```
'''
   CS 5180    Fall 2022
   Exercise 3: Dynamic Programming
   Hongyan Yang
'''
```

1. RL 2e 3.25-3.29, Fun with Bellman

(a) Equation for V^* in terms of q^*:

$$V^*(s) = \max_{a \in A(s)} q^*(s,a)$$

(b) Equation for q^* in terms of V^* and the four-argument p

$$q^*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma V^*(s')\right]$$

(c) Equation for π^* in terms of q^*

$$\pi^*(a|s) = \operatorname{argmax}_a q^*(s,a)$$

(d) Equation for π^* in terms of V^* and the four-argument p

$$\pi^*(a|s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V^*(s')\right]$$

(e) Rewrite four Bellman equations

① $V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V_\pi(s')\right]$, for all $s \in S$

② $V^*(s) = \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^*(s')\right]$

③ $q_\pi(s,a) = \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')\right]$

④ $q^*(s,a) = \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma \max_{a'} q^*(s',a')\right]$

2. RL2e 4.5, 4.10: Policy iteration for action values:

(a) 1. Initialization

$Q(s,a) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$;

2. Policy Evaluation

Loop: $\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

Loop for each $a \in \mathcal{A}(s)$:

$q \leftarrow Q(s,a)$

$Q(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma Q(s', \pi(s'))]$

$\Delta \leftarrow \max(\Delta, |q - Q(s,a)|)$

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable $\leftarrow$ true

For each $s \in \mathcal{S}$:

old-action $\leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_a Q(s,a)$

If old-action $\neq \pi(s)$, then policy-stable $\leftarrow$ false

If policy-stable, then stop and return $Q \approx q^*$ and $\pi \approx \pi^*$; else go to 2

(b) $q_{k+1}(s,a) = \max_{a'} \sum_{s',r} p(s',r|s,a) \cdot [r + \gamma q_k(s',a')]$

3. Policy iteration by hand

(a) Just by looking at the transition and reward structure, optimal policy in state x is action C, optimal policy in state y is b. Because all states have negative reward and reward y = 2x(reward x). This policy takes least penalty along the way.

(b) Apply action C as the initial policy:

k	V_k for policy: State X, Y	policy: State X, Y
k=0	0, 0	C, C
k=1	-10, -20	c, b

① At this policy evaluation step, according to the given policy:

$$\begin{cases} V_\pi(x) = 0.1\times[-1+0] + 0.9\times[-1+V_\pi(x)] & ① \\ V_\pi(y) = 0.1\times[-2+0] + 0.9\times[-2+V_\pi(y)] & ② \end{cases}$$

$$\Rightarrow \begin{cases} 0.1 V_\pi(x) = -1 \\ 0.1 V_\pi(y) = -2 \end{cases} \Rightarrow \begin{cases} V_\pi(x) = -10 \\ V_\pi(y) = -20 \end{cases}$$

② At policy improvement step,

$$\begin{cases} Q(x,b) = -1 + [0.8\times(-20) + 0.2\times(-10)] = -19 \\ Q(x,C) = -1 + [0.1\times 0 + 0.9\times(-10)] = -10 \end{cases} \Rightarrow \pi^*(x) = C$$

for k=1

$$\begin{cases} Q(y,b) = -2 + [0.8\times(-10) + 0.2\times(-20)] = -14 \\ Q(y,C) = -2 + [0.1\times 0 + 0.9\times(-20)] = -20 \end{cases} \Rightarrow \pi^*(y) = b$$

①'

Continue the policy evaluation step with $\begin{cases} \pi(x) = c \\ \pi(y) = b \end{cases}$

$$\begin{cases} V_\pi(x) = -1 + 0.1 \times 0 + 0.9 \times V_\pi(x) & ① \\ V_\pi(y) = -2 + 0.8 \times V_\pi(x) + 0.2 \times V_\pi(y) & ② \end{cases}$$

$$\Rightarrow \begin{cases} V_\pi(x) = -10 \\ V_\pi(y) = -12.5 \end{cases}$$

②' Continue the policy improvement step:

$$\begin{cases} Q(x,b) = -1 + 0.8 \times (-12.5) + 0.2 \times (-10) = -13 \\ Q(x,c) = -1 + 0.1 \times 0 + 0.9 \times (-10) = -10 \end{cases} \Rightarrow \pi^*(x) = c$$

$$\begin{cases} Q(y,b) = -2 + 0.8 \times (-10) + 0.2 \times (-12.5) = -12.5 \\ Q(y,c) = -2 + 0.1 \times 0 + 0.9 \times (-12.5) = -13.25 \end{cases} \Rightarrow \pi^*(y) = b$$

$\because$ old policy = new policy, the policy converged.

policy converged at $\begin{cases} \pi^*(x) = c \\ \pi^*(y) = b \end{cases}$, and $\begin{cases} V_\pi(x) = -10 \\ V_\pi(y) = -12.5 \end{cases}$

(c) When action b is applied:

	V_k for policy	policy
	State x, y	x, y
k=0	0, 0	b, b

① At policy evaluation step:

$$\begin{cases} V_\pi(x) = -1 + 0.8 V_\pi(y) + 0.2 V_\pi(x) & ① \\ V_\pi(y) = -2 + 0.8 V_\pi(x) + 0.2 V_\pi(y) & ② \end{cases}$$

$$\Downarrow$$

$$\begin{cases} 0.8[V_\pi(x) - V_\pi(y)] = -1 & ③ \\ 0.8[V_\pi(y) - V_\pi(x)] = -2 & ④ \end{cases} \xRightarrow{③+④} 0 = -3$$

$\therefore$ there's no solution for $V_\pi(x)$ and $V_\pi(y)$ if initial policy is action b.

2'

After adding the discounting factor γ, we have

$$\begin{cases} V_\pi(x) = -1 + 0.8\gamma V_\pi(y) + 0.2\gamma V_\pi(x) & ① \\ V_\pi(y) = -2 + 0.8\gamma V_\pi(x) + 0.2\gamma V_\pi(y) & ② \end{cases}$$

$$\Downarrow ①+②$$

$$V_\pi(x) + V_\pi(y) = -3 + \gamma\left[V_\pi(x) + V_\pi(y)\right] \Rightarrow V_\pi(x) + V_\pi(y) = \frac{-3}{1-\gamma} \quad ③$$

①-②

$$\Longrightarrow V_\pi(x) - V_\pi(y) = 1 - 0.8\gamma\left[V_\pi(x) - V_\pi(y)\right] + 0.2\gamma\left[V_\pi(x) - V_\pi(y)\right] \Rightarrow V_\pi(x) - V_\pi(y) = \frac{1}{1+0.6\gamma} \quad ④$$

$\frac{③+④}{2}$, we have $V_\pi(x) = \frac{1}{2}\left(\frac{3}{\gamma-1} + \frac{1}{1+0.6\gamma}\right)$

$\frac{③-④}{2}$, we have $V_\pi(y) = \frac{1}{2}\left(\frac{3}{\gamma-1} - \frac{1}{1+0.6\gamma}\right)$

$\therefore$ The discounting factor helps to solve the value for $V_\pi(x)$ and $V_\pi(y)$.
In this paticular MDP, optimal policy does not depend on the discount factor because of its transition and reward structure.

4. 2 points. Implementing dynamic programming algorithms.

Plot:

(a): Implement value iteration.

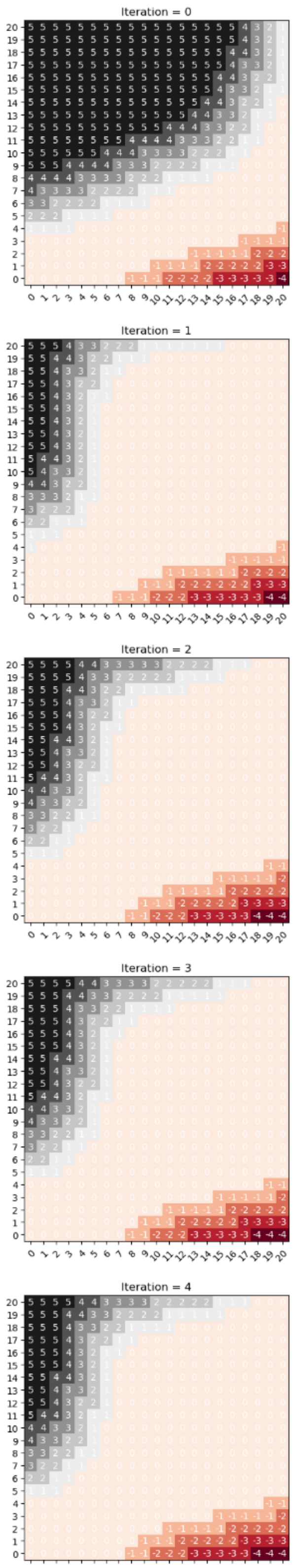
```
==============================
==  Optimal State Value     ==
==============================
[[22.  24.4 22.  19.4 17.5]
 [19.8 22.  19.8 17.8 16. ]
 [17.8 19.8 17.8 16.  14.4]
 [16.  17.8 16.  14.4 13. ]
 [14.4 16.  14.4 13.  11.7]]
==============================
==============================
==     Optimal Policy       ==
==============================
[0, 0] = ['east']
[0, 1] = ['north', 'south', 'west', 'east']
[0, 2] = ['west']
[0, 3] = ['north', 'south', 'west', 'east']
[0, 4] = ['west']
------------------------------
[1, 0] = ['north', 'east']
[1, 1] = ['north']
[1, 2] = ['north', 'west']
[1, 3] = ['west']
[1, 4] = ['west']
------------------------------
[2, 0] = ['north', 'east']
[2, 1] = ['north']
[2, 2] = ['north', 'west']
[2, 3] = ['north', 'west']
[2, 4] = ['north', 'west']
------------------------------
[3, 0] = ['north', 'east']
[3, 1] = ['north']
[3, 2] = ['north', 'west']
[3, 3] = ['north', 'west']
[3, 4] = ['north', 'west']
------------------------------
[4, 0] = ['north', 'east']
[4, 1] = ['north']
[4, 2] = ['north', 'west']
[4, 3] = ['north', 'west']
[4, 4] = ['north', 'west']
------------------------------
```

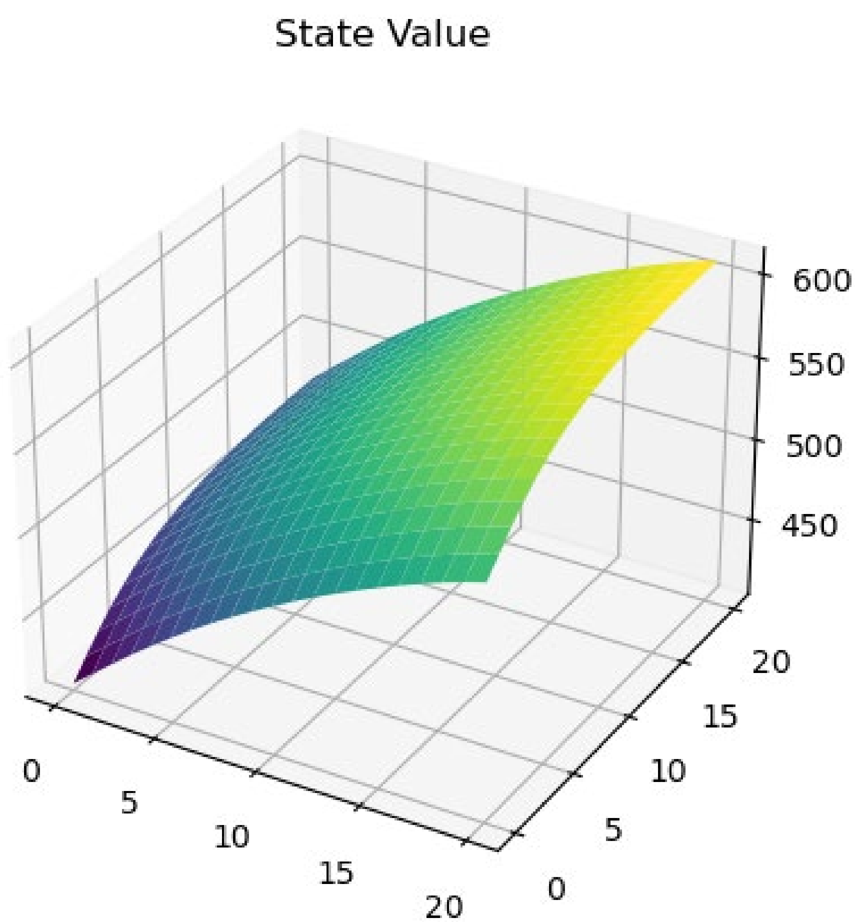
(b): Implement policy iteration.

```
=============================
==  Optimal State Value    ==
=============================
[[22.  24.4 22.  19.4 17.5]
 [19.8 22.  19.8 17.8 16. ]
 [17.8 19.8 17.8 16.  14.4]
 [16.  17.8 16.  14.4 13. ]
 [14.4 16.  14.4 13.  11.7]]
=============================
=============================
==     Optimal Policy      ==
=============================
[0, 0] = ['east']
[0, 1] = ['north', 'south', 'west', 'east']
[0, 2] = ['west']
[0, 3] = ['north', 'south', 'west', 'east']
[0, 4] = ['west']
------------------------------
[1, 0] = ['north', 'east']
[1, 1] = ['north']
[1, 2] = ['north', 'west']
[1, 3] = ['west']
[1, 4] = ['west']
------------------------------
[2, 0] = ['north', 'east']
[2, 1] = ['north']
[2, 2] = ['north', 'west']
[2, 3] = ['north', 'west']
[2, 4] = ['north', 'west']
------------------------------
[3, 0] = ['north', 'east']
[3, 1] = ['north']
[3, 2] = ['north', 'west']
[3, 3] = ['north', 'west']
[3, 4] = ['north', 'west']
------------------------------
[4, 0] = ['north', 'east']
[4, 1] = ['north']
[4, 2] = ['north', 'west']
[4, 3] = ['north', 'west']
[4, 4] = ['north', 'west']
------------------------------
```

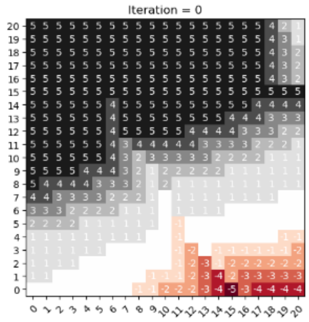
5. 3 points.[5180] (RL2e 4.7) Jack's car rental problem.

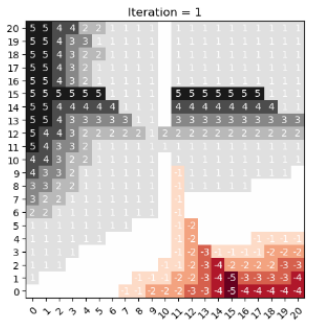
(a): Implement the policy iteration algorithm and generate the Figure 4.2

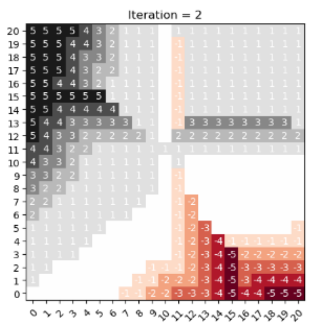


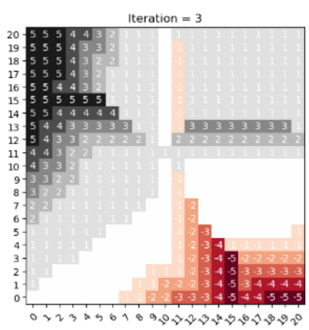


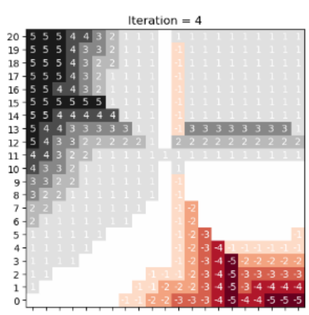
(b): Apply the implemented policy iteration on the modified Jack's car rental problem

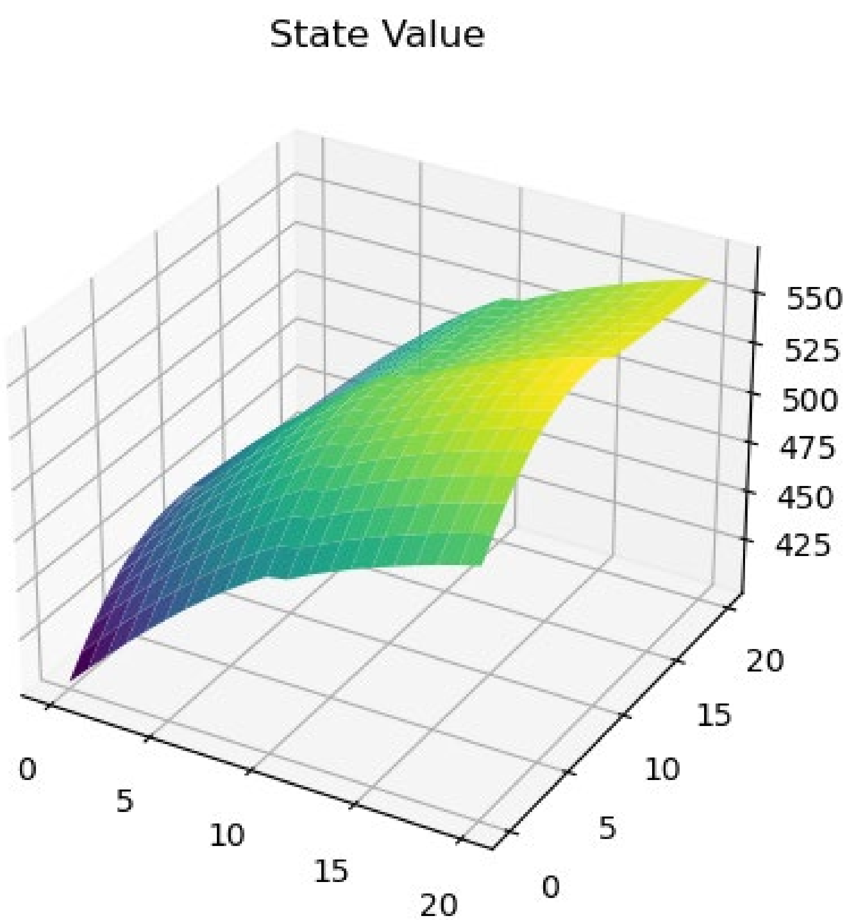












Written:

Describe how you will change the reward function (i.e. compute reward modified function in

the JackCarRental class) to reflect the following changes.

Based on the information given,

1. I will adjust the moving fee calculation as follows: if there's more than 0 cars need to be moved from loc1 to loc2, moving fee = 2 * (number of cars to move - 1).
2. I will add the additional parking fee if # of cars after moving > 10. Parking fee = 4 * number of locations where # of cars after moving > 10.

Written:

How does your final policy differ from Q5(a)? Explain why the differences make sense.

The final policy differs from Q5(a) in the following ways:

1. More cars are moved from loc1 to loc2 because there's one car free to move from loc1 to loc2. And according to the assumption "3 and 4 for rental requests at the first and second locations and 3 and 2 for returns", number of cars is biased to reduce at loc2. So, it's biased to move more cars from loc1 to loc2 when loc2 is relatively short of cars.
2. Cars are moved to another location to avoid pay high excess parking fee and cars are more biased to move to loc2 for the same reason stated above.
3. When all parking lots have more than 10 cars. In other words, both will pay excessive parking fee. One free car is arranged to move from loc1 to loc2 because loc2 is tend to be short of cars in the long run.