RL Homework 4: EX 3

Question 1:

a. $v_*(s) = max_a(q_*(s,a))$

b. $q_*(s,a) = \sum_{s',a} p(s',a|s,a)[r + \gamma v_*(s')]$

c. $\pi_*(s) = argmax_a(q_*(s,a))$

d. $\pi_*(s) = argmax_a(\sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')])$

e.

- $v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a) + \gamma v_*(s')]$
- $v_*(s) = max_a(\sum_{s'} p(s'|s,a)[r(s,a) + \gamma v_*(s')])$
- $q_\pi(s,a) = \sum_{s'} p(s'|s,a)[r(s,a) + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')]$
- $q_*(s,a) = \sum_{s'} p(s'|s,a)[r(s,a) + \gamma max_a q_*(s',a')]$

Question 2:

a.

Initialize:

$Q(s,a) = R\ arbitrarily\ \forall\ s \epsilon S\ and\ a \epsilon A$

$\pi(s) = A(s)\ arbitrarily\ \forall\ s \epsilon S$

Policy Evaluation:

Loop:

$\Delta = 0$

For each $s \epsilon S\ and\ a \epsilon A$:

$q = Q(s,a)$

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

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

Until $\Delta < \theta$ (Theta is a small positive number)

Policy Improvement:

policy-stable = true

For each $s \epsilon S$

$a = \pi(s)$

$\pi(s) = argmax_a Q(s,a)$

If $a \neq \pi(s)$ policy-stable = false

If policy-stable return $Q \; and \; \pi$; else go to 2

b.

$q_{k+1}(s,a) = E[R_{t+1} + \gamma max_{a'} q_k(s',a')]$

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

Question 3:

a. Qualitatively it seems that the optimal policy would be to always choose c in x and b in y as the penalty for being in y is twice that of being in x and the goal as all rewards are negative is to hit the terminal state as soon as possible.

b. Policy Iteration

   Policy Evaluation for given π: $V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$

   $V(z) = 0$, as it is a terminal state with no reward

   $V(x) = .9(-1 + .9(V(x)) + .1(-1 + .9(V(z))$

   $V(x) = .9(-1 + .9(V(x)) + -.1$

   $V(x) = -.9 + .81V(x) - .1$

   $.19V(x) = -1$

$V(x) = -5.26$

$V(y) = .9(-2 + .9(V(y)) + .1(-2 + .9(V(z))$

$V(y) = .9(-2 + .9(V(y)) + -.2$

$V(y) = -1.8 + .81(V(y)) - .2$

$.19V(y) = -2$

$V(y) = -10.53$

Policy Improvement

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

$\pi'(x) = max(Q(x, c), Q(x, b))$

$Q(x, c) = -5.26$ (from last question)

$Q(x, b) = .8(-1 + .9(-10.53)) + .2(-1 + .9(-5.26)) = -9.52$

$Q(x, c) < Q(x, b)$ so c is optimal action for state x

$\pi'(y) = max(Q(y, c), Q(y, b))$

$Q(y, c) = -10.53$ (from last question)

$Q(y, b) = .8(-2 + .9(-5.26)) + .2(-2 + .9(-10.53)) = -7.68$

$Q(y, b) < Q(y, c)$ so b is the optimal action for state y, this results in a change in policy so we go back to policy evaluation with our new policy.

$V(x) = -5.26$ (same as last time)

$V(y) = .8(-2 + .9(V(x)) + .2(-2 + .9(V(y))$

$V(y) = -5.3872 + .18V(y) - .4$

$.82V(y) = -5.7872$

$V(y) = -7.01$

Now Policy Improvement 1 more time

$\pi'(x) = max(Q(x, c), Q(x, b)) = Q(x, c)$

$\pi'(y) = max(Q(y, c), Q(y, b)) = Q(y, b)$

Our policy is stable through this phase so the optimal policy is take action c in x and take action b in y.

c. If we were to do policy iteration where the initial policy is action b in both states, it would still end up converging to the same optimal policy. The discount factor helps to weigh immediate vs future rewards, it also speeds up convergence and gives us more representative state value calculations. In this specific case the optimal policy doesn't depend on the discount factor.

Question 4:

a. Plot:

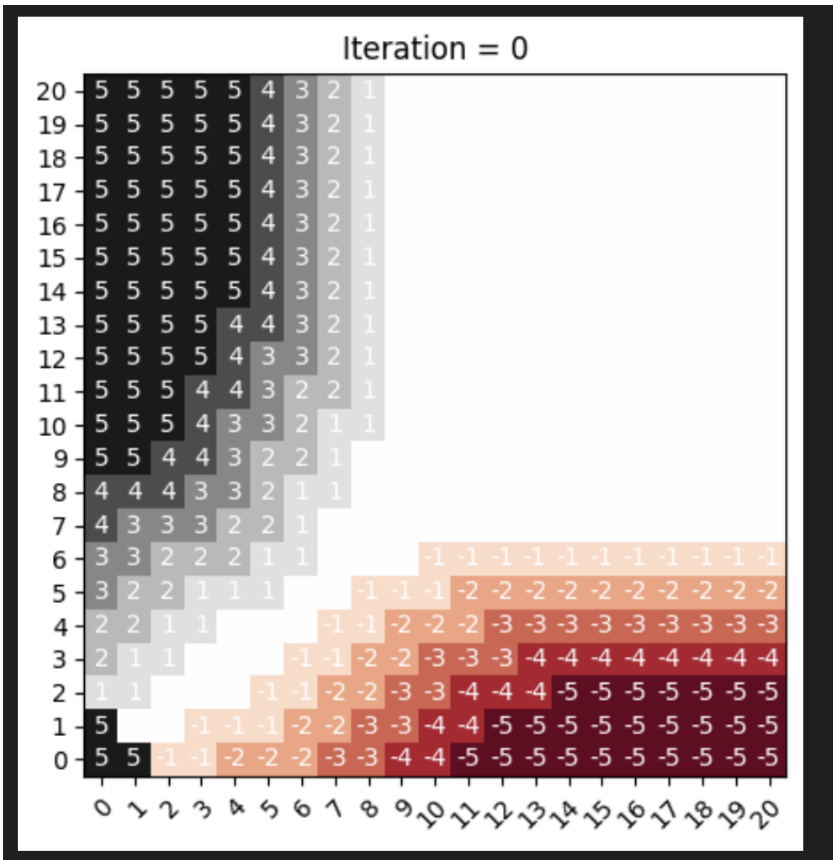
```
============================
==  Optimal State Value    ==
============================
[[22.  24.4 22.  19.4 17.5]
 [19.8 22.  19.8 17.8 16. ]
 [17.8 19.8 17.8 16.  14.4]
 [16.  17.8 16.  14.4 13. ]
 [14.4 16.  14.4 13.  11.7]]
============================
============================
==     Optimal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

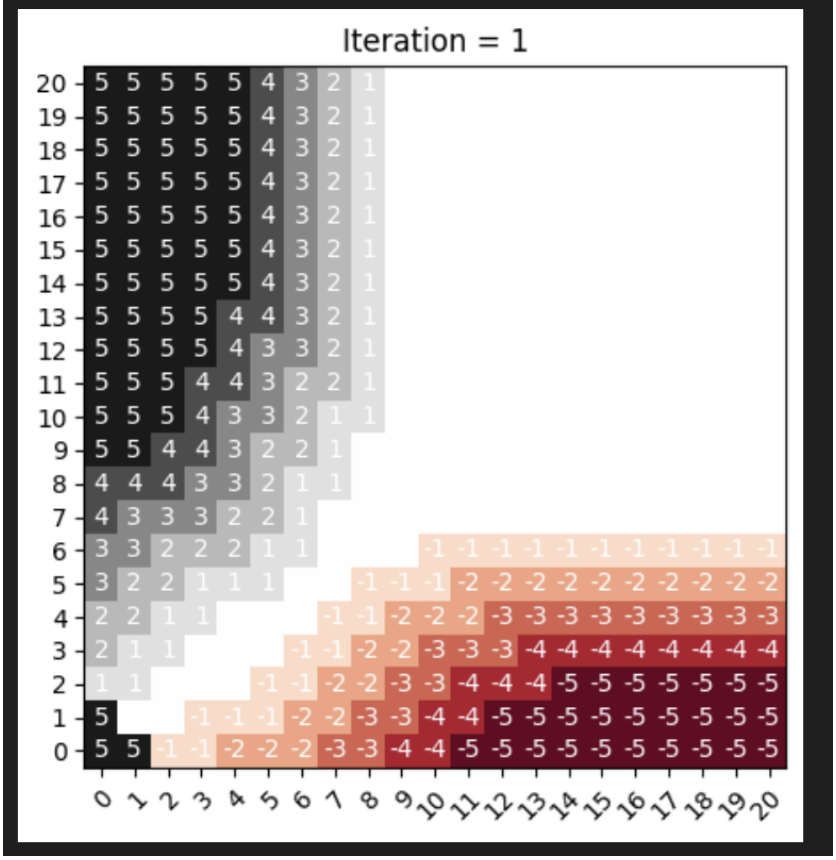
b. Plot:

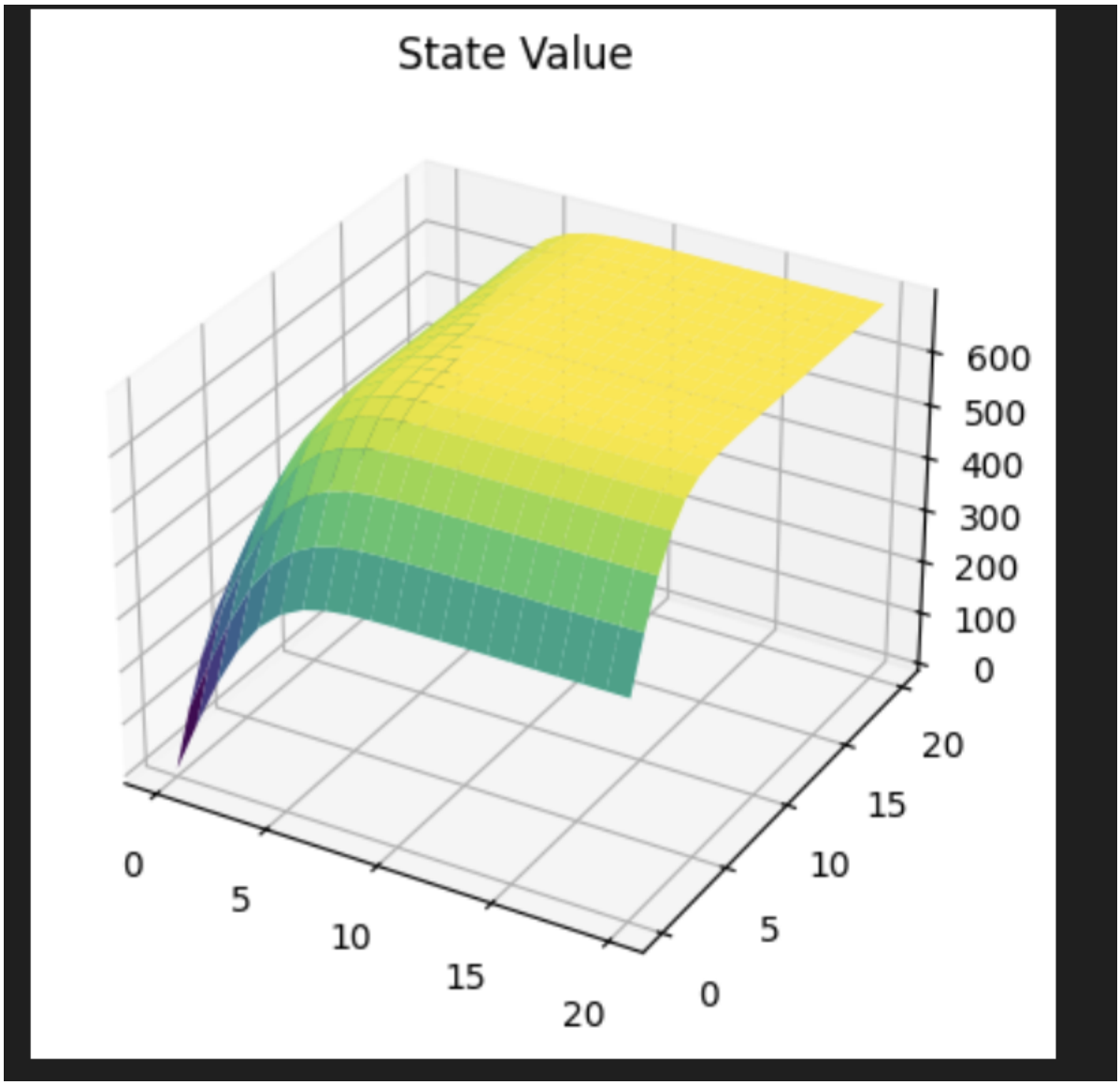
```
============================
==  Optimal State Value   ==
============================
[[22.  24.4 22.  19.4 17.5]
 [19.8 22.  19.8 17.8 16. ]
 [17.8 19.8 17.8 16.  14.4]
 [16.  17.8 16.  14.4 13. ]
 [14.4 16.  14.4 13.  11.7]]
============================
============================
==     Optimal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

Question 5:

a. Plot:


Iteration = 0
20 5 5 5 5 5 4 3 2 1
19 5 5 5 5 5 4 3 2 1
18 5 5 5 5 5 4 3 2 1
17 5 5 5 5 5 4 3 2 1
16 5 5 5 5 5 4 3 2 1
15 5 5 5 5 5 4 3 2 1
14 5 5 5 5 5 4 3 2 1
13 5 5 5 5 4 4 3 2 1
12 5 5 5 5 4 3 3 2 1
11 5 5 5 4 4 3 2 2 1
10 5 5 5 4 3 3 2 1 1
9 5 5 4 4 3 2 2 1
8 4 4 4 3 3 2 1 1
7 4 3 3 3 2 2 1
6 3 3 2 2 2 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
5 3 2 2 1 1 1 -1 -1 -1 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
4 2 2 1 1 -1 -1 -2 -2 -2 -3 -3 -3 -3 -3 -3 -3 -3 -3
3 2 1 1 -1 -1 -2 -2 -3 -3 -3 -4 -4 -4 -4 -4 -4 -4 -4
2 1 1 -1 -1 -2 -2 -3 -3 -4 -4 -4 -5 -5 -5 -5 -5 -5 -5
1 5 -1 -1 -1 -2 -2 -3 -3 -4 -4 -5 -5 -5 -5 -5 -5 -5 -5 -5
0 5 5 -1 -1 -2 -2 -2 -3 -3 -4 -4 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Iteration = 1
20 5 5 5 5 5 4 3 2 1
19 5 5 5 5 5 4 3 2 1
18 5 5 5 5 5 4 3 2 1
17 5 5 5 5 5 4 3 2 1
16 5 5 5 5 5 4 3 2 1
15 5 5 5 5 5 4 3 2 1
14 5 5 5 5 5 4 3 2 1
13 5 5 5 5 4 4 3 2 1
12 5 5 5 5 4 3 3 2 1
11 5 5 5 4 4 3 2 2 1
10 5 5 5 4 3 3 2 1 1
9 5 5 4 4 3 2 2 1
8 4 4 4 3 3 2 1 1
7 4 3 3 3 2 2 1
6 3 3 2 2 2 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
5 3 2 2 1 1 1 -1 -1 -1 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
4 2 2 1 1 -1 -1 -2 -2 -2 -3 -3 -3 -3 -3 -3 -3 -3 -3
3 2 1 1 -1 -1 -2 -2 -3 -3 -3 -4 -4 -4 -4 -4 -4 -4 -4
2 1 1 -1 -1 -2 -2 -3 -3 -4 -4 -4 -5 -5 -5 -5 -5 -5 -5
1 5 -1 -1 -1 -2 -2 -3 -3 -4 -4 -5 -5 -5 -5 -5 -5 -5 -5 -5
0 5 5 -1 -1 -2 -2 -2 -3 -3 -4 -4 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

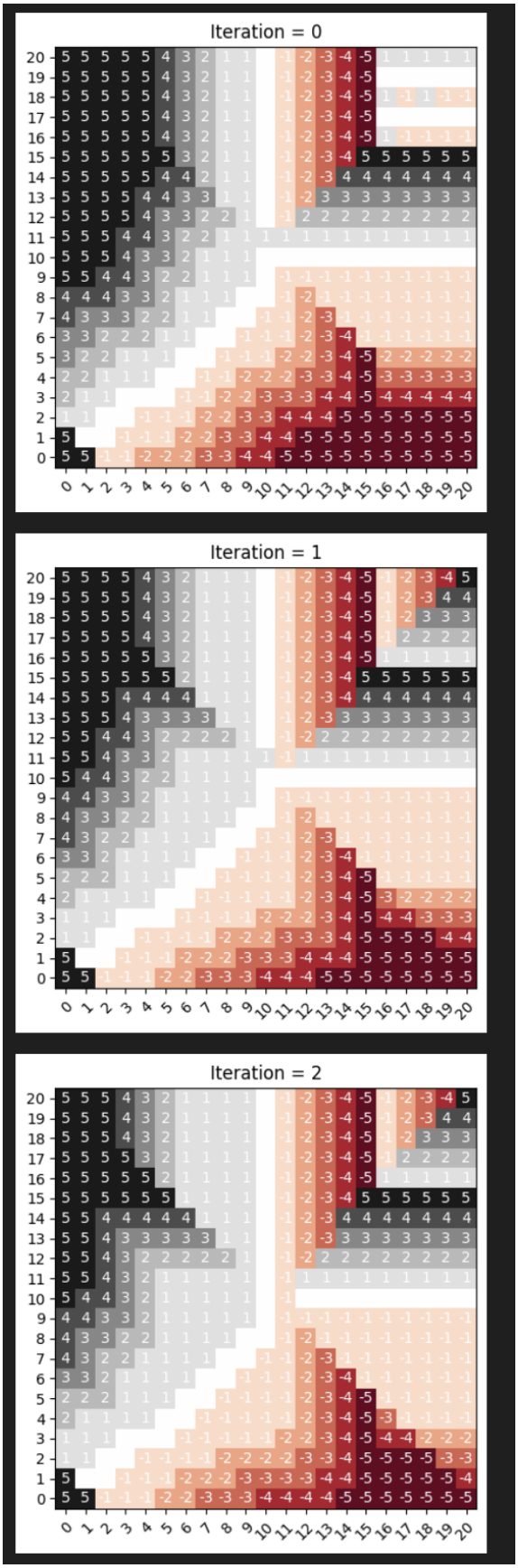


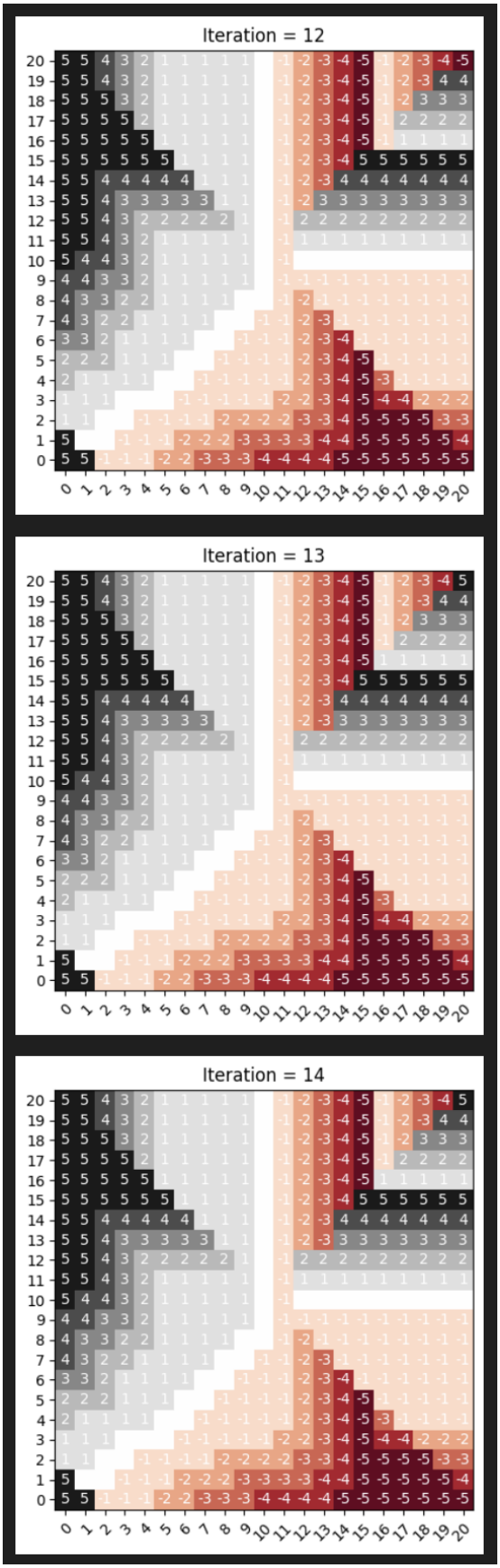
b. Written:

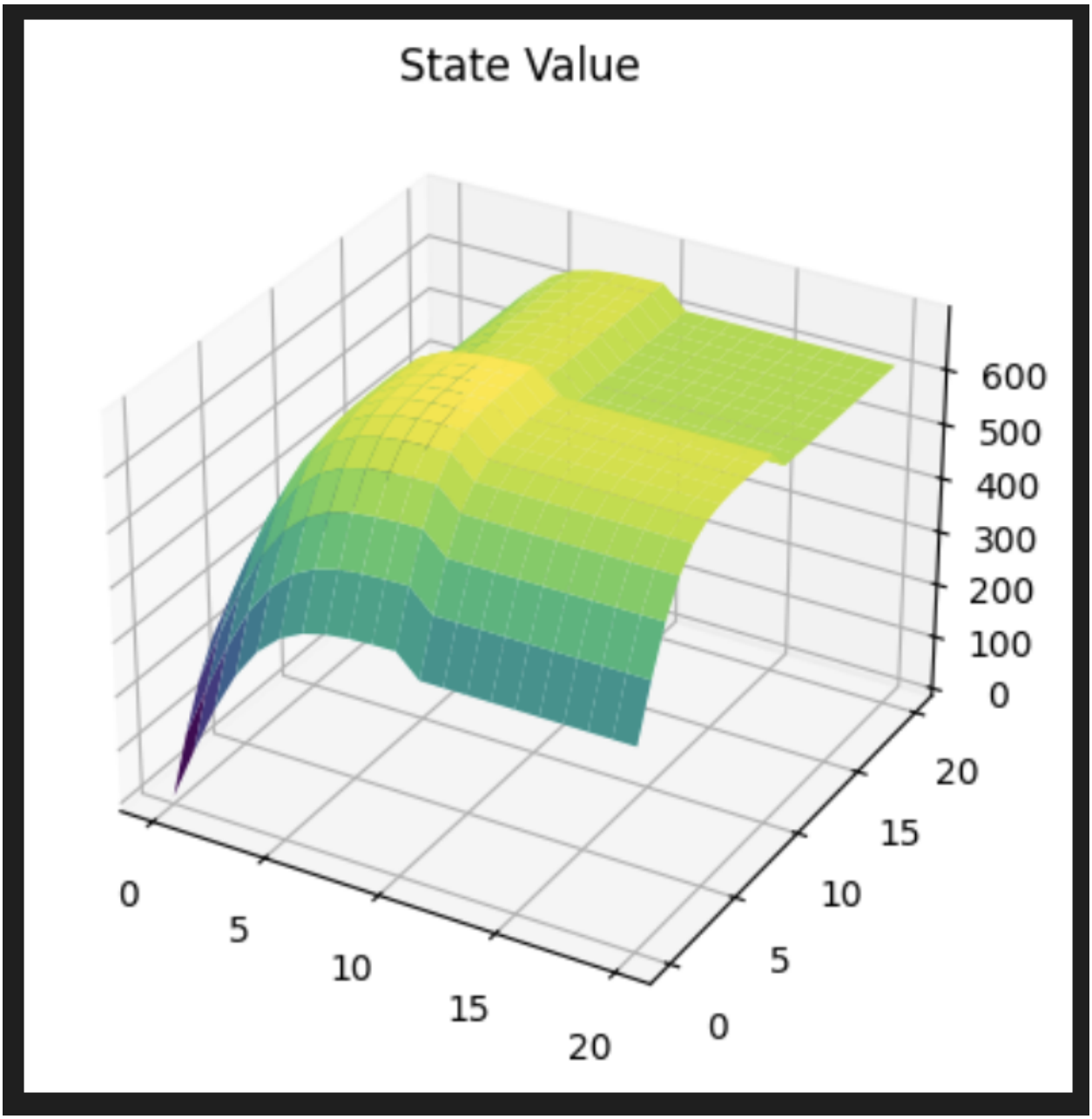
I changed the reward function so that if the absolute value of moved_cars > 0 i.e. if at least 1 car was moved from lot 1 to lot 2 then we have a moving cost of -2* (abs(moved_cars)-1) this reflects that one of Jacks employees can drive 1 car from lot 1 to lot 2 for free if needed.

I also added in that if car_num_lot1 is greater than 10 then the cost for lot 1 is self.r_lot1[10] - 4 to represent the cost for storing the 10 cars and the cost of 4 for the overflow parking lot. I did this same if statement for car_num_lot2 as well.

Plot:


Iteration = 0
Iteration = 1
Iteration = 2


Iteration = 12
Iteration = 13
Iteration = 14



Written:

The lefthand side of the policy graph(up to 8/9 on the x axis) looks very to each other and the slight indent right 1 for the modified rewards case makes sense as we have the new employee who can take a car for free. The other main difference is the top right of the modified policy is full of values while the non-modified is all 0. This makes sense as in the modified problem we add in the extra parking lots to make it feasible to handle overflow, this means that there are ways to handle having 10+ cars in both lots.

6.

a. Written:

During the policy improvement step, when taking the argmax over actions, the algorithm might oscillate indefinitely between two actions that are both optimal. Currently, ties are broken by randomly choosing between the value-maximizing actions. To prevent this, we can always select the first action that appears from the argmax. This would ensure that the same optimal action is chosen in each iteration, setting the policy-stable flag to true and guaranteeing convergence.

b. Written:

No, an analogous bug does not exist in value iteration. While value iteration uses the argmax to select the best action, it focuses on updating the value function rather than alternating between policies. Even if multiple actions have the same value, value iteration will continue updating the values until they converge to a fixed point. At that stage, a stable optimal policy is derived. Therefore, oscillation between equally good actions doesn't affect convergence, ensuring that value iteration will always reach a stable solution.