INFO8003-1 Optimal decision making for complex problems

PART 1 – MOTIVATION

FEBRUARY 2020

s152764 HOCKERS Pierre – s150986 MIFTARI Bardhyl

1 General structure of the code

In the following report, the basic chosen policy is to move always right. All the other basic movements (always move left, up,down) and a uniform random policy can be called for section 2 and section 3. Three different classes are used. The agent class contains all the data needed to go from one state to another and information about the current state. This where the choser policy is called, too. The grid class contains the environment in which the agent is. The game class is the manager code for both classes and collects data about what happened during the game.

2 Section 2

The trajectory of the always go right policy is given by :

```
(((3.0, 0.0), 'RIGHT'), -9)
(((3.0, 1.0), 'RIGHT'), 3.96)
(((3.0, 2.0), 'RIGHT'), 18.6219)
(((3.0, 3.0), 'RIGHT'), -4.851495)
(((3.0, 4.0), 'RIGHT'), -4.8029800499999995)
(((3.0, 4.0), 'RIGHT'), -4.754950249499999)
(((3.0, 4.0), 'RIGHT'), -4.707400747004999)
(((3.0, 4.0), 'RIGHT'), -4.660326739534949)
(((3.0, 4.0), 'RIGHT'), -4.6137234721396)
(((3.0, 4.0), 'RIGHT'), -4.567586237418204)
```

(a) Deterministic trajectory

```
(((3.0, 0.0), 'RIGHT'), -3)
(((0.0, 0.0), 'RIGHT'), 0.99)
(((0.0, 1.0), 'RIGHT'), -2.9402999999999997)
(((0.0, 0.0), 'RIGHT'), -2.910897)
(((0.0, 0.0), 'RIGHT'), 0.96059601)
(((0.0, 1.0), 'RIGHT'), -2.8529701496999995)
(((0.0, 0.0), 'RIGHT'), -2.8244404482029997)
(((0.0, 0.0), 'RIGHT'), -2.7961960437209696)
(((0.0, 0.0), 'RIGHT'), 0.92274469442792)
(((0.0, 1.0), 'RIGHT'), -4.567586237418204)
```

(b) Stochastic trajectory

Figure 1 – Evolution of the trajectory for 10 steps starting at position (3,0) in deterministic and stochastic case ($\beta = 0.5$). Trajectory contains the position before the move, the chosen move and the reward it got after the move

It can be seen that as expected in the deterministic case, the agent moves to the right and stays in the position (3,4) as he can not go further right. Whereas in the stochastic case, a probability of 0.5 sends the agent to position (0,0) when it wanted to go right.

3 Section 3

Similarly to section 2, we used an always right policy. In this section, the agents reward is simulated 10 times for 1000 movements and the mean of this result is plotted for each starting position. We chose our number of timesteps (1000) arbitrarily at first, but quickly noticed it was a correct amount of steps to consider, as it allowed us to see the convergences of the scores of the diverses starting positions. As can be seen in Figure 2 and 3, N = 500 could also have been a good choice. Before 500, there are a lot of oscillations still and after that the rewards does not evolve anymore.

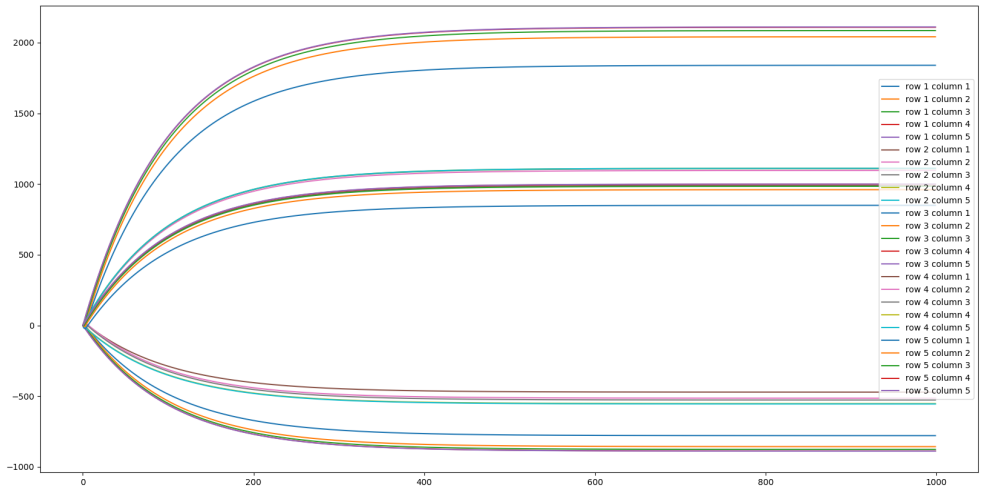

Figure 2 – Deterministic evolution of the rewards over 1000 timesteps for each starting position

In Figure 2, it can be seen that for a deterministic always right policy the column at which the agent starts has a little influence on the expected final reward received as the agent will stay in column 4 once it is there and the rewards to get from the initial state to the column 4 are also different. Whereas, the row has a huge influence on the expected final reward as we can clearly see. If another policy such as always up was used the column would have a huge influence on the final result and the influence of the row would be lesser.

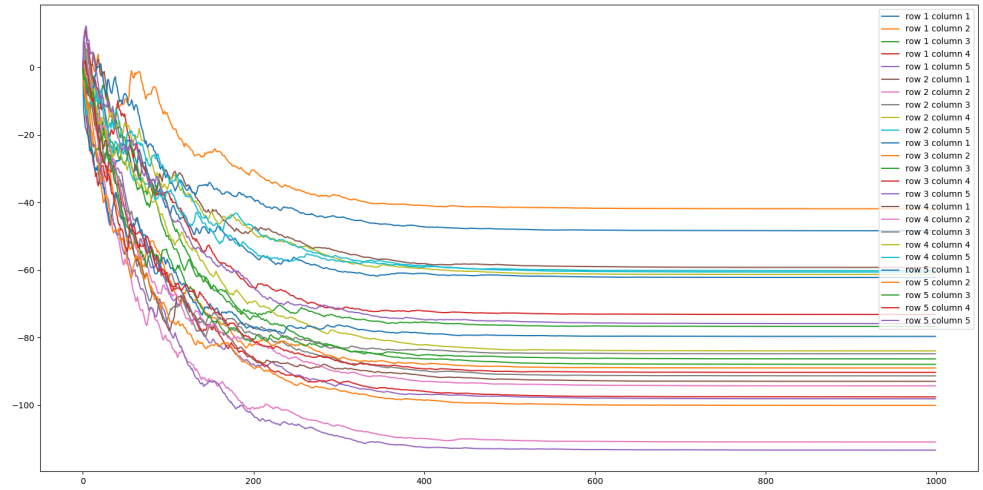

Figure 3 – Stochastic evolution of the rewards over 1000 timesteps for each starting position

In the stochastic case, the starting position does not influence much the expected final reward. For each action that is taken, we have a probability of $\beta = 0.5$ to end up in state (0,0). As we have, $\sum_{n=1}^{\inf}(\frac{1}{2})^n = 1$ the agent will always end up in the (0,0) position which explain why in this case the initial starting position is not that important.

4 Section 4

For the deterministic case, $P(x'|x,u) = 1$ if the action u is such that it takes the agent from state x to x' and is equal to zero otherwise. When it comes to the reward r(x,u), in this section, we consider that the rewards in the grid are known so r(x,u) is already determined. Q_n can simply be determined by using :

$$Q_n(x,u) = r(x,u) + \gamma * max_{u' \in U} Q(f(x,u,w),u')$$

For a given x, we compute $Q_n(x,u)$ for each possible u. We then chose the u that gave the best value as our action. This is supposed to make us chose the action that yields the best expected score over N iterations of the game.

We have noticed that increasing the size of N doesn't give us better result. To get both good results and fairly quick execution time we decided to go with a fixed value of N = 3.

For the stochastic case,

$$P(x'|x,u) = \begin{cases} \beta & \text{for all x} \in \text{X/(0,0) and x' = (0,0)} \\ 1 & \text{if x' = x + u and x' = (0,0)} \\ 1-\beta & \text{if x + u = x'} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$r(x,u) = \mathbb{E}_w[r(x,u,w)]$$

In this particular case, the reward associated for a particular movement starting on a given state x will also depend on β.

We observed with this implementation that the score rises consistently throughout the game, as expected. It is of course far higher on average if the game is set on stochastic, as it doesn't have a 50% chance to go back to the (0,0) square that has a negative reward. Scores range between 100 and 200 for a stochastic game, while they reach as high as 2000 in deterministic games.

When launching the code, you will get on the terminal the expected scores at the end of the game for every starting position. These scores are computer by averaging 5 games, with a N value of 3 for the Q infered policy.