Dylan Shadduck
EEC 289A
Spring 2021

Homework 2

Program Runtime: About 30 seconds

1. Introduction

The objective of this assignment is to find the state value for a given problem under a specific policy. The problem we consider in this assignment is one in which a player or agent is given a certain amount of money and is then asked to bet on the outcome of a coin flip. If the result of the coin flip is heads, the agent wins the bet and increases their total money. If the coin lands tails, the agent loses the bet and the total money they have is decreased by the amount bet. In our example we do not assume that the coin being flipped is fair. We will first make the assumption that this coin will land heads with a probability of 0.9. The game will end when the agent has no money left or the total money is equal to or exceeds $10. With this in mind we will evaluate the value of each possible state for three different policies.

2. The State Value Equation

The state value equation or the Bellman equation is an equation that serves to estimate the outcome of a certain state and action. This is done by considering a current state and then considering all the possible actions. From these actions there is a certain probability that determines the next state. In our example an action would be placing a bet and the next state would be determined by the outcome of the coin flip. The bellman equation puts all of these factors together and considers every possible combination of actions and states. This equation can be written as $v_{\pi}(s) = \sum_{a,s} \pi(a|s) \sum_{s',r} p(s', r|s, a)(r - \gamma v_{\pi}(s'))$. Where $v_{\pi}(s)$ is the state value at state s, $\pi(s|a)$ is the policy used to choose possible actions a, s' is the next state, r is

the reward, γ is a discount factor (set to one for this assignment because our task is episodic), and p(s', r|s, a) is the probability of a certain reward given a certain action and state.

3. Aggressive Policy

The aggressive policy is one in which the agent always bets all of their money. If the agent has $2, the agent bets $2. If the agent has $9 the agent bets $9. Given this policy, we know that the agent will always choose the action that maximizes the total profit or lose all the money. With this information we can write an algorithm to find the value for a certain state. For a given state we first check to see if we have reached the end of the game. If we have $0 or $10 or more we will return a value of zero since the game can no longer be played and there is no reward available. Otherwise, we say that the state value is equal to the probability of landing heads multiplied by the sum of the reward and the value of the next state. We also need to consider the case of losing, so we will add to our definition of the value the probability of the coin landing tails multiplied by the sum of the reward for losing.

This type of function that calls itself is known as a recursive function. These functions will tend to run forever unless there are certain conditions set in place that stop the recursion. In this case, assigning 0 to the value when the s' reaches $0 or $10 or more prevents infinite recursion. We will see in the next two policies that this isn't sufficient. After running this algorithm on each of the possible states 0$ - 10$ I found the following for a probability of heads landing 0.9.

1x11 double

	1	2	3	4	5	6	7	8	9	10	11
1	0	9.4976	9.6640	6.7200	8.9600	4	4.8000	5.6000	6.4000	7.2000	0

Figure 1: Aggressive Policy p(heads) = 0.9

When the probability of heads was changed to 0.1, the following state values were found using this algorithm.

1x11 double

	1	2	3	4	5	6	7	8	9	10	11
1	0	-0.9984	-1.9840	-2.8800	-3.8400	-4	-4.8000	-5.6000	-6.4000	-7.2000	0

Figure 2: Aggressive Policy p(heads) = 0.1

4. Conservative Policy

The conservative policy is similar to the aggressive policy in that there is only one action for each state. Under this policy we will always bet $1 regardless of how much money we have. The algorithm I implemented for this policy also involves recursion since the state value equation is the same as before, the only difference is in the action taken. Since the action taken is always betting one dollar, the state always moves either up or down by 1 dollar and therefore doesn't always exit the recursion if the agent loses the game. For this reason it is possible for our program to get stuck in infinite recursion. To prevent the program from timing out I set a parameter known as depth. This parameter tells the program how many actions in a row can be taken before we stop evaluating the value. If we think about this in terms of a decision tree, it's how many generations we are examining. For this algorithm I set a depth of 25 which I found to be sufficient for estimating the state values while still running in an acceptable amount of time. With a probability of heads of 0.9, I found the state values to be

1x11 double

	1	2	3	4	5	6	7	8	9	10	11
1	0	7.8882	7.8762	6.9861	5.9984	4.9998	4.0000	3.0000	2.0000	1.0000	0

Figure 3: Conservative Policy p(heads) = 0.9, depth = 25

Running this algorithm again with a heads probability of 0.1 gives the following state values

1x11 double

	1	2	3	4	5	6	7	8	9	10	11
1	0	-1.0000	-2.0000	-3.0000	-4.0000	-4.9998	-5.9984	-6.9861	-7.8762	-7.8882	0

Figure 4: Conservative Policy p(heads) = 0.1, depth = 25

5. Random Policy

In the random policy we now have a variety of possible actions based on the amount of money we currently have. We will bet a random integer amount of money from $1 up to our total amount of money. The probability of betting each amount follows a uniform distribution so that each possible bet has the same probability. Now that we have more than one action, we have a higher degree of complexity to the algorithm structure we have been using. We will still make use of the recursion exit conditions mentioned in the above algorithms with the min and max amount of money and the recursion depth. The difference in this algorithm is that we can take multiple actions. I start by setting the state value equal to the probability of heads multiplied by the probability of choosing this action multiplied by the sum of the reward and the value of the next state given that we bet one dollar. Again we will also need to include a term for the negative reward where our agent loses the game and loses money. Once this has been done, I continue adding more terms for the other actions in the same fashion. These other actions are added by using a for loop within the conditional statement.

Now that there are much more actions to choose from, there are a lot more possible state/action combinations and therefore we need to reduce our depth in order to preserve run time. I found that a depth of around 11 was suitable for this policy algorithm. The state values for this policy with a heads probability of 0.9 were found to be

1x11 double

	1	2	3	4	5	6	7	8	9	10	11
1	0	8.5525	8.6618	8.1099	7.5823	6.6512	5.6128	5.0435	4.7430	4.6083	0

Figure 5: Random Policy p(heads) = 0.9, depth = 11

Setting the probability of heads to 0.1 gives the following state values

1x11 double

	1	2	3	4	5	6	7	8	9	10	11
1	0	-0.9993	-1.9927	-2.9603	-3.9021	-4.6751	-5.3086	-5.9477	-6.5908	-7.2371	0

Figure 6: Random Policy p(heads) = 0.1, depth = 11

6. Conclusion

This assignment didn't come from the textbook, so there is no comparison between my work and the textbook to show. I'd like to reiterate that the values found by the conservative and random methods are only estimates because it is not possible to run an infinitely recurring function on my machine (or any other for that matter).

The array values shown in all of the figures follow the same format. Index one shows the state value when the initial state is $0. The second index is when the initial state is $1 and so on.