Exercise 2

Nguyen Xuan Binh - 887799
ELEC-E8125 - Reinforcement Learning

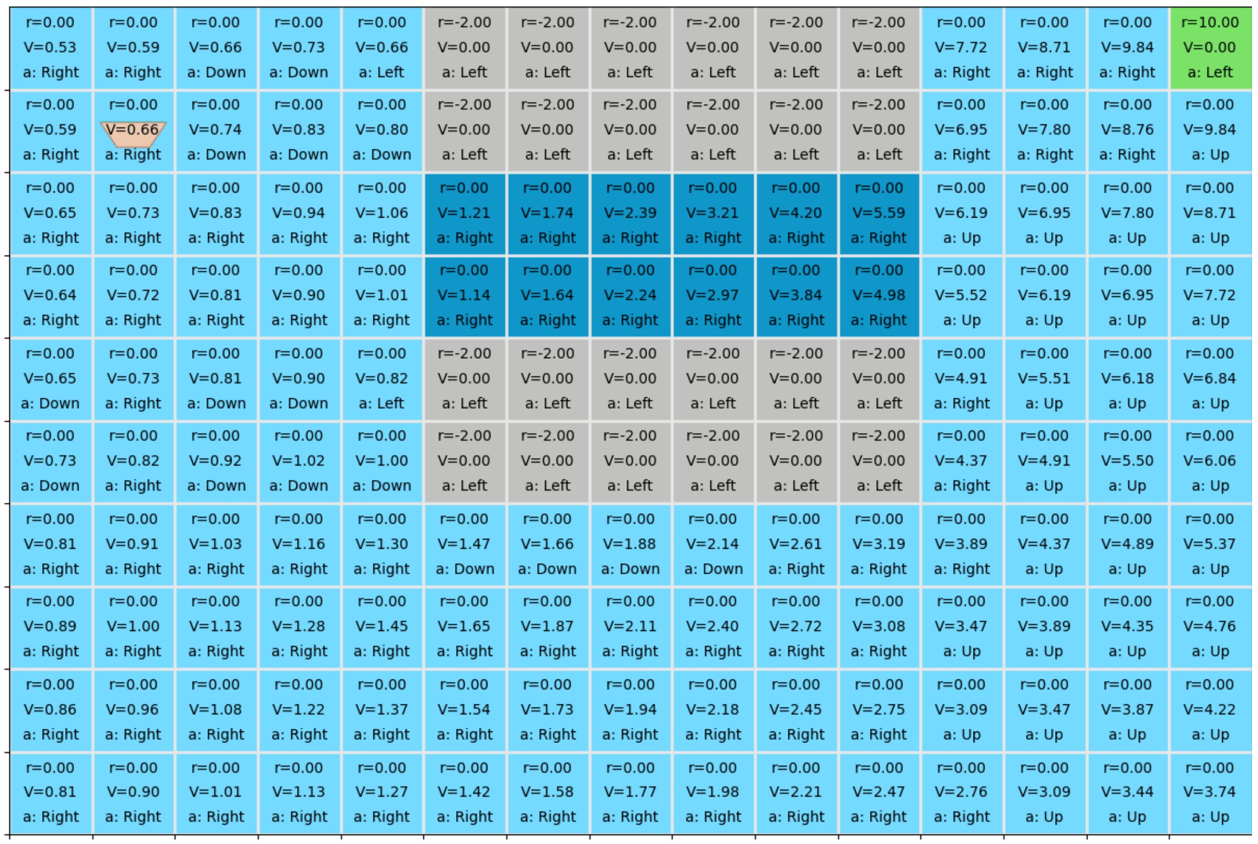
July 16, 2023

Task 1 - 30 points

i) Implement value iteration for the sailor example in file value_iteration.ipynb, assuming the discount factor value $\gamma = 0.9$.

ii) In addition to the state values, compute the policy – path to the harbour, using computed state values. Run your implementation for 100 iterations. Attach this .pkl file with the estimated state values and policy into your submission.

This is the converged value and policy after 100 iterations and the Python code



r=0.00 V=0.53 a: Right	r=0.00 V=0.59 a: Right	r=0.00 V=0.66 a: Down	r=0.00 V=0.73 a: Down	r=0.00 V=0.66 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=0.00 V=7.72 a: Right	r=0.00 V=8.71 a: Right	r=0.00 V=9.84 a: Right	r=10.00 V=0.00 a: Left
r=0.00 V=0.59 a: Right	r=0.00 V=0.66 a: Right	r=0.00 V=0.74 a: Down	r=0.00 V=0.83 a: Down	r=0.00 V=0.80 a: Down	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=0.00 V=6.95 a: Right	r=0.00 V=7.80 a: Right	r=0.00 V=8.76 a: Right	r=0.00 V=9.84 a: Up
r=0.00 V=0.65 a: Right	r=0.00 V=0.73 a: Right	r=0.00 V=0.83 a: Right	r=0.00 V=0.94 a: Right	r=0.00 V=1.06 a: Right	r=0.00 V=1.21 a: Right	r=0.00 V=1.74 a: Right	r=0.00 V=2.39 a: Right	r=0.00 V=3.21 a: Right	r=0.00 V=4.20 a: Right	r=0.00 V=5.59 a: Right	r=0.00 V=6.19 a: Up	r=0.00 V=6.95 a: Up	r=0.00 V=7.80 a: Up	r=0.00 V=8.71 a: Up
r=0.00 V=0.64 a: Right	r=0.00 V=0.72 a: Right	r=0.00 V=0.81 a: Right	r=0.00 V=0.90 a: Right	r=0.00 V=1.01 a: Right	r=0.00 V=1.14 a: Right	r=0.00 V=1.64 a: Right	r=0.00 V=2.24 a: Right	r=0.00 V=2.97 a: Right	r=0.00 V=3.84 a: Right	r=0.00 V=4.98 a: Right	r=0.00 V=5.52 a: Up	r=0.00 V=6.19 a: Up	r=0.00 V=6.95 a: Up	r=0.00 V=7.72 a: Up
r=0.00 V=0.65 a: Down	r=0.00 V=0.73 a: Right	r=0.00 V=0.81 a: Down	r=0.00 V=0.90 a: Down	r=0.00 V=0.82 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=0.00 V=4.91 a: Right	r=0.00 V=5.51 a: Up	r=0.00 V=6.18 a: Up	r=0.00 V=6.84 a: Up
r=0.00 V=0.73 a: Down	r=0.00 V=0.82 a: Right	r=0.00 V=0.92 a: Down	r=0.00 V=1.02 a: Down	r=0.00 V=1.00 a: Down	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=-2.00 V=0.00 a: Left	r=0.00 V=4.37 a: Right	r=0.00 V=4.91 a: Up	r=0.00 V=5.50 a: Up	r=0.00 V=6.06 a: Up
r=0.00 V=0.81 a: Right	r=0.00 V=0.91 a: Right	r=0.00 V=1.03 a: Right	r=0.00 V=1.16 a: Right	r=0.00 V=1.30 a: Right	r=0.00 V=1.47 a: Down	r=0.00 V=1.66 a: Down	r=0.00 V=1.88 a: Down	r=0.00 V=2.14 a: Down	r=0.00 V=2.61 a: Right	r=0.00 V=3.19 a: Right	r=0.00 V=3.89 a: Right	r=0.00 V=4.37 a: Up	r=0.00 V=4.89 a: Up	r=0.00 V=5.37 a: Up
r=0.00 V=0.89 a: Right	r=0.00 V=1.00 a: Right	r=0.00 V=1.13 a: Right	r=0.00 V=1.28 a: Right	r=0.00 V=1.45 a: Right	r=0.00 V=1.65 a: Right	r=0.00 V=1.87 a: Right	r=0.00 V=2.11 a: Right	r=0.00 V=2.40 a: Right	r=0.00 V=2.72 a: Right	r=0.00 V=3.08 a: Right	r=0.00 V=3.47 a: Up	r=0.00 V=3.89 a: Up	r=0.00 V=4.35 a: Up	r=0.00 V=4.76 a: Up
r=0.00 V=0.86 a: Right	r=0.00 V=0.96 a: Right	r=0.00 V=1.08 a: Right	r=0.00 V=1.22 a: Right	r=0.00 V=1.37 a: Right	r=0.00 V=1.54 a: Right	r=0.00 V=1.73 a: Right	r=0.00 V=1.94 a: Right	r=0.00 V=2.18 a: Right	r=0.00 V=2.45 a: Right	r=0.00 V=2.75 a: Right	r=0.00 V=3.09 a: Up	r=0.00 V=3.47 a: Up	r=0.00 V=3.87 a: Up	r=0.00 V=4.22 a: Up
r=0.00 V=0.81 a: Right	r=0.00 V=0.90 a: Right	r=0.00 V=1.01 a: Right	r=0.00 V=1.13 a: Right	r=0.00 V=1.27 a: Right	r=0.00 V=1.42 a: Right	r=0.00 V=1.58 a: Right	r=0.00 V=1.77 a: Right	r=0.00 V=1.98 a: Right	r=0.00 V=2.21 a: Right	r=0.00 V=2.47 a: Right	r=0.00 V=2.76 a: Right	r=0.00 V=3.09 a: Up	r=0.00 V=3.44 a: Up	r=0.00 V=3.74 a: Up

```python
def get_values_policy(iterations):
    v_est = np.zeros((env.w, env.h))
    policy = np.zeros((env.w, env.h))
    # env.draw_values_policy(v_est, policy)

    value_converge_index = -1
    policy_converge_index = -1
    check_value_converge = False
    check_policy_converge = False

    for i in range(1, iterations + 1):
        # TODO: Task 1, implement the value iteration and policy
        # TODO: Task 2, convergency of the value function and policy
        old_v_est = v_est.copy()
        new_v_est = v_est.copy()
        old_policy = policy.copy()
        new_policy = np.zeros((env.w, env.h))

        # Iterate over all state coordinates
        for x in range(env.w):
            for y in range(env.h):
                Q_values = np.zeros(env.n_actions)
                # Iterate over all actions and calculate the Q value for each action
                for action in range(env.n_actions):
                    transitions = env.transitions[x, y, action]
                    Q_value = 0
                    for next_state, reward, done, probability in transitions:
                        if not done:
                            next_state_value = v_est[next_state]
                        else:
                            next_state_value = 0 # Terminal state value should be zero
                        Q_value += probability * (reward + gamma * next_state_value)
                    Q_values[action] = Q_value
                new_v_est[x, y] = np.max(Q_values) # Updating state value with maximum Q value
                new_policy[x, y] = np.argmax(Q_values) # Updating state policy with action that maximizes Q values
    # Calculate the change in the value function among all states
        deltas = np.abs(new_v_est - old_v_est)

        # Check if the value function has converged
        if np.all(deltas < eps):
            if not check_value_converge:
                value_converge_index = i
                check_value_converge = True
            # The draw_values_policy function is slow and called only once
            # env.draw_values_policy(new_v_est, new_policy)

        # Check if the policy function has converged
        if np.array_equal(old_policy, new_policy):
            if not check_policy_converge:
                policy_converge_index = i
                check_policy_converge = True

        #print(f"Iteration {i+1} finishes")

        # Updating the value function and policy
        v_est = new_v_est
        policy = new_policy
        #print(f"Iteration {i} finishes")

        # The draw_values_policy function is slow and called only once
        if i == iterations:
            env.draw_values_policy(v_est, policy)
        #print(i)
        #print(iterations)

    print(f"Value iteration has converged after {value_converge_index} iterations")
    print(f"Policy function has converged after {policy_converge_index} iterations")

    return v_est, policy
```

Question 1.1 - 5 points

What is the agent and the environment in this sailor gridworld?

The agent is the sailor boat and the environment is the whole gridworld, consisting of the sea, the harbour and the rocks.

Question 1.2 - 5 points

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

The harbour and the rocks are the terminal states. Therefore, their values should not contribute to updating the values of the sea states, so the values of the harbour and rock states are zero.

Question 1.3 - 5 points

Which path did the sailor choose, the safe path below the rocks, or the dangerous path between the rocks? If you change the reward for hitting the rocks to -10 (that is, make the sailor value life more), does he still choose the same path?

The path the sailor chose is the dangerous path between the rocks when the terminal state value of the rocks is -2. However, if we change the reward (or penalty) for hitting the rocks to -10, the sailor no longer chose the dangerous path. Instead, he chose the safer detour around the rocks. This is probably because the reward of reaching the harbour is not high enough to mitigate the risk of travelling through the dangerous passage between the rocks.

Task 2 - 15 points

What happens if you run the algorithm for 30 iterations? Do the value function and policy still converge? For the value function, you can assume they have converged if the maximum change in value is lower than a certain threshold $\epsilon = 10^{-4}$:

$$max_s |V_k(s) - V_{k-1}(s)| < \epsilon, (2)$$

where $V_k(s)$ is the estimated value of state s in k-th iteration of the algorithm. Generally, which of them - the policy or value function - needs less iterations to converge, if any? Justify your answer.

When I run the algorithms for 30 iterations, the value estimation has not managed to converge within the allowed threshold, while the policy still manages to converge after 26 iterations. Generally, the policy function needs less iterations to converge than the value function. This is because the policy space of the sailor world is relatively small (4 actions), so the algorithm can find a clear strategy structure quickly, while the value iteration is a continuous float number, which takes more iterations to get to the correct decimal precision.

Task 3 - 5 points

Set the reward for crashing into the rocks back to -2. Change the termination condition of your algorithm to make it run until convergence, see Eq. (2). Report the number of iterations required for the value function to converge.

The number of iterations required for the value function to converge is 39 (my iterations start counting from 1, not 0).

Task 4 - 10 points

Evaluate your learned policy for N = 1000 episodes, and compute the discounted return of the initial state, see [1] Eq. (3.8), for each episode. The reward for crashing into rocks must be kept at -2 for this exercise. Report the average and standard deviation of the initial state's discounted return over the N=1000 episodes.

The average of the initial state's discounted return discounted return is 0.6437
The standard deviation of the initial state's discounted return discounted return is 1.3685

Question 4.1 - 10 points

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

Definition: The value function is the expected cumulative reward of each state when the agent follows the policy starting from the initial state. Additionally, the discounted return is the sum of the rewards accumulated over N steps, with each reward discounted by a factor γ after each step. Therefore, the value function at a state s is the expected sum of discounted rewards if the agent starts from the known initial state and follows a fixed policy.

Question 4.2 - 15 points

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

My answer is yes and no. Yes in that the principles and the environment settings are the same, but no because some assumptions in this exercise is too simple for the RL problem involving a robot exploring unknown environments. The unrealistic assumptions in this exercise that cannot be applied to the robot problem is:
- Four discrete actions: Left, Right, Up, Down. In reality, the robot movement is much more flexible, which could be described as a continuous domain instead of the discrete domain.
- Value iteration algorithm assumes that the agent has full access to the state space and and transition probabilities. In reality, these information could not be easily be observed.
- the searching space would be very large (which could have more than million states), rendering the value iteration too slow and impractical.