar{x}_h, \bar{u}_h)$  to get the best approximation?

We’ll once again use an iterative approach, similarly to value iteration, where at each step we update the waypoints *greedily* with respect to the current iteration. Let’s use a superscript to denote the iteration of the algorithm. We can start off by initializing some (bad) sequence of waypoints  $\bar{u}_0^0, \dots, \bar{u}_{H-1}^0$  by some approximate method such as local linearization. Applying these actions at their respective timesteps gives a sample trajectory

$$\bar{x}_0, \bar{u}_0, \bar{x}_1, \bar{u}_1, \dots, \bar{x}_{H-1}, \bar{u}_{H-1} \quad \text{where} \quad \bar{x}_0^0 = \bar{x}_0 = \mathbb{E}_{x_0 \sim \mu_0} [x_0], \quad \bar{x}_{h+1}^0 = f(\bar{x}_h^0, \bar{u}_h^0).$$

Now, at each timestep  $h$ , we linearize  $f$  and quadratize (and convexify)  $c$  around the point generated in the sample trajectory, using the same Taylor expansion techniques we saw in the



previous section:

$$\begin{aligned}
 f_h(x, u) &\approx f(\bar{x}_h^i, \bar{a}_h^i) + \nabla_x f(\bar{x}_h^i, \bar{a}_h^i)(x - \bar{x}_h^i) + \nabla_a f(\bar{x}_h^i, \bar{a}_h^i)(a - \bar{a}_h^i) \\
 c_h(x, u) &\approx c(\bar{x}_h^i, \bar{a}_h^i) + \begin{bmatrix} x - \bar{x}_h^i & a - \bar{a}_h^i \end{bmatrix} \begin{bmatrix} \nabla_x c(\bar{x}_h^i, \bar{a}_h^i) \\ \nabla_a c(\bar{x}_h^i, \bar{a}_h^i) \end{bmatrix} \\
 &\quad + \frac{1}{2} \begin{bmatrix} x - \bar{x}_h^i & a - \bar{a}_h^i \end{bmatrix} \begin{bmatrix} \nabla_{xx} c(\bar{x}_h^i, \bar{a}_h^i) & \nabla_{xu} c(\bar{x}_h^i, \bar{a}_h^i) \\ \nabla_{ax} c(\bar{x}_h^i, \bar{a}_h^i) & \nabla_{uu} c(\bar{x}_h^i, \bar{a}_h^i) \end{bmatrix} \begin{bmatrix} x - \bar{x}_h^i \\ a - \bar{a}_h^i \end{bmatrix}
 \end{aligned}$$

Now let's use the time-dependent LQR solution to compute an optimal policy  $\pi_0^i, \dots, \pi_{H-1}^i$ . We can then generate a new sample trajectory by taking actions according to this optimal policy:

$$\bar{x}_0^{i+1} = \bar{x}_0, \quad \tilde{u}_h = \pi_h^i(\bar{x}_h^{i+1}), \quad \bar{x}_{h+1}^{i+1} = f(\bar{x}_h^{i+1}, \tilde{u}_h).$$

Note that the states are drawn by sampling from the *true* dynamics, and also that we've denoted these actions as  $\tilde{u}_h$  and aren't directly using them for the next iteration  $u_h^{i+1}$ . Rather, we want to interpolate between them and the actions from the previous iteration  $\bar{u}_0^i, \dots, \bar{u}_{H-1}^i$ . This is so that the cost will *increase monotonically*, since if the new policy turns out to actually be worse, we can stay closer to the previous trajectory. (Can you think of an intuitive example where this might happen?) Formally, we want to find  $\alpha \in [0, 1]$  to generate the next iteration of actions:

$$\begin{aligned}
 \min_{\alpha \in [0, 1]} \quad & \sum_{h=0}^{H-1} c(x_h, \bar{u}_h^{i+1}) \\
 \text{where} \quad & x_{h+1} = f(x_h, \bar{u}_h^{i+1}) \\
 & \bar{u}_h^{i+1} = \alpha \tilde{u}_h + (1 - \alpha) \bar{u}_h \\
 & x_0 = \bar{x}_0.
 \end{aligned}$$

Note that this is only optimizing over the closed interval  $[0, 1]$ , so by the Extreme Value Theorem it's guaranteed to have a global maximum.

The final output of this algorithm is a policy  $\pi^{n_{\text{steps}}}$  derived after  $n_{\text{steps}}$  of the algorithm. Though the proof is somewhat complex, one can show that for many nonlinear control problems, this solution converges to a locally optimal solution (in the policy space).

## 3.8 Programming and Implementation

Not sure how much to include here yet. WIP. Walk through a basic Python solution to OpenAI Gym and CartPole? (Or save this for homework?)

## 3.9 Exercises

1. Consider a cleaning robot with one wheel on each side of its body. Your pet has made a mess nearby, and you want to steer the robot to go clean it up. Let's represent the state of the robot as a 3-dimensional vector containing its  $(x, y)$  coordinates and its angle  $\theta$ ,

relative to some global reference frame. We can control the robot using its linear velocity  $v$  (change in  $x, y$ ) and angular velocity  $\omega$  (change in  $\theta$ ). For simplicity, we'll assume the robot is perfectly ideal and there's no noise in the system.

- (a) Formally describe the true of the system as a function  $f : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ . Assume the system works in timesteps of  $dt = 1x$  and that all distances are measured in meters.
- (b) Linearize the dynamics that you derived using a first-order Taylor approximation. Is this approximation stationary or time-dependent?
- (c) Suppose the mess is at location  $(x^*, y^*) = (2, 2)$ . We want to reach that state using a small amount of energy. Write down the quadratic cost function using  $Q = \text{diag}(1, 1, 0.5)$ , expressing that we care about the final position more than the state, and  $R = 0.2I$  to penalize large action steps.

Now that our LQR is set up, let's find the optimal policy.

- (a) Let the horizon be  $H = 50$  timesteps. Write down the recursive update formula for  $P_h$  (Hint: use the Riccati equation).
- (b) Write the closed form solution of the optimal policy at time  $H - 1$ . Verify that this is a linear function of the state.

2. (Todo.)