Dylan Shadduck
EEC 289A Spring 2021

Homework 4

Matlab Code Runtime: About 2 minutes

1. Introduction

The goal of this assignment is to implement the Monte Carlo Exploring Start method for determining the optimal policy for the game of Black Jack. This is a common betting game found at many casinos and has some fairly simple rules.

The goal of Black Jack is to get closest to 21 points without going over. Points are awarded based on the value of the cards in your hand. Each card is worth the value shown on the card except for the Ace and face cards (Jack, Queen, King). All of these cards are worth 10 points except for the Ace. The value of the Ace can vary depending on what benefits the player more. The Ace can be worth 11 points, but if the total score of the cards will cause the player to exceed 21 points, the Ace can also be counted as 1 point. If the player has multiple Aces, they can all be set to either value independently of each other.

The game starts with both the player and the dealer being dealt two cards from a deck. The player has full knowledge of both cards in their hand, but can only see one face up card of the dealer. The player can then add cards to their hand (and their score). This action is known as a “hit”. When the player has reached a score that they would like to stop at this is known as a “stick”. If the player draws a card that causes their score to exceed 21 points, the player is said to have “bust”.

Once the player takes their turn, the dealer must then take their turn. The dealer must “hit” whenever their score is 16 points or less. In this assignment we consider two different cases for the dealer when they have a score of 17. In the first case, if the dealer has 17 points they must "stick". In the second case, if the dealer has 17 points and one or more of their cards is an ace, they must "hit". Otherwise, the dealer must "stick" on a 17.

2. Monte Carlo ES Method

Given the rules discussed above, a Monte Carlo Exploring Starts simulation was run to determine the optimal policy under the two different conditions outlined above for the dealer with a score of 17.

To start this method we must first initialize our policy for each possible state. The states we will consider are the dealer's face up card, the player's score, and whether or not the player has an Ace in their hand. With these three conditions, this gives a total of 200 different states. We can initialize the policy array as a three dimensional array (for the three conditions) with random values of either 0 or 1. A "0" represents "stick" and the "1" represents "hit". Next we need to initialize our Q matrix. This matrix will need to add two more dimensions to the three discussed for the policy matrix. The reason for this is that we must consider the value for "stick" and "hit" for each of the 200 states. On top of this we need to keep track of how many times a certain state and action has been taken, so we need a 5th dimension to store the value q and the number n of the times the state/action pair has been chosen.

Once these matrices have been set, we are ready to start simulating Black Jack. We start by defining the deck of cards to choose from. I made a 1 dimensional array of integers. All of the face cards were represented by their value, 10, and the Ace is represented by its maximum value, 11. We start an episode by drawing 2 cards for both the player and the dealer. We consider that the deck is infinite, so drawing does not remove any cards from the deck.

Next we check the score of the player. If the combined score of the player's cards are not greater than or equal to 12, we "hit". Once the player has 12 points or more, we take a random action, either “hit” or “stick”. This random action and random initial state satisfies the “exploring starts” part of this Monte Carlo method. From this point, we continue to take actions based on the policy matrix we constructed at the beginning. Once the player has declared “stick” or exceeds 21 points, it is the dealer’s turn. If the player goes over 21, or busts, the game is over and the player loses. If the player does not “bust”, the dealer must “hit” until their score is greater than or equal to 17.

Once both the player and dealer have completed their drawing of cards into their respective hands, it is time to tally up the score and find the reward. If the player “busts”, the reward is -1. If the dealer “busts” and the player does not, the player wins and sees a reward of 1. If neither bust and the player has a higher score, the reward is 1. If neither bust and the dealer has a higher score, the reward is -1. In all other cases, the player and the dealer will have the same score and will tie. In this case the reward is 0.

Now that we have our reward, it is time to update the q function for each state reached during this episode. This is done by looping through all the cards in the player’s hand and checking the state at this position. If this is the starting position, then we make sure to include the random action taken at the beginning of this episode. From here we update the q function for this state action pair according to the following equation

$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$. Here R_n is the reward we calculated for the episode and, n is the number of times this state/action was reached, and Q_n is the current value for this state/action pair. Once this value is updated, we make sure to increment the value of n.

The final step is to update our optimal action. We do this by checking the state and comparing Q_n for both possible actions. If Q_n for the “hit” action is greater than for the “stick”, we set the policy for this state to “hit”. Otherwise we set the policy to “stick”.

This process is repeated for 5,000,000 episodes so that we have a reasonable estimate for Q for each possible state and action pair. Once this is complete we move on to plotting the results of this MC simulation. After the plotting is done, we will do another MC simulation with one change. This simulation will consider the case that the dealer must hit on a "soft 17". This means that if the dealer has a score of 17 and a usable Ace, the dealer must "hit". Otherwise the dealer will "stick" on a 17 and above.

3. Plotting Figures

Plotting is a fairly straightforward process. The figures required for this assignment are the optimal policy for the 200 states and the corresponding value for these optimal policies. These policies are represented as 2 10 by 10 matrices with the card shown by the dealer on the x axis and the player score on the y axis. To plot the "hit" or "stick" action, I used the pcolor plot from matlab. This results in yellow squares for "hit" and blue squares for "stick". The policy is plotted for two cases. Whether or not the player has a usable Ace. To accompany these plots we also have their corresponding values. I compiled these values from the Q matrix and placed them in a separate array that I called V. Then all I needed to do is plot V using the surf plotting method from matlab. This is a three dimensional plot with the x and y axis the same as the policy plot and the z axis is the value for the state/action pair.

4. Results

Plotting was done for both cases of the dealer "hitting" or "sticking" on a "soft 17". Despite the change in the game rules, I found that the optimal policy did not change between these two cases. The results of case 1, "stick on soft 17" are shown below.

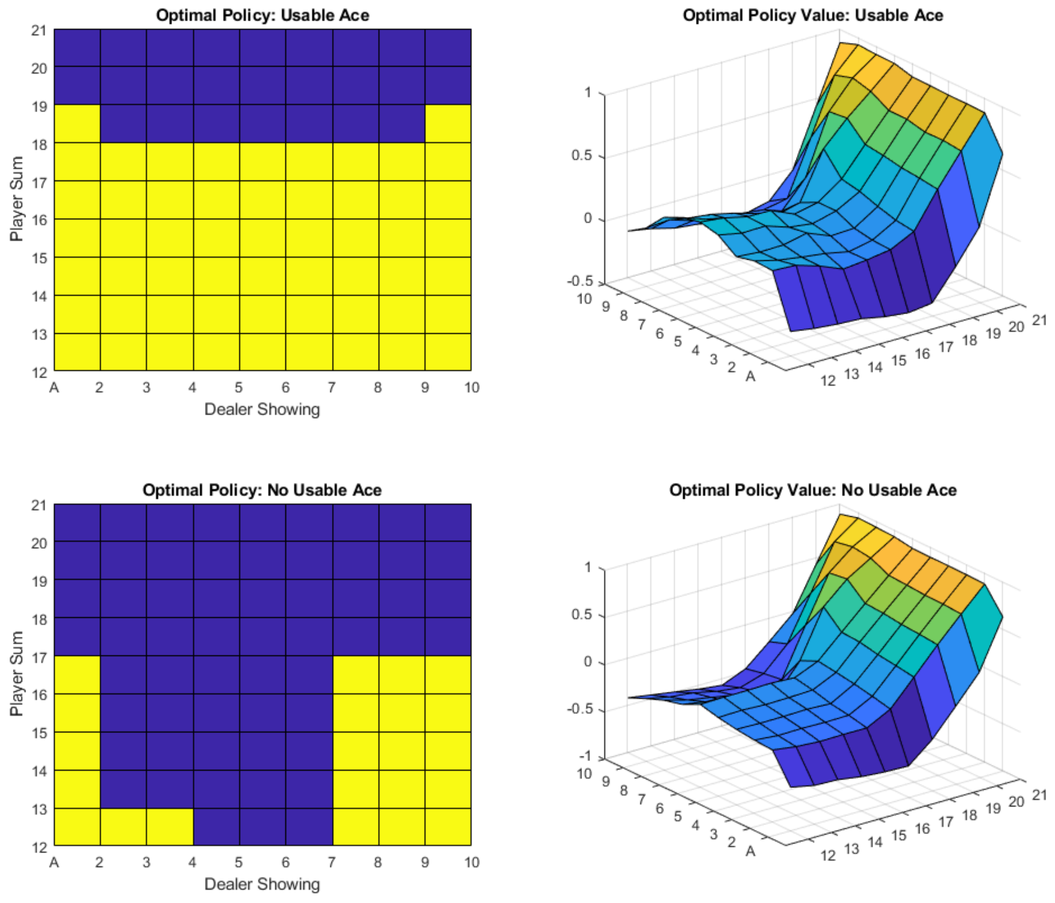

Figure 1: Optimal Policy and Value Function (Stick Soft 17)

In the figure, the yellow squares represent a “hit” and the blue represents a “stick”. After this simulation I ran the second case when the dealer must “hit” on a soft 17. The results of this simulation are shown in figure 2 below.

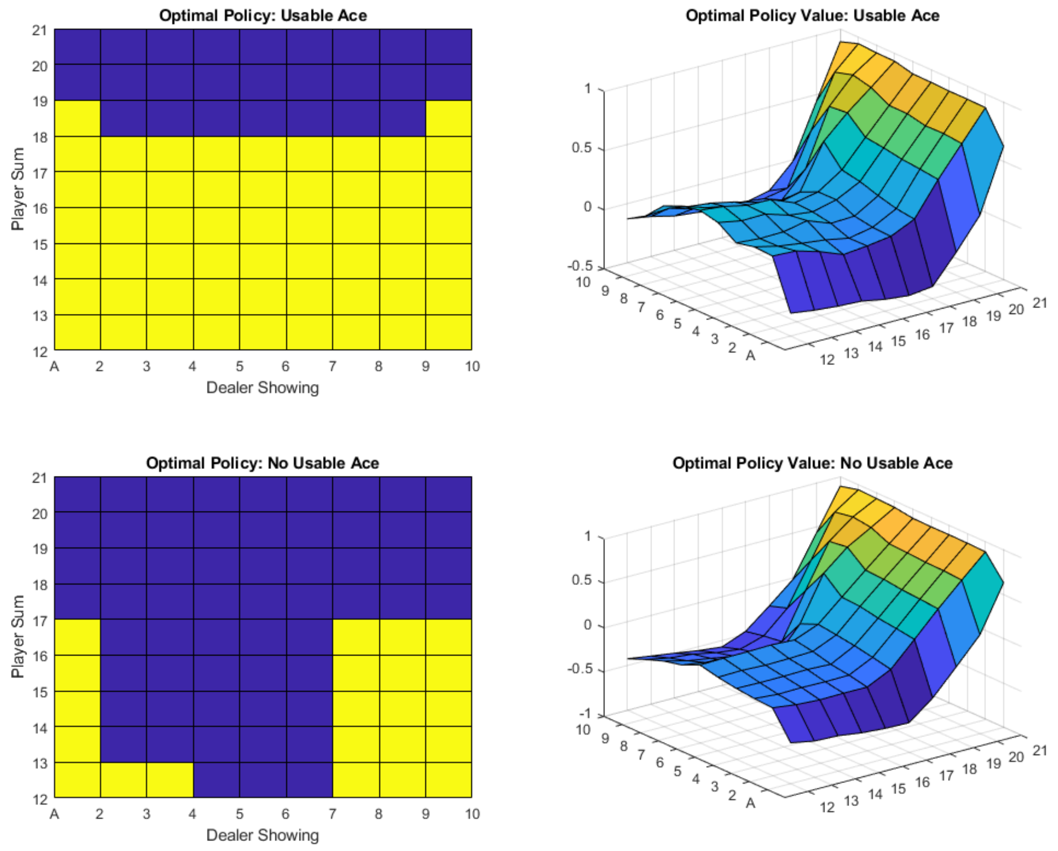

Figure 2: Optimal Policy and Value Function (Hit Soft 17)

As can be seen from the comparison of figures 1 and 2, we see an identical policy and nearly identical value function. This suggests that the policy of the dealer on a soft 17 has very little impact on the policy taken by the player.

Comparing our results with figure 5.2 from the textbook, I see almost no difference. I would take this to mean that I performed both MC simulations correctly and found the optimal policy and corresponding state values.

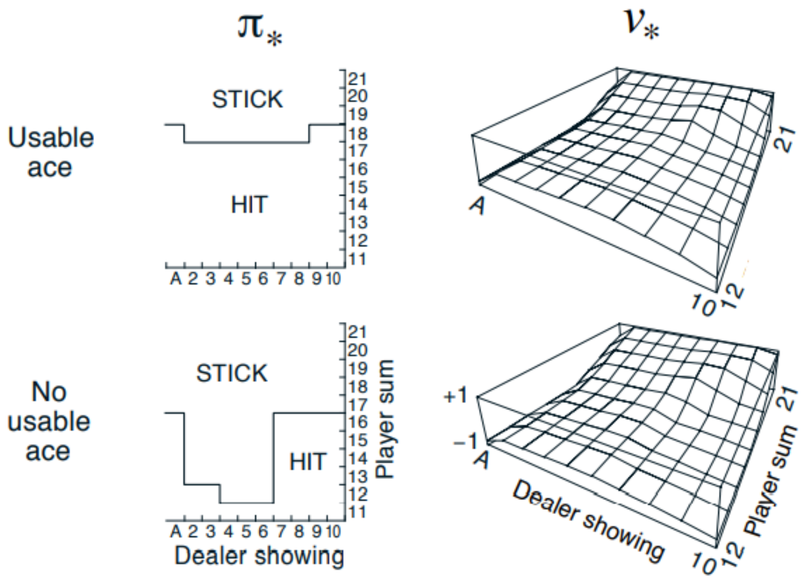

Figure 5.2: The optimal policy and state-value function for blackjack, found by Monte Carlo ES. The state-value function shown was computed from the action-value function found by Monte Carlo ES. ■