CS/Stat 184
Introduction to Reinforcement Learning

Initially created by **Alexander D. Cai** during the first iteration of the course in Fall 2022.

Contents

This book expects some knowledge of linear algebra and multivariable calculus. Students should be familiar with the following concepts:

- **Linear Algebra:** Vectors, matrices, matrix multiplication, matrix inversion, eigenvalues and eigenvectors, and the Gram-Schmidt process.
- **Multivariable Calculus:** Partial derivatives, gradient, directional derivative, and the chain rule.

Chapter 1

Bandits

1.1 Multi-Armed Bandits

Chapter 2

Markov Decision Processes

For now, we'll assume that the world is known. This involves the state transitions and the reward.

Unknown systems are similar to complex systems. In both, once we don't access the world everywhere, we need to actually *learn* about the world around us.

2.1 Optimality

Theorem 2.1.1: Value Iteration

Initialize:

$$V^0 \sim \|V^0\|_\infty \in [0, 1/1-\gamma]$$

Iterate until convergence:

$$V^{t+1} \leftarrow \mathcal{J}(V^t)$$

Analysis

This algorithm runs in $O(|\mathcal{S}|^3)$ time since we need to perform a matrix inversion.

Theorem 2.1.2: Exact Policy Evaluation

Represent the reward from each state-action pair as a vector

$$R^\pi \in \mathbb{R}^{|\mathcal{S}|} \qquad R^\pi_s = r(s, \pi(s))$$

Also represent the state transitions

$$P^\pi \in \mathbb{R}^{|\mathcal{S}\times\mathcal{S}} \qquad P^\pi_{s,s'} = P(s'|s,\pi(s))$$

That is, row i of P^π is a distribution over the *next state* given that the current state is s_i and we choose an action using policy π.

Using this notation, we can express the Bellman consistency equation as

$$\begin{pmatrix} \vdots \\ V^\pi(s) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ r(s,\pi(s)) \\ \vdots \end{pmatrix} + \gamma \begin{pmatrix} \vdots \\ P(s' \mid s,\pi(s)) \\ \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ V^\pi(s') \\ \vdots \end{pmatrix}$$

$$\begin{aligned} V^\pi &= R^\pi + \gamma P^\pi V^\pi \\ (I - \gamma P^\pi)V^\pi &= R^\pi \\ V^\pi &= (I - \gamma P^\pi)R^\pi \end{aligned}$$

if $I - \gamma P^\pi$ is invertible, which we can prove is the case.

Theorem 2.1.3: Iterative Policy Evaluation

How can we calculate the value function V^π of a policy π?

Above, we saw an exact function that runs in $O(|\mathcal{S}|^2)$. But say we really need a fast algorithm, and we're okay with having an approximate answer. Can we do better? Yes!

Using the same notation as above, let's initialize V^0 such that the elements are drawn uniformly from $[0, 1/(1-\gamma)]$.

Then we can iterate the fixed-point equation we found above:

$$V^{t+1} \leftarrow R + \gamma P V^t$$

How can we use this fast approximate algorithm?

Theorem 2.1.4: Policy Iteration

Remember, for now we're only considering policies that are *stationary and deterministic*. There's $|\mathcal{S}|^{|\mathcal{A}}$ of these, so let's start off by choosing one at random. Let's call this initial policy π^0, using the superscript to indicate the time step.

Now for $t = 0, 1, \ldots,$ we perform the following:

1. *Policy Evaluation*: First use the algorithm from earlier to calculate $V^{\pi^t}(s)$ for

all states s. Then use this to calculate the state-action values:

$$Q^{\pi^t}(s,a) = r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^{\pi^t}(s')$$

2. *Policy Improvement*: Update the policy so that, at each state, it chooses the action with the highest action-value:

$$\pi^{t+1}(s) = \arg\max_a Q^{\pi^t}(s,a)$$

In other words, we're setting it to act greedily with respect to the new Q-function.

What's the computational complexity of this?

2.2 Finite Horizon MDPs

Suppose we're only able to act for H timesteps.

Now, instead of discounting, all we care about is the (average) total reward that we get over this time.

$$\mathbb{E}[\sum_{t=0}^{H-1} r(s_t, a_t)]$$

To be more precise, we'll consider policies that depend on the time. We'll denote the policy at timestep h as $\pi_h : \mathcal{S} \to \mathcal{A}$. In other words, we're dropping the constraint that policies must be stationary.

This is also called an *episodic model*.

Note that since our policy is nonstationary, we also need to adjust our value function (and Q-function) to account for this. Instead of considering the total infinite-horizon discounted reward like we did earlier, we'll instead consider the *remaining* reward from a given timestep onwards:

$$V_h^\pi(s) = \mathbb{E}\left[\sum_\tau^{H-1} r(s_\tau, a_\tau) \mid s_h = s, a_\tau = \pi_h(s_h)\right]$$
$$Q_h^\pi(s,a) = \mathbb{E}\left[\sum_\tau^{H-1} r(s_\tau, a_\tau) \mid (s_h, a_h) = (s,a)\right]$$

We can also define our Bellman consistency equations, by splitting up the total reward into the immediate reward (at this time step) and the future reward, represented by our state value function from that next time step:

$$Q_h^\pi(s,a) = r(s,a) + \mathop{\mathbb{E}}_{s' \sim P(s,a)}[V_{h+1}^\pi(s')]$$

Theorem 2.2.1: Computing the optimal policy

We can solve for the optimal policy using dynamic programming.

- *Base case.* At the end of the episode (time step $H-1$), we can't take any more actions, so the Q-function is simply the reward that we obtain:

  $$Q_{H-1}^\star(s,a) = r(s,a)$$

  so the best thing to do is just act greedily and get as much reward as we can!

  $$\pi_{H-1}^\star(s) = \arg\max_a Q_{H-1}^\star(s,a)$$

  Then $V_{H-1}^\star(s)$, the optimal value of state s at the end of the trajectory, is simply whatever action gives the most reward.

  $$V_{H-1}^\star = \max_a Q_{H-1}^\star(s,a)$$

- *Recursion.* Then, we can work backwards in time, starting from the end, using our consistency equations!

Note that this is exactly just value iteration and policy iteration combined, since our policy is nonstationary, so we can exactly specify its decisions at each time step!

Analysis

Total computation time $O(H|\mathcal{S}|^2|\mathcal{A}|)$

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 cases we studied the previous chapter, the state and action spaces are not finite or even discrete. Instead, they are continuous and uncountably infinite. In addition, the state transitions governing the system – that is, the laws of physics – are nonlinear and complex. Swapping out a pen in your palm for a pole on a cart, we have the following classic *control problem:*

Example 3.1.1: CartPole

Consider a pole balanced on a cart. The state s 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 a. [a]

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

[a] *Controls* are the continuous analogue to *actions* in the discrete setting. In control theory, the state and controls are typically denoted as x and u, but we'll stick with the s and a notation to highlight the similarity with the discrete case.

Beyond this simple scenario, there are many real-world examples that involve continuous control:

- **Robotics.** Autonomous driving; Controlling a drone's position; Automation in warehouses and manufacturing; Humanoid robots with joints.

- **Temperature.** Controlling the temperature in a room; Keeping greenhouses warm; Understanding weather patterns.
- **Games.** Sports; MMORPGs (Massively Multiplayer Online Role-Playing Games); Board games.
- **Finance.** Stock trading; Portfolio management; Risk management.

How can we teach computers to solve these kinds of problems?

In the last chapter, we developed efficient dynamic programming algorithms (*value iteration* and *policy iteration*) for calculating $V^\star$ and $\pi^\star$ in the finite setting. In this chapter, we'll derive similar results in the continuous case by imposing some additional structure on the problem.

Note that we're still assuming that the entire environment is *known* – that is, we understand 'how the world works'. (Hang tight – we'll get to the unknown case soon!)

3.2 Optimal control

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

- The state and action spaces are *continuous* rather than finite. That is, $\mathcal{S} = \mathbb{R}^{n_s}$ and $\mathcal{A} = \mathbb{R}^{n_a}$, where n_s and n_a 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_t at each timestep from some distribution p_{w_t}. We denote these dynamics as the function $s_{t+1} = f_t(s_t, a_t, w_t)$. Of course, we can simplify to cases where the dynamics are *deterministic/noise-free* (no w_t term) or are *stationary/time-homogenous* (just f without the t subscript).
- Instead of a reward function, it's more intuitive to consider a **cost function** $c_t : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that describes *how far away* we are from our goal state/action. Like the dynamics, a special case is when this function is time-homogenous (and we drop the t subscript).
- We consider the *undiscounted* case with a *time horizon* T. Note that we end an episode at s_T – there is no a_T, and so we denote the cost for the final state as $c_T(s_T)$.

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

Definition 3.2.1: Optimal control problem

$$\begin{aligned} \min_{\pi_0,\ldots,\pi_{T-1}:\mathcal{S}\to\mathcal{A}} \quad & \mathbb{E}\left[\left(\sum_{t=0}^{T-1} c_t(s_t,a_t)\right) + c_T(s_T)\right] \\ \text{where} \quad & s_{t+1} = f_t(s_t,a_t,w_t), \\ & a_t = \pi_t(s_t) \\ & s_0 \sim \mu_0 \\ & w_t \sim p_{w_t} \end{aligned} \tag{3.1}$$

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 DP algorithms we saw before. As a matter of fact, let's consider what happens if we *discretize* the problem. For intuition, suppose $n_s = n_a = 1$ (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're just rounding s and a to two decimal spaces. [1]

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, the existing algorithms might take unfeasibly long to complete. 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 using our rounding method, we must divide each dimension into intervals of length ε, resulting in $(B_s/\varepsilon)^{n_s}$ and $(B_a/\varepsilon)^{n_a}$ 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 of size $|\mathcal{S}|^2|\mathcal{A}|$ is on the order of $(100^{10})^2(100^{10}) = 10^{60}$! Try finding a computer that'll fit that in memory! (For reference, 32 GB of memory can store 10^9 32-bit floating point numbers!) 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?

Note that in order for discretization to yield and accurate solution, we implicitly relied assume that rounding each 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 topic of **Linear Quadratic Regulators.**

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, and the state transitions might be very complex, making it difficult to satisfy the constraints. Is there a relevant simplification that we can analyze?

[1] Formally, we can consider an ϵ-net over the original continuous space. Let V be some normed space. A subset $V_\epsilon \subseteq V$ is called an ϵ-net if for all $v \in V$, there exists a $v_\epsilon \in V_\epsilon$ such that $\|v - v_\epsilon\| \leq \epsilon$. The rounding example given is technically a 0.005-net.

A natural structure to impose is *linear dynamics* and a *quadratic cost function* (in both arguments). Why is this a useful assumption to make? Note that given any continuous function, we can locally approximate it using a *Taylor approximation.* The simplest non-trivial case for the dynamics is a first-order linear approximation, whereas for the cost function it must be quadratic, since we want it to have an optimum. This means that even given *nonlinear* dynamics and a *nonlinear* cost function, as long as they're continuous, we can write out their first and second order approximations respectively and use the results that we're about to derive!

These assumptions result in the extremely popular **linear quadratic regulator** model. In fact, some people even design systems to be linear in order to use results from LQR! We'll see later on that even for more complex setups, we can generalize these simple ideas to get surprisingly good solutions.

Definition 3.3.1: The linear quadratic regulator

Linear dynamics (time-homogeneous):

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

Quadratic cost function (time-homogeneous): [a]

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

We want c to be a convex function, so Q and R must both be positive definite.

Isotropic Gaussian noise:

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

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

$$\begin{aligned} \min_{\pi_0,\ldots,\pi_{T-1}:\mathcal{S}\to\mathcal{A}} \quad & \mathbb{E}\left[\left(\sum_{t=0}^{T-1} s_t^\top Q s_t + a_t^\top R a_t\right) + s_T^\top Q s_T\right] \\ \text{where} \quad & s_{t+1} = As_t + Ba_t + w_t \\ & a_t = \pi_t(s_t) \\ & w_t \sim \mathcal{N}(0, \sigma^2 I) \\ & s_0 \sim \mu_0 \end{aligned}$$

[a] For some intuition into this expression, consider the simple case where a_t and s_t are scalars (and so are Q and R), so $c(s_t, a_t) = Qs_t^2 + Ra_t^2$. If this notation is unfamiliar to you, we recommend this tutorial from Khan Academy!

Example 3.3.1: 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:

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

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

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

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 define an optimal time-dependent policy in this setting?

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 3.4.2. 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 3.3.2: 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.

3.4 Optimality for LQR

Definition 3.4.1: 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 3.4.1 and 3.4.2 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 3.4.1: $V_t^\star$ 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 3.4.2: 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

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

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 3.4.3: Optimal Q-Function in LQR

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

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 3.4.4: $Q_t^\star$ is convex in a

Consider the part of Theorem 3.4.3 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 3.3.1). 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:

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

Setting this to zero, we get

$$\begin{aligned} 0 &= (R + B^\top P_{t+1} B)a + B^\top P_{t+1} A s \\ \pi_t^\star(s) := a &= -(R + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A s \\ &= -K_t s, \end{aligned} \tag{3.2}$$

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 3.4.5: 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.)

3.5 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 3.5.1: 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 3.4.5, 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.

3.6 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:

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

3.7 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:

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

3.8 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 3.4.5). 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
$P \leftarrow Q + A^\top PA - A^\top PB(R + B^\top PB)^{-1}B^\top PA$	$V(s) \leftarrow \max_a \left[r(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim P(s,a)} V(s')\right]$

Figure 3.1: Comparison of control and finite MDPs.

3.9 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 3.2.1). 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 $a^\star$.