IMPERIAL

Department of Bioengineering

Reinforcement Learning for Bioengineers (BIOE 70077)

Acknowledgements

Thanks to **Professor Aldo Faisal** and his team, for preparing parts of programming tasks enclosed in this coursework.

Submission Components

1. Report (PDF):

File Name coursework1_report_CID.pdf
(replace **CID** with your College ID).

2. Code (Python):

File Name: coursework1.py
Base this code on the provided template, and ensure it contains the final code used to generate your results.

Important: Submission Nov 14th by 3 PM on Blackboard. Emailed or printed submissions cannot be accepted.

Report Guidelines

- Maximum of 8 single-sided A4 pages (or 10 including any references). Shorter reports are acceptable. Appendices are not allowed.
- Margins: At least 2 cm on all sides.
- Font Size: Minimum 11pt.
- Format: PDF format only (you may typeset using LaTeX or Word).
- Answer questions marked **[Report]** in the report.
- Answer questions marked **[Notebook]** in the Jupyter Notebook only (do not include these answers in your report).

Note: Reports that do not meet these guidelines may be penalised by up to 20 marks.

Code Submission Guidelines

- **Function Inputs**: Do not add non-optional inputs to any functions or methods, as this can disrupt automated grading.
- **Self-contained Code**: Ensure all functions and classes are fully self-contained and do not rely on global variables.
- **Class Attributes**: Only access the class attributes explicitly permitted in the instructions.
- **Modifiable Sections**: Modify only the classes and functions specified.
- **Code Reuse**: You may reuse code from lab assignments, but make sure the final submission is your own work.

Handling Experiment and Plotting Code

- Stall any code that runs experiments or generates plots before submitting.
- **Option**: Wrap experiment and plotting code in a conditional block:

```
if __name__ == '__main__':
                # Experiment code
```

This prevents experiments and plots from running when the code is imported for grading.

Placement of Notebook Tags

- Code for questions marked **[Notebook]** should appear **before** the if __name__ == '__main__': block so that grading scripts can access them.

Final Import Check

Verify that importing coursework1.py does not trigger any lengthy experiments (such as hyperparameter tuning) or run any plotting code. Failing to prevent these actions upon import could result in a penalty of 10 marks.

You may discuss coursework questions with others, but **all answers must be in your own words** to show your understanding. Both report and code will be checked for plagiarism.

Answers: Must be clear, complete, and concise.

Figures: Must be readable, labelled, and captioned.

Unclear or incomplete answers, poor figures, or irrelevant content will lose marks. Question marks are indicative, with a maximum score of 50. This coursework counts for **50% of your final grade**.

For questions, use computer labs or EdStem. GTAs are available to help but cannot give direct answers.

Maze Environment Overview

In bioengineering, many complex problems, like optimising a drug delivery system to target specific cells, can be modelled as a maze. The goal is to maximise drug efficacy while minimising side effects, navigating biological obstacles along the way.

- **State Representation**
  - **Maze Layout**: Each position in the maze represents a state in the drug delivery path, with each cell corresponding to a specific body location, like tissues or target cells.
  - **Obstacles**: Black squares represent biological barriers (e.g., cell membranes), making the environment more challenging for the drug to reach its target.
- **Rewards and Absorbing States**
  - **Rewards**: Dark-grey squares indicate absorbing states with rewards, such as successfully reaching target cells (e.g., tumour cells).
  - **Absorbing States**: These terminal points represent where the drug has been delivered and becomes inactive, like binding irreversibly to a target receptor.
- **Reinforcement Learning Application**
  - **Defining Actions**: Possible moves (up, down, left, right) simulate how the drug navigates through the biological environment.
  - **Learning Policy**: The RL agent aims to find the best actions from each state to maximise total rewards, learning through trial and error to discover the most effective drug delivery paths.
  - **Training the Model**: By simulating scenarios, the RL algorithm improves its strategy, adjusting based on results (e.g., hitting an obstacle or reaching a target cell) to develop an efficient drug delivery route.

This environment is similar to the Grid World from Lab Assignment-2, so feel free to reuse any code from that session. However, unlike the tutorial, this coursework focuses on reinforcement learning techniques that do not rely on knowing the environment's full dynamics, such as Monte Carlo and Temporal-Difference methods. Here, we assume we do not have access to the transition matrix T or the reward function R. Instead, the goal is to find the optimal policy π by sampling episodes within this maze environment. The agent can choose from four actions at each time step:

a_0: *Move North* (↑) a_2: *Move South* (↓)

a_1: *Move East* (→) a_3: *Move West* (←)

- When the agent selects an action, there is a probability p that it will move in the intended direction. In other words, with probability p, the agent will successfully go in the chosen direction (e.g., moving North if it chose "Move North"). However, if the action fails—which occurs with probability $1-p$, the agent will instead move in one of the other three directions, each with an equal chance.

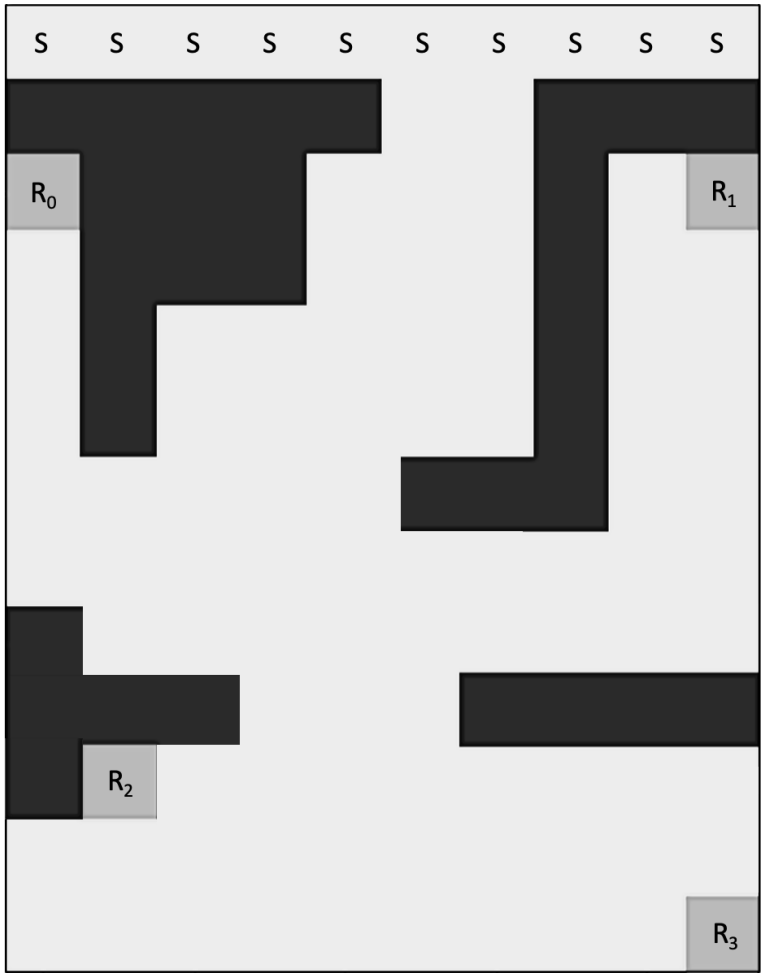

Figure 1: Illustration of the Maze environment, which is central to this coursework. Absorbing states, marked as "Ri," represent specific absorbing rewards relevant to bioengineering scenarios. Cells marked "*S*" indicate possible starting points within the maze.

- For instance, if the agent chooses a_0: *Move North (↑)*, it will move North with probability p. If it does not move North (with probability 1−p1), it has an equal probability of moving East, South, or West. In this case, the probability of moving in each of these alternate directions is $\frac{1-p}{3}$

- The parameter p is a hyperparameter that controls the environment's predictability or "noise": a higher p makes actions more likely to succeed in the intended direction, while a lower p introduces more randomness in the movement outcomes.

- Every time the agent takes an action in the environment, it receives a reward of −1. This means that each move has a small "cost" or penalty, regardless of the direction the agent moves. The goal, then, is likely to find the shortest or most efficient path, as each step adds to the total penalty.

- The episode will end when the agent reaches any of the absorbing states, $R_{i\ (for\ i=0,1,2,\ 3)}$, or if the agent takes more than 500 steps.

- At the start of each episode, the agent's initial position is chosen randomly from a set of possible starting states, which are marked as "*S*" on the maze diagram. Each starting state has an equal probability of being chosen, given by $1/N_{start\_states}$ that is the total number of possible starting positions.

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

- The probability p is set as $p = 0.8 + 0.02 \times (9 - y)$
- The discount factor γ is set as $\gamma = 0.8 + 0.02 \times y$

– The reward state is determined by taking the number z and dividing it by 4, using the remainder to choose the state. Meaning:

  If z is 0, 4, or 8 (since these give a remainder of 0), the reward state is $\boldsymbol{R_0}$
  If z is 1, 5, or 9 (remainder of 1), the reward state is $\boldsymbol{R_1}$
  If z is 2, 6, or 10 (remainder of 2), the reward state is $\boldsymbol{R_2}$
  If z is 3, 7, or 11 (remainder of 3), the reward state is $\boldsymbol{R_3}$
  In short, the reward state is $\boldsymbol{R_i}$ where **i** is the remainder when z is divided by 4.

The reward state R_i gives a reward $r = 500$ and all the other absorbing states gives a re- ward $r = -50$. As stated before, any action performed by the agent in the environment also gives 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

This sets up the main structure of the Maze environment and guides you on progressively completing functions in the Maze class that are marked "**[Action required]**".

Notebook Structure

– get_CID() and get_login(): Fill in these functions with your CID and login details. They are essential for auto-marking.

– GraphicsMaze class: Handles all graphical visualisation of the Maze. You do not need to understand or modify this class, just call its methods as needed.

– Maze class: The main Maze class. You'll complete specific functions in this class. Note:

  - Attributes and methods starting with an underscore (e.g., env._T or env._build_maze()) are private and should not be accessed outside the class.
  - Use "get" functions like get_T() to access these private attributes if needed.

– DP_agent class: Defines a Dynamic Programming agent for solving the Maze. You'll complete parts of this class in upcoming questions.

– MC_agent class: Defines a Monte Carlo Learning agent for the Maze. You'll be asked to complete parts of this class.

– A template structure given for comparative analysis, but feel free to change. Make sure this section (Question 3) does not run for auto-marker to evaluate. This may be reviewed by GTAs separately.

Question 1: Dynamic programming (DP) – 15 pts

You 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.

Fill in the following parts of the **Jupyter** [Notebook]: - 5pts

- 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. **(1 pts)**

- The_init_() method of the Maze class, to set you personalised reward state, absorbing reward, p, and γ in the environment. **(1 pts)**

Begin by using a DP agent, assuming access to the environment's transition matrix T and reward function R.

Complete the solve() method in DP_agent. Choose any DP method and reuse lab code if needed. **(4 pts)**

Allowed Resources: env.get_T(), env.get_R(), env.get_absorbing(), env.get_action_size(), env.get_state_size(), and env.get_gamma()

Rules: Only solve() will be marked; additional methods are allowed. **Do not** add inputs or outputs to solve().

Include in the [Report] – 9 pts

1. **Method and Parameter Justification:** Describe and justify your chosen DP method, including parameters, design choices, or assumptions. Cite literature reference(s) to back up your choice. **(2 pts)**
2. **Graphical Representation**: Include visuals of your optimal policy and value function. For policy, provide a grid with arrows representing the optimal moves for each state. For value function, generate a heatmap where darker shades represent higher values in the value function **(2 pts)**
3. **Effect of gamma γ and p**, along with heat-maps

    - Explain how values of gamma γ and p impact your optimal policy and value function. **(2 pts)**
    - Consider cases: **p<0.25p; p=0.25; p>0.25**; also consider: **γ<0.5 and γ>0.5** **(3 pts)**
      For example, For p=0.85, you might observe smoother paths as the agent's moves closely align with the chosen policy. For $\gamma < 0.5$, the agent may focus on shorter-term rewards, which could lead to suboptimal paths.

Extended Task – Chance to earn extra marks

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

Consider how assigning a reward that reflects the probability of reaching the goal might impact value convergence, as states close to the goal would yield higher values. **(2 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 wellas 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

1. State which method you choose to solve the problem and justify your choice. Give andjustify any parameters that you set, design that you chose or assumptions you made. **(3 pts)**

2. 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. **(3 pts)**

3. Plot the learning curve of your agent: the total non-discounted sum of rewards (vertical axis) against the number of episodes (horizontal axis). Include the mean and standard deviation of total rewards over 25 runs. Use shaded regions or error bars to represent standard deviation. Meaning, your plot should show total rewards (y-axis) versus episodes (x-axis), with shaded areas indicating the variability across runs

4. Standard deviation on the same graph across 25 training runs. Plot the learning curve of your agent: the total non-discounted sum of reward against the number of episodes. The figure should picture the mean and standard deviation on the same graph across 25 training runs. **(4 pts)**

Extended Task – Chance to earn extra marks

5. Consider an implementation which ignores trajectories that dont 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 concept.

   Meaning, how would ignoring episodes over 500 steps affect the completeness of value estimates? Could this limit introduce a bias toward states that are closer to terminal states? **(2 pt)**

Notes:

- Please note that all figures should be reproducible from your code if there is a need to check deeper
- You should be able to answer any questions and explain your code if invited for an interview
- You will only earn extra marks if you have lost in the corresponding questions

Question 3: Self-directed Research and Comparative Analysis – 20 pts

For this question, you'll compare the effects of three exploration strategies for **Question 2** and analyse their impact on learning outcomes. You will analyse MC performance using three exploration strategies—Fixed Epsilon, Decaying Epsilon, and Softmax

1. **Exploration Strategy Comparison**: **[Notebook]** - **4 pts**
    - Implement and compare the exploration strategies:
        - **Epsilon-greedy**: Where actions are chosen based on an epsilon-greedy policy. Use two variations:
            - **Fixed Epsilon**: Set a constant epsilon (e.g., $\varepsilon = 0.1$). **(1 pt)**
            - **Decaying Epsilon**: Gradually reduce epsilon over time (e.g., $\varepsilon =$ 1/episode_number). **(1 pt)**
        - **Softmax Approach**: Where action probabilities are proportional to their Q-values, adjusted with a temperature parameter. Higher temperatures result in more exploration, while lower temperatures lead to more exploitation. **(2 pt)**
2. **Analysis Requirements**: **[Report]** - **6 pts**
    - **Learning Curves**: Plot and compare learning curves for each exploration strategy (Fixed Epsilon, Decaying Epsilon, Softmax). The curves should show total rewards per episode with the mean and standard deviation over multiple runs to visualise differences in convergence rates and stability**.** **(2 pt)**
    - **Final Policy and Value Function**: Visualize and compare the final policy and value function for each strategy after a set number of episodes.
        - **Figure 3.1: Optimal Policy Grid**: Use arrows to show the optimal moves for each cell, with black cells representing obstacles. **(2 pt)**
        - **Figure 3.2: Optimal Value Function Heatmap**: Show the heatmap of value functions, where darker cells indicate higher values. **(2 pt)**
3. **Comparative Analysis**: - **10 pts**
    - **Explain the Chosen Strategy**: Describe which strategy you believe is optimal for this environment, based on the learning curves and final policy results. **(2 pt)**
    - **Discuss Observed Differences**:
        - Evaluate the impact of each strategy on learning speed, stability, and policy optimality. **(2 pt)**
        - Explain how the exploration-exploitation balance (controlled by epsilon and temperature parameters) affects the agent's ability to explore efficiently. **(2 pt)**

Cite at least four reference from key authors in Harvard style to support your explanation, particularly focusing on theoretical insights into exploration strategies in reinforcement learning. Also, when illustrating your anaysis, make use of a "*literature table*", and graphs in the context of ***Epsilon-Greedy vs Softmax***.

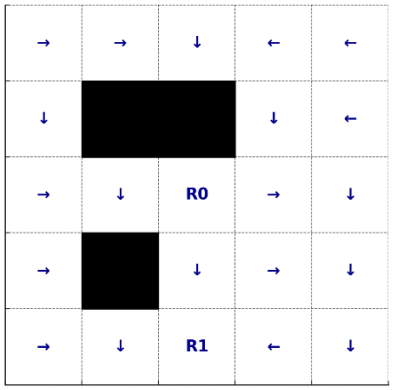

Figure 3.1: Example Optimal Policy Grid (Arrows Indicate Best Moves): This grid shows the optimal actions for the agent from each cell, represented by arrows. Arrows point in the direction the agent should move to reach absorbing states efficiently. Black cells mark obstacles that block movement.

Figure 3.2: Example optimal Value Function Heatmap (Higher Values are Darker): This heatmap visualises the value function across the maze. Darker cells indicate higher values, guiding the agent towards areas with greater reward potential, closer to the goal states.

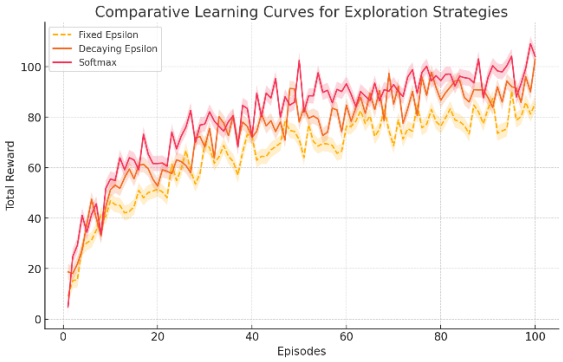

Figure 3.3: Example comparative Learning Curves for Exploration Strategies: Evaluating Fixed Epsilon, Decaying Epsilon, and Softmax approaches, showing total rewards per episode with mean and standard deviation, highlighting differences in convergence rates and stability across strategies.

Figures 3.1 and 3.2 (for final policy and value function) should be presented after the learning curve analysis and could incorporate insights from sensitivity analysis instead of requiring them separately.

Sensitivity Analysis: Vary the number of episodes and observe how sensitive the Monte Carlo returns and resulting policy are to the total number of episodes. Do this only for 1 approach among the three listed. For example, compare the results of running the agent for 25 episodes versus 200 versus 1000 episodes. **(2 pt)**

Discuss the effects on policy convergence, learning speed, and stability of the value function when the number of episodes is adjusted. Plot the learning curves and policy obtained at episode counts. **(2 pt)**

Please note that the marks allocated to each component do not necessarily reflect the difficulty of the task. Rather, they ensure that students of all skill levels have the opportunity to earn marks, as this is a very diverse cohort in terms of background and pre-requisite knowledge. Achieving higher marks, however, will require significantly more effort and depth of understanding.

**** End of the coursework document ****