

Midterm date: April 27th, 2023

Course title: Introduction to reinforcement learning and control

Course number: 02465

Aids allowed: All aids allowed

Midterm duration: 2 hours

Weighting: The exam is divided into 3 parts:

- Multiple-Choice questions
- Conceptual questions
- Programming questions

The overall scores of each part contribute roughly equally towards the overall result. Each question in each part contribute equally towards the score of that part.

Part I: Questions 1-4 are multiple choice. The score of a correct answer is 3 points. The score of an incorrect answer is -1 points. The score of option E or blank is 0 points.

Part II and part III: Each completed sub-task contribute towards your score.

Preparing your handin: The three parts are prepared as follows:

Part I: Edit the file `irlc/exam/midterm2023b/multiple_choice_answers.py`. Don't include calculations. Only answer with `'A'`, `'B'`, `'C'`, `'D'`, `'E'`.

Part II: Create a PDF file with your answers and justifications.

Part III: Program your answer in the `.py`-files indicated in the question and run `irlc/exam/midterm2023b/midterm2023b_tests_grade.py` to generate your `.token`-file.

Handing in: To hand in, you should upload the files:

- The `irlc/exam/midterm2023b/multiple_choice_answers.py`-file with your answer to part I
- The `.pdf`-file with your answers to part II
- The `.token`-file with your answers to part III
- The `irlc/exam/midterm2023b/question_td0.py` source file containing your solution
- The `irlc/exam/midterm2023b/question_mdp.py` source file containing your solution

Note on part II: The main quantities asked for are highlighted as $f(x)$. Answer unambiguously, concisely, and if applicable with algebraic simplifications. Your final result must be clearly indicated at the end of your answer: $f(x) = 3 \sin(x)$. To get credit, you must state the relevant theory and equations, and all relevant calculations must be included. Credit is not given for answers with missing or erroneous justifications.

Note on part III: To get started, move the folder `irlc/exam/midterm2023b` from the `.zip` file to the corresponding location in your existing exercise directory. The `.py` source files must be **reproducible** and **readable** so that someone else can run and fully understand your solution. You can freely use the `irlc`-toolbox (including solutions) and the packages we have used in the course, but you **must** include additional code you write during the exam or have prepared beforehand in the source files listed above. The source files must not be renamed. The `.token` file contains your results and must be up to date with your source files, i.e., generate it just prior to handin. In the case they differ, the `.token` file takes precedence. Credit is given for correct implementations defined by the problem description. The points in the `.token` file name is computed from the public tests, and might not correspond to overall correctness.

Part I: Multiple-Choice

Question 1:

Which one of the following options are correct?

- A. It is true that $\sum_a \pi(a|s)q_\pi(s, a) = v_\pi(s)$
- B. It is true that $\max_a \pi(a|s)q_\pi(s, a) = v_*(s)$
- C. Sarsa is an example of an off-policy reinforcement learning method
- D. Q -learning can only be applied to problems which eventually terminate
- E. Don't know.

Solution: The correct answer is A.

- A. The first equation is true by the definition of the value function (see chapter 3 in [SB18])
- B. This expression is meaningless
- C. Sarsa is on-policy, as it requires the subsequent action a' to be generated by the current policy
- D. Q -learning only use local information for its updates and can therefore be applied to any MDP

Question 2:

Consider an infinite-horizon Markov Decision Process (MDP) where the actions can take values in a set \mathcal{A} , the states can take values in a set \mathcal{S} , and rewards can take values in a set \mathcal{R} . All of these sets are finite, you can assume $0 < \gamma < 1$, and π stands for any policy.

Which one of the following statements are true in general?

- A. $v_\pi(s) < \infty$ (i.e., v_π is finite for all policies π)
- B. $v_\pi(s) \geq \min \mathcal{R}$
- C. $p(s', r|s, a) \neq 0$ for all $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ and $r \in \mathcal{R}$ where p is the transition probability.
- D. $\mathbb{E}[R_{t+1}|S_t = s, A_t = a] = q_*(s, a)$
- E. Don't know.

Solution: The correct answer is A.

- A. Since $\gamma < 1$, and there are only a finite number of different rewards, the value function must be finite as future terms are discounted by an exponentially vanishing factor
- B. This is true if the rewards are all positive, but if they are negative the value function can be a sum of many negative rewards which can obviously be smaller than the smallest single reward
- C. False; consider for instance a gridworld where s and s' are in opposite ends of the maze
- D. False since the Q function is determined as the sum of all future rewards

Question 3:

Consider a k -armed bandit problem with $k = 3$. In time step t , the payoff of selecting an arm $a \in \{0, \dots, k-1\}$ is:

$$r_t \sim \mathcal{N}(\mu = q_a^*, \sigma^2 = 0.64)$$

(i.e., the same type of reward distribution as in [SB18, Chapter 2]). The values of q^* are:

$$q_0^* = -1.7, \quad q_1^* = -4.6, \quad q_2^* = 0.3$$

Question 3 continues on the next page...

Consider a fixed policy π which with probability $1 - \epsilon$ selects the optimal arm and with probability $\epsilon = 0.1$ selects arm $a = 0$, i.e. the one associated with q_0^* above. What is the average per-step reward, i.e. $\mathbb{E}_\pi[R_t]$, of this policy?

- A. -1.5
- B. 0.065
- C. 0.44
- D. 0.1
- E. Don't know.

Solution: The correct answer is D.

The optimal arm is $a^* = 2$, and the average reward when a specific arm is selected is given by q_a^* . This means: The expected reward of this policy is therefore:

$$\mathbb{E}_\pi[R_t] = \sum_a \mathbb{E}[R_t|a]\pi(a) = (1 - \epsilon)\mathbb{E}[R_t|a = a^*] + \epsilon\mathbb{E}[R_t|a = 0] = (1 - \epsilon)0.3 + \epsilon(-1.7) = 0.1$$

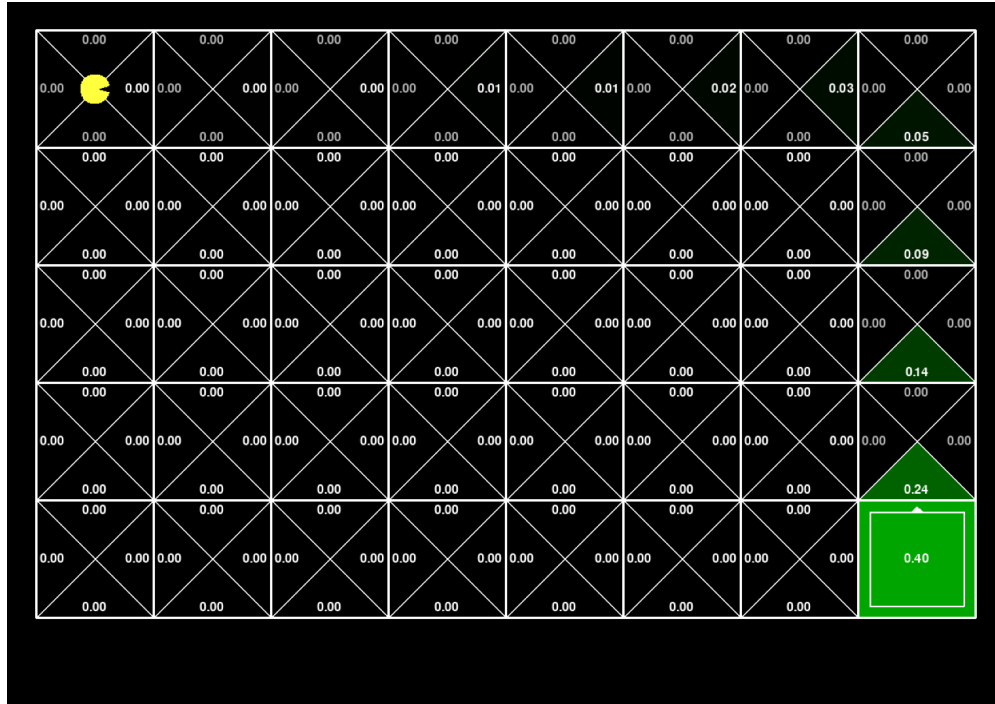


Figure 1: Monte-Carlo applied to a gridworld environment.

Question 4:

We consider the familiar gridworld environment discussed in [Her25, section 4.2.4] shown in fig. 1. The agent only receives a reward of +1 on completion and otherwise no reward. We train a first-visit Monte-Carlo agent on the gridworld for one episode and fig. 1 shows the resulting Q-values. What value of the discount factor γ was used?

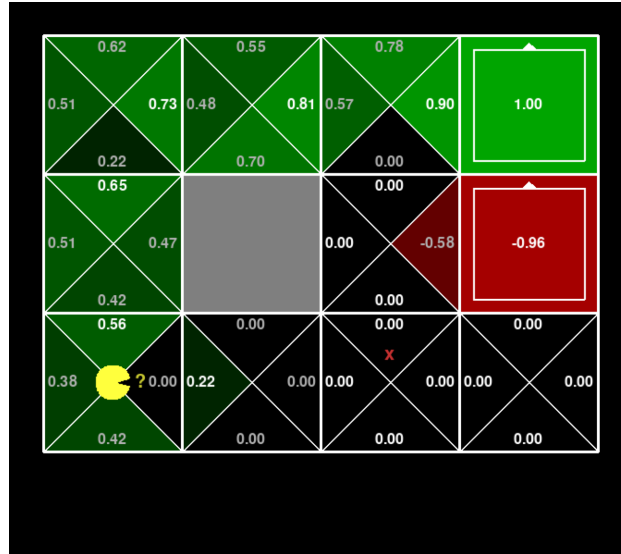
- A. $\gamma = 0.5$
- B. $\gamma = 0.4$
- C. $\gamma = 0.6$
- D. $\gamma = 0.3$
- E. Don't know.

Solution: The correct answer is C.

The Monte-Carlo control method updates the Q -values towards the future γ -discounted return obtained in a given state action pair. Since there is only a terminal reward, the γ -discounted return will be $G = \gamma^k$ where k is the number of steps to the goal; this means that in a single episode, the Q -values are:

$$Q(s, a) = \alpha \gamma^k$$

More to the point, it means that the ratio between two subsequent Q -values is simply $\frac{Q(s', a')}{Q(s, a)} = \frac{\alpha \gamma^{k+1}}{\alpha \gamma^k} = \gamma$. Picking any two Q -values will give the right answer of $\gamma = 0.6$

Figure 2: An example gridworld environment with Q -values

Part II: Conceptual questions

Question 5:

Suppose that Q -learning (using discount factor $\gamma = 0.9$ and learning rate $\alpha = 0.8$) is applied to the Gridworld MDP shown in fig. 2. The living reward is 0, and the dynamics is deterministic. The agent can only move to a terminal state from one of the two states to the right (with the inserted squares), and the current state s_t of the agent is as indicated in the figure.

- (a) The agent has 4 possible actions in the current state. It is possible the next state s_{t+1} of the agent will be the one just to the north of its current location. Express, as a function of the learning rate ϵ , the probability that this will be the case.

Solution: When the agent is acting greedily it will always move North. When the agent is acting randomly, it will move north with probability $\frac{1}{4}$. The chance of moving north is thus

$$(1 - \epsilon) \cdot 1 + \epsilon \frac{1}{4} = \frac{4 - 3\epsilon}{4}$$

- (b) Consider again the state shown in fig. 2 and assume that the agent takes the action **east**, after which the agent transition to the state immediately to the east of its present location. This transition will update the Q -value $Q(s_t, \text{east})$ indicated by the yellow question mark. What will be the new value of $Q(s_t, \text{east})$?

Solution: The immediate reward is 0, and the largest Q -value in the next state is $\max_{a'} Q(s', a') = 0.22$. Using that the old Q -value is zero the new Q -value is therefore updated to

$$Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) = \alpha \gamma (\max_{a'} Q(s', a')) = 0.72 \times 0.22 = 0.1584$$

- (c) Assume we keep the exploration rate fixed at $\epsilon = 0.1$, but that the learning rate decrease as $\alpha \propto \frac{1}{n}$ to satisfy the usual convergence criteria for Q -learning ([SB18, Equation 2.7]). After convergence, what is the Q -value indicated by the red cross?

Question 5 continues on the next page...

Solution: Since Q -learning is an off-policy method the Q -values will converge to the value they obtain under the optimal policy. Clearly the optimal policy just moves to the goal in 3 steps, and due to the discount factor this will be discounted by a factor of γ^3 . The solution is thus

$$\gamma^3 = 0.729.$$

	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$
A_t	2	1	1	1	0	2
R_t	1	-3	2	1	1	-1

Table 1: Sequence of arm-selections and rewards used to train the bandit algorithm

Question 6:

Consider the simple bandit algorithm described in [SB18, Section 2.4]. Suppose there are $k = 3$ arms, labelled as $A \in \{0, 1, 2\}$ and that table 1 contains the outcome of trying six actions and observing the resulting reward

- (a) Given these observations, what is Q -value associated with arm $a = 1$, $Q(1)$?

Solution: The Q -value is simply the sum of reward obtained by trying arm 1 divided by the number of times it was tried. Thus:

$$Q(1) = \frac{-3 + 2 + 1}{3} = 0.0.$$

- (b) For this question, consider the non-stationary bandit algorithm described in [SB18, Section 2.5] which uses the modified update rule:

$$Q_{n+1} = Q_n + \alpha [R_n - Q_n]$$

where Q_n is the Q -value of the selected arm and α is the learning rate. Assume this bandit algorithm is still trained on the data in table 1. Determine the Q -value of arm $a = 2$, $Q(2)$ as a function of α .

Solution: The Q -value is first initialised to $q_0 = 0$. It is then updated to $q_1 = 0 + \alpha(1 - 0) = \alpha$, and finally to $q_2 = \alpha + \alpha(-1 - \alpha) = -\alpha^2$.

- (c) Consider again the non-stationary bandit algorithm described in [SB18, Section 2.5]. Assume we exclusively focus on a specific arm a . During training, we try this arm m times, and observe m corresponding rewards R_1, R_1, \dots, R_m .

The non-stationary bandit algorithm initialize the Q -values to $Q_1(a) = 0$ by default, however, we consider an alternative method which initialize them to $Q_1(a) = q$ for a constant q .

The initial value of q will affect the final value of $Q_{m+1}(a)$ after training on the m rewards, and therefore we write the final Q -values after training as $Q_{m+1,q}(a)$. Determine an analytical expression of $h(q, m)$ defined as the difference:

$$h(q, m) = Q_{m+1,q}(a) - Q_{m+1,q=0}(a).$$

The expression should be simple, i.e. not involve sums of many terms etc.

Solution: Assume $Q_{n+1} = (1 - \alpha)^n q + b$ where b is independent for q . This is true for $n = 0$ where $Q_1 = q$. Now:

$$Q_{n+1} = Q_n + \alpha(R_n - Q_n) = (1 - \alpha)Q_n + \alpha R_n = (1 - \alpha)((1 - \alpha)^{n-1} q + b) + \alpha R_n = (1 - \alpha)^n q + b'.$$

The difference cancels out the constant factor and we get:

$$h(q, m) = (1 - \alpha)^m q.$$

A simpler (but completely valid) solution is to adapt the approach from equation 2.6 from [SB18].

Part III: Programming questions

Question 7:

To solve this question, you should edit the file `irlc/exam/midterm2023b/question_td0.py`. In this problem, we will consider the TD(0) method described in [SB18, Section 6.1]. In particular, we will consider how TD(0) change the value function as a result of a single episode as discussed in [SB18, Example 6.1].

To this end, recall that the TD(0) algorithm, as defined in the beginning of [SB18, Section 6.1], is comprised of:

- Initializing the value function v to $v(s) = 0$
- For each episode:
 - loop over the time steps t of the episode and perform an update of v involving the current state s_t , reward r_{t+1} , discount factor γ and learning rate α

We are concerned with the last bullet point. In other words, for a given episode comprised of a list of states (s_0, s_1, \dots, s_T) and list of rewards (r_1, r_2, \dots, r_T) , we can compute the full update of the value function v resulting from these observations. This is the view we will take in this exercise.

We will as usual represent the value function as a dictionary, where the keys `s` are states s , and the values `v[s]` is the value-function $v(s)$. It will be pre-initialized to zero in the test code. The states will be integers and the rewards floating point numbers.

- (a) Complete `def a_compute_deltas(v: dict, states: list, rewards: list, gamma: float)`: Given a value function v as a dictionary, lists of states and rewards corresponding to an episode, and the discount factor γ , the function should return a list comprised of the values:

$$(\delta_0, \delta_1, \dots, \delta_{T-1})$$

where δ_t is the TD error defined in [SB18, Equation 6.5].

- (b) Complete `def b_perform_td0(v: dict, states: list, rewards: list, gamma: float, alpha: float)`: Given inputs of the previously described format, as well as a learning rate α , the function should compute the TD(0) update to v resulting from a single episode and return the updated value function v (as a dictionary).
- (c) Complete `def c_perform_td0_batched(v: dict, states: list, rewards: list, gamma: float, alpha: float)`: We will now consider the batched view of TD(0) as described in [SB18, Section 6.3]. In the batched version, all changes to v that arise during an episode (i.e., the factors $\alpha\delta_t$ in [SB18]) are computed as usual using the *same* value function v , but the changes are only applied to v at the *end* of the episode all at once. The function should compute the batched update and return the updated version of v in the usual format.

Solution:

The solution can be found in this directory as a `.py` file.

Question 8:

To solve this question, you should edit the file `irlc/exam/midterm2023b/question_mdp.py`. This problem will consider a variant of the Gambler environment discussed in [SB18] which has been simplified as follow:

- The goal is 40 instead of a 100, and the initial state is $s_0 = 20$ instead of $s_0 = 50$
- There is a small betting fee, so that the reward of betting a dollars is $-\frac{a}{100}$
- The problem does not terminate. If you go bust, $s = 0$, or win, $s = 40$, only the action $a = 0$ is available and the problem no longer gives a reward.

The problem with these modifications have been implemented as the file `smalltime_gambler.py` in the exam folder using the MDP class. The changes makes the problem more manageable. In particular, you no longer have to deal with terminal states since there are none.

In the following, we will consider a couple of sub-optimal planning approaches which can be thought of as steps towards value iteration. You may assume that $\gamma = 1$.

- (a) Complete `def a_get_reward(s : int, a : int)`: Given a state $S_t = s$ and an action $A_t = a$, compute the reward function, i.e. the average of r_{t+1} . Formally, the function should return the number:

$$r(s, a) = \mathbb{E}[R_{t+1} | A_t = a, S_t = s] = \sum_{r, s'} P(s', r | s, a) r$$

Hint: Use the transition probabilities.

- (b) Complete `def b_get_best_immediate_action(s : int)`: Given a state $S_t = s$, compute the action which leads to the immediate best reward. I.e., compute and return:

$$a^* = \arg \max_a \mathbb{E}[R_{t+1} | A_t = a, S_t = s] = \arg \max_a r(s, a)$$

- (c) Complete `def c_get_best_action_twosteps(s : int)`: Our policy is myopic, meaning that it plan on a very short horizon of 1.

We can improve that by planning on a horizon of 2. Specifically, this function should compute and return the action a^* that maximize $R_{t+1} + R_{t+2}$:

$$a^* = \arg \max_a \mathbb{E} \left[R_{t+1} + \max_{a'} \mathbb{E} [R_{t+2} | S_{t+1}, A_{t+1} = a'] | A_t = a, S_t = s \right]$$

Hint: It may help you to notice that:

$$\arg \max_a \mathbb{E} \left[R_{t+1} + \max_{a'} \mathbb{E} [R_{t+2} | S_{t+1}, A_{t+1} = a'] | A_t = a, S_t = s \right] = \arg \max_a \sum_{r, s'} p(r, s' | a, s) \left(r + \max_{a'} r(s', a') \right).$$

Solution:

The solution can be found in this directory as a `.py` file.

References

[Her25] Tue Herlau. Sequential decision making. (Freely available online), 2025.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. (Freely available online).

This line concludes the exam. Document build: 2025/05/20, 16:13:24.