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

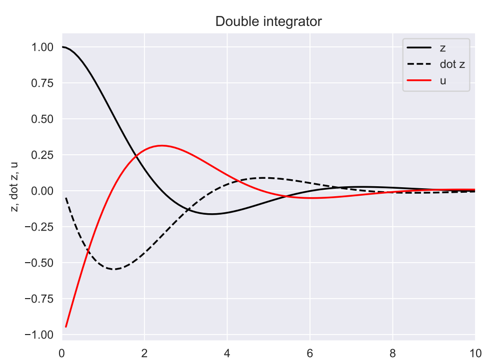

Figure 6.1: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a stabilizing (i.e., one that makes the system asymptotically stable) controller $u = -z(t) - \dot{z}(t)$. Notice how the trajectory starts from some initial condition (in this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

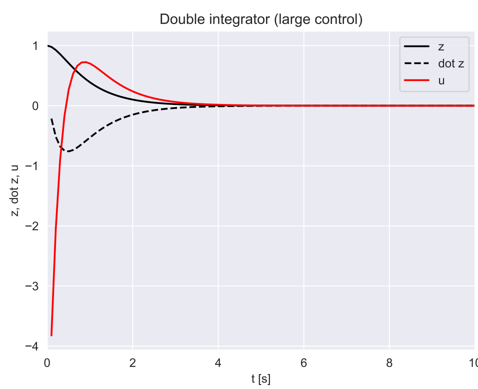

Figure 6.2: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a large stabilizing control at each time $u = -5z(t) - 5\dot{z}(t)$. Notice how quickly the state trajectory converges to the equilibrium without much oscillation as compared to 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 $x_{k+1} = Ax_k + Bu_k$ 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

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

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

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

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.

   ℹ 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

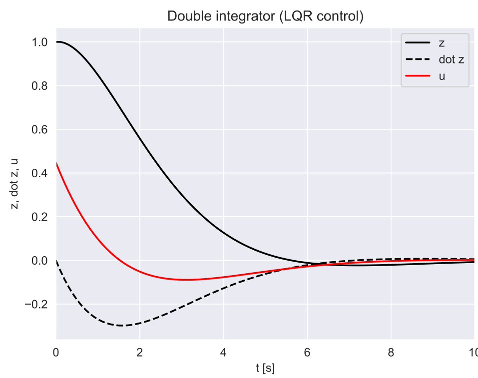

Figure 6.3: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a controller obtained from LQR with $Q = I$ and $R = 5$. This gives the controller to be about $u = -0.45z(t) - 1.05\dot{z}(t)$. Notice how we still get stabilization but the control 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:

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

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

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

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

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

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

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

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

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

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).

> 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.

6.2.1 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 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)

$$-\partial_t J^*(x,t) = \min_{u \in U} \left\{ \frac{1}{2} \left(x^\top Q x + u^\top R\, u \right) + (A\,x + B\,u)^\top\, \partial_x J^*(x,t) \right\} \tag{6.8}$$

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.

$$\begin{aligned} 0 &= \frac{\text{d RHS of HJB}}{\text{d}u} \\ \Rightarrow u^*(t) &= -R^{-1}\, B^\top P(t)\, x(t) \\ &\equiv -K(t)\, x(t). \end{aligned} \tag{6.9}$$

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 $P(t) \in \mathbb{R}^{d \times d}$

$$-\dot{P} = PA + A^\top P + Q - PBR^{-1}B^\top P. \tag{6.10}$$

This is an ordinary differential equation for the matrix P. The derivative $\dot{P} = \frac{\text{d}P}{\text{d}t}$ 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 (6.9) and (6.10) compared to those for discrete-time ((6.4) and (6.5)). As it happens, the continuous-time LQR formulation is more convenient because 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 and LQR are the exact same after you replace appropriate quantities (!).

6.3 Stochastic LQR

6.4 Linear Quadratic Gaussian (LQG)

6.5 Iterative LQR (iLQR)

6.6 Model Predictive Control (MPC)

Bibliography