CS747 Assignment-2

Poojan Sojitra 200050137

October 12, 2022

Contents

1 Task_1

An important observation in Value Iteration and Howard's Policy Iteration is that the intermediate value functions for both the algorithms is different but the final value function is the same for both.

1.1 Random policy

We use Bellman's equations to get the linear equation in V(s). Then we matrix and matrix inversions to solve for each V(s).

1.2 Value Iteration

We here used Bellman's Optimality equations to solve through value iterations. We can use vectorization in numpy to solve the equations quickly. We used the difference between the V and V_next to be in scale of $\sim 10^{-11}$ to terminate the algorithm. Also

1.2.1 Howard's Value Iteration

Here also we use the Bellman's Optimality equations to solve it. But instead of getting the V(s) in Value iteration, we use argmax to get the better policy than the the previous policy. Then we use the same algorithm that we used in random policy to get the V(s) from policy. Again, we used the difference between the V and V_next to be in scale of $\sim 10^{-11}$ to terminate the algorithm.

1.3 Linear Programming

We use the PuLP library to encode the linear equations using the transition matrix and rewards matrix. The library helps to solve the linear programming problems.

2 Task_2

The States are encoded as the first run$\times$ balls states represent the states of the players that is controlled through actions. Next first run$\times$ balls states represent the states of environment controlled player B. Last two states represent the losing and winning states respectively. Transition function represent the probability that we transition from a given state to another state through a certain action which is derived from the player A and player B parameters. The rewards matrix is pretty simple. All transition that end up in winning state gets a probability 1 if the transition is possible.
The transition matrix is encoded in a numpy array which is of size (balls*100 + runs,5,balls*100 + runs) which is easier to access using the states name. At last section of encoder.py these states are mapped from states name to the states number i.e. 0-2*balls*runs.

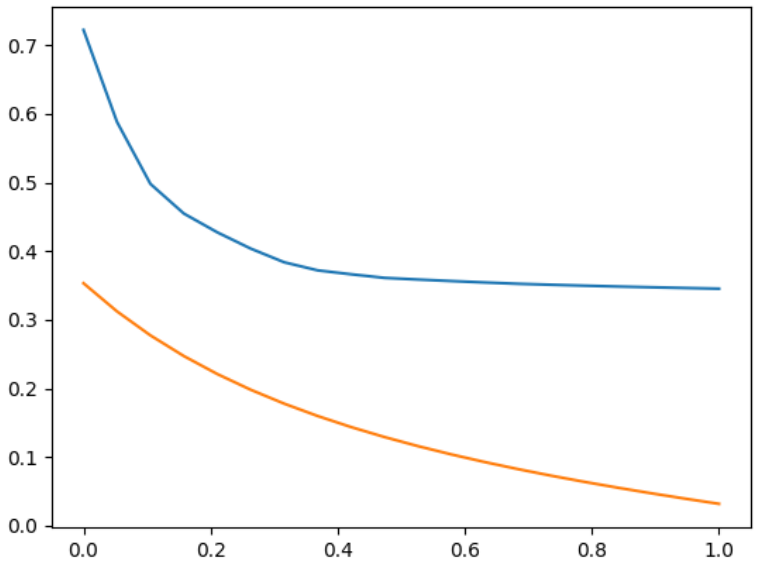

Figure 1: Varying q

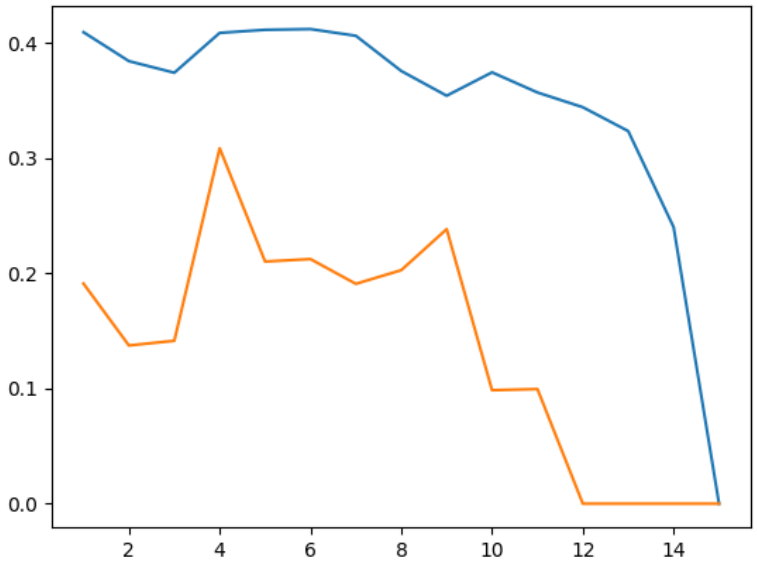

Figure 2: Varing balls

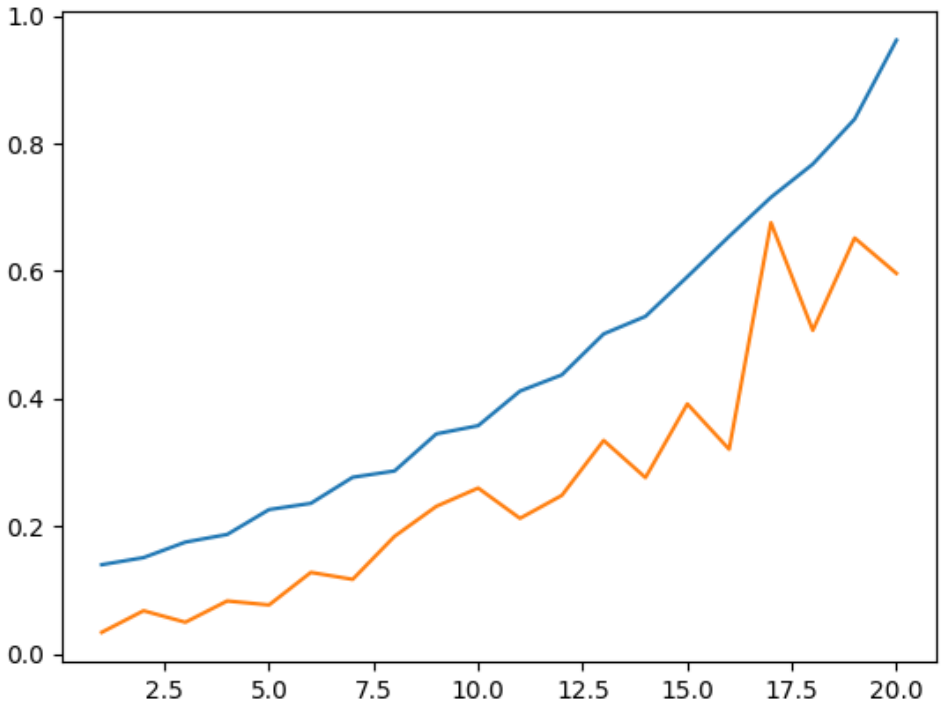

Figure 3: Varing runs