1.

(a) State Space: $\mathcal{X} = \{0, 1, 2, 3, 4, 11, 12, 22\}$, which means the money we have, e.g. if $x = 1$ then we have \$10.000. Then terminal states are $\mathcal{T} = \{x \leq 0 \text{ or } x \geq 3 \mid x \in \mathcal{X}\}$.

Control Space: $\mathcal{U} = \{1(r), 0(b)\} \times \{m\} \subseteq \mathbb{R}^2$. where 1 means betting on red and 0 means betting on black. m is integer and $m \geq 1$, this is the money we bet, e.g. if $m = 2$ then we bet \$20.000 this round.

Motion model: If $x_t \in \mathcal{X} \setminus \mathcal{T}$

$$x_{t+1} \mid x_t, u_t = (1, m) = \begin{cases} x_t + m, \text{ with prob. } 0.7 \\ x_t - m, \text{ with prob. } 0.3 \end{cases}$$

$$x_{t+1} \mid x_t, u_t = (0, m) = \begin{cases} x_t + 10m, \text{ with prob. } 0.2 \\ x_t - m, \text{ with prob. } 0.8 \end{cases}$$

If $x_t \in \mathcal{T}$.

$$x_{t+1} \mid x_t, u_t = x_t, \text{ i.e. } \begin{cases} P_f(x_t \mid x_t, u_t) = 1 \\ P_f(x_{t+1} \neq x_t \mid x_t, u_t) = 0 \end{cases}$$

Stage cost: $\ell(x, u, x') = -(x' - x)$

i.e. Once we get into terminal costs, we get $-x'$ reward.

Then $\tilde{\ell}(x, u) = -\sum_{x'} P_f(x' \mid x, u)(x' - x)$

Terminal cost: $g(x_T) = -x_T$

(b) $V^*(x) = \min_{u \in U(x)} \tilde{\ell}(x,u) + \mathbb{E}_{x' \sim p_f(\cdot|u,x)}[V^*(x)]$

In this case, $\pi = u_t = (1,1)$. then

$\mathcal{X} = \{0,1,2,3\}$. terminal states $\mathcal{T} = \{0,3\}$

Suppose at iteration $k=0$. $V_0^\pi(x) = 0$ for all $x \in \mathcal{X} \setminus \mathcal{T}$

$V_0^\pi(x_T) = -x_T$ for all $x_T \in \mathcal{T}$.

When $k=1$.

$$V^\pi(x) = \tilde{\ell}(x,u,x') + \sum_{x' \in \mathcal{X}} p_f(x'|x,u) V^\pi(x')$$

By program.

$\bar{V}^\pi = [0, -0.4, -2.5, -3, -4, -11, -12, -22]^T$

for $x = 0, 1, 2, 3, 4, 11, 12, 22$

Also. we get precise estimate

$\hat{V}^\pi = [0, -2.7215, -3.3165, -3, -4, -11, -12, -22]^T$

for $x = 0, 1, 2, 3, 4, 11, 12, 22$

(c) State space $\mathcal{X} = \{0,1,2,3,4,11,12,22\}$, $\mathcal{T} = \{0,4,11,12,22\}$

Given $\bar{V}^\pi$. using computer we get

$\bar{\pi}'(1) = (0,1)$, $\bar{\pi}'(2) = (0,2)$, $\bar{\pi}'(x) = \{\text{go home}\}$, $x \in \mathcal{T}$.

which both means to bet on black and bet all money.

Given $\hat{V}^\pi$, using computer we get

$\hat{\pi}'(1) = (0,1)$, $\hat{\pi}'(2) = (0,2)$, $\hat{\pi}'(x) = \{\text{go home}\}$ $x \in \mathcal{T}$.

so if we have \$10,000, we bet on black and bet \$10,000;
if we have \$20,000, we bet on black and bet \$20,000.

$\bar{\pi}'$ and $\hat{\pi}'$ are the same. Compared to $\pi = (1, 1)$, the policy $\bar{\pi}'$, $\hat{\pi}'$ become bolder, trying to earn more money.

(d) Using Policy Iteration (PI), we'll have

$V^*(1) = -6.6364$, $\pi^*(1) = (1, 1) \longrightarrow$ bet red, \$10,000

$V^*(2) = -8.9091$, $\pi^*(2) = (0, 1) \longrightarrow$ bet black, \$10,000

$V^*(x) = x$, $\pi^*(x) = \{\text{stay}\}$ for $x \in \mathcal{T}$.

Therefore, we bet red, \$10,000 when we have \$10,000; and we bet black, \$10,000 when we have \$20,000.

Under this policy, I simulate the gambling for 50,000 times, and the average money I take home is \$38,614.

Celebration!

Problem 2

Problem Formulation

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Figure 1: Environment

- State space: $\mathcal{X} = \{1, 2, 3, ..., 25\}$. Then special states are: A $= 2$, A$' = 22$, B $= 4$, B$' = 14$.
- Control space: $U = \{n, s, w, e\}$.
- Motion model:
  When $x \notin \{2, 4\}$:

$$
\begin{aligned}
x'|(x, u = n) &= \begin{cases} x - 5, & \text{if } x \geq 6 \\ x, & \text{else} \end{cases} \\
x'|(x, u = s) &= \begin{cases} x + 5, & \text{if } 1 \leq x \leq 20 \\ x, & \text{else} \end{cases} \\
x'|(x, u = e) &= \begin{cases} x + 1, & \text{if } x \in \{5, 10, 15, 20, 25\} \\ x, & \text{else} \end{cases} \\
x'|(x, u = w) &= \begin{cases} x - 1, & \text{if } x \in \{1, 6, 11, 16, 21\} \\ x, & \text{else} \end{cases} .
\end{aligned} \tag{1}
$$

When $x \in \{2, 4\}$, i.e. $x \in \{\text{A}, \text{B}\}$:

$$\begin{aligned} x'|(x = 2, u \in U) &= 22 \\ x'|(x = 4, u \in U) &= 14 \end{aligned} \tag{2}$$

- Stage cost:

$$\begin{aligned} l(x, u = n) &= \begin{cases} 0, & \text{if } x \notin \{2, 4\} \text{ and } x \geq 6 \\ 1, & \text{if } x \notin \{2, 4\} \text{ and } 1 \leq x \leq 5 \\ -10, & \text{if } x = 2 \\ -5, & \text{if } x = 4 \end{cases}, \\ l(x, u = s) &= \begin{cases} 0, & \text{if } x \notin \{2, 4\} \text{ and } 1 \leq x \leq 20 \\ 1, & \text{if } x \notin \{2, 4\} \text{ and } 21 \leq x \leq 25 \\ -10, & \text{if } x = 2 \\ -5, & \text{if } x = 4 \end{cases}, \\ l(x, u = e) &= \begin{cases} 0, & \text{if } x \notin \{2, 4\} \text{ and } x \notin \{5, 10, 15, 20, 25\} \\ 1, & \text{if } x \notin \{2, 4\} \text{ and } x \in \{5, 10, 15, 20, 25\} \\ -10, & \text{if } x = 2 \\ -5, & \text{if } x = 4 \end{cases}, \\ l(x, u = w) &= \begin{cases} 0, & \text{if } x \notin \{2, 4\} \text{ and } x \notin \{1, 6, 11, 16, 21\} \\ 1, & \text{if } x \notin \{2, 4\} \text{ and } x \in \{1, 6, 11, 16, 21\} \\ -10, & \text{if } x = 2 \\ -5, & \text{if } x = 4 \end{cases}. \end{aligned} \tag{3}$$

- No Terminal Cost, or $q(x) = 0$.
- Bellman Equation:

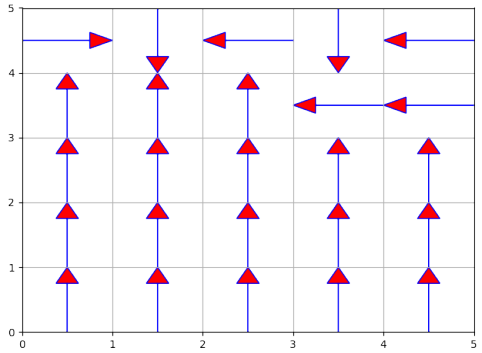
$$V^*(x) = min_{\pi \in U(x)}\{l(x, \pi(x)) + \gamma \mathbb{E}_{x' \sim p_f(.|x, \pi(x)}[V(x')]\}. \tag{4}$$

(a)

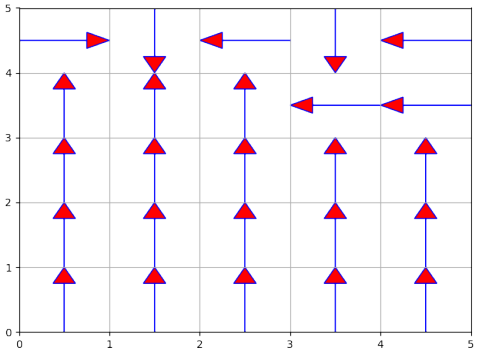
After implementing Value Iteration (VI), we can get the optimal value function (see Table 1) and optimal policy (see Fig. 2(a)), where $\gamma = 0.9$, and the initial value function is drawn from standard normal distribution.

(b)

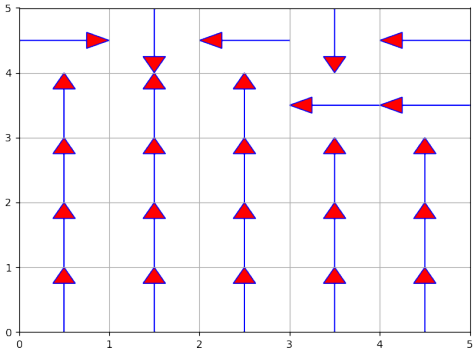
After implementing Policy Iteration (PI), we can get the optimal value function (see Table 2) and optimal policy (see Fig. 2(b)), where $\gamma = 0.9$, the initial policy is randomly generated, and the initial value function is drawn from standard normal distribution.



(a) VI, $\gamma = 0.9$



(b) PI, $\gamma = 0.9$



(c) QI, $\gamma = 0.9$

Figure 2: Optimal Policy, γ=0.9

Table 1: Optimal Value Function, Using VI

States	1	2	3	4	5
$V^*(x)$	-21.9775	-24.4194	-21.9775	-19.4194	-17.4775
States	6	7	8	9	10
$V^*(x)$	-19.7797	-21.9775	-19.7797	-17.8018	-16.0216
States	11	12	13	14	15
$V^*(x)$	-17.8018	-19.7797	-17.8018	-16.0216	-14.4194
States	16	17	18	19	20
$V^*(x)$	-16.0216	-17.8018	-16.0216	-14.4194	-12.9775
States	21	22	23	24	25
$V^*(x)$	-14.4194	-16.0216	-14.4194	-12.9775	-11.6797

Table 2: Optimal Value Function, Using PI

States	1	2	3	4	5
$V^*(x)$	-21.9775	-24.4194	-21.9775	-19.4194	-17.4775
States	6	7	8	9	10
$V^*(x)$	-19.7797	-21.9775	-19.7797	-17.8018	-16.0216
States	11	12	13	14	15
$V^*(x)$	-17.8018	-19.7797	-17.8018	-16.0216	-14.4194
States	16	17	18	19	20
$V^*(x)$	-16.0216	-17.8018	-16.0216	-14.4194	-12.9775
States	21	22	23	24	25
$V^*(x)$	-14.4194	-16.0216	-14.4194	-12.9775	-11.6797

(c)

After implementing Q-value Iteration (QI), we can get the optimal value function (V-value) (see Table 3) and optimal policy (see Fig. 2(c)), where $\gamma = 0.9$, and the initial Q-value function is drawn from standard normal distribution.

Table 3: Optimal Value Function, Using QI

States	1	2	3	4	5
$V^*(x)$	-21.9775	-24.4194	-21.9775	-19.4194	-17.4775
States	6	7	8	9	10
$V^*(x)$	-19.7797	-21.9775	-19.7797	-17.8018	-16.0216
States	11	12	13	14	15
$V^*(x)$	-17.8018	-19.7797	-17.8018	-16.0216	-14.4194
States	16	17	18	19	20
$V^*(x)$	-16.0216	-17.8018	-16.0216	-14.4194	-12.9775
States	21	22	23	24	25
$V^*(x)$	-14.4194	-16.0216	-14.4194	-12.9775	-11.6797

The optimal solutions obtained by 3 algorithms are actually the same.

(d)

When we change γ to 0.8, the optimal solution is shown in Table 4 and Fig. 3. From the result, the optimal solution is changed.

Table 4: Optimal Value Function, $\gamma = 0.8$

States	1	2	3	4	5
$V^*(x)$	-11.8991	-14.8738	-11.8991	-10.2459	-8.1967
States	6	7	8	9	10
$V^*(x)$	-9.5193	-11.8991	-9.5193	-8.1967	-6.5574
States	11	12	13	14	15
$V^*(x)$	-7.6154	-9.5193	-7.6154	-6.5574	-5.2459
States	16	17	18	19	20
$V^*(x)$	-6.0923	-7.6154	-6.0923	-5.2459	-4.1967
States	21	22	23	24	25
$V^*(x)$	-4.8739	-6.0923	-4.8739	-4.1967	-3.3573

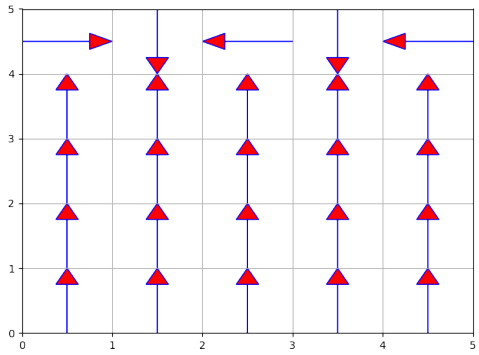


Figure 3: Optimal Policy, $\gamma = 0.8$

Once again, if we change γ to 0.99, the optimal solution is shown in Table 5 and Fig. 4. At this time, the optimal value is different but the policy is the same as $\gamma = 0.9$. Since γ becomes larger, the optimal value decreases (increasing in absolute value).
The reason for same optimal policy is that, when γ is small, we actually do not expect too much upon our next state x', according to Eq. 4. Consequently, it results in that we only consider a small, local area of current state x. For instance, compared Fig. 3 to Fig. 2 and Fig. 4, for the optimal policy of state 9 and 10, in Fig. 3 ($\gamma = 0.8$) those two optimal policies suggest we move to special state B, which offers -5 cost. However, if we consider a little bit "further", since we don't get non-negative cost unless we move off the grid, we can actually move to special state A, which will provide a larger negative cost -15. That's why in Fig. 2 ($\gamma = 0.9$) and Fig.4($\gamma = 0.99$), the optimal policy suggests we head to special state A to get a larger negative cost. This is good but it will also take more iterations for convergence.
In general, with a larger γ we will have better optimal policy but more iterations; with a smaller

γ the value function will converge quickly, but it might result in worse policy. In addition, as γ increases, the final convergent value function will increase in magnitude or absolute value.

Table 5: Optimal Value Function, $\gamma = 0.99$

States	1	2	3	4	5
$V^*(x)$	-201.9998	-204.0402	-201.9998	-199.0402	-197.0498
States	6	7	8	9	10
$V^*(x)$	-199.9798	-201.9998	-199.9798	-197.9800	-196.0002
States	11	12	13	14	15
$V^*(x)$	-197.9800	-199.9798	-197.9800	-196.0002	-194.0402
States	16	17	18	19	20
$V^*(x)$	-196.0002	-197.9800	-196.0002	-194.0402	-192.0998
States	21	22	23	24	25
$V^*(x)$	-194.0402	-196.0002	-194.0402	-192.0998	-190.1788

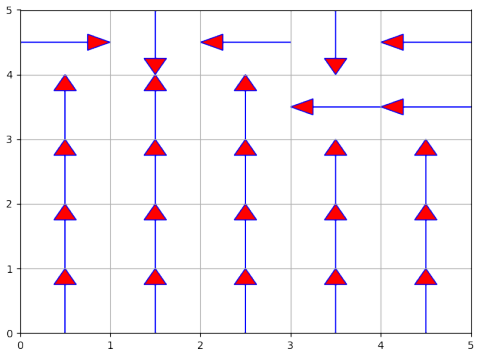


Figure 4: Optimal Policy, $\gamma = 0.99$

Problem 3

1 Problem Formulation

We formulate this Mountain Car problem as a finite-state MDP with unknown motion model and cost. With this idea, we are able to formulate the following definitions:

- State space $\mathcal{X} = \mathcal{S} \times \mathcal{V}$, where $s \in \mathcal{S} = \{s| -1.2 \leq s \leq 0.6\}$ is the position of the car and the velocity $v \in \mathcal{V} = \{v| -0.07 \leq v \leq 0.07\}$.

  The terminal state is $\mathbf{x} = (0.5, v)$ for any $v \in \mathcal{V}$.

  The initial state is $\mathbf{x} = (s_0, 0)$, where s_0 is a random position from -0.6 to -0.4.

  In order to solve this problem in a discrete form we can define two discrete steps d_s and d_v such that $\mathcal{S} = \{-1.2, -1.2 + d_s, -1.2 + 2d_s, ..., 0.6\}$, $\mathcal{V} = \{-0.07, -0.07 + d_v, -0.07 + 2d_v, ..., 0.07\}$.

- Control space $\mathcal{U}(x) = \{0 \text{ (push car left)}, 1 \text{ (no push)}, 2 \text{ (push car right)}\}$ for $x_1 \neq 0.5$.

- Motion model $p_f(\mathbf{x}'|\mathbf{x}, u)$: We do not know it but we can somehow learn it from the environment.

- Cost: We have no idea of the cost so we have to depend on the feedback from the environment.

Our goal is find the optimal policy enabling us to reach the terminal state. Mathematically, we can formulate it as an optimization problem for Q-value function, or a model-free reinforcement learning control problem:

$$Q^*(\mathbf{x}, u) = \min_{\pi} l(\mathbf{x}, u) + \gamma \mathbb{E}_{\mathbf{x}' \sim p_f(.|\mathbf{x},u)}[\min_{u \in \mathcal{U}(x)} Q^*(x', u')] \tag{1}$$

$$\pi^*(x) = \arg\min_{u \in \mathcal{U}(x)} Q^*(\mathbf{x}, u) \tag{2}$$

$$s.t.\ \mathbf{x}_0 = (s_0, 0) \tag{3}$$

where s_0 is a randomly drawn from -0.6 to -0.4.
We are asked to implement both on- and off-policy TD (Temporal Differential) Policy Iteration algorithms to solve this problem.

2 Technical Approach

We want to solve this problem by both on- and off-policy TD algorithms, and then compare their difference. For on-policy algorithm we will implement SARSA algorithm; for off-policy, since Q-Learning is easily adapted from SARSA, it is implemented in this report.

2.1 SARSA Algorithm

SARSA, or State-Action-Reward-State-Action, sends an agent to interact with the environment and updates the policy based on action taken, hence it is known as an on-policy algorithm. The main update is shown as

$$Q(\mathbf{x}, u) \leftarrow Q(\mathbf{x}, u) + \alpha[l(\mathbf{x}, u) + \gamma Q(\mathbf{x}', u') - Q(\mathbf{x}, u)], \tag{4}$$

where α is the learning rate, $0 \leq \gamma \leq 1$ is the discount factor. Q-values represent the possible cost in the next time step for taking action u in state $\mathbf{x}$, plus the discounted future cost received from the next state-action observation.
The algorithm is shown in Algorithm 1.

Algorithm 1: SARSA algorithm

Initialize $Q(\mathbf{x}, u) \forall \mathbf{x} \in \mathcal{X}, u \in \mathcal{U}(\mathbf{x})$ except $Q(\mathbf{x}_T, .) = 0$.
for *Each episode* **do**
  Choose u from $\mathbf{x}$ using policy derived from Q(e.g. ϵ-greedy)
  repeat
    Take action u, observe cost l and $\mathbf{x}'$
    Choose next action u' from $\mathbf{x}'$ using policy derived from Q(e.g. ϵ-greedy)
    $Q(\mathbf{x}, u) \leftarrow Q(\mathbf{x}, u) + \alpha[l(\mathbf{x}, u) + \gamma Q(\mathbf{x}', u') - Q(\mathbf{x}, u)]$
    $\mathbf{x} \leftarrow \mathbf{x}'; u \leftarrow u'$
  until $\mathbf{x}$ *is terminal.*;
end

2.2 Q-Learning Algorithm

Q-learning is an off-policy TD control algorithm. Different from SARSA, as an off-policy algorithm Q-learning improves a policy π that is different from the behavior policy, which is used to generate next state data. Then it explores to find optimal policy greedily. See Algorithm 2.

Algorithm 2: Q-Learning algorithm

Initialize $Q(\mathbf{x}, u) \forall \mathbf{x} \in \mathcal{X}, u \in \mathcal{U}(\mathbf{x})$ except $Q(\mathbf{x}_T, .) = 0$.
for *Each episode* **do**
  repeat
    Choose u from $\mathbf{x}$ using policy derived from Q(e.g. ϵ-greedy)
    Take action u, observe cost l and $\mathbf{x}'$
    $Q(\mathbf{x}, u) \leftarrow Q(\mathbf{x}, u) + \alpha[l(\mathbf{x}, u) + \gamma \min_u Q(\mathbf{x}', u') - Q(\mathbf{x}, u)]$
    $\mathbf{x} \leftarrow \mathbf{x}'$
  until $\mathbf{x}$ *is terminal.*;
end

2.3 ϵ-greedy Exploration

To make sure the exploration of the control space $\mathcal{U}(\mathbf{x})$ must encounter all possible controls at state $\mathbf{x}$ with non-zero probability, we need to develop a strategy for choosing actions. Also, we want to take optimal control more. Therefore, we could use ϵ-greedy. From Eq. 5 we can see that as long as we choose ϵ properly we could explore all possible actions while have relatively larger bias towards optimal control.

$$\pi(\mathbf{x}|u) = \mathbb{P}(u_t = u|\mathbf{x}_t = \mathbf{x}) := \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{U}(\mathbf{x})|} & \text{if } u = \arg\min_{u' \in \mathcal{U}(\mathbf{x})} Q(\mathbf{x}, u') \\ \frac{\epsilon}{|\mathcal{U}(\mathbf{x})|} & \text{otherwise} \end{cases} \tag{5}$$

2.4 Conditions for Convergence

One necessary conditions for convergence to Q^* is that, as episodes continue, the length of each episodes will not change too much (See Fig.4), and Q value will be stable around some value (See Fig.1(a)).

3 Results

3.1 Experiments and Parameters Setting

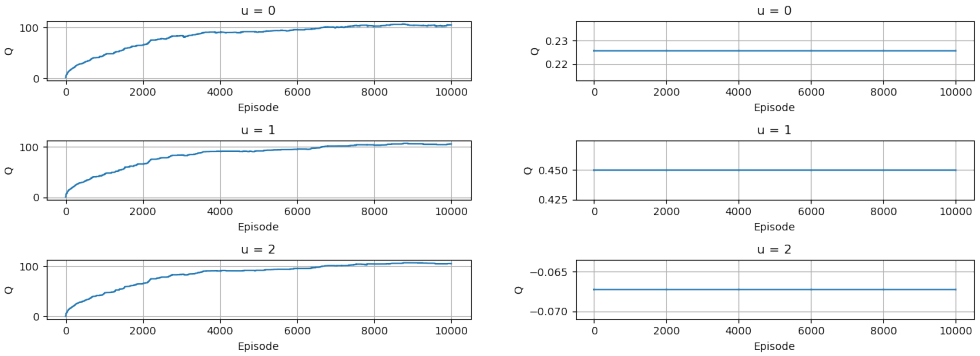
We use `gym` package from OpenAI to construct the Mountain Car problem environment. In particular, there are a few hyper-parameters in this problem: discrete step d_s and d_v, learning rate α, discount factor γ and ϵ for ϵ-greedy. For training, the terminal horizon $T = 5000$ for each episode, and we totally train 10000 episodes for each combination of the parameters; for testing, we consider to use the optimal policy we find to finish the climbing in 200 steps and we will test our optimal policy for 100 different s_0 drawn from -0.6 to -0.4. For initialization, we initialize Q-values by drawing random values from standard normal distribution.

3.2 Results

The quantitative results include Q-value over time and the optimal policy over the state space. Also, we have to try different combinations of hyper-parameters. A qualitative result is included in the zip. file as a video. For learning, the terminal horizon $T = 5000$; for testing, we consider to use our policy to finish the climbing in 200 steps.

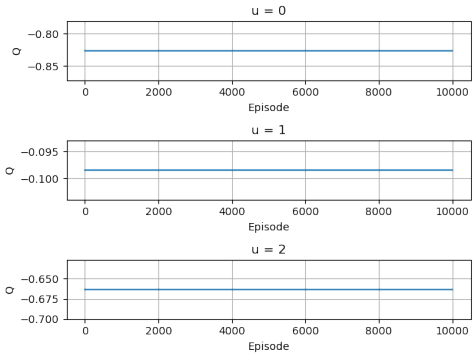
For Q-values, in order to see the processing over time, we choose several representative states to plot the figures. The states are: $\mathbf{x} = (0, 0), (-1.0, 0.05), (0.25, -0.05)$. The figures are shown in Fig.1 and Fig.2 for on- and off-policy algorithms, respectively. The hyper-parameters are as follows: $d_s = 0.1, d_v = 0.01, \gamma = 1.0, \alpha = 0.1, \epsilon = 0.1$. As we can see, the Q-value will rise and then converges to some value. For $Q(-1.0, 0.05)$ and $Q(0.25, -0.05)$, these two states are hardly reach so their values keep the same over the episode.

The optimal policy is shown in Fig.3, with the same hyper-parameters: $d_s = 0.1, d_v = 0.01, \gamma = 1.0, \alpha = 0.1, \epsilon = 0.1$. We can see that, for position $s < 0$ and velocity $v < 0$, the optimal policy is "push left", this is reasonable since while the car are on the left side it is better to push left to get the potential of the gravity; for position $s < 0$ and velocity $v > 0$, the policy is "no push" or "push right", this is because we



(a) Q(0,0)

(b) Q(-1.0, 0.05)



(c) Q(0.25, -0.05)

Figure 1: Q-values over Episode, SARSA

There are also some results with different parameter combinations.

3.3 Parameters Analysis

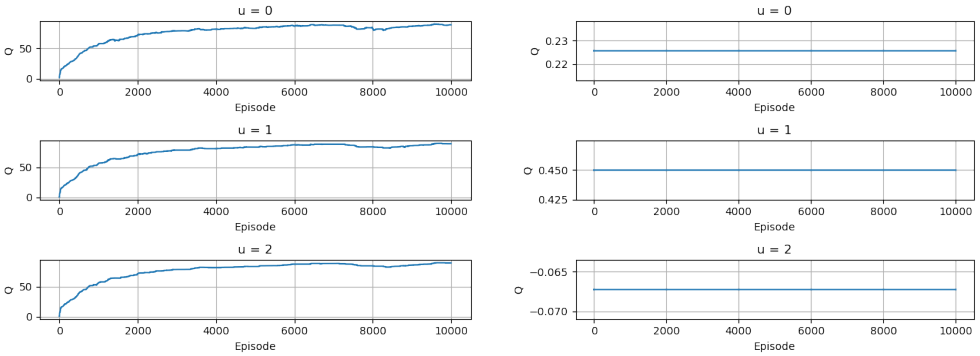
Actually, there are quite a few hyper-parameters in this problem. Since this is a problem with rather small control space, we can discuss the effect of the parameters on the convergence of the Q-values and the number of steps to reach the goal in the test.

For discrete steps d_s and d_v, smaller those two parameters are, larger the state space becomes, which means we need more episodes for training.

For discount factor γ, it is clear that the larger γ is, the further we see in the next state, which will provide a much more accurate policy. A small γ will decrease Q-values. See Table 1. The steps are taken mean values of 100 tests. Except γ, other parameters keep the same, i.e. $d_s = 0.1, d_v = 0.01, \alpha = 0.1, \epsilon = 0.1$.

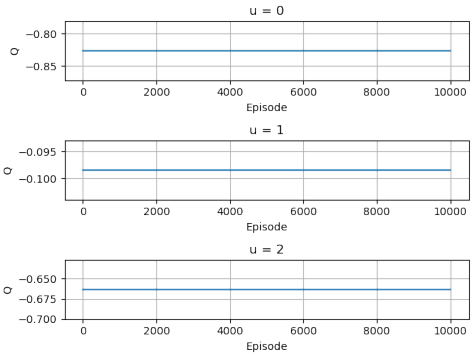
For learning rate α, by my experiment if we want to have fewer steps to reach the goal, we need a smaller α with the same number of training episodes to converge to the "more" optimal point. However, a small α will decrease the speed of convergence. See Table. 2.

For ϵ, it decides the degree how sparse for exploration, see Table 3. From Eq. 5, the smaller ϵ,



(a) Q(0,0)

(b) Q(-1.0, 0.05)



(c) Q(0.25, -0.05)

Figure 2: Q-values over Episode, Q-Learning

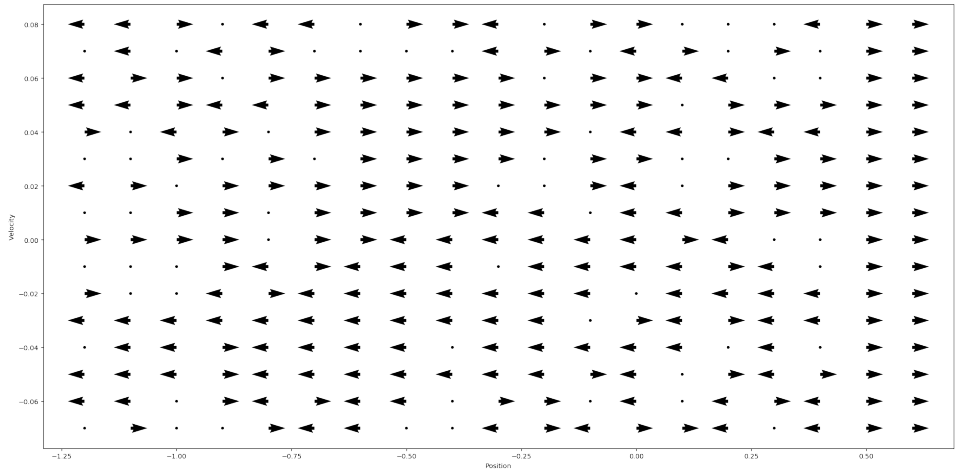
Table 1: Different γ For Testing

γ	1.0	0.9	0.7	0.5
Ave. Steps (SARSA)	143.80	144.96	225.04	241.43
Ave. Steps (Q-Learning)	170.31	213.36	227.44	235.19

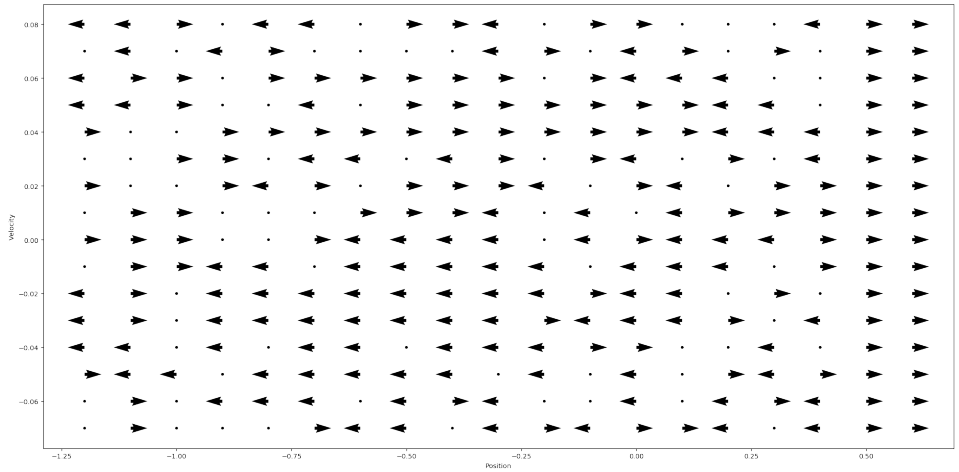
Table 2: Different α For Testing

α	0.1	0.3	0.7	0.9
Ave. Steps (SARSA)	143.80	226.04	239.78	301.0
Ave. Steps (Q-Learning)	177.11	199.59	236.71	300.89

the larger chance we choose the current "optimal" control, which means we trust optimal policy more. However, this would narrow other control options. Since the control space of this problem is not big (only 3 options for each state except for terminal states), this parameter doesn't have big influence on the final results given 10000 episodes in training if we are using SARSA. However, if we are trying to use Q-Learning, since we choose actions greedily, a larger ϵ will decrease the speed



(a) Optimal Policy, SARSA



(b) Optimal Policy, Q-Learning

Figure 3: Optimal Policy, $\rightarrow$ means "Push Right", $\leftarrow$ means "Push left", $\cdot$ means "No Push"

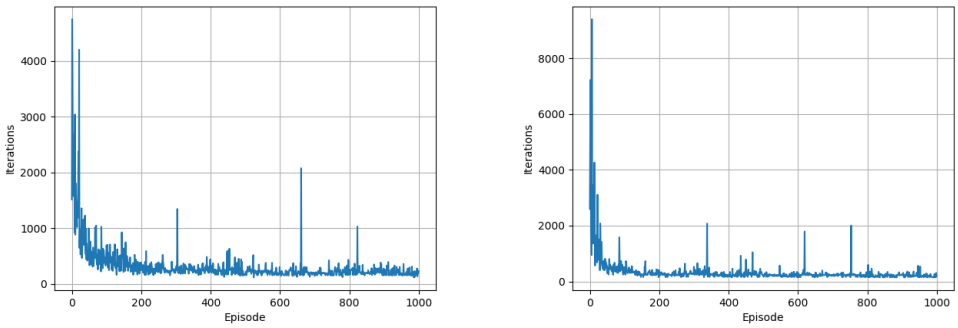
of convergence.

Table 3: Different ϵ For Testing

ϵ	0.1	0.3	0.7	0.9
Ave. Steps (SARSA)	143.80	175.09	165.1	183.53
Ave. Steps (Q-Learning)	177.11	178.15	236.71	300.89

3.4 SARSA and Q-Learning

A difference between two algorithms is that, SARSA explores the control space using ϵ-greedy to update Q while Q-Learning chooses control with ϵ-greedy but greedily updates Q using current best policy. From Fig. 4 we can see, the number of iteration of each episode of Q-Learning is nearly stable after the very beginning, while SARSA still oscillates since it is using ϵ-greedy to choose random policy. However, it is hard to say which one converges faster, since we don't know for a problem, it is better to trust what we have (Q-Learning) or explore more to introduce random action (SARSA). In general, SARSA keeps exploration when it tries to update Q, however, Q-learning has a bias towards the best control for now, which is more greedy.



(a) Training Processing, SARSA

(b) Training Processing, Q-Learning

Figure 4: Training Episodes

Problem 4

1 Problem Formulation

1.1 MDP Formulation for Inverted Pendulum

Inverted Pendulum (IP) optimization control problem can also be solved by formulating it into an infinite horizon MDP with continuous state space and control space.

- State space: $\mathbf{x} \in \mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 = [-\pi, \pi] \times [-v_{max}, v_{max}]$, where $v_{max} > 0$ is the max angular velocity, $x_1 \in \mathcal{X}_1$ is the angle of the pendulum and $x_2 \in \mathcal{X}_2$ is the angular velocity.
- Control space: $\mathcal{U} = [-u_{max}, u_{max}]$, where $u_{max} > 0$ is the max torque.
- Motion model: $(\mathbf{x}'|\mathbf{x}, u) \sim N(\mathbf{x} + f(\mathbf{x}, u)dt, \sigma\sigma^T dt)$, where

$$f(\mathbf{x}, u) = \begin{bmatrix} x_2 \\ a\text{sin}x_1 - bx_2 + u \end{bmatrix}, \tag{1}$$

  $\sigma \in \mathbb{R}^2$ is covariance, $a > 0$ summarizes the effects of the effects of gravity, mass and length of the pendulum, $b > 0$ is the damping and friction, dt is the differential of time t.

- Stage cost. The stage cost at each state $\mathbf{x}$ and control u is

$$l(\mathbf{x}, u) = 1 - \exp(k\text{cos}x_1 - k) + \frac{r^2}{2}u^2. \tag{2}$$

- Terminal cost $\mathfrak{q} = 0$. In fact, there is no terminal states.

Then we need to solve Bellman Equation for optimization:

$$V^*(\mathbf{x}) = \min_{\pi} l(\mathbf{x}, u) + \gamma \mathbb{E}_{\mathbf{x}' \sim p_f(.|\mathbf{x},u)}[\min_{u \in \mathcal{U}(x)} V^*(x')] \tag{3}$$

$$\pi^*(x) = \arg\min_{u \in \mathcal{U}(x)} V^*(\mathbf{x}) \tag{4}$$

$$s.t.\ \mathbf{x}_0 = (x_{10}, 0) \tag{5}$$

In order to actually solve this problem, one possible way is to discretize the state space and control space and then using interpolation techniques to fill out remaining the rest of the space.
So we will discretize state space $\mathcal{X}$ and control space $\mathcal{U}$ as discrete step d_{x_1}, d_{x_2} and d_u:

$$\mathcal{X} = \{-\pi, -\pi + d_{x_1}, -\pi + 2d_{x_1}, ..., \pi\} \times \{-v_{max}, -v_{max} + d_{x_2}, -v_{max} + 2d_{x_2}, ..., v_{max}\} \tag{6}$$

and

$$\mathcal{U} = \{-u_{max}, -u_{max} + d_u, -u_{max} + 2d_u, ..., u_{max}\}, \tag{7}$$

then using fixed time step δt to calculate the motions. Hence the model then becomes a finite-space MDP.
After this, we can use interpolation techniques to fill up to make the spaces more smooth.

1.2 Interpolation for Policy

Since the state $\mathbf{x} \in \mathbb{R}^2$, we consider this problem as a 2D interpolation problem for policy $\pi(\mathbf{x})$ over the discrete state space as Eq. 6 states.
After interpolation, we will have our policy over a precise and "continuous" state space $\mathcal{X}' = \{-\pi, -\pi + d_{x_1'}, -\pi + 2d_{x_1'}, ..., \pi\} \times \{-v_{max}, -v_{max} + d_{x_2'}, -v_{max} + 2d_{x_2'}, ..., v_{max}\}$, where $d_{x_1'} < d_{x_1}, d_{x_2'} < d_{x_2}$.

2 Technical Approach

Since the optimal cost of discounted problem satisfies the Bellman Equation (Eq. 3) via its equivalence to Stochastic Shortest Problem (SSP), we can use Value Iteration (VI) or Policy Iteration (PI) for solution.

2.1 Value Iteration

VI starts at the terminals and then works backwards (Dynamic Programming) although there is no real "terminals" probably. It usually initializes V_0 and updates V for convergence as Eq. 8. The pseudocode is shown in Algorithm 1. Usually, a threshold θ is required for termination of the algorithm.

$$V_k(\mathbf{x}) \leftarrow \min_{u \in \mathcal{U}} [l(\mathbf{x}, u) + \gamma \sum_{x'} p_f(\mathbf{x}'|\mathbf{x}, u) V_{k-1}(x')] \tag{8}$$

Algorithm 1: VI Algorithm

Initialize $V(\mathbf{x})$, given threshold θ.
$k \leftarrow 0$
repeat
  $k \leftarrow k + 1$
  for *each state* **x do**
    $V_k(\mathbf{x}) \leftarrow \min_{u \in \mathcal{U}} [l(\mathbf{x}, u) + \gamma \sum_{x'} p_f(\mathbf{x}'|\mathbf{x}, u) V_{k-1}(x')]$
  end
until $max|V_k(\mathbf{x}) - V_{k-1}(\mathbf{x})| < \theta$;
for *each state* **x do**
  $\pi(\mathbf{x}) = \arg\min_{u \in \mathcal{U}} [l(\mathbf{x}, u) + \gamma \sum_{x'} p_f(\mathbf{x}'|\mathbf{x}, u) V_{k-1}(x')]$
end
return π, V_k

2.2 Policy Iteration

PI starts at a different point of view: what if we update V given a policy $\pi(\mathbf{x})$? This is valid due to the Policy Evaluation Theorem.
In general, we want solve equations:

$$(I - \tilde{\mathbf{P}})\mathbf{v} = \mathbf{l} \tag{9}$$

for $\mathbf{v}$, where $\mathbf{v}_i = V(i)$, $\mathbf{l}_i = l(i, \pi(i))$, $\tilde{\mathbf{P}}_{ij} = \tilde{p}_f(j|i, \pi(i))$. Since the uniqueness of the solution to $\mathbf{v}$, we can get iterative solution to Eq. 9 by:

$$\begin{aligned}
&\mathbf{v}_1 = \mathbf{l} + \tilde{\mathbf{P}}\mathbf{v}_0 \\
&\mathbf{v}_2 = \mathbf{l} + \tilde{\mathbf{P}}\mathbf{v}_1 = \mathbf{l} + \tilde{\mathbf{P}}\mathbf{l} + \tilde{\mathbf{P}}^2\mathbf{v}_0 \\
&... \\
&\mathbf{v}_T = (I + \tilde{\mathbf{P}} + \tilde{\mathbf{P}}^2 + ... + \tilde{\mathbf{P}}^{T-1})\mathbf{l} + \tilde{\mathbf{P}}^T\mathbf{v}_0 \\
&... \\
&\mathbf{v}_\infty \to (I - \tilde{\mathbf{P}})^{-1}\mathbf{l}
\end{aligned}$$

if $I - \tilde{\mathbf{P}}$ is invertible. This iterative processing is called Policy Evaluation.
Therefore, the procedure is first we do Policy Evaluation given policy π until V is convergent to solve Eq. 9, then we improve policy π according to V by

$$\pi(\mathbf{x}) = \arg\min_{u \in \mathcal{U}} [l(\mathbf{x}, u) + \gamma \sum_{x'} p_f(\mathbf{x}'|\mathbf{x}, u) V_{k-1}(\mathbf{x}')] \tag{10}$$

until the policy is stable for all states.

Algorithm 2: PI Algorithm

Initialize $V(\mathbf{x})$ and policy $\pi(\mathbf{x})$, given threshold θ.
$k \leftarrow 0$
repeat
  $k \leftarrow k + 1$
  for *each state* $\mathbf{x}$ **do**
    $V_k^\pi(\mathbf{x}) \leftarrow l(\mathbf{x}, \pi(\mathbf{x})) + \gamma \sum_{x'} p_f(\mathbf{x}'|\mathbf{x}, \pi(\mathbf{x}) V_{k-1}^\pi(\mathbf{x}')$
  end
until $max|V_k(\mathbf{x}) - V_{k-1}(\mathbf{x})| < \theta$;
repeat
  for *each state* $\mathbf{x}$ **do**
    $\pi'(\mathbf{x}) \leftarrow \pi(\mathbf{x})$
    $\pi(\mathbf{x}) = \arg\min_{u \in \mathcal{U}} [l(\mathbf{x}, u) + \gamma \sum_{x'} p_f(\mathbf{x}'|\mathbf{x}, u) V_{k-1}(\mathbf{x}')]$
  end
until $\pi(\mathbf{x}) = \pi'(\mathbf{x})$ *for every state*;
return π, V_k

2.3 Cubic Interpolation

In this problem we simply choose cubic interpolation to expand policy over state space. Cubic interpolation is the simplest method taht offers true continuity between the segments. It computes a spline where each piece is a third-degree polynomial specified in Hermite form, i.e by its values and first derivatives at the end points of the correspondence domain interval. A brief example of cubic interpolation is as follows[1]:
On the unit interval [0, 1], given a start point p_0 at $t = 0$ and an ending point p_1 at $t = 1$ with starting tangent m_0 at $t = 0$ and the ending tangent m_1 at $t = 1$, the polynomial can be defined by

$$p(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 + 3t^2)p_1 + (t^3 - t^2)m_1, \tag{11}$$

where $t \in [0, 1]$.
More generally, we can adapt Eq. 11 to arbitrary interval:

$$p(x) = h_{00}(t)p_k + h_{10}(t)(x_{k+1} - x_k)m_k + h_{01}(t)p_{k+1} + h_{11}(t)(x_{k+1} - x_k)m_{k+1} \tag{12}$$

with $t = \frac{x - x_k}{x_{k+1} - x_k}$ and h refers to the basis functions (See Table 1).

$$B_k(t) = \binom{3}{k} t^k (1 - t)^{3-k} \tag{13}$$

Table 1: Basis Functions, the "expanded" column shows the representation used in example above. The "factorized" column shows immediately, that h_{10} and h_{11} are zero at the boundaries.The "Bernstein" column shows the decomposition of the Hermite basis functions into Bernstein polynomials of order 3 shown in Eq.13

	expanded	factorized	Bernstein
$h_{00}(t)$	$2t^3 - 3t^2 + 1$	$(1 + 2t)(1 - t)^2$	$B_0(t) + B_1(t)$
$h_{10}(t)$	$t^3 - 2t^2 + t$	$t(1 - t)^2$	$\frac{1}{3}B_1(t)$
$h_{01}(t)$	$-2t^3 + 3t^2$	$t^2(3 - 2t)$	$B_3(t) + B_2(t)$
$h_{11}(t)$	$t^3 - t^2$	$t^2(t - 1)$	$-\frac{1}{3}B_2(t)$

For this problem, we need to consider using 2D cubic interpolation, fortunately, which can be done directly by `scipy.interpolate.interp2d(**, kind='cubic')`.

3 Results and Discussion

3.1 Environment and Experiment Setting

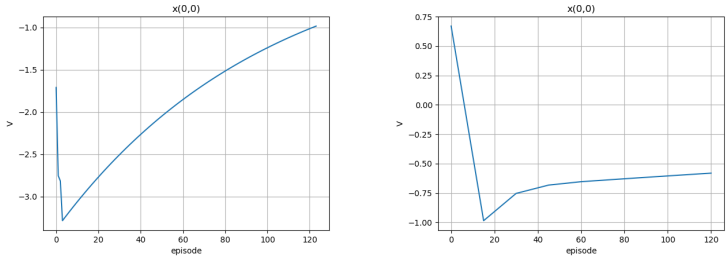
For pendulum, assume $a = 4.0$, damping $b = 1.0$, shape of the cost $k = 3.0$, and scale of the cost $r = 0.001$. The covariance matrix $\Sigma = \sigma\sigma^T = 0.001 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Time step $\delta t = 0.5$.
For discretization, $d_{x_1} = 0.1$, $d_{x_2} = 0.2$, $d_u = 0.25$, $v_{max} = 4$, $u_{max} = 3$, then the discrete state space $\mathcal{X} = \{-\pi, -\pi + 0.1, ..., \pi\} \times \{-4.0, -3.8, -3.6, ..., 4.0\}$, the discrete control space $\mathcal{U} = \{-3.0, -2.75, ..., 3.0\}$.
For VI and PI, the threshold $\theta = 1e - 3$ for termination, discount factor $\gamma = 0.99$. Value functions V is initialized by random values from a standard Gaussian distribution.

By this setting, $\frac{2v_{max}}{n_2}\delta t/\frac{2\pi}{n_1} = 0.9937 \approx 1$. After we get the optimal policy, a test is executed to test the policy. After VI or PI, a cubic interpolation is implemented, making $d_{x_1} = 0.01$, $d_{x_2} = 0.1$.

3.2 Results

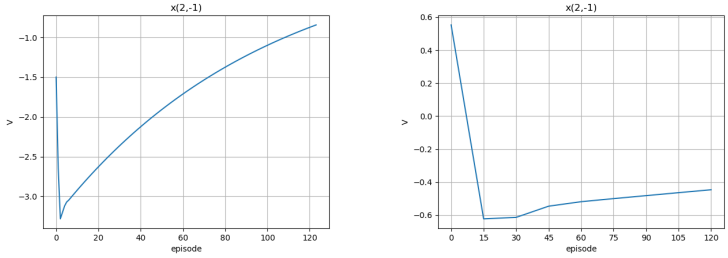
Two qualitative results can be found in two video .mp4 files for VI and PI, respectively.

Value function V(x) over episodes is shown in Fig.3 and Fig.4. As we can see, value functions gradually converge and it looks like a "convex" function centered at (0,0). This makes sense since we want to push the pendulum to the origin position. For a specific set of states $\mathbf{x} \in \{(0,0),(2,-1)\}$, the trending of V over the episodes is shown in Fig.1 and Fig.2.



(a) Value Iteration (b) Policy Iteration

Figure 1: V((0,0))

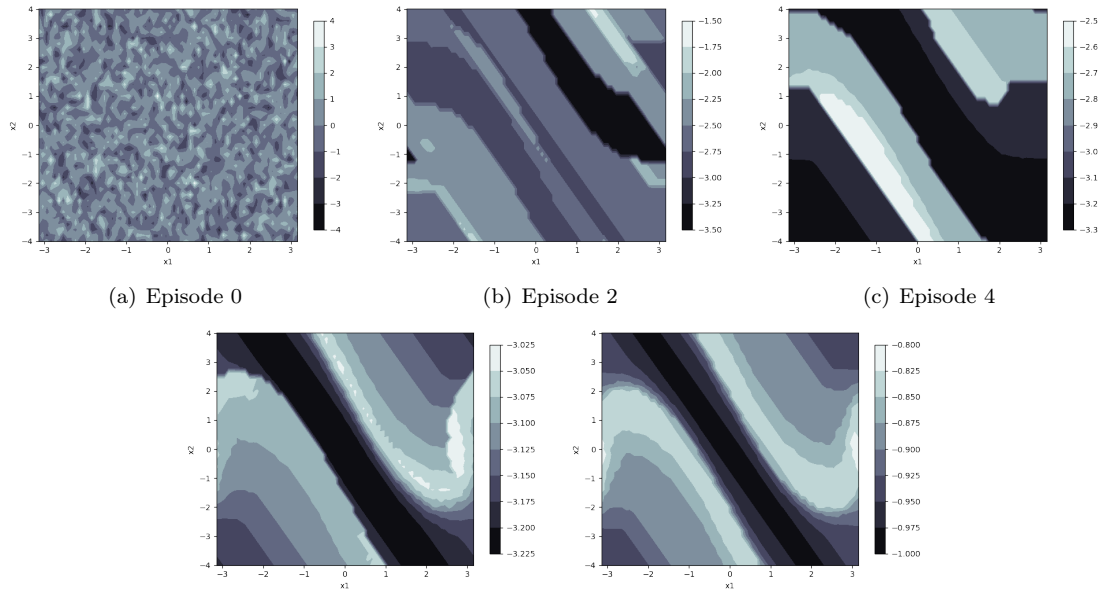


(a) Value Iteration (b) Policy Iteration

Figure 2: V((2,-1))

From Fig.1, Fig.2, Fig.3 and Fig.4 we can see that VI will converge a little bit faster for this problem.

The policy $\pi(x)$ after interpolation over the discrete space is shown in Fig.5. We can see that the final optimal policy via two algorithms are almost the same. The results show that with the policy is centered around the origin $\mathbf{x} = (0,0)$. There is little difference between VI and PI in the final result.



(a) Episode 0 (b) Episode 2 (c) Episode 4

(d) Episode 6 (e) Final

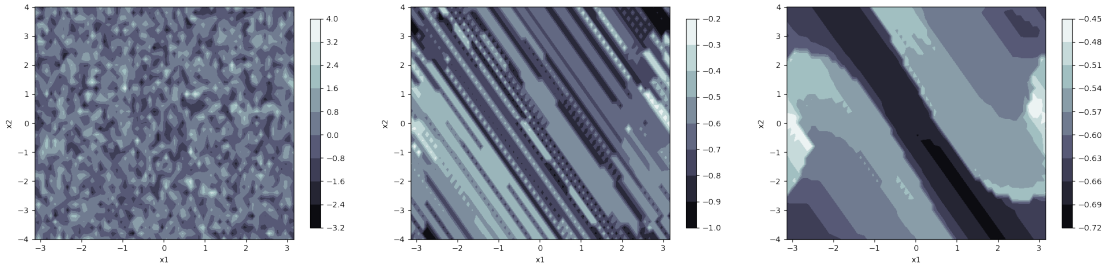
Figure 3: V(x) over State Space, Value Iteration

3.3 Discussion

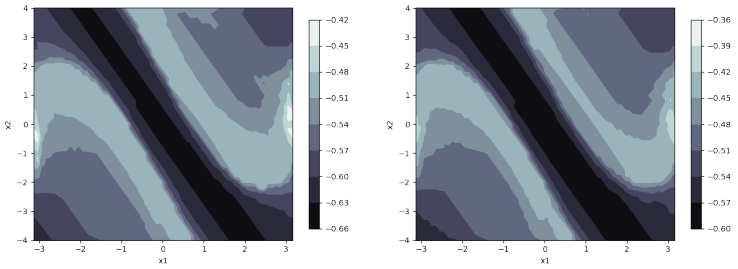
Parameters Setting In this problem, we need to define parameters of pendulum by ourselves. The most critical one is r (Eq.2), which is used to scale the control cost. The purpose of r is that, we cannot let the cost contributed by control u exceeds the part from angle displacement, while we also want to control influences the cost somehow. If the cost contributed by u is more significant than the state x, u will suppress the angle change, which will make V has little to do with the state x but dependent on u. Once we choose a larger r, it is better to use a larger k to amplify the contribution of state. By my observation, $r = 0.001$, $k = 3$ are appropriate for $a = 4.0$.

Another thing I notice is the time step δt and the control space discretization. If δt is small, then we need a more "continuous" and larger control space, i.e a smaller d_u and a larger u_{max}. For example, if $\delta t = 0.05$ and $d_u = 0.1, u_{max} = 2$, by Eq.1, we can see that the contribution of u to $f(\mathbf{x}, u)$ is $u_{max} \times \delta t = 0.1$ at most. Meanwhile, if the discretization of state space in x_2 is large, e.g. $d_{x_2} = 0.2 > 0.1$, then the contribution of u will probably be ignored. This will cause the motion model is significantly dependent on $ax_1 - bx_2$, which is the gravity (almost) and the damping.

How noise influence the motion? Noise level is related to the system. If a, which is related to the gravity, mass and lengthx, is large, then it can take more noise. Assume for motion model, the covariance matrix $\Sigma = f \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, where$f > 0$. In my case, $a = 5$, then $f > 1.0$ the motion model will be influenced badly and we cannot achieve stable equilibrium pretty well.

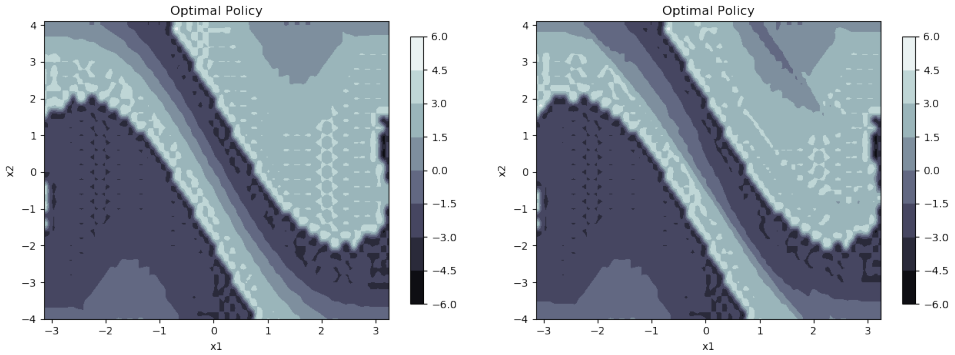


(a) Episode 0 (b) Episode 2 (c) Episode 4



(d) Episode 6 (e) Final

Figure 4: V(x) over State Space, Policy Iteration



(a) Optimal Policy, Value Iteration (b) Optimal Policy, Policy Iteration

Figure 5: Optimal Policy with Interpolation, colorbar shows the values of control inputs

4 Acknowledge

Thanks for Shih-Chen Lin and Shuo Xu for discussion about parameters of the pendulum.

References

[1] WikiPedia: https://en.wikipedia.org/wiki/Cubic_Hermite_spline