# Reinforcement Learning Assignment 2

Thomas Klimek

October 2, 2018

## 1 Introduction

Dynamic programming is a core concept to the field of Reinforcement learning. It refers to a set of algorithms that are used to solve finite Markov Decision Processes, where the model is known to the agent. Two of the main methods for solving such models are policy iteration and value iteration. Policy iteration works by evaluating a policy, calculating the value function, and then repeating until the policy or value function converges, however value iteration is an improvement of this idea. By combining the policy improvement and a truncated policy evaluation, that does not need to sweep the entire state space, we achieve the value iteration algorithm. The value iteration follows this update rule, which is closely related to the Belman optimality equation.

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

By truncating the policy evaluation, value iteration maintains the convergence guarantee from policy iteration, but often achieves this convergence at a much faster rate. This paper will be focused on analyzing the performance of value iteration, and implementing value iteration with respect to the Gambler's problem to discover an optimal policy, or set of optimal policies

### 1.1 Gambler's Problem

The gambler's problem can be explained as follows: "A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake. The game ends when the gambler wins by reaching his goal of 100, or loses by running out of money" (Sutton and Barto, p. 69)

We will formulate this game as a an undiscounted, episodic, and finite MDP. The state space is defined by the gambler's capital $s \in \{0, 1, 2, ..., 100\}$, and the actions are stakes for each coin flip $a \in \{0, 1, 2, ..., min(s, 100 - s)\}$. The reward is 0 for all transitions, except where the gambler reaches his goal of 100 where the reward is $+1$.

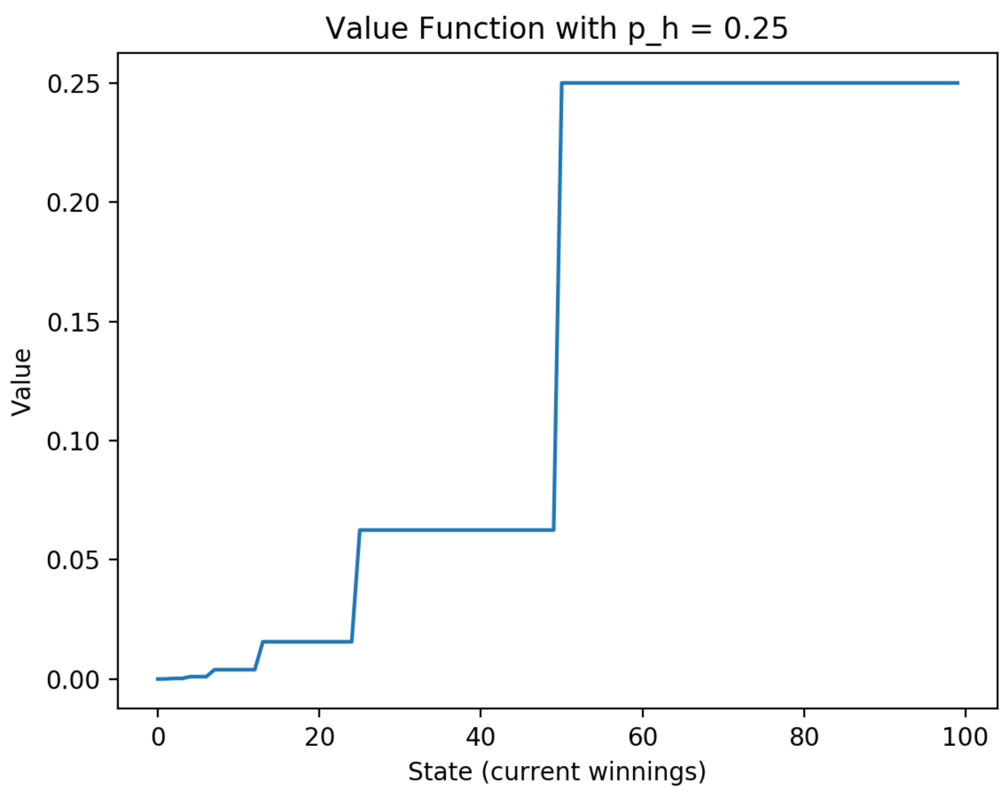
This problem is particularly interesting because there is not one optimal policy, but a set of optimal policies. This can be seen because there exist ties within the value function, meaning an optimal policy could utilize a number of choices, each with the same expected value. The set of optimal policies is also discussed here: http://incompleteideas.net/book/gamblers.html in a conversation between the author Sutton, and one of his students. It is also shown in the paper *How to Gamble If You Must*, by Kyle Siegrist that an optimal policy would be "bold play" which involves betting the largest stake possible to either win the game, or prevent you from losing the game. So it will be important to keep in mind the existence of more than one optimal policies, and how our computed optimal policy will be determined by how ties are resolved in the value function.

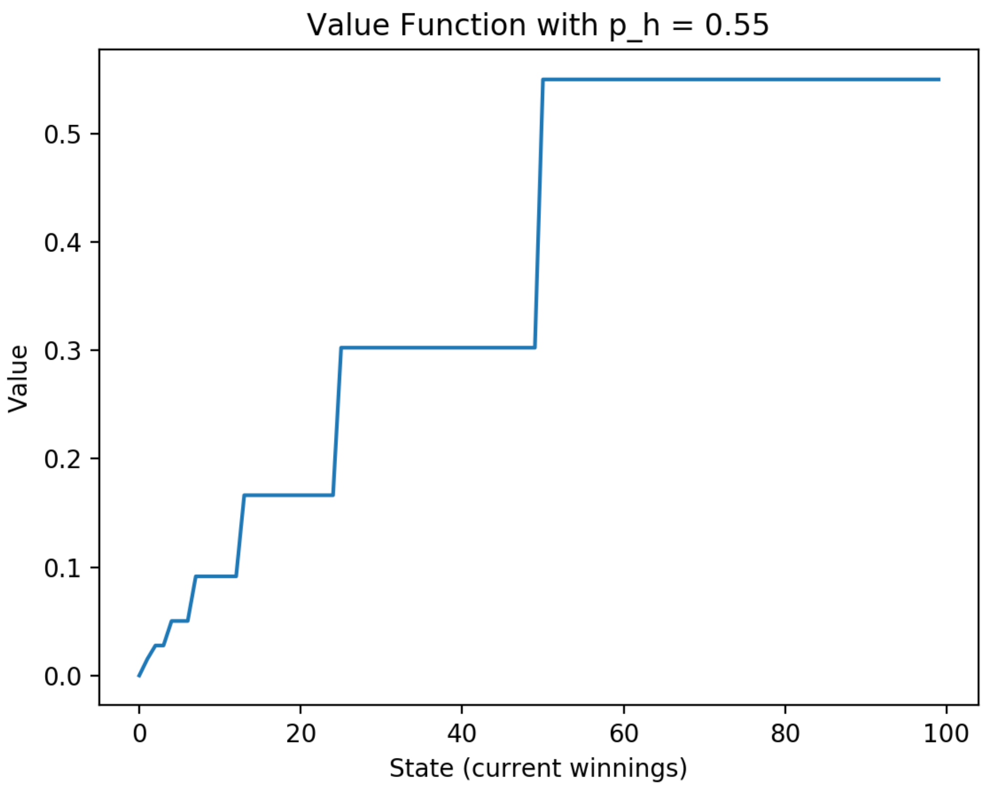
## 2 Methodology

To implement the value iteration, I used Python 3.6 with the external libraries numpy and matplot lib for storing and plotting the value function and optimal policy. I utilized a Python class to represent the gambler, which can be instantiated specifying probability for heads ($p_h$), theta threshold for convergence, discount factor, and number of states. This class utilizes member functions update_value to apply the Belman update equation, value_iteration to apply the algorithm, and plot to generate the plots. The value function is represented as an array with the indices corresponding to states, and the values at those indices corresponding to the expected value calculated through the value iteration. The optimal policy is then represented also as an array whose indices correlate to states, and who's values correlate to stakes to place in that state. The optimal policy is selected by choosing the maximum value for each state, and when there is more than one maximum values, the tie is decided by the first occurrence which in our case is the minimal stake. I do however change the way to tie is decided to see other examples of optimal policies generated by the value iteration.

## 3 Results and Analysis

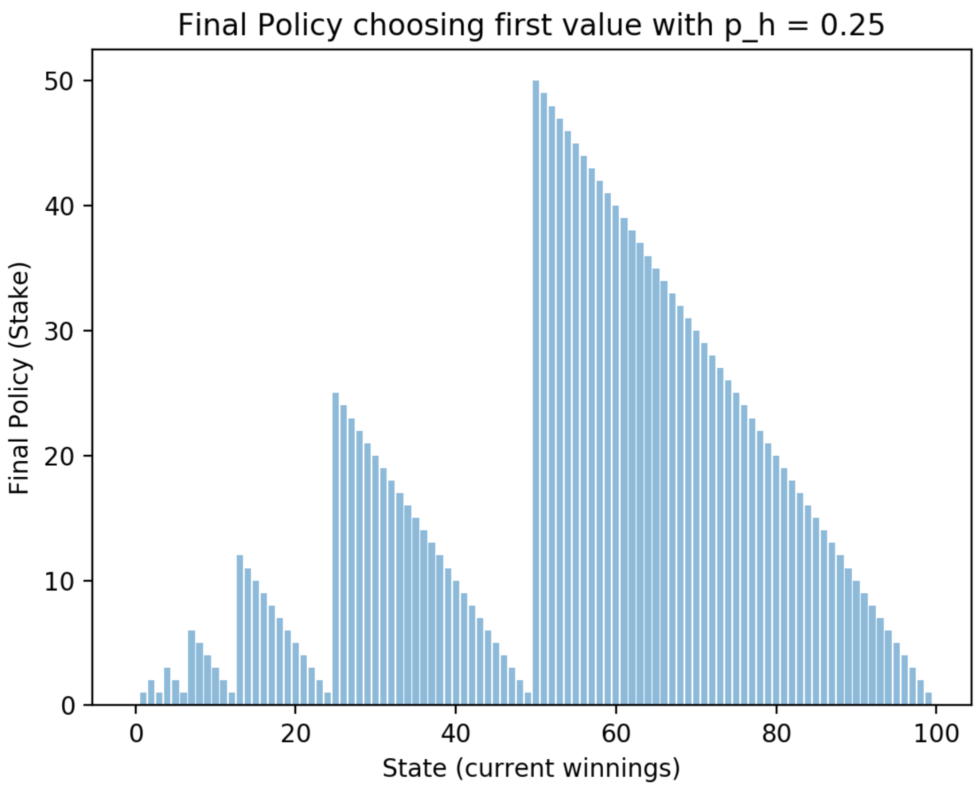
We begin our analysis with examining the value function. First we take a look at how the value function changes with a different probability of winning $p_h$

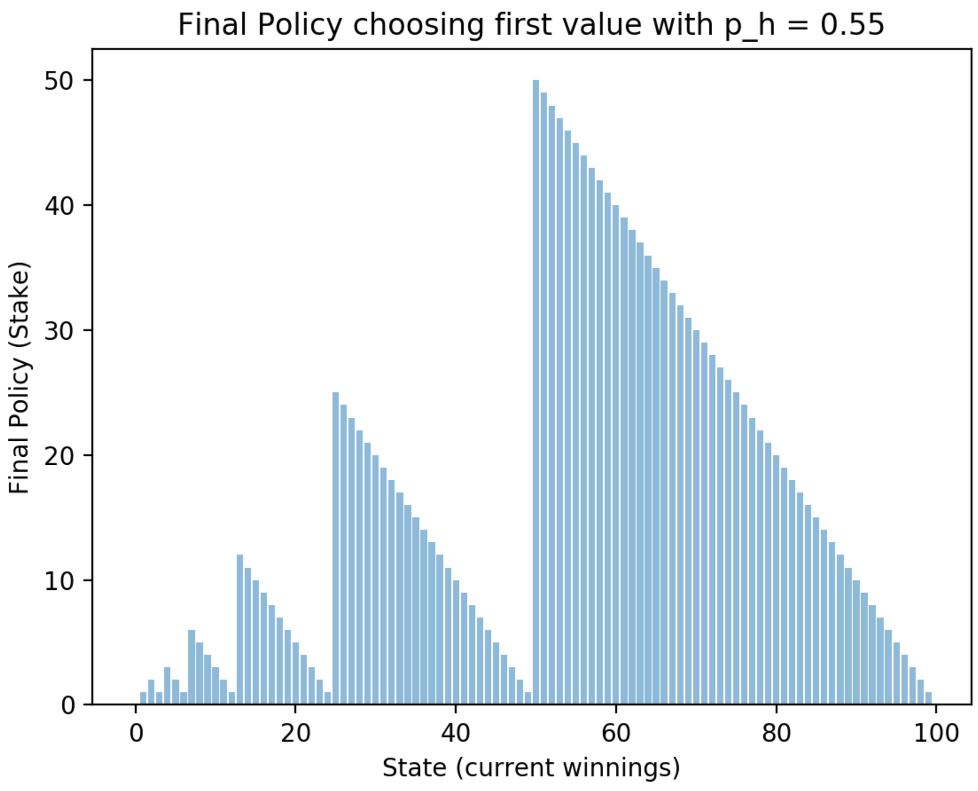




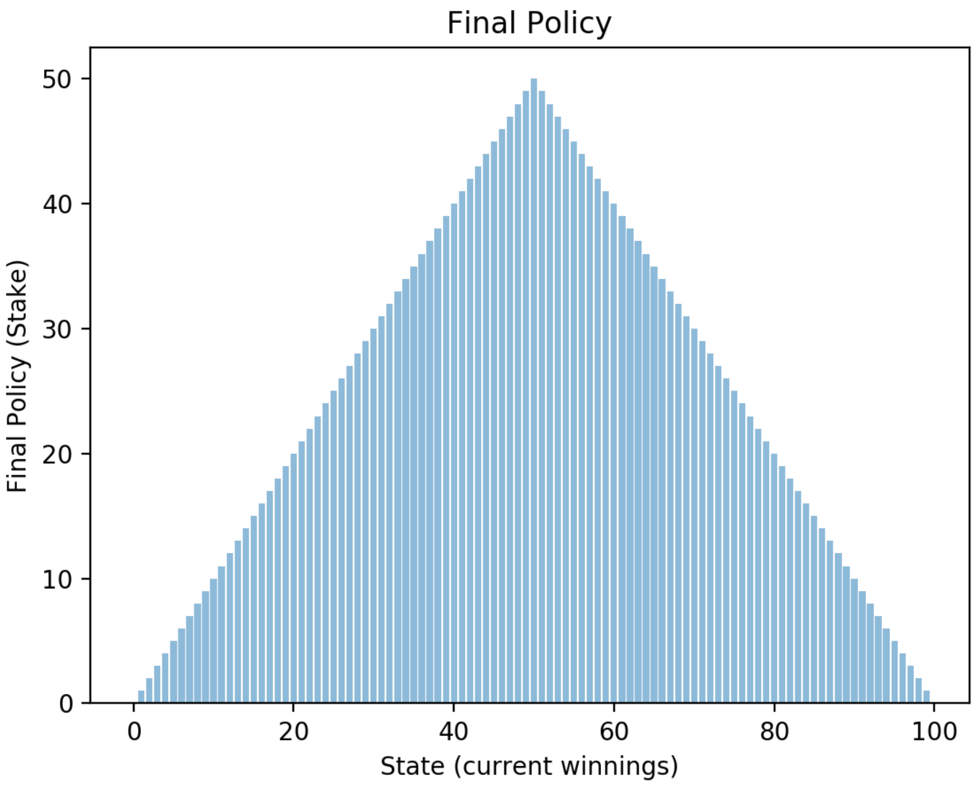
As we can see, the value function with a higher probability of winning reflects higher expected values for each state. This behavior is a result of the update rule, and is consistent with our understanding of the problem. Another interesting feature of these graphs is the spikes in value, occurring in the same states (12.5, 25, 50) for each graph. This is also consistent with the optimal policy represented in our textbook. These value functions will inform the optimal policy, and based on the graphs I hypothesize the optimal policy will be quite similar for the two different probabilities.

Initially plotting the optimal policy, I generated the following graphs. These graphs demonstrate the strategy of one optimal policy, which is to bet in order to reach certain increments (12.5, 25, 50), at which point you try to bet as much as you can to win.





As we hypothesized, the policies are similar: in fact they are identical. After generating these graphs, I took into consideration the ties in the value function, and my method of resolving these ties. Since these ties are resolved with the argmax function, by default this will return the first occurrence of the maximum. What this means for our analysis is that in the case that two or more stakes have the same expected value, our policy will choose the smaller stake. Now lets take a look at a different way to resolve this tie, instead of choosing the smallest stake, now we will choose the largest stake for our optimal policy. Our new graphs are as follows.



This policy reflects the "bold play" strategy mentioned earlier in the paper. Bet as much as you can to either win, or not eliminate you from the game. Both of these policies belong to the set of optimal policies, and our algorithm has managed to converge to optimally with delta = 0.0001. What is interesting from our findings is that the policy and winning probability are independent. Despite the chances to win, you will still utilize the same strategy for the gambler's problem. The winning chances only scale the value function.

## References

[1] Sutton & Barto (2017) *Reinforcement Learning: An Introduction*

[2] http://incompleteideas.net/book/gamblers.html

[3] https://www.maa.org/sites/default/files/pdf/joma/Volume8/Siegrist/RedBlack.pdf

[4] http://mi.eng.cam.ac.uk/ mg436/MEC_PythonCourse/Tutorial.html

[5] https://link.springer.com/content/pdf/10.1007%2F0-387-23152-8_63.pdf