Student No. 756011
Zeinab R. Yousefi

Assignment 2

ELEC-E8125
October 13, 2019

1 Question 1 What is the agent and the environment in this setup?

The agent is the sailor (or the ship in a higher level) and the environment is the introduced gridworld. Environment is actually anything that cannot be changed arbitrarily by the agent, therefore the sea, rocks, wind are considered as the environment.

2 Task 1 and 2: Implement value iteration for 100 iterations

The implementation of value iteration for 100 times with discount factor of = 0.9 is shown in the algorithm 1. The state values and policy are saved in numpy array "values" and "policy" (attached to the submission files).

Algorithm 1: Run for 100 iterations

```
# Reset the environment
state = env.reset()

# Compute state values and the policy
value_est, policy = np.zeros((env.w, env.h)), np.zeros((env.w, env.h))

n_actions = env.transitions.shape[2]
gamma = 0.9
Q = np.zeros((env.w, env.h, n_actions))
for _ in range(100):
    for i in range(env.w):
        for j in range(env.h): #-1,-1,-1):
            for a in range(n_actions):
                value_s_primes = 0
                for s in range(len(env.transitions[i, j, a])):
                    s_prime = env.transitions[i, j, a][s][0]
                    r = env.transitions[i, j, a][s][1]
                    tr = env.transitions[i, j, a][s][3]
                    if s_prime is not None:
                        value_s_primes += tr * (r + gamma*value_est[s_prime[0], s_prime
                Q[i, j, a] = value_s_primes
            value_est[i, j] = np.max(Q[i, j])
    policy = np.argmax(Q, axis=2)
    env.clear_text
```

Question 2 What is the state value of the harbour and rock states? Why?

State values of the harbour and rock states are 0 because they are terminal states. The value of a state is the expected sum of all future rewards when starting in that state and following

a specific policy. For the terminal state, this is zero - there are no more rewards to be had.

Question 3 Which path did the sailor choose? If you change the reward for hitting the rocks to -10 (that is, make the sailor value his life more), does he still choose the same path?

According to the obtained policy, as it can be seen in the figure 1 the sailor goes to right through the windy passage to reach to the harbour. If I change the rock penalty to -10, the sailor choose another path that bypass the windy passage and goes down at first. Figure 2 shows the policy related to rock penalty of -10.

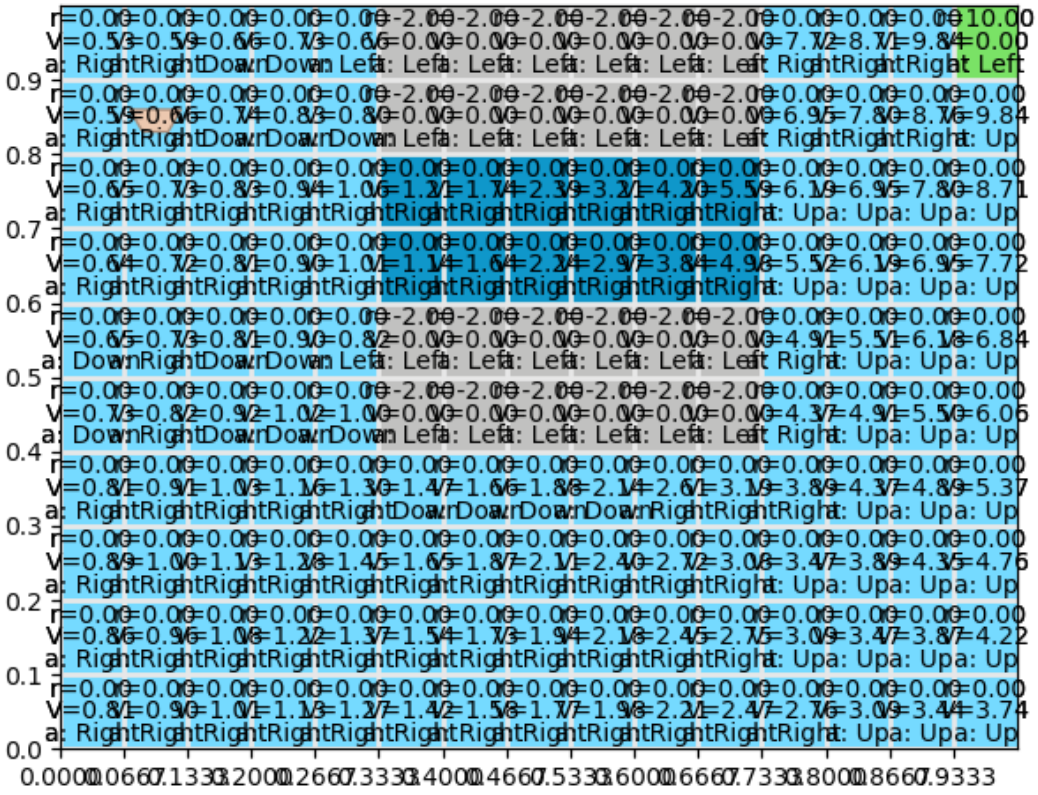

Figure 1: Reward, values and policy, when rock penalty is -2

Question 4 What happens if you run the algorithm for a smaller amount of iterations? Do the value function and policy still converge? Which of them - the policy or value function - needs less iterations to converge, if any? Justify your answer.

I run the algorithm for different amount of iterations, it seems that the value function converges after 31 iterations and policy function converges after 24 iterations. Usually policy function needs less iteration to converge. For the policy to converge it is sufficient that the relations between the values are correct.

Assignment 2

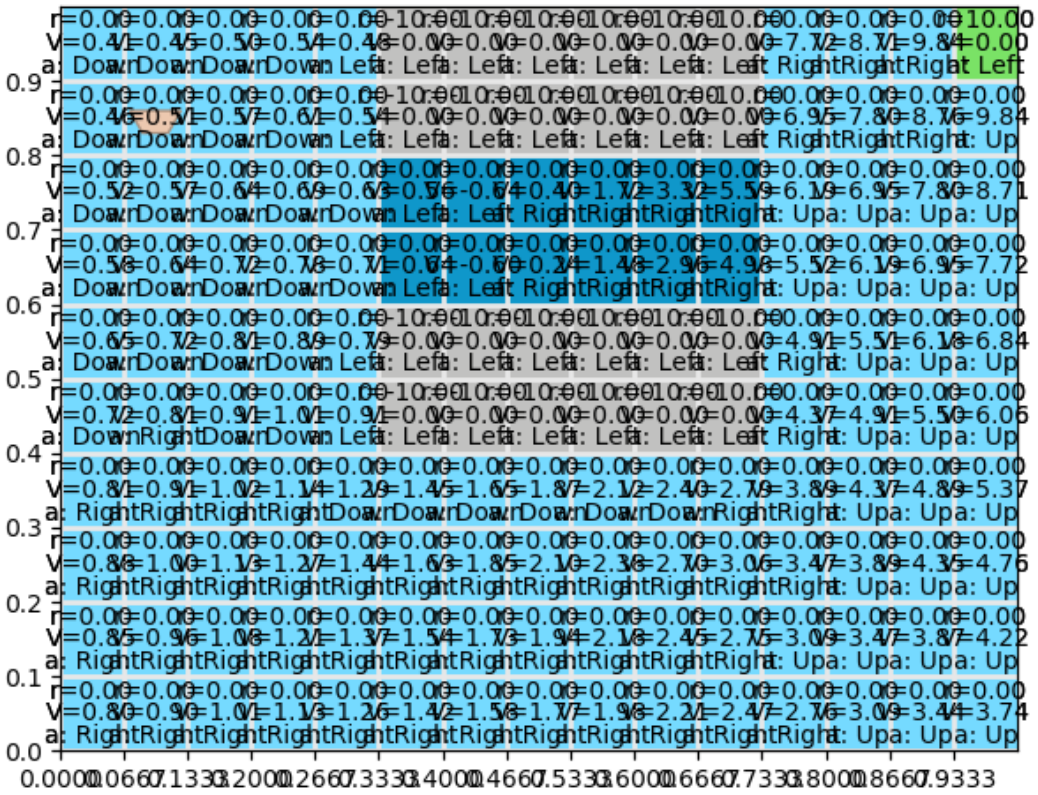

Figure 2: Reward, values and policy, when rock penalty is -10

3 Task 3: run until convergence

The algorithm that is implemented for changing the termination condition is shown in Algorithm 2.

Algorithm 2: Run until convergence

```
# Reset the environment
    state = env.reset()

    # Compute state values and the policy
    value_est, policy = np.zeros((env.w, env.h)), np.zeros((env.w, env.h))

    n_actions = env.transitions.shape[2]
    gamma = 0.9
    Q = np.zeros((env.w, env.h, n_actions))
    epsilon = 1e-4
    deltaV = [] # np.zeros((env.w, env.h))
    deltaQ = []
```

```
v_last = np.zeros((env.w, env.h))
p_last = np.zeros((env.w, env.h))
for_ in range(100):
    for i in range(env.w):
        for j in range(env.h): #-1,-1,-1):
            for a in range(n_actions):
                value_s_primes = 0
                for s in range(len(env.transitions[i, j, a])):
                    s_prime = env.transitions[i, j, a][s][0]
                    r = env.transitions[i, j, a][s][1]
                    tr = env.transitions[i, j, a][s][3]
                    if s_prime is not None:
                        value_s_primes += tr * (r + gamma*value_est[s_prime[0], s_prime
                Q[i, j, a] = value_s_primes
            value_est[i, j] = np.max(Q[i, j])
    policy = np.argmax(Q, axis=2)
    deltaV.append(np.sum(np.abs(value_est - v_last)))
    deltaQ.append(np.sum(np.abs(policy - p_last)))
    if(deltaV[-1] < epsilon):
        break
    # if(deltaQ[-1] < epsilon):
    #     break
    for i in range(env.w):
        for j in range(env.h):
            v_last[i, j] = value_est[i, j]
            p_last[i, j] = policy[i, j]
```

I noticed that the value matrix converges after 31 iterations and it will not change very much after that.

4 Task 4: discounted return of the initial state

The program that I implemented to compute the discounter return of the initial state is shown in 3. A loop has been created to run the program for $N = 1000$ episodes. The average and standard deviation of the discounted return for these 1000 episodes is : 0.6104 ± 1.3583. The values of discounted return has been saved in "return_history.npy" and attached to the submission.

Algorithm 3: Reward modes

```
return_history, timestep_history = [], []
gamma = 0.9
train_episodes = 1000
for episode_number in range(train_episodes):
    return_ini_state, timesteps = 0, 0
```

```
        done = False
        # Reset the environment and observe the initial state
        state = env.reset()
        while not done:
            # Select a random action
            # action = int(np.random.random()*4)
            # select action based on the policy obtained in previous tasks
            action = policy[state[0], state[1]]

            # Step the environment
            state, reward, done, _ = env.step(action)

            # Render and sleep
            # env.render()
            # sleep(0.5)
            return_ini_state += (gamma**timesteps) * reward
            timesteps += 1

        print("Episode {} finished. Discounted return: {:.3g} ({} timesteps)"
              .format(episode_number, return_ini_state, timesteps))

        # Bookkeeping (mainly for generating plots)
        return_history.append(return_ini_state)
        timestep_history.append(timesteps)

avg = np.mean(return_history)
sd = np.std(return_history)
print(avg)
print(sd)
```

Question 5 What is the relationship between the discounted return and the value function? Explain briefly.

There is a strong relationship between a value function and a return. The value of a state s under a certain policy π, $V_\pi(s)$, is defined as the "expected return" starting from state s. It means that the value function calculates the expected return from being in a certain state, or taking a specific action in a specific state. The return is a random variable of the score of a single episode, while the value function is an expectation over all possible episodes and can start from any state. In task 4 we calculated the average of return for 1000 episodes, so it is an estimation of expectation over some episodes, therefore it should be very close to the value of initial state at value matrix acquired from task 1 which was 0.6577.

Question 6 Imagine a reinforcement learning problem involving a robot exploring an unknown environment. Could the value iteration approach used here be applied directly to that problem? Why/why not? Which of the assumptions are unrealistic, if any?

In my opinion as the value iteration is a model-based approach, it is only applicable for the environment that it has been learned on it. It uses the transition models and rewards for

one specific environment, it cannot be applied for another environment.