<u>Reinforcement Learning</u>

Homework 2 Report

Abhishek Agarwal
2016126

Ans 1)

*Attached below

Ans2)

There are 5 types of cells in the grid: Corner cells, Edge cells, Middle cells, cell A and cell B.
For each type of these cells equation is of a standard form corresponding to the cell type. The cell type is determined first and then the corresponding coefficients for each state are calculated and the equation is determined.
After forming linear equation of type Ax = b, x is calculated as x = inv(A) * b.

The values obtained are as follows:

```
[[ 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. ]]
```

These values exactly match with the values given in the book. (Values have been rounded to 1 decimal digit).

Ans3)

*Attached below

Ans4)

In this part since we need to find the optimal policy which is done using the bellman optimality equations. These equations are non-linear.

Since in the bellman equation we need to take a max over all actions, the matrix which we make contains the equation for each possible action on each state. We have 25 states and 4 actions on each state, thus we have a matrix of size 100*25.

To solve these nonlinear equations, we write them in the form Ax >= b.
We take the cost function as 1 for each coefficient. Next we use the optimise function in the scipy library to find the optimal solution.

The optimal value function obtained is:

```
v* =    [[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 corresponding policy is:

```
[['R', 'URDL', 'L', 'URDL', 'L'],
 ['UR', 'U', 'UL', 'L', 'L'],
 ['UR', 'U', 'UL', 'UL', 'UL'],
 ['UR', 'U', 'UL', 'UL', 'UL'],
 ['UR', 'U', 'UL', 'UL', 'UL']]
```

Ans5)
*Attached below

Ans6) With both the methods namely policy iteration and value iteration, the same matrix for v* and optimal policy is obtained.

```
v* =
[[ 0 -1 -2 -3]
 [-1 -2 -3 -2]
 [-2 -3 -2 -1]
 [-3 -2 -1  0]]
```

Optimal Policy:
[['' 'L' 'L' 'DL']
 ['U' 'UL' 'URDL' 'D']
 ['U' 'URDL' 'RD' 'D']
 ['UR' 'R' 'R' '']]

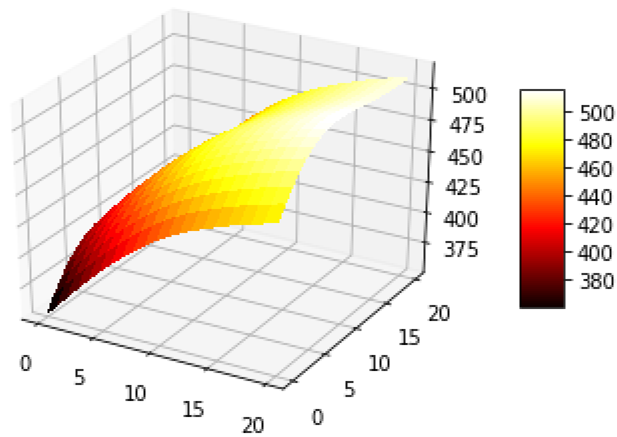
First and last states are empty because they are terminal states.

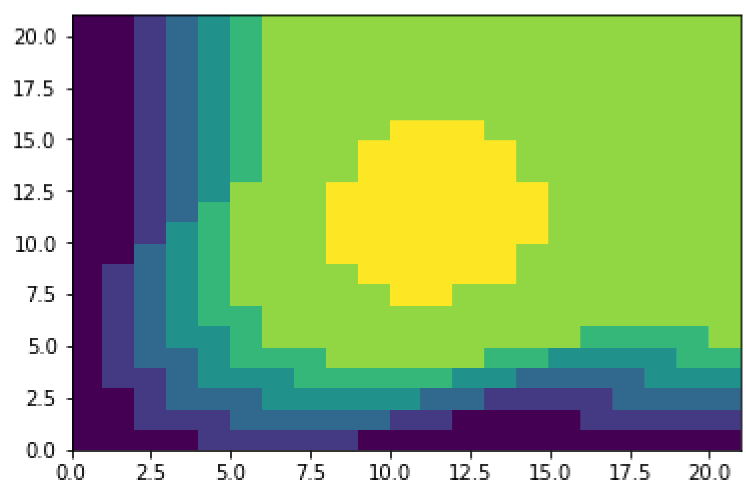
Bug: As per the pseudo code we are supposed to take the optimal action deterministically and thus if we do not have a proper ordering of the actions, it may lead to different optimal policy on every run. To fix this we have given equal weightage to all the possible optimal actions by assigning them equal probabilities.

Ans7)
We can see that the value of delta converges to less than 0.01 in about 25 iterations. After this improvement takes place and again the policy evaluation takes place. The subsequnet plots are generated after every run. With time the policy gets better.
The plots and delta values are as below:

```
delta =  167.55783985680554
delta =  116.1443174827184
delta =  78.3254865984508
delta =  61.72408268838737
delta =  48.503390805397544
delta =  37.19069812693397
delta =  28.598928100794808
delta =  22.308565753999233
delta =  18.35509133824175
delta =  15.116113205895601
delta =  12.42031099180889
delta =  10.183732320496347
delta =  8.33323620086992
delta =  6.806248840529122
delta =  5.549508072344793
delta =  4.51777784212851
delta =  3.672747748726181
delta =  2.9821012277603813
delta =  2.41870412851938
delta =  1.9598826563426996
delta =  1.586776958903954
delta =  1.2837653539295388
```

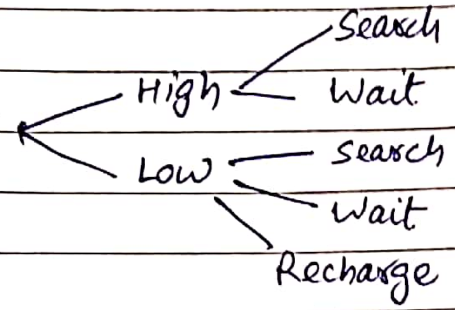
delta = 1.0379566963522961
delta = 0.8387479868881655





The rest of the plots after every iteration are in the notebook file(q7.ipynb).

Ans 1)



We need to find $p(s', r \mid s, a)$

$$p(s', r \mid s, a) = p(s' \mid s, a) \times p(r \mid s, a, s')$$

(Assuming the independence of s' and r).

expected reward i.e.

$$r(s, a, s') = \sum_{r} r \, p(r \mid s, a, s')$$

$\therefore$ The table can be constructed using this formula

state	action	s'	r	$p(s', r \mid s, a)$
high	search	high	0	$\alpha(1 - r_{search})$
high	search	high	1	αr_{search}
high	wait	high	0	$1 - r_{wait}$
high	wait	high	1	r_{wait}
high	search	low	0	$(1-\alpha)(1-r_{search})$
high	search	low	1	$\beta(1-\alpha) r_{search})$
low	search	high	-3	$1-\beta$
low	wait	low	1	r_{wait}
low	wait	low	0	$1 - r_{wait}$
low	recharge	high	0	1
low	search	low	0	$\beta(1 - r_{search})$
low	search	low	1	βr_{search}

__/__/__

3.15 The learning of the agent is not influenced by the sign of the rewards in an absolute scale. It only depends on how the rewards are relatively given. A state from which we know that a particular action performs better in the long run should have a higher reward relative to the other actions leading to a less preffered state. Thus, the rewards might all be -ve / all be +ve / Or of mixed signs, just the relative difference matters.

Although, if the signs of the reward are inverted, the learning would be affected because the ~~relative~~ difference b/w them is changed.

Eq. 3.8 $\Rightarrow$ $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$

$$\Rightarrow G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Also, $v_\pi(s) = E_\pi [G_t \mid S_t = s]$

$$= E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$$

Now on adding c to every reward,

$$G_t = (R_{t+1} + c) + \gamma (R_{t+2} + c) + \dots$$

$$\Rightarrow G_t = \sum_{k=0}^{\infty} \gamma^k (R_{t+k+1} + c)$$

$$\therefore v'_\pi(s) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k (R_{t+k+1} + c) \mid S_t = s \right]$$

$$= E_\pi \left[\sum_{k=0}^{\infty} R_{t+k+1} \mid S_t = s \right] + c \, E_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mid S_t = s \right]$$

$= E_\pi\left[\sum_{k=0}^{\infty} R_{t+k+1} \mid S_t = s\right] + c \times \frac{1}{1-\gamma}$ {since $\gamma<1$}

$V'_\pi(s) = V_\pi(s) + \frac{c}{1-\gamma}$

$\Rightarrow \underline{V'_\pi(s) = V_\pi(s) + V(c)}$

where $V(c) = c/_{1-\gamma}$.

(b) episodic task.

similar to the above part, here $t = T$ instead of $t = \infty$,

$\therefore V'_\pi(s) = V_\pi(s) + c\, E\left[\sum_{k=0}^{T} \gamma^k \mid S_t = s\right]$

$$\underline{V_c = c\left[\frac{\gamma^{T+1} - 1}{\gamma - 1}\right]}$$

We see that V_c depends on T.

$\therefore$ for the same episode (single episode) the value of T is constant and hence it won't affect the learning algo. Whereas, across different episodes we might obtain different values of V_c.

Q.5 $V_*(s) = \max_\pi V_\pi(s)$

$= \max_{a \in A(s)} q_{\pi_*}(s,a)$

$= \max_{a \in A(s)} E[G_t \mid S_t = s, A_t = a]$

__/__/__

$$= \max_{a \in A(s)} \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma V_{\pi}(s')\right]$$

Also,

$$V_{*}(s') = \max_{a' \in A(s')} q_{*}(s', a')$$

$$\Rightarrow \boxed{V_{*}(s) = \max_{a \in A(s)} \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma \max_{a' \in A(s')} q_{\pi*}(s', a')\right]}$$

Scanned by CamScanner