CS 747 Programming Assignment 2

Adish Shah

11th October, 2022

Contents

1 Task 1

The tie-breaking is done automatically by the np.argmax function, so nothing specific is done to resolve that. Also the algorihms implementation doesn't need the mdptype and endstates, as the transitions itself model that. To improve the performance I have implemented the algorithms in matrix form and the reference for same is added in the reference.txt.

Some observations : value iteration performs better on large size MDPs as the cost of inverting matrices is higher in hpi and LP is also slower for larger MDPs. But accuracy for value iteration is lower.

2 Task 2

2.1 MDP Formulation for the cricket problem

The number of states is max-runs $\times$ max-balls $\times$ 2 + 2. This is because, I have considered the set of states to be given by number of balls left $\times$ number of runs to be scored $\times$ strike. There are 2 additional states for winning and losing condition. The transitions are self-explanatory and are explained in my code through comments. The set of actions are 5, which are used on batsman1 and for batsman 2 they are dummy actions, as all transitions are given in terms of q. The reward is 1 if we reach the winning state and 0 for all other cases. This is an episodic MDP with discount = 1.

The plots have been genereted by the script `plot.py`, which I have included in my submission. It requires that a `plots` directory be made before running this, as all the relevant statefiles, mdps and random policies (`rand_pol.txt` extrapolated to the set of required states) that I generate are placed inside this directory. The following are the graphs plotted, which include 2 lines - one for the optimal policy and another for the arbitrary policy provided.

2.2 Analysis 1

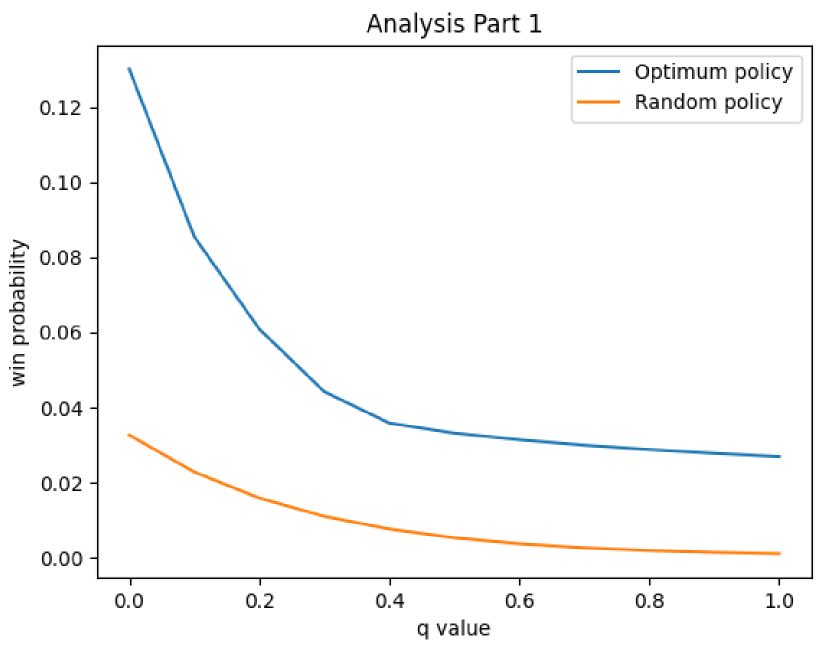

Figure 1: Win probability for fix state(15 balls, 30 runs) vs B's strength(varying q)

For both the optimal and random policy, as the value of q increases the winning probability monotonically decreases, which is as expected, because the probability of second batsman getting out is proportional to q.

Overall for a given q, the optimal policy has more winning probability than the random policy, which should be the case.

2.3 Analysis 2

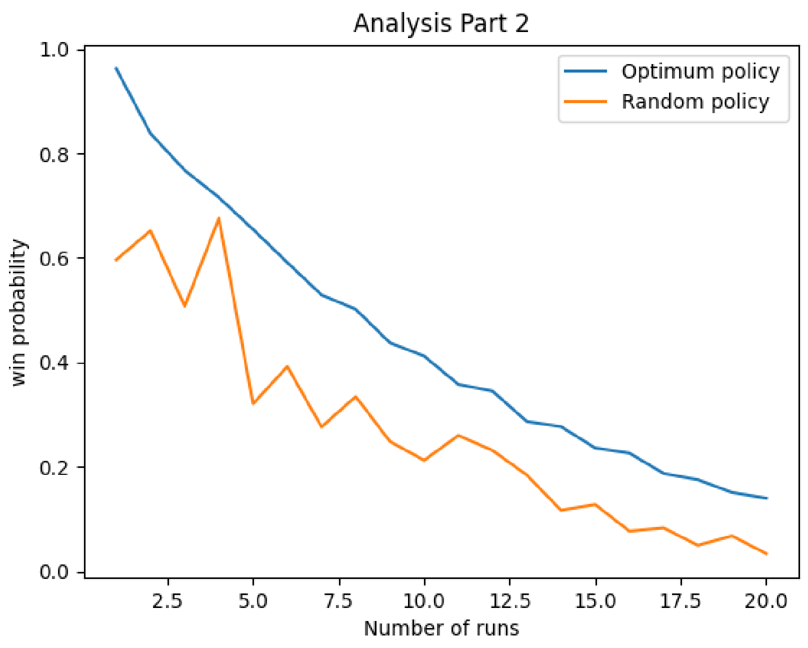

Figure 2: Winning probability v/s varying number of runs, 10 balls and q = 0.25

For the optimal policy, as the number of runs increase the winning probability decreases almost monotonically, which is as expected, as scoring more runs in fixed number of balls leads to lower win probability(at least for the sample parameters). The random policy also decreases overall, but has random spikes in between which could be due to strike rotation.

Overall for a given number of runs, the optimal policy has more winning probability than the random policy, which should be the case.

2.4 Analysis 3

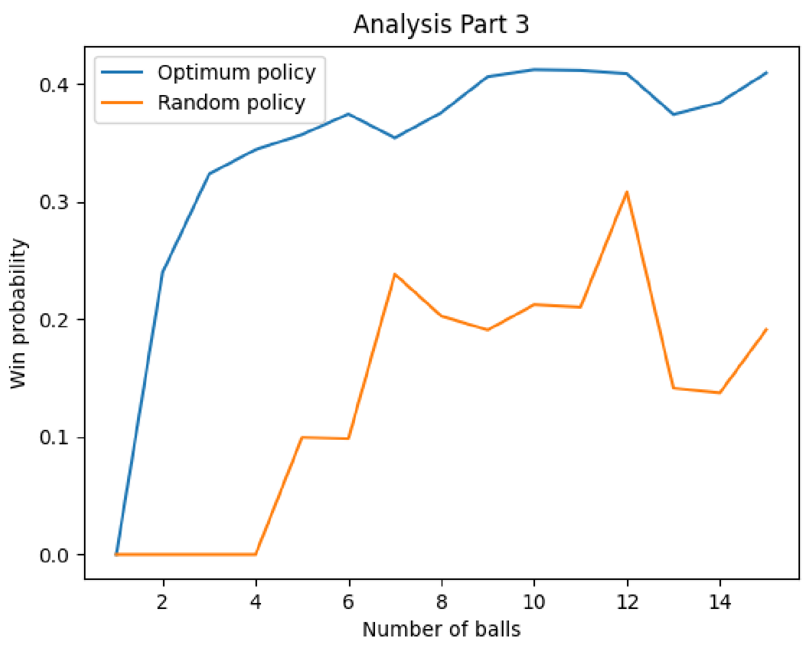

Figure 3: Winning probability v/s varying number of balls, 10 runs and q = 0.25

For the optimal policy, as the number of balls increase the winning probability increases which is as expected, as scoring a fixed number of runs in more number of balls leads to higher win probability. We can notice several dips near balls 7 and 13, which are due to strike change, due to which batsman scores fewer runs.

The random policy also increases overall, but has random spikes in between which could be due to strike rotation.

Overall for a given number of balls, the optimal policy has more winning probability than the random policy, which should be the case.