ESE 680-004: Learning and Control **Fall 2019**

Lecture 19: Q-learning for LQR

Lecturer: Nikolai Matni *Scribe: Raphael Van Hoffelen*

In the previous lectures, we have talked about the model-based control approach of learned systems. But what learning methods and algorithms can be used in a model-free control approach? In this lecture, we will see how we can use reinforcement learning techniques such as Q-Learning , TD-Learning and Policy Iteration for LQR problems.

1 Stochastic Dynamic Programming

Let's consider the following dynamical system:

$$x_{t+1} = f_t\left(x_t, u_t, w_t\right), t = 0, 1, \ldots, N-1$$

We denote the cost per stage by $c_t(x_t, u_t)$ and assume that $x_t \in \mathcal{X}_t$ and $u_t \in \mathcal{U}_t$. Our task in this problem, is to optimize over all polices $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ such that $u_t = \mu_k(x_t) \in \mathcal{U}_t$.

We start by defining the expected cost incurred by a policy π starting at state x_0 such that

$$J_\pi(x_0) = \mathbb{E}\left\{c_N(x_N) + \sum_{t=0}^{N-1} c_t(x_t, \mu_t(x_t))\right\},$$

where the expectation cost is taken over $\{w_t, x_t, \mu_t\}$. We can now define a optimal policy $\pi_\star$ that minimizes the cost function $J_{\pi_\star}(x_0)$ for all $\pi \in \Pi$ at state x_0. In other words, $\pi_\star$ can be define such that $J_{\pi_\star}(x_0) = \min_{\pi\in\Pi} J_\pi(x_0)$. We will now denote this cost with $J^\star$ and use $J^\star(x_0)$ as the optimal cost-to-go function that uses the optimal policy $\pi_\star$ at state x_0.

Now that our system is set up, we will lay out the steps of the dynamic programming approach to solve stochastic finite horizon problems and find the optimal policy $\pi_\star$.

- The first step consists to initialize our optimal cost function from the "bottom up" such that $J_N^\star(x_N) = c_N(x_N)$ where $c_N(x_N)$ is the cost of our system at the final state x_N.
- We then iterate through all states in reverse order such that $t = N-1, N-2, \ldots, 1, 0.$ at each new state x_t, we can calculate the optimal cost function $J_t^\star(x_t)$ by minimizing the expected sum of the cost current state $c_t(x_t, u_t)$ with the optimal cost $J_{t+1}^\star(f_t(x_t, u_t, w_t)$ at time $t+1$ for all $u_t \in U_t$. In other words:

$$\text{for } t = N-1, N-2, \ldots, 1, 0. \text{ let } J_t^\star(x_t) = \min_{u_t\in\mathcal{U}_t} \mathbb{E}\left\{c_t(x_t, u_t) + J_{t+1}^\star(f_t(x_t, u_t, w_t))\right\} \tag{1}$$

- Finally if all optimal $u_t^\star = \mu_t^\star(x_t)$ minimizes the right-hand-side of equation (1) for each state x_t and time t, then $\pi_\star = \{\mu_0^\star, \ldots, \mu_{N-1}^\star\}$ is optimal and $J_0^\star(x_0)$ is the optimal cost to go $J^\star(x_0)$.

2 Approximate Dynamic Programming

There are 2 main implementation of the dynamic programming method described above.
The first implementation consists in computing the optimal cost-to-go functions $J_k^\star$ and policies $\mu_k^\star$ ahead of time and store them in look-up-tables. This puts all the compute power in advance and allows for a fast inexpensive run time.
The second implementation describes a method to use in "real-time". It consists in computing the dynamic programming recursion online using a 1-step look-ahead. By doing this you only need to compute the cost-to-go $J_t^\star(x_t)$ for the N states seen during execution.

Unfortunately, most real world applications requires the computation to be done in "real-time" but the second implementation discussed ends up being too computationally expensive. This is why approximation techniques are usually used to trade off optimally for speed.
One common approach consists of conducting approximations in value space. This means that at any state x_t encountered at time t compute and apply the following equation:

$$\tilde{\mu}(x_t) \in \arg\min_{u_t \in \mathcal{U}_t} \mathbb{E}\left\{c_t(x_t, u_t) + \tilde{J}_{t+1}(f_t(x_t, u_t, w_t))\right\}$$

3 Q-Functions or Q-Factors

Dynamic programming can also be defined using Q-factors (or Q-Value). A Q-Factor is define by a Q-Function that takes in a state/action pair and calculates the expected sum of the current state/action cost and optimal cost-to-go function $J_{t+1}^\star$ at time $t+1$. Each state has a Q-factor for each action that can be taken at that state. We can therefore define the optimal Q-Factor with the following equation:

$$Q_t^\star(x_t, u_t) = \mathbb{E}\left\{c_t(x_t, u_t) + J_{t+1}^\star(f_t(x_t, u_t, w_t))\right\} \tag{2}$$

Looking back at equation (1), we can see that equation (2) can substituted be in. We now have $J_t^\star(x_t)$ defined by $Q_t^\star(x_t, u_t)$. This gives us the following set of equations:

$$J_t^\star(x_t) = \min_{u_t \in \mathcal{U}_t} Q_t^\star(x_t, u_t) \tag{3}$$

with optimal policy $\mu_k^\star$ defined as:

$$\mu_k^\star(x_t) = \arg\min_{u_t \in U_t} Q_t^\star(x_t, u_t) \tag{4}$$

We can now use the Q-Factor dynamic programming algorithm to find $Q_t^\star(x_t, u_t)$.

- First initialize $Q_N^\star(x_N, u_N) = c_N(x_N)$ (Cost of the last state at $t = N$)
- then solve the backwards recursion define by this equation:

$$Q_t^\star\left(x_t, u_t\right) = \mathbb{E}\left\{c_t\left(x_t, u_t\right) + \min_{u_{t+} \leq \mathcal{U}_{t+1}} Q_{t+1}^\star\left(f_t\left(x_t, u_t, w_t\right), u_{t+1}\right)\right\} \tag{5}$$

4 Discounted Problems and TD-Learning

The problem with the methods defined earlier is that they do not work for infinite horizon problems. In order to solve this issue with minimal technical overhead we can introduce a discounted cost $\gamma \in [0, 1]$ in our algorithms. this allows use to disregard state cost as t moves towards ∞. This is shown in the following equation:

$$J_\pi(x_0) = \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t c(x_t, \mu(x_t))\right\} \tag{6}$$

Where the cost function $c(c_t, \mu(x_t))$ converges to 0 exponentially with respect to γ^t as t moves towards ∞.

Now that we have a way to define infinite horizon problems for dynamic programming, we can use the Temporal Difference (TD) learning to approximate the cost-to-go function from data. We will define $\hat{J}_\pi(x_t)$ as the current estimate of the cost-to-go function at state x_t. TD-learning proposes the following update with $\alpha > 0$:

$$\hat{J}_\pi(x_t) \leftarrow (1-\alpha)\hat{J}_\pi(x_t) + \alpha\left[c(x_t, \mu(x_t)) + \gamma \hat{J}_\pi(x_{t+1})\right] \tag{7}$$

5 Q-Learning: TD-Learning for Q-Factors

We now know how TD-Learning update works for the cost-to-go function, and we know how to define $J_t^\star(x_t)$ with respect to $Q_t^\star(x_t, u_t)$. The next step will be to introduce a discount cost in the equation (2) in order for us to use TD-Learning with Q-Factors.

Similarly to how we did for equation (6), we can introduce a discounted cost $\gamma \in [0, 1]$ in our equation (2). By doing so we are left with the following equation:

$$Q^\star(x, u) = \mathbb{E}\left\{c(x, u) + \gamma \min_{u' \in \mathcal{U}} Q^\star(f(x, u, w), u')\right\} \tag{8}$$

We can now finally use the TD-Learning update with $\alpha > 0$ to estimate the current Q-function:

$$\hat{Q}(x_t, u_t) \leftarrow (1-\alpha)\hat{Q}(x_t, u_t) + \alpha\left[c(x_t, u_t) + \gamma \min_{u' \in \mathcal{U}} \hat{Q}(x_{t+1}, u')\right] \tag{9}$$

TD-Learning on cost-to-go functions and Q-functions are shown to converge in the tabular discounted settings [1] [2]. But convergence and optimality proofs using the function approximations $\hat{J}_\pi(x_t)$ and $\hat{Q}(x_t, u_t)$ harder to prove [3].

6 Adaptive LQR using Policy Iteration

The first convergence results for DP-based RL algorithms for continuous control problems was first proved in the paper entitled "Adaptive linear quadratic control using policy iteration" [2]. The paper does so by using recursive least-square for Q-learning combined with policy iteration and strongly exploits the prior knowledge that Q-functions are quadratic [1]. The results of the paper shows asymptotic convergence for modern (non-asymptotic) treatment [2].

6.1 Problem Setup

Lets consider some system dynamics described by

$$x_{t+1} = Ax_t + Bu_t = f(x_t, u_t),\ u_t = Kx_t,$$

$A + BK$ is stable with per-stage cost $c(x_t, u_t) = x_t^\top S x_t + u_t^\top R u_t$.

Using section 4 we can define the discounted cost-to-go function for control policy K beginning at time t from state x_t to be $V_K(x_t) := \sum_{i=0}^{\infty} \gamma^i c(x_{t+i}, u_{t+i})$ for $\gamma \in [0,1]$. From control theory, we know that $V_K(x_t) = x_t^\top P_K x_t$ for $P_K \succ 0$.. We now denote $K^\star$ as the optimal controller and $P^\star$ as the optimal cost matrix. By substituting $V_K(x_t)$ as our cost-to-go function in equation(8) we now have a new set of equations to define our Q-Function in relation to our system dynamics set up:

$$Q_K(x,u) = c(x,u) + \gamma V_K(f(x,u)) \tag{10}$$

$$Q_K\left(x_t, u_t\right) = c\left(x_t, u_t\right) + \gamma Q_K\left(x_{t+1}, K x_{t+1}\right) \tag{11}$$

6.2 Exploiting Quadratic Structure

Q-Functions have a nice quadratic structure [2]. This enable us to simplify our problem by rewriting the the Q-Function as

$$Q_K(x,u) = [x;u]^\top \begin{bmatrix} Q_K(1,1) & Q_K(1,2) \\ Q_K(1,2)^\top & Q_K(2,2) \end{bmatrix} [x;u], \quad \begin{array}{l} Q_K(1,1) = S + \gamma A^\top P_K A \\ Q_K(1,2) = \gamma A^\top P_K B \\ Q_K(2,2) = R + \gamma B^\top P_K B \end{array} \tag{12}$$

It is easier now to take a standard policy iteration approach as defined in [1] to find an improved policy. To do so we must first fix a policy K_k and cost-to-go function J_k where k denotes the Dynamic Programming iterations. We then can find the improved policy of our system via the following recursive algorithm:

$$K_{k+1}x = \arg\min_u \left[c(x,u) + \gamma J_K(f(x,u))\right] = \arg\min_u Q_K(x,u) \tag{13}$$

This leads us to set $\nabla_u Q_K(x,u) = 0$ and solve for u which yields the following equation:

$$u = -\gamma(R + \gamma B^\top P_{K_k} B)^{-1} B^\top P_{K_k} A x =: K_{k+1} x \tag{14}$$

Finally, via pattern matching we can conclude that:

$$K_{k+1} = -Q_K(2,2)^{-1} Q_K(1,2)^\top \tag{15}$$

We can make few observations from the equations defined above. We can see that if we assume that we begin at a stabilizing policy, then K_{k+1} must also be stabilizing as the cost can only decrease by running policy iteration. The leads to a new Q-function that can then be learned while repeating the process over. This shows us that proving convergence is reduced to analyzing the direct estimation of Q-functions and the convergence of adaptive policy iteration for LQR. Finally we also see that the discount factor changes the traditional meaning of stability in control theory to finite cost. More work is therefore needed to ensure actual stability of our dynamic system.

6.3 Direct Estimation of the Q-Function

As explained in section 6.2, Q-Factors are quadratic in $[x;u]$. This means that they can be made linear in the right basis and therefore be reduced to a least-squares problem.

Let $\bar{x} = [x_1^2, \ldots, x_1 x_n, x_2^2, \ldots, x_2 x_n, \ldots, x_n^2]^\top$ and define $\Theta(P)$ such that $x^\top P x = \bar{x}\Theta(P)$. Note that Θ is invertible when restricted to symmetric matrices. We can now rewrite equation (12) as:

$$Q_K(x,u) = [x;u]^\top Q_K [x;u] = \overline{[x;u]}^\top \Theta(Q_K) \tag{16}$$

Consequently we can plug in this equation in equation (11) in order to get:

$$\begin{aligned} c(x_t, u_t) &= Q_K(x_t, u_t) - \gamma Q_K(x_{t+1}, K x_{t+1}) \\ &= \left(\overline{[x_t; u_t]}^\top - \gamma \overline{[x_{t+1}; K x_{t+1}]}^\top\right) \Theta(Q_K) \\ &=: \phi_t^\top \theta_K \end{aligned} \tag{17}$$

Now we can use the recursive least-square method on our dynamic system with both i and t denote time and k denoting the Dynamic Programming step by substituting the equations we defined above. This gives us the following values.

- error update: $e_k(i) = c(x_t, u_t) - \phi_t^\top \hat{\theta}_k(i-1)$;
- estimate update: $\hat{\theta}_k(i) = \hat{\theta}_k(i-1) + \frac{\Sigma_k(i-1)\phi_t e_k(i)}{1+\phi_t^\top \Sigma_k(i-1)\phi_t}$;
- covariance update: $\Sigma_k(i) = \Sigma_k(i-1) - \frac{\Sigma_k(i-1)\theta_t\theta_t^\top \Sigma_k(i-1)}{1+\phi_t^\top \Sigma_k(i-1)\phi_t}$;
- covariance initialization: $\Sigma_k(0) = \Sigma_0 = \beta I \gg I$;

This recursive least-square is guarantee to converge asymptotically to true parameters if $\theta_k = Q_{K_k}$ remains fixed, and ϕ_t satisfy the persistence of excitation condition:

$$\epsilon_0 I \leq \frac{1}{N}\sum_{i=1}^{N} \phi_{t-i}\phi_{t-i}^\top \leq \bar{\epsilon}_0 I \text{ for all } t \geq N_0 \text{ and for all } N \geq N_0, \text{ for some } N_0 > 0.$$

6.4 Adaptive Policy Iteration Algorithm for LQR

Finally with everything defined in section 6, we can now propose an Adaptive Policy Iteration algorithm for LQR problems.

Given a initial state x_0 and stabilizing controller K_0 and using k to denote the policy iteration steps, t the total time steps, and i the time steps since the last policy change. We can propose the following algorithm:

Algorithm 1: Adaptive Policy Iteration for LQR

Initialize parameters: $\hat{\theta}_1(0)$;
$t \leftarrow 0$;
for $k = 0$ *to* ∞ **do**
  Initialize RLS: $\Sigma_k(0) = \Sigma_0$;
  for $i = 1$ *to* N **do**
    $u_t = K_k x_t + e_t$, for e_t exploratory noise signal ;
    Apply u_t and observe x_{t+1} ;
    Update $\hat{\theta}_k(i)$ using RLS ;
    $t \leftarrow t + 1$;
  end
  Find matrix $\hat{Q_{K_k}}$ corresponding $\hat{\theta}_k(N)$;
  set $K_{k+1} = -\hat{Q}_{K_k}^{-1}(2,2)\hat{Q}_{K_k}^\top(1,2)$;
  $\hat{\theta}_{k+1}(0) = \hat{\theta}k(N)$;
end

7 Convergence Analysis

Now that we have seen an algorithm that we can use for LQR problems, we need to ask ourselves why would it break? The reason the method that we laid out above might break is due to the fact that the Policy improvement step is based on an estimate of Q-factor parameterized by $\Theta(Q_{K_k})$. We also have no a priory reason to expect the Policy Iteration based on approximate Q-factors as defined in section 6.4 to converge to an optimal policy $K^\star$. We therefore need to prove this convergence.

Theorem 1. *Suppose (A, B) are controllable, K_0 is stabilizing, and ϕ_t is persistently excited. Then $\exists N < \infty$ such that the adaptive policy iteration mechanism described previously generates a sequence $\{K_k\}$ of stabilizing controllers satisfying:*

$$\lim_{k\to\infty} ||K_k - K_\star||_2 = 0 \tag{18}$$

for $K_\star$ the optimal LQR controller.

7.1 Proof: Strategy

Lets suppose that we are policy iterate k, and define the scalar "Lyapunov" like function

$$s_k = \sigma_k(K_{k-1}) + \left|\left|\theta_{k-2} - \hat{\theta}_{k-2}\right|\right|_2 \tag{19}$$

We can propose the following high level idea that if the estimation horizon N is sufficiently long, and the persistence of excitation conditions holds, then $s_{k+1} \leq s_k$.

We will therefore develop recursions for $v_k := \left\|\theta_{k=1} - \hat{\theta}_{k-1}\right\|_2$ and $\sigma\left(K_{k-1}\right)$ and then identify conditions on both estimation horizons to ensure that they both decrease.
The key idea of this proof is that if things are well behaved in the past, and that if we have (a nearly) true Q-Factor, our updates are guaranteed to (nearly) improve on performance.

7.2 Proof: intermediate results

Before we start we need two intermediate results. We let $\sigma(K_k) := tr(P_{K_k})$.

Lemma 1. *If (A, B) is controllable, K_1 is stabilizing with associated cost matrix P_1, and K_2 the result of one policy improvement step, i.e., $K_2 = -\gamma(R + \gamma B^\top P_1 B)^{-1} B^\top P_1 A$, then:*

$$\Delta\, ||K_1 - K_2||_2^2 \leq \sigma(K_1) - \sigma(K_2) \leq \delta\, ||K_1 - K_2||_2^2$$

where $0 < \Delta = \sigma_{\min}(R) \leq \delta = tr(R + \gamma B^\top P_1 B)\, \left|\left|\sum_{i=0}^{\infty} \gamma^{i/2} (A + BK_2)^i\right|\right|_2^2$

Lemma 2. *If ϕ_t is persistently excited and $N \geq N_0$, then*

$$\left|\left|\theta_k - \hat{\theta}_k\right|\right|_2 \leq \epsilon_N \left(||\theta_k - \theta_{k-1}||_2 + \left|\left|\theta_{k-1} - \hat{\theta}_{k-1}\right|\right|_2\right),$$

where $\epsilon_N = (\epsilon_0 N \sigma_0)^{-1}$ and $\sigma_0 = \sigma_{\min}(\Sigma_0)$.

7.3 Proof: nearby control laws are stable

Lets suppose that we are at policy iteration k, and define the scalar "Lyapunov" like function:

$$s_k = \sigma_k(K_{k-1}) + \left|\left|\theta_{k-2} - \hat{\theta}_{k-2}\right|\right|_2 \tag{20}$$

We can make the induction assumption that $s_i \leq \bar{s}_0 < \infty$ for all $0 \leq i \leq k$. From this assumption we can conclude that K_{k-1} is stabilizing as $\sigma(K_{k-1}) \leq \bar{s}_0$, and that the parameter estimation error is bounded as $\left|\left|\theta_{k-2} - \hat{\theta}_{k-2}\right|\right|_2 \leq \bar{s}_0$.

Now if the actual Q-function were available, we could compute the next iteration $K_k^\star$, and this controller would be stabilizing, and by the improvement property of policy iteration, would satisfy $\sigma(K_k^\star) \leq \bar{s}_0$. Therefore by continuity of the optimal policy update (Lemma 1), we have:

$$\forall \delta > 0 \, \exists \epsilon_\delta > 0, \text{ s.t.} |\sigma(K) - \sigma(K_k^\star)| \leq \delta \, ||K_k^\star - K||_2 \, \forall \, ||K_k^\star - K||_2 \leq \epsilon_\delta \tag{21}$$

This tells us that nearby control laws are also stabilizing.

7.4 Proof: bounding estimation error

We now need to show that $s_{k+1} \leq s_k$ if the policy estimation is chosen to be large enough, i.e., if we estimate a sufficiently accurate enough Q-factor, one of the estimation error or the performance must improve.

We first define $v_k = \left|\left|\theta_{k-1} - \hat{\theta}_{k-1}\right|\right|_2$; then, applying the inequality of Lemma 2 we have for all k:

$$v_k \leq \epsilon_N (v_{k-1} + ||\theta_{k-1} - \theta_{k-2}||_2) \tag{22}$$

where recall that $\epsilon_N \to 0$ as $N \to \infty$.

We now recall that by our induction assumption that $s_i \leq \bar{s}_0 < \infty$ for all $0 \leq i \leq k$, we immediately conclude that

- $v_{k-1} = \left|\left|\theta_{k-2} - \hat{\theta}_{k-2}\right|\right|_2 \leq \bar{s}_0$
- $\|\theta_{k-1} - \theta_{k-2}\|_2 \leq \kappa_1$ for some constant κ_1.

This follows because K_i is stabilizing for $0 \leq i \leq k$ and hence the induce Q-functions, and corresponding parameters $\theta_i = \Theta(Q_i)$ must be bounded. From this we can therefore write

$$v_k \leq \epsilon_N (\bar{s}_0 + \kappa_1) = \frac{1}{\epsilon_0 N \sigma_0} (\bar{s}_0 + \kappa_1). \tag{23}$$

which can be made arbitrarily small by choosing the estimation interval N to be sufficiently large.

We then define $K_k^\star$ to be a one-step policy iteration using the correct Q-function, i.e., $K_k^\star = -(Q_{k-1}^\star(2,2))^{-1}(Q_{k-1}^\star(1,2))^\top$, and let K_k be the one-step policy iteration using the estimated Q-function, i.e., $K_k = -(\hat{Q}_{k-1}(2,2))^{-1}(\hat{Q}_{k-1}(1,2))^\top$. We can note that if N is sufficiently large, then $\hat{Q}_{k-1}(2,2)$ is invertible.

This leads us to conclude that

$$||K_k - K_k^\star||_2 \leq \bar{\kappa}_0(1 + \left|\left|\hat{\theta}_{k-1}\right|\right|_2) \left|\left|\theta_{k-1} - \hat{\theta}_{k-1}\right|\right|_2 \tag{24}$$

for some $\bar{\kappa}_0 > 0$, for N sufficiently large.

Now since everything is bounded, we can further simplify this expression and write:

$$\|K_k - K_k^\star\|_2 \leq \kappa_0 \left\|\theta_{k-1} - \hat{\theta}_{k-1}\right\|_2 = \kappa_0 v_k \leq \epsilon_N \kappa_0 \left(\bar{s}_0 + \kappa_1\right) \tag{25}$$

where we used out previously derived bound on v_k.

From equation (21) we can see that:

$$|\sigma(K_k) - \sigma(K_k^\star)| \leq \delta \, ||K_k - K_k^\star||_2 \, \forall N \text{ s.t. } \epsilon N \kappa_0 (\bar{s}_0 + \kappa_1) \leq \epsilon_\delta \tag{26}$$

This implies that K_k is stabilizing if N is large enough, and achieves performance similar to that achieved by $K_k^\star$. Similarly, if we pick an even larger N_1, then this also holds true for K_{k-1}, allowing us to conclude that there exists a constant $\bar{\delta}$ such that

$$|\sigma(K_k) - \sigma(K_{k-1})| \leq \bar{\delta} \, ||K_k - K_{k-1}||_2 \text{ for all } N \geq N_1. \tag{27}$$

As the paper [2] explains: "In other words, if the estimation interval is long enough, then the difference between two consecutive costs is bounded by the difference between two consecutive controls. We use the definition of the parameter estimation vector to write, for δ_1 a constant:"

$$||\theta_{k-1} - \theta_{k-2}||_2 \leq \delta_1 \, ||K_k - K_{k-1}||_2^2 \; \forall N \geq N_1. \tag{28}$$

This can be explained given a Q-Factor, if we optimize a set $u^\star = -Q^{-1}(2,2)Q(1,2)^\top x =: Kx$ we obtain the value function $J(x) = x^\top \left(Q(1,1) - K^\top Q(2,2)K\right) x$ which is quadratic in K.

We now can use the quadratic inequality $(a+b)^2 \leq 2\left(a^2 + b^2\right)$ to get:

$$\begin{aligned} ||\theta_{k-1} - \theta_{k-2}||_2 &\leq 2\delta_1 \left(||K_k^\star - K_{k-1}||_2^2 + ||K_k^\star - K_k||_2^2\right) \\ &\leq 2\delta_1 (w_k^2 + (\kappa_0 v_k)^2) \\ &\leq 2\delta_1 (w_k^2 + (\kappa_0 v_k)) \end{aligned} \tag{29}$$

Recalling our recursion on $v_k = \left\|\theta_{k-1} - \hat{\theta}_{k-1}\right\|_2$ we also have:

$$v_k \leq \epsilon_N \left(v_{k-1} + \|\theta_{k-1} - \theta_{k-1}\|_2\right) \leq \epsilon_N \left(v_{k-1} + 2\delta_1 \left(w_k^2 + \kappa_0 v_k\right)\right) \tag{30}$$

We can now rearrange our equations to obtain:

$$v_k \leq \frac{\epsilon_N}{1 - 2\delta_1 \kappa_0 \epsilon_N} \left(v_{k-1} + 2\delta_1 w_k^2\right) \tag{31}$$

Making sure that $\epsilon_N \to 0$ as $N \to 0$, we see that v_k can be made to converge to 0 so long as $w_k^2 = \left\|K_k^\star - K_{k-1}^2\right\|_2$ is well-behaved.

By noticing that $\sigma(K_k) - \sigma(K_{k-1}) = \sigma(K_k^\star) - \sigma(K_{k-1}) + \sigma(K_k) - \sigma(K_k^\star)$. We can now use the continuity of cost of (Lemma 1) and previous bounds to conclude that id we choose our update interval N to be sufficiently large, then there exists positive constants Δ and δ_2 such that:

$$\begin{aligned} \sigma(K_k) - \sigma(K_{k-1}) &\leq -\Delta \, ||K_k^\star - K_{k-1}||_2^2 + \delta_2 \, ||K_k^\star - K_k||_2^2 \\ &\leq -\Delta \, ||K_k^\star - K_{k-1}||_2^2 + \delta_2 \left|\left|\theta_{k-1} - \hat{\theta}_{k-1}\right|\right|_2^2 \\ &\leq -\Delta w_k^2 + \delta_2 \kappa_0 v_k \end{aligned} \tag{32}$$

7.5 Proof: putting it all together

Finally by combining the estimation error and cost recursions (setting them to equality) we define the system such as:

$$\begin{bmatrix} v_k \\ \sigma\left(K_k\right) \end{bmatrix} = \begin{bmatrix} \frac{\epsilon_N}{1-2\delta_1\kappa_0\epsilon_N} & 0 \\ \delta_2\kappa_0\frac{\epsilon_N}{1-2\delta_1\kappa_0\epsilon_N} & 1 \end{bmatrix} \begin{bmatrix} v_{k-1} \\ \sigma\left(K_{k-1}\right) \end{bmatrix} + \begin{bmatrix} 2\frac{\epsilon_N}{1-2\delta_1\kappa_0\epsilon_N}\delta_2 \\ -\Delta + 2\delta_2\kappa_0\frac{\epsilon_N}{1-2\delta_1\kappa_0\epsilon_N} \end{bmatrix} w_k^2 \tag{33}$$

We now recall our Lyapunov function $s_k = \sigma(K_k) + \left|\left|\theta_{k-2} - \hat{\theta}_{k-2}\right|\right|_2 = \sigma(K_k) - v_{k-1}$. Combining this with equation (33) above we then have:

$$s_{k+1} = s_k + (-1 + \frac{\epsilon_N}{1-2\delta_1\kappa_0\epsilon_N}(1+\delta_2\kappa_0))v_{k-1} + (-\Delta + 2\frac{\epsilon_N}{1-2\delta_1\kappa_0\epsilon_N}\delta_2(1+\kappa_0))w_k^2. \tag{34}$$

One can now notice that v_{k-1} and w_k^2 are non-negative, hence it suffices to pick N large enough to ensure that the coefficients in front of them are non-positive to ensure that $s_{k+1} \leq s_k$.

References

[1] Richard S. Sutton and Andrew G. Barto: *Reinforcement Learning: An Introduction* (The MIT Press Cambridge, Massachusetts, London, England)

[2] S.J. Bradtke, B.E. Ydstie, & A.G. Barto: *Adaptive linear quadratic control using policy iteration* (Proceedings of 1994 American Control Conference - ACC '94).

[3] Francisco S. Melo & M. Isabel Ribeiro *Convergence of Q-learning with linear function approximation* (Proceedings of the European Control Conference 2007 Kos, Greece, July 2-5, 2007)