4180/5180: Reinforcement Learning and Sequential Decision Making (Spring 2026) | Robert Platt & Chris Amato
Northeastern University | Due Feb 18, 2025

Exercise 4: Monte-Carlo Methods

Please remember the following policies:

- Exercise due at **11:59 PM EST Feb 18, 2026**.
- Submissions should be made electronically on Canvas. Please ensure that your solutions for both the written and programming parts are present. You can upload multiple files in a single submission. Please do **not** zip them into a single file.
- You can make as many submissions as you wish, but only the latest one will be considered. Please do **not** make any submissions after the deadline unless you would like the submission to be considered as a late submission.
- For **Written** questions, solutions may be handwritten or typeset. If you write your answers by hand and submit images/scans of them, please please ensure legibility and order them correctly in a single PDF file.
- The PDF file should also include the figures from the **Plot** questions.
- For both **Plot** and **Code** questions, submit your source code in Jupyter Notebook (.ipynb file) along with reasonable comments of your implementation. Please make sure the code runs correctly. Please also submit a pdf of the Jupyter Notebook (.ipynb file).
- You are welcome to discuss these problems with other students in the class, but you must understand and write up the solution and code yourself. Also, you *must* list the names of all those (if any) with whom you discussed your answers at the top of your PDF solutions page.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 10% per day late. Submissions later than two days will not be accepted.
- Contact the teaching staff if there are medical or other extenuating circumstances that we should be aware of.
- Generative AI can only be used for finding bugs or polishing English—all code and solutions should be written up by the student.
- **Notations: RL2e is short for the reinforcement learning book 2nd edition. x.x means the Exercise x.x in the book.**
- Please note that questions 4(a) and 5 are required for 5180 students and extra credit for 4180 students.

1. **2 point.** (RL2e 5.2, 5.5, 5.8) *First-visit vs. every-visit.*
   Written:
   (a) Read the Example 5.1 in RL2e. Suppose every-visit MC was used instead of first-visit MC on the blackjack task. Would you expect the results to be very different? Why or why not?
   (b) Consider an MDP with a single nonterminal state and a single action that transitions back to the nonterminal state with probability p and transitions to the terminal state with probability $1-p$. Let the reward be $+1$ on all transitions, and let $\gamma = 1$. Suppose you observe one episode that lasts 10 steps, with a return of 10. What are the first-visit and every-visit estimators of the value of the nonterminal state?
   (c) **[Extra credit (1 point)]** Read and understand example 5.5 first. The results with Example 5.5 and shown in Figure 5.4 used a first-visit MC method. Suppose that instead an every-visit MC method was used on the same problem. Would the variance of the estimator still be infinite? Why or why not?
   Code/plot: Implement Example 5.5 and reproduce Figure 5.4 to verify your answer.

2. **2 points.** *Blackjack.*
   Code/plot:
   (a) Implement first-visit Monte-Carlo policy evaluation (prediction).
   Apply it to the Blackjack environment for the "sticks only on 20 or 21" policy to reproduce Figure 5.1.

Question 1

a) No, it wouldn't change much. In blackjack in a single episode you almost never visit the same state (player sum, dealer show, usable ace) because player sum typically increases with hit action. So first-visit and every-visit MC is essentially same for this problem mostly same states.

b) (Same as in-class MC exercise)

first visit MC: ($t = 0, 1, \dots 9$)

$V(S) = G_0 = 10$

every-visit MC:

$$V(S) = \frac{(1+2+3+4+\dots+10)}{10} = 5.5$$

$G_0 = 10 \ \dots \ G_9 = 1$

c) Infinite variance

If we used every-visit MC it would still have infinite variance since importance sampling have very heavy tail

<u>Code/plot:</u> (need to do it at end)

<u>Question 3:</u>

b) Written: Explain how results of $\epsilon = 0$ settings demonstrate importance of doing exploring starts in MC-ES

— without any exploration agent will pick only greedy actions. Since every new episode starts from (0,0) the agent might n visit all states. Thus it cannot update Q-values. So it gets stuck in some path gradually and it might miss optimal paths.

MC exploring starts fixes this by giving non-zero probability to each state-action pairs. This guarentees th every state-action pair will be visited in the limit. So MC-ES replaces ε-Greedy in the case of exploration

To sum up: ε=0 without exploring starts leads to poor learnig because the agent is stuck in Greedy trajectory. Exploring starts solve this by giving each state-action pair non-zero probability to start so agent can try differ conditions.

Q4) (RL 5.10, 5.11) off policy methods

eq 5.8 $V_{n+1} = V_n + \frac{W_n}{C_n}\left[G_n - V_n\right]$

a) Derive eq. 5.8 from 5.7 $\boxed{n \geq 1}$

$$V_n \doteq \frac{\sum^{n-1} W_k G_k}{\sum W_k}, \quad n \geq 2 \quad (\text{eq } 5.7)$$

Incremental version! (cumulative weight update)

$$C_0 \doteq 0$$

$$C_{n+1} \doteq C_n + W_{n+1}$$

$$\Rightarrow C_n \doteq \sum W_k$$

So: numerator $\sum_{k=1}^{n-1} W_k G_k = N_{n-1}$

denominator $\sum_{k=1}^{n-1} W_k = C_{n-1}$

and $V_n = \dfrac{N_{n-1}}{C_{n-1}}$, $V_{n+1} = \dfrac{N_{n-1} + W_n G_n}{\underbrace{C_{n-1} + W_n}_{C_n}}$

$$N_{n-1} = C_{n-1} \cdot V_n$$

$$\Rightarrow V_{n+1} = \frac{C_{n-1} \cdot V_n + W_n \cdot G_n}{C_{n-1} + W_n}$$

Let $C_n \doteq C_{n-1} + W_n$

$$V_{n+1} \doteq \frac{C_n V_n - W_n V_n + W_n G_n}{C_n}$$

$$\frac{C_n V_n}{C_n} + \frac{(-W_n V_n)}{C_n} + \frac{W_n G_n}{C_n}$$

$$= V_n + \frac{(-W_n V_n + W_n G_n)}{C_n}$$

$$= V_n + \left(\frac{W_n (G_n - V_n)}{C_n}\right)$$

$$= V_n + \frac{W_n}{C_n}(G_n - V_n), \quad n \geq 1$$

b) Because our policy is determnistic greedy one, we are only observing trajectories where $\pi(a|s) = 1$, hence numerator is 1. if $A_t \neq \pi(S_t)$ it exits loop. Meaning no more updates thats why $\frac{\Pi(a|s)}{b(a|s)} = \frac{1}{b(a|s)}$

Question 5

c) written

(b) Implement first-visit Monte-Carlo control with exploring starts (Monte-Carlo ES).
Apply it to the Blackjack environment to reproduce Figure 5.2.
Note that the reset mechanism already selects all states initially with probability > 0, but you must ensure that all actions are also selected with probability > 0.

Useful tools for implementation:

- Instead of writing your own Blackjack environment, we recommend that you use the implementation provided by Gymnasium (maintained fork of OpenAI Gym), or at least refer to it closely if you are re-implementing your own version.
  This would also be a good opportunity to start setting up and learning about the library.
- For installation instructions and a brief introduction: `https://gymnasium.farama.org`
- Once you have installed Gymnasium, you can instantiate the environment by calling:
  ```
  import gymnasium as gym
  env = gym.make("Blackjack-v1")
  ```
- For more specifics on the interface and implementation of Blackjack, see:
  `https://gymnasium.farama.org/environments/toy_text/blackjack`
  `https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py`
- To plot the value functions and policies, consider using: `matplotlib.pyplot.imshow`

3. **2 points.** *Four Rooms, re-visited.*
We are now finally ready to re-visit the Four Rooms domain from Ex0, now with better learning algorithms. We provide you with the implementation of the Four Rooms environment in the Jupyter Notebook and make the domain episodic to apply Monte-Carlo methods. The modifications are as follows:

- Instead of teleporting to $(0, 0)$ after reaching the goal, we make the goal a terminal state (i.e., end of episode). In other words, the episode terminates after the agent reaches the goal state (i.e. $(10, 10)$).
- We add a timeout to the episodes, i.e., an episode terminates after some maximum number of steps even the agent doesn't reach the goal. In the current implementation, we set T = 459.

(a) **Code/plot:** Implement on-policy first-visit Monte-Carlo control (for ϵ-soft policies). Let us solve the FourRooms problem with a fixed goal state = $(10, 10)$, which is initially unknown to the agent.
To verify the agent is learning, plot learning curves similar to those in Ex1.

- The horizontal axis should be in episodes; the vertical axis should be each episode's discounted return.
- Plot curves for $\varepsilon = 0.1, 0.01, 0$. For clear trends, running for 10 trials with 10^4 episodes within each trial is recommended, but if it is too time-consuming you may run less. You can use the provided plotting function to plot the curves. You can test your implementation by running for 5 trials with 10^3 episodes within each trial for each ϵ.

(b) **Written:** Explain how the results of the $\varepsilon = 0$ setting demonstrate the importance of doing exploring starts in Monte-Carlo ES.

4. **1 point.** (RL2e 5.10, 5.11) *Off-policy methods.*

(a) **[5180 only]** Derive the weighted-average update rule (Equation 5.8) from (Equation 5.7). Follow the pattern of the derivation of the unweighted rule (Equation 2.3).

(b) **Written:** In the boxed algorithm for off-policy MC control, you may have been expecting the W update to have involved the importance-sampling ratio $\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$, but instead it involves $\frac{1}{b(A_t|S_t)}$. Why is this correct?

5. **3 points.** **[5180 only]** (RL2e 5.12) *Racetrack.*
Consider driving a race car around a turn like those shown in Figure 5.5. You want to go as fast as possible, but not so fast as to run off the track. In our simplified racetrack, the car is at one of a discrete set of grid positions, the cells in the diagram. The velocity is also discrete, a number of grid cells moved horizontally and vertically per time step. The actions are increments to the velocity components. Each may be changed by $+1$, -1, or 0 in each step, for a total of nine (3×3) actions. Both velocity components are restricted to be nonnegative and less than 5, and they cannot both be zero except at the starting line. Each episode begins in one of the randomly selected start states with both velocity components zero and ends when the car crosses

the finish line. The rewards are -1 for each step until the car crosses the finish line. If the car hits the track boundary, it is moved back to a random position on the starting line, both velocity components are reduced to zero, and the episode continues. Before updating the car's location at each time step, check to see if the projected path of the car intersects the track boundary. If it intersects the finish line, the episode ends; if it intersects anywhere else, the car is considered to have hit the track boundary and is sent back to the starting line. To make the task more challenging, with probability 0.1 at each time step the velocity increments are both zero, independently of the intended increments.

(a) **Code:** Use the provided implementation of the Racetrack. Apply on-policy first-visit Monte-Carlo control (for ε-soft policies) to the racetrack domain (both tracks), with $\varepsilon = 0.1$ – ideally, this would be a simple application of the code from Q3(a).
Plot: For each racetrack, plot the learning curve (multiple trials with confidence bands), similar to Q3(a). Note that, trials number = 10, episode number per trials = 2000.

(b) **Code:** Implement off-policy Monte-Carlo control and apply it to the racetrack domain (both tracks). For the behavior policy, use an ε-greedy action selection method, based on the latest estimate of $Q(s, a)$ – i.e., this is similar to on-policy Monte-Carlo control, except that the target policy is kept as a greedy policy.
Plot: For each racetrack, plot the learning curve (multiple trials with confidence bands), similar to Q3(a). Plot the learning curve for the target policy, do this by collecting one rollout after each episode of training, which is collected solely for evaluation purposes. Visualize several rollouts of the optimal policy (i.e., the target policy); Consider using: `matplotlib.pyplot.imshow`

(c) **Written:** Do you observe any significant differences between the on-policy and off-policy methods? Are there any interesting differences between the two racetracks?

Tip: You can find NumPy arrays containing the racetracks in `Jupyter Notebook`.
Think about which racetrack you expect is easier, and develop your methods in that domain.

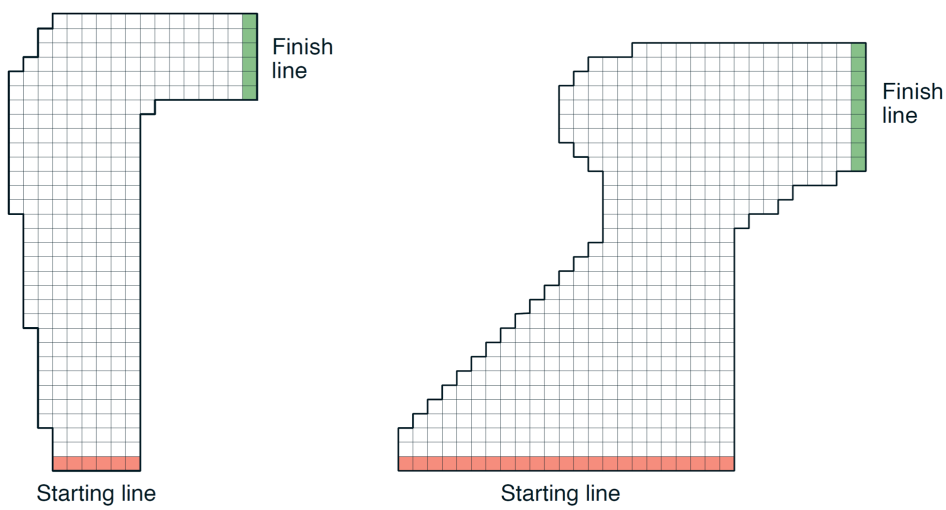

Figure 5.5: A couple of right turns for the racetrack task.