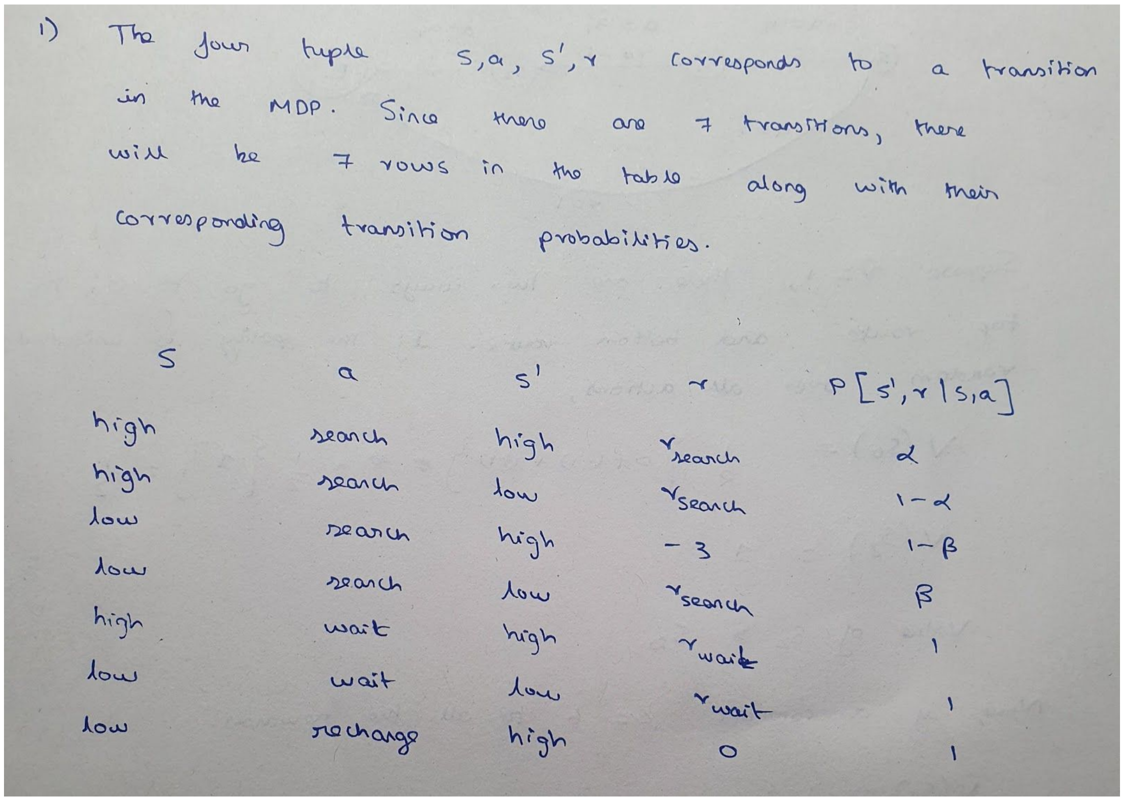
RL Assignment 2

Author
Nihesh Anderson (2016059)

Problem 1



1) The four tuple s, a, s', r corresponds to a transition in the MDP. Since there are 7 transitions, there will be 7 rows in the table along with their corresponding transition probabilities.

s	a	s'	r	$p[s', r \mid s, a]$
high	search	high	r_{search}	α
high	search	low	r_{search}	$1-\alpha$
low	search	high	-3	$1-\beta$
low	search	low	r_{search}	β
high	wait	high	r_{wait}	1
low	wait	low	r_{wait}	1
low	recharge	high	0	1

Problem 2:
Simulation: python3 q2.py
Running the code will output the value function obtained for the given policy

In this problem, we formulate the value function V(s) as the variables of a linear program. There are 25 such variables. Corresponding to every state V(S), the bellman equation is V(S) = sum over all actions a { PI(a | S) * (reward + GAMMA * (V (S’ | S,a))}

There are exactly 25 equations and 25 variables. Therefore, a closed form solution can be obtained. The code has been explained inline

Problem 3

Q3 Ex 3.15) Suppose V_c gets added to all the state values. We need to show that there exists such a V_c such that bellman's optimality holds.

We know $v(s) = \sum_{a \in A} 0.25\,(r(s,a) + \gamma\, v(s'|s,a))$

Suppose c gets added to $r(s,a)$ $\forall a$,

$$v(s) + V_c = \sum_{a \in A} 0.25\,(r(s,a) + c + \gamma\{v(s'|s,a) + V_c\})$$

$$v(s) + V_c = (c + \gamma V_c) + \sum_{a \in A} 0.25\,(r(s,a) + \gamma\{v(s'|s,a)\})$$

$$\cancel{v(s)} + V_c = (c + \gamma V_c) + \cancel{v(s)}$$

$$\Rightarrow V_c = \frac{c}{1-\gamma}$$

Ex 3.16)

Consider the following MDP: Start state $= S_0$
End state $= E$

$$\begin{tikzcd}
S_0 \arrow[r, "a=a_1,\ r=0"] \arrow[rrr, bend right, "a=a_2"', "r=1"] & S_1 \arrow[r, "a=a_1,\ r=-1"] & S_2 \arrow[r, "a=a_1,\ r=1"] & E
\end{tikzcd}$$

Suppose $\gamma = 1$, there are two ways to go to E; the top route and bottom route. If the policy is uniformly random over all actions,

$$V(S_0) = \frac{1}{2}\{0 + (-1) + (+1)\} + \frac{1}{2}\{1\} = \frac{1}{2}$$

$$V(S_2) = 1$$

Value of $S_2 > S_0$

Now, if a constant $c = 6$ to all the rewards.

$$V(S_0) = \frac{1}{2}\{6 + 5 + 7\} + \frac{1}{2}\{7\} = 12.5$$

$$V(S_2) = 7$$

Now, value of $S_0 > S_2$

The relative valuation has changed, therefore adding a constant does have an effect in an episodic task

Problem 4

Simulation: python3 q4.py

When the program is simulated, the code prints the optimal value function returned by the linear program and the optimal greedy policy

In this problem, we wish to solve bellman's optimality condition. However, the formulation itself is nonlinear.

V(S) = max over all actions a { reward + GAMMA * (V (S' | S , a)) }

We break down this condition for every single action a_i and write 4 constraints for every state, as follows

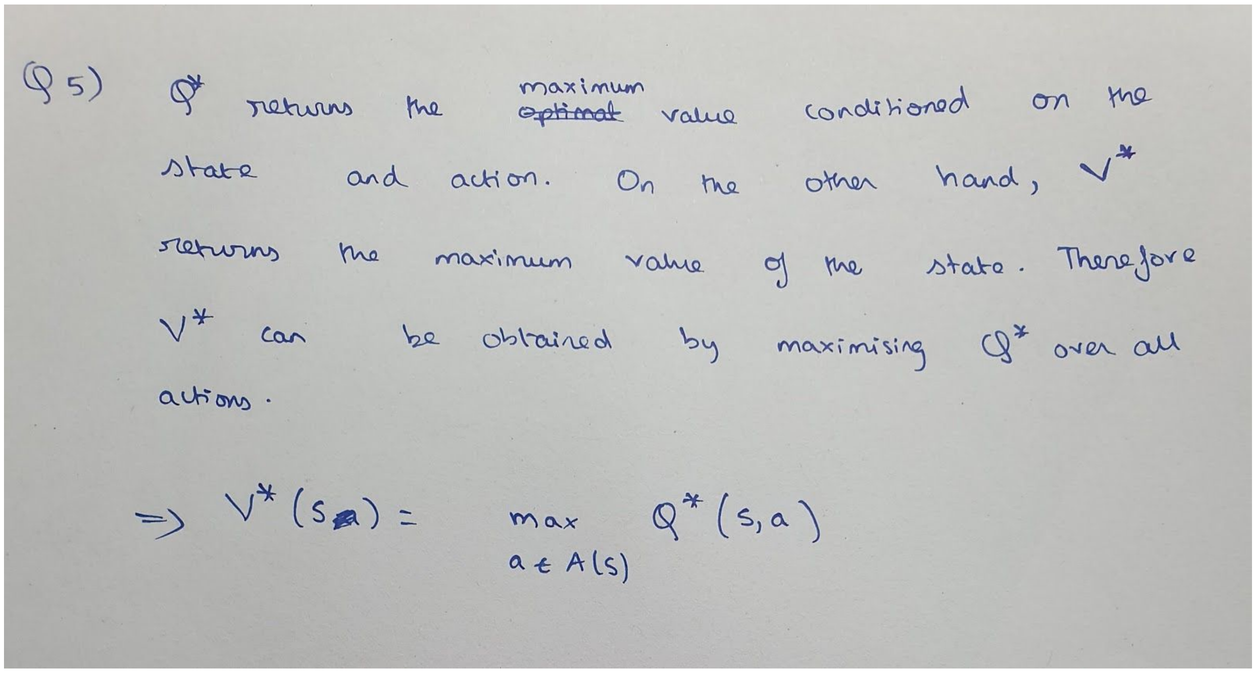
V(S) >= reward + GAMMA * (V (S' | S , a_i))

Now, the objective is to find the minimum such V(S) for which this inquality is satisfied, because we know what this inequality is an equality for one of the constraints.

Therefore, we minimise sum over all states s { V(s) }

The code has been explained inline.

Problem 5



Q5) Q^* returns the ~~optimal~~ maximum value conditioned on the state and action. On the other hand, V^* returns the maximum value of the state. Therefore V^* can be obtained by maximising Q^* over all actions.

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

Problem 6

Policy iteration
Simulation: python3 q6a.py
Output: Log of value function for each epoch

In this problem, policy iteration algorithm has been implemented. It can be seen that the value of every state increases monotonically, showing that the policy improvement step works. The policy oscillation bug has been fixed by detecting if the value function converged, instead of checking if the policy function converged

Value iteration
The implementation is very similar to policy iteration. The value function has been printed after each epoch and it can be observed that the value function improves as the epochs increase. We

The code has been explained inline.

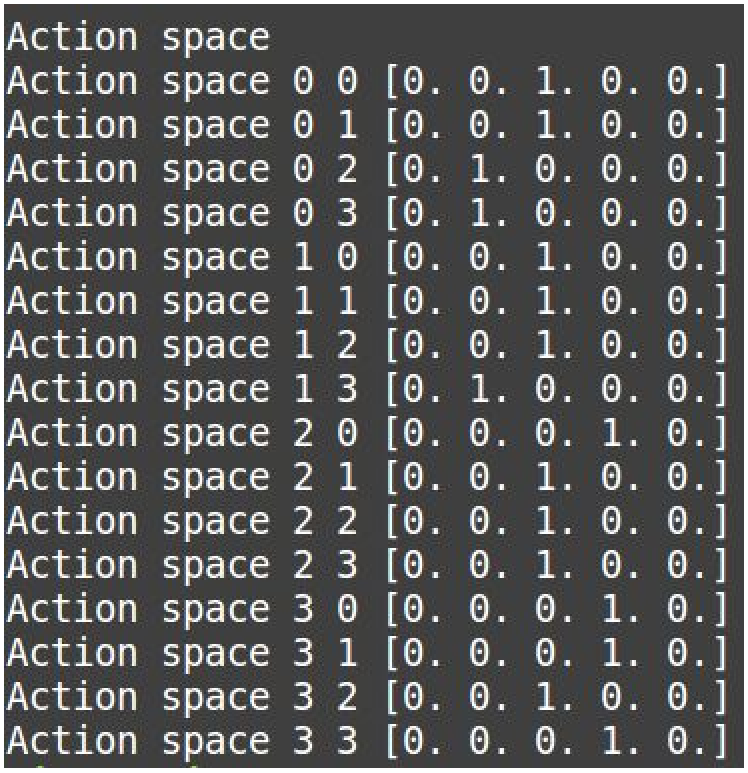
Problem 7

Simulation: python3 q7.py
Output: Log of value function after each epoch and the final action space

In this problem, the state space is very high. The poisson random variable can generate any value between 0-20 and there are 4 such variables. So, we have 20^4 possible states that we can go to, given the current state. It’s not possible to compute the answer for such a high number in a few minutes, therefore, I have reduced the value of the following parameters:

Maximum number of cars that can be stored = 3
Maximum number of cars that can be requested or returned = 2
Maximum number of cars that can be transferred = 2
Mean of cars requested in both locations = 1
Mean of cars returned in both locations = 2
Max cars that can be stored without paying a fee = 2

After implementing the program, the following output was obtained. Intuitively, these are the right set of actions, therefore, it is justified that the program is working as desired.



```
Action space
Action space 0 0 [0. 0. 1. 0. 0.]
Action space 0 1 [0. 0. 1. 0. 0.]
Action space 0 2 [0. 1. 0. 0. 0.]
Action space 0 3 [0. 1. 0. 0. 0.]
Action space 1 0 [0. 0. 1. 0. 0.]
Action space 1 1 [0. 0. 1. 0. 0.]
Action space 1 2 [0. 0. 1. 0. 0.]
Action space 1 3 [0. 1. 0. 0. 0.]
Action space 2 0 [0. 0. 0. 1. 0.]
Action space 2 1 [0. 0. 1. 0. 0.]
Action space 2 2 [0. 0. 1. 0. 0.]
Action space 2 3 [0. 0. 1. 0. 0.]
Action space 3 0 [0. 0. 0. 1. 0.]
Action space 3 1 [0. 0. 0. 1. 0.]
Action space 3 2 [0. 0. 1. 0. 0.]
Action space 3 3 [0. 0. 0. 1. 0.]
```

In this image, the 5 dimensional vector corresponds to actions = -2, -1, 0, 1, 2 where positive number indicates that the car is moved from first location to the second location.

When there is an excess in one location, it is evident that the agent is trying to move the car to the second location. Also, when both the places are full, the agent is trying to move the car to create an overflow so that the car is removed, thereby minimising parking charges.