ECE276B HW3 Shan Huang A53279925

1. (a) Formulate an infinite horizon stochastic shortest path problem with $\gamma = 1$.

Consider \$10000 as 1 unit

State Space: $\mathcal{X} := \{0, 1, 2, 3, 4, 11, 12, 22\}$

Starting State: $x_s = 1$

Terminal State: $x_T \in \{0, 3, 4, 11, 12, 22\} = \mathcal{X}_T$

Control Space: $\mathcal{U} := \{(r,1), (r,2), (b,1), (b,2)\}$ where r - bet on red b - bet on black

Motion Model: $x' \sim P_f(x' | x, u) =$

$P(x'=0 | x=1, u=(r,1)) = 0.3$
$P(x'=2 | x=1, u=(r,1)) = 0.7$
$P(x'=0 | x=2, u=(r,2)) = 0.3$
$P(x'=4 | x=2, u=(r,2)) = 0.7$
$P(x'=11 | x=2, u=(r,1)) = 0.3$
$P(x'=3 | x=2, u=(r,1)) = 0.7$
$P(x'=0 | x=1, u=(b,1)) = 0.8$
$P(x'=11 | x=1, u=(b,1)) = 0.2$
$P(x'=0 | x=2, u=(b,2)) = 0.8$
$P(x'=22 | x=2, u=(b,2)) = 0.2$
$P(x'=1 | x=2, u=(b,1)) = 0.8$
$P(x'=12 | x=2, u=(b,1)) = 0.2$

Stage Cost: $l(x,u) = 0$ for $u \in \mathcal{U}$, $x \in \mathcal{X}$

Terminal Cost: $q(x_T) = -x_T$

(b) Consider $\mathcal{U} := \{(r,1)\}$ (play red and bet \$10000 on every non-terminal state)

Initialize $V_0^\pi(x) = \begin{cases} 0 & \text{for } x \in \{1,2\} \\ -x & \text{for } x \in \{0,3,4,11,12,22\} \end{cases}$ (virtual terminal state)

① Crude estimate:

Since $V_1^\pi(x) = l(x,u) + \sum_{x' \in \mathcal{X} \setminus \{\blacksquare\}} P_f(x' | x, u) V_0^\pi(x')$, $\forall x \in \mathcal{X} \setminus \{\blacksquare\}$ (0)

$\therefore V_1^\pi(1) = P(x'=2 | x=1, u=(r,1)) \times V_0^\pi(2)$ + $P(x'=0 | x=1, u=(r,1)) \times V_0^\pi(0) = 0$

$V_1^\pi(2) = P(x'=1 | x=2, u=(r,1)) \times V_0^\pi(1) + P(x'=3 | x=2, u=(r,1)) \times V_0^\pi(3)$

$= -3 \times 0.7$

$= -2.1$

for $x \in \mathcal{X}_T$, since $P(x'=x_T | x=x_T, u) = 1$

$\therefore V_1^\pi(x_T) = 1 \times V_0^\pi(x_T) = V_0^\pi(x_T)$

2.

In conclusion, $\bar{V}^{\pi}(x) = \begin{cases} V_0^{\pi}(x) = -x & \text{for } x \in \mathcal{X}_T \\ 0 & \text{when } x = 1 \\ -2.1 & \text{when } x = 2 \end{cases}$

② Precise estimate:

By solving the policy evaluation equation exactly:

Since $V_{t+1}^{\pi}(1) = P(x'=2|x=1, u=(\Gamma,1)) \cdot V_t^{\pi}(2) + P(x'=0|x=1, u=(\Gamma,1)) \cdot \underbrace{V_t^{\pi}(0)}_{0}$

$= 0.7 \cdot V_t^{\pi}(2)$

$V_{t+1}^{\pi}(2) = P(x'=1|x=2, u=(\Gamma,1)) \cdot V_t^{\pi}(1) + P(x'=3|x=2, u=(\Gamma,1)) \cdot V_t^{\pi}(3)$

$= 0.3 \cdot V_t^{\pi}(1) - 0.7 \times 3$

$= 0.3 V_t^{\pi}(1) - 2.1$

$V_{t+1}^{\pi}(x_T) = V_0^{\pi}(x_T) = -x_T \quad \text{for } x_T \in \mathcal{X}_T$

As $t \to \infty$, V_t^{π} converges to $\hat{V}^{\pi}$

$\therefore \begin{cases} \hat{V}^{\pi}(1) = 0.7 \cdot \hat{V}^{\pi}(2) \\ \hat{V}^{\pi}(2) = 0.3\, \hat{V}^{\pi}(1) - 2.1 \\ \hat{V}^{\pi}(x_T) = -x_T \end{cases} \Rightarrow \begin{cases} \hat{V}^{\pi}(x_1) \approx -1.861 \\ \hat{V}^{\pi}(x_2) \approx -2.658 \end{cases}$

In conclusion, $\hat{V}^{\pi}(x) = \begin{cases} -x & \text{for } x \in \mathcal{X}_T \\ -1.861 & \text{when } x = 1 \\ -2.658 & \text{when } x = 2 \end{cases}$

(c)

① For $\bar{V}^{\pi}$

since $\bar{\pi}'(x) = \underset{u \in U}{\arg\min} \left[l(x,u) + \sum_{x' \in \mathcal{X} \setminus \{0\}} P_f(x'|x;u) \bar{V}^{\pi}(x') \right] \quad \forall x \in \mathcal{X} \setminus \{0\}$ (virtual terminal state)

$\therefore \bar{\pi}'(1) = \underset{u \in U}{\arg\min} \{ P(x'=0|x=1, u=(\Gamma,1)) \bar{V}^{\pi}(0) + P(x'=2|x=1, u=(\Gamma,1)) \bar{V}^{\pi}(2),$

$P(x'=0|x=1, u=(b,1)) \bar{V}^{\pi}(0) + P(x'=11|x=1, u=(b,1)) \bar{V}^{\pi}(11) \}$

$= \underset{u \in U}{\arg\min} \{ 0.7 \times (-2.1), 0.2 \times (-11) \}$

$= \underset{u \in U}{\arg\min} \{ -1.47, -2.2 \}$

$= (b, 1)$

similarly

$\bar{\pi}'(2) = \underset{u \in U}{\arg\min} \{ P(x'=11|x=2, u=(\Gamma,11) \bar{V}^{\pi}(1) + P(x'=3|x=2, u=(\Gamma,11) \bar{V}^{\pi}(3),$

$P(x'=0|x=2, u=(\Gamma,2)) \bar{V}^{\pi}(0) + P(x'=4|x=2, u=(\Gamma,2)) \bar{V}^{\pi}(4),$

$P(x'=1|x=2, u=(b,1))\bar{V}^{\pi}(1) + P(x'=12|x=2, u=(b,1))\bar{V}^{\pi}(12)$,

$P(x'=0|x=2, u=(b,2))\bar{V}^{\pi}(0) + P(x'=22|x=2, u=(b,2))\bar{V}^{\pi}(22)\}$

$= \underset{u\in U}{\operatorname{argmin}}\{0.3\times 0 + 0.7\times(-3),\ 0.7\times(-4),\ 0.8\times 0 + 0.2\times(-12),\ 0.2\times(-22)\}$

$= \underset{u\in U}{\operatorname{argmin}}\{-2.1,\ -2.8,\ -2.4,\ -4.4\}$

$= (b,2)$

In conclusion, $\bar{\pi}'(x) = \begin{cases} (b,1) & \text{when } x=1 \\ (b,2) & \text{when } x=2 \end{cases}$

② For $\hat{V}^{\pi}$

similarly, $\hat{\pi}'(1) = \underset{u\in U}{\operatorname{argmin}}\{P(x'=0|x=1, u=(r,1))\hat{V}^{\pi}(0) + P(x'=2|x=1, u=(r,1))\hat{V}^{\pi}(2)$,

$P(x'=0|x=1, u=(b,1))\hat{V}^{\pi}(0) + P(x'=11|x=1, u=(b,1))\hat{V}^{\pi}(11)\}$

$= \underset{u\in U}{\operatorname{argmin}}\{0.7\times(-2.658),\ 0.2\times(-11)\}$

$= \underset{u\in U}{\operatorname{argmin}}\{-1.8606,\ -2.2\}$

$= (b,1)$

$\hat{\pi}'(2) = \underset{u\in U}{\operatorname{argmin}}\{P(x'=1|x=2, u=(r,1))\hat{V}^{\pi}(1) + P(x'=3|x=2, u=(r,1))\hat{V}^{\pi}(3)$,

$P(x'=0|x=2, u=(r,2))\hat{V}^{\pi}(0) + P(x'=4|x=2, u=(r,2))\hat{V}^{\pi}(4)$,

$P(x'=1|x=2, u=(b,1))\hat{V}^{\pi}(1) + P(x'=12|x=2, u=(b,1))\hat{V}^{\pi}(12)$,

$P(x'=0|x=2, u=(b,2))\hat{V}^{\pi}(0) + P(x'=22|x=2, u=(b,2))\hat{V}^{\pi}(22)\}$

$= \underset{u\in U}{\operatorname{argmin}}\{0.3\times(-1.861) + 0.7\times(-3),\ 0.7\times(-4),\ 0.8\times(-1.861) + 0.2\times(-12),\ 0.2\times(-22)\}$

$= \underset{u\in U}{\operatorname{argmin}}\{-2.6583,\ -2.8,\ -3.8888,\ -4.4\}$

$= (b,2)$

In conclusion, $\hat{\pi}'(x) = \begin{cases} (b,1) & \text{when } x=1 \\ (b,2) & \text{when } x=2 \end{cases}$

Therefore, the policy $\bar{\pi}'(x)$ and $\hat{\pi}'(x)$ are the same, that is to bet all your money on black, which is very different from π, which is to bet least money on red. We can say π is the most conservative policy while $\bar{\pi}'(x)$ and $\hat{\pi}'(x)$ are both the most bold policy.

4.

(d) Using value iteration

since $V_{t+1}(x) = \min_{u \in U} \{ l(x,u) + \sum_{x' \in X \cup \{0\}} P_t(x'|x,u) V_t(x') \}$

notice for $x \in X_T$, $V_k(x) = -x$ (known terminal cost)

$= \min \{ \sum_{x' \in X \cup \{0\}} P_t(x'|x,u) V_t(x') \}$

$\therefore V_{t+1}(1) = \min_{u \in U} \{ P(x'=0|x=1, u=(r,1)) \cdot V_t(0) + P(x'=2|x=1, u=(r,1)) \cdot V_t(2),$

$P(x'=0|x=1, u=(b,1)) \cdot V_t(0) + P(x'=11|x=1, u=(b,1)) \cdot V_t(11) \}$

$= \min_{u \in U} \{ 0.7 V_t(2), -2.2 \}$

similarly,

$V_{t+1}(2) = \min_{u \in U} \{ P(x'=1|x=2, u=(r,1) V_t(1) + P(x'=3|x=2, u=(r,1)) V_t(3),$

$P(x'=0|x=2, u=(r,2) V_t(0) + P(x'=4|x=2, u=(r,2)) V_t(4),$

$P(x'=1|x=2, u=(b,1) V_t(1) + P(x'=12|x=2, u=(b,1)) V_t(12),$

$P(x'=0|x=2, u=(b,2) V_t(0) + P(x'=22|x=2, u=(b,2)) V_t(22) \}$

$= \min_{u \in U} \{ 0.3 V_t(1) - 2.1, -2.8, 0.8 V_t(1) - 2.4, -4.4 \}$

as $t \to \infty$, V_t converges to V^*

Suppose $V^*(1) = -2.2 \Rightarrow V^*(2) = \min\{-2.76, -2.8, -4.16, -4.4\} = -4.4$

$\therefore V^*(1) = \min\{-3.08, -2.2\} = -3.08$ contradicts $V^*(1) = -2.2$

$\therefore V^*(1) = 0.7 V^*(2)$ ①

Suppose $V^*(2) = -4.4 \Rightarrow V^*(1) = 0.7 \times -4.4 = -3.08$

$\therefore V^*(2) = \min\{-3.024, -2.8, -4.864, -4.4\} = -4.864$ contradicts $V^*(2) = -4.4$

~~Suppose~~ since $V^*(x) \leq 0$ $\therefore 0.8 V^*(1) - 2.4 < 0.3 V^*(1) - 2.1$

$\therefore V^*(2) = 0.8 V^* - 2.4$ ②

from ①②, we get $\begin{cases} V^*(1) \approx -3.82 \\ V^*(2) \approx -5.45 \end{cases} \Rightarrow \begin{cases} \pi^*(1) = (r,1) \\ \pi^*(2) = (b,1) \end{cases}$

Since $x_5 = 1$, $V^*(1) = -3.82 \Rightarrow$ reward $= \$38200$

In conclusion, if I have \$10000, I should bet \$10000 on red;

if I have \$20000, I should bet \$10000 on black.

The money I expect to walk away with is \$38200.

There is going to be a celebration.

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

2. Problem Formulation:

State Space : $X := \{1, 2, \cdots 25\}$

Control Space : $U := \{N, S, E, W\}$

Motion Model / Stage Cost : [redacted]

	$(x', L(x,u))$ x \ u	N	S	E	W
	1	(1, 1)	(6, 0)	(2, 0)	(1, 1)
A ☆	2	(22, -10)	(22, -10)	(22, -10)	(22, -10)
	3	(3, 1)	(8, 0)	(4, 0)	(2, 0)
B ☆	4	(14, -5)	(14, -5)	(14, -5)	(14, -5)
	5	(5, 1)	(10, 0)	(5, 1)	(4, 0)
	6	(1, 0)	(11, 0)	(7, 0)	(6, 1)
	7	(2, 0)	(12, 0)	(8, 0)	(6, 0)
	8	(3, 0)	(13, 0)	(9, 0)	(7, 0)
	9	(4, 0)	(14, 0)	(10, 0)	(8, 0)
	10	(5, 0)	(15, 0)	(10, 1)	(9, 0)
	11	(6, 0)	(16, 0)	(12, 0)	(11, 1)
	12	(7, 0)	(17, 0)	(13, 0)	(11, 0)
	13	(8, 0)	(18, 0)	(14, 0)	(12, 0)
B'	14	(9, 0)	(19, 0)	(15, 0)	(13, 0)
	15	(10, 0)	(20, 0)	(15, 1)	(14, 0)
	16	(11, 0)	(21, 0)	(17, 0)	(16, 1)
	17	(12, 0)	(22, 0)	(18, 0)	(16, 0)
	18	(13, 0)	(23, 0)	(19, 0)	(17, 0)
	19	(14, 0)	(24, 0)	(20, 0)	(18, 0)
	20	(15, 0)	(25, 0)	(20, 1)	(19, 0)
	21	(16, 0)	(21, 1)	(22, 0)	(21, 1)
A'	22	(17, 0)	(22, 1)	(23, 0)	(21, 0)
	23	(18, 0)	(23, 1)	(24, 0)	(22, 0)
	24	(19, 0)	(24, 1)	(25, 0)	(23, 0)
	25	(20, 0)	(25, 1)	(25, 1)	(24, 0)

Our goal is to find the optimal control and value function to minimize the expected long term cost: $V^*(x) = \min_{u \in U(x)} (\ell(x,u) + \gamma \sum_{x' \in X} P_f(x' | x, u) V^*(x'))$, $\forall x \in X$

Technical Approach:

The problem can be solved using:

(a) value iteration:

$$V_{k+1}(x) = \min_{u \in U(x)} [\ell(x,u) + \gamma \sum_{x' \in X} P(x'|x,u) V_k(x')], \quad \forall x \in X$$

iterate until convergence: $\max_x |V_{k+1}(x) - V_k(x)| < \text{threshold}$

(b) policy iteration:

loop until convergence {

① Policy Evaluation:

$$V^\pi(x) = \ell(x, \pi(x)) + \gamma \sum_{x' \in X} P_f(x'|x, \pi(x)) V^\pi(x'), \quad \forall x \in X$$

② Policy Improvement:

$$\pi'(x) = \arg\min_{u \in U(x)} [\ell(x,u) + \gamma \sum_{x' \in X} P_f(x'|x,u) V^\pi(x')], \quad \forall x \in X$$

(c) Q-Value iteration:

$$Q^*(x,u) = \ell(x,u) + \gamma E_{x' \sim P_f(\cdot|x,u)} [\min_{u' \in U(x')} Q^*(x', u')]$$

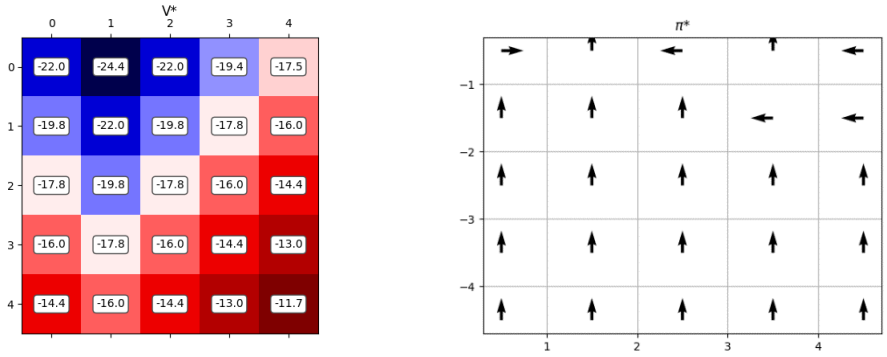
where $\pi^*(x) = \arg\min_{u \in U(x)} Q^*(x,u)$

- **Results**

(a)

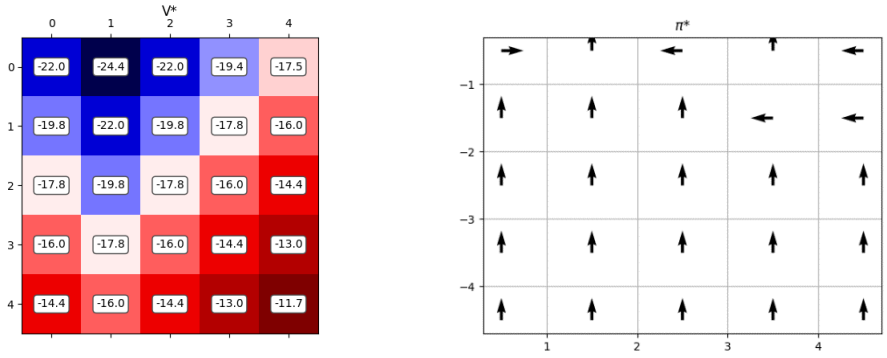
Regular Value Iteration

Number of iterations = 133



Gauss-Seidel Value Iteration

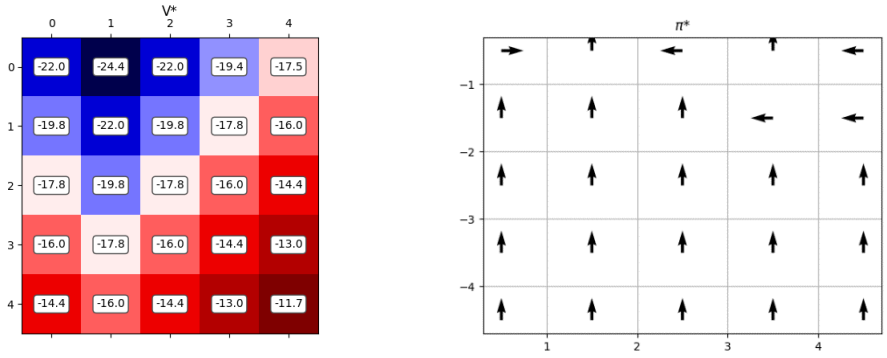
Number of iterations = 29



(b)

Policy Iteration

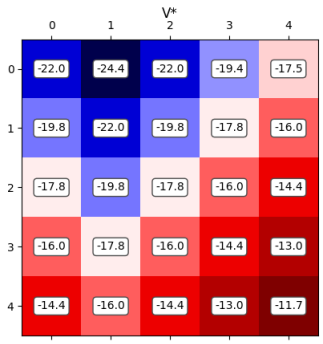
Number of iterations = 5

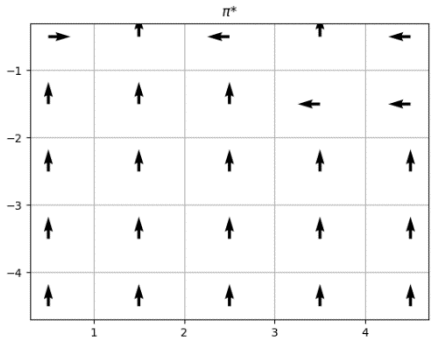


(c)

Q-Value Iteration

Number of iterations = 29

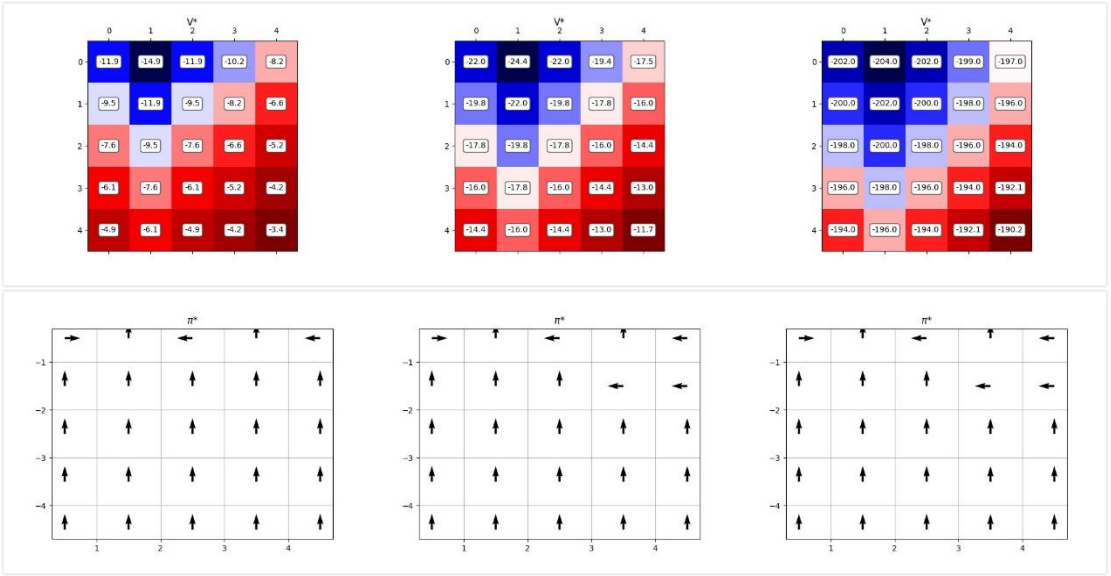




(d)

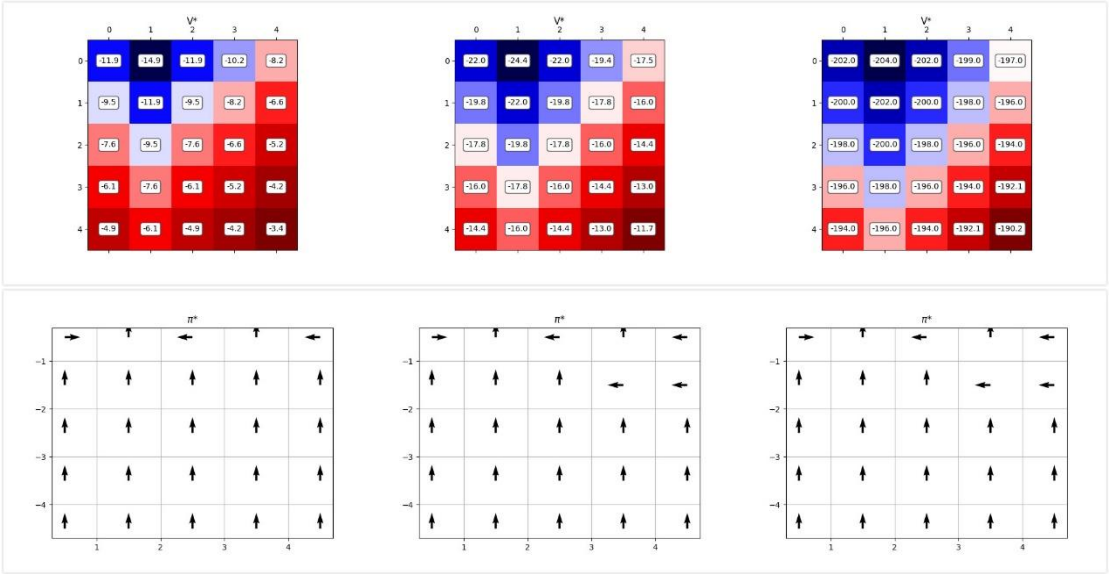
Gauss-Seidel Value Iteration

Gamma = 0.8 **Gamma = 0.9** **Gamma = 0.99**



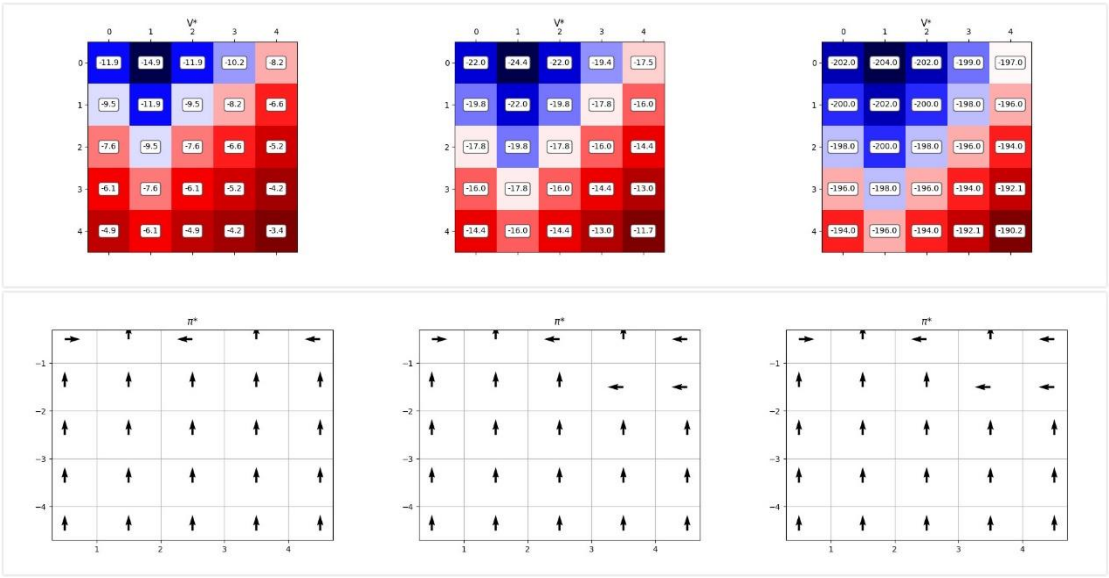
Policy Iteration

Gamma = 0.8 **Gamma = 0.9** **Gamma = 0.99**



Q-Value Iteration

Gamma = 0.8 **Gamma = 0.9** **Gamma = 0.99**



Analysis:

Notice when gamma = 0.8, the policy of states 9 and 10 is go north, when gamma gets bigger, the policy of states 9 and 10 become go west. This is a good example of how much we rely on the future cost. When we rely less on the future cost, i.e. gamma is smaller, the policy tends to favor the reward in the near future, so policy in state 9 and 10 will lead to state 4, which has a stage cost of -5. When we rely more on the long-term cost, i.e. when gamma is bigger, the policy tends to favor the long-term reward, so the policy in 9 and 10 will lead to state 2, which has a stage cost of -10 that is smaller than state 4, although state 2 is further than state 4 for state 9 and 10.

ECE276B HW3 Problem 3

- ## Introduction

Model-free prediction and control, also known as reinforcement learning, aims to solve the MDP problem with unknown motion model and cost function but access to examples of system transitions and incurred costs. Since it is usually the case that the exact motion model and cost function are hard to define, the reinforcement learning is of great importance in the field of robotics for obtaining the optimal control policy using the experimental data. In this project, we are trying to find the optimal control policy to help an underpowered car to climb a steep hill. Two typical on-policy and off-policy algorithms are implemented to solve this problem:

1. **On-Policy TD Policy Iteration Algorithm (SARSA)**
   The policy we use to generate the next episode is based on current Q function.
2. **Off-Policy TD Policy Iteration Algorithm Using Q-learning**
   Other policies are used to evaluate and update the Q function.

This report describes these two approaches and compare their performance on different hyperparameters.

- ## Problem Formulation

Given the following problem setting:

Car Position(P) — continuous in range [-1.2, 0.6]

Car Velocity(V) — continuous in range [-0.07, 0.07]

Control(U) — {0, 1, 2} where 0-push left, 1-no push, 2-push right

Target Position(T) — {0.5}

Formulate a Markov Decision Process:

State space: X := {[P, V]}, where P/V is the discretized position/velocity space

Control space: U := {0,1,2}

Motion model: unknown

Stage cost: unknown

Control policy: $\pi(x)$

Value function: $V^{\pi}(X)$

Q function: $Q^{\pi}(x,u) = l(x,u)+\gamma * E_{x' \sim pf(.|x,u)}[V^{\pi}(x')]$

Our goal is to construct an optimal value function and optimal control policy such that:

$Q^*(x,u) := l(x,u) + \gamma * E_{x' \sim pf(.|x,u)}[\min Q^*(x',u')]$

$\pi^*(x) = \text{argmin } Q^*(x,u)$

Since the stage cost and the motion model is unknown, we will use the experimental data obtained from the environment to estimate and update the corresponding Q value, and finally obtain our optimal policy.

- **Technical Approach**

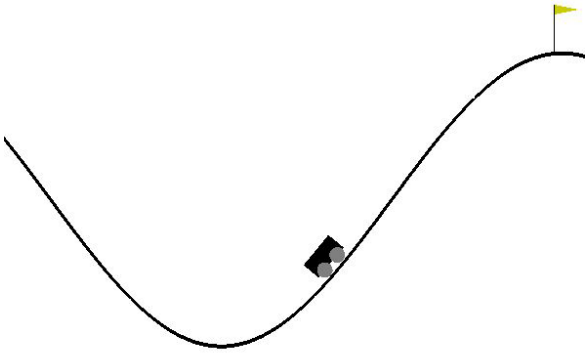


Fig. 1

Fig. 1 demonstrates problem setting, an underpowered car is trying to climb a steep hill to reach the target flag.

State Discretization

Since our car position and car velocity are continuous, we need to appropriately discretize these two space to form our state space X := {[P,V]}. We choose suitable discretize resolution for position and velocity respectively, and finally form our 2D state space. Theoretically, the smaller the resolution for these two spaces is, the better result we will have, but exhaustive state space also increase our computational complexity. Therefore, it is important to choose appropriate resolutions.

1. On-Policy TD Policy Iteration Algorithm (SARSA)

We generate current ε-greedy policy based on current Q value and feed this policy to the environment to get an episode, then update Q value using TD updates after every transition in the episode. Iterate until convergence on Q value. This way we can estimate the Q value using experimental data even without explicit motion model and stage cost.

Algorithm Detail

This is the logic flow of SARSA algorithm:

1: Init: Q(x,u) for all $x \in X$ and all $u \in U$

2: while Q does not converge do

3: $\pi \leftarrow$ ε-greedy policy derived from Q

4: $\varepsilon = \varepsilon * 0.999$

5: Generate episode $p := (x_{0:T}, u_{0:T-1})$ from π

6: for $(x,u,x',u') \in p$ do

7: $Q(x,u) \leftarrow Q(x,u) + \alpha*[l(x,u) + \gamma*Q(x',u') - Q(x,u)]$

- The ε-greedy policy is obtained by:

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

This is to maintain the "exploring" ability of the algorithm, under which every control has a non-zeros probability of being encountered.

- α is learning rate, a good choice is using the Robbins-Monro step sizes. Which suggests:

$$\sum_{k=1}^{\infty} \alpha_k = \infty \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

The second constrains ensure convergence of Q.

- γ is discount factor, indicating how much you rely on the future cost.

2. Off-Policy TD Policy Iteration Algorithm (Q-Learning)

Very similar to on-policy TD policy iteration, but now for state x', we consider all control input u' and pick the optimal Q(x',u') to update the Q.

Algorithm Detail

This is the logic flow of Q-Learning algorithm:

1: Init: Q(x,u) for all x ∈ X and all u ∈ U
2: while Q does not converge do
3: π ← ε-greedy policy derived from Q
4: Generate episode p := ($x_{0:T}$,$u_{0:T-1}$) from π
5: for (x,u,x') ∈ p do
6: Q(x,u) ← Q(x,u) + α*[l(x,u) + γ*$\min_{u'}$Q(x',u') – Q(x,u)]

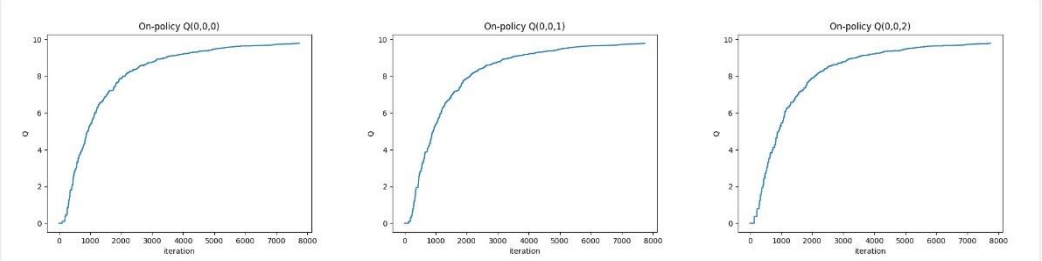
• Results

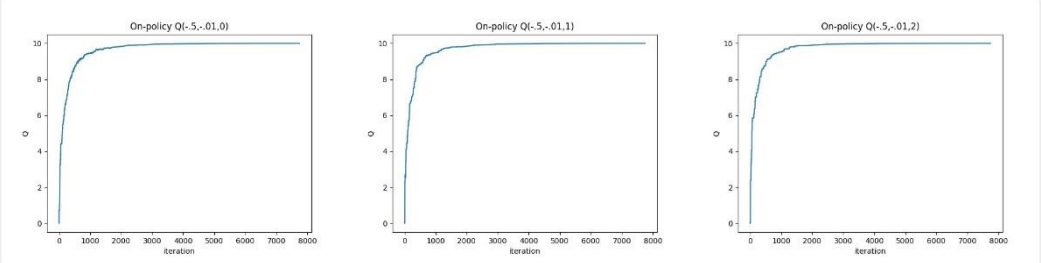
On-Policy TD Policy Iteration (SARSA)

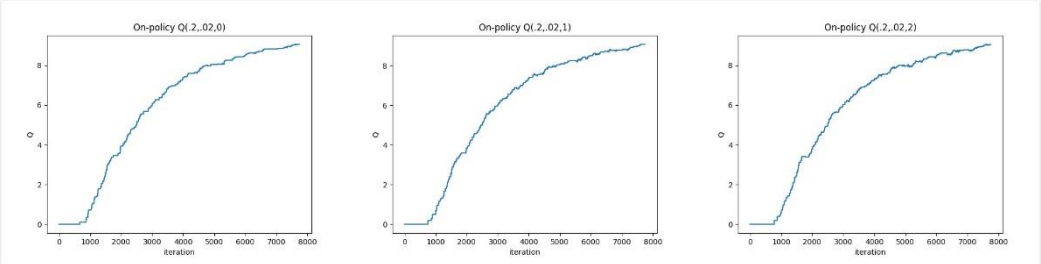
The following results are obtained with parameters:

Resolution	Gamma	Alpha	Epsilon	Threshold
[0.05,0.005]	0.9	0.1	0.3	0.0001

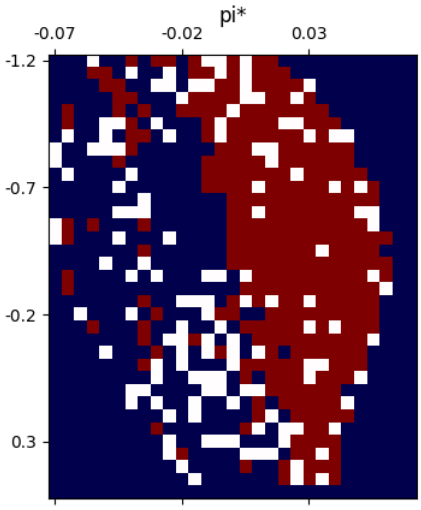
Number of iterations = 7746







Q(p,v,u) over episode



π*(p,v)

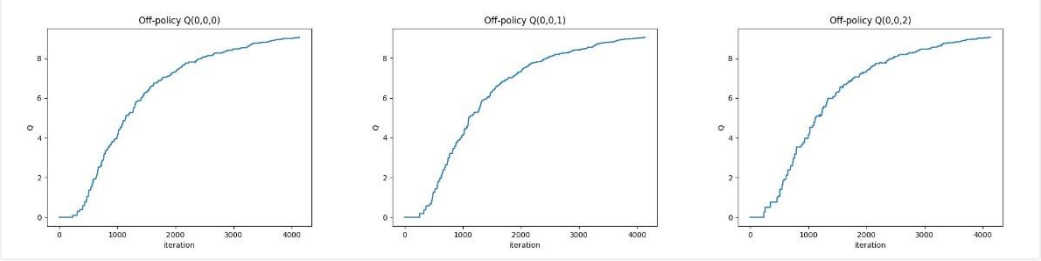
(blue-push left white-no push red-push right)

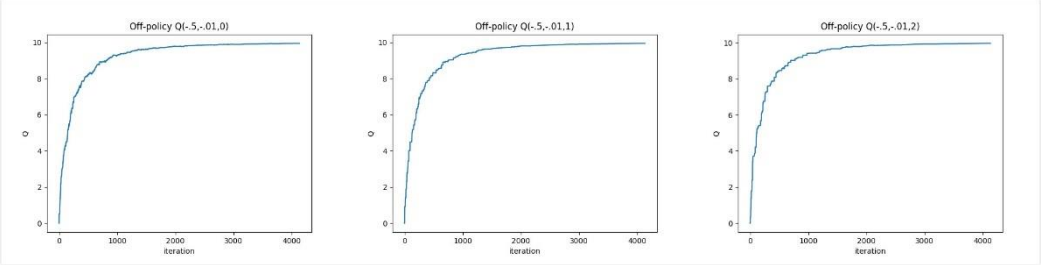
Off-Policy TD Policy Iteration (Q-Learning)

The following results are obtained with parameters:

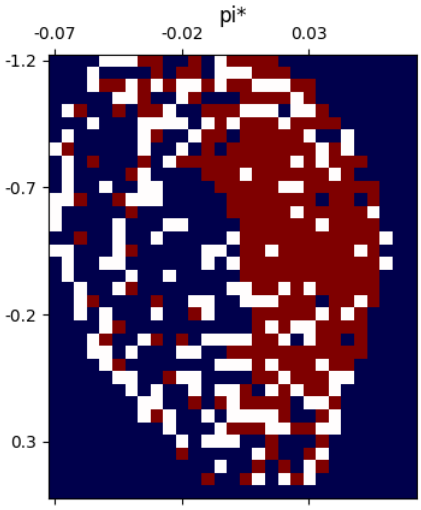
Resolution	Gamma	Alpha	Epsilon	Threshold
[0.05,0.005]	0.9	0.1	0.3	0.0001

Number of iterations = 4133





Q(p,v,u) over episode



π*(p,v)

(blue-push left white-no push red-push right)

Analysis:

For both on-policy and off-policy methods, the selected Q(p,v,u) all show a trend of convergence. The off-policy converge faster than on-policy.

On-policy VS Off-policy

The following results are obtained with parameters:

Resolution	Gamma	Alpha	Epsilon	Threshold
[0.05,0.005]	0.9	0.1	0.3	0.0001

Videos of on-policy and off-policy is included in the zip file.

Policy / Statistic	On-Policy	Off-Policy
I	10127	4260

D	2.859	4.090
S	S	S

I — number of iterations to converge

D — Total horizontal distance traveled under optimal policy

S — Whether target is reached within 200 time-step (S-success/F-fail)

Analysis:

Under good parameters, off-policy converges faster than on-policy, but the optimal policy obtained using on-policy iteration is better than off-policy iteration for that it has smaller travel distance to drive the car to the target.

Hyper-parameters comparison

In the following table:

I — number of iterations to converge

D — Total horizontal distance traveled under optimal policy

S — Whether target is reached within 200 time-step (S-success/F-fail)

Policy / Resolution	On-Policy			Off-Policy		
	I	D	S	I	D	S
[0.1,0.01]	376	4.345	F	693	4.542	F
[0.05,0.005]	1541	3.431	S	1651	4.237	S
[0.02,0.002]	3625	4.004	S	5188	3.909	F

gamma = 0.9 alpha = 0.5 epsilon = 0.3 threshold = 1e-4

Analysis:

Generally speaking, if the resolution is smaller, our policy will become better. If the resolution is too large, the policy we get may not be able to drive the car to the target since it is not descriptive enough, but smaller resolution also increases the search space, which increase the computational complexity dramatically. Under same resolution, the off-policy converge slower than on-policy, but the policy obtained using on-policy is more optimal.

Policy / Gamma	On-Policy			Off-Policy		
	I	D	S	I	D	S
0.5	151	3.733	F	153	3.473	F
0.8	635	5.090	F	815	4.416	F
0.95	9011	4.157	S	1092	4.748	S

resolution = [0.05,0.005] alpha = 0.5 epsilon = 0.3 threshold = 1e-4

Analysis:

The discount factor indicates how much you rely on the future cost, as gamma

increases, you trust more on the future cost. From the above statistic, we can conclude that if gamma is too small (gamma = 0.5/0.8), neither of the two policy can drive the car to the target within t=200 episode. This indicate that this problem value much on the future cost. Since to reach the peak on the right, we should drive the car to the left steep to make use of the potential energy, in other words, we may sacrifice the current cost for future cost.

Policy / Alpha	On-Policy			Off-Policy		
	I	D	S	I	D	S
0.1	9961	2.915	S	6679	2.334	S
0.5	1321	3.244	S	1180	4.016	S
0.8	913	3.376	F	653	3.231	F

resolution = [0.05,0.005] gamma = 0.9 epsilon = 0.3 threshold = 1e-4

Analysis:

As the learning rate alpha increase, both two policy converge faster, but it is not the case that learning rate should be as big as possible. Since as step size grow big, our algorithm may diverge. But if the learning rate is too small, our algorithm may be trapped in a local minimum and may take long time to converge. Therefore, it is important to choose a suitable learning rate to balance the trade-off between convergence, convergent rate and policy optimality.

Policy / Epsilon	On-Policy			Off-Policy		
	I	D	S	I	D	S
0.1	8604	3.353	S	3630	3.584	S
0.3	4926	4.253	S	4004	2.813	S
0.5	3130	3.916	S	1774	4.486	S

resolution = [0.05,0.005] gamma = 0.9 alpha = 0.2 threshold = 1e-4

Analysis:

The epsilon-greedy factor ensures the “exploring” ability of algorithm, i.e. the policy that is not optimal under current Q value still has a non-zero probability of being evaluated. If the epsilon is small, that means we focus on optimal policy, if the epsilon is big, that means we focus on “exploring”. From the above statistic, we can conclude that within a reasonable range, as epsilon increases, our algorithm converges faster. This suggests that it is good to maintain the “exploring” ability of the algorithm. However, the optimality of obtained optimal policy is not necessarily increases as epsilon increases.

ECE276B HW3 Problem 4

- ## Introduction

Infinite horizon problems are very common in the field of robotics, as it is usually the case that the planning horizon is not determined. Therefore, it is very important to find the optimal control policy based on this infinite horizon problem formulation. Such problems are usually solved by formulating an infinite horizon discounted problem and find the optimal value function based on bellman equation. In this project, we are trying to balance an inverted pendulum by formulating it as an infinite horizon discounted problem. Two typical value iteration and policy iteration algorithms are implemented to solve this problem:

1. **Value Iteration**

   We iterate on value function until convergence is reached.

2. **Policy Iteration**

   We iterate on policy until convergence is reached.

This report describes these two approaches and compare their performance on different hyperparameters.

- ## Problem Formulation

Given the following problem setting:

Pendulum Position(P) — continuous in range [0(up), 2π] and it is wrap-around

Angular Velocity(V) — continuous in range [-Vmax, Vmax]

Control(U) — continuous in range [-Umax, Umax]

Target Position(T) — {0}

Formulate a Markov Decision Process:

State space: X := {[P, V]}, where P/V is the discretized position/velocity space

Control space: U := {u}, where u is the discretized control space

Motion model:

$$d\mathbf{x} = f(\mathbf{x}, u)\, dt + \sigma\, d\omega \qquad f(\mathbf{x}, u) := \begin{bmatrix} x_2 \\ a \sin x_1 - b\, x_2 + u \end{bmatrix}$$

$$Pf(x'|x,u) \sim N(x+f(x,u)dt, \sigma\sigma^{T}dt)$$

Stage cost:

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

Control policy: $\pi(x)$

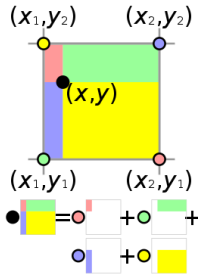
Value function: $V^{\pi}(X)$

Our goal is to use value/policy iteration algorithm to solve the bellman equation and find the optimal value/control policy such that:

$$V^*(x) = \min_{u \in \mathcal{U}(x)} \Big(\ell(x,u) + \gamma \sum_{x' \in \mathcal{X}} p_f(x' \mid x,u) V^*(x') \Big), \quad \forall x \in \mathcal{X}$$

$$\pi^*(x) = \text{argmin}_u \, V^*(x,u)$$

After obtaining our optimal policy, we will use interpolation to extend the discrete control policies to continuous space and test on inverted pendulum to see if it can actually be inverted. The way we use to interpolate control policy is bilinear interpolation as shown in the graph:



So that the value at (x,y) can be computed using:

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}$$

- ## Technical Approach

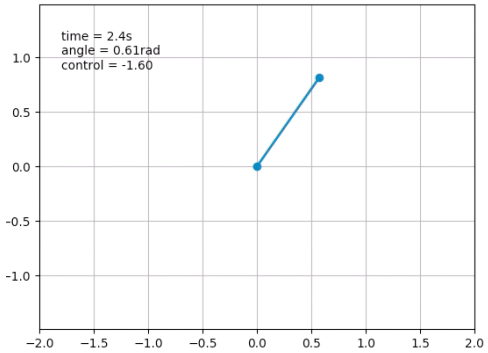


Fig. 1

Fig. 1 demonstrates problem setting, a pendulum can rotate around one of its end point. Our target is to push and balance the pendulum so that it is inverted and point up.

State Discretization

Since our pendulum position, pendulum velocity and control input are in continuous space, we need to appropriately discretize these three space to form our finites state space MDP problem:

n1 – angle grid, so the resolution is 2*π/n1

n2 – angular velocity grid, so the resolution is 2*Vmax/n2

nu – control grid, so the resolution is 2*Umax/nu

The state space is defined as X := {(P,V)}, where P/V is the discretized angle/velocity.

Notice that when designing these hyper parameters, it should roughly satisfy the constraints that:

$$\frac{2\text{Vmax}}{\text{n2}}dt \approx \frac{2\pi}{\text{n1}}$$

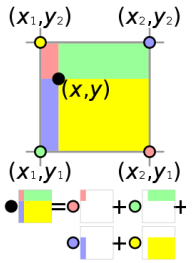
$$\frac{2\text{Umax}}{\text{nu*m}}dt \approx \frac{2\text{Vmax}}{\text{n2}}$$

Where m is the mass of the pendulum.

In this way, the transition between states in MDP is distinct, and the capability of our grid is suitable to solve this problem.

Policy Interpolation

Since the final optimal policy we get is defined on discrete state space, we need to interpolate it so that the control will be more precise at any physical state. Suppose that the physical state we get is (x,y), and its nearby four grid states is (x_1,y_2), (x_2,y_2), (x_1,y_1), (x_2,y_1) as shown in the following graph:



Then the V value at these four grid states are defined as $f(x_1,y_2)$, $f(x_2,y_2)$, $f(x_1,y_1)$, $f(x_2,y_1)$. If we take (x_1,y_1) to the origin, i.e. only consider the distance between (x,y) to these four grid states, we can compute the f(x,y) using bilinear interpolation:

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}$$

Now we have extended our policy to the continuous states space.

Gaussian Motion Model

As stated in the problem, our motion model is in the form:

$$Pf(x'|x,u) \sim N(x+f(x,u)dt, \sigma\sigma^T dt)$$

Where

$$d\mathbf{x} = f(\mathbf{x},u)\,dt + \sigma\,d\omega \qquad f(\mathbf{x},u) := \begin{bmatrix} x_2 \\ a\sin x_1 - b\,x_2 + u \end{bmatrix}$$

For the efficiency of the algorithm, since the grids that are far from the mean grid will have very little probability, we will only consider the mean grid and the adjacent 8 grids to compute the discounted expected future cost.

1. Value Iteration

The Gauss-Seidel value iteration algorithm is implemented, where for each state, we analyze each control input and pick the one that will incur minimum stage plus future cost, and update the value of this state in place with this total cost. The core equation of value iteration algorithm is:

$$V(x) \leftarrow \min_{u \in \mathcal{U}(x)} \left(\ell(x,u) + \gamma \sum_{x' \in \mathcal{X}} p_f(x' \mid x,u) V(x') \right), \qquad \forall x \in \mathcal{X}$$

Algorithm Detail

This is the logic flow of value iteration algorithm:

1: Init: V(x) for all x ∈ X, π(x) for all x ∈ X

2: while V does not converge do

3: for every u ∈ U

4: evaluate $V(x,u) = l(x,u) + \gamma \sum_{x' \in X} Pf(x'|x,u)V(x')$

5: $V(x) \leftarrow \min_{u \in U} V(x,u)$

6: $\pi(x) \leftarrow \operatorname{argmin}_{u \in U} V(x,u)$

- The Gauss-Seidel value iteration often leads to faster convergence and requires less memory than regular value iteration algorithm.
- γ is discount factor, indicating how much you rely on the future cost.

2. Policy Iteration

Policy iteration can be divided into two steps:

a. Policy Evaluation

Given current policy π, compute and update V(x) under policy π until convergence

$$V^{\pi}(x) = \ell(x,\pi(x)) + \gamma \sum_{x' \in \mathcal{X}} p_f(x' \mid x,\pi(x)) V^{\pi}(x'), \qquad \forall x \in \mathcal{X}$$

b. Policy Improvement

Using the converged V(x), obtain new stationary policy π'

$$\pi'(x) = \arg\min_{u \in \mathcal{U}(x)} \left[\ell(x,u) + \gamma \sum_{x' \in \mathcal{X}} p_f(x' \mid x,u) V^{\pi}(x') \right], \qquad \forall x \in \mathcal{X}$$

Iterate the above two steps until convergence.

Algorithm Detail

This is the logic flow of policy iteration algorithm:

1: Init: V(x) for all x ∈ X, π(x) for all x ∈ X

2: while V does not converge do

3: while V does not converge do

4: for all x ∈ X

5: $V^{\pi}(x) = l(x,\pi(x)) + \gamma \sum_{x' \in X} Pf(x'|x,\pi(x))V(x')$

7: $\pi'(x) \leftarrow \underset{u \in U(x)}{\operatorname{argmin}}[l(x,u) + \gamma \sum_{x' \in X} Pf(x'|x,u) V^{\pi}(x')]$

- The nature of value iteration and policy iteration is equivalent.
- Policy iteration can be seen as running the value update step of value iteration an infinite number of times.
- The value update step of value iteration is an iterative solution to the linear system of equations in policy evaluation.

• Results

The following results are obtained with predefined good parameters:

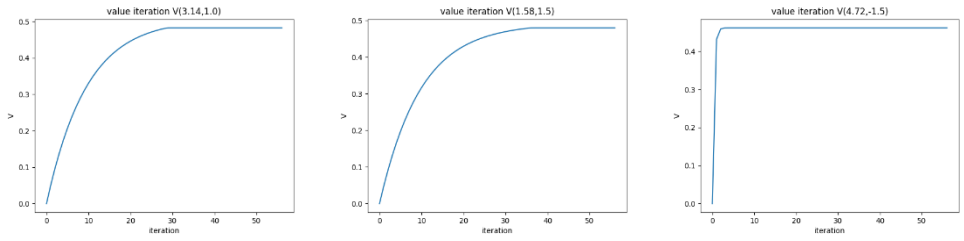
a	b	k	r	gamma	dt
0.98	0.1	3	0.01	0.9	0.05

n1	n2	nu	Vmax	Umax	sigma
180	50	50	5	5	(0.1,0.04)

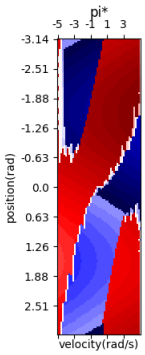
dw	threshold
(0.1,0.1)	1e-5

Value Iteration

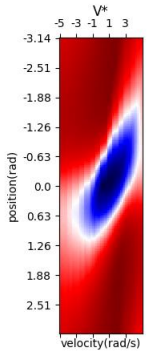
Number of iterations = 56



V(P,V) over episode



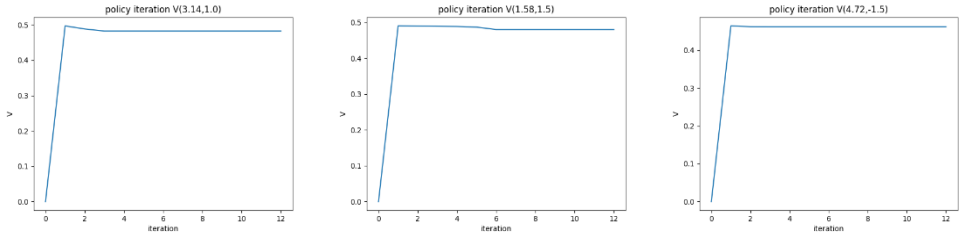
π*(p,v)



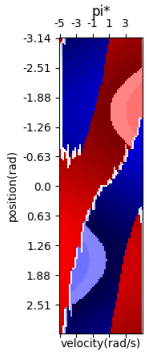
V*

Policy Iteration

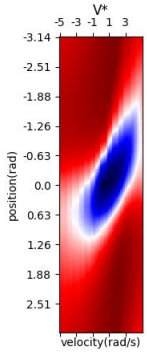
Number of iterations = 12



V(P,V) over episode



π*(p,v)



V*

Analysis:

From the above result, we can conclude that the selected V value of certain states from both value iteration and policy iteration both demonstrates a trend of convergence over the training episode. The policy iteration requires less iteration to converge than value iteration does. But the result of two algorithm is roughly the same. The optimal value function is hyperbola-like in the state space, which is reasonable since the (0,0) position (middle of the image) should have the least cost, and the cost should be symmetric around the center. Similar for the optimal control policy, which would be roughly described as push right when on right half plane and push left when on the left half plane.

Hyper-parameters comparison

In the following table:

I — number of iterations to converge

T — total training time (s)

S — The ability to invert the pendulum (S-Success/F-Fail)

Except the value that is being evaluated, all other values are using the predefined good values above.

Algorithm / a = m*g*L	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
0.098	47	475	S	9	443	S
0.98	42	423	S	10	487	S
9.8	40	394	F	10	495	F

Analysis:

a summarizes the effect of gravity, mass and length of the pendulum. As a grow larger, if the Umax is unchanged, it would be harder to push the pendulum up. Therefore, we should increase our Umax to ensure that Umax > a so that our physical model make sense, or we might not be able to push the pendulum up.

Algorithm / b	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
0.01	43	442	S	11	496	S
0.1	43	439	S	10	441	S
1	45	459	S	12	520	S

Analysis:

b represents the damping and friction factor, unlike a, it always add resistance in the opposite direction of current motion, therefore, the actual control will be larger to

counter this resistance. Ideally b should be as small as possible so that it would be easier to push the pendulum up.

Algorithm / sigma	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
[0.01,0.01]	47	492	S	13	566	S
[0.1,0.1]	43	445	S	11	503	S
[1,1]	32	384	F	8	410	F

Analysis:

sigma controls the distribution of gaussian motion model. If sigma is larger, our motion model is more undetermined, i.e. the probability distribution is more wide spread. If sigma is smaller, our motion model is more determined, i.e. the weights concentrates on the mean. It seems that smaller sigma will lead to faster convergence in both algorithms. However, large sigma also incurs large noise, which may result in the failure of keeping pendulum inverted.

Algorithm / k	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
1	42	437	S	10	442	S
5	43	448	S	10	435	S
10	42	433	F	11	498	F

Analysis:

k value determines the shape of the stage cost function, if k is too big, the cost function is more “convex up”. If we set k too large, the pendulum will finally stay near the vertical point, this is because the position cost is no comparison to the control cost near the vertical position, so the system would rather stay near the equilibrium point instead of reaching it.

Algorithm / r	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
0.01	42	461	S	10	430	S
0.1	42	457	S	10	438	S
1	45	520	F	10	432	F

Analysis:

r scales the control cost, it works with k to determine the final equilibrium point. If r is too large, the control cost will dominate the position cost, resulting in too small control selected and the vertical equilibrium point is never reached.

Algorithm / gamma	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
0.3	58	694	S	17	816	S
0.6	50	646	S	15	735	S
0.9	43	453	S	11	512	S

Analysis:

Gamma is the discount factor, it controls how much you rely on the future cost. In this experiment, it seems that it is better to choose larger gamma. i.e. believe more on the future cost. And both algorithms converge faster.

Algorithm / dt	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
0.01	40	453	S	10	438	S
0.05	42	470	S	10	433	S
0.1	42	474	S	10	433	S

Algorithm / n1	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
50	29	261	F	6	228	F
100	35	379	S	9	366	S
180	43	488	S	10	440	S

Algorithm / n2	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
10	32	307	F	8	360	F
30	36	345	S	10	429	S
50	43	449	S	10	440	S

Algorithm / nu	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
10	32	315	F	8	373	F
30	35	377	S	10	439	S
50	43	471	S	10	440	S

Vmax \ Algorithm	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
1	42	424	S	10	452	S
5	43	445	S	10	448	S
10	42	419	F	11	487	F

Umax \ Algorithm	Value Iteration			Policy Iteration		
	I	T	S	I	T	S
1	40	453	F	10	441	F
5	42	465	S	10	436	S
10	42	474	S	10	434	S

Analysis:

These six parameters work together, they should satisfy the following condition:

$$\frac{2\mathrm{Vmax}}{\mathrm{n2}}dt \approx \frac{2\pi}{\mathrm{n1}}$$

$$\frac{2\mathrm{Umax}}{\mathrm{nu*m}}dt \approx \frac{2\mathrm{Vmax}}{\mathrm{n2}}$$

If the difference between left and right part of the equation is dramatic, it means our discretization is not descriptive enough, i.e. a transition in one states barely incur any transition in other states, making it very hard for our value function to update. Individually, if dt is too large, the transition is significant, resulting in shaking in the equilibrium point, if dt too small, it is hard for transition between states happen. If n1/n2/nu is too small, the grid is too sparse and is not descriptive enough, if too large, the computational time is increased dramatically. If Umax is too small, it is hard to push the pendulum up. In conclusion, we should choose these hyperparameters wisely to solve the problem.