Chapter 6

Linear Quadratic Regulator (LQR)

Reading

1. http://underactuated.csail.mit.edu/lqr.html, Lecture 3-4 at https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-323-principles-of-optimal-control-spring-2008/lecture-notes
2. Optional: Applied Optimal Control by Bryson & Ho, Chapter 4-5

This chapter is the analogue of Chapter 3 on Kalman filtering. Just like Chapter 2, the previous chapter gave us two algorithms, namely value iteration and policy iteration, to solve dynamic programming problems for a finite number of states and a finite number of controls. Solving dynamic programming problems is difficult if the state/control space are infinite. In this chapter, we will look at an important and powerful special case, called the Linear Quadratic Regulator (LQR), when we can solve dynamic programming problems easily. Just like a lot of real-world state-estimation problems can be solved using the Kalman filter and its variants, a lot of real-world control problems can be solved using LQR and its variants.

6.1 Discrete-time LQR

Consider a deterministic, *linear* dynamical system given by

$$x_{k+1} = Ax_k + Bu_k; \quad x_0 \text{ is given.}$$

where $x_k \in \mathbb{R}^d$ and $u_k \in \mathbb{R}^m$ which implies that $A \in \mathbb{R}^{d\times d}$ and $B \in \mathbb{R}^{d\times m}$. In this chapter, we are interested in calculating a feedback control $u_k = u(x_k)$ for such a system. Just like we formulated the problem in dynamic programming, we want to pick a feedback control which leads to a trajectory that achieves a minimum of some run-time cost and a terminal cost. We will assume that both the run-time and terminal costs are *quadratic* in the state and control input, i.e.,

$$q(x, u) = \frac{1}{2} x^\top Q x + \frac{1}{2} u^\top R u \tag{6.1}$$

where $Q \in \mathbb{R}^{d \times d}$ and $R \in \mathbb{R}^{m \times m}$ are symmetric, positive semi-definite matrices

$$Q = Q^\top \succeq 0, \quad R = R^\top \succeq 0.$$

Effectively, if Q were a diagonal matrix, a large diagonal entry would Q_{ii} models our desire that the trajectory of the system should not have a large value of the state x_i along its trajectories. We want these matrices to be positive semi-definitive to prevent dynamic programming from picking a trajectory which drives down the run-time cost to negative infinity by picking.

Example Consider the discrete-time equivalent of the so-called double integrator $\ddot{z}(t) = u(t)$. The linear system in this case (obtained by creating two states $x := [z(t), \dot{z}(t)]$ is

$$x_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k.$$

ℹ This system is called the double integrator because of the structure $\ddot{z} = u$; if z denotes the position of an object the equation is simply Newton's law which connects the force applied u to the acceleration.

First, note that a continuous-time linear dynamical system $\dot{x} = Ax$ is asymptotically stable, i.e., from any initial condition $x(0)$ its trajectories go to the equilibrium point $x = 0$ ($x(t) \to 0$ as $t \to \infty$). Asymptotic stability occurs if all eigenvalues of A are strictly negative. A discrete-time linear dynamical system $x_{k+1} = Ax_k$ is asymptotically stable if all eigenvalues of A have magnitude strictly smaller than 1, $|\lambda(A)| < 1$.

A typical trajectory of the double integrator will look as follows. Suppose

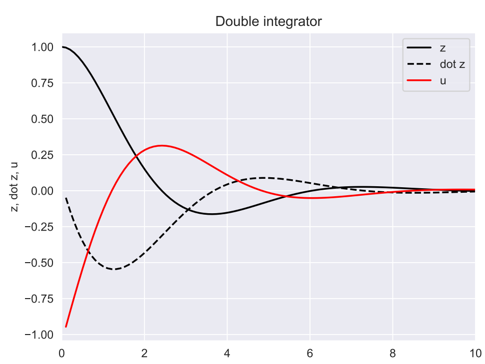

Figure 6.1: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a stabilizing (i.e., one that makes the system asymptotically stable) controller $u = -z(t) - \dot{z}(t)$. Notice how the trajectory starts from some initial condition (in this case $z(0) = 1$ and $\dot{z}(0) = 0$) and moves towards its equilibrium point $z = \dot{z} = 0$.

we would like to pick a different controller that more quickly brings the system to its equilibrium. One way of doing so is to minimize

$$J = \sum_{k=0}^{T} \|x_k\|^2$$

which represents how far away both the position and velocity are from zero over all times k. The following figure shows the trajectory that achieves a small value of J.

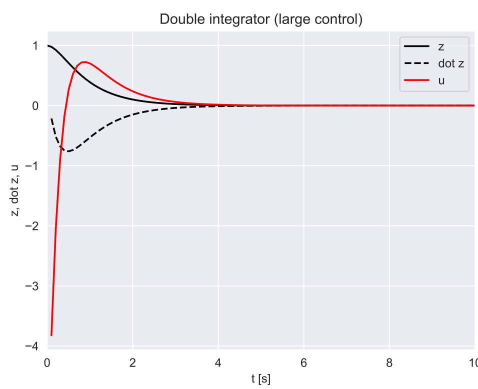

Figure 6.2: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a large stabilizing control at each time $u = -5z(t) - 5\dot{z}(t)$. Notice how quickly the state trajectory converges to the equilibrium without much oscillation as compared to Figure 6.1 but how large the control input is at certain times.

This is obviously undesirable for real systems where we may want the control input to be bounded between some reasonable values (a car cannot accelerate by more than a certain threshold). A natural way of enforcing this is to modify our our desired cost of the trajectory to be

$$J = \sum_{k=0}^{T} \left(\|x_k\|^2 + \rho\|u_k\|^2\right)$$

where the value of the parameter ρ is something chosen by the user to give a good balance of how quickly the trajectory reaches the equilibrium point and how much control is exerted while doing so. Linear-Quadratic-Regulator (LQR) is a generalization of this idea, notice that the above example is equivalent to setting $Q = I_{d\times d}$ and $R = \rho I_{m\times m}$ for the run-time cost in (6.1).

Back to LQR With this background, we are now ready to formulate the Linear-Quadratic-Regulator (LQR) problem which is simply dynamic programming for a linear dynamical system with quadratic run-time cost. In order to enable the system to reach the equilibrium state even if we have only a finite time-horizon, we also include a quadratic cost

$$q_f(x) = \frac{1}{2} x^\top Q_f x. \tag{6.2}$$

The dynamic programming problem is now formulated as follows.

Finite time-horizon LQR problem Find a sequence of control inputs $(u_0, u_1, \dots, u_{T-1})$ such that the function

$$J(x_0; u_0, u_1, \dots, u_{T-1}) = \frac{1}{2} x_T^\top Q_f x_T + \frac{1}{2} \sum_{k=0}^{T-1} \left(x_k^\top Q x_k + u_k^\top R u_k \right) \tag{6.3}$$

is minimized under the constraint that $x_{k+1} = A x_k + B u_k$ for all times $k = 0, \dots, T-1$ and x_0 is given.

6.1.1 Solution of the discrete-time LQR problem

We know the principle of dynamic programming and can apply it to solve the LQR problem. As usual, we will compute the cost-to-go of a trajectory that starts at some state x and goes further by $T-k$ time-steps, $J_k(x)$ backwards. Set

$$J_T^*(x) = \frac{1}{2} x^\top Q_f x \quad \text{for all } x.$$

Using the principle of dynamic programming, the cost-to-go J_{T-1} is given by

$$\begin{aligned} J_{T-1}^*(x_{T-1}) &= \min_u \left\{ \frac{1}{2} \left(x_{T-1}^\top Q x_{T-1} + u^\top R u \right) + J_T^*(A x_{T-1} + B u) \right\} \\ &= \min_u \left\{ \frac{1}{2} \left(x_{T-1}^\top Q x_{T-1} + u^\top R u + (A x + B u)^\top Q_f (A x_{T-1} + B u) \right) \right\}. \end{aligned}$$

We can now take the derivative of the right-hand side with respect to u to get

$$\begin{aligned} 0 &= \frac{\mathrm{dRHS}}{\mathrm{d}u} \\ &= \frac{1}{2} \left\{ R u + B^\top Q_f (A x_{T-1} + B u) \right\} \\ \Rightarrow u_{T-1}^* &= -(R + B^\top Q_f B)^{-1} B^\top Q_f A \, x_{T-1} \\ &\equiv -K_{T-1} \, x_{T-1}. \end{aligned} \tag{6.4}$$

where

$$K_{T-1} = (R + B^\top Q_f B)^{-1} B^\top Q_f A$$

is (surprisingly) also called the Kalman gain. The second derivative is positive semi-definite

$$\frac{\mathrm{d}^2\mathrm{RHS}}{\mathrm{d}u^2} = R + B^\top Q_f B \succeq 0$$

so we know that u_{T-1}^* is a minimum of the convex quantity on the right-hand side. Notice that the optimal control u_{T-1}^* is a linear function of the state x_{T-1}. Let us now expand the cost-to-go J_{T-1} using this optimal value (the subscript $T-1$ on the curly bracket simply means that all quantities are at time $T-1$)

$$\begin{aligned} J^*_{T-1}(x_{T-1}) &= \frac{1}{2}\left\{x^\top Q x + {u^*}^\top R u^* + (Ax+Bu^*)^\top Q_f (Ax+Bu^*)\right\}_{T-1} \\ &= \frac{1}{2} x_{T-1}^\top \left\{Q + K^\top R K + (A-BK)^\top Q_f (A-BK)\right\}_{T-1} x_{T-1} \\ &\equiv \frac{1}{2} x_{T-1}^\top P_{T-1} x_{T-1} \end{aligned}$$

where we set the stuff inside the curly brackets to the matrix P which is also positive semi-definite. This is great, the cost-to-go is also a quadratic function of the state x_{T-1}. Let us assume that this pattern holds for all time steps and the cost-to-go of the optimal LQR trajectory starting from a state x and proceeding forwards for $T-k$ time-steps is

$$J^*_k(x) = \frac{1}{2} x^\top P_k x.$$

We can now repeat the same exercise to get a recursive formula for P_k in terms of P_{k+1}. This is the *solution* of dynamic programming for the LQR problem as looks as follows.

$$\begin{aligned} P_T &= Q_f \\ K_k &= \left(R + B^\top P_{k+1}\, B\right)^{-1}\; B^\top P_{k+1}\, A \\ P_k &= Q + K_k^\top R\, K_k + (A - BK_k)^\top\, P_{k+1}\; (A - BK_k)\,, \end{aligned} \tag{6.5}$$

for $k = T-1, T-2, \ldots, 0$. There are a number of important observations to be made from this calculation:

1. The optimal controller $u^*_k = -K_k x_k$ is a linear function of the state x_k. This is only true for linear dynamical systems with quadratic costs. Notice that both the state and control space are infinite sets but we have managed to solve the dynamic programming problem to get the optimal controller. We could not have done it if the run-time/terminal costs were not quadratic or if the dynamical system were not linear. Can you say why?
2. The cost-to-go matrix P_k and the Kalman gain K_k do not depend upon the state and can be computed ahead of time if we know what the time horizon T is going to be.
3. The Kalman gain changes with time k. Effectively, the LQR controller picks a large control input to quickly reduce the run-time cost at the beginning (if the initial condition were such that the run-time cost of the trajectory would be very large) and then gets into a balancing act where it balances the control effort and the state-dependent part of the run-time cost. LQR is an optimal way to strike a balance between the two examples in Figure 6.1 and Figure 6.2.

The careful reader will notice how the equations in (6.5) and our remarks about them are similar to the update equations of the Kalman filter and our remarks there. In fact we will see shortly how spookily similar the two are. The key difference is that Kalman filter updates run forwards in time and update the covariance while LQR updates run backwards in time and update the cost-to-go matrix P. This is not surprising because LQR is an optimal control problem, its update equations run backward in time.

ℹ If you are trying this example yourself, I used the formula for continuous-time LQR and then discretized the controller while implementing it. We will see this in Section 6.2

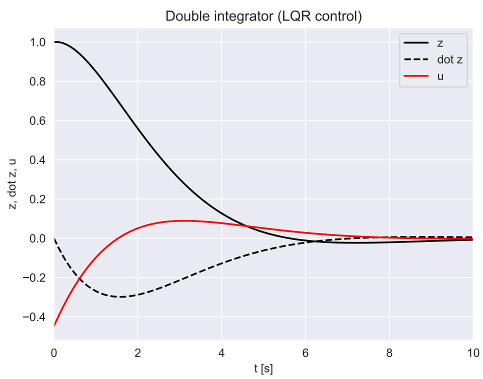

Figure 6.3: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a controller obtained from LQR with $Q = I$ and $R = 5$. This gives the controller to be about $u = -0.45z(t) - 1.05\dot{z}(t)$. Notice how we still get stabilization but the control acts more gradually. Using different values of R, we can get many different behaviors. Another key aspect of LQR as compared to Figure 6.1 where the control was chosen in an ad hoc fashion is to let us prescribe the quality of state trajectories using high-level quantities like Q, R.

6.2 Hamilton-Jacobi-Bellman equation

This section will show how the principle of dynamic programming looks for continuous-time deterministic dynamical systems

$$\dot{x} = f(x, u), \quad \text{with } x(0) = x_0.$$

As we discussed in Chapter 3, we can think of this as the limit of discrete-time dynamical system $x_{k+1} = f^{\text{discrete}}(x_k, u_k)$ as the time discretization goes to zero. Just like we have a sequence of controls in the discrete-time case, we have a continuous curve that determines the control (let us also call it the control sequence)

$$\{u(t) : \ t \in \mathbb{R}_+\}$$

which gives rise to a trajectory of the states

$$\{x(t) : \ t \in \mathbb{R}_+\}$$

for the dynamical system. Let us consider the case when we want to find control sequences that minimize the integral of the cost along the trajectory that stops at some fixed, finite time-horizon T:

$$q_f(x(T)) + \int_0^T q(x(t), u(t)) \, dt.$$

This cost is again a function of the run-time cost and a terminal cost.

ℹ Since $\{x(t)\}_{t \geq 0}$ and $\{u(t)\}_{t \geq 0}$ are continuous curves and the cost is now a function of a continuous-curve, mathematicians say that the cost is a "functional" of the state and control trajectory.

Continuous-time optimal control problem We again want to solve for

$$J^*(x_0) = \min_{u(t),\ t\in[0,T]} \left\{ q_f(x(T)) + \int_0^T q(x(t), u(t))\ \mathrm{d}t \right\} \tag{6.6}$$

with the system satisfying $\dot{x} = f(x, u)$ at each time instant. Notice that the minimization is over a function of time $\{u(t) : t \in [0, T]\}$ as opposed to a discrete-time sequence of controls that we had in the discrete-time case. We will next look at the Hamilton-Jacobi-Bellman equation which is a method to solve optimal-control problems of this kind.

The principle of dynamic programming principle is still valid: if we have an optimal control trajectory $\{u^*(t) : t \in [0, T]\}$ we can chop it up into two parts at some intermediate time $t \in [0, T]$ and claim that the tail is optimal. In preparation for this, let us define the cost-to-go of going forward by $T - t$ time as

$$J^*(x, t) = \min_{u(s),\ s\in[t,T]} \left\{ q_f(x(T)) + \int_t^T q(x(s), u(s))\ \mathrm{d}s \right\},$$

the cost incurred if the trajectory starts at state x and goes forward by $T - t$ time. This is very similar to the cost-to-go $J_k^*(x)$ we had in discrete-time dynamic programming. Dynamic programming now gives

$$\begin{aligned} J^*(x(t), t) &= \min_{u(s),\ t\leq s\leq T} \left\{ q_f(x(T)) + \int_t^T q(x(s), u(s))\ \mathrm{d}s \right\} \\ &= \min_{u(s),\ t\leq s\leq T} \left\{ q_f(x(T)) + \int_t^{t+\Delta t} q(x(s), u(s))\ \mathrm{d}s + \int_{t+\Delta t}^T q(x(s), u(s))\ \mathrm{d}s \right\} \\ &= \min_{u(s),\ t\leq s\leq T} \left\{ J^*(x(t+\Delta t), t+\Delta t) + \int_t^{t+\Delta t} q(x(s), u(s))\ \mathrm{d}s \right\}. \end{aligned}$$

We now take the Taylor approximation of the term $J^*(x(t + \Delta t), t + \Delta t)$ as follows

$$\begin{aligned} &J^*(x(t+\Delta t), t+\Delta t) - J^*(x(t), t) \\ &\approx \partial_x J^*(x(t), t)\ (x(t+\Delta t) - x(t)) + \partial_t J^*(x(t), t)\Delta t \\ &\approx \partial_x J^*(x(t), t)\ f(x(t), u(t))\ \Delta t + \partial_t J^*(x(t), t)\Delta t \end{aligned}$$

where $\partial_x J^*$ and $\partial_t J^*$ denote the derivative of J^* with respect to its first and second argument respectively. We substitute this into the minimization and collect terms of Δt to get

$$0 = \partial_t\ J^*(x(t), t) + \min_{u(t)\in U} \left\{ q(x(t), u(t)) + f(x(t), u(t))\ \partial_x J^*(x(t), t) \right\}. \tag{6.7}$$

Notice that the minimization in (6.7) is only over *one* control input $u(t) \in U$, this is the control that we should take at time t. (6.7) is called the Hamilton-

Jacobi-Bellman (HJB) equation. Just like the Bellman equation

$$J_k^*(x) = \min_{u \in U} \left\{ q_k(x,u) + J_{k+1}^*(f(x,u)) \right\}.$$

has two quantities x and the time k, the Hamilton-Jacobi-Bellman equation also has two quantities x and continuous time t. Just like the Bellman equation is solved backwards in time starting from T with $J_k^*(x) = q_f(x)$, the HJB equation is solved backwards in time by setting

$$J^*(x,T) = q_f(x).$$

> You should think of the HJB equation as the continuous-time, continuous-space analogue of Dijkstra's algorithm when the number of nodes in the graph goes to infinity and the length of each edge is also infinitesimally small.

6.2.1 Infinite-horizon HJB

The infinite-horizon problem with the HJB equation is easy: since we know that the optimal cost-to-go is not a function of time, we have

$$\partial_t J^*(x,t) = 0$$

and therefore $J^*(x)$ satisfies

$$0 = \min_{u \in U} \left\{ q(x,u) + f(x,u)\, \partial_x J^*(x) \right\}. \tag{6.8}$$

In this case, the above equation makes sense only if the integral of the run-time cost with the optimal controller $\int_0^\infty q(x(t), u^*(x(t)))\, \mathrm{d}t$ remains bounded and does not diverge to infinity. Therefore typically in this problem we will set $q(0,0) = 0$, i.e., there is no cost for the system being at the origin with zero control, otherwise the integral of the run-time cost will never be finite. This also gives the boundary condition $J^*(0) = 0$ for the HJB equation.

6.2.2 Solving the HJB equation

The HJB equation is a partial differential equation (PDE) because there is one cost-to-go from every state $x \in X$ and for every time $t \in [0,T]$. It belongs to a large and important class of PDEs, collectively known as Hamilton-Jacobi-type equations. As you can imagine, since dynamic programming is so pervasive and solutions of DP are very useful in practice for a number of problems, there have been many tools invented to solve the HJB equation. These tools have applications to a wide variety of problems, from understanding how sound travels in crowded rooms to how light diffuses in an animated movie scene, to even obtaining better algorithms to train deep networks (https://arxiv.org/abs/1704.04932). HJB equations are usually never exactly solvable and a number of approximations need to be made in order to solve it.

In this course, we will not solve the HJB equation. Rather, we are interested in seeing how the HJB equation looks for continuous-time linear dynamical systems (both deterministic and stochastic ones) and LQR problems for such systems, as done in the following section.

An example We will look at a classical example of the so-called car-on-the-hill problem given below. The state of the problem is the position and

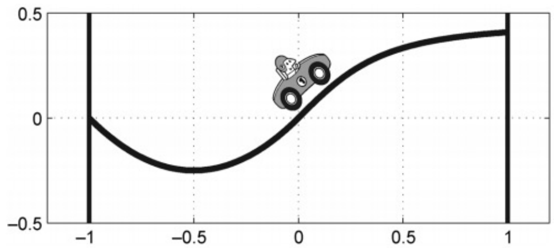

Figure 6.4: A car whose position is given by $z(t)$ would like to climb the hill to its right and reach the top with minimal velocity. The car rolls on the hill without friction. The run-time cost is zero everywhere inside the state-space. Terminal cost is -1 for hitting the left boundary ($z = -1$) and $-1 - \dot{z}/2$ for reaching the right boundary ($z = 1$). The car is a single integrator, i.e., $\dot{z} = u$ with only two controls ($u = 4$ and $u = -4$) and cannot exceed a given velocity (in this case $|\dot{z}| \leq 4$. This looks like a simple dynamic programming problem but it is quite hard due to the constraint on the velocity. The car may need to make multiple swing ups before it gains enough velocity (but not too much) to climb up the hill.

velocity $(z, \dot{z})$ and we can solve a two-dimensional HJB equation to obtain the optimal cost-to-go from any state, as done by the authors Yuval Tassa and Tom Erez in "Least Squares Solutions of the HJB Equation With Neural Network Value-Function Approximators" (https://homes.cs.washington.edu/~todorov/courses/amath579/reading/NeuralNet.pdf). In practice, while solving the HJB PDE, one discretizes the state-space at given set of states and solves the HJB equation (6.7) on this grid using numerical methods (these authors used neural networks to solve it). The end result looks as follows.

6.2.3 Continuous-time LQR

Consider a linear continuous-time dynamical system given by

$$\dot{x} = A\,x + B\,u; \quad x(0) = x_0.$$

In the LQR problem, we are interested in finding a control trajectory that minimizes, as usual, a cost function that is quadratic in states and controls, except that we have an integral of the run-time cost because our system is a

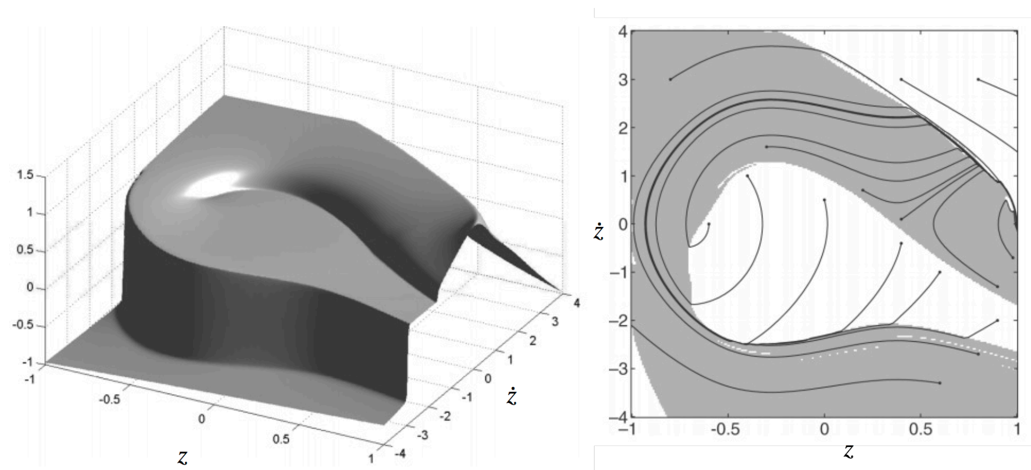

Figure 6.5: The left-hand side picture shows the infinite-horizon cost-to-go $J^*(z, \dot{z})$ for the car-on-the-hill problem. Notice how the value function is non-smooth at various places. This is quite typical of difficult dynamic programming problems. The right-hand side picture shows the optimal trajectories of the car $(z(t), \dot{z}(t))$; gray areas indicate maximum control and white areas indicate minimum control. The black lines show a few optimal control sequences taken the car starting from various states in the state-space. Notice how the optimal control trajectory can be quite different even if the car starts from nearby states (-0.5,1) and (-0.4,1.2)). This is also quite typical of difficult dynamic programming problems.

continuous-time system

$$\frac{1}{2}\, x(T)^\top Q_f\, x(T) + \frac{1}{2} \int_0^T x(t)^\top Q\, x(t) + u(t)^\top R\, u(t)\, \mathrm{d}t.$$

This is a very nice setup for using the HJB equation from the previous section.

Let us use our intuition from the discrete-time LQR problem and say that the optimal cost is quadratic in the states, namely,

$$J^*(x,t) = \frac{1}{2} x(t)^\top P(t)\, x(t);$$

notice that as usual the optimal cost-to-go is a function of the states x and the time t because is the optimal cost of the continuous-time LQR problem if the system starts at a state x at time t and goes on until time $T \geq t$. We will now check if this J^* satisfies the HJB equation (we don't write the arguments $x(t)$, $u(t)$ etc. to keep the notation clear)

$$-\partial_t J^*(x,t) = \min_{u \in U} \left\{ \frac{1}{2}\left(x^\top Q x + u^\top R\, u\right) + (A\, x + B\, u)^\top\, \partial_x J^*(x,t) \right\} \tag{6.9}$$

from (6.7). The minimization is over the control input that we take *at time* t.

Also notice the partial derivatives

$$\begin{aligned}\partial_x J^*(x,t) &= P(t)\, x.\\ \partial_t J^*(x,t) &= \frac{1}{2} x^\top \dot{P}(t)\, x.\end{aligned}$$

It is convenient in this case to see that the minimization can be performed using basic calculus (just like the discrete-time LQR problem), we differentiate with respect to u and set it to zero.

$$\begin{aligned} 0 &= \frac{\mathrm{d}\text{ RHS of HJB}}{\mathrm{d}u} \\ \Rightarrow u^*(t) &= -R^{-1}\, B^\top P(t)\, x(t) \\ &\equiv -K(t)\, x(t). \end{aligned} \tag{6.10}$$

where $K(t) = R^{-1}\, B^\top P(t)$ is the Kalman gain. The controller is again linear in the states $x(t)$ and the expression for the gain is very simple in this case, much simpler than discrete-time LQR. Since $R \succ 0$, we also know that $u^*(t)$ computed here is the global minimum. If we substitute this value of $u^*(t)$ back into the HJB equation we have

$$\{\}\, \Big|_{u^*(t)} = \frac{1}{2} x^\top \left\{ PA + A^\top P + Q - PBR^{-1}B^\top P \right\}\, x.$$

If order to satisfy the HJB equation, we must have that the expression above is equal to $-\partial_t J^*(x,t)$. We therefore have, what is called the Continuous-time Algebraic Riccati Equation (CARE), for the matrix $P(t) \in \mathbb{R}^{d\times d}$

$$-\dot{P} = PA + A^\top P + Q - PBR^{-1}B^\top P. \tag{6.11}$$

This is an ordinary differential equation for the matrix P. The derivative $\dot{P} = \frac{\mathrm{d}P}{\mathrm{d}t}$ stands for differentiating every entry of P individually with time t. The terminal cost is $\frac{1}{2} x(T)^\top Q_f\, x(T)$ which gives the boundary condition for the ODE as

$$P(T) = Q_f.$$

Notice that the ODE for the $P(t)$ travels backwards in time.

Continuous-time LQR has particularly easy equations, as you can see in (6.10) and (6.11) compared to those for discrete-time ((6.4) and (6.5)). Special techniques have been invented for solving the Riccati equation. I used the function scipy.linalg.solve_continuous_are to obtain Figure 6.3 using the continuous-time equations; the corresponding function for solving Discrete-time Algebraic Riccati Equation (DARE) which is given in (6.5) is scipy.linalg.solve_discrete_are. The continuous-time point-of-view also gives powerful connections to the Kalman filter, where you can show that the Kalman filter and LQR are duals of each other: in fact the equations for the Kalman filter (in continuous-time) and continuous-time LQR turn out to be exactly the same after you interchange appropriate quantities (!).

Infinite-horizon LQR Just like the infinite-horizon HJB equation has $\partial_t J^*(x,t) = 0$, if we have an infinite-horizon LQR problem, the cost matrix P should not be a function of time

$$\dot{P} = 0.$$

The continuous-time algebraic Riccati equation in (6.11) now becomes

$$PA + A^\top P + Q - PBR^{-1}B^\top P.$$

with the cost-to-go being given by $J^*(x) = \frac{1}{2} x^\top P x$.

6.3 Stochastic LQR

We will next look at a very powerful result. Say we have a stochastic linear dynamical system

$$\dot{x}(t) = Ax(t) + Bu(t) + B_\epsilon \epsilon(t);\ x(0) \text{ is given}$$

where $\epsilon(t)$ is standard Gaussian noise $\epsilon(t) \sim N(0, I)$ that is uncorrelated in time and would like to find a control sequence $\{u(t) : t \in [0,T]\}$ that minimizes a quadratic run-time and terminal cost

$$\underset{\epsilon(t):t\in[0,T]}{\mathrm{E}} \left[\frac{1}{2} x(T)^\top Q_f x(T) + \frac{1}{2} \int_0^T x(t)^\top Q x(t) + u(t)^\top R u(t) \,\mathrm{d}t \right].$$

over a finite-horizon T. Notice that since the system is stochastic now, we should minimize the expected value of the cost over all possible realizations of the noise $\{\epsilon(t) : t \in [0,T]\}$. This is a very challenging problem, conceptually it is the equivalent of dynamic programming for an MDP with an infinite number of states $x(t) \in \mathbb{R}^d$ and an infinite number of controls $u(t) \in \mathbb{R}^m$.

However, it turns out that the optimal controller that we should pick in this case is also given by the standard LQR problem

$$\begin{aligned} u^*(t) &= -R^{-1} B^\top P(t)\, x(t) \\ \text{with } -\dot{P} &= PA + A^\top P + Q - PBR^{-1}B^\top P;\ P(T) = Q_f. \end{aligned}$$

We will not do the proof (it is easy but tedious, you can try to show it by writing the HJB equation for the stochastic LQR problem). This is a very surprising result because it says that even if the dynamical system had noise, the optimal control we should pick is exactly the same as the control we would have picked had the system been deterministic. It is a special property of the LQR problem and not true for other dynamical systems (nonlinear ones, or ones with non-Gaussian noise) or other costs.

We know that the control $u^*(t)$ is the same as the deterministic case. Is the cost-to-go $J^*(x,t)$ also the same? If you think about this, the cost-to-go in the stochastic case has to be a bit larger than the deterministic case because the noise $\epsilon(t)$ is always going to non-zero when we run the system, the LQR cost $J^*(x_0, 0) = \frac{1}{2} x_0^\top P(0) x_0$ is, after all, only the cost of the deterministic problem. It turns out that the cost for the stochastic LQR case for an initial

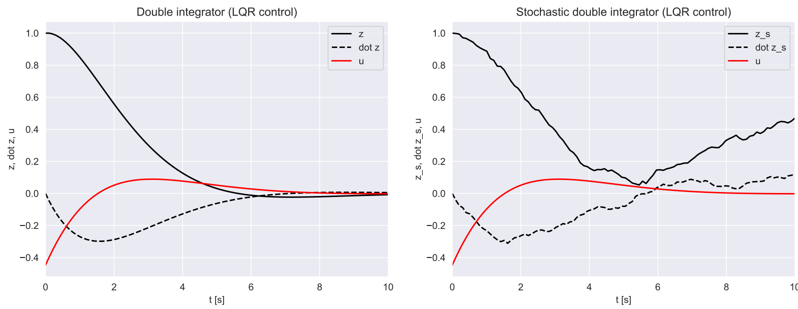

Figure 6.6: Comparison of the state trajectories of deterministic LQR and stochastic LQR problem with $B_\epsilon = [0.1, 0.1]$. The left panel is the same as that in Figure 6.3. The control input is the same in both cases but notice that the states in the plot on the right need not converge to the equilibrium due to noise. The cost of the trajectory will also be higher for the stochastic LQR case due to this. The total cost is $J^*(x_0) = 32.5$ for the deterministic case (32.24 for the quadratic state-cost and 0.26 for the control cost). The total cost $J^*(x_0)$ is much higher for the stochastic case, it is 81.62 (81.36 for the quadratic state cost and 0.26 for the control cost).

state x_0 is

$$\begin{aligned} J^*(x_0, 0) &= \mathop{\mathrm{E}}_{\epsilon(t):t\in[0,T]} \left[\frac{1}{2} x(T)^\top Q_f x(T) + \frac{1}{2} \int_0^T \ldots \, \mathrm{d}t \right] \\ &= \frac{1}{2} x_0^\top P(0) x_0 + \frac{1}{2} \int_0^T \mathrm{tr}(P(t) B_\epsilon B_\epsilon^\top) \, \mathrm{d}t. \end{aligned}$$

The first term is the same as that of the deterministic LQR problem. The second term is the penalty we incur for having a stochastic dynamical system. This is the minimal cost achievable for stochastic LQR but it is not the same as that of the deterministic LQR.

6.4 Linear Quadratic Gaussian (LQG)

Our development in the previous sections and the previous chapter was based on a Markov Decision Process, i.e., we know the state $x(t)$ at each instant in time t even if this state $x(t)$ changes stochastically. We said that the optimal control for the linear dynamics is still $u^*(t) = -K(t)\, x(t)$. What should one do if we cannot observe the state exactly?

Imagine a "continuous-time" form the observation equation in the Kalman filter where we receive observations of the form

$$y(t) = C x(t) + D\nu.$$

where $\nu \sim N(0, I)$ is standard Gaussian noise that corrupts our observations y. If we extrapolate the definitions of the Kalman filter mean and covariance to this continuous-time setting, we can write the KF as follows. We know that the Kalman filter is the optimal estimate of the state given all past observations, so it computes

$$\mu(t) = \underset{\epsilon(s),\nu(s):\ s\in[0,t]}{\mathrm{E}} \left[x(t) \mid y(s) : s \in [0,t] \right].$$

There exists a "continuous-time version" of the Kalman filter (which was actually invented first), called the Kalman-Bucy filter. If the covariance of the estimate is

$$\Sigma(t) = \underset{\epsilon(s),\nu(s):\ s\in[0,t]}{\mathrm{E}} \left[x(t)\, x(t)^\top \mid y(s) : s \in [0,t] \right],$$

the Kalman-Bucy filter updates $\mu(t), \Sigma(t)$ using the differential equation

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\mu(t) &= Ax(t) + Bu(t) + K(t)\left(y(t) - C\mu(t)\right) \\ \frac{\mathrm{d}}{\mathrm{d}t}\Sigma(t) &= A\Sigma(t) + \Sigma(t)A^\top + B_\epsilon B_\epsilon^\top - K(t)DD^\top K(t)^\top \end{aligned} \tag{6.12}$$

where $K(t) = \Sigma(t)\, C^\top (DD^\top)^{-1}$.

ℹ As we discussed while introducing stochastic dynamical systems, there are various mathematical technicalities associated with conditioning on a continuous-time signal $\{y(s) : s \in [0,t]\}$. To be precise mathematicians define what is called a "filtration" $\mathcal{Y}(t)$ which is the union of the Borel σ-fields constructed using increasing subsets of the set $\{y(s) : s \in [0,t]\}$. Let us not worry about this here.

This equation is very close to the Kalman filter equations you saw in Chapter 3. In particular, notice the close similarity of the expression for the Kalman gain $K(t)$ with the Kalman gain of the LQR problem. You can read more at https://en.wikipedia.org/wiki/Kalman_filter.

Linear Quadratic Gaussian (LQG) It turns out that we can plug in the Kalman filter estimate $\mu(t)$ of the state $x(t)$ in order to compute optimal control for LQR if we know the state only through observations $y(t)$

$$u^*(t) = -K(t)\, \mu(t). \tag{6.13}$$

It is almost as if, we can blindly run a Kalman Filter in parallel with the deterministic LQR controller and get the optimal control for the stochastic LQR problem even if we did not observe the state of the system exactly. This method is called Linear Quadratic Gaussian (LQG).

This is a very powerful and surprising result. It is only true for linear dynamical systems with linear observations, Gaussian noise in both the dynamics and the observations and quadratic run-time and terminal costs. It is not true in other cases. However, it is so elegant and useful that it inspires essentially all other methods that control a dynamical system using observations from sensors.

Certainty equivalence For instance, even if we are using a particle filter to estimate the state of the system, we usually use the mean of the state estimate at time t given by $\mu(t)$ "as if" it were the true state of the system. Even if we were using some other feedback control $u(x)$ different than the LQR control (say feedback linearization), we usually plug in this estimate $\mu(t)$ in place of $x(t)$. Doing so is called "certainty equivalence" in control theory/robotics, which is a word borrowed from finance where one takes decisions (controls) directly using the estimate of the state (say stock price) while fully knowing the the stock price will change in the future stochastically.

6.4.1 (Optional material) The duality between the Kalman Filter and LQR

We can re-write the covariance in (6.12) using the identity

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\Sigma(t)^{-1}\right) = \Sigma(t)^{-1}\dot{\Sigma}(t)\Sigma(t)^{-1}$$

to get

$$\dot{S} = C^\top \left(DD^\top\right)^{-1} C - A^\top S - SA - SB_w B_w^\top S \tag{6.14}$$

where we have defined $S := \Sigma^{-1}$.

Notice that the two equations, updates to the LQR cost matrix in (6.11)

$$-\dot{P} = PA + A^\top P + Q - PBR^{-1}B^\top P$$

look quite similar to this equation. In fact, they are identical and you can substitute the following.

LQR	Kalman-Bucy filter
P	Σ^{-1}
A	$-A$
$BR^{-1}B$	$B_w B_w^\top$
Q	$C^\top \left(DD^\top\right)^{-1} C$
t	$T - t$

Let us analyze this equivalence. Notice that the inverse of the Kalman filter covariance is like the cost matrix of LQR. This is conceptually easy to understand, our figure of merit for filtering is the inverse covariance matrix (smaller the better) and our figure of merit for the LQR problem is the cost matrix P (smaller the better). Similarly, smaller the LQR cost, better the controller. The "dynamics" of the Kalman filter is the reverse of the dynamics of the LQR problem, this shows that the P matrix is updated backwards in time while the covariance Σ is updated forwards in time. The next identity

$$BR^{-1}B^\top = B_w B_w^\top$$

is very interesting. Imagine a situation where we have a fully-actuated system with $B = I$ and B_w being a diagonal matrix. This identity suggests that larger the control cost R_{ii} of a particular actuator i, lower is the noise of using that actuator $(B_w)_{ii}$, and vice-versa. This is how muscles in your body have evolved: muscles that are cheap to use (low R) are also very noisy in what they do whereas muscles that are expensive to use (large R) which are typically the biggest muscles in the body are also the least noisy and most precise. You can read more about this in the paper titled "General duality between optimal control and estimation" by Emanuel Todorov. The next identity

$$Q = C^\top \left(DD^\top\right)^{-1}\ C$$

is related to the quadratic state-cost in LQR. Imagine the situation where both Q, D are diagonal matrices. If the noise in the measurements D_{ii} is large, this is equivalent to the state-cost matrix Q_{ii} being small; roughly there is no way we can achieve a low state-cost $x^\top Q x$ in our system that consists of LQR and a Kalman filter (this combination is known as Linear Quadratic Gaussian LQG as saw before) if there is lots of noise in the state measurements. The final identity

$$t = T - t$$

is the observation that we have made many times before: dynamic programming travels backwards in time and the Kalman filter travels forwards in time.

6.5 Iterative LQR (iLQR)

This section is analogous to the section on the Extended Kalman Filter. We will study how to solve optimal control problems for a nonlinear dynamical system

$$\dot{x} = f(x, u);\; x(0) = x_0 \text{ is given.}$$

We will consider a deterministic continuous-time dynamical system, the modifications to following section that one would make if the system is discrete-time, or stochastic, are straightforward and follow the same strategy. First consider the problem where the run-time and terminal costs are quadratic

$$\frac{1}{2} x(T)^\top Q_f x(T) + \frac{1}{2} \int_0^T \; x(t)^\top Q \, x(t) + u(t)^\top R u(t) \, \mathrm{d}t.$$

Receding horizon control and Model Predictive Control (MPC) One easy way to solve the dynamic programming problem, i.e., find a control trajectory of the *nonlinear* system that minimizes this cost functional, approximately, is by linearizing the system about the initial state x_0 and some reference control u_0 (this can usually be zero). Let the linear system be

$$\dot{z} = A_{x_0,u_0} z + B_{x_0,u_0} v;\; z(0) = 0; \tag{6.15}$$

where $A_{x_0,u_0} = \frac{\mathrm{d}f}{\mathrm{d}x}\Big|_{x=x_0,u=u_0}$ and $B_{x_0,u_0} = \frac{\mathrm{d}f}{\mathrm{d}u}\Big|_{x=x_0,u=u_0}$ are the Jacobians of the nonlinear function $f(x, u)$ with respect to the state and control respectively. The state of the linearized dynamics is

$$z := x - x_0, \text{ and } v := u - u_0,$$

We have emphasized the fact that the matrices A_{x_0,u_0}, B_{x_0,u_0} depend upon the reference state and control using the subscript. Given the above linear system, we can find a control sequence $u^*(\cdot)$ that minimizes the cost functional using the standard LQR formulation. Notice now that even we computed this control trajectory using the approximate linear system, it can certainly be *executed* on the nonlinear system, i.e., at run-time we will simply set $u \equiv u^*(z)$.

The linearized dynamics in (6.15) is potentially going to be very different from the nonlinear system. The two are close in the neighborhood of x_0 (and

u_0) but as the system evolves using our control input to move further away from x_0, the linearized model no longer is a faithful approximation of the nonlinear model. A reasonable way to fix matters is to linearize about another point, say the state and control after $t = 1$ seconds, x_1, u_1 to get a new system

$$\dot{z} = A_{x_1,u_1} z + B_{x_1,u_1} v;\ z(0) = 0$$

and take the LQR-optimal control corresponding to this system for the next second.

The above methodology is called "receding horizon control". The idea is that we compute the optimal control trajectory $u^*(\cdot)$ using an approximation of the original system and recompute this control every few seconds when our approximation is unlikely to be accurate. This is a very popular technique to implement optimal controllers in typical applications. The concept of using an approximate model (almost invariably, a linear model with LQR cost) to plan for the near-term future and resolving the problem in receding horizon fashion once the system is at the end of this short time-horizon is called "Model Predictive Control".

MPC is, perhaps, the second most common control algorithm implemented in the world. It is responsible for running most complex engineering systems that you can think of—power grids, oil refineries, chemical plants, rockets, aircrafts etc. Essentially, one never implements LQR directly, it is always implemented inside an MPC. For instance, in autonomous driving, the trajectory that the vehicle plans for traveling between two points A and B depends upon the current locations of the other cars/pedestrians in its vicinity, and potentially some prediction model of where they will be in the future. As the vehicle starts moving along this trajectory, the rest of the world evolves around it and we recompute the optimal trajectory to take into account the actual locations of the cars/pedestrians in the future.

❓ Can you guess what is *the* most common control algorithm in the world?

6.5.1 Iterative LQR (iLQR)

Now let us consider the situation when in addition to a nonlinear system,

$$\dot{x} = f(x, u);\ x(0) = x_0,$$

the run-time and terminal cost is also nonlinear

$$q_f(x(T)) + \int_0^T q(x(t), u(t))\, \mathrm{d}t.$$

We can solve the dynamic programming problem in this case approximately using the following iterative algorithm.

Assume that we are given an initial control trajectory $u^{(0)}(\cdot) = \left\{u^{(0)}(t) : t \in [0, T]\right\}$. Let $x^{(0)}(\cdot)$ be the state trajectory that corresponds to taking this control on the nonlinear system, with of course $x^{(0)}(0) = x_0$. At each iteration k, the Iterative LQR algorithm performs the following steps.

Step 1 Linearize the nonlinear system about the state trajectory $x^{(k)}(\cdot)$ and

$u^{(k)}(\cdot)$ using

$$z(t) := x(t) - x^{(k)}(t), \text{ and } v(t) := u(t) - u^{(k)}(t)$$

to get a new system

$$\dot{z} = A^{(k)}(t)z + B^{(k)}(t)v; z(0) = 0$$

where

$$A^{(k)}(t) = \frac{\mathrm{d}f}{\mathrm{d}x}\Big|_{x(t)=x^{(k)}(t),u(t)=u^{(k)}(t)}$$
$$B^{(k)}(t) = \frac{\mathrm{d}f}{\mathrm{d}u}\Big|_{x(t)=x^{(k)}(t),u(t)=u^{(k)}(t)}$$

and compute the Taylor series approximation of the nonlinear cost up to the second order

$$\begin{aligned} q_f(x(T)) \approx \text{constant} &+ z(T)^\top \frac{\mathrm{d}q_f}{\mathrm{d}x}\Big|_{x(T)=x^{(k)}(T)} \\ &+ z(t)^\top \frac{\mathrm{d}^2 q_f}{\mathrm{d}x^2}\Big|_{x(T)=x^{(k)}(T)} z(t), \end{aligned}$$

$$\begin{aligned} q(x, u, t) \approx \text{constant} &+ z(t)^\top \underbrace{\frac{\mathrm{d}q}{\mathrm{d}x}\Big|_{x(t)=x^{(k)}(t),u(t)=u^{(k)}(t)}}_{\text{affine term}} \\ &+ v(t)^\top \underbrace{\frac{\mathrm{d}q}{\mathrm{d}u}\Big|_{x(t)=x^{(k)}(t),u(t)=u^{(k)}(t)}}_{\text{affine term}} \\ &+ z(t)^\top \underbrace{\frac{\mathrm{d}^2 q}{\mathrm{d}x^2}\Big|_{x(t)=x^{(k)}(t),u(t)=u^{(k)}(t)}}_{\equiv Q} z(t) \\ &+ v(t)^\top \underbrace{\frac{\mathrm{d}^2 q}{\mathrm{d}u^2}\Big|_{x(t)=x^{(k)}(t),u(t)=u^{(k)}(t)}}_{\equiv R} v(t). \end{aligned}$$

❓ How will you solve for the optimal controller for a linear dynamics for the cost

$$\int_0^T \left(q^\top x + \frac{1}{2} x^\top Q x \right) \,\mathrm{d}t,$$

i.e., when in addition the quadratic cost, we also have an affine term?

This is an LQR problem with run-time cost that depends on time (like our discrete-time LQR formulation, the continuous-time formulation simply has Q, R to be functions of time t in the Riccati equation) and which also has terms that are affine in the state and control in addition to the usual quadratic cost terms.

Step 2 Solve the above linearized problem using standard LQR formulation to get the new control trajectory

$$u^{(k+1)}(t) := u^{(k)}(t) - Kz(t).$$

Simulate the *nonlinear* system using the control $u^{(k+1)}(\cdot)$ to get the new state trajectory $x^{(k+1)}(\cdot)$.

Some important comments to remember about the iLQR algorithm.

1. There are many ways to pick the initial control trajectory $u^{(0)}(\cdot)$, e.g.,

using a spline to get an arbitrary control sequence, using a spline to interpolate the states to get a trajectory $x^{(0)}(\cdot)$ and then back-calculate the control trajectory, using the LQR solution based on the linearization about the initial state, feedback linearization/differential flatness (https://en.wikipedia.org/wiki/Feedback_linearization) etc.

2. The iLQR algorithm is an approximate solution to dynamic programming for nonlinear system with general, nonlinear run-time and terminal costs. This is because the the algorithm uses a linearization about the previous state and control trajectory to compute the new control trajectory. iLQR is not guaranteed to find the optimal solution of dynamic programming, although in practice with good implementations, it works excellently.
3. We can think of iLQR as an algorithm to track a given state trajectory $x^g(t)$ by setting

$$q_f = 0, \text{and } q(x, u) = \|x^g(t) - x(t)\|^2.$$

This is often how iLQR is typically used in practice, e.g., to make an autonomous race car closely follow the racing line (see the paper "BayesRace: Learning to race autonomously using prior experience" https://arxiv.org/abs/2005.04755 and https://www.youtube.com/watch?v=dgIpf0Lg8Ek for a clever application of using MPC to track a challenging race line), or to make a drone follow a given desired trajectory (https://www.youtube.com/watch?v=QREeZvHg0lQ).

Differential Dynamic Programming (DDP) is a suite of techniques that is a more powerful version of iterated LQR. Instead of linearizing the dynamics and taking a second order Taylor approximation of the cost, DDP takes a second order approximation of the Bellman equation directly. The two are not the same; DDP is the more correct version of iLQR but is much more challenging computationally.

Broadly speaking, iLQR and DDP are used to perform control for some of the most sophisticated robots today, you can see an interesting discussion of the trajectory planning of some of the DARPA Humanoid Robotics Challenge at https://www.cs.cmu.edu/~cga/drc/atlas-control. Techniques like feedback linearization work excellently for drones where we do not really care for optimal cost (see "Minimum snap trajectory generation and control for quadrotors" https://ieeexplore.ieee.org/document/5980409) while LQR and its variants are still heavily utilized for satellites in space.