Contents

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 the classic CartPole *control problem*:

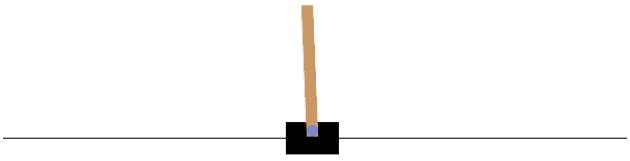

Figure 3.1: CartPole: A pole balanced on a cart.

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 $u \in \mathbb{R}$.

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

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 $V^\star$ and optimal policy $\pi^\star$ 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 γ 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** $c_h : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that describes *how far away* we are from our **goal state-action pair** $(x^\star, u^\star)$. 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

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

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 ϵ, and simply round states and actions to the nearest multiple of ϵ. For example, if $\epsilon = 0.01$, then we round 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 $\max_{x\in\mathcal{S}} \|x\| \le B_x$ and $\max_{u\in\mathcal{A}} \|u\| \le B_u$. Then using our rounding method, we must divide *each dimension* into intervals of length ε, resulting in $(B_x/\varepsilon)^{n_x}$ and $(B_u/\varepsilon)^{n_u}$ total points. To get a sense of how quickly this grows, let's consider $\varepsilon = 0.01, n_x = n_u = 10$. 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 ε wouldn't change the behavior of the system by much; namely, that the cost and dynamics 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 *(convex) quadratic cost function* (in both arguments). This model is called the **linear quadratic regulator** (LQR), and is a fundamental tool in control theory.

Why are these assumptions useful? As we'll see later in the chapter, it lets us *locally approximate* more complex 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 adapt the LQR model 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 require Q and R to both be positive definite matrices so that c has a well-defined unique minimum. 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 $(x^\star, u^\star)$. This can easily be done by replacing x_h with $(x_h - x^\star)$ and u_h with $(u_h - u^\star)$ in the expression above.

Spherical Gaussian noise:

$$w_h \sim \mathcal{N}(0, \sigma^2 I) \quad \forall h \in [H]$$

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

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

Exercise: We've set Q and R to be *symmetric* positive definite (SPD) matrices. 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 from MDPs (**??**), to reason about the behavior of the system over the time horizon.

Definition 3.3.2: Value functions for LQR

Given a policy $\pi = (\pi_0, \ldots, \pi_{H-1})$, we can define the value function $V_h^\pi : \mathcal{S} \to \mathbb{R}$ 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 the optimal policy π 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 **??**.

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

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

Definition 3.4.1: Optimal value functions for LQR

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

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

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

$$V_h^\star(x) = x^\top P_h x + p_h \quad \forall h \in [H]$$

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 SPD. Note that there is no linear term.

Theorem 3.4.2: Optimal policy in LQR is linear

$$\pi_h^\star(x) = -K_h x \quad \forall h \in [H]$$

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

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

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

Step 1. Show that $Q_h^\star(x,u)$ is a convex quadratic (in both x and u).

Step 2. Derive the optimal policy $\pi_h^\star(x) = \arg\min_u Q_h^\star(x,u)$ and show that it's linear.

Step 3. Show that $V_h^\star(x)$ 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 the inductive hypothesis that our theorems are true at time $h+1$. That is,

$$V^\star_{h+1}(x) = x^\top P_{h+1} x + p_{h+1} \quad \forall x \in \mathcal{S}.$$

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

$$Q^\star_h(x,u) = c(x,u) + \mathop{\mathbb{E}}_{x' \sim f(x,u,w_{h+1})} V^\star_{h+1}(x').$$

Recall $c(x,u) = x^\top Q x + u^\top R u$. 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:

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

Summing and combining like terms, we get

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

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

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

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

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

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 $\sigma^2 P_{h+1}$:

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

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

$$\boxed{\begin{aligned} Q_h^\star(x,u) = x^\top(Q + A^\top P_{h+1} A)x + u^\top(R + B^\top P_{h+1} B)u + 2x^\top A^\top P_{h+1} B u \\ + \operatorname{Tr}(\sigma^2 P_{h+1}) + p_{h+1}. \end{aligned}} \tag{3.2}$$

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, i.e.

$$\nabla_{uu} Q_h^\star(x,u) = R + B^\top P_{h+1} B$$

is positive definite, so that its minimum with respect to u is well-defined. This is fairly straightforward: 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 $Q_h^\star$ is indeed a convex quadratic with respect to u. A similar proof shows that it's also convex with respect to x.

Step 2. Now we aim to show that $\pi_h^\star$ is linear. Since $Q_h^\star$ is a convex quadratic, finding its minimum over u is easy: we simply set the gradient with respect to u equal to zero and solve for u. First, we calculate the gradient:

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

Setting this to zero, we get

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

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 μ_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 $V_h^\star(x)$ is a convex quadratic. Using the identity $V_h^\star(x) = Q_h^\star(x, \pi^\star(x))$, we have:

$$\begin{aligned} V_h^\star(x) &= Q_h^\star(x, \pi^\star(x)) \\ &= x^\top(Q + A^\top P_{h+1} A)x + (-K_h x)^\top(R + B^\top P_{h+1} B)(-K_h x) + 2x^\top A^\top P_{h+1} B(-K_h x) \\ &\quad + \operatorname{Tr}(\sigma^2 P_{h+1}) + p_{h+1} \end{aligned}$$

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 $p_h = \mathrm{Tr}(\sigma^2 P_{h+1}) + p_{h+1}$. 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

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

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

1. It's defined **recursively.** Given the dynamics defined by A and B, and the state cost matrix 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 $V_h^\star, Q_h^\star$, and $\pi_h^\star$.
3. Together with the dynamics given by A and B, and the action coefficients R, it fully defines the optimal policy.

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

In summary, we just demonstrated that:

- The optimal value function $V_h^\star$ is convex at all h.
- The optimal Q-function $Q_h^\star$ is convex (in both arguments) at all h.
- The optimal policy $\pi_h^\star$ 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

How can we compute the expected state at time h when acting according to the optimal policy? Let's first express x_h in a cleaner way in terms of the history. Note that having linear dynamics makes it easy to expand terms backwards in time:

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

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

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

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

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

Exercise: Verify this.

This introdces 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_is 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

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

Otherwise, if $|\lambda_{\max}| < 1$, then it's impossible for your original state to explode as dramatically.

3.5 Extensions

We've now formulated an optimal solution for the time-homogeneous LQR and computed the expected state under the optimal policy. However, real world tasks rarely have such simple dynamics, and we may wish to design more complex cost functions. In this section, we'll consider more general extensions of LQR 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. As another 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 A and B that determine the dynamics and the matrices Q and R that determine the cost. (As an exercise, walk through the derivation and verify this claim!)

The modified problem is now defined as follows:

Definition 3.5.1: Time-dependent LQR

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

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 3.5.2: Time-dependent Riccati Equation

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

Note that this is just the time-homogeneous Riccati equation (Definition 3.4.2), 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:

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

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

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

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 $(x^\star, u^\star)$. However, consider applying LQR to a task like autonomous driving, where the desired state changes over time, instead of remaining in one location. In these cases, we want the robot to follow a predefined *trajectory* of states and actions $(x_h^\star, u_h^\star)_{h=0}^{H-1}$. To express this as a control problem, we'll need a corresponding time-dependent cost function:

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

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):

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

3.6 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 3.6.1: Nonlinear control problem

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

Here, d denotes some general distance metric between its two arguments.

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. To clarify, consider the case where the dynamics are given by real world physics. We can't (yet) write down an expression for the dynamics that we can differentiate or integrate analytically. However, we can still *simulate* the dynamics and cost function by running a real-world experiment and measuring the resulting states and costs. How can we adapt LQR to this more general nonlinear case?

3.6.1 Local linearization

How can we apply LQR when the dynamics are nonlinear or the cost function is more complex? We'll exploit the useful fact that we can take any *locally continuous* function and approximate it using a Taylor expansion of low-order polynomials. In particular, as long as the dynamics f are differentiable around $(x^\star, u^\star)$ and the cost function c is twice differentiable at $(x^\star, u^\star)$, we can take a linear approximation of f and a quadratic approximation of c to bring us back to the regime of LQR.

Linearizing the dynamics around $(x^\star, u^\star)$ gives:

$$f(x,u) \approx f(x^\star,u^\star) + \nabla_x f(x^\star,u^\star)(x-x^\star) + \nabla_u f(x^\star,u^\star)(u-u^\star)$$
$$(\nabla_x f(x,u))_{ij} = \frac{df_i(x,u)}{dx_j}, \quad i,j \le n_x \qquad (\nabla_u f(x,u))_{ij} = \frac{df_i(x,u)}{du_j}, \quad i \le n_x, j \le n_u$$

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

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

where the gradients and Hessians are defined as

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

Exercise: Note that this cost can be expressed in the general quadratic form seen in Equation 3.4. Derive the corresponding quantities Q, R, M, q, r, c.

3.6.2 Finite differencing

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 limit definition of the derivative, and see how the function changes as we add or subtract a tiny δ to the input. Note that this only requires us to be able to *query* the function, not to have an analytical expression for it, which is why it's so useful in practice.

3.6.3 Local convexification

However, simply taking the second-order approximation of the cost function is insufficient, since for LQR we required that the Q and R matrices were positive definite, i.e. that all of their eigenvalues were positive.

One way to naively *force* some symmetric matrix D to be positive definite is to set any non-positive eigenvalues to some small positive value $\varepsilon > 0$. Recall that any real symmetric matrix $D \in \mathbb{R}^{n \times n}$ has an basis of eigenvectors $u_1, \ldots, u_n$ with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$ such that $Du_i = \lambda_i u_i$. Then we can construct the positive definite approximation by

$$\widetilde{D} = \sum_{i=1,\ldots,n | \lambda_i > 0} \lambda_i u_i u_i^\top + \varepsilon I.$$

Exercise: Convince yourself that $\widetilde{D}$ is indeed positive definite.

Note that Hessian matrices are generally symmetric, so we can apply this process to Q and R to obtain the positive definite approximations $\widetilde{Q}$ and $\widetilde{R}$. 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 we enter states far away from $x^\star$ or want to use actions far from $u^\star$? A Taylor approximation is only accurate in a *local* region around the point of linearization, so the performance of our LQR controller will degrade as we move further away. We'll see how to address this in the next section using the **iterative LQR** algorithm.

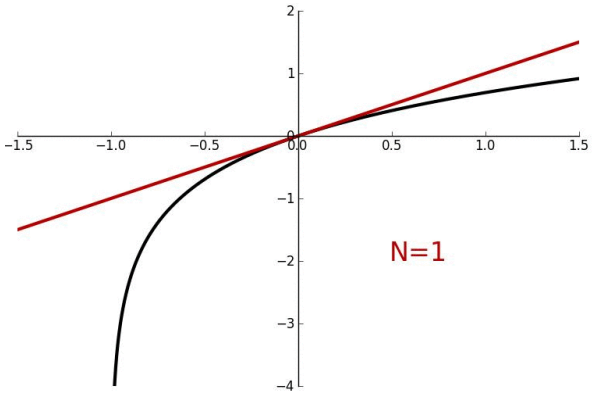

Figure 3.3: A visual intuition for local linearization.

3.6.4 Iterative LQR

Earlier, we tried to use local linearization to solve nonlinear control problems with the LQR, but we ran into the issue that the approximation is only accurate in a local region around the point of linearization. Instead, we'll use an iterative approach, where we repeatedly linearize around

different points to create a *time-dependent* approximation of the dynamics, and then solve the resulting time-dependent LQR problem to obtain a better policy. This is known as **iterative LQR** or **iLQR**:

Definition 3.6.2: Iterative LQR (high-level)

For each iteration of the algorithm:

Step 1. Form a time-dependent LQR problem around the current candidate trajectory using local linearization.

Step 2. Compute the optimal policy using subsection 3.5.1.

Step 3. Generate a new series of actions using this policy.

Step 4. Compute a better candidate trajectory by interpolating between the current and proposed actions.

Now let's go through the details of each step. We'll use superscripts to denote the iteration of the algorithm. We'll also denote $\bar{x}_0 = \mathbb{E}_{x_0 \sim \mu_0}[x_0]$ as the expected initial state.

At iteration i of the algorithm, we begin with a **candidate** trajectory $\bar{\tau}^i = (\bar{x}_0^i, \bar{u}_0^i, \ldots, \bar{x}_{H-1}^i, \bar{u}_{H-1}^i)$.

Step 1: Form a time-dependent LQR problem. At each timestep $h \in [H]$, we use the techniques from section 3.6 to linearize the dynamics and quadratize the cost function around $(\bar{x}_h^i, \bar{u}_h^i)$:

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

Step 2: Compute the optimal policy. We can now solve the time-dependent LQR problem using the Riccati equation from subsection 3.5.1 to compute the optimal policy $\pi_0^i, \ldots, \pi_{H-1}^i$.

Step 3: Generate a new series of actions. We can then generate a new sample trajectory by taking actions according to this optimal policy:

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

Note that the states are sampled according to the *true* dynamics, which we assume we have query access to.

Step 4: Compute a better candidate trajectory., Note that we've denoted these actions as $\widetilde{u}_h$ and aren't directly using them for the next iteration $\bar{u}_h^{i+1}$. Rather, we want to *interpolate* between them and the actions from the previous iteration $\bar{u}_0^i, \ldots, \bar{u}_{H-1}^i$. 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 $\bar{u}_0^{i+1}, \ldots, \bar{u}_{H-1}^{i+1}$ such that the cost is minimized:

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

Note that this optimizes 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_{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).