REPORT

CS 747 Assignment 2
Samarjeet Sahoo
150100017
CSE B.Tech

References to libraries used:

I have used the PuLP library as was suggested in the problem statement for solving the linear programming problem.
For solving the Bellman's equations which was required for policy iteration, I used the linalg solver of the numpy library.

Change to handle episodic tasks with gamma = 1

Linear Programming: In the case of linear programming the only change to handle episodic tasks was constraining the value of the last state (which is guaranteed to be the terminal state as per input specifications) to have a value of 0. (Done in lines 68-69 of mdp.py in extras)
Howard's Policy Iteration: In the case of Howard's policy iteration, the only change to handle episodic tasks was again constraining the value of the last state (which is guaranteed to be the terminal state as per input specifications) to have a value of 0. This is done in a way different from the linear programming obviously. The only point where this could cause a problem was while solving the Bellman's equations (to find the Value functions given some policy). So, the equation corresponding to the terminal state 's', becomes V[s] = gamma*V[s], which for gamma = 1 becomes V = V. So, to handle this, I have made the coefficient of the terminal state to be 1 and the constant term is guaranteed to be zero because the rewards for transitions from the terminal state is 0. Hence, the equation becomes V = 0, which holds for any gamma, even gamma = 1. Note that I have assumed that any reward for the transitions from the terminal state is 0. If that doesn't hold, then this may not work. (Done in lines 131-146 of mdp.py in extras)

Change to handle multiple optimal policies

Also, in Howard's Policy Iteration, when the difference between the old Values and the newly calculate optimal value becomes very small(change of 1e-6) for all states, then I have stopped the iteration. This is to take care of cases where there are multiple optimal policies possible theoretically and one should not keep oscillating between two optimal policies, rather, we quit and display any one of them.

MDP Encoding used for Gambler's Problem:

I encoded the gambler's problem as an MDP with 101 states, each corresponding to the amount of money currently with the gambler. More precisely, state i corresponds to the state where the amount of money with the gambler is i. The number of actions for each state is 51, with each action corresponding to the amount that can be placed as a bet when on a particular state. The maximum amount that can be placed on a bet is 50 and the minimum is 0, hence 51 possible actions.
I have used a discount factor of 1 for the MDP because only then the value function are same as the probability of winning the game from that state. For any other value of the discount factor, the value function would not be equal to the probability of winning from that state.

The transition function is such that for a state s and action a, the probability of going to s+a is same as the probability of heads, which is ph, and the probability of going to s-a is same as the

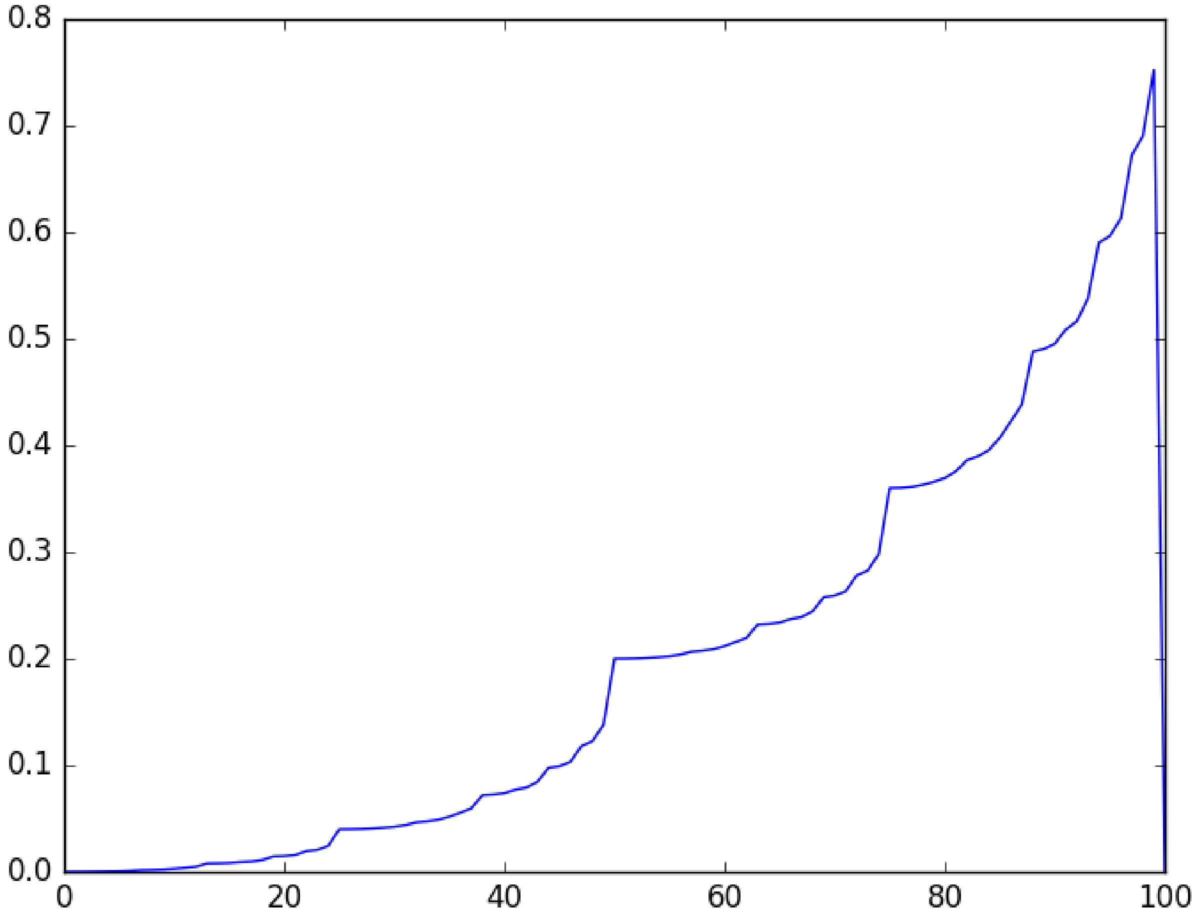
probability of tails, that is 1-ph. This is because the gambler gains money equal to the amount placed on the bet when the coin comes up heads and loses the same amount if it comes up tails.
I have handled some corner cases as follows –
I have made the transition probability of [s][0][s] = 0 for all s. (Though s+0 = s). The reason for this is that I want to disallow the amount ‘0’ to be bet. This is because, when we bet an amount of 0, we are guaranteed to stay in the same state, thus essentially, the game hasn’t changed. We want the game to approach towards the end by either going ahead by gaining more money and coming closer to state 100 or the game should approach the end by going backward by losing money and coming closer to state 0. So, it makes no difference if I place a bet of 0 and then place a bet of 2 or if I place a bet of 2 straight away, because when I place the bet of 2, I am in the same state in both the cases. Hence, I have disallowed the action ‘0’ by making its transition probability 0 for all cases.
I have also made all the transition probabilities for transitions from the terminal states to be 0, because once we reach the terminal state, the game is over, and we are not allowed to take any more actions. Hence T[100][a][s] = 0 for all a,s & T[0][a][s] = 0 for all a,s.

The reward function is such that there is a positive reward (more precisely, reward of 1) when you reach the state 100 from any other state and it is possible to reach the state 100 from that state. In all other cases, the reward is 0.

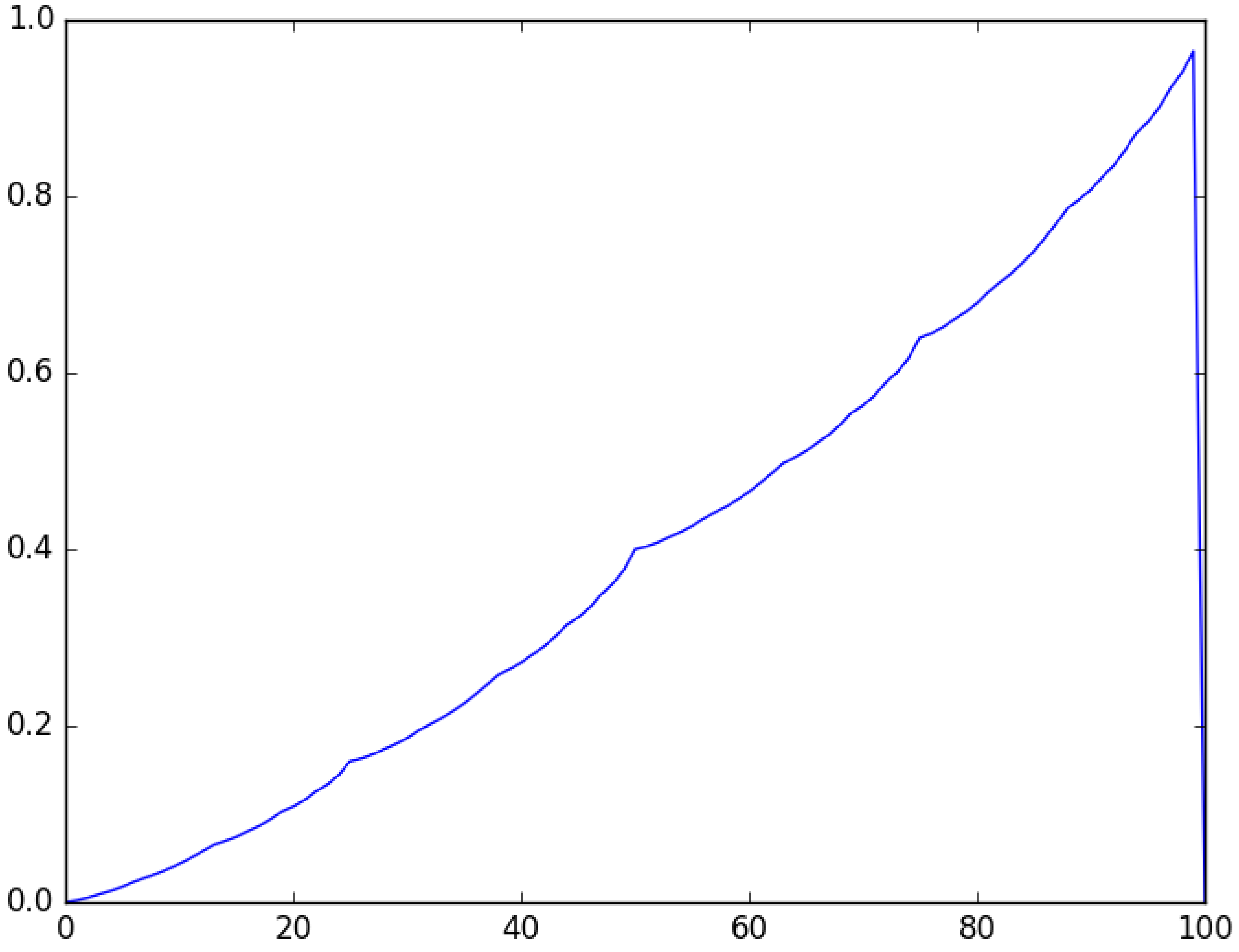
The terminal states of the MDP are 0 and 100, and this is encoded by making the transition probability of any action taken from the terminal state to be 0, which essentially means you are not allowed to take any action from the terminal states. The reward for such impossible transitions have been set to 0 as well.

Results:

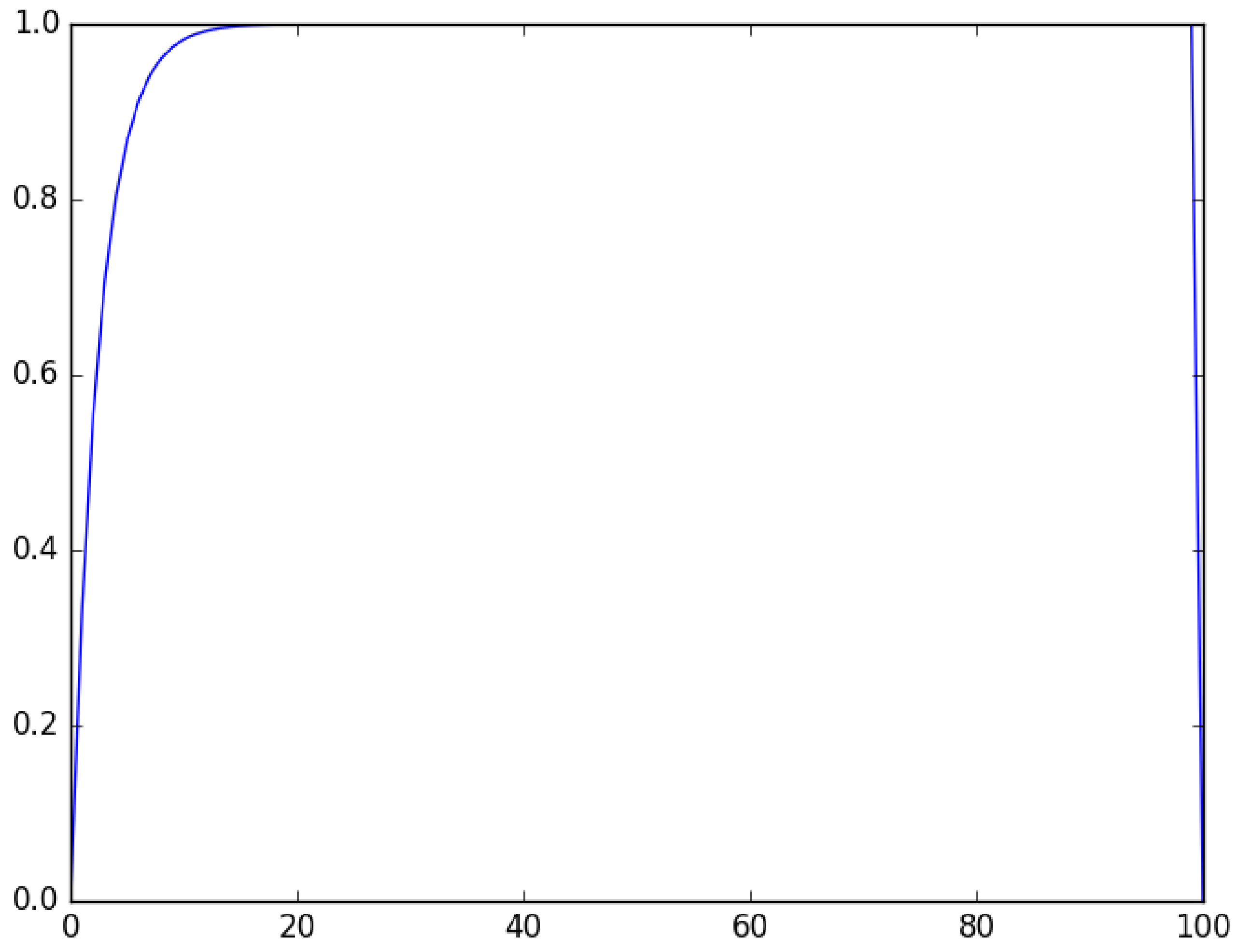
Here are the results of the experiment for probability of heads being 0.2, 0.4, 0.6, 0.8
The following graphs show the value function for each of the ph values:
Note that the value function graphs are identical for both Linear Programming and Howard’s Policy Iteration, hence I have only one graph to represent both the algorithms for a particular ph. You can find the graphs separately for each algorithm in the plots folder in the extras folder.
This is the graph for ph=0.2



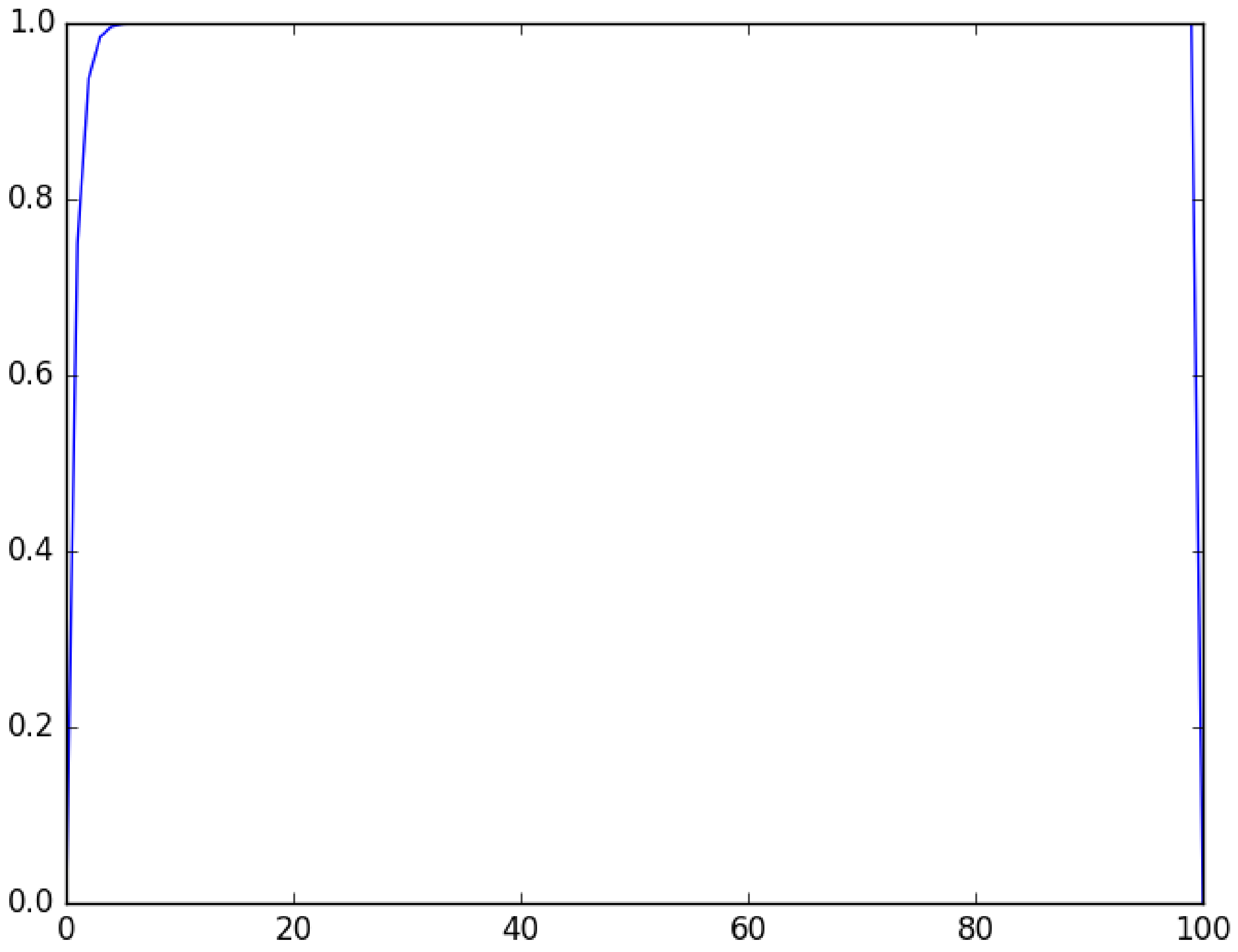
This is the graph for ph=0.4


1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

This is the graph for ph=0.6



This is the graph for ph=0.8



Observations:

As we can see, in all the graphs, the value function is an increasing function of the state. This makes sense intuitively, because it means that the more money the gambler has currently, the more likely he is to win the game by reaching 100. The value function at state 0 and state 100 is zero because the game is over and the gambler won't get any more rewards.
We also observe that the probability of winning at state 99 is more for higher ph, this also makes sense intuitively because, the gambler is more likely to win the remaining amount of 1 if the probability of heads is more.
Finally, we observe that the value function reaches 1.0 exactly in the graphs of ph=0.6 and ph=0.8. This is due to the numerical precision constraints of the machine. The actual probability is quite high and is actually very close to 1, though not equal to 1, but due to the number representation constraints, it is taken to be exactly 1. We observe that the value function reaches the highest probability earlier for ph=0.8 compared to ph=0.6. Again, this makes sense, because you are more likely to win from a particular state with probability of heads being more. Hence, we reach the top earlier in case of ph=0.8.