Chapter 3

Linear Quadratic Regulators

3.1 Motivation and control theory

Have you ever tried balancing a pen upright in your palm? If not, try it! It's a lot harder than it might seem. Unlike the cases we studied the previous chapter, the state and action spaces are not finite or even discrete. Instead, they are continuous and uncountably infinite. How can we teach computers to solve these sorts of *continuous control* problems?

In the last chapter, we covered Markov Decision Processes (MDPs) in the case where the state and action spaces were finite. We showed that, if we know the state transitions, we can calculate the optimal policy using efficient polynomial-time algorithms (Value Iteration and Policy Iteration).

What about the case where state and action spaces are infinite? This doesn't necessarily mean that they need to be super "complex", in a heuristic sense. They could simply be continuous, such as choosing the angle to tilt your steering wheel, or the force on a joint of a robotic drone.

Iterative in computation time, not samples. (Running `for` loop to compute some quantity as quickly as possible.)

Example 3.1.1: CartPole

Consider a pole balanced on a cart. The state 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.

The *control* we apply is a force on the cart, moving it left and right.

Goal: To stabilize the cart around an ideal state $s^\star$.

If you've ever tried to balance a pen upright in the palm of your hand, this is essentially the same problem!

How do we formulate the cost function $c : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ for this problem? We want something that we can easily generalize to other sorts of tasks. One way we can do so is simply by choosing a cost function that is *quadratic* in its arguments:

$$c(s_t, a_t) = a_t^\top R a_t + (s_t - s^\star)^\top Q (s_t - s^\star). \tag{3.1}$$

For some intuition into this expression, consider the simple case where a_t and s_t are one-dimensional, so $c(s_t, a_t) = r a_t^2 + q(s_t - s^\star)^2$.

This expression has the nice property that for any continuous cost function, we can write out its *Taylor approximation* around a given point. Then, we can think of the quadratic cost function we've written above as the second-order approximation to that smooth cost function.

Remark 3.1.1: Quadratic forms

If this notation is unfamiliar to you, we recommend checking out this video from Khan Academy!

Remark 3.1.2: Notation

We call actions *controls*. In the literature, these are commonly represented with the letter u instead of a, but here we'll stick to a in order to illustrate the similarities with the discrete case of RL.

Definition 3.1.1: Optimal control problem

$$\min_{\pi_0, \ldots, \pi_{T-1} : \mathcal{S} \to \mathcal{A}} \mathbb{E}\left[\sum_{t=0}^{T-1} c(s_t, a_t) \mid s_{t+1} = f(s_t, a_t), s_0 \sim \mu_0 \right] \tag{3.2}$$

The function $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the analogue of the state transitions. However, this might not be totally deterministic; there might be some underlying noise, or there might also be some measurement error in the state.

Our goal is to find a control policy π_t, which minimizes the *total cost* (undiscounted) over some finite number of steps T.

Note that this is a pretty hard problem the way it's written right now! We have some pretty strict constraints in the form of the state transitions.

How does this relate to the finite horizon case? If s_t and a_t were discrete, then we'd be able to work backwards using the dynamic programming algorithm we saw before.

As a matter of fact, let's consider what happens if we *discretize* the problem. Recall that $s \in \mathbb{R}^d$ and $a \in \mathbb{R}^k$, which are continuous. To make them discrete, let's round them to the nearest multiple of ϵ (for some choice of ϵ), so now the set of possible s and a is discrete. (Formally, we call this new set of rounded values an ϵ-grid over the original continuous space. For example, if $\epsilon = 0.01$, then we're just rounding to two decimal spaces.)

If both these state and action spaces can be bounded, then the resulting sets are actually finite, so now we can use our previous tools for MDPs! But is this actually a feasible solution? Even if our $\mathcal{S}$ and $\mathcal{A}$ are finite, it might still be unfeasible to run the existing algorithms.

Indeed, this happens to be the case. Suppose our state and action spaces are bounded by some constants $\max_{s\in\mathcal{S}} \|s\| \leq B_s$ and $\max_{a\in\mathcal{A}} \|a\| \leq B_a$. Then to form our ε-net, we must divide each dimension into intervals of length ε, resulting in $(B_s/\varepsilon)^d$ and $(B_a/\varepsilon)^k$ points!

To get a sense of how quickly this grows, let's consider $\varepsilon = 0.01, d = k = 10$. Then the number of elements in our transition matrix $|\mathcal{S}|^2|\mathcal{A}|$, is on the order of $(100^{10})^2(100^{10}) = 10^{60}$! Try finding a computer that'll fit that in memory!

So as we've seen, discretizing the problem isn't a feasible solution as soon as our action and state spaces are even moderately high-dimensional. How can we do better? Well, when doing the discretization, we implicitly relied on the assumption that rounding our value by some tiny amount ε wouldn't change the behavior much; namely, that the functions involved are relatively *continuous*. Can we use this structure in other ways? This brings us to the next topic, where we make some simplifying assumptions:

3.2 The Linear Quadratic Regulator Problem

Definition 3.2.1: Linear quadratic regulator problem

Linear dynamics: assume that the state transitions are linear with respect to the inputs:

$$s_{t+1} = f(s_t, a_t, w_t) = As_t + Ba_t + w_t$$

Note that we've also assumed that f is *time-homogeneous:* our state transitions behave the same way across all time steps.

Quadratic cost function:

$$c(s_t, a_t) = \begin{cases} s_t^\top Q s_t + a_t^\top R a_t & t < T \\ s_T^\top Q s_T & t = T \end{cases}$$

We want c to be a convex function so that there actually exists a minimum. This means that Q and R must both be positive definite.

In other words, we assume that we only take T actions total, and just ignore a_T.

Gaussian noise: $w_t \sim \mathcal{N}(0, \Sigma)$

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

$$\begin{aligned} \min_{\pi_0,\ldots,\pi_{T-1}:\mathcal{S}\to\mathcal{A}} \quad & \mathbb{E}\left[\left(\sum_{t=0}^{T-1} s_t^\top Q s_t + a_t^\top R a_t\right) + s_T^\top Q s_T\right] \\ \text{s.t.} \quad & s_{t+1} = A s_t + B a_t + w_t \\ & a_t = \pi_t(s_t) \\ & w_t \sim \mathcal{N}(0, \Sigma) \\ & s_0 \sim \mu_0 \end{aligned}$$

It might seem like we're oversimplifying, but in fact, like we mentioned above, one way to think of these simplifications is that we're just taking the best linear approximation to some general f and c. This is part of the reason why LQR is so well-studied. In fact, humans might even design systems to be linear in order to use results from LQR!

Of course, this only works if the state transitions and cost functions we're approximating are *roughly* smooth or quadratic respectively. For more complex, nonlinear systems, this basic design will break down. But later on, we'll see that we can generalize these ideas to get surprisingly good solutions.

Example 3.2.1: Driving down a road

Suppose we're driving down a road. At each time step, we can choose an action a_t: either we accelerate and apply a force forward ($a_t > 0$), or reverse and apply a force backward ($a_t < 0$). Suppose we can choose an action every δ seconds, and that our car has mass m.

Recall that Newtonian mechanics says that force = mass $\times$ acceleration. We can write the acceleration as the change in velocity over time, and write the velocity as the change in position over time:

$$\text{acceleration}_t = \frac{v_t - v_{t-1}}{\delta}$$
$$v_t = \frac{p_t - p_{t-1}}{\delta}$$

How should we construct our state? We want to express everything in terms of these linear dynamics, and we also want our state to be Markov, so that we can apply dynamic programming like before. Then if we write our state as consisting of the position and velocity, then we can write

$$p_{t+1} = p_t + \delta v_t$$
$$v_{t+1} = v_t + \frac{\delta}{m} a_t$$

Writing everything out in matrix notation, we get:

$$s_{t+1} = \begin{bmatrix} 1 & \delta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\delta}{m} \end{bmatrix} a_t$$

Let's derive a more compressed form for the state at time t as a summation over past time steps. Note that

$$\begin{aligned} s_t &= As_{t-1} + Ba_{t-1} + w_{t-1} \\ &= A(As_{t-2} + Ba_{t-2} + w_{t-2}) + Ba_{t-1} + w_{t-1} \\ &= \cdots \\ &= A^t s_0 + \sum_{i=0}^{t-1} A^i (Ba_{t-i-1} + w_{t-i-1}) \end{aligned}$$

Let's consider the expected value of the state at this time. Since we assume that $\mathbb{E}\, w_t = 0$ (this is the zero vector in d dimensions), by linearity of expectation, the w_t term vanishes, and so we're left with

$$\mathbb{E}[s_t \mid s_{0:t-1}, a_{0:t-1}] = A^t s_0 + \sum_{i=0}^{t-1} A^i B a_{t-i-1}.$$

So now we have a good overview of the LQR setting. How can we now define an optimal policy in this setting? Recall that we allow our policies to be *time-dependent.*

It turns out that the optimal policy is one that is deterministic and *linear* at each time step! That is,

$$\pi_t^\star(s_t) = -K_t s_t.$$

We'll prove this more formally in Theorem 3.3.2. This should remind you somewhat of the way in which the optimal policy in the previous MDP setting was stationary and deterministic. In both cases, it turns out that the optimal policy has special structure!

Note that the average state at time t for the optimal policy is then

$$\mathbb{E}[s_t \mid s_0, a_t = -K_t s_t] = \left(\prod_{i=0}^{t-1}(A - BK_i)\right) s_0.$$

This introdces the quantity $A - BK_i$, which will show up frequently in discussions of LQR! For example, one important question is: will s_t remain bounded, or will it go to infinity as time goes on? We can answer this by analyzing this quantity $A - BK_i$, in particular its largest eigenvalue. Intuitively, if we imagine that these K_is are equal (call this matrix K), then this expression looks like $(A - BK)^t s_0$. Now consider the maximum eigenvalue of $A - BK$, which we denote as $\lambda_{\max}$. If $\lambda_{\max} > 1$, then there's some initial state $s_0^\star$ for which

$$(A - BK)^t s_0^\star = \lambda_{\max}^t s_0^\star \xrightarrow{t\to\infty} \infty.$$

Definition 3.2.2: Value functions for LQR

Given a policy $\pi = (\pi_0, \ldots, \pi_{t-1})$, we can define the value function $V_t^\pi : \mathcal{S} \to \mathbb{R}$ as

$$\begin{aligned} V_t^\pi(s) &= \mathbb{E}\left[\sum_{i=t}^{T} c(s_i, a_i)\right] \\ &= \mathbb{E}\left[\left(\sum_{i=t}^{T-1} s_i^\top Q s_i + a_i^\top R a_i\right) + s_T^\top Q s_T\right] \\ \text{where} \quad s_t &= s \\ a_i &= \pi_i(s_i) \quad \forall i \geq t. \end{aligned}$$

We call this expression inside the equation the **cost-to-go**, since it's just the total cost starting from timestep t.

Similarly, the Q function just additionally conditions on the first action we take:

$$\begin{aligned} Q_t^\pi(s, a) &= \mathbb{E}\left[\sum_{i=t}^{T} c(s_i, a_i)\right] \\ &= \mathbb{E}\left[\left(\sum_{i=t}^{T-1} s_i^\top Q s_i + a_i^\top R a_i\right) + s_T^\top Q s_T\right] \\ \text{where} \quad (s_t, a_t) &= (s, a) \\ a_i &= \pi_i(s_i) \quad \forall i > t \end{aligned}$$

As it turns out, we can now solve for the optimal policy π via dynamic programming in terms of these value and action-value functions.

3.3 Optimality for LQR

Definition 3.3.1: Optimal value functions for LQR

The optimal value function is the one that, in all states and across all timesteps, achieves *lowest cost* across all policies:

$$\begin{aligned} V_t^\star(s) &= \min_\pi V_t^\pi(s) \\ &= \min_{\pi_{t:T-1}} \mathbb{E}\left[\left(\sum_{i=t}^{T-1} s_t^\top Q s_t + a_t^\top R a_t\right) + s_T^\top Q s_T\right] \\ \text{where} \quad a_i &= \pi_i(s_i) \quad \forall i \geq t \\ s_t &= s \end{aligned}$$

Additionally, we'll show theorems 3.3.1 and 3.3.2 below, showing that the $V_t^\star$ is quadratic and that $\pi_t^\star$ is linear. Then, we'll show how to calculate the actual coefficients that specify these functions.

Theorem 3.3.1: $V_t^\star$ in LQR is a quadratic function

Formally, we claim that

$$V_t^\star(s) = s^\top P_t s + p_t$$

for some $P_t \in \mathbb{R}^{d\times d}$ and $p_t \in \mathbb{R}^d$ where P_t is positive-definite. Note that this doesn't have a linear term, just a quadratic term plus a constant.

Theorem 3.3.2: Optimal policy in LQR is linear

That is,

$$\pi_t^\star(s) = -K_t s$$

for some $K_t \in \mathbb{R}^{k\times d}$. (The negative is just there by convention.)

We'll derive these theorems by induction, starting from the last timestep and working backwards in time. Note that induction has a very fundamental connection with dynamic programming: our inductive proof will naturally lend itself to a DP algorithm that allows us to calculate the optimal value and policy!

Base case: $V_T^\star(s)$ is quadratic.

Inductive hypothesis: Show that if $V_{t+1}^\star(s)$ is quadratic, then:

1. $Q_t^\star(s,a)$ is quadratic (in both s and a)

2. Derive the optimal policy $\pi_t^\star(s) = \arg\min_a Q_t^\star(s,a)$, and show that it's linear.
3. Show $V_t^\star(s)$ is quadratic.

Finally, this will have shown that $V_t^\star(s)$ is quadratic and $\pi_t^\star(s)$ is linear.

This is essentially the same proof as a finite-horizon MDP, except that now the state and action are *continuous* instead of finite.

Base case. Let's start by considering the final timestep V_T^π, for some policy π. Then the only expression is

$$V_T^\star(s) = s^\top Q s,$$

which is quadratic, as we desired. Pattern-matching to the expression from earlier, we see that $P_T = Q$ and $p_t = 0$.

Inductive step. Assume $V_{t+1}^\star(s) = s^\top P_{t+1} s + p_{t+1}$ for all states s. We'll start off by demonstrating that $Q_t^\star(s)$ is quadratic. Recall that the definition of $Q_t^\star : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is

$$Q_t^\star(s,a) = c(s,a) + \mathop{\mathbb{E}}_{s' \sim f(s,a,w_{t+1})} V_{t+1}^\star(s').$$

We know $c(s,a) := s^\top Q s + a^\top R a$. Let's consider the average value over the next timestep. The only randomness in the dynamics comes from the noise w_{t+1}, so we can write out this expected value as:

$$\begin{aligned}\mathop{\mathbb{E}}_{s' \sim f(s,a,w_{t+1})} V_{t+1}^\star(s') &= \mathop{\mathbb{E}}_{w_{t+1} \sim \mathcal{N}(0,\sigma^2 I)} V_{t+1}^\star(As + Ba + w_{t+1}) \\ &= \mathop{\mathbb{E}}_{w_{t+1}} [(As + Ba + w_{t+1})^\top P_{t+1}(As + Ba + w_{t+1}) + p_{t+1}].\end{aligned}$$

Summing these two expressions and combining like terms, we get

$$\begin{aligned}Q_t^\star(s,a) &= s^\top Q s + a^\top R a + \mathop{\mathbb{E}}_{w_{t+1}} [(As + Ba + w_{t+1})^\top P_{t+1}(As + Ba + w_{t+1}) + p_{t+1}] \\ &= s^\top (Q + A^\top P_{t+1} A) s + a^\top (R + B^\top P_{t+1} B) a + 2 s^\top A^\top P_{t+1} B a + p_{t+1} \\ &\quad \mathop{\mathbb{E}}_{w_{t+1}} w_{t+1}^\top P_{t+1} w_{t+1}.\end{aligned}$$

Now consider this last term. By writing out the product and using linearity of expectation, we can write this out as

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

When dealing with these quadratic forms, it's often helpful to consider the terms on the diagonal separately from those off the diagonal. On the diagonal, the expectation becomes $\mathbb{E}(w_{t+1})_i^2 = \mathrm{Var}\left((w_{t+1})_i\right) = \sigma^2$. Off the diagonal, since the elements of w_{t+1} are independent, the expectation factors into $\mathbb{E}(w_{t+1})_i \, \mathbb{E}(w_{t+1})_j = 0$. Thus, the only terms left are the ones on the diagonal, so the sum of these can be expressed as the trace of $\sigma^2 P_{t+1}$:

$$\mathop{\mathbb{E}}_{w_{t+1}} w_{t+1}^\top P_{t+1} w_{t+1} = \mathrm{Tr}(\sigma^2 P_{t+1}).$$

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

Theorem 3.3.3: Optimal Q-Function in LQR

$Q_t^\star(s,a) = s^\top(Q+A^\top P_{t+1}A)s+a^\top(R+B^\top P_{t+1}B)a+2s^\top A^\top P_{t+1}Ba+\mathrm{Tr}(\sigma^2 P_{t+1})+p_{t+1}.$

As we'd hoped, this expression is quadratic in s and a! (Phew!)

Now let's move on to the next part of the next part of proving the inductive hypothesis: showing that $\pi_t^\star(s) = \arg\min_a Q_t^\star(s,a)$ is linear. This becomes easy if $Q_t^\star$ is convex w.r.t. $a\ldots$ Which it is!

Theorem 3.3.4: $Q_t^\star$ is convex in a

Consider the part of Theorem 3.3.3 that is quadratic in a, namely $a^\top(R+B^\top P_{t+1}B)a$. Then $Q_t^\star$ is convex w.r.t. a if $R + B^\top P_{t+1}B$ is positive definite.

To show this, recall that in our definition of LQR, we assumed that R is positive definite (see Definition 3.2.1). Also note that $B^\top P_{t+1}B$ is symmetric, and therefore positive definite. Since the sum of two positive-definite matrices is also positive-definite, we have that $R + B^\top P_{t+1}B$ is positive-definite, and so $Q_t^\star$ is convex w.r.t. a.

This means that finding the minimum is easy: we can just take the gradient w.r.t. a and set it to zero! First, we calculate the gradient:

$$\begin{aligned}\nabla_a Q_t^\star(s,a) &= \nabla_a[a^\top(R + B^\top P_{t+1}B)a + 2s^\top A^\top P_{t+1}Ba] \\ &= 2(R + B^\top P_{t+1}B)a + (2s^\top A^\top P_{t+1}B)^\top\end{aligned}$$

Setting this to zero, we get

$$\begin{aligned}0 &= (R + B^\top P_{t+1}B)a + B^\top P_{t+1}As \\ \pi_t^\star(s) := a &= -(R + B^\top P_{t+1}B)^{-1}B^\top P_{t+1}As \\ &= -K_t s, \end{aligned} \tag{3.3}$$

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

We're now almost there! To complete our inductive proof, we must show that the inductive hypothesis is true at time t; that is, we must prove that $V_t^\star(s)$ is quadratic. Using the identity $V_t^\star(s) = Q_t^\star(s,\pi^\star(s))$, we have:

$$\begin{aligned}V_t^\star(s) &= Q_t^\star(s,\pi^\star(s)) \\ &= s^\top(Q + A^\top P_{t+1}A)s + (-K_t s)^\top(R + B^\top P_{t+1}B)(-K_t s) + 2s^\top A^\top P_{t+1}B(-K_t s) \\ &\quad + \mathrm{Tr}(\sigma^2 P_{t+1}) + p_{t+1}\end{aligned}$$

Note that w.r.t. s, this is the sum of a quadratic term and a constant, which is exactly what we were aiming for! The constant term is clearly $p_t = \mathrm{Tr}(\sigma^2 P_{t+1}) + p_{t+1}$. We can simplify the quadratic term by substituting in K_t. Notice that when we do this, the $(R + B^\top P_{t+1}B)$ term in the expression is cancelled out by its inverse, and the remaining terms combine to give what is known as the *Ricatti equation:*

Theorem 3.3.5: Ricatti equation

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

There are several nice things to note about this expression:

1. It's defined recursively; Given P_T, A, B, and the state coefficients Q, we can recursively calculate all values of P_t across timesteps.
2. It appears frequently in calculations surrounding optimality, such as in $V^\star$ and $Q^\star$.
3. Together with A, B, and the action coefficients R, it fully defines the optimal policy.

The optimal policy also has some interesting properties: in addition to being independent of the starting distribution μ_0, which also happened for our finite-horizon MDP solution, it's fully deterministic and doesn't depend on any noise! (Compare this with the discrete MDP case, where calculating our optimal policy required taking an expectation over the state transitions.)

3.4 Time-dependent dynamics

We've closely studied the standard case of LQR, where the state and action spaces are both continuous, the dynamics are linear and time-homogeneous, and the cost is quadratic. We can also consider situations where some, or all, of these assumptions are relaxed.

So far, we've considered the *time-homogeneous* case, where the dynamics are the same at every timestep. We can also loosen this restriction, and consider the case where the dynamics are *time-dependent*. Our analysis remains almost the same, except we just add a time index to each of the relevant matrices.

The problem is now defined as follows:

$$\begin{aligned}
&\arg \min_{\pi_{0:T-1}: \mathcal{S} \to \mathcal{A}} \quad \mathbb{E}\left[\left(\sum_{t=0}^{T-1} (s_t^\top Q_t s_t) + a_t^\top R_t a_t\right) + s_T^\top Q_T s_T\right] \\
\text{where} \quad & s_{t+1} = f_t(s_t, a_t, w_t) = A_t s_t + B_t a_t + w_t \\
& s_0 \sum \mu_0 \\
& a_t = \pi_t(s_t) \\
& w_t \sim \mathcal{N}(0, \sigma^2 I).
\end{aligned}$$

The derivation remains almost exactly the same, and so the Ricatti equation then becomes the *time-dependent Ricatti equation:*

Theorem 3.4.1: Time-dependent Ricatti Equation

$$P_t = Q_t + A_t^\top P_{t+1} A_t - A_t^\top P_{t+1} B_t (R_t + B_t^\top P_{t+1} B_t)^{-1} B_t^\top P_{t+1} A_t.$$

Note that this is just Theorem 3.3.5, but with the time index added to each of the matrices.

Additionally, by allowing the dynamics to vary across time, this gives us a *locally linear* approximation of nonlinear dynamics at each timestep! We'll discuss this later in the chapter.

3.5 More general quadratic cost functions

Our original cost function had only second-order terms w.r.t. 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, we get the following expression

$$c_t(s_t, a_t) = (s_t^\top Q_t s_t + s_t^\top M_t a_t + a_t^\top R_t a_t) + (s_t^\top q_t + a_t^\top r_t) + c_t.$$

We can also include a constant term $v_t \in \mathbb{R}^d$ in the dynamics:

$$s_{t+1} = f_t(s_t, a_t, w_t) = A_t s_t + B_t a_t + v_t + w_t.$$

3.6 Tracking a predefined trajectory

So far, we've been trying to get the robot to stay as close as possible to a single point $(s^\star, a^\star)$. We can also consider the case where we want the robot to follow a predefined trajectory, which is a sequence of states and actions $(s_t^\star, a_t^\star)_{t=1}^T$. In this case, we can use the following cost function:

$$c_t(s_t, a_t) = (s_t - s_t^\star)^\top Q (s_t - s_t^\star) + (a_t - a_t^\star)^\top R (a_t - a_t^\star).$$

By expanding out these multiplication, we can see that this is actually a special case of the more general quadratic cost function we discussed above:

$$M_t = 0, \qquad q_t = -2Q s_t^\star, \qquad r_t = -2R a_t^\star, \qquad c_t = (s_t^\star)^\top Q (s_t^\star) + (a_t^\star)^\top R (a_t^\star).$$

3.7 Infinite-horizon

So far, we've been considering finite-horizon problems, where we have a fixed number of timesteps T. We can also consider the case where we have an infinite horizon, and so we want to minimize the expected cost over all future timesteps. Additionally, we'll compare this *undiscounted finite-horizon* case to the *discounted infinite-horizon* case, where we have a discount factor $\gamma \in [0, 1]$ that discounts future rewards.

Instead of considering the *total* expected cost, which is going to diverge as time goes on, we'll consider the average cost per timestep. That is, we'll divide our objective function by T, the number of timesteps. What happens is that the recursion in our Ricatti equation (the recursive equation for P_t) converges to a fixed value P.

From an intuitive perspective of 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. But as the deadline gets closer and closer, suddenly the horizon becomes clear, and you'll spend more time thinking about it. Now the infinite-horizon case is just where the deadline is infinitely far away, so you can just "behave" like normal!

This ties in exactly with our analysis of value iteration, which gave us a sequence of policies, and then we just took the final policy. We're doing the same thing in the large-horizon setting, but note that here there's no discount term γ; the structure of the problem allows us to analyze it even without one.

Note that several parts of this proof are very similar to our derivations of value iteration and policy iteration in the previous chapter. Let's spend some time on formalizing this connection.

In the discounted case, instead of considering the limit as $T \to \infty$, we consider the limit as $\gamma \to 1$. This is because as the discount factor γ approaches 1, time discounting becomes less and less important, and we're left with the undiscounted case. Just like above, this means we need to multiply our cost function by $1-\gamma$, and then we can use the same analysis as before.

Let's consider value iteration, which uses the Bellman operator to update V:

$$V_{t+1}(s) = \max_a \left(r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(s,a)} V_t(s') \right).$$

The analogue of this in the undiscounted finite-horizon case is the *Ricatti equation* (Theorem 3.3.5). Instead of thinking of P_{t+1} as defining the value function for the *next timestep,* though, let's think of it as the *next version* of the value function.

Control	**MDP**
Finite-horizon	Infinite-horizon
Continuous states and actions	Finite states and actions
Total undiscounted cost	Total discounted reward
$\lim_{T\to\infty} \frac{1}{T} \mathbb{E}\left[\left(\sum_{t=0}^{T-1} c(s_t, a_t) \right) + c(s_T) \right]$	$\lim_{\gamma\to 1}(1-\gamma)\, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) \right]$
Ricatti equations	Value iteration
$P \leftarrow Q + A^\top P A - A^\top P B (R + B^\top P B)^{-1} B^\top P A$	$V(s) \leftarrow \max_a \left[r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(s,a)} V(s') \right]$

Figure 3.1: Comparison of control and finite MDPs.

3.8 Approximating nonlinear dynamics with LQR

Let's return to the CartPole example (3.1.1) from the start of the chapter. We want to stabilize the pole around some optimal state and action $(s^\star, a^\star)$. The 'Q' in LQR stands for

"quadratic cost," so previously we've modelled the cost as $c(s,a) = (s-s^\star)^\top Q(s-s^\star) + (a-a^\star)^\top R(a-a^\star)$. Here, let's relax those constraints and consider some more general measure of distance to the optimal state and action:

$$c(s,a) = d(s,s^\star) + d(a,a^\star).$$

We'll consider the noise-free setting, since as we previously saw, the noise doesn't actually affect the optimal policy.

This is essentially just a general control problem (see 3.1.1). We can use LQR to solve it, but we'll need to approximate the dynamics of the system. Here, we don't know the dynamics f or the cost function c, but we suppose that we're able to *query/sample/simulate* them to get their values at a given state and action.

We also assume that f is differentiable and that c is twice-differentiable. This makes sense since we want to approximate f as linear and c as quadratic so that we can apply LQR. For our approximation to be accurate, we need to make sure that all states are close to the optimal state $s^\star$, and we can stay close using actions that are close to $a^\star$.