RL Assignment 2
Vishaal Udandarao
2016119

For answers to theory questions 1, 3 and 5, please check theory answers in hw2_theory.pdf.

Note: For question 1, there is an assumption that in each state transition because of one action, there will only be two possible rewards that can be accumulated i.e. 1 if the can was collected and 0 if a can was not collected.

2) For solving the Bellman equations, we need to solve a system of linear equations. Therefore, we solve a linear system of equations of the form Ax=b. Each linear equation will have at max 4 such variables since there are 4 possible actions in each case. Also, there will be 25 such unique variables that are possible in this linear system of equations. There will be 25 such equations, one for each state. Hence, the coefficient matrix that we will solve is A=25x25 dimensional. and the constant vector will be b=25x1. Hence, the final state value function will be a 25x1 dimensional vector which will be the final state values. The final state value function that we converge to is:

```
array([[ 3.3,  8.8,  4.4,  5.3,  1.5],
       [ 1.5,  3. ,  2.3,  1.9,  0.5],
       [ 0.1,  0.7,  0.7,  0.4, -0.4],
       [-1. , -0.4, -0.4, -0.6, -1.2],
       [-1.9, -1.3, -1.2, -1.4, -2. ]])
```

4) For solving the Bellman optimality equations, we need to solve a system of non-linear equations. For each action, we will have a separate non linear equation since each action will give us a different state value function update. Therefore, since we have to take the max of all the state value function updates from each action, we essentially solve a maximisation problem of Ax>=b. Hence, we will have num_states*4 such equations i.e. there are 100 equations and similar to the above case, there will be 25 variables. Hence, A in our case will be 100x25 dimensional and the constant vector b will be 100x1 dimensional. Hence, the final state value function will be a 25x1 dimensional vector similar to the question above. The optimal state value function that we converge to, and the optimal policy is:

```
The optimal value function is:
[[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]]

The optimal policy is:
E,SENW,W,SENW,W
EN,N,NW,W,W
EN,N,NW,NW,NW
EN,N,NW,NW,NW
EN,N,NW,NW,NW
```

6) **Policy Iteration**:

```
Final optimal policy value function:
[[ 0. -1. -2. -3.]
 [-1. -2. -3. -2.]
 [-2. -3. -2. -1.]
 [-3. -2. -1.  0.]]

Final optimal policy
-,W,W,SW,
N,NW,SENW,S,
N,SENW,SE,S,
EN,E,E,-
```

The above diagram shows the stable optimal state value function as well as the optimal policy that we converge to after policy iteration.

Value Iteration:

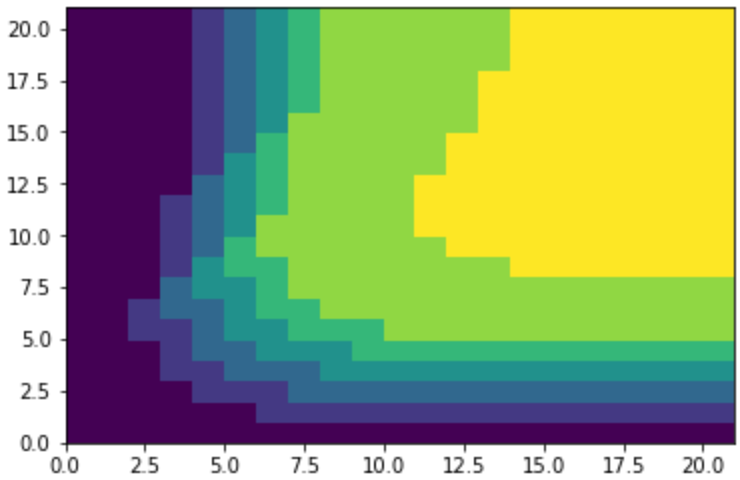
```
Final optimal policy value function:
[[ 0. -1. -2. -3.]
 [-1. -2. -3. -2.]
 [-2. -3. -2. -1.]
 [-3. -2. -1.  0.]]

Final optimal policy
-,W,W,SW,
N,NW,SENW,S,
N,SENW,SE,S,
EN,E,E,-
```

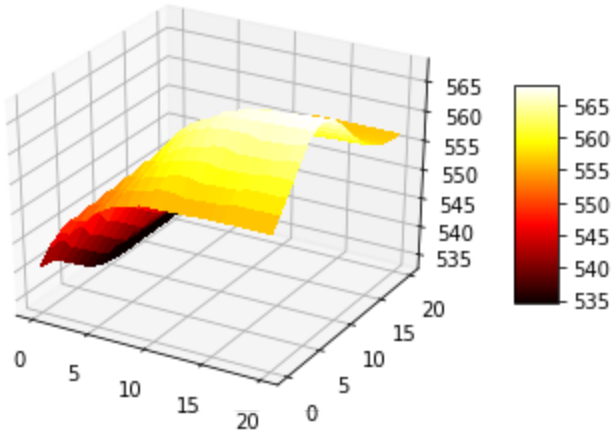
Similarly as shown above, the above diagram shows the stable optimal state value function as well as the optimal policy that we converge to after value iteration.

Bug fix (ex. 4.4): For fixing the bug mentioned in ex 4.4, we need to fix the deterministic action policy that they choose at every policy improvement step. The deterministic action policy enforces that we choose only a single optimal action (at random), from all the optimal actions possible for that particular state. Hence, this may lead to oscillations in the optimal policy. To fix this, we consider a stochastic policy update wherein we take all of the optimal policy actions in every step and then weight each of these optimal actions equiprobably. Hence, this will ensure that there is no randomness in which optimal action we select out of all the optimal actions, since we keep a track of all optimal actions possible for that state.

7) For the optimal policy pi* and optimal state value function v_pi* are plotted as follows:



Optimal policy pi* (Here: dark blue indicates 5 and yellow indicates 0, gradient is smooth decrease from 5 (dark blue)to 0 (yellow))



Optimal state value function v_pi*

We can therefore clearly see from the optimal policy that whenever both the locations have high number of cars, then the optimal action is to not transfer any cars. However, when any one location has high number of cars and the other has less number of cars, then optimal action is to send 5 cars from the location with higher number of cars to the other location.