Reinforcement Learning
Exercise 2

September 13, 2022

Sailor gridworld

Consider a sailor who managed to escape from a sinking ship, and now has to find the 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 is 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.

In this exercise round you only need to modify the provided **`value_iteration.ipynb`** Jupyter notebook file.

Aalto University
School of Electrical
Engineering

Reinforcement Learning course staff
Aalto Robot Learning Lab
aalto.fi, rl.aalto.fi

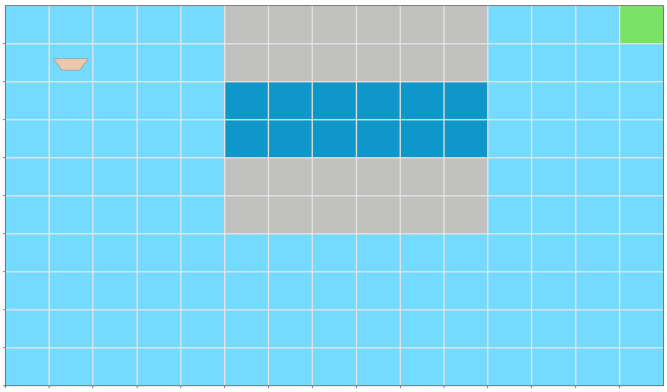
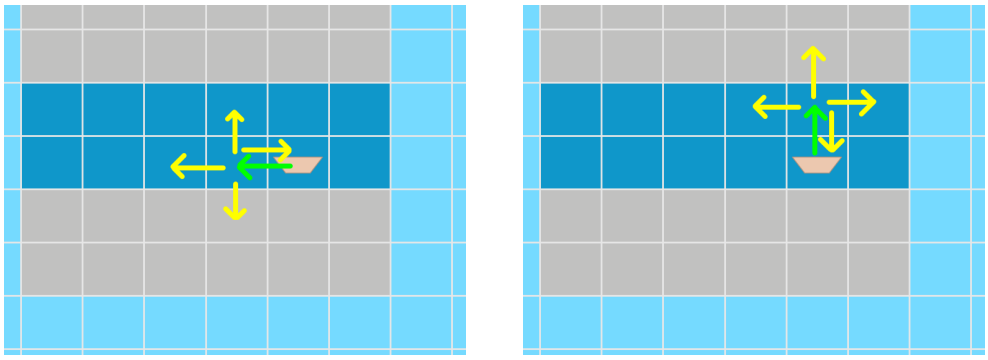


Figure 1: The Sailor gridworld environment. Light blue squares represent the calm part of the sea, gray squares – the rocks, dark blue – the windy passage between the rocks. The green square in upper right corner is the target harbour. The current (in this picture also the initial) position of the sailor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



(a) (b)

Figure 2: Possible state transitions in windy passage when the issued action was to go left (a) or to go up (b). The sailor may end up in the square to the left (a) or up (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.

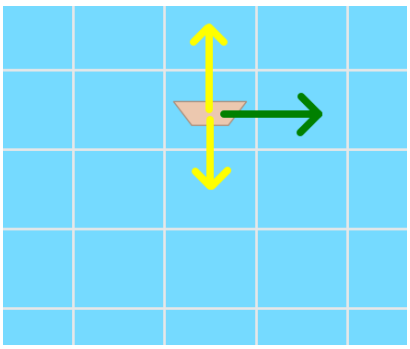


Figure 3: Possible state transitions in calm water when the issued action was to go right. The sailor may end up in the square to the right, as indicated by the green arrow. There is a small chance p_{calm} that the sailor will move in the perpendicular direction, as indicated by the yellow arrows.

Value iteration

Value iteration is a method for computing an optimal MDP (Markov Decision Process) 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 [1] Section 4.4.

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. Render the values and policy after every iteration and observe how the values and policy are updated. Also, run the program a few times and check if the sailor is able to reach the goal every time. **The last cell in `value_iteration.ipynb` saves the values and policy as a .pkl file. Attach this .pkl file with the estimated state values and policy into your submission.**

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 a list of Python *named-tuples* Transition=$(s', reward, done, p)$. The components of the named-tuple can be accessed as `transition.state`, `transition.reward`, `transition.done`, `transition.prob`.

For example, `env.transitions[3, 3, env.UP]` would return a list of three possible state transitions:

```
(state=(3, 2), reward=0.0, done=0.0, prob=0.05),
(state=(3, 4), reward=0.0, done=0.0, prob=0.05),
(state=(4, 3), reward=0.0, done=0.0, prob=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_policy` function to draw the state values and policy on the grid (the values and policy must be passed inside a (x, y) NumPy array respectively). The environment dimensions can be accessed for x by `env.w`, and for y by `env.h`.

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

Question 1.1 - 5 points What is the agent and the environment in this sailor gridworld?

Question 1.2 - 5 points What is the state value of the harbour and rock states? Why?

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?

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 $\epsilon = 10^{-4}$:

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

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.**

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.**

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.**

Question 4.1 - 10 points What is the relationship between the discounted return and the value function? Explain briefly.

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?

Submitting

The deadline to submit the solutions through MyCourses is Sep.26, 23:55. Example solutions will be presented in the exercise session on Monday Oct.03.

The report *must* include:

1. Answers to all Questions.
2. Number of iterations required for the value function to converge in Task 3, and average and standard deviation for the discounted return in Task 4.

In addition to the report, you must submit as separate files:

1. Python code used to solve **all Task exercises**.
2. `.pkl` files for the **state values** and the **policy** (Task 1).

All the attached files **must** be solved for the initial setup $r_{harbour} = 10$, $r_{rocks} = -2$).

Please remember that not submitting a PDF report following the **Latex template** provided by us will lead to subtraction of points.

For more formatting guidelines and general tips please refer to the README.md.

If you need help or clarification solving the exercises, you are welcome to come to the exercise sessions. Good luck!

References

[1] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.