Wet2 - 046203

Zohar Rimon

17.06.2020

1 Black Jack

1.1

We will define the game as a markov decision process. We will define each state as the sum of the cards of the player (X) and the value of

the first dealers card (Y) (When its the dealer turn there are no more actions to be taken so those states are not important. There are 10 states for the dealer card value, 17 Player sum values(that are lower to 21) and 1 terminal state. The number of states:

$$|s| = 17 * 10 + 1 = 171$$

We don't care about the different terminal states because the player can't take any action in those states. Furthermore, we will model the reward function such that we receive a stochastic reward in each state based on the action taken and in the terminal state we don't receive a reward. This is fine because the value function at the different terminal states are is trivial.

The action space (A) is to take a card (hit = 1) or to not take a card (stick = 0).

The reward is stochastic, we will calculate the expectancy of it grammatically for the value iteration algorithm.

1.2

Like we mentioned in the previous subsection, we will not plot the terminal states, due to the fact that the value function there is trivial (just the reward in each of those states).

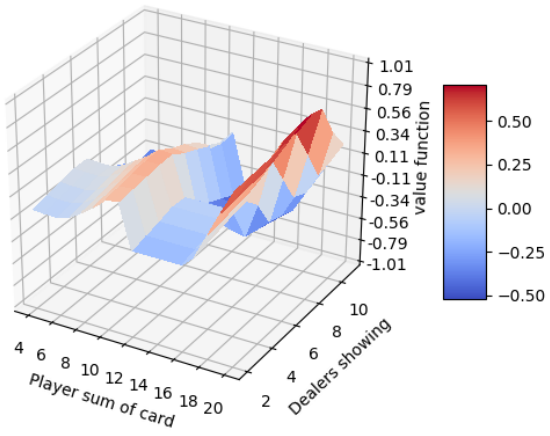

Figure 1: The value function for the different states of the game

We can see that the maximum value is achieved when the player has cards that sums up to 20. In those states (as we will see clearly in the next subsection), the player will stick and will have a very large chance of winning. Another big value group of states are the ones where the player has cards that sums up to 10. In those states the player will hit, and will have a large chance of getting a 10 (we have four cards with a value of 10) and then will stick.

1.3

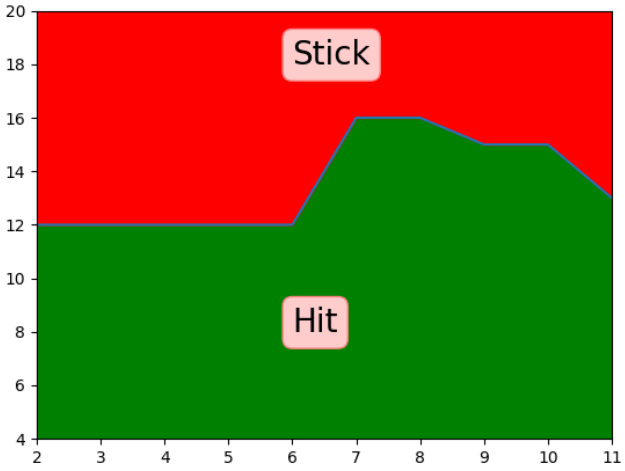

Figure 2: The optimal policy

We can see some interesting things from Figure 2, such as:

- For a hand greater or equal to 16, we will always stick.
- For a hand lower than 12 we will always hit
- For a Dealer showing of around 7, we prefer to hit more than other states. This is due to the fact that the dealer has a higher chance of getting close to 21 in this states.

2 The $c\mu$ rule revisited

2.1

The states are the jobs left and the action is which job to take. Thus, the number of states is the number of subsets in the set of jobs, and the number of actions is the number of total jobs:

$$|S| = 2^N = 2^5 = 32, \quad |A| = N = 5$$

We will order the states as binary code, where the state ”00000” corresponds to the state which no job is left and ”00011” corresponds to the state which only the fourth and the fifth jobs left.

2.2

CODE: Solved using iterative solution, rewards = -costs

2.3

The policy π_c and its value function:

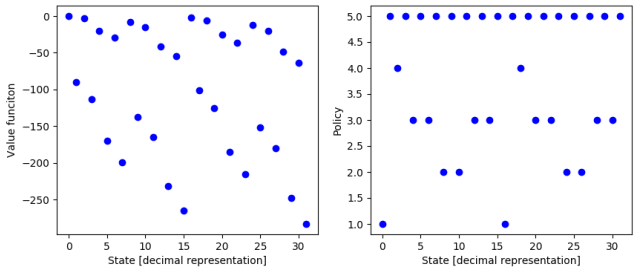

Figure 3: The ”cost only” policy and its value function

We can see for example that the state with the lowest value function is (obviously) s_0 - the state where all jobs are not finished.

2.4

The value function at s_0 during policy iteration, and starting from π_c:

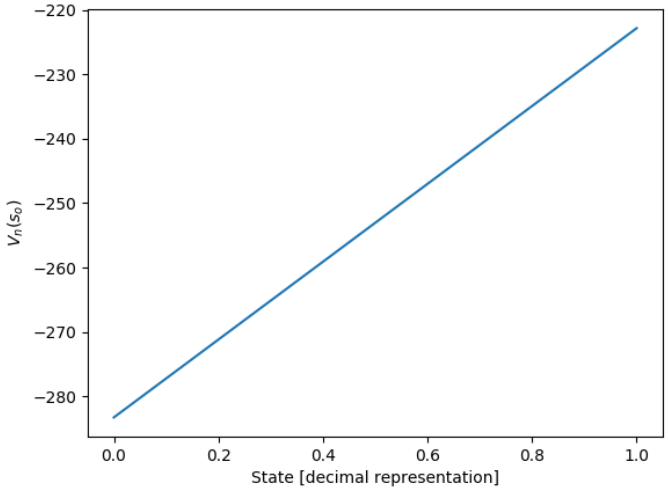

Figure 4: The value function at state s_0 during policy iteration

Reached convergence after only one step

2.5

After comparing the optimal policy (achieved from the policy iteration algorithm) to the $c\mu$ policy, we can see that the two policies are identical. This means, that our result from the previous HW is correct and the $c\mu$ policy is optimal.

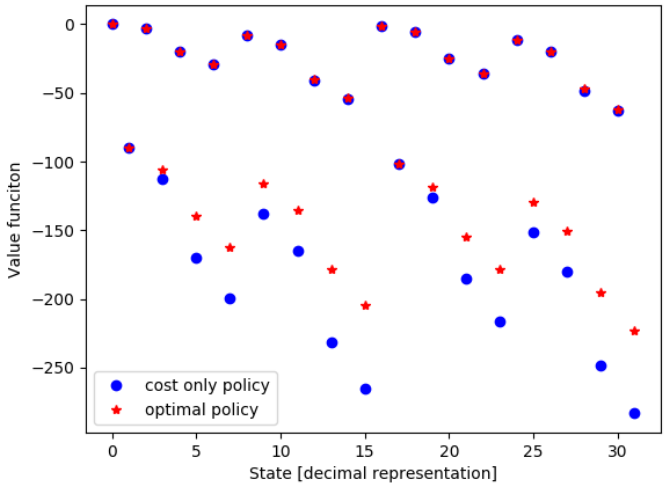

Figure 5: The value function of the cost only policy vs the value function of the optimal policy

We can see in figure 5, that indeed, the optimal policy is greater than the cost only policy for every state.

2.6

CODE

2.7

In order to get the entire value function to converge, and not only the value at s_0, I had to start each iteration of the algorithm at a random state, that allowed the algorithm to "see" each state.

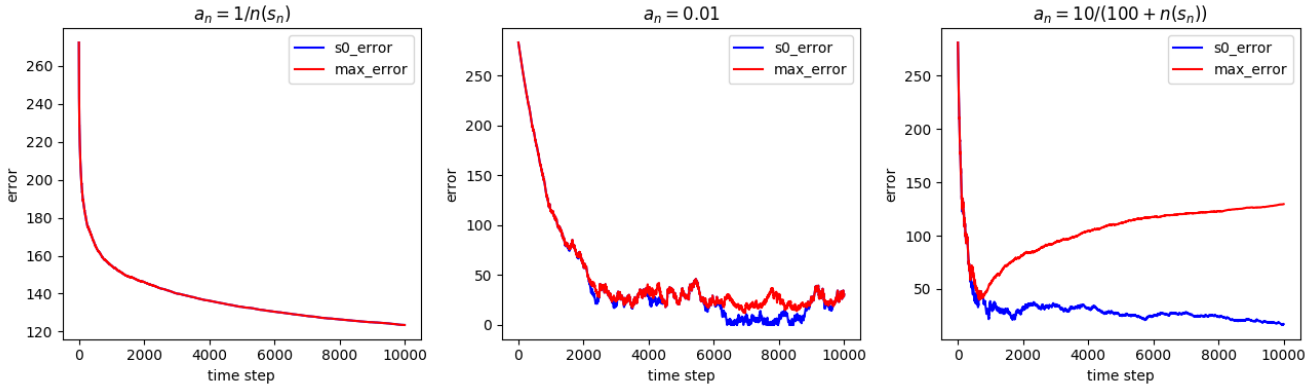

Figure 6: The error of the value function of the cost only policy over the iterations of TD(0)

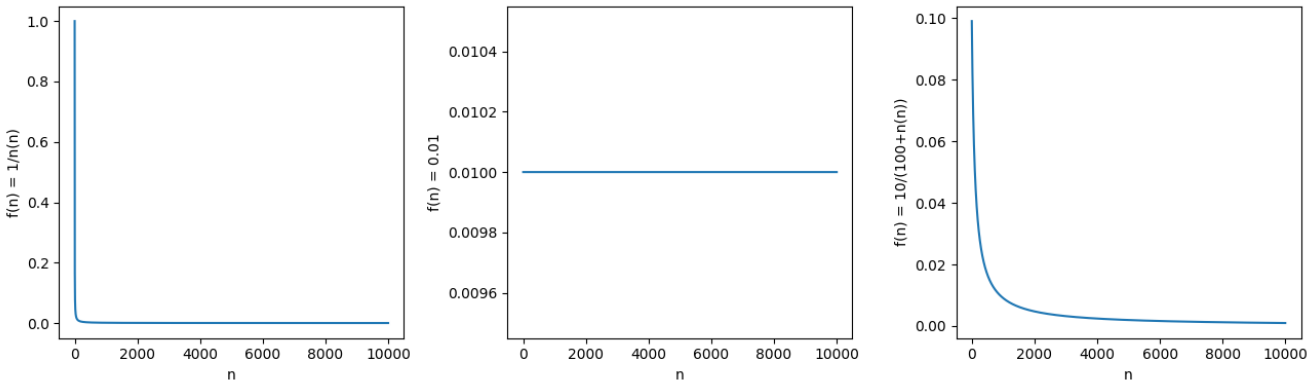

Figure 7: The functions used in each of the time steps

The motivation of the different step sizes:

- $a_n = \frac{1}{N(s_n)}$: The step size decreases very quickly, thus after a few iterations we can see that the value function converges, but not on the real value.
- $a_n = 0.01$: In this case we have solved the issue from the previous option. Now the step size does not decrease at all and we can see that there is no constant error, But in this case we can also see that due to the fact that the step size does not decrease at all, there is a noise, even for high number of iterations, that is because we are never stopping the update of the value function.
- $a_n = \frac{10}{100+N(s_n)}$: As we can see in figure 7, the third option is a function with a much heavier tail than the first one. We can see from that two phenomenons. First, we can see that there is convergence at large number of iterations. Second, due to the heavy tail, we that the maximum error has some noise in the beginning of the run and might get stuck at a high error (this changes between runs)

2.8

Results using the first option of the step size:

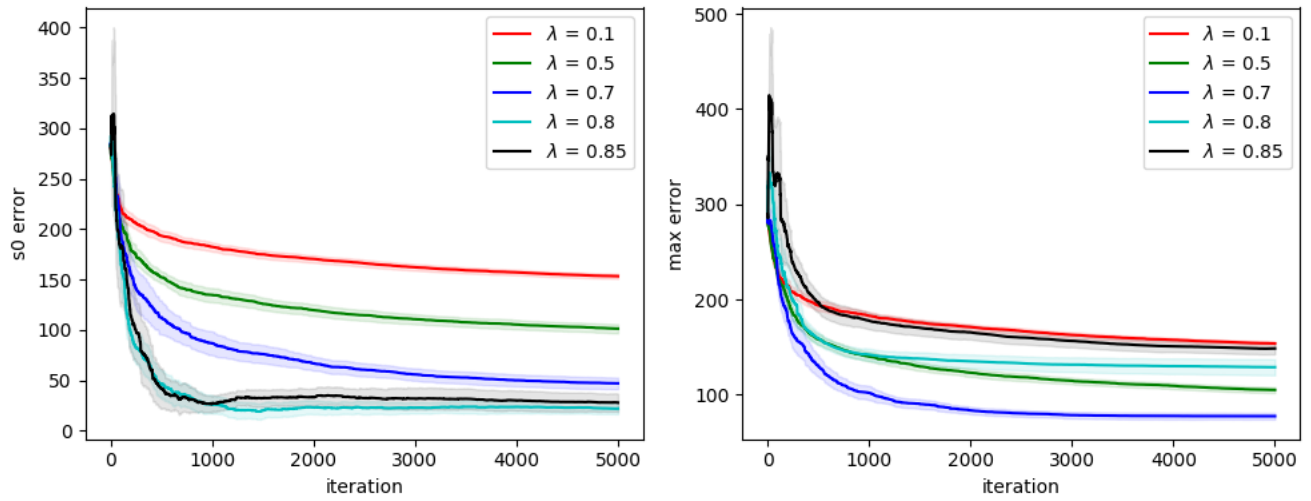

Figure 8: The max error and the s_0 during the $TD(\lambda)$ algorithm with different lambdas. Also the 0.95 confidence interval, based on the 20 runs, is also present

2.9

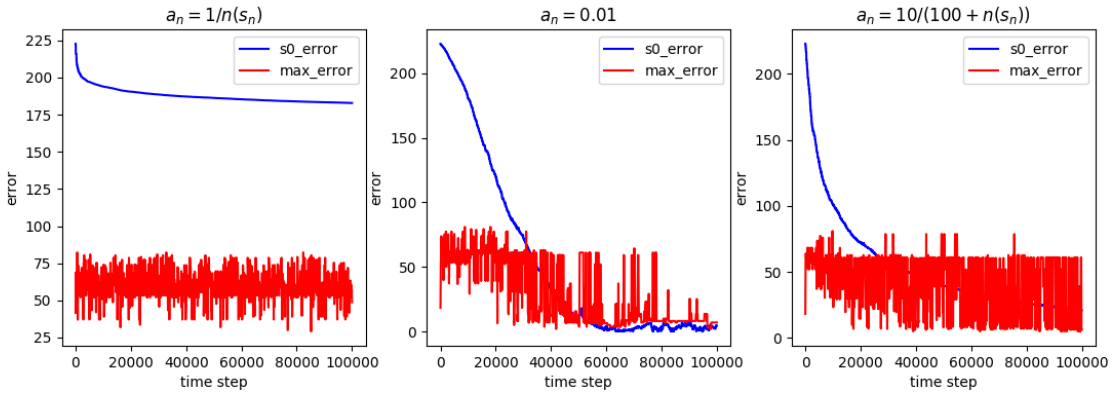

Figure 9: The Q learning error

We can see that the error with respect to the greedy policy is very noisy, this makes sense due to the argmax operator that we used in order to calculate the greedy policy. This will result in a jump in the value function every time this argmax operator will change value. and this operator is not continues, resulting in a noisy error function.

2.10

Results using the second option of the step size:

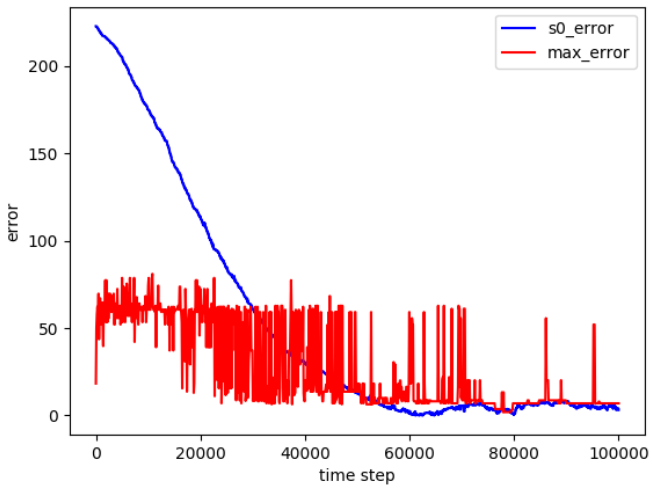

Figure 10: The Q learning error for $\epsilon = 0.01$ and with $a_n = 0.01$

We can see that there is not much of a difference in this case. This is probably a result of the fact that our state space is very small and a very small exploration rate is sufficient in order to visit the entire state space.