IN 4320 Machine Learning Assignment 4

Xiangwei Shi
4614909

Exercise 1:
Proof:
The return $R_t = \sum_{h=0}^{\infty} \gamma^h r_{t+h+1}$ has the condition that $0 \leq \gamma < 1$ and bounded reward $-10 \leq r_{t+h+1} \leq 10$. For $\forall h \in \{0, 1, \cdots, \infty\}$, we can get

(i) When $\gamma = 0$,

$$\gamma^h = \begin{cases} 1, & h = 0 \\ 0, & \text{otherwises.} \end{cases}$$

Therefore, the return $R_t = r_{t+1}$, which is bounded because of the bounded reward, $-10 \leq R_t \leq 10$.

(ii) When $0 < \gamma < 1$, we can get that γ^h is always positive. Consider the return R_t is a function with the bounded reward r_{t+h+1} as the variable. Because of the positive values of γ^h, the function of return R_t is a monotonically increasing function. Therefore, when the rewards r_{t+h+1} are the same and equal to 10 for all h, the return has a maximum. And it will have a minimum, when the rewards r_{t+h+1} are the same and equal to -10 for all h. Now consider γ^h as a geometric sequence, the sum over this sequence is

$$\sum_{h=0}^{\infty} \gamma^h = \frac{\gamma^0 - \gamma^h \gamma}{1 - \gamma} = \frac{1}{1 - \gamma}.$$

Therefore, the return has a maximum, $\frac{10}{1-\gamma}$, and a minimum, $-\frac{10}{1-\gamma}$. The return R_t is bounded, $-\frac{10}{1-\gamma} \leq R_t \leq \frac{10}{1-\gamma}$.

Exercise 2:
First iteration:

action \ state	1	2	3	4	5	6
left	0	1	0	0	0	0
right	0	0	0	0	5	0

Second iteration:

action \ state	1	2	3	4	5	6
left	0	1	0.5	0	0	0
right	0	0	0	2.5	5	0

Third iteration:

action \ state	1	2	3	4	5	6
left	0	1	0.5	0.25	1.25	0
right	0	0.625	1.25	2.5	5	0

Fourth iteration:

action \ state	1	2	3	4	5	6
left	0	1	0.5	0.625	1.25	0
right	0	0.625	1.25	2.5	5	0

The Q values are calculated by $Q^*(s,a) = \sum_{s' \in S} P^a_{ss'}[R^a_{ss'} + \gamma(\max_{a' \in A} Q^*(s',a'))]$, where $R^a_{ss'} = 0$. Since the optimal policy $\pi^* = \arg\max_{a \in A}(Q^*(s,a))$, we can get $\pi^*(2) =$ left, $\pi^*(3) =$ right, $\pi^*(4) =$ right and $\pi^*(5) =$ right.

Exercise 3:
For $\gamma = 0$, the optimal value function Q^* is shown below.

action \ state	1	2	3	4	5	6
left	0	1	0	0	0	0
right	0	0	0	0	5	0

For $\gamma = 0.1$, the optimal value function Q^* is shown below.

action \ state	1	2	3	4	5	6
left	0	1	0.1	0.01	0.05	0
right	0	0.01	0.05	0.5	5	0

For $\gamma = 0.9$, the optimal value function Q^* is shown below.

action \ state	1	2	3	4	5	6
left	0	1	3.2805	3.645	4.05	0
right	0	3.645	4.05	4.5	5	0

For $\gamma = 1$, the optimal value function Q^* is shown below.

action \ state	1	2	3	4	5	6
left	0	1	5	5	5	0
right	0	5	5	5	5	0

The discount factor γ, first of all, influences the optimal Q values, which will influence the optimal policy π^* eventually. Secondly, for this problem, the large value of γ makes it more possible to choose right when in state 2 and 3. And small value of γ makes it more possible to choose left when in state 2 and 3. In general, small value of γ tends to get reward quickly, and large value of γ tends to get high reward.
As for the reason why $\gamma = 1$ works here in contrast to Exercise 1, it is because there is a final state and the limited iteration instead of infinity for $\gamma = 1$ here.

Exercise 4:
Zero value initialization:
To explore the influence of α and ϵ, different values of α and ϵ are tried. Figure 1 and Figure 2 show the influence of α and ϵ.From Figure 1, we can find that when α does not change, for large value of ϵ, the difference between two value functions decrease faster than the one with small value of ϵ. And when ϵ is large to some degree, the difference between two value functions will become similar. This can be explained that the large value of ϵ means large probability to randomly choose action which will bring new information and decrease the difference fast. When the value of ϵ is large enough, the new information brought by randomly choosing actions will be similar.
From Figure 2, we can find that when ϵ is fixed, for large value of α, the difference between two value functions decrease faster than the one with small value of α. And the difference with large value of α decrease with 'larger step' than the one with small value. The reason for this can be that the learning rate α reveals the magnitude of step that is taken towards the solution. The large value of α means large step to learn the solution, which also causes decreasing the difference fast.
Therefore, the suitable α and ϵ for Q-learning should be neither too larger nor too small. Small value of α and ϵ will take long time to converge, and large value of α will make it oscillate around the minimal value function.
Random value initialization:
For this scenario, I initialize the state-action value randomly, which is between 0 and 1.

Similar with zero value initialization, to explore the influence of α and ϵ, difference values of them are tested. Figure 3 and Figure 4 show their influence. Compared with zero value initialization scenario, we can find the similar results, which are large values of α and ϵ will decrease the difference faster than the small values and large value of α will produce 'large step'. However, the initial difference is smaller in some condition, which is because the difference between random initialization and the optimal value function is smaller than the one between zero value and the optimal value function.

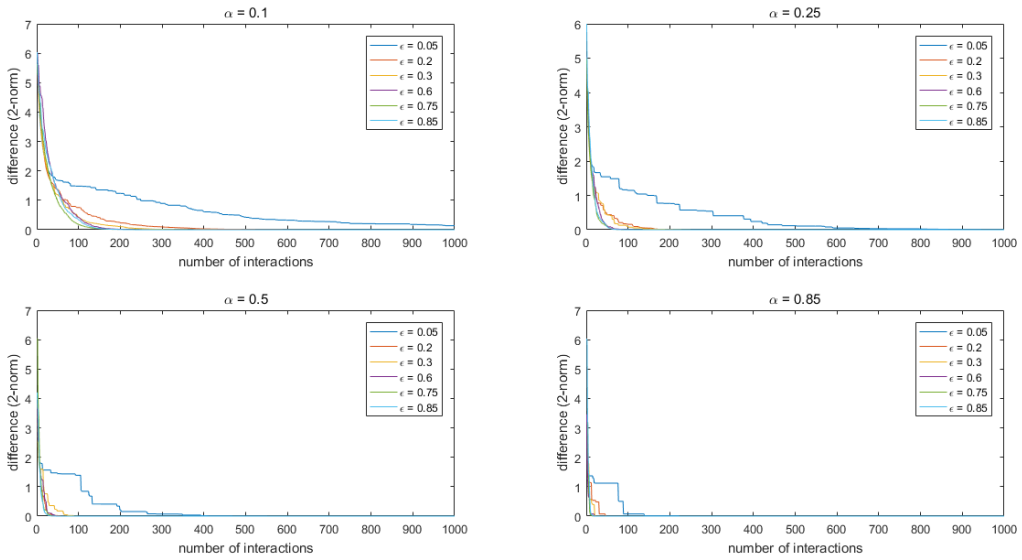

Figure 1: Difference between value function by Q-learning and true value function with a fixed alpha (Zero initialization)

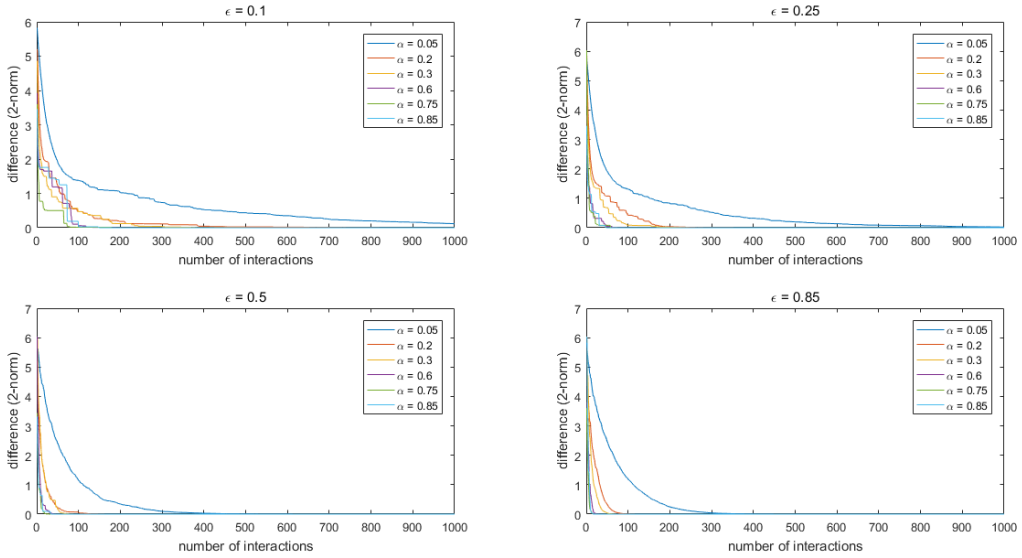

Figure 2: Difference between value function by Q-learning and true value function with a fixed epsilon (Zero initialization)

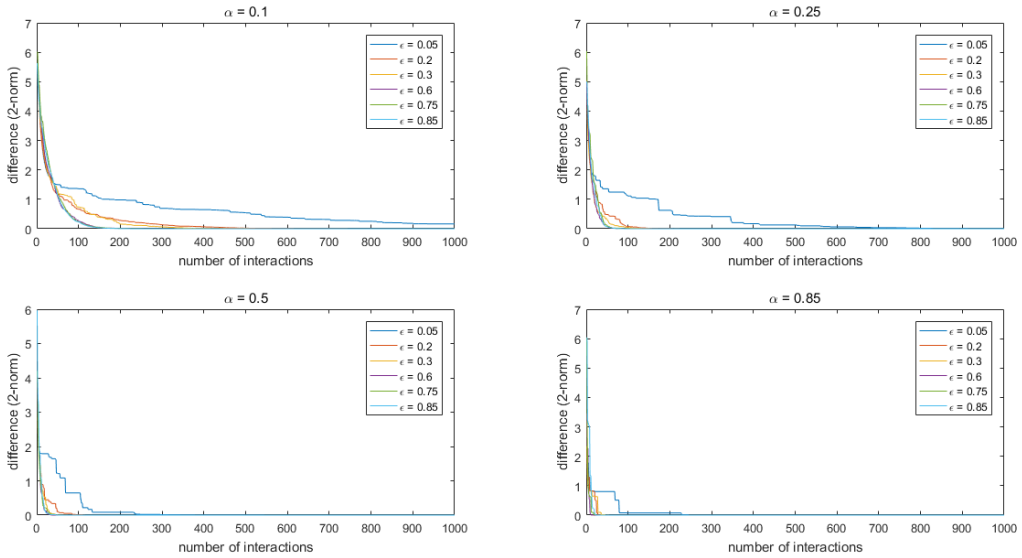

Figure 3: Difference between value function by Q-learning and true value function with a fixed alpha (Random initialization)

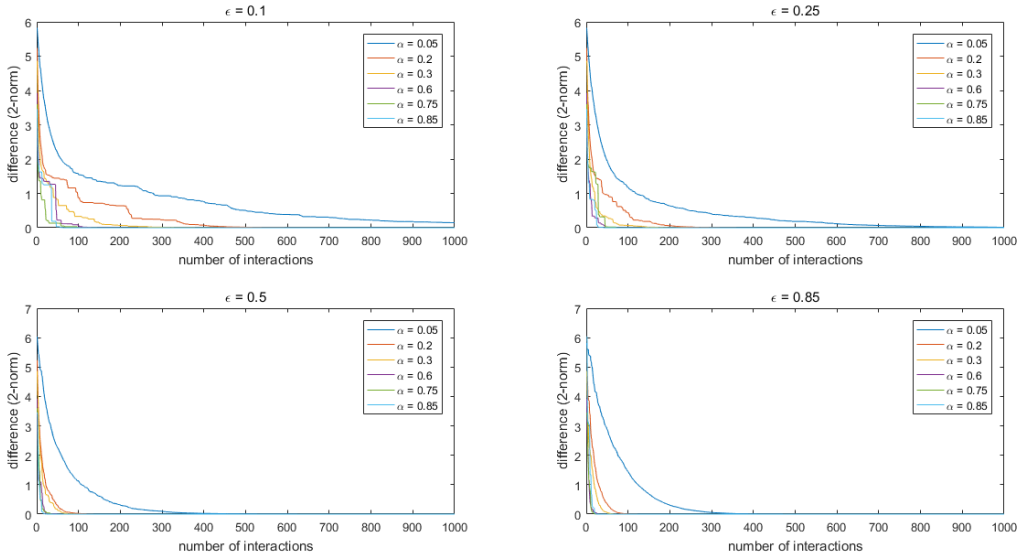

Figure 4: Difference between value function by Q-learning and true value function with a fixed epsilon (Random initialization)

Exercise 5:
In this scenario, implementing Q-iteration will get the following results, which are deterministic. Instead of updating the value function by $Q^*(s,a) = R^a_{ss'} + \gamma(\max_{a' \in A} Q^*(s',a'))$, the value function is updated by $Q^*(s,a) = 0.7 * (R^a_{ss'} + \gamma(\max_{a' \in A} Q^*(s',a'))) + 0.3 * (R^a_{ss} + \gamma(\max_{a' \in A} Q^*(s,a')))$ for Q-iteration.

action \ state	1	2	3	4	5	6
left	0	0.8235	0.3930	0.4987	1.2111	0
right	0	0.3679	0.6981	1.6955	4.1176	0

As for Q-learning, the value function depends on the value of α and ϵ. And in this scenario, since there is a 30% probability staying at the same state, the value function of Q-learning cannot converge. However, similar with Q-iteration, all values of the value function of Q-learning are smaller than the optimal value function Q^* in exercise 2.
Since the probability, 30%, only affects the states, it does not conflict the learning rate(α) for Q value, discount factor(γ) and exploration(ϵ) for choosing action. Therefore, the same Q-learning parameters can still be used in this scenario. Figure 5 and Figure 6 show the difference between the value function of Q-learning in this scenario and the optimal value function with zero value initialization, fixed α and ϵ. From both figures, we can find that in this scenario, Q-learning is no longer deterministic. And Figure 5 also shows that the value of α determines range of difference. Small value of α has a small range of difference, vice versa. The reason for causing this is because learning rate α determines how much to learn from the new state and new action. Large value of α means learning from the new state and new action more, which are not deterministic in this scenario. Clearly, the suitable parameters α and ϵ must be different from the ones in exercise 4. And Figure 7 and Figure 8 show the difference with random value initialization, fixed α and ϵ. By comparing the figures with different initialization, we can find that different initialization get similar results, which means initialization does not affect Q-learning in this non-deterministic scenario.

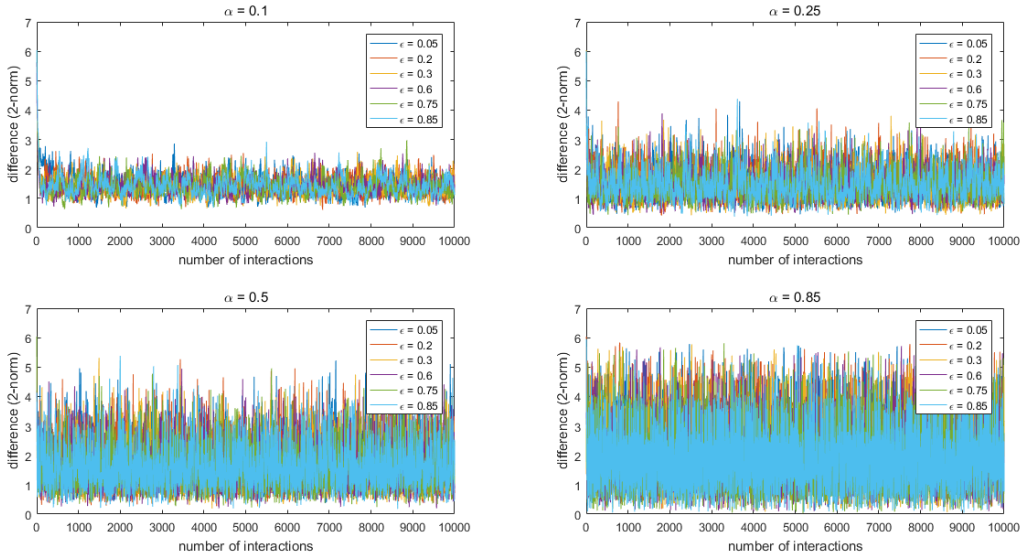

Figure 5: Difference between value function by Q-learning and true value function with a fixed alpha (Zero initialization)

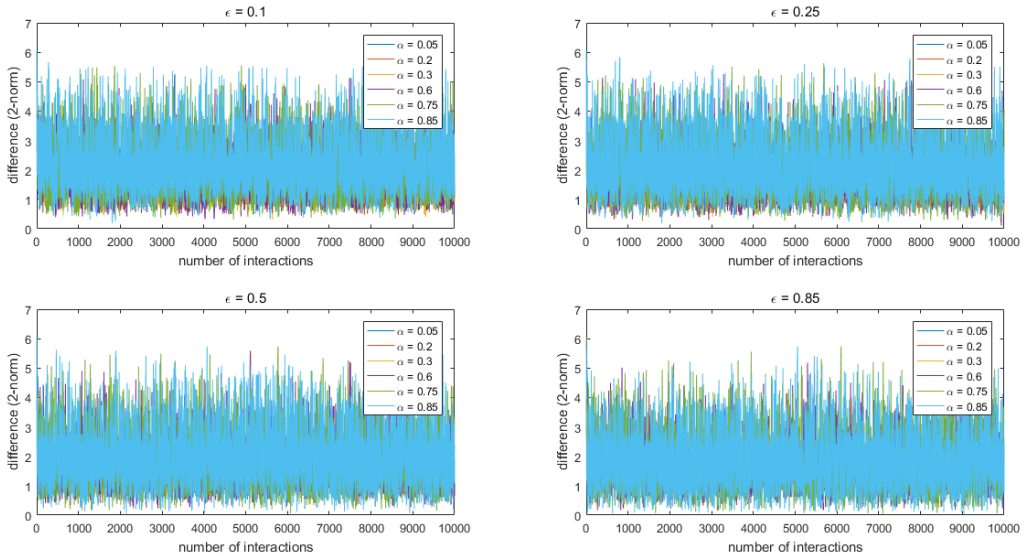

Figure 6: Difference between value function by Q-learning and true value function with a fixed epsilon (Zero initialization)

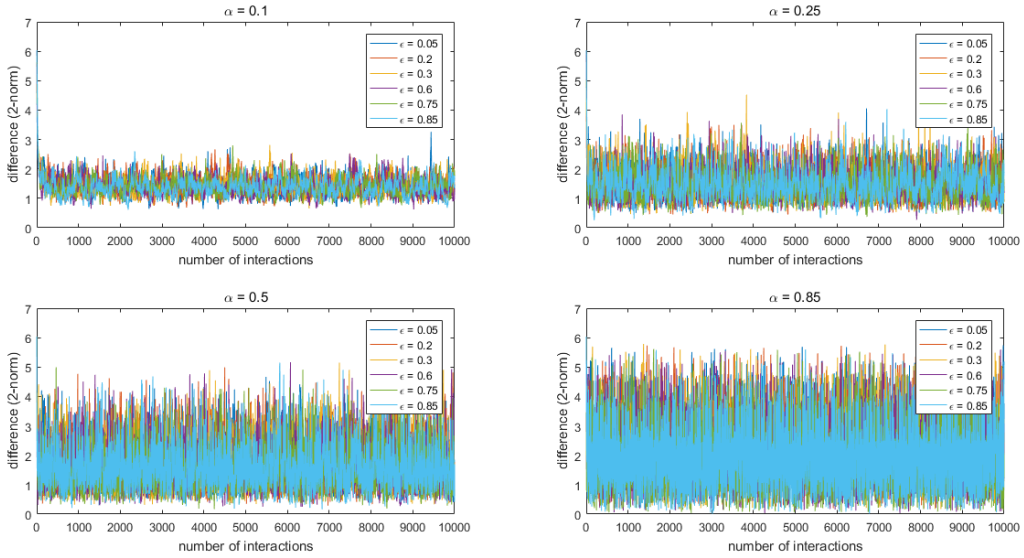

Figure 7: Difference between value function by Q-learning and true value function with a fixed alpha (Random initialization)

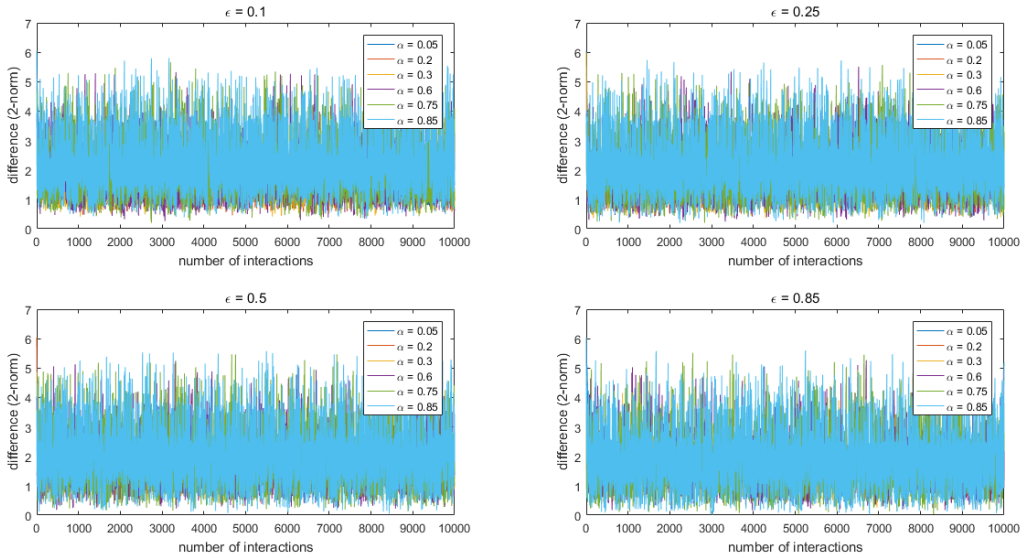

Figure 8: Difference between value function by Q-learning and true value function with a fixed epsilon (Random initialization)

Exercise 6:
In this exercise, Q-learning value function approximation with radial basis function is implemented. The pseudo-code is shown below.
Algorithm: Radial basis function approximation for Q-learning

1. Initialize the weights(θ_k)
2. Input: α. ϵ and γ
3. Loop until convergence
   (a) randomly select the state(s)
   (b) a = arg max(Q(s,a)), with probability ϵ, a = rand(action)
   (c) s_new, reward <- state(s) + action(a) + $N(0,0.01)$
   (d) updating θ_k: $\theta_k = \theta_k + \alpha(\text{reward} + \gamma(\max(\text{Q(s\_new,action)}))\text{-Q(s,a)})\frac{\partial Q(s,a)}{\partial \theta_k}$, where the state-action value Q(s,a) = $\sum \theta_k \cdot f_{RBF}(s, a, \theta_k)$

Since there are two final states with reward in this scenario, and the radial basis function is one-dimensional function with single peak. In order to bring the information of the final states into the RBF kernel, we need at least two basis function, which can interpret the Q-learning in this scenario. To investigate the influence of number of basis functions, I implement 3 Q-learning with different number of basis functions, 3(1,2,3), 6(1,2,3,4,5,6) and 11(1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6). Figure 9, Figure 10 and Figure 11 show the relation between Q-value and state with fixing Q-learning parameters. Comparing the deterministic and discrete Q-learning, Figure 9 shows a bad approximation, which is because the basis functions are located at one side of [1,6]. And Figure 10 has similar Q-values with the optimal Q-values, which we can say is a good approximation. As for Figure 11, since the basis functions are uniformly located, the approximation becomes more accurate than the one from Figure 10. Therefore, to get a good approximation of Q-value in this scenario, the number of basis functions and how to select basis functions is important.

To test the influence of width, I also implement three different Q-learning in this scenario, $\sigma = 0.1$, $\sigma = 0.2$ and $\sigma = 0.4$. Figure 12, Figure 13 and Figure 14 show the relation between Q-value and width of basis function. Comparing these three figures, we can get the width of basis function will affect the performance of Q-learning in this scenario. The basis function with large width will provide smoother approximation, which is also more accurate.

To compare the previous Q-learning algorithm, I test the time of convergence with six basis functions in Figure 15 and Figure 16. Comparing these figures, we can find large value of α is more difficult to converge than the one with small value, which conflicts with the previous Q-learning algorithm. This is possibly because $\alpha = 0.4$ is a larger learning rate in this scenario, and the Q-values will stuck at a local minimal, which will take more time to converge or never converges. In both figures, the vertical axis represents the difference of θ between two iterations.

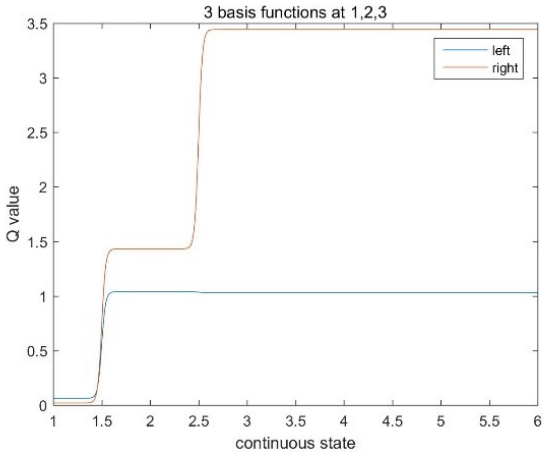

Figure 9: Q-value with three basis functions

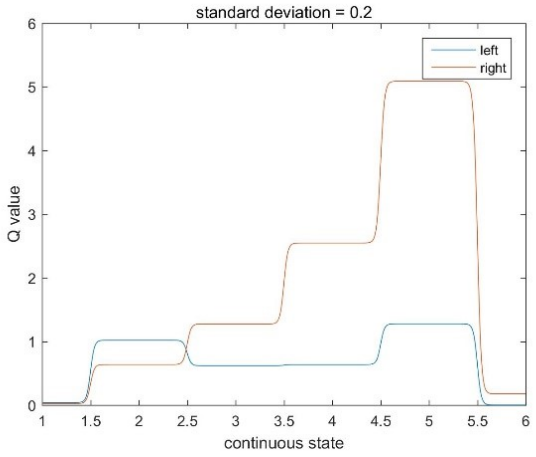

Figure 10: Q-value with six basis functions

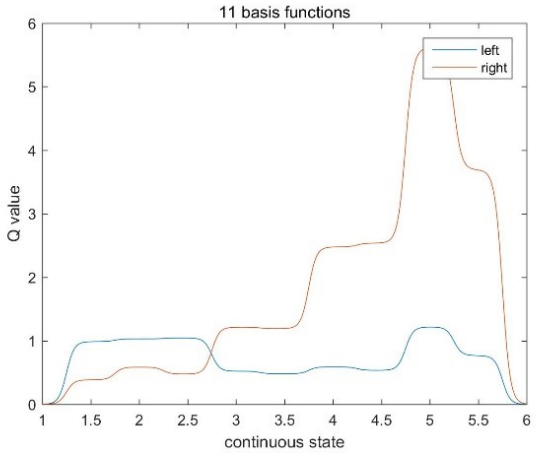

Figure 11: Q-value with eleven basis functions

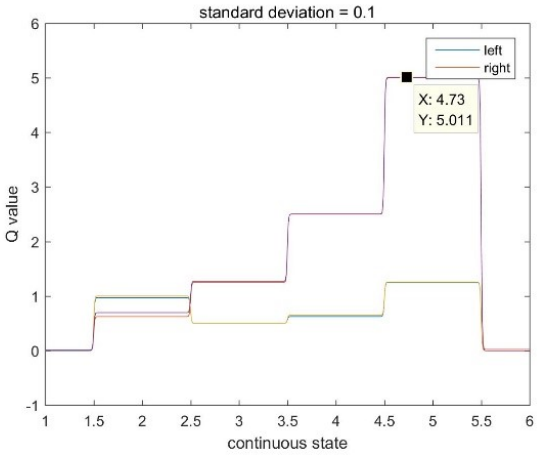

Figure 12: Q-value with $\sigma = 0.1$

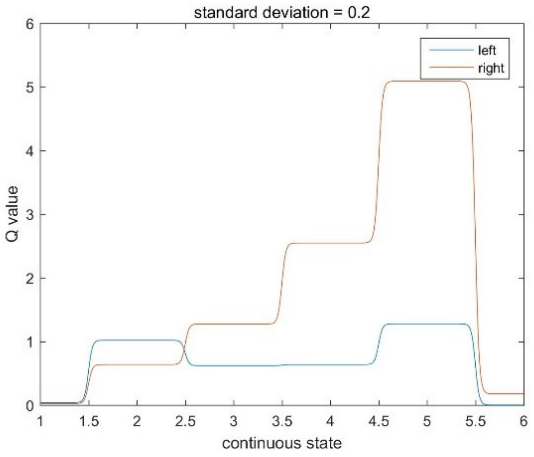

Figure 13: Q-value with $\sigma = 0.2$

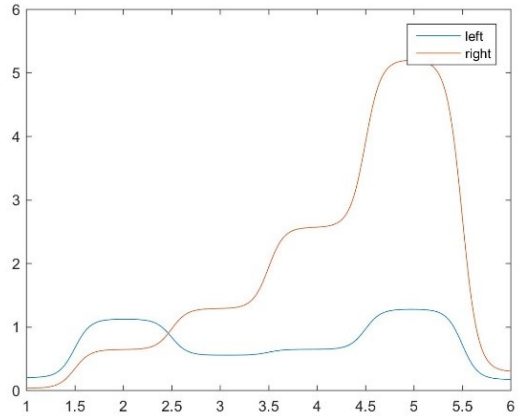

Figure 14: Q-value with $\sigma = 0.4$

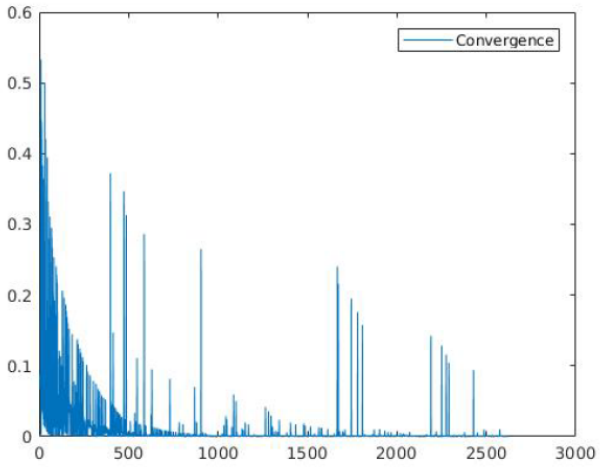

Figure 15: Convergence with six basis functions, $\alpha = 0.1$ and $\epsilon = 0.1$

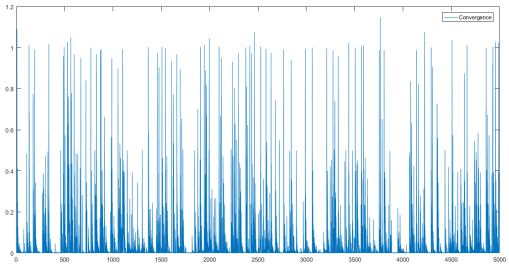

Figure 16: Convergence with six basis functions, $\alpha = 0.4$ and $\epsilon = 0.1$

Exercise 7:
In this exercise, I implement another RL algorithm - SARSA, and the pseudo-code is shown below.

1. Initialize Q(s,a) randomly
2. Repeat (for each episode):
   (a) Initialize state s
   (b) Choose action a from s using policy derived from Q (ϵ-greedy)
   (c) Repeat (for each step of episode):
      i. Take action a, observe r, s'
      ii. Choose a' from s' using policy derived from Q (ϵ-greedy)
      iii. Update Q(s,a) <- Q(s,a)+α[r+γQ(s',a')-Q(s,a)]
      iv. s <- s'; a <- a'
   (d) until s is terminal

To compare SARSA and Q-learning, firstly, I implement the SARSA algorithm in the deterministic and non-deterministic set-ups with random value initialization, Figure 17 and Figure 18.

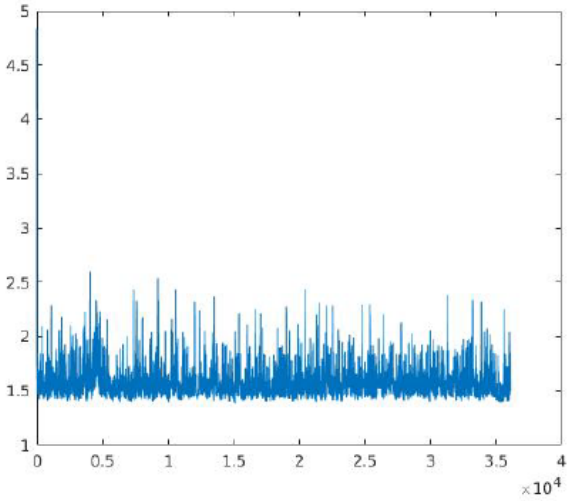

Figure 17: Convergence with deterministic set-up, $\alpha = 0.1$ and $\epsilon = 0.1$

According to the pseudo-code above, we can get one advantage of SARSA algorithm is that in the procedure of updating the Q-value, next state and next action are computed at the same iteration, which is different from Q-learning where only computes the state and takes the action computed under the optimal policy into consideration, which may not next action.

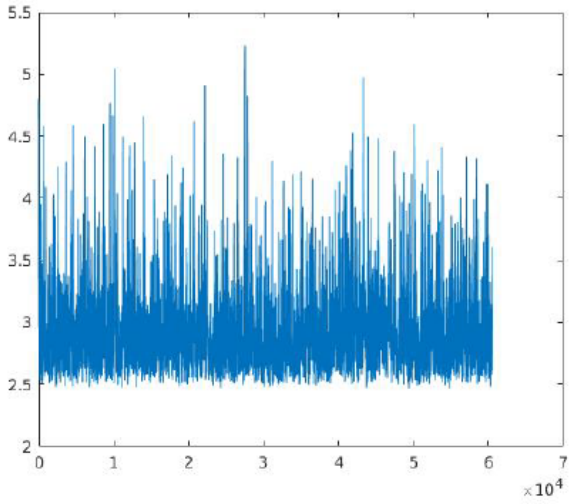

Figure 18: Convergence with non-deterministic set-up, $\alpha = 0.1$ and $\epsilon = 0.1$

Therefore, SARSA is computationally less intensive. Since SARSA considers the next state and next action, which makes on-policy updating an advantage compared to the Q-learning algorithm (Figure 17 and Figure 18). However, the convergence iteration of SARSA is much larger than the one of Q-learning, which is a disadvantage even though the optimal policy does not change.