Imperial College London

October 24, 2023

Department of Computing
Reinforcement Learning – Dr Edward Johns, Prof Aldo Faisal and Dr Nicole Salomons
Coursework design: Prof Aldo Faisal, Manon Flageat, Dr Edward Johns & Filippo Valdettaro

Assessed Coursework 1

The Coursework should be submitted by November 7th, 7 pm, and consist of:
- A **PDF** of your written report, to be submitted on Scientia, named ***coursework1_report.pdf***.
- A **ipynb** notebook of your code, to be submitted on Scientia along with the report, named ***coursework1.ipynb***, with the final version of the code you used to generate the plots in your report.
- A **Python file** of the code exported from your notebook with your solutions to the coursework coding tasks, named ***coursework1.py***. This is to be submitted, and tested on LabTS, by pushing the Python file to the GitLab repository that you will be given access to for this coursework (see the `README` file in the repository for more details) which is not to be uploaded to Scientia.

Please ensure that you are familiar with the Scientia and LabTS submission processes well before the deadline. Unfortunately, emailed or printed submissions cannot be accepted.

Report: Your report should not be longer than **7 single-sided A4 pages with at least 2-centimeter margins all around and a font size of minimum 11pt**. Appendixes are not allowed. 7 pages is a **maximum** length, shorter submissions are fine, and those over the page limit will incur a penalty. Questions that should be answered in the report are indicated in the following as **[Report]**, questions indicated as **[Notebook]** do not need to be answered in the report. Submissions that violate these requirements will receive mark penalties.
Notebook: Please include the final version of your notebook in the submission. This will not be formally graded but may be looked at by markers in case it is deemed appropriate.
Code: Your code should follow the guidelines detailed below to ensure it can be automatically marked. **DO NOT** add any non-optional inputs to the functions and methods. You can run the code on LabTS to check that your answers are in the correct format and are behaving as expected with the automated marking. There is a 5 minute runtime limit for your submission on LabTS, which is more than enough time to train successful agents for the tasks considered here. Make sure that the functions and agent classes you write are fully self-contained (i.e. they don't rely on global variables) and only access external class attributes that you are explicitly told you can access in the question. You should only modify those classes and functions that you are explicitly told to modify in the specifications. You can reuse code you developed in the lab assignments and computer labs, but your code should be your own. When submitting your `coursework1.py` file to LabTS, please remove the code to run experiments or generate plots so that experiments are not run and plots are not generated when the code is imported for testing.

You are encouraged to discuss the general coursework questions with other students, but your answers should be yours, i.e., written by you, in your own words, showing your own understanding. Your report and code will be automatically verified for plagiarism. Written answers should be clear, complete and concise. Figures should be clearly readable, labelled, captioned and visible. Poorly produced answers, incomplete figures, irrelevant text not addressing the point and unclear text, may lose points.

Marks are shown next to each question. Note that the maximum mark achievable is 45 pts. Please note that this coursework counts 25% of your final grade, and so carries a substantially higher grade than regular courseworks, and correspondingly, is likely to take you longer to complete than regular courseworks. If you have questions about the coursework please make use of the computer labs or EdStem; the Graduate Teaching Assistants cannot provide you with answers that directly solve the coursework but are very happy to help you with questions.

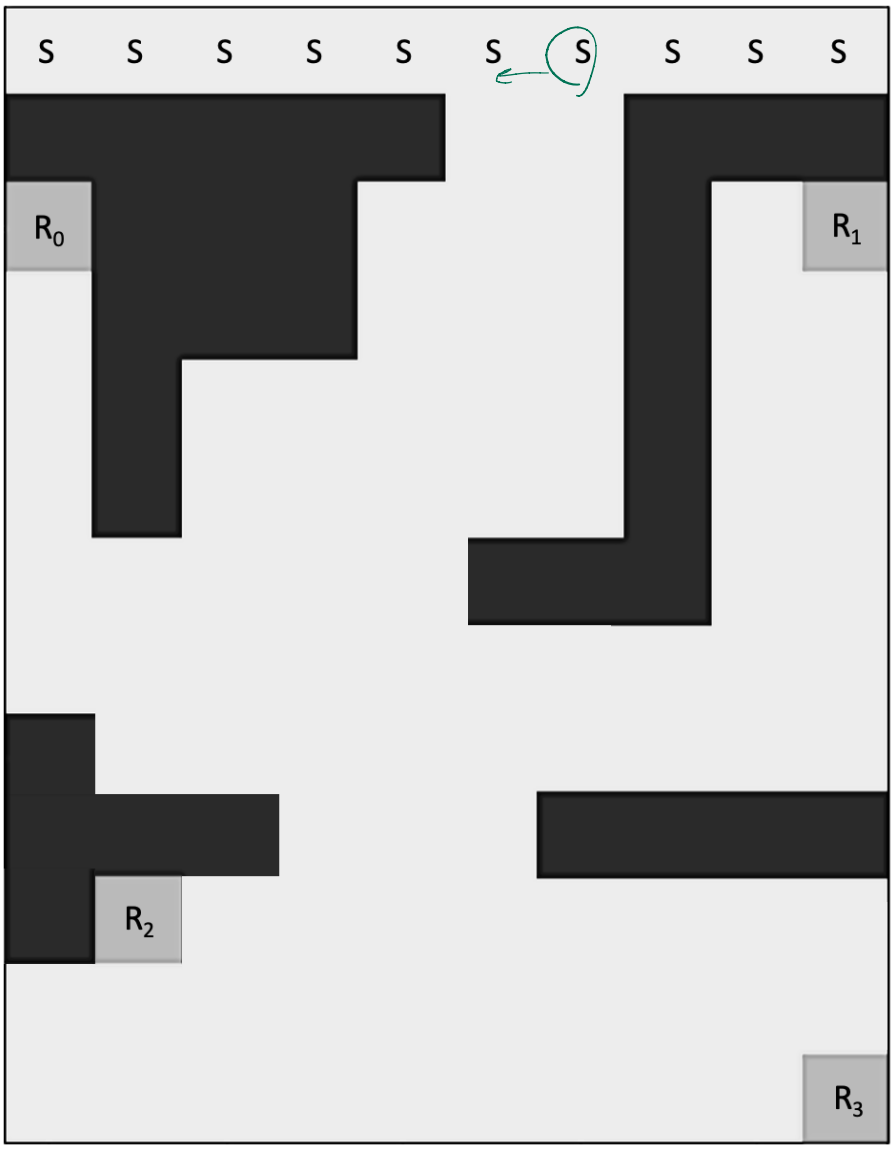


Figure 1: Illustration of the Maze environment, the focus of this Coursework. The absorbing states are indicated as "R_i", they correspond to absorbing rewards. The "S" indicate the potential start states.

Description of the Maze environment

The focus of this Coursework is to solve a Maze environment, illustrated in Figure 1, modelled as a Markov Decision Process (MDP). In this illustration, the black squares symbolise obstacles, and the dark-grey squares absorbing states, that correspond to specific rewards. Absorbing states are terminal states, there is no transition from an absorbing state to any other state.

This environment shares a lot of similarities with the Grid World environment studied in Lab 2, you are allowed to reuse any code you might have developed for this tutorial. However, unlike the lab, this Coursework aims to help you implement RL techniques that do not require full knowledge of the dynamic of the environment, namely Monte-Carlo and Temporal-Difference learning. In the following, we will thus assume we don't have access to the transition matrix T and reward function R of the environment, and try to find the optimal policy π by sampling episodes in it.

Consider an agent moving inside this maze, trying to get the best possible reward. To do so, it can choose at any time-step between four actions:

- $a_0 =$ *going north* of its current state
- $a_1 =$ *going east* of its current state
- $a_2 =$ *going south* of its current state
- $a_3 =$ *going west* of its current state

The chosen action has a probability p to succeed and lead to the expected direction. If it fails, it has equal probability to lead to any other direction. For example, if the agent chooses the action $a_0 =$ *going north*, it is going to succeed and go north with probability p, and it has probability $\frac{1-p}{3}$ to go east, probability $\frac{1-p}{3}$ to go south and probability $\frac{1-p}{3}$ to go west. p is given as a hyperparameter, defining the environment.

If the outcome of an action leads the agent to a wall or an obstacle, its effect is to keep the agent where it is. For example, if the agent chooses the action $a_1 =$ *going east* and there is a wall east but no wall in the other directions, it is going to stay in place in his current state with probability p and to go north, south or west with the same probability $\frac{1-p}{3}$.

Any action performed by the agent in the environment gives it a reward of -1.

The episode ends when the agent reach one of the absorbing states $(R_i)_{0 \leq i \leq 3}$ or if it performs more than 500 steps in the environment.

Finally, the starting state is chosen randomly at the beginning of each episode from the possible start states with equal probability $1/N_{start-states}$. The possible start states are indicated on the Maze illustration as "S".

This environment is personalised by your College ID (CID) number, specifically the last 2 digits (which we refer to as y and z), as follows: 06009527 (y = 2, z = 7)

- The probability p is set as $p = 0.8 + 0.02 \times (9 - y)$. $= 0.8 + 0.02 \times (7) = 0.94$
- The discount factor γ is set as $\gamma = 0.8 + 0.02 \times y$. $= 0.8 + 0.02 \times 2 = 0.84$
- The reward state is R_i with $i = z \ mod \ 4$ (meaning that if $z = 0$ or $z = 4$ or $z = 8$, the reward state would be R_0, if $z = 1$ or $z = 5$ or $z = 9$, it would be R_1, etc). $7 \ mod \ 4 = R_3$
- The reward state R_i gives a reward $r = 500$ and all the other absorbing states gives a reward $r = -50$. As stated before, any action performed by the agent in the environment also gives it a reward of -1.

Jupyter Notebook

This Coursework is based on the provided Jupyter Notebook, also available on Colab: `https://colab.research.google.com/drive/1G3KJGCqR13NBt4F7oH5s6KYFS9UBigD1?usp=sharing`. It defines most of the Maze structure. In the following questions, you will be asked to progressively complete the functions of the Maze class indicated with the tag "[Action required]".

The Notebook has the following structure:

1. `get_CID()` and `get_login()` functions definition. These functions should return your CID and short login. They are used by the auto-marking script, make sure to fill them in.
2. `GraphicsMaze` class definition: allow all the graphical visualisation of the Maze. You **DO NOT NEED** to read or understand this class, you can simply call its methods when needed.

3. `Maze` class definition: this class is the main Maze class, you will be asked to complete it in the following questions. All attributes and methods of this class starting with `_` such as `env._T` or `env._build_maze()` are private and should NOT be accessed outside the `Maze` class. `Get` functions such as `get_T()` are defined to access some of them from outside the class.
4. `DP_agent` class definition: this class defines a Dynamic Programming agent to solve the Maze, you will be asked to complete it in the following questions.
5. `MC_agent` class definition: this class defines a Monte-Carlo Learning agent to solve the Maze, you will be asked to complete it in the following questions.
6. `TD_agent` class definition: this class defines a Temporal-Difference Learning agent to solve the Maze, you will be asked to complete it in the following questions.

Question 0: Defining the environment – 1 pts

We are first going to define the Maze environment used in the following. This question is only code-based and does not need to be part of your Report.

[Notebook] – 1 pts

Fill in the following parts of the Jupyter Notebook:

- The functions `get_CID()` and `get_login()` to return your CID number and you short login. Make sure you fill in these functions as they will be used by the auto-marking script.
- The `__init__()` method of the `Maze` class, to set you personalised reward state, p, and γ in the environment.

Question 1: Dynamic programming – 14 pts

For comparison purposes, we are first going to assume we have access to the transition T and reward function R of this environment and consider a Dynamic Programming agent to solve this problem.

[Notebook] – 4 pts

Complete the `solve()` method of the `DP_agent` class. You can choose any Dynamic Programming method and you are allowed to reuse code from the lab assignment. Note that *for this agent only*, you are allowed to access the transition matrix through `env.get_T()`, the reward matrix through `env.get_R()` and the list of absorbing states through `env.get_absorbing()` of the `Maze` class. You might also need to use `env.get_action_size()`, `env.get_state_size()` and `env.get_gamma()`.

You can define any additional methods inside the `DP_agent` class, but only the `solve()` method will be called for automatic marking. **DO NOT** add any inputs or outputs to `solve()`.

[Report] – 10 pts

a. State which method you choose to solve the problem. Give and justify any parameters that you set. **– 3 pts**

b. Provide the graphical representation of your optimal policy and optimal value, which can be generated with the provided code to visualise policies and value functions. **– 2 pts**

For parts (c) and (d), ignore the 500 step limit per episode. You do not need to run additional experiments or produce plots for either of these parts. Instead, give answers and explain your reasoning.

c. What is the effect on the optimal policy and value function when rewards are scaled by a factor of 2? **– 2 pts**

d. Suppose you now wish to design the reward function such that, for any state s, the value for that state is equal to the probability of eventually reaching the goal state, when starting in state s. Specifically, you are tasked with finding an appropriate choice of:

- discount factor γ
- reward for reaching the goal state
- reward for reaching other terminal states

that would result in such a value function. **– 3 pts**

Question 2: Monte-Carlo Reinforcement Learning – 15 pts

We are now assuming that we do not have access to the transition T and reward function R of this environment. We want to solve this problem using a Monte-Carlo learning agent.

[Notebook] – 5 pts

Complete the `solve()` method of the `MC_agent` class. Note that for this agent you are only allowed to use the `env.reset()` and `env.step()` methods of the `Maze` class, as well as `env.get_action_size()`, `env.get_state_size()` and `env.get_gamma()`. **DO NOT** use the transition matrix `env.get_T()`, the reward matrix `env.get_R()` and the list of absorbing states `env.get_absorbing()`, or any other `env` attribute not mentioned above.

You can define any additional methods inside the `MC_agent` class, but only the `solve()` method will be called for automatic marking. **DO NOT** add any inputs or outputs to `solve()`.

[Report] – 10 pts

a. State which particular MC algorithm you choose to solve the problem. Give and justify the value of any parameters that you set. **– 3 pts**

b. Provide the graphical representation of your optimal policy and optimal value function, which can be generated with the provided code to visualise policies and value functions. **– 2 pts**

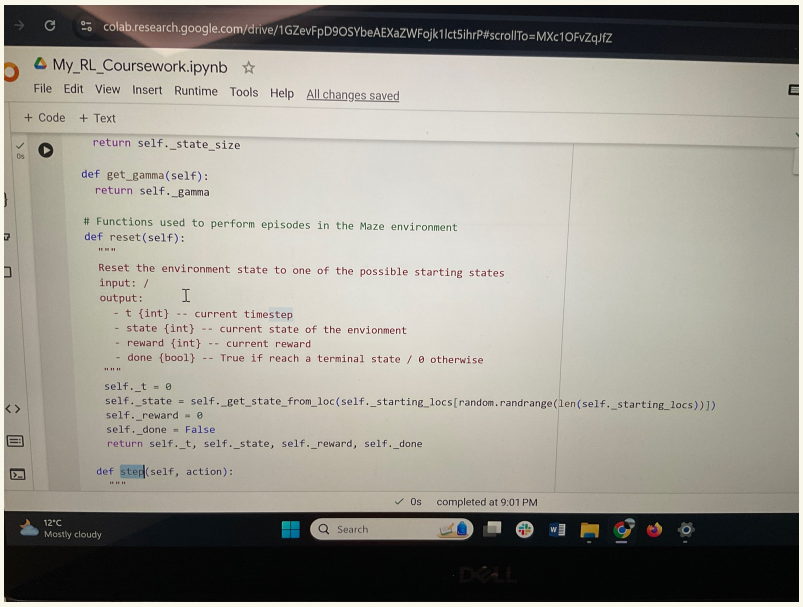
c. Plot the learning curve of your agent: the total non-discounted sum of rewards (vertical axis) against the number of episodes (horizontal axis). The figure should show the mean and shaded standard deviation on the same graph across 25 training runs. **– 2 pts**

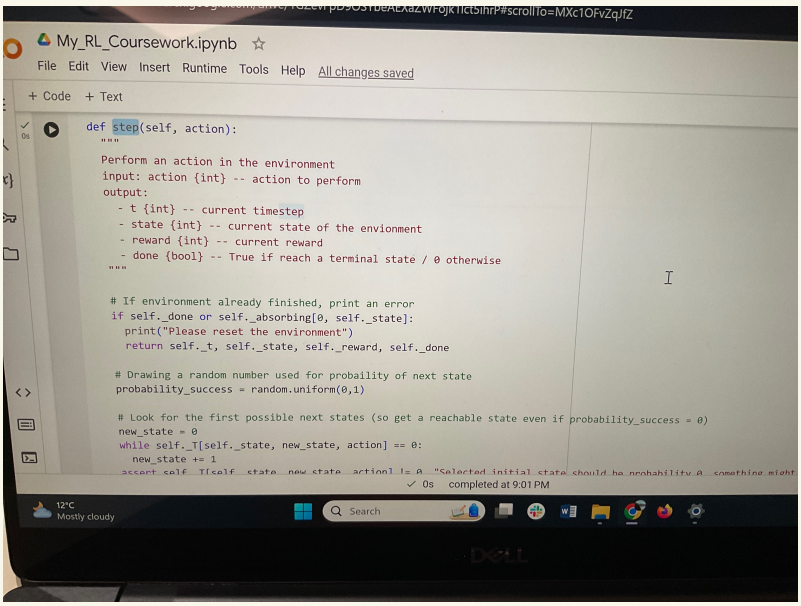
d. Consider an implementation which ignores trajectories that don't reach terminal states within 500 steps. In what ways would this affect the MC estimate of the value function compared to an implementation that does not impose a step limit per episode? You do not need to run experiments or provide plots, instead answer and give your reasoning based on your understanding of the lecture material. **– 3 pt**

Question 3: Temporal Difference Reinforcement Learning – 15 pts

We are still going to assume we do not have access to the transition T and reward function R of this environment. We want to solve this problem using a Temporal-Difference learning agent.

[Notebook] – 5 pts


colab.research.google.com/drive/1GZevFpD9OSYbeAEXaZWFojk1lct5ihrP#scrollTo=MXc1OFvZqJfZ
My_RL_Coursework.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
return self._state_size
def get_gamma(self):
return self._gamma
# Functions used to perform episodes in the Maze environment
def reset(self):
"""
Reset the environment state to one of the possible starting states
input: /
output:
- t {int} -- current timestep
- state {int} -- current state of the envionment
- reward {int} -- current reward
- done {bool} -- True if reach a terminal state / 0 otherwise
"""
self._t = 0
self._state = self._get_state_from_loc(self._starting_locs[random.randrange(len(self._starting_locs))])
self._reward = 0
self._done = False
return self._t, self._state, self._reward, self._done
def step(self, action):
"""
0s completed at 9:01 PM
12°C
Mostly cloudy
Search


My_RL_Coursework.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
def step(self, action):
"""
Perform an action in the environment
input: action {int} -- action to perform
output:
- t {int} -- current timestep
- state {int} -- current state of the envionment
- reward {int} -- current reward
- done {bool} -- True if reach a terminal state / 0 otherwise
"""
# If environment already finished, print an error
if self._done or self._absorbing[0, self._state]:
print("Please reset the environment")
return self._t, self._state, self._reward, self._done
# Drawing a random number used for probability of next state
probability_success = random.uniform(0,1)
# Look for the first possible next states (so get a reachable state even if probability_success = 0)
new_state = 0
while self._T[self._state, new_state, action] == 0:
new_state += 1
0s completed at 9:01 PM
12°C
Mostly cloudy
Search

Complete the `solve()` method of the `TD_agent` class. Note that for this agent you are only allowed to use the `env.reset()` and `env.step()` methods of the `Maze` class, as well as `env.get_action_size()`, `env.get_state_size()` and `env.get_gamma()`. **DO NOT** use the transition matrix `env.get_T()`, the reward matrix `env.get_R()` and the list of absorbing states `env.get_absorbing()`, or any other `env` attribute not mentioned above, because your implementation will not be correct and will not be awarded full credit.

You can define any additional methods inside the `TD_agent` class, but only the `solve()` method will be called for automatic marking. **DO NOT** add any inputs or outputs to `solve()`.

[Report] – 10 pts

a. State which particular TD learning algorithm you choose to solve the problem. Give and justify the value of any parameters that you set. **– 3 pts**

b. Provide the graphical representation of your optimal policy and optimal value function, which can be generated with the provided code to visualise policies and value functions. **– 2 pts**

c. Plot the learning curve of your agent: the total non-discounted sum of rewards (vertical axis) against the number of episodes (horizontal axis). The figure should show the mean and shaded standard deviation on the same graph across 25 training runs. **– 2 pts**

d. Do off-policy algorithms need to be greedy in the limit with infinite exploration (GLIE), in order to guarantee that the optimal action-value function is learned? You do not need to run experiments or provide plots, instead answer and give your reasoning based on your understanding of the lecture material. **– 3 pts**