ECE 598: Interplay between Control and Machine Learning **Fall 2020**

Lecture 10

Q-Factors and Policy Iteration

Lecturer: Bin Hu, Date:10/06/2020

In this lecture, we discuss the state-action value function (which is also called Q-function).

10.1 Discrete Space Case

Recall that a MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition kernel, R is the reward, and γ is the discount factor. Suppose both $\mathcal{S}$ and $\mathcal{A}$ are finite. Given a policy π, the state-action value function is defined as

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \big| a_k \sim \pi(\cdot|s_k)\, \forall k \geq 1, s_0 = s,\, a_0 = a\right].$$

For a deterministic policy π, the Q-function satisfy

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \big| s_0 = s,\, a_0 = a,\, a_k = \pi(s_k) \forall k \geq 1\right].$$

Now we can apply the law of total probability to show:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}R(s_0,a_0) + \mathbb{E}\left[\sum_{k=1}^{\infty} \gamma^k R(s_k, \pi(s_k)) \big| s_0 = s,\, a_0 = a\right] \\ &= \mathbb{E}R(s,a) + \sum_{s' \in \mathcal{S}} \left(\mathbb{E}[\sum_{k=1}^{\infty} \gamma^k R(s_k, \pi(s_k)) \big| s_1 = s'] \right) P(s_1 = s' \big| s_0 = s,\, a_0 = a) \end{aligned}$$

which can be rewritten as

$$Q^\pi(s,a) = \bar{R}^\pi(s,a) + \gamma \sum_{s' \in \mathcal{S}} Q^\pi(s', \pi(s')) P^a_{ss'} \tag{10.1}$$

This is just the Bellman equation for Q-factor. Similarly, when π is stochastic, we have

$$Q^\pi(s,a) = \bar{R}^\pi(s,a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s', a') P^a_{ss'} \tag{10.2}$$

which could also be cast as a linear equation

$$Q^\pi = \bar{R}^\pi + \gamma \bar{P}^\pi Q^\pi \tag{10.3}$$

Therefore, the Bellman equation for Q-factor is also linear in this case.

Connections between V^π and Q^π. If we know V^π, we can calculate Q^π as

$$Q^\pi(s,a) = \bar{R}^\pi(s,a) + \gamma \sum_{s' \in \mathcal{S}} V^\pi(s') P^a_{ss'} \tag{10.4}$$

If we know Q^π, we can calculate V^π as $V^\pi(s) = Q(s, \pi(s))$ (the deterministic policy case) or $V^\pi(s) = \sum_{a \in \mathcal{A}} Q(s,a)\pi(a|s)$ (the stochastic policy case).

Optimal Q function. Now suppose the optimal policy is π^*. Let the state-action value function of π^* be Q^*, which is the optimal state-action value function. Clearly, Q^* can be calculated from V^* as

$$Q^*(s,a) = \bar{R}^\pi(s,a) + \gamma \sum_{s' \in \mathcal{S}} V^*(s') P^a_{ss'} \tag{10.5}$$

From this, we can immediately see that a deterministic optimal policy can be given as

$$\begin{aligned}
\pi^*(s) &= \arg\max_{a \in \mathcal{A}} \left(\bar{R}^\pi(s,a) + \gamma \sum_{s' \in \mathcal{S}} V^*(s') P^a_{ss'} \right) \\
&= \arg\max_{a \in \mathcal{A}} Q^*(s,a)
\end{aligned}$$

Therefore, once Q^* is known, we can obtain an optimal policy without knowing the model. The above fact also states that $V^*(s) = \max_{a \in \mathcal{A}} Q(s,a)$. In general, if the state-action value function is known, the state value function can be calculated without knowing the model. We can substitute $V^*(s') = \max_{a' \in \mathcal{A}} Q(s', a')$ into (10.5) to obtain the optimal Bellman equation for Q-factor:

$$Q^*(s,a) = \bar{R}^\pi(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} \max_{a' \in \mathcal{A}} Q^\pi(s', a') \tag{10.6}$$

Now recall that the optimal Bellman equation for V^* is

$$V^*(s) = \max_{a \in \mathcal{A}} \left[\bar{R}(s,a) + \gamma \sum_{s'} P^a_{ss'} V^*(s') \right] \tag{10.7}$$

From the comparison, we can see that in the optimal Bellman equation for Q-factor, one first applies maximization and then does some average. For V^*, one first averages things and then apply maximization. This difference is important for the development of value-based RL methods.

Why is Q-factor useful? One big advantage is that once Q^* is known, one can calculate the optimal policy without knowing the model. Another big advantage is related to the development of the policy iteration method.

Policy iteration. One form of policy iteration works as follows. For every iteration k, one first evaluates the Q-function of the policy π_k. Then obtain a new policy as $\pi_{k+1}(s) = \max_{a\in\mathcal{A}} Q^{\pi_k}(s,a)$. Then evaluate the Q function for π_{k+1} and regenerate new policies by greedily planning according to the new Q functions. If we have a data driven method to estimate Q for any given policy, then we can implement the policy iteration method in a model free manner. This is called approximate policy iteration (API).

Q-Learning. Instead of doing policy iteration, one can also try to estimate Q^* from data by extending the idea of value iteration. We will talk about this later.

10.2 Continuous Space Case

10.2.1 LQR without Process Noise

For simplicity, again let's first consider the LQR setup:

$$x_{t+1} = Ax_t + Bu_t \tag{10.8}$$

The goal is to design a controller to minimize the quadratic cost.

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

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

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

$$\mathcal{Q}(x,u) = x^\top Qx + u^\top Ru + V(Ax + Bu) \tag{10.9}$$

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

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

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

$$\begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22} \end{bmatrix} = \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} + \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22} \end{bmatrix} \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix} \tag{10.10}$$

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

$$\text{vec}\left(\begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12} \\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22} \end{bmatrix}\right) = \left(I - \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix}^\mathsf{T} \otimes \begin{bmatrix} A & B \\ -KA & -KB \end{bmatrix}^\mathsf{T}\right)^{-1} \text{vec}\left(\begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix}\right) \tag{10.11}$$

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 $V(x) = x^\mathsf{T} Px$. By (10.9), we can obtain

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

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 (10.11) 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 (10.11) using an iterative algorithm.

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:

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

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

   $$\mathcal{Q}^*(x,u) = x^\mathsf{T} Qx + u^\mathsf{T} Ru + (Ax+Bu)^\mathsf{T} P (Ax+Bu) \tag{10.12}$$

   $$= \begin{bmatrix} x \\ u \end{bmatrix}^\mathsf{T} \begin{bmatrix} Q + A^\mathsf{T} PA & A^\mathsf{T} PB \\ B^\mathsf{T} PA & R + B^\mathsf{T} PB \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{10.13}$$

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 (10.12) we know $\mathcal{Q}_{22}^* = R + B^\mathsf{T} PB$ and $\mathcal{Q}_{12}^* = A^\mathsf{T} PB$. Hence , we have $K^* = -(Q + K^\mathsf{T} RK)^{-1} B^\mathsf{T} PA$, which is consistent with the result in the last lecture.

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

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

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).

10.2.2 LQR with Process Noise

In this case, the dynamics become

$$x_{t+1} = Ax_t + Bu_t + w_t \tag{10.14}$$

where w_t is an IID Gaussian noise. Given a policy K, we can calculate the $\mathcal{Q}$-function as

$$\begin{aligned}
\mathcal{Q}(x,u) &= x^\top Qx + u^\top Ru + \gamma\mathbb{E}V(Ax+Bu+w)\\
&= x^\top Qx + u^\top Ru + \gamma\mathbb{E}(Ax+Bu+w)^\mathsf{T} P_K(Ax+Bu+w) + \gamma r_K\\
&= x^\top Qx + u^\top Ru + \gamma(Ax+Bu)^\mathsf{T} P_K(Ax+Bu) + \gamma(\mathbb{E}(w^\mathsf{T} P_K w) + r_K)\\
&= \begin{bmatrix} x\\ u\end{bmatrix}^\mathsf{T} \begin{bmatrix} Q+\gamma A^\mathsf{T} P_K A & \gamma A^\mathsf{T} P_K B\\ \gamma B^\mathsf{T} P_K A & R+\gamma B^\mathsf{T} P_K B\end{bmatrix} \begin{bmatrix} x\\ u\end{bmatrix} + r_K
\end{aligned}$$

We can also directly parameterize $\mathcal{Q}(x,u)$ as

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

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

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

which is equivalent to

$$\begin{aligned}
\begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12}\\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22}\end{bmatrix} &= \begin{bmatrix} Q & 0\\ 0 & R\end{bmatrix} + \gamma \begin{bmatrix} A & B\\ -KA & -KB\end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12}\\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22}\end{bmatrix} \begin{bmatrix} A & B\\ -KA & -KB\end{bmatrix}\\
r_K &= \gamma\mathbb{E}\begin{bmatrix} w\\ -Kw\end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathcal{Q}_{11} & \mathcal{Q}_{12}\\ \mathcal{Q}_{12}^\mathsf{T} & \mathcal{Q}_{22}\end{bmatrix} \begin{bmatrix} w\\ -Kw\end{bmatrix} + \gamma r_K
\end{aligned}$$

The PI algorithm is the same. 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).

Therefore, if we have a data-driven method to estimate the $\mathcal{Q}$ function for any stabilizing policy, then we can implement PI in a model-free manner. One way to estimate $\mathcal{Q}$ function from data is to apply the LSTD-Q. Just to give you a rough idea, here is how LSTD-Q works in the 2D case.

LSTD algorithm for estimating $\mathcal{Q}$ from data. 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.