<u>ASSIGNMENT – 2</u>

Part2. Fig 3.2

v(s) with random policy

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

To solve the system of linear equations of Bellman Equation, following steps were used :
Known:

general form of linear equations is Ax = B ....(1)

bellman equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \quad \forall s \in \mathcal{S},$$

1) so for one state let's say s = (0,0), $v_\pi(0,0)$ is :

where $\pi(a/s)$ = 0.25, discount = 0.9

$v_\pi(0,0)$ = 0.25*1*(-1 + discount*$v_\pi(0,0)$) + 0.25*1*(-1 + discount*$v_\pi(0,0)$) + 0.25*1*(-1 + discount*$v_\pi(0,1)$) + 0.25*1*(-1 + discount*$v_\pi(1,0)$) ....(2)

rearranging (2), we get

0.55*$v_\pi(0,0)$ + **0.225***$v_\pi(0,1)$ + **0.225***$v_\pi(1,0)$ = **0.5**

which is now of the form (1)

2) similarly, created the coefficient matrix A of the size 25*25 which was the p(a/s) * discount forr every state. Subtracted -1 from the state for which coefficients are build up.

3) the B vector(25*1) here created was rewards * p(a/s)

where p(a/s) is the probability of the action a taken given state s

4) the values computed using np.linalg.solve(A,B) are the x values.

Part 4. Fig 3.5

$v_*(s)$ with optimal policy

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

To solve the system of non- linear equations of Bellman Equation, following steps were used :

Known:

general form of linear equations is Ax >= B ....(1)

bellman equation = instead of summation over all actions, we have max operation

1) so for one state let's say s = (0,0), $v_\pi(0,0)$ is :

where $\pi(a/s) = 0.25$, discount = 0.9

$v_\pi(0,0) = \max(1*(-1 + discount*v_\pi(0,0)) , 1*(-1 + discount*v_\pi(0,0)) , 1*(-1 + discount*v_\pi(0,1)) , 1*(-1 + discount*v_\pi(1,0)))$ ....(2)

rearranging (2), we get

$\max(0.1 *v_\pi(0,0) , -0.9*v_\pi(0,1) ,0.1 *v_\pi(0,0) , -0.9*v_\pi(1,0)) = [-1,0,-1,0]$

which is now of the form (1)

2) similarly, created the coefficient matrix A of the size 100*25 which was the discount forr every state.

3) the B vector(100*1) here created was rewards

4) for defining the objective function, we created the C matrix of size (25*1) with all ones which minimizes , since computed values were passed with reversed signs.

5) the values computed using scipy.optimize.linprog(C,A,B) are the x values.

Part 6.
Example 4.1

Grid world

0.0	-14.0	-20.0	-22.0
-14.0	-18.0	-20.0	-20.0
-20.0	-20.0	-18.0	-14.0
-22.0	-20.0	-14.0	0.0

With Value Iteration : Convergence in 4 iterations
Sample Iterations :

	index	VI
0	0	[[0. -1. -1. -1.] [-1. -1. -1. -1.] [-1. -1. -1. -1.] [-1. -1. -1. 0.]]
1	1	[[0. -1. -2. -2.] [-1. -2. -2. -2.] [-2. -2. -2. -1.] [-2. -2. -1. 0.]]
2	2	[[0. -1. -2. -3.] [-1. -2. -3. -2.] [-2. -3. -2. -1.] [-3. -2. -1. 0.]]
3	3	[[0. -1. -2. -3.] [-1. -2. -3. -2.] [-2. -3. -2. -1.] [-3. -2. -1. 0.]]

$V_*(s)$	**$pi_*(s)$**
Grid world : V(s) with Value Iteration	Grid world : pi(s) with Value Iteration

Grid world : V(s) with Value Iteration

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

Grid world : pi(s) with Value Iteration

0.0	0.0	0.0	0.0
1.0	0.0	0.0	3.0
1.0	0.0	2.0	3.0
1.0	2.0	2.0	0.0

With Policy Iteration : Convergence in 3 iterations

Sample Iterations

	index	PE	PI	Pol_stable
0	0	[[0. -1. -2. -3.] [-1. -2. -3. -2.] [-2. -3. -2. -1.] [-3. -2. -1. 0.]]	[[0. 0. 0. 0.] [1. 0. 0. 3.] [1. 1. 2. 3.] [1. 2. 2. 0.]]	False
1	1	[[0. -1. -2. -3.] [-1. -2. -3. -2.] [-2. -3. -2. -1.] [-3. -2. -1. 0.]]	[[0. 0. 0. 0.] [1. 0. 0. 3.] [1. 0. 2. 3.] [1. 2. 2. 0.]]	False
2	2	[[0. -1. -2. -3.] [-1. -2. -3. -2.] [-2. -3. -2. -1.] [-3. -2. -1. 0.]]	[[0. 0. 0. 0.] [1. 0. 0. 3.] [1. 0. 2. 3.] [1. 2. 2. 0.]]	True

$V_*(s)$

Grid world : V(s) with Policy Iteration

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

$pi_*(s)$

Grid world : pi(s) with Policy Iteration

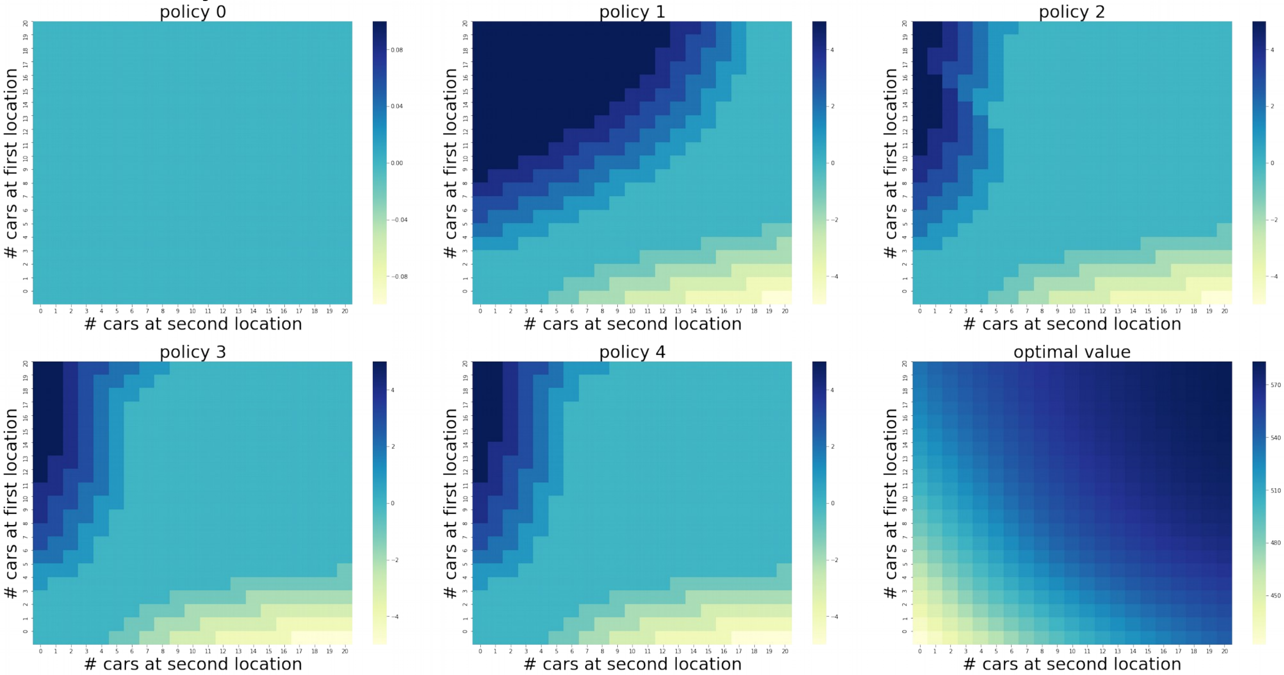
0.0	0.0	0.0	0.0
1.0	0.0	0.0	3.0
1.0	0.0	2.0	3.0
1.0	2.0	2.0	0.0

Analysis :

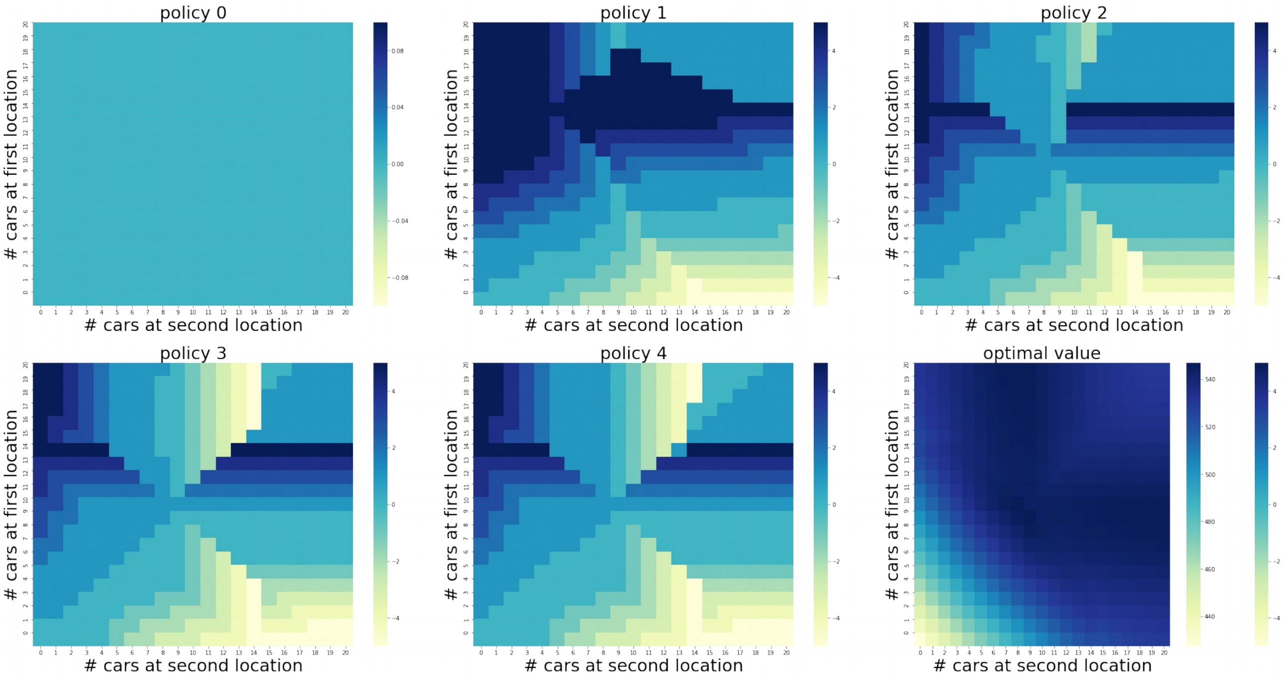
- The convergence with policy iteration happens earlier than value iteration one iteration before.
- Both methods give same optimal policy and value function.

  The bug can be fixed by comparing the values computed after every evaluation using policy. If the policy_stable flag is false, then we can compute the difference between the values from previous iteration and the current iteration and if this diference is very small(let say below some epsilon 1e-4) then we can declare the policy is stable.

Part 7.
Car Rent Policy



Exercise 4.7



Analysis :

- the policy at every state in modified car rental problem has more number of cars moving from location1 to location2 and vice versa2 than the original problem.

- The optimal values also in the modified version are lower than the values in the original, that is why also graph is more dense in original. After 10^{th} state, the values decreases as also should be the case since at any location if there are more than 10 number of cars, then additional penalty of 4 is there. This justifies the optimal values in modified version.