CS 747 - Foundations of Intelligent and Learning Agents

Programming Assignment 2

Prakhar Vijay Diwan, Roll no. 180100083

Note:
I have completed all the tasks 1,2,3. Task 1 and task 2 are completed as specified. As task 3 implementation and analysis was upto us, hence I have specified details regarding it below:

TASK 3:
For task3, kindly refer to the "README_FOR_TASK3.txt" file I assume that analysis_task3.py is in same directory as task3.py (so that it's ouptuts are in the same directory), and the initial policy for player (as desired) has to be copied as file "pi_0.txt" in the same directory as in task3.py. I have attached a pi_0.txt file as well in the submission which is same as p2policy1. Also pi_0.txt has a constraint that it has to be determnistic (which I took as the consequent policies generated are anyways deterministic (i.e. $\pi[1], \pi[2]$ *and so on..*))

1. Task 1: MDP Planning Algorithms

First of all a parser was required for planner.py for obtaining the various inputs. These were then appropriately stored in data structures, so as to be used later for obtaining optimal value function and policy.
This part was common to all three approaches.

A: Value Iteration (vi)
In this method, I used separate functions for computing optimal value function in case of episodic and continuing MDPs. I initialised the value function as all "0" actions, and then ran an infinite loop consisting of computing Q(s,a) and then getting $V_{t+1}[s]$ from max(Q[s]) and going on unless exit condition is reached. Exit condition being error squared sum of value functions (prev and next) (i.e. $(V_{t+1} - V_t)^2 < \epsilon$) less than epsilon. Here I chose $\epsilon = 1e-18$ for obtaining upto 6 digits of precision. In case of episodic states only change is just that value function of terminal states are always assigned to 0.
After obtaining the optimal value function I found Q function and from it using argmax extracted π^*(optimal policy). And then printing optimal value functions and policies to output.

B: Linear Programming (lp)
Here the pulp library was used as indicated. I made the LpProblem (with LpMinimize)and initialised LpVariables as Value function for every state. Objective function was set as sum of the value function of all states. The constraints were same as in lecture for continuing mdp:
For all s and all a,

$$V(s) \geq \sum_{s' \in S} T(s,a,s')\{R(s,a,s') + \gamma V(s')\}.$$

Figure 1: Constraint

And in case of episodic tasks, difference in constraints being that: value function was assigned 0 for terminal states, and the constraint in Figure1 being applied to the non-terminal states.
After this I used the PULP_CBC_CMD solver and used it to obtain optimal value function.
After obtaining the optimal value function I found Q function and from it using argmax extracted π^*(optimal policy). And then printing optimal value functions and policies to output.

C: Howard's Policy Iteration (hpi)
Here first I computed the optimal value function (though optimal policy is obtained first here, I returned optimal value function from my functions) separately in case of episodic and continuing tasks. So mainly this value function consisted of 2 steps: policy evaluation and policy improvement. These 2 steps were done untill the next_policy and prev_policy converged. The exit condition was that next_policy and prev_policy must be identical for all states. The initial policy was taken as all 0 actions again.

The Policy evaluation step was different for episodic and continuing MDPs. The evaluation function was obtained by solving the set of NumStates linear equations using the AV=B $\rightarrow V = A^{-1}B$ matrices. Here the coefficients of A and B were obtained as per the following (as in lecture slides):

- For $s \in S$:

$$V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s')\{R(s, \pi(s), s') + \gamma V^{\pi}(s')\}.$$

Again in case of episodic tasks, the equations corresponding to terminal states were just that the corresponding value functions were 0. So the A and B matrix were appropriately edited as you can see in code.
Next comes policy improvement: here I chose the improving actions for all improvable states, by choosing the action at the end (i.e. the max i (action) belonging to [1,9] which gives a greater Q(s_p,i) value than the Policy evaluated function for that state s_p). Here an epsilon =(1e-6) was used as a threshold before switching policy at state s_p:

```
if(q_val>(eval_Q[s1]+eps2)):
    policy_improved[s1] = a
```

Figure 2: eps2

This was done since otherwise in case of terminal states since eval_Q will be 0 it will lead to unnecessary switching.
After obtaining the optimal value function I found Q function and from it using argmax extracted π^*(optimal policy). And then printing optimal value functions and policies to output.

A nice optimization: I did the following and got significant improvement in performance hence thought should mention: (Related to computation of Q matrix)

```
Q = np.zeros((num_states,num_actions))
#for s1 in range(num_states):
#    for a in range(num_actions):
#        Q[s1][a] = sum((Tran_prob[s1][a][s2]*(Reward[s1][a][s2] + gamma*prev_valfn[s2])) for s2 in range(num_states))
Q = np.sum(Tran_prob*(Reward + gamma*prev_valfn),axis=2) #optimization
```

Figure 3: Optimization

The uncommented code was used before and last line does it's work in an optimizaed fashion (vectorization)

2. Task 2: MDP for Anti-Tic-Tac-Toe Game

Also describe how you formulated the MDP for the Anti-Tic-Tac-Toe problem: that is, for Task 2

Formulating the MDP for ATTT: (Building of encoder.py)

Here, I just read in the input data, then observed that the opponent player being fixed policy acts as an environment, and hence I computed intermediate states that is which would be faced by opponent which actually is environment. From the opponent file based on its action probablity then the corresponding destination state for mdp (s') and the transition probability was obtained. For the Reward function I simply assigned 0 reward for loser or draw states and 1 for winner states. The winner states were obtained using the check_if_winner functions which checked the rows, cols and diagonals with the opponent ID and returned whether that state is a winner state or not. I chose a value of $\gamma = 1$ as I wanted to weight the end reward (like wanted to see far-away reward) with equal weights as others. Other than that task is episodic so was printed appropraitely as mdptype. ***The 2 Terminal states****:* The main issue I observed was about the terminal states. Here I encoded all of the terminal states as 2 super states (hence you see the += in tran_prob inside planner so as to sort this out): one which has upon reaching gives reward 1 and other which gives 0 reward upon reaching. Again there were 2 kinds of terminal states, 1 where agent itself goes into terminal states and others. These 2 had state values of unrewarding_terminal_S = "111111111" and rewarding_terminal = "222222222" which makes sure they don't interfere with other calculations. Then appropriately printing was done; and also adding above 2's index as end states. I used index of the state values as state number. Actions were already provided in a similar fashion.

Decoding the planner output: (Building of decoder.py)

Here, I took care of the fact that in case value function was 0 corresponding to a state then the lowest possible action be chosen (0→ lowest, 8→ highest), this had to be done to prevent it from choosing illegal actions. Apart from this when value function was non-zero then the action as per the optimal policy was taken.

Many tests were done and the policies it gave were indeed found optimal wrt fixed opponent policy inputted.

3. Task 3: Anti-Tic-Tac-Toe Optimal Strategy

Question: Is the sequence of policies generated for each player guaranteed to converge? If yes, provide a proof of convergence, and describe the properties of the converged policy. If not, either give a concrete counter-example, or qualitative arguments for why the process could go on for ever. If your answer depends on implementational aspects such as tie-breaking, be sure to specify your assumptions.
Answer: Yes, the policies are guaranteed to converge, as per the results I obtained upon emperical analysis of task 3 with different starting policies for player 2 & 1 i.e. $\pi[0]$ "pi_0.txt". I observed that both player 1 and player 2 policies converge after some iterations (as in the problem statement of task3). The snapshots of output of analysis_task3.py are shown below for policies of player 1and player 2.
A. $\pi[0] = p2\_policy1.txt$ as init policy for player 2 : [Deterministic input policy]

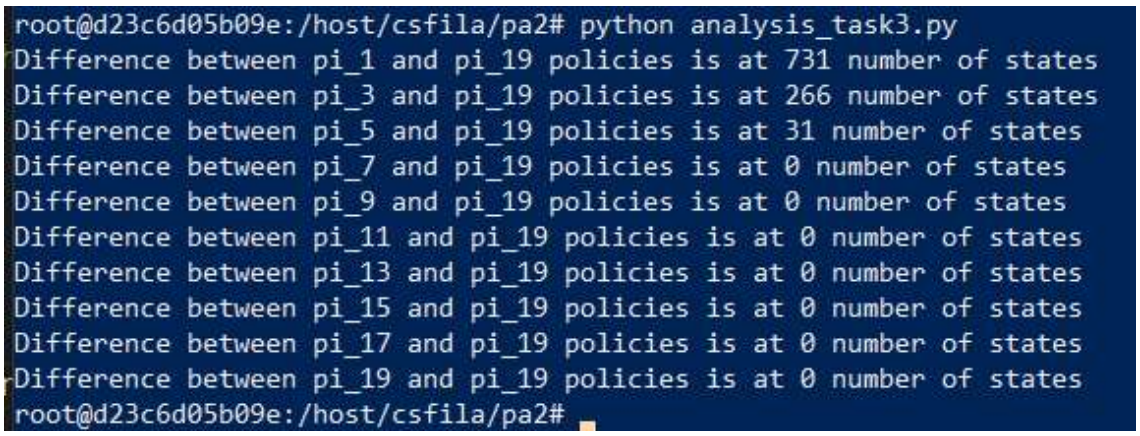


Figure 4: Player 1 Policies Converging

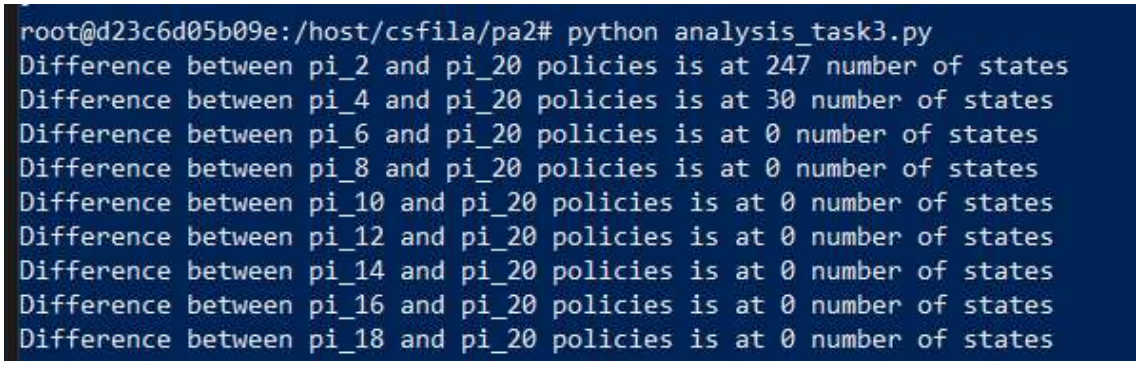


Figure 5: Player 2 Policies Converging

B. $\pi[0] = p2\_policy2.txt$ as init policy for player 2 : [Stochastic input policy]

```
root@d23c6d05b09e:/host/csfila/pa2# python analysis_task3.py
Difference between pi_1 and pi_19 policies is at 927 number of states
Difference between pi_3 and pi_19 policies is at 113 number of states
Difference between pi_5 and pi_19 policies is at 12 number of states
Difference between pi_7 and pi_19 policies is at 0 number of states
Difference between pi_9 and pi_19 policies is at 0 number of states
Difference between pi_11 and pi_19 policies is at 0 number of states
Difference between pi_13 and pi_19 policies is at 0 number of states
Difference between pi_15 and pi_19 policies is at 0 number of states
Difference between pi_17 and pi_19 policies is at 0 number of states
Difference between pi_19 and pi_19 policies is at 0 number of states
root@d23c6d05b09e:/host/csfila/pa2#
```

Figure 6: Player 1 Policies Converging

```
root@d23c6d05b09e:/host/csfila/pa2# python analysis_task3.py
Difference between pi_2 and pi_20 policies is at 129 number of states
Difference between pi_4 and pi_20 policies is at 12 number of states
Difference between pi_6 and pi_20 policies is at 0 number of states
Difference between pi_8 and pi_20 policies is at 0 number of states
Difference between pi_10 and pi_20 policies is at 0 number of states
Difference between pi_12 and pi_20 policies is at 0 number of states
Difference between pi_14 and pi_20 policies is at 0 number of states
Difference between pi_16 and pi_20 policies is at 0 number of states
Difference between pi_18 and pi_20 policies is at 0 number of states
root@d23c6d05b09e:/host/csfila/pa2#
```

Figure 7: Player 2 Policies Converging

C. $\pi[0] = p1\_policy1.txt$ as init policy for player 1 : [Deterministic input policy]

```
root@d23c6d05b09e:/host/csfila/pa2# python analysis_task3.py
Difference between pi_2 and pi_20 policies is at 316 number of states
Difference between pi_4 and pi_20 policies is at 36 number of states
Difference between pi_6 and pi_20 policies is at 0 number of states
Difference between pi_8 and pi_20 policies is at 0 number of states
Difference between pi_10 and pi_20 policies is at 0 number of states
Difference between pi_12 and pi_20 policies is at 0 number of states
Difference between pi_14 and pi_20 policies is at 0 number of states
Difference between pi_16 and pi_20 policies is at 0 number of states
Difference between pi_18 and pi_20 policies is at 0 number of states
root@d23c6d05b09e:/host/csfila/pa2#
```

Figure 8: Player 1 Policies Converging

```
root@d23c6d05b09e:/host/csfila/pa2# python analysis_task3.py
Difference between pi_1 and pi_19 policies is at 392 number of states
Difference between pi_3 and pi_19 policies is at 40 number of states
Difference between pi_5 and pi_19 policies is at 0 number of states
Difference between pi_7 and pi_19 policies is at 0 number of states
Difference between pi_9 and pi_19 policies is at 0 number of states
Difference between pi_11 and pi_19 policies is at 0 number of states
Difference between pi_13 and pi_19 policies is at 0 number of states
Difference between pi_15 and pi_19 policies is at 0 number of states
Difference between pi_17 and pi_19 policies is at 0 number of states
Difference between pi_19 and pi_19 policies is at 0 number of states
root@d23c6d05b09e:/host/csfila/pa2#
```

Figure 9: Player 2 Policies Converging

D. $\pi[0] = p1\_policy2.txt$ as init policy for player 1 : [Stochastic input policy]

```
root@d23c6d05b09e:/host/csfila/pa2# python analysis_task3.py
Difference between pi_2 and pi_20 policies is at 91 number of states
Difference between pi_4 and pi_20 policies is at 16 number of states
Difference between pi_6 and pi_20 policies is at 0 number of states
Difference between pi_8 and pi_20 policies is at 0 number of states
Difference between pi_10 and pi_20 policies is at 0 number of states
Difference between pi_12 and pi_20 policies is at 0 number of states
Difference between pi_14 and pi_20 policies is at 0 number of states
Difference between pi_16 and pi_20 policies is at 0 number of states
Difference between pi_18 and pi_20 policies is at 0 number of states
root@d23c6d05b09e:/host/csfila/pa2#
```

Figure 10: Player 1 Policies Converging

```
root@d23c6d05b09e:/host/csfila/pa2# python analysis_task3.py
Difference between pi_1 and pi_19 policies is at 451 number of states
Difference between pi_3 and pi_19 policies is at 16 number of states
Difference between pi_5 and pi_19 policies is at 0 number of states
Difference between pi_7 and pi_19 policies is at 0 number of states
Difference between pi_9 and pi_19 policies is at 0 number of states
Difference between pi_11 and pi_19 policies is at 0 number of states
Difference between pi_13 and pi_19 policies is at 0 number of states
Difference between pi_15 and pi_19 policies is at 0 number of states
Difference between pi_17 and pi_19 policies is at 0 number of states
Difference between pi_19 and pi_19 policies is at 0 number of states
root@d23c6d05b09e:/host/csfila/pa2#
```

Figure 11: Player 2 Policies Converging

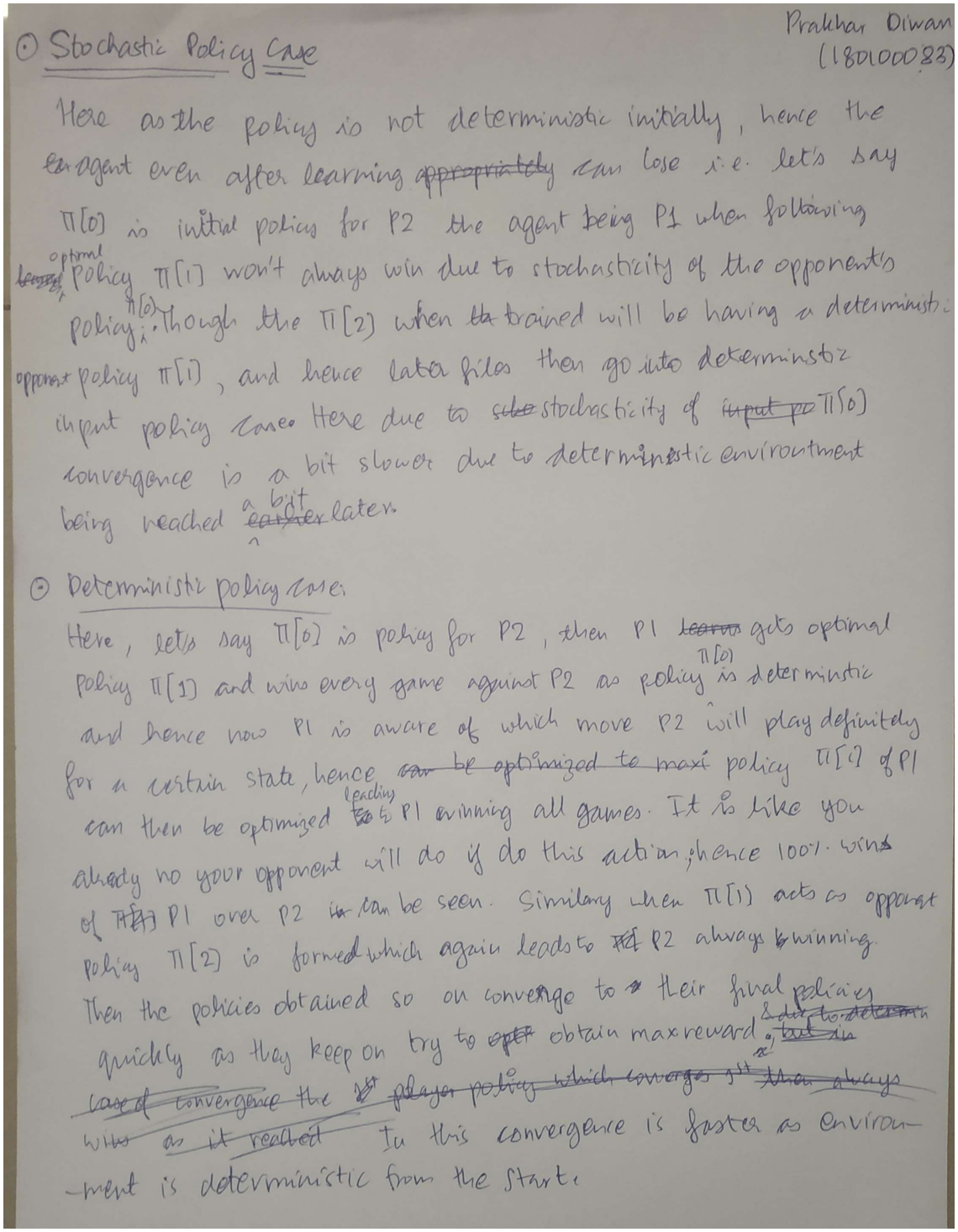
I explain the emperical patterns observed below by a proof:

High-Level Proof:

For proving the convergence, first I observed the the following to be true:

1. Since the optimal policies computed by my decoder (as choosing argmax for pi_star in planner) are **deterministic** hence it leads to the that in case policies haven't converged then the later optimized policy player always win which is expected (i.e. let's say $\pi[3]$ & $\pi[4]$ have not yet converged to their final policies, then as $\pi[4]$ is optimized based on $\pi[3]$, the player using policy $\pi[4]$ always wins). Though in case if **stochastic** input policy, then $\pi[1]$ computed will not always lead to win due to stochasticity of $\pi[0]$. Though later policies behave identically.

2. When the policies of player 1 and 2 converge, the player whose policies converges first always wins from the time onwards (This is another observation I made).
We see that there are 2 cases, one in which $\pi[0]$ is given as deterministic (case A and case C) and other where it's stochastic (case B and case D). I have handwritten the convergence argument in the below picture:



* In task3 I have attached a readme file for running, 2 files task3.py and analysis_task3.py are there. Task 3 generates the policies $\pi[1]$, $\pi[2]$ .. *so on untill* $\pi[20]$, as per the input $\pi[0]$ file and the player corresponding to which it was given, and appropriately minor edits have to be done in both files for performing a given combination. I have anyways attached the 80 policy files (20 for each of the 4 input policies provided under policy folder)
The code for the assignment can be found along with this report.pdf file in the submission.tar.gz file.