ECE586RL: MDPs and Reinforcement Learning
A Note on Linear Quadratic Regulator

Instructor: Bin Hu

March 16, 2020

In this note, we review some basic facts related to the linear quadratic regulator (LQR) which is used as a benchmark problem for reinforcement learning.

1 LQR without Process Noise

We start from a simple setup. Consider the following linear dynamical system

$$x_{t+1} = Ax_t + Bu_t \tag{1.1}$$

where A is the state matrix, B is the input matrix, u_t is the control action, and x_t is the system state. In the class, we start with the case where there is a stochastic process noise w_t in the dynamics. For simplicity, here we start with the case where there is no process noise w_t.

The objective is to choose $\{u_t\}$ to minimize the following cost

$$\mathcal{C} = \mathbb{E}_{x_0 \sim \mathcal{D}} \sum_{t=0}^{\infty} (x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t) \tag{1.2}$$

where Q and R are positive definite matrices. Since there is no process noise, the only randomness stems from the randomness in x_0. So there is an initial distribution $\mathcal{D}$ where x_0 is sampled from. The choices of Q and R reflect the conflicting design objectives in control: We want to achieve small tracking error by using small control inputs. The above setup is a little different from the discounting setup covered in the class. One main difference is that there is no discounting factor in the cost (1.2). Since there is no process noise w_t, we do not need a discounting factor γ to make $\mathcal{C}$ finite. When there is the process noise term w_t, the cost in (1.2) is never finite for $\gamma = 1$ due to the fact that x_t does not converge to 0. In that case, a discounting factor is needed. Nevertheless, the above LQR problem can still be viewed as a MDP on a continuous state space. A few key features are summarized as follows.

1. Continuous state space: x_t is a real vector and hence can take any values in $\mathbb{R}^x$.
2. Continuous action space: u_t is also a real vector.
3. Transition dynamics: Given x_t and u_t, then x_{t+1} is also known due to (1.1). The transition dynamics can be viewed a stochastic kernel centering at $(Ax_t + Bu_t)$ with probability 1 .

4. Additive structure of cost function: $\mathcal{C}$ is a sum of cost values at different t. The one-step cost depends on both the state and the input at that step.

5. No discounting factor in cost $\mathcal{C}$.

Given an initial condition x_0, we denote the state value function as $V(x_0) = \sum_{t=0}^{\infty}(x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t)$. Therefore, we have $\mathcal{C} = \mathbb{E}_{x\sim\mathcal{D}} V(x)$.

1.1 Policy Parameterization

As discussed in the class, it is reasonable to design u_t using the past information of the system. This type of closed-loop design is more robust to disturbance and model uncertainty. When the state x_t can be measured, it is reasonable to just consider the stationary policy in the form of state feedback, i.e. $u_t = \mu(x_t)$. For the finite state space case, μ can be represented as a table. Here, μ is a function mapping from $\mathbb{R}^x$ to $\mathbb{R}^u$. We want to design μ to minimize $\mathcal{C}$. In practice, we have to confine the search of μ within certain tractable classes of functions. Here are a few examples.

1. Linear policy: μ is parameterized by a matrix K, and we have $x_t = -K u_t$. Eventually the problem becomes finding K to minimize $\mathcal{C}$.

2. Neural network: μ can be parameterized by a neural network, i.e. $u_t = W_L \phi(W_{L-1} \ldots W_2 \phi(W_1 x_t))$ where L is the depth of the network. This is a more popular choice for complex nonlinear control tasks. Then the problem becomes minimizing $\mathcal{C}$ over the weights $(W_1, W_2, \ldots, W_L)$.

1.2 Policy Evaluation

Notice the cost (1.2) provides a measure for the performance of the policy μ. The smaller the cost $\mathcal{C}$ is, the better the policy μ performs. Policy evaluation refers to the calculation of $\mathcal{C}$ for a given policy μ.

Stability issue. Unlike the finite state space case, $\mathcal{C}$ may not be finite for a given μ. So there is a stability issue here. If μ stabilizes the system (1.1), then the cost (1.2) is finite.

Parameterization of value function In this note, we focus the policy evaluation for a linear policy $u_t = -K x_t$. Substituting $u_t = -K x_t$ into (1.1) leads to $x_{t+1} = (A - BK) x_t$. Hence we have

$$x_t = (A - BK)^t x_0 \tag{1.3}$$

We denote the spectral radius as ρ. If K stabilizes the system (1.1), then $\rho(A - BK) < 1$ and $x_t \to 0$ at a geometric rate. The cost (1.2) can be rewritten as

$$\mathcal{C}(K) = \mathbb{E}_{x_0\sim\mathcal{D}} \quad x_0^\mathsf{T} \left(\sum_{t=0}^{\infty} ((A - BK)^\mathsf{T})^t (Q + K^\mathsf{T} R K)(A - BK)^t \right) x_0 \tag{1.4}$$

When $\rho(A - BK) \geq 1$, the above cost blows up to infinity. It makes sense to restrict the policy search within the class of stabilizing K. When $\rho(A - BK) < 1$, we know $\sum_{t=0}^{\infty}((A - BK)^\mathsf{T})^t (Q + K^\mathsf{T} R K)(A - BK)^t$ will converge to a fixed constant matrix. We denote this matrix by P_K. Therefore, it is reasonable to parameterize the value function as $x_0^\top P_K x_0$ which is a quadratic function of x_0. When a nonlinear policy is used, we typically need to parameterize the value function as a neural network.

Bellman equation for policy evaluation. From the above discussion, we have already known $P_K = \sum_{t=0}^{\infty}((A-BK)^\intercal)^t(Q+K^\intercal RK)(A-BK)^t$. The bellman equation can be derived as follows.

$$\begin{aligned}
x_0^\intercal P_K x_0 &= \sum_{t=0}^{\infty} x_t^\intercal (Q + K^\intercal RK)x_t \\
&= x_0^\intercal (Q + K^\intercal RK)x_0 + \sum_{t=1}^{\infty} x_t^\intercal (Q + K^\intercal RK)x_t \\
&= x_0^\intercal (Q + K^\intercal RK)x_0 + x_1^\intercal P_K x_1 \\
&= x_0^\intercal (Q + K^\intercal RK)x_0 + x_0^\intercal (A - BK)^\intercal P_K (A - BK)x_0 \\
&= x_0^\intercal \left(Q + K^\intercal RK + (A - BK)^\intercal P_K (A - BK)\right) x_0
\end{aligned}$$

Therefore, the Bellman equation takes the following form:

$$P_K = Q + K^\intercal RK + (A - BK)^\intercal P_K (A - BK) \tag{1.5}$$

For any fixed K, the above equation is a linear equation of P_K. Hence the existence and uniqueness of the solution to the above Bellman equation can be established using linear equation theory.

To obtain a closed-form solution for P_K, we need to introduce the Kronecker product and the vectorization operation. The Kronecker product of two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ is denoted by $A \otimes B$ and given by:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

where a_{ij} is the (i, j)-th entry of A. Clearly, we have $A \otimes B \in \mathbb{R}^{pm\times qn}$. Notice $(A \otimes B)^T = A^T \otimes B^T$ and $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ when the matrices have compatible dimensions.

Next, let vec denote the standard vectorization operation that stacks the columns of a matrix into a vector. For example, we have

$$\text{vec}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 2 \\ 4 \\ 6 \end{bmatrix}.$$

An important fact is that we always have $\text{vec}(AXB) = (B^\intercal \otimes A)\,\text{vec}(X)$. Therefore, we have

$$\text{vec}\left((A - BK)^\intercal P_K (A - BK)\right) = \left((A - BK)^\intercal \otimes (A - BK)^\intercal\right) \text{vec}(P_K)$$

Then we can vectorize both sides of the Bellman equation (1.5) to obtain

$$\text{vec}(P_K) = \text{vec}(Q + K^\top RK) + \left((A - BK)^\intercal \otimes (A - BK)^\intercal\right) \text{vec}(P_K)$$

which can be easily solved for P_K:

$$\operatorname{vec}(P_K) = \left(I - (A-BK)^\mathsf{T} \otimes (A-BK)^\mathsf{T}\right)^{-1} \operatorname{vec}(Q + K^\mathsf{T} R K)$$

Now we have a closed-form solution for P_K. Using properties of the Kronecker product, one can show $\left(I - (A-BK)^\mathsf{T} \otimes (A-BK)^\mathsf{T}\right)$ is nonsingular under the assumption $\rho(A - BK) < 1$. We skip the details here. The key message here is that for any stabilizing K, we can solve (1.5) to obtain P_K and then the value function for K is $V(x) = x^\mathsf{T} P_K x$.

For general nonlinear policy, the existence and uniqueness conditions for Bellman equation are much more complicated.

1.3 Optimal Bellman Equation: Riccati Equation

For the above LQR problem, the optimal Bellman equation becomes the following Riccati equation

$$P = A^\mathsf{T} P A + Q - A^\mathsf{T} P B (B^\mathsf{T} P B + R)^{-1} B^\mathsf{T} P A \tag{1.6}$$

When (A, B) is stabilizable, the above equation has a unique positive definite[1] stabilizing solution P such that $\rho(A - B(B^\mathsf{T} P B + R)^{-1} B^\mathsf{T} P A) < 1$. Then the optimal action is given by a linear policy, i.e. $u_t = -(B^\mathsf{T} P B + R)^{-1} B^\mathsf{T} P A x_t$. This linear policy is optimal among all (potentially nonlinear) policies. This can be shown using completion of square. We can rewrite the cost as follows.

$$\begin{aligned}
\sum_{t=0}^{\infty} (x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t) &= x_0^\mathsf{T} P x_0 + \sum_{t=0}^{\infty} (x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t + x_{t+1}^\mathsf{T} P x_{t+1} - x_t^\mathsf{T} P x_t) \\
&= x_0^\mathsf{T} P x_0 + \sum_{t=0}^{\infty} \left(x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t + (A x_t + B u_t)^\mathsf{T} P (A x_t + B u_t) - x_t^\mathsf{T} P x_t \right) \\
&= x_0^\mathsf{T} P x_0 + \sum_{t=0}^{\infty} \left(x_t^\mathsf{T} (Q + A^\mathsf{T} P A - P) x_t + u_t^\mathsf{T} (R + B^\mathsf{T} P B) u_t + x_t^\mathsf{T} A^\mathsf{T} P B u_t + u_t^\mathsf{T} B^\mathsf{T} P A x_t \right)
\end{aligned}$$

By the Riccati equation, we have $Q + A^\mathsf{T} P A - P = A^\mathsf{T} P B (B^\mathsf{T} P B + R)^{-1} B^\mathsf{T} P A$. Hence we have

$$\begin{aligned}
&\sum_{t=0}^{\infty} (x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t) \\
=& x_0^\mathsf{T} P x_0 + \sum_{t=0}^{\infty} \left(x_t^\mathsf{T} A^\mathsf{T} P B (B^\mathsf{T} P B + R)^{-1} B^\mathsf{T} P A x_t + u_t^\mathsf{T} (R + B^\mathsf{T} P B) u_t + x_t^\mathsf{T} A^\mathsf{T} P B u_t + u_t^\mathsf{T} B^\mathsf{T} P A x_t \right) \\
=& x_0^\mathsf{T} P x_0 + \sum_{t=0}^{\infty} \left((B^\mathsf{T} P B + R) u_t + B^\mathsf{T} P A x_t \right)^\mathsf{T} (B^\mathsf{T} P B + R)^{-1} \left((B^\mathsf{T} P B + R) u_t + B^\mathsf{T} P A x_t \right)
\end{aligned}$$

Notice $B^\mathsf{T} P B + R$ is positive definite. Therefore, the cost achieves its minimal value $x_0^\mathsf{T} P x_0$ if the action is chosen to satisfy $(B^\mathsf{T} P B + R) u_t + B^\mathsf{T} P A x_t = 0$. We have skipped a few technical details but the above calculation roughly explain why the optimal policy is given by $K = (B^\mathsf{T} P B + R)^{-1} B^\mathsf{T} P A$.

[1] For simplicity, we assume Q is positive definite in this note. Hence the solution P is positive definite. In general, it is possible to allow Q to be only positive semidefinite and the solution P can also become positive semidefinite. Some extra observability assumption is then needed.

How does the Riccati equation arise? Usually one first looks at LQR on a finite horizon and then extends the result there to the infinite horizon setup. Instead of doing that, we will provide an alternative informal derivation that starts from the assumption that the optimal policy is linear. Under this assumption, the optimal value function is quadratic and takes the form of $x^\mathsf{T}Px$. The optimal Bellman equation states the following fact.

$$x^\mathsf{T}Px = \min_u (x^\mathsf{T}Qx + u^\mathsf{T}Ru + (Ax+Bu)^\mathsf{T}P(Ax+Bu)) \tag{1.7}$$

Clearly, the right side is a quadratic function in u. Taking the gradient of the right hand side with respect to u leads to the following equation:

$$(R + B^\mathsf{T}PB)u + B^\mathsf{T}PAx = 0$$

The optimal u is given as $u = -(R+B^\mathsf{T}PB)^{-1}B^\mathsf{T}PAx$. We can substitute this relation to the right side of (1.7) and obtain

$$\begin{aligned}P = Q &+ A^\mathsf{T}PB(R+B^\mathsf{T}PB)^{-1}R(R+B^\mathsf{T}PB)^{-1}B^\mathsf{T}PA \\ &+ \left(A - B(B^\mathsf{T}PB+R)^{-1}B^\mathsf{T}PA\right)^\mathsf{T} P \left(A - B(B^\mathsf{T}PB+R)^{-1}B^\mathsf{T}PA\right)\end{aligned}$$

Simplifying the above equation leads to the Riccati equation (1.6).

Key message. When writing out the optimal Bellman equation, we need to know how to parameterize the optimal value function. In the LQR problem, we know we can parameterize the optimal value function as a quadratic function. Therefore, eventually the optimal Bellman equation (1.7) is equivalent to an algebraic Riccati equation on P. If we just use a general value function $V(x)$, then (1.7) becomes

$$V(x) = \min_u (x^\mathsf{T}Qx + u^\mathsf{T}Ru + V(Ax+Bu)) \tag{1.8}$$

How to figure out V becomes a difficult task. For nonlinear control, the situation is even worse. Suppose the dynamics become $x_{t+1} = f(x_t, u_t)$ and the one-stage cost is $c(x,u)$ where c is some complicated function. Then the optimal Bellman equation is in the following nonlinear form:

$$V(x) = \min_u (c(x,u) + V(f(x,u))) \tag{1.9}$$

which is extremely difficult to solve. One has to approximate V using neural networks.

1.4 Value Iteration

Suppose we decide to parameterize the value function as $V(x) = x^\mathsf{T}Px$. The Bellman operator T maps P to P' as follows

$$x^\mathsf{T}P'x = \min_u (x^\mathsf{T}Qx + u^\mathsf{T}Ru + (Ax+Bu)^\mathsf{T}P(Ax+Bu)) \tag{1.10}$$

Clearly, the optimal Bellman equation (1.7) is equivalent to $P = T(P)$ and just gives the fixed point for the above Bellman operator. Again, the right side of (1.10) is a quadratic function of u, and minimizing over u gives

$$P' = A^\mathsf{T}PA + Q - A^\mathsf{T}PB(B^\mathsf{T}PB+R)^{-1}B^\mathsf{T}PA \tag{1.11}$$

Therefore, we can apply the Bellman operator recursively and obtain a value iteration algorithm:

$$P^{n+1} = A^\intercal P^n A + Q - A^\intercal P^n B(B^\intercal P^n B + R)^{-1} B^\intercal P^n A \tag{1.12}$$

We can clearly see, for the continuous state space case, we need to parameterize the value function and then iteratively update these parameters.

1.5 Policy Iteration

For policy iteration, we need to parameterize the policy and recursively update these policy parameters. Let's say we use a linear policy K. At step n, the policy update is K^n. Then we evaluate the value for this policy by solving the Bellman equation $(A-BK^n)^\intercal P^n(A-BK^n)+Q+(K^n)^\intercal RK^n = P^n$. Once P^n is obtained, we update $x = -K^{n+1}u$ by solving

$$\arg\min_u (x^\intercal Qx + u^\intercal Ru + (Ax+Bu)^\intercal P^n(Ax+Bu)) \tag{1.13}$$

Minimizing the above cost over u leads to $u = -(B^\intercal P^n B + R)^{-1} B^\intercal P^n A$, and hence we have $K^{n+1} = (B^\intercal P^n B+R)^{-1} B^\intercal P^n A$ where P_n solves $(A-BK^n)^\intercal P^n(A-BK^n)+Q+(K^n)^\intercal RK^n = P^n$.

Differences between PI and VI. In VI, the optimal u for (1.10) is substituted back to the Bellman operator to obtain P^{n+1}. In PI, P^{n+1} is solved by evaluating the value for this u (hence we need to solve a fixed point here).

1.6 State-Action Value Function: $\mathcal{Q}$-Factor

Given a fixed policy K, the state-action value function can be calculated as

$$\mathcal{Q}(x,u) = x^\top Qx + u^\top Ru + \sum_{t=1}^{\infty} x_t^\intercal (Q + K^\top RK)x_t$$

where $x_1 = Ax + Bu$. Notice the control actions at all steps except step $k=0$ are selected as $u_t = -Kx_t$. At $k=0$, the control action is u and independent of the policy K. This initial control action u is an input to the state-action value function.

Suppose $V(x)$ is the state value function. Then obviously, we have

$$\mathcal{Q}(x,u) = x^\top Qx + u^\top Ru + V(Ax+Bu) \tag{1.14}$$

In addition, we also have $V(x) = \mathcal{Q}(x, -Kx)$. Therefore, we can substitute this into the above equation to obtain the Bellman equation for $\mathcal{Q}$:

$$\mathcal{Q}(x,u) = x^\top Qx + u^\top Ru + \mathcal{Q}\left(Ax+Bu, -K(Ax+Bu)\right)$$

Suppose we know $\mathcal{Q}$ is a quadratic function of (x,u), and we parameterize the function as

$$\begin{aligned}
\begin{bmatrix} x \\ u \end{bmatrix}^\intercal \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\intercal & \mathcal{Q}_{22} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} &= \begin{bmatrix} x \\ u \end{bmatrix}^\intercal \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + \begin{bmatrix} Ax+Bu \\ -K(Ax+Bu) \end{bmatrix}^\intercal \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\intercal & \mathcal{Q}_{22} \end{bmatrix} \begin{bmatrix} Ax+Bu \\ -K(Ax+Bu) \end{bmatrix} \\
&= \begin{bmatrix} x \\ u \end{bmatrix}^\intercal \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + \begin{bmatrix} x \\ u \end{bmatrix}^\intercal \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix}^\intercal \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\intercal & \mathcal{Q}_{22} \end{bmatrix} \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}
\end{aligned}$$

Therefore, the Bellman equation for $\mathcal{Q}$-factor is just the following equation for $\mathcal{Q}_{11}$, $\mathcal{Q}_{12}$, and $\mathcal{Q}_{22}$.

$$\begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^{\mathsf{T}} & \mathcal{Q}_{22} \end{bmatrix} = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} + \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^{\mathsf{T}} & \mathcal{Q}_{22} \end{bmatrix} \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix} \tag{1.15}$$

The above equation is linear in $\mathcal{Q}_{11}$, $\mathcal{Q}_{12}$, and $\mathcal{Q}_{22}$. Hence, it can be solved by linear equation theory. We have

$$\operatorname{vec}\left(\begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^{\mathsf{T}} & \mathcal{Q}_{22} \end{bmatrix}\right) = \left(I - \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix}^{\mathsf{T}} \otimes \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix}^{\mathsf{T}}\right)^{-1} \operatorname{vec}\left(\begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix}\right) \tag{1.16}$$

Why is the quadratic parameterization of $\mathcal{Q}$-factor valid? Notice that we can obtain the $\mathcal{Q}$-factor using the value function directly. Suppose $V(x) = x^{\mathsf{T}} P x$. By (1.14), we can obtain

$$\begin{aligned} \mathcal{Q}(x,u) &= x^\top Q x + u^\top R u + (Ax+Bu)^{\mathsf{T}} P_K (Ax+Bu) \\ &= x^\top Q x + u^\top R u + \begin{bmatrix} x \\ u \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} A^{\mathsf{T}} P_K A & A^{\mathsf{T}} P_K B \\ B^{\mathsf{T}} P_K A & B^{\mathsf{T}} P_K B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \\ &= \begin{bmatrix} x \\ u \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} Q + A^{\mathsf{T}} P_K A & A^{\mathsf{T}} P_K B \\ B^{\mathsf{T}} P_K A & R + B^{\mathsf{T}} P_K B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \end{aligned}$$

Therefore, for the LQR problem, we can parameterize $\mathcal{Q}$-factor as a quadratic function. For general nonlinear control problems, we need to use more general function approximators such as neural networks.

Now it is obvious that the only solution for (1.16) is provided by

$$\begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^{\mathsf{T}} & \mathcal{Q}_{22} \end{bmatrix} = \begin{bmatrix} Q + A^{\mathsf{T}} P_K A & A^{\mathsf{T}} P_K B \\ B^{\mathsf{T}} P_K A & R + B^{\mathsf{T}} P_K B \end{bmatrix}.$$

One can also solve (1.16) using an iterative algorithm.

1.7 Optimal State-Action Value Function

By definition, we have $\mathcal{Q}^*(x,u) = \min_\pi \mathcal{Q}(x,u)$. Now we discuss various properties of $\mathcal{Q}^*$.

1. **Relation between $\mathcal{Q}^*$ and V^*:** For the LQR problem, the optimal state value function and the optimal state-action value function satisfy the following equation:

   $$\mathcal{Q}^*(x,u) = x^{\mathsf{T}} Q x + u^{\mathsf{T}} R u + V^*(Ax+Bu)$$

   Suppose the optimal state value function is $V^*(x) = x^{\mathsf{T}} P x$ where P is the stabilizing solution for the Riccati equation (1.6). Then we can solve $\mathcal{Q}^*$ by revoking its relation with V^*:

   $$\begin{aligned} \mathcal{Q}^*(x,u) &= x^{\mathsf{T}} Q x + u^{\mathsf{T}} R u + (Ax+Bu)^{\mathsf{T}} P (Ax+Bu) && (1.17) \\ &= \begin{bmatrix} x \\ u \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} Q + A^{\mathsf{T}} P A & A^{\mathsf{T}} P B \\ B^{\mathsf{T}} P A & R + B^{\mathsf{T}} P B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} && (1.18) \end{aligned}$$

2. **Obtaining Optimal Policy from $\mathcal{Q}^*$:** As discussed in the class, a big advantage of the state-action value function is that one can obtain the optimal policy from the optimal state-action value function directly without using the model information (A, B). Suppose we have a positive semidefinite quadratic optimal $\mathcal{Q}$-function:

$$\mathcal{Q}^*(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11}^* & \mathcal{Q}_{12}^* \\ (\mathcal{Q}_{12}^*)^\mathsf{T} & \mathcal{Q}_{22}^* \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

Minimizing the above quadratic function over u gives the optimal action $u^* = -(\mathcal{Q}_{22}^*)^{-1}(\mathcal{Q}_{12}^*)^\mathsf{T} x$. Hence the optimal policy is $K^* = -(\mathcal{Q}_{22}^*)^{-1}(\mathcal{Q}_{12}^*)^\mathsf{T}$. One can clearly see that K^* can be directly calculated from $\mathcal{Q}$. As a sanity check, from (1.17) we know $\mathcal{Q}_{22}^* = R + B^\mathsf{T} P B$ and $\mathcal{Q}_{12}^* = A^\mathsf{T} P B$. Hence , we have $K^* = -(Q + K^\mathsf{T} R K)^{-1} B^\mathsf{T} P A$, which is consistent with the result in Section 1.3.

3. **Optimal Bellman Equation:** The optimal Bellman equation for $\mathcal{Q}^*$ is formulated as

$$\begin{aligned} \mathcal{Q}^*(x,u) &= x^\mathsf{T} Q x + u^\mathsf{T} R u + \min_{u'} \mathcal{Q}^*(Ax + Bu, u') \\ &= x^\mathsf{T} Q x + u^\mathsf{T} R u + \mathcal{Q}^* \Big(Ax + Bu, -(\mathcal{Q}_{22}^*)^{-1}(\mathcal{Q}_{12}^*)^\mathsf{T}(Ax + Bu) \Big) \end{aligned}$$

which eventually leads to the following equation:

$$\begin{bmatrix} \mathcal{Q}_{11}^* & \mathcal{Q}_{12}^* \\ (\mathcal{Q}_{12}^*)^\mathsf{T} & \mathcal{Q}_{22}^* \end{bmatrix} = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} +$$
$$\begin{bmatrix} A & B \\ -(\mathcal{Q}_{22}^*)^{-1}(\mathcal{Q}_{12}^*)^\mathsf{T} A & -(\mathcal{Q}_{22}^*)^{-1}(\mathcal{Q}_{12}^*)^\mathsf{T} B \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11}^* & \mathcal{Q}_{12}^* \\ (\mathcal{Q}_{12}^*)^\mathsf{T} & \mathcal{Q}_{22}^* \end{bmatrix} \begin{bmatrix} A & B \\ -(\mathcal{Q}_{22}^*)^{-1}(\mathcal{Q}_{12}^*)^\mathsf{T} A & -(\mathcal{Q}_{22}^*)^{-1}(\mathcal{Q}_{12}^*)^\mathsf{T} B \end{bmatrix}$$

4. **Value Iteration for $\mathcal{Q}$-Factor:** The algorithm is initialized at $(\mathcal{Q}_{11}^0, \mathcal{Q}_{12}^0, \mathcal{Q}_{22}^0)$ and then iterated as:

$$\begin{bmatrix} \mathcal{Q}_{11}^{n+1} & \mathcal{Q}_{12}^{n+1} \\ (\mathcal{Q}_{12}^{n+1})^\mathsf{T} & \mathcal{Q}_{22}^{n+1} \end{bmatrix} = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} +$$
$$\begin{bmatrix} A & B \\ -(\mathcal{Q}_{22}^n)^{-1}(\mathcal{Q}_{12}^n)^\mathsf{T} A & -(\mathcal{Q}_{22}^n)^{-1}(\mathcal{Q}_{12}^n)^\mathsf{T} B \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11}^n & \mathcal{Q}_{12}^n \\ (\mathcal{Q}_{12}^n)^\mathsf{T} & \mathcal{Q}_{22}^n \end{bmatrix} \begin{bmatrix} A & B \\ -(\mathcal{Q}_{22}^n)^{-1}(\mathcal{Q}_{12}^n)^\mathsf{T} A & -(\mathcal{Q}_{22}^n)^{-1}(\mathcal{Q}_{12}^n)^\mathsf{T} B \end{bmatrix}$$

5. **Policy Iteration for $\mathcal{Q}$-Factor:** At every step n, we first solve the $\mathcal{Q}$ Bellman equation to obtain $(\mathcal{Q}_{11}^n, \mathcal{Q}_{12}^n, \mathcal{Q}_{22}^n)$ (the policy evaluation step), and then update the policy as $K_{n+1} = -(\mathcal{Q}_{22}^n)^{-1}(\mathcal{Q}_{12}^n)^\mathsf{T}$ (the policy improvement step).

2 LQR with Process Noise

In this case, the dynamics become

$$x_{t+1} = Ax_t + Bu_t + w_t \tag{2.1}$$

where w_t is an IID Gaussian noise. Given any x_0, the state x_t does not converge to 0 and the cost in (1.2) is not finite. A fix is to use the discounting factor. Now we consider the cost function

$$\mathcal{C} = \mathbb{E} \sum_{t=0}^{\infty} \gamma^t (x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t) \tag{2.2}$$

where $0 < \gamma < 1$ is the discounting factor. Again, this is a MDP problem. A key fact is that the probability distribution of x_{t+1} is completely known if x_t and u_t are seen. This is due to the IID nature of w_t. When w_t is Gaussian, the transition density will also be Gaussian.

Now we briefly highlight some differences introduced by this noise term.

1. Policy evaluation: Given a linear policy K, the cost becomes

$$\mathcal{C}(K) = r_K + \mathbb{E}_{x_0 \sim \mathcal{D}} \quad x_0^\mathsf{T} \left(\sum_{t=0}^{\infty} \gamma^t ((A - BK)^\mathsf{T})^t (Q + K^\mathsf{T} R K)(A - BK)^t \right) x_0 \tag{2.3}$$

   where r_K is some extra term introduced by the noise w_t. Therefore, we can parameterize the value function as $x^\mathsf{T} P_K x + r_K$. Then the Bellman equation becomes

$$P_K = Q + K^\mathsf{T} R K + \gamma (A - BK)^\mathsf{T} P_K (A - BK)$$

2. Optimal Bellman equation: The Riccati equation now depends on γ and becomes

$$P = Q + \gamma A^\mathsf{T} P A - \gamma^2 A^\mathsf{T} P B (R + \gamma B^\mathsf{T} P B)^{-1} B^\mathsf{T} P A$$

   The optimal policy becomes $u_t = -\gamma(R + \gamma B^\mathsf{T} P B)^{-1} B^\mathsf{T} P A$.

3. Value iteration: Algorithm (1.12) should be modified as

$$P^{n+1} = \gamma A^\mathsf{T} P^n A + Q - \gamma^2 A^\mathsf{T} P^n B (\gamma B^\mathsf{T} P^n B + R)^{-1} B^\mathsf{T} P^n A \tag{2.4}$$

4. Policy iteration: In this case, the PI algorithm iterates as $K^{n+1} = \gamma(\gamma B^\mathsf{T} P^n B + R)^{-1} B^\mathsf{T} P^n A$ where P_n solves the Bellman equation $\gamma(A - BK^n)^\mathsf{T} P^n (A - BK^n) + Q + (K^n)^\mathsf{T} R K^n = P^n$.

5. $\mathcal{Q}$-Learning: Similar results can be obtained for $\mathcal{Q}$-factors. The main difference is that there is a residue term in $\mathcal{Q}$-factor and the Bellman equation involves the discounting factor γ.

You should try to derive the above results yourself! All you need to do is to modify the derivations in Section 1.

2.1 Implementation of LSTD-$\mathcal{Q}$

More details on the LSTD-$\mathcal{Q}$ algorithm is provided to help you finish homework.

1. LSTD for state value function: Let's consider a 2D example. Suppose $x = \begin{bmatrix} a \\ b \end{bmatrix}$ and u_t is a scalar. The feature can be calculated as

$$\phi(x) = \begin{bmatrix} a^2 \\ ab \\ b^2 \\ 1 \end{bmatrix}$$

The state value function is parameterized as

$$V(x) = \theta^\top \phi(x) = \begin{bmatrix} p_1 & p_2 & p_3 & r \end{bmatrix} \begin{bmatrix} a^2 \\ ab \\ b^2 \\ 1 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}^\top \begin{bmatrix} p_1 & \frac{1}{2}p_2 \\ \frac{1}{2}p_2 & p_3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} + r$$

Now we want to fit $\{p_1, p_2, p_3, r\}$ from sampled data. We just generate a trajectory of $\{x_t\}_{t=0}^T$ using $x_{k+1} = (A - BK)x_k + w_k$. We fit θ to minimize the target difference error. Specifically, we have

$$\left(\sum_{t=0}^{T-1} \phi(x_t)(\phi(x_t) - \gamma\phi(x_{t+1}))^\top \right) \theta \approx \sum_{t=0}^{T-1} c(x_t)\phi(x_t)$$

where $c(x_t) = x_t^\top (Q + K^\top RK)x_t$. So we just estimate θ as

$$\theta \approx \left(\sum_{t=0}^{T-1} \phi(x_t)(\phi(x_t) - \gamma\phi(x_{t+1}))^\top \right)^{-1} \left(\sum_{t=0}^{T-1} c(x_t)\phi(x_t) \right)$$

Sometimes the inverse does not exist and we can replace it with pseudo inverse. In `Matlab`, you can use `pinv` to calculate pseudo inverse. The above implementation requires the noise w_t to provide sufficient exploration. If the noise is too small, it will require a huge T to make the above implementation work. In this case, you can use another implementation. Generate x_t randomly for all t using some distribution which explores the state space thoroughly. For example, use a uniform distribution over $[-1000, 1000]$. For all t, generate x'_t as $x'_t = (A - BK)x_t + w_t$. Now estimate θ as

$$\theta \approx \left(\sum_{t=0}^{T} \phi(x_t)(\phi(x_t) - \gamma\phi(x'_t))^\top \right)^{-1} \left(\sum_{t=0}^{T} c(x_t)\phi(x_t) \right)$$

You can try different distribution to generate x_t to see which one explores best.

2. LSTD for $\mathcal{Q}$ function: Again, we demonstrate the algorithm for 2D case. Suppose $x = \begin{bmatrix} a \\ b \end{bmatrix}$ and u_t is a scalar. The feature can be calculated as

$$\phi(x, u) = \begin{bmatrix} a^2 \\ ab \\ b^2 \\ au \\ bu \\ u^2 \\ 1 \end{bmatrix}$$

The $\mathcal{Q}$ function can be parameterized as

$$\mathcal{Q}(x,u) = \theta^\top \phi(x,u) = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & r \end{bmatrix} \begin{bmatrix} a^2 \\ ab \\ b^2 \\ au \\ bu \\ u^2 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} a \\ b \\ u \end{bmatrix}^\top \begin{bmatrix} q_1 & \frac{1}{2}q_2 & \frac{1}{2}q_4 \\ \frac{1}{2}q_2 & q_3 & \frac{1}{2}q_5 \\ \frac{1}{1}q_4 & \frac{1}{2}q_5 & q_6 \end{bmatrix} \begin{bmatrix} a \\ b \\ u \end{bmatrix} + r$$

Now we need to fit a 7-dimensional vector θ. We just generate a trajectory of $\{x_t, u_t\}_{t=0}^T$ using $x_{k+1} = Ax_k + Bu_k + w_k$ and $u_k = -Kx_k + v_k$. Here v_k is some noise added for exploration. We fit θ to minimize the target difference error as

$$\theta \approx \left(\sum_{t=0}^{T-1} \phi(x_t, u_t)(\phi(x_t, u_t) - \gamma \phi(x_{t+1}, -Kx_{t+1}))^\top \right)^{-1} \left(\sum_{t=0}^{T-1} c(x_t, u_t)\phi(x_t, u_t) \right)$$

Notice typically we do **not** estimate θ as

$$\theta \approx \left(\sum_{t=0}^{T-1} \phi(x_t, u_t)(\phi(x_t, u_t) - \gamma \phi(x_{t+1}, u_{t+1}))^\top \right)^{-1} \left(\sum_{t=0}^{T-1} c(x_t, u_t)\phi(x_t, u_t) \right)$$

The above estimation only works when the exploration noise is some reasonable Gaussian noise. For general exploration noise v_k, we use the estimation

$$\theta \approx \left(\sum_{t=0}^{T-1} \phi(x_t, u_t)(\phi(x_t, u_t) - \gamma \phi(x_{t+1}, -Kx_{t+1}))^\top \right)^{-1} \left(\sum_{t=0}^{T-1} c(x_t, u_t)\phi(x_t, u_t) \right)$$

Notice v_k should be large enough to explore the space thoroughly. For our problem, you can start by trying uniform noise from $[-500, 500]$ or Gaussian noise $\mathcal{N}(0, 100I)$. You should try various exploration noise to figure out which one works best. Similarly, you can also generate (x_t, u_t) completely randomly for all t. Here u_t can be completely random, and does not need to be generated from policy K. For example, use a uniform distribution over $[-1000, 1000]$ to generate (x_t, u_t). For all t, generate x'_t as $x'_t = Ax_t + Bu_t + w_t$. now estimate θ as

$$\theta \approx \left(\sum_{t=0}^{T} \phi(x_t, u_t)(\phi(x_t, u_t) - \gamma \phi(x'_t, -Kx'_t))^\top \right)^{-1} \left(\sum_{t=0}^{T} c(x_t, u_t)\phi(x_t, u_t) \right)$$

The above implementation is super efficient for policy iteration. You just generate the sampled data $\{x_t, u_t, x'_t\}$ once. Then for all K, you can use the same data to estimate θ by renewing the calculations of $\phi(x'_t, -Kx'_t)$. This makes LSPI sample efficient.