ME 7247: Advanced Control Systems **Fall 2022–23**

Lecture 13: Linear Quadratic Regulator (LQR)

Friday, October 21, 2022

Lecturer: Laurent Lessard *Scribe: Milad Alipour Shahraki*

In this lecture, we will talk about the topic of control and Linear Quadratic Regulator (LQR). We will see that LQR control is the counterpoint to Kalman filter estimation and explore the connections between the two. We will study the deterministic LQR now.

1 Problem Formulation

The model of the dynamical system is described as follows (discrete-time case):

$$x_{t+1} = Ax_t + Bu_t, \tag{1}$$

Eq. (1) is a standard state-space model where $x_{t+1} \in \mathbb{R}^n$ and $x_t \in \mathbb{R}^n$ are the system states at times $t+1$ and t, respectively. $u_t \in \mathbb{R}^m$ is the system input at time t, $A \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{n\times m}$. The initial state is x_0. This lecture considers a system without noise; in future class we will cover the case with noise as well.

Goal: The goal is to find a sequence of inputs $u_0, u_1, \ldots, u_{N-1}$ to minimize the cost function J in the following optimization problem:

$$J = \underset{u_0,\ldots,u_{N-1}}{\text{minimize}} \quad \sum_{t=0}^{N-1} \underbrace{(x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t)}_{\text{Stage Cost}} + \underbrace{x_N^\mathsf{T} Q_f x_N}_{\text{Terminal Cost}} \ . \tag{2}$$

under the assumptions that:

$$\boxed{Q \succeq 0, \quad R \succ 0, \quad Q_f \succeq 0}$$

The reason for these assumptions is that we want the minimum possible cost to be zero. We are allowing Q to be positive semi-definite since we do not have to penalize all of the states. However, we want R to be positive definite since we want to penalize every single input we use.

Note: There is a misconception that the LQR is somehow better than other classical controllers, such as the PID control, mainly because it is referred to as an *optimal* method. However, the reality is you still have to *design* the controller. In PID control, we design the controller by tuning the control gains directly. So if we have a good understanding and intuition of how the various gains affect the system performance, we can get the performance we want out of a PID controller. In LQR, we instead design the cost matrices Q, R, and Q_f. We can think of them as trade-off parameters (penalizing large state vs. penalizing large input), and we solve the control problem in an indirect fashion, but it is still a design problem. This does not mean that the LQR is superior to PID; they are just different design methods that require different sorts of intuition about the problem.

Since the problem is just a Least Squares problem, we can rewrite the problem as a quadratic expression. Then we can take the derivative of the expression and set it to zero to find the solution. Therefore, we have:

$$\underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}}_{\bar{x}} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ B & 0 & 0 & \dots & 0 \\ AB & B & 0 & \dots & 0 \\ A^2B & AB & B & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{bmatrix}}_{\bar{F}} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_N \end{bmatrix}}_{\bar{u}} + \underbrace{\begin{bmatrix} I \\ A^1 \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}}_{\bar{G}} x_0 \tag{3}$$

where $\bar{x} \in \mathbb{R}^{(N+1)n\times 1}$, $\bar{F} \in \mathbb{R}^{(N+1)n\times Nm}$, $\bar{u} \in \mathbb{R}^{Nm\times 1}$, $\bar{G} \in \mathbb{R}^{(N+1)n\times n}$, and $x_0 \in \mathbb{R}^{n\times 1}$. Therefore, we can rewrite the dynamical system and the cost as follows:

$$\bar{x} = \bar{F}\bar{u} + \bar{G}x_0 \tag{4}$$

$$J = \bar{x}^\mathsf{T}\bar{Q}\bar{x} + \bar{u}^\mathsf{T}\bar{R}\bar{u} \tag{5}$$

Substituting (4) into (5), we obtain:

$$J(\bar{u}) = (\bar{F}\bar{u} + \bar{G}x_0)^\mathsf{T} Q(\bar{F}\bar{u} + \bar{G}x_0) + \bar{u}^\mathsf{T}\bar{R}\bar{u}, \tag{6}$$

where

$$\bar{Q} = \begin{bmatrix} Q & & & \\ & Q & & \\ & & \ddots & \\ & & & Q_f \end{bmatrix}, \qquad \bar{R} = \begin{bmatrix} R & & & \\ & R & & \\ & & \ddots & \\ & & & R \end{bmatrix}.$$

2 Non-recursive Solution

We can write the equation (6) in the quadratic form:

$$J(\bar{u}) = \begin{bmatrix} x_0 \\ \bar{u} \end{bmatrix}^\mathsf{T} \begin{bmatrix} \bar{G}^\mathsf{T}\bar{Q}\bar{G} & \bar{G}^\mathsf{T}\bar{Q}\bar{F} \\ \bar{F}^\mathsf{T}\bar{Q}\bar{G} & \bar{F}^\mathsf{T}\bar{Q}\bar{F} + \bar{R} \end{bmatrix} \begin{bmatrix} x_0 \\ \bar{u} \end{bmatrix} \tag{7}$$

The optimal $\bar{u}$ and J can be obtained as

$$\bar{u}_\star = -(\bar{F}^\mathsf{T}\bar{Q}\bar{F} + \bar{R})^{-1}\bar{F}^\mathsf{T}\bar{Q}\bar{G}x_0 \tag{8}$$

$$J_\star = x_0^\mathsf{T}(\bar{G}^\mathsf{T}\bar{Q}\bar{G} - \bar{G}^\mathsf{T}\bar{Q}\bar{F}(\bar{F}^\mathsf{T}\bar{Q}\bar{F} + \bar{R})^{-1}\bar{F}^\mathsf{T}\bar{Q}\bar{G})x_0 \tag{9}$$

Note: In general, the problem can be time-varying. For example, the matrices A, B, Q, R, and Q_f can be time-varying, and everything about the above solution still works. The only exception is some formulas become more complicated. For example, in (3), instead of $A^k x_0$ we would have $A_{k-1}A_{k-2}\cdots A_1A_0x_0$.

There are two main issues with the non-recursive solution. The first issue is that it is not scalable. At every timestep, the relevant matrices such as $\bar{F}$ grow, and we must solve ever-growing systems of linear equations. This is similar to the issue with the non-recursive Kalman filter solution, where it did not make sense to solve a large least squares problem at every timestep.

There is a better way of writing the solution where we can write each decision as some constant gain multiplied by the current state. This approach benefits from additional robustness in practical implementations because it tends to correct cases where there is noise in the states. We will see this phenomenon in more detail when we study the stochastic LQR problem later.

3 Recursive LQR using Dynamic Programming

Now we are going to derive the recursive solution to LQR. There are several ways of deriving the recursive solution, and we will use dynamic programming since we will also use dynamic programming in the future. We will work our way back in time and pretend we are only one step away from the end horizon. We now will define a function called the *value function* (also known as the *cost-to-go*):

$$\begin{aligned} V_k(z) := \underset{u_k,\dots,u_{N-1}}{\text{minimize}} \quad & \sum_{t=k}^{N-1} (x_t^\top Q x_t + u_t^\top R u_t) + x_N^\top Q_f x_N \\ \text{s.t.} \quad & x_k = z \quad \text{and} \\ & x_{t+1} = Ax_t + Bu_t \quad \text{for} \quad t = k, \dots, N-1 \end{aligned} \tag{10}$$

The value function is a function at every time step, and instead of starting at time zero, it starts at time $t = k$. The only decisions we need to make are each k all the way to $N-1$, and the function's argument is actually our state at time k. We are trying to find $V_0(x_0)$; therefore, the larger the k gets, the easier the problem is in the sense that fewer decisions have to be made. At the terminal timestep, there is no decision to make. We have:

$$\boxed{V_N(z) = z^\top Q_f z} \tag{11}$$

Now we will solve for V_k in terms of V_{k+1} and so on. We will work our way back starting from $k = N-1$ until we get to zero, which will solve the problem. So to do this, we take equation (10) and pull the first stage cost out of the sum:

$$\begin{aligned} V_k(z) &= \min_{u_k} \underset{u_{k+1},\dots,u_{N-1}}{\text{minimize}} \left(x_k^\top Q x_k + u_k^\top R u_k + \sum_{t=k+1}^{N-1} (x_t^\top Q x_t + u_t^\top R u_t) + x_N^\top Q_f x_N \right) \\ &= \min_{u_k} \left(x_k^\top Q x_k + u_k^\top R u_k + \underset{u_{k+1},\dots,u_{N-1}}{\text{minimize}} \sum_{t=k+1}^{N-1} (x_t^\top Q x_t + u_t^\top R u_t) + x_N^\top Q_f x_N \right) \end{aligned} \tag{12}$$

Now to make things a bit easier, we give u_k a new name, i.e., w, since we are going to optimize over this one separately. Now we can rewrite the Eq. (12) as

$$V_k(z) = \min_w \left(z^\top Q z + w^\top R w + \underbrace{\underset{u_{k+1},\dots,u_{N-1}}{\text{minimize}} \sum_{t=k+1}^{N-1} (x_t^\top Q x_t + u_t^\top R u_t) + x_N^\top Q_f x_N}_{V_{k+1}(Az+Bw)} \right) \tag{13}$$

Therefore, we can write the following recursive formula (the Bellman equation):

$$V_k(z) = \min_w \left(z^\mathsf{T} Q z + w^\mathsf{T} R w + V_{k+1}(Az + Bw) \right) \tag{14}$$

Suppose $V_k(z) = z^\mathsf{T} P_k z$ (V is quadratic). This is true for $k = N$, so we will use induction to show that if it's true for $k+1$, it is also true for k.

$$\begin{aligned} V_k(z) &= \min_w \left(z^\mathsf{T} Q z + w^\mathsf{T} R w + V_{k+1}(Az + Bw) \right) \\ &= \min_w \left(z^\mathsf{T} Q z + w^\mathsf{T} R w + (Az + Bw)^\mathsf{T} P_{k+1} (Az + Bw) \right) \\ &= \min_w \left(\begin{bmatrix} z \\ w \end{bmatrix}^\mathsf{T} \begin{bmatrix} A^\mathsf{T} P_{k+1} A + Q & A^\mathsf{T} P_{k+1} B \\ B^\mathsf{T} P_{k+1} A & B^\mathsf{T} P_{k+1} B + R \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix} \right) \end{aligned} \tag{15}$$

So now we can obtain the optimal decision and the optimal value as follows:

$$w_\star = -(B^\mathsf{T} P_{k+1} B + R)^{-1} B^\mathsf{T} P_{k+1} A z \tag{16}$$

$$V_k(z) = z^\mathsf{T} \underbrace{\left(A^\mathsf{T} P_{k+1} A + Q - A^\mathsf{T} P_{k+1} B (B^\mathsf{T} P_{k+1} B + R)^{-1} B^\mathsf{T} P_{k+1} A\right)}_{P_k} z \tag{17}$$

So we started with a quadratic function of z, i.e., equation (11), and ended up with another quadratic function of z. If we continue this approach, we will obtain quadratic functions of z for every step. So we can get all the way back to time zero. Therefore, to formalize this, we define:

$$V_k(z) = z^\mathsf{T} P_k z \tag{18}$$

Now we can formulate the recursive update formula as

$$\begin{aligned} &\boxed{P_N = Q_f} \\ &\boxed{P_k = A^\mathsf{T} P_{k+1} A + Q - A^\mathsf{T} P_{k+1} B (B^\mathsf{T} P_{k+1} B + R)^{-1} B^\mathsf{T} P_{k+1} A} \\ &\boxed{u_k = \underbrace{-(B^\mathsf{T} P_{k+1} B + R)^{-1} B^\mathsf{T} P_{k+1} A}_{K_k \text{ (LQR Gain)}} x_k} \end{aligned} \tag{19}$$

So the optimal policy (LQR optimal policy) is $u_t = K_t x_t$ and the cost-to-go is $V_t(z) = z^\mathsf{T} P_t z$.

Note: The matrices P_t can be computed ahead of time (without knowledge of the future states). Therefore, we can pre-compute our optimal *policy* K_t. Of course, the optimal *actions* u_t will still depend on our current state x_t.

Note: If we have $x_0 = 0$, i.e., we start at zero, then the optimal cost is zero because the optimal policy will be to do nothing forever, and then the state will stay at zero, and we will incur zero cost. However, if we pick an x_0 that is not zero, then decisions are going to be made that bring the states down asymptotically to zero over time. The relative weight that we place on x versus u has to do with the relative size of Q and R. If R is large, it will be expensive to use a large u. Therefore, we would be more inclined to use a very small u and just let x go to zero on its own if the system is stable. If R is very small, in other words, it does not cost very much to take action, we might take significant actions to drive the state x to zero rapidly, which incurs a higher cost, so it is a trade-off.

4 Comparison to Kalman Filter

In this section, we compare the recursive LQR with the Kalman filter. The recursive formulation for the Kalman filter was:

$$\begin{aligned}
\Sigma_0 &= \Sigma_x \\
\Sigma_{t+1} &= A\Sigma_t A^\top + W - A\Sigma_t C^\top (C\Sigma_t C^\top + V)^{-1} C\Sigma_t A^\top \qquad (20a) \\
L_t &= -A\Sigma_t C^\top (C\Sigma_t C^\top + V)^{-1}
\end{aligned}$$

If we compare (19) to (20), we realize that the solution to the LQR problem and Kalman filtering problems are the same, provided we map the parameters as follows:

$$\underbrace{(A, B, Q, R, K, P)}_{\text{LQR}} \longleftrightarrow \underbrace{(A^\top, C^\top, W, V, L^\top, \Sigma)}_{\text{Kalman Filter}}$$

Also, time flows backwards for LQR and forwards for the Kalman filter iterations.

5 Steady-State LQR

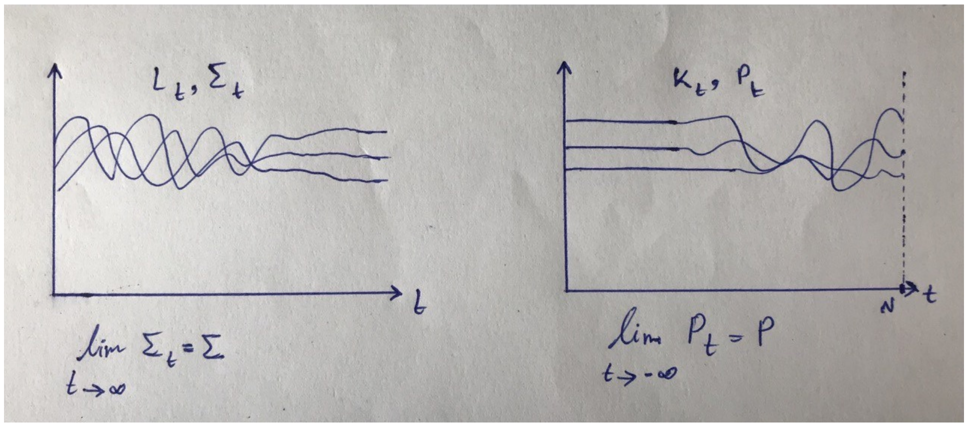

Figure 1: Comparison between Kalman filter and LQR

Fig. 1 shows the steady-state Kalman filter and the steady-state LQR. We can see that because initially, each measurement that we get gives us a lot of new information; as we move forward in time, we accrue more and more measurements. However, the measurements that we observed from a long time ago are less relevant since they were measurements of where the state was a long time ago. The state has since moved, and a lot of noise has been added in the meantime. So if we have gone far enough in time, all that matters is the new measurements that come in, and we can ignore the ones that are very far in the past. We reach this steady state where the rate of new information coming in equals the rate of losing information from all measurements.

Now for the steady-state LQR, since we move backward in time, what we get is the opposite of the steady-state Kalman filter. Therefore, at time $t = N$, the control gains, and the cost-to-go might have a lot of fluctuations; however, the farther you go back in time, the more constant they become. In practice, systems are often run for a relatively long time; therefore, the horizon is always infinitely far away. So it makes sense to use the steady-state LQR solution directly.

Now recall that the error dynamics $(A + LC)$ were stable with the Kalman filter. Similarly, since the system dynamics are as follows:

$$x_{t+1} = Ax_t + Bu_t \tag{21}$$
$$u_t = Kx_t \tag{22}$$

we can form the closed-loop equations by eliminating u_t:

$$x_{t+1} = (A + BK)x_t \tag{23}$$

with the LQR, the counterpart $(A + BK)$ is stable. The Lyapunov equation, therefore, can be obtained as

$$(A + BK)^\mathsf{T} P(A + BK) + (Q + K^\mathsf{T} RK). \tag{24}$$

Note: With the Kalman filter, the steady state can be reached very fast, and it is common to compute the steady-state Kalman filter and use that from the beginning. Even though the estimates are not optimal initially, they will become optimal quickly. Therefore, we can ignore the estimates that we were getting at the beginning. With LQR, the same is true. We will often compute the solution to the Riccati equation and determine the optimal steady-state cost from the start and implement that from the beginning. The difference is that with LQR, this procedure is correct from the beginning, and if the horizon is always infinitely far away, then it is going to be correct forever. Now similar to what we had for the steady-state Kalman filter, the Riccati results for steady-state LQR are as follows:

If $Q \succeq 0$ and $R \succ 0$, (A, B) stabilizable and (A, Q) detectable, then the Algebraic Riccati Equation (ARE)

$$A^\mathsf{T} PA - P + Q - A^\mathsf{T} PB(B^\mathsf{T} PB + R)^{-1} B^\mathsf{T} PA = 0$$

has a unique stabilizing solution $P \succeq 0$, meaning that if we define $K = -(B^\mathsf{T} PB + R)^{-1} B^\mathsf{T} PA$, then $A + BK$ is Schur-stable. Finally, iterating the Riccati recursion converges to the stabilizing solution, so $\lim_{t \to -\infty} P_t = P$.

6 MATLAB Commands

Relevant Matlab commands for estimation and control problems.

```
idare(A',C',W,V) % steady-state covariance for the Kalman filter
dlyap(A,W)       % steady-state covariance when there are no measurements (L=0)

idare(A,B,Q,R)   % stabilizing solution to the ARE
dlyap(A',Q)      % infinite-horizon cost matrix for the open-loop (K=0)
```