Reinforcement Learning
Exercise 2

September 30, 2019

1 Sailor gridworld

Consider a sailor who managed to escape from a sinking ship, and now has to find his way to the nearest harbour. The sea is divided into a grid, with each grid cell corresponding to a state. Therefore, the state can be thought of as a two dimensional vector:

$$s = \begin{pmatrix} x \\ y \end{pmatrix} \tag{1}$$

There are four actions available: moving left, right, up and down. When the sailor reaches the harbour, the episode terminates and a reward of 10 is given. If the sailor hits the rocks, the episode terminates and a reward of -2 is given. On all other steps, the reward is 0. The environment is shown in Figure 1.

The shortest way to the harbour goes through a narrow passage between rocks, which is known to have unpredictable heavy wind conditions. When moving in that area, the sailor can be carried an extra "square" in a random direction — that is, land in any of the squares adjacent to the desired target square. This is shown in Figure 2.

The sea around the passage is generally calm, but there's a low probability that the sailor will be carried in the direction perpendicular to where he was heading, as shown in Figure 3.

All of these probabilities (p_{calm} and p_{wind}) as well as the effects of the wind and the exact location of the harbour are perfectly known to the sailor.

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

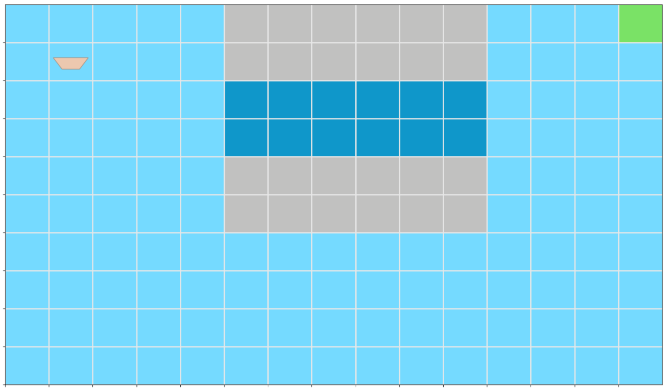
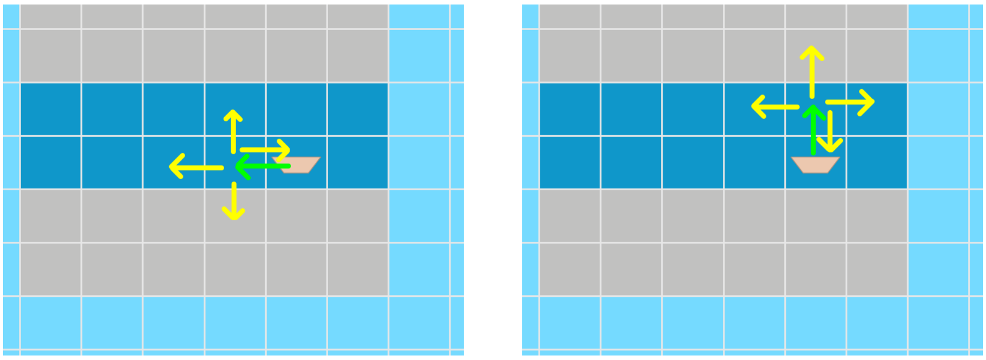


Figure 1: The Sailor gridworld environment. Light blue squares represent the calm part of the sea, gray squares - the rocks, dark blue - the windy passage between the rocks. The green square in upper right corner is the target harbour. The current (in this picture also the initial) position of the sailor is denoted with a brown "boat".



(a) (b)

Figure 2: Possible state transitions in windy passage when the issued action was to go right (a) or to go down (b). The sailor may end up in the square to the right (a) or bottom (b), as indicated by the green arrow. There is also a small p_{wind} that the sailor will move for an additional unit in a random direction, as indicated by one of the yellow arrows. Therefore, in addition to moving one square in the target direction, it can (1) move two squares in the desired direction, (2) stay in place, or (3) be carried sideways to one of the squares on the diagonal.

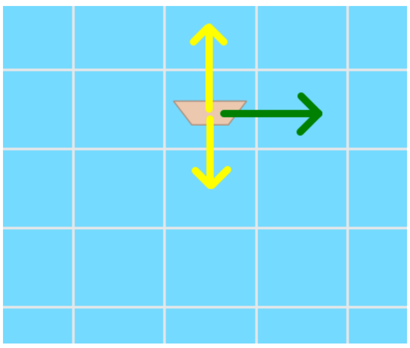


Figure 3: Possible state transitions in calm water when the issued action was to go right. The sailor may end up in the square to the right, as indicated by the green arrow. There is a small chance p_{calm} that the sailor will move in the perpendicular direction, as indicated by the yellow arrows.

2 Value iteration

Value iteration is a method of computing an optimal MDP policy. We start with arbitrary initial state values and iteratively update our estimate of every state's value by using Bellman equation as an update rule. A more detailed description, along with the exact equations, can be found in Sutton & Barto, Section 4.4.

Task 1 Implement value iteration for the sailor example, assuming the discount factor value $\gamma = 0.9$. Run your implementation for 100 iterations. Render the environment after every iteration and observe how the values are updated.

Hint: The environment contains a 3-D array (`env.transitions`) of shape $[n_x, n_y, n_a]$, which contains all possible state transitions. The transitions for state (x, y) and action a can be accessed by `env.transitions[x, y, a]`. This will return an array of Python tuples structured as $(s', reward, done, p)$. For example, `env.transitions[3, 3, env.UP]` would return an array of three possible state transitions:

```
((3, 2), 0.0, 0.0, 0.05),
((3, 4), 0.0, 0.0, 0.05),
((4, 3), 0.0, 0.0, 0.9),
```

which corresponds to moving to state (4, 3) with probability 0.9, or moving to states (3, 4) and (3, 2) with probability 0.05 for each. None of these transitions results in a reward or in terminating the episode (the second and third elements are zero). When the episode has already terminated, the next state will be set to `None`.

Hint: Use the `env.draw_values` function to draw the state values on the grid (the values must be passed inside a $[w \text{ x } h]$ NumPy array). Use `env.clear_text` between iterations to remove previously drawn values.

Caveat: Pay extra attention to indices in the Bellman equation - specifically, where V_k and where V_{k-1} must be used.

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

Task 2 In addition to the state values, compute the optimal path to the harbour using previously computed state values. Run the program a few times and see if the sailor is able to reach his goal every time.

Hint: Use the `env.draw_actions` function to draw the actions on the grid.

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?

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.

Task 3 Set the reward for crashing into the rocks back to -2. Change the termination condition of your algorithm to make it run until convergence. You can assume the values to have converged if the maximum change in value is lower than a certain threshold ϵ:

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

where $V_k(s)$ is the estimated value of state s in k-th iteration of the algorithm. Assume $\epsilon = 10^{-4}$ in your implementation.

Task 4 Modify the program to compute the discounted return of the initial state. Create a loop to run the program for $N = 1000$ episodes and see what values you get. Report the average and standard deviation.

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

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?

3 Submission

Your **submission** *must* include the `.npy` files for the **state values** and the **policy**. Your report *must* include the discounted return values (Task 4). All of those should be computed for the initial setup ($r_{harbour} = 10$, $r_{rocks} = -2$).

Include the code used for all the tasks and answers to all questions asked in the instructions.

The **deadline** to submit your answers is **13th October at 23:55**. If you need help with the assignment, you are welcome to join the exercise sessions in weeks 40-41.