# [S&B Book] Chapter 5: Monte Carlo Methods

Tags

- In this chapter, we do not assume complete knowledge of the environment;
- **Monte Carlo methods** are ways of solving the reinforcement learning problem based on averaging sample returns. To ensure well-defined returns are available, we define Monte Carlo methods only for episodic tasks. Only on the completion of an episode are value estimates and policy changes.

## ▼ 5.1 Monte Carlo Prediction

- Expected return - expected cumulative future discounted reward starting from that state.
- As more returns are observed, the average should converge to the expected value. This idea underlies all Monte Carlo methods.
- A *visit* to $s$ - each occurrence of state $s$ in an episode is called a *visit* to $s$.
- **First-visit and every-visit MC method:**

  In an episode, $s$ may be visited multiple times:

  1. First-visit

     The first time $s$ is visited in an episode. The first-visit MC method estimates $v_\pi(s)$ as the average of the returns following first visits to $s$.

  2. Every-visit:

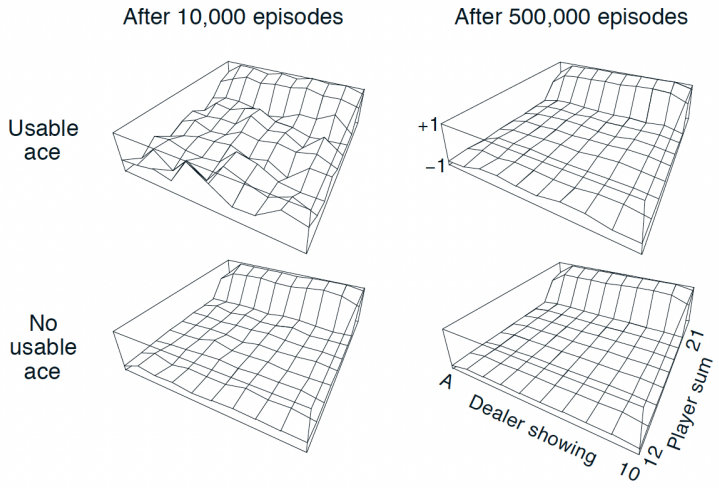
     The every-visit MC method averages the returns following all visits to $s$.
- First-visit MC prediction algorithm

**First-visit MC prediction, for estimating $V \approx v_\pi$**

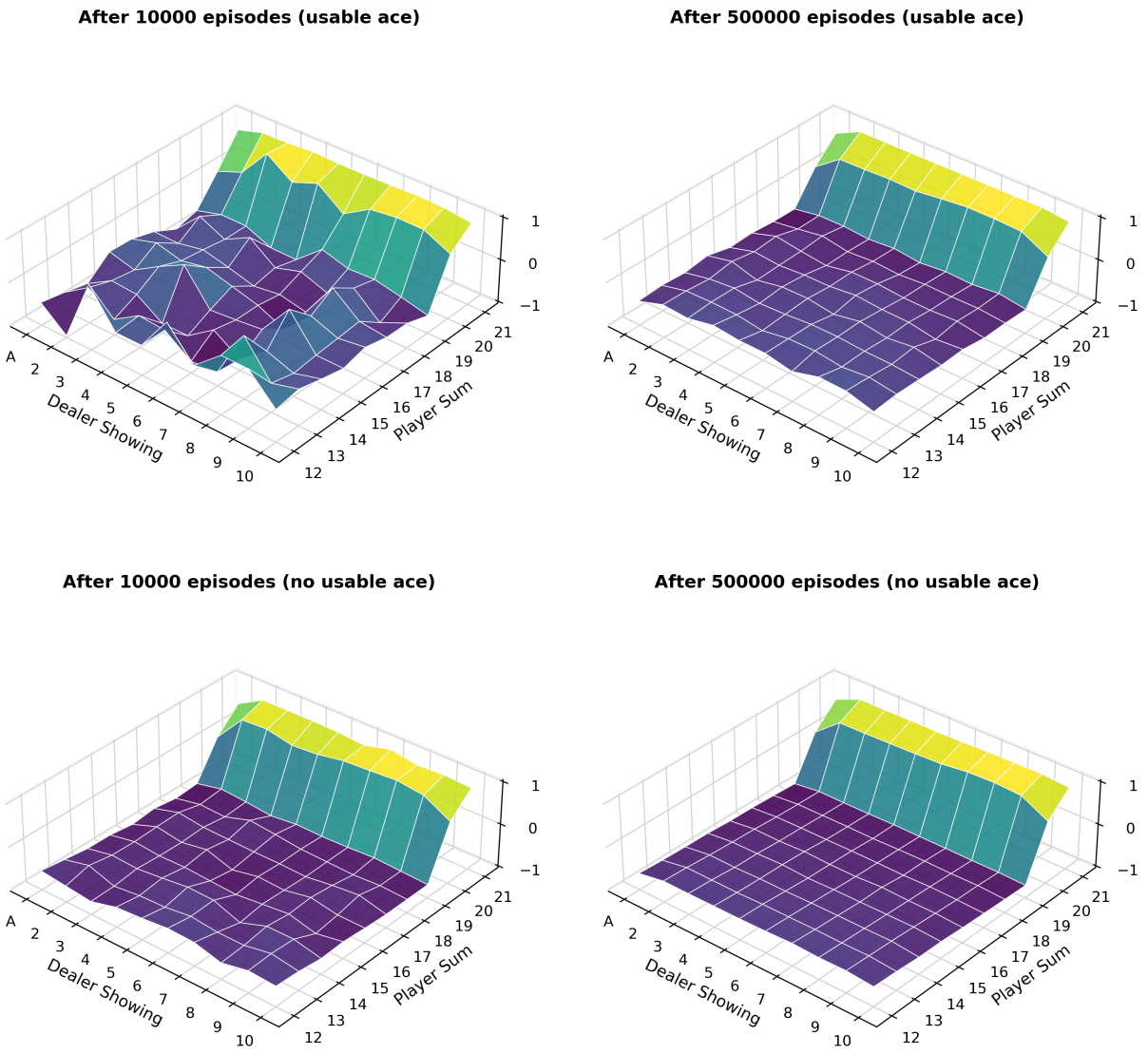
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless $S_t$ appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append $G$ to $Returns(S_t)$
            $V(S_t) \leftarrow \text{average}(Returns(S_t))$

**Example 5.1: Blackjack**




*Implementation:*




Reinforcement-Learning-2nd-Edition-Notes-Codes/chapter_05_monte_carlo_methods/example_5_1_blackjack.py at main · terrence-ou/Reinforcemer

Notes and code implementations of examples and algorithms of the book Reinforcement Learning, 2nd Edition - Reinforcement-Learning-2nd-Edition-Notes-Codes/chapter_05_monte_carlo_methods/example_5_1...

https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_05_monte_carlo_methods/example_5_1_blackjack.py

## ▼ 5.2 Monte Carlo Estimation of Action Values

- **One of the primary goals for Monte Carlo method is to estimate $q_*$**

  If the model is not available, then it is particularly useful to estimate *action* values (the values of state-action pairs) rather than *state* values. Without a model, state value alone are not sufficient. One must explicitly estimate the value of each action in order for the values to be useful in suggesting a policy

- ***exploring starts:***

Specifying that the episode *starts in a state-action pair*, and that every pair has a nonzero probability of being selected as the start.

## ▼ 5.3 Monte Carlo Control

- **Monte Carlo policy iteration:**

  After each episode, the observed returns are used for policy evaluation, and then the policy is improved at all the states visited in the episode.

- **Monte Carlo ES (Exploring Stares)** algorithm:

**Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$**

Initialize:
  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

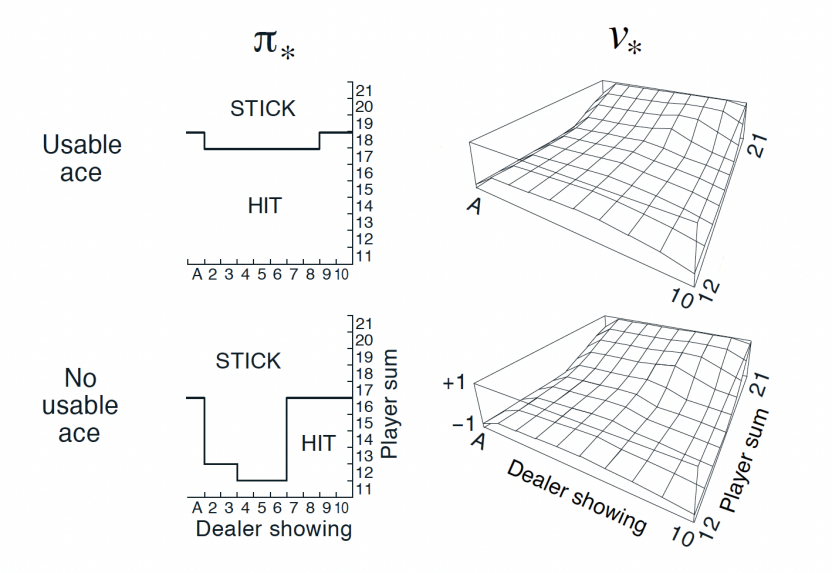
Loop forever (for each episode):
  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability $> 0$
  Generate an episode from $S_0, A_0$, following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append $G$ to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

In Monte Carlo ES, all the returns for each state-acton pair are accumulated and averaged, irrespective of what policy was in force when they were observed. It is easy to see that Monte Carlo ES cannot converge to any suboptimal policy. Stability achieved only when both the policy and the value function are optimal.

---

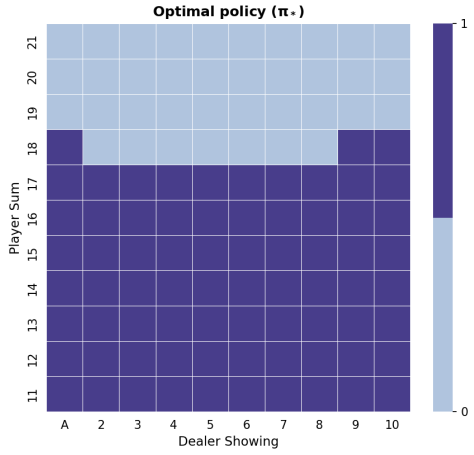
**Example 5.3: Solving Blackjack**

Apply Monte Carlo ES to blackjack. The initial policy is to stick only on the player's sum is 20 or 21, and the initial action-value function is zero for all state-action pairs.

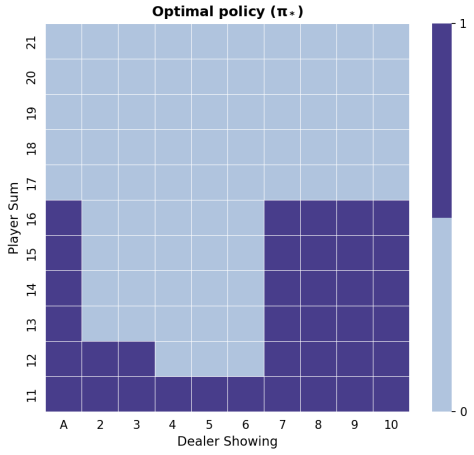


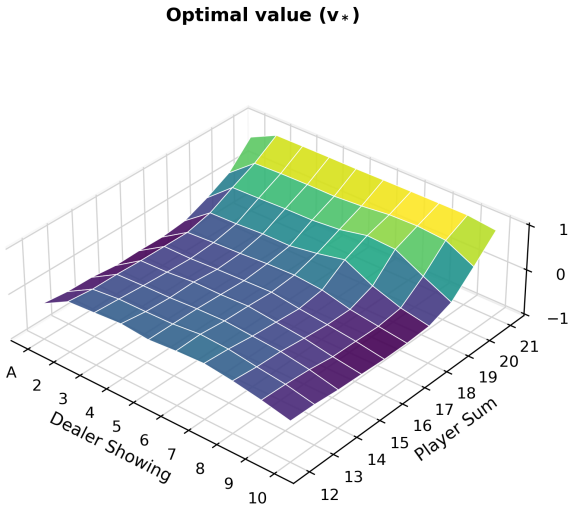

*Implementation:*

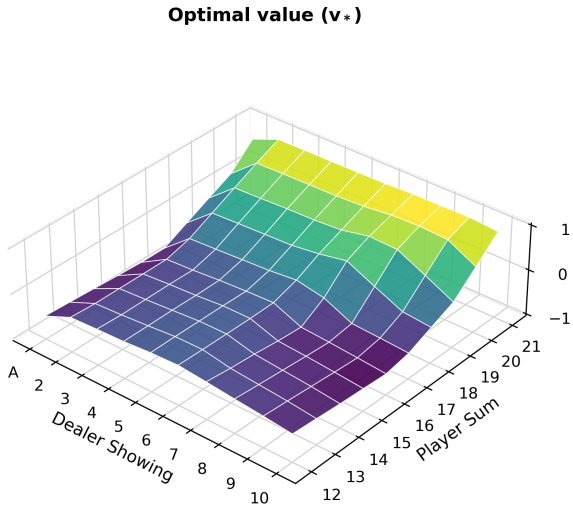
**Usable ace**




**No usable ace**






https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_05_monte_carlo_methods/example_5_3_solving_blackjack.py

## ▼ 5.4 Monte Carlo Control without Exploring Starts

- *On-policy* methods and *off-policy* methods:

  On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data.

- *soft* policy:

  In on-policy control methods the policy is generally *soft*, meaning that $\pi(a|s) > 0$ for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}(s)$, but gradually shifted closer and closer to a deterministic optimal policy.

- **On-policy first-visit MC control (for $\varepsilon$-soft policies), estimates $\pi \approx \pi_*$**

**On-policy first-visit MC control (for $\varepsilon$-soft policies), estimates $\pi \approx \pi_*$**

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append $G$ to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

## ▼ 5.5 Off-policy Prediction via Importance Sampling

> The dilemma of learning control methods: They seek to learn action values conditional on subsequent *optimal* behavior, but they need to behave non-optimally in order to explore all actions (to find the optimal actions).

For off-policy methods, the approach is to use two policies, one that is learned about and that becomes the optimal policy, and one that is more exploratory and is used to generate behavior.

- *Target policy:* The policy being learned about;
- *Behavior policy:* The policy used to generate behavior;

This method is generally learning from data "off" the target policy, and the overall process is termed *off-policy learning.*

- Assumption of *coverage*: In order to use episodes from $b$ to estimate values for $\pi$, we require that every action taken under $\pi$ is also taken, at least occasionally, under $b$. That is, we require that $\pi(a|s) > 0$ implies $b(a|s) > 0$.

**Almost all off-policy methods utilize *importance sampling*,** a general technique for estimating expected values under one distribution given samples from another.

- We apply importance sampling to off-policy learning by weighting returns according to the relative probability of their trajectories occurring under that target and behavior policies, called the *importance-sampling ratio*.

$$\begin{aligned} \Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T | S_t, A_{t:T-1} \sim \pi\} &= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k) \end{aligned}$$

The relative probability of the trajectory under of the trajectory under the target and behavior policy:

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

The ratio $\rho_{t:T-1}$ transforms the returns to have the right expected value.

In this section, the episode numbering time step in a way that increases across episode boundaries.

- ***Ordinary importance sampling***

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

$\mathcal{T}(s)$ denotes the set of all time steps in which state $s$ is visited;

$T(t)$ denotes the first time of termination following time $t$;

$G(t)$ denotes the return after $t$ up through $T(t)$.

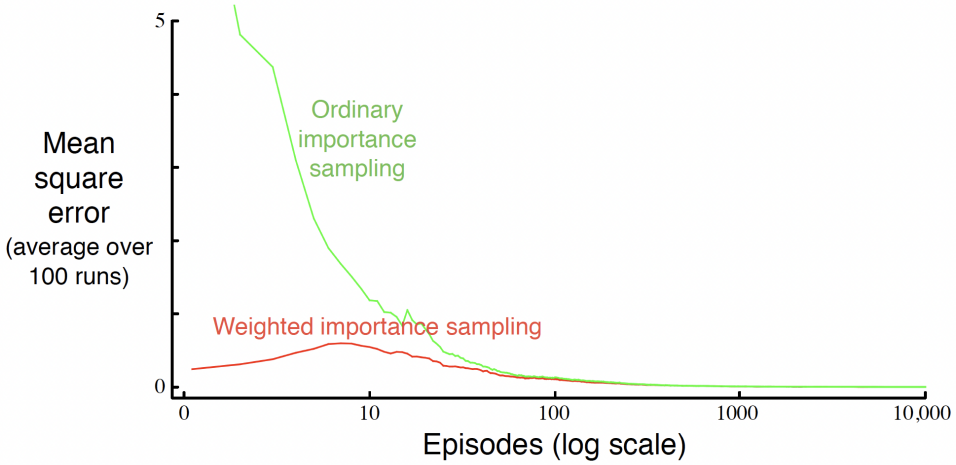
- ***Weighted importance sampling***

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

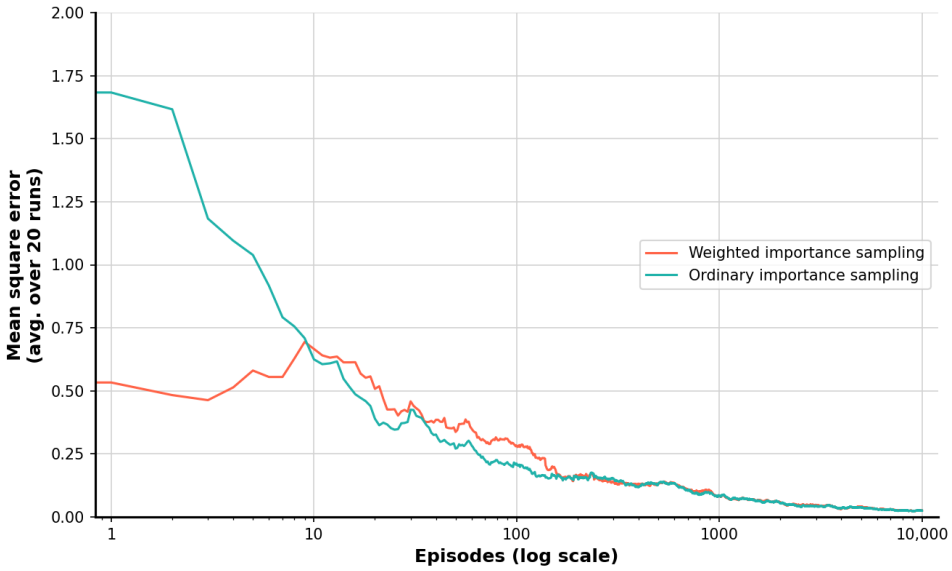
---

**Example 5.4 Off-policy Estimation of a Blackjack State Value**

We evaluated the state in which the dealer is showing a deuce, the sum of the player's cards is 13, and the player has a usable ace (that is, the player holds an ace and a deuce, or equivalently three aces). The data was generated by starting in this state then choosing to hit or stick at random with equal probability (the behavior policy). The target policy was to stick only on a sum of 20 or 21, as in Example 5.1. The value of this state under the target policy is approximately −0.27726 (this was determined by separately generating one-hundred million episodes using the target policy and averaging their returns)




*Implementation:*

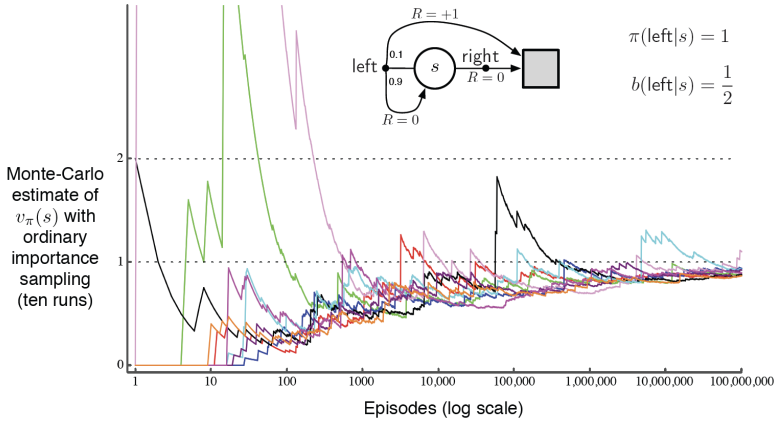



https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_05_monte_carlo_methods/example_5_4_off_policy_estimation.py

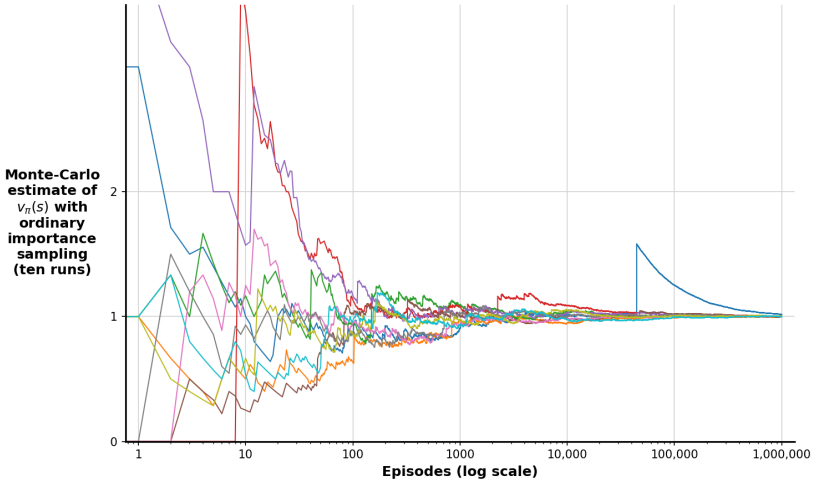
---

### Example 5.5 Infinite Variance

Ordinary importance sampling produces surprisingly unstable estimates on the one-state MDP shown inset (Example 5.5). The correct estimate here is 1 ($\gamma$ = 1), and, even though this is the expected value of a sample return (after importance sampling), the variance of the samples is infinite, and the estimates do not converge to this value. These results are for on-policy first-visit MC.




*Implementation*




https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_05_monte_carlo_methods/example_5_5_infinite_var.py

## ▼ 5.6 Incremental Implementation

- The weighted-average updated rule:

$$V_{n+1} \doteq \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k} = V_n + \frac{W_n}{C_n}(G_n - V_n)$$

  $C_n$ is the cumulative sum of weights given to the first $n$ returns.

- Weight sampling algorithm:

**Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$**

Input: an arbitrary target policy $\pi$
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$

Loop forever (for each episode):
    $b \leftarrow$ any policy with coverage of $\pi$
    Generate an episode following $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

**Exercise 5.10**

Derive the weighted-average update rule.

*Solution*

$$
\begin{aligned}
V_{n+1} &\doteq \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k} \\
&= \frac{W_n G_n + \sum_{k=1}^{n-1} W_k G_k}{W_n + \sum_{k=1}^{n-1} W_k} \frac{W_n + \sum_{k=1}^{n-1} W_k}{\sum_{k=1}^{n-1} W_k} \frac{\sum_{k=1}^{n-1} W_k}{W_n + \sum_{k=1}^{n-1} W_k} \\
&= \frac{W_n G_n + \sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k} \frac{\sum_{k=1}^{n-1} W_k}{W_n + \sum_{k=1}^{n-1} W_k} \\
&= \left( \frac{W_n G_n}{\sum_{k=1}^{n-1} W_k} + V_n \right) \frac{\sum_{k=1}^{n-1} W_k}{W_n + \sum_{k=1}^{n-1} W_k} \\
&= \frac{W_n}{W_n + \sum_{k=1}^{n-1} W_k} G_n + \frac{\sum_{k=1}^{n-1} W_k}{W_n + \sum_{k=1}^{n-1} W_k} V_n \\
&= \frac{W_n}{W_n + \sum_{k=1}^{n-1} W_k} G_n + \left( 1 - \frac{W_n}{W_n + \sum_{k=1}^{n-1} W_k} \right) V_n \\
&= V_n + \frac{W_n}{W_n + \sum_{k=1}^{n-1} W_k} (G_n - V_n)
\end{aligned}
$$

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

therefore:

$$
V_{n+1} \doteq \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k} = V_n + \frac{W_n}{C_n} (G_n - V_n)
$$

---

## ▼ 5.7 Off-policy Monte Carlo Control

- Off-policy Monte Carlo control methods requires that the behavior policy has a non-zero probability of selection all actions that might be selected by the target policy (coverage). To explore all possibilities, we require that the behavior policy be soft (i.e., that it select all actions in all states with non-zero probability).
- Off-policy MC control, for estimating $\pi \approx \pi_*$

**Off-policy MC control, for estimating $\pi \approx \pi_*$**

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$  (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$  (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

- A potential problem is that this method learns only from the tails of episodes, when all of the remaining actions in the episode are greedy. If nongreedy actions are common, then learning will be slow, particularly for states appearing in the early portions of long episodes.

---

**Exercise 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 out of 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.

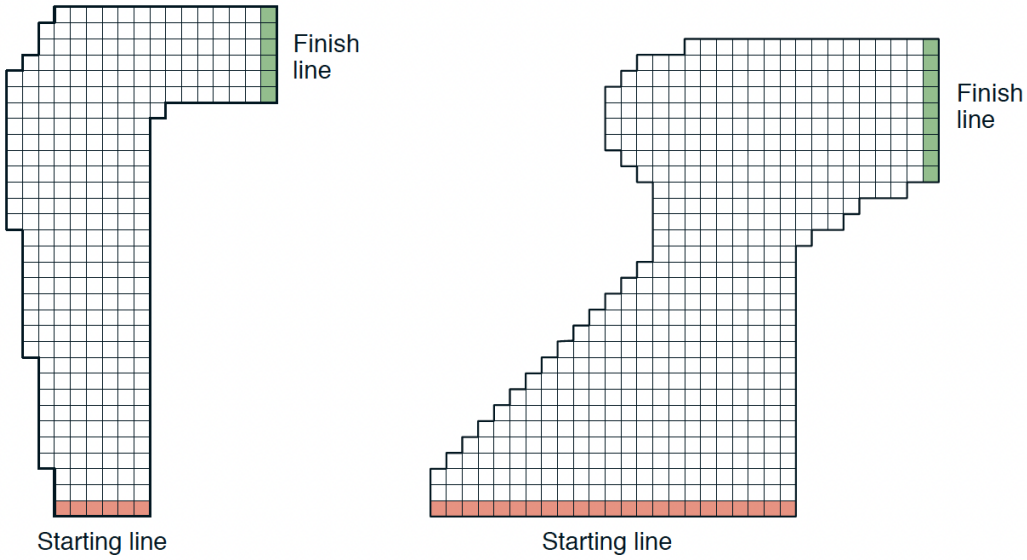
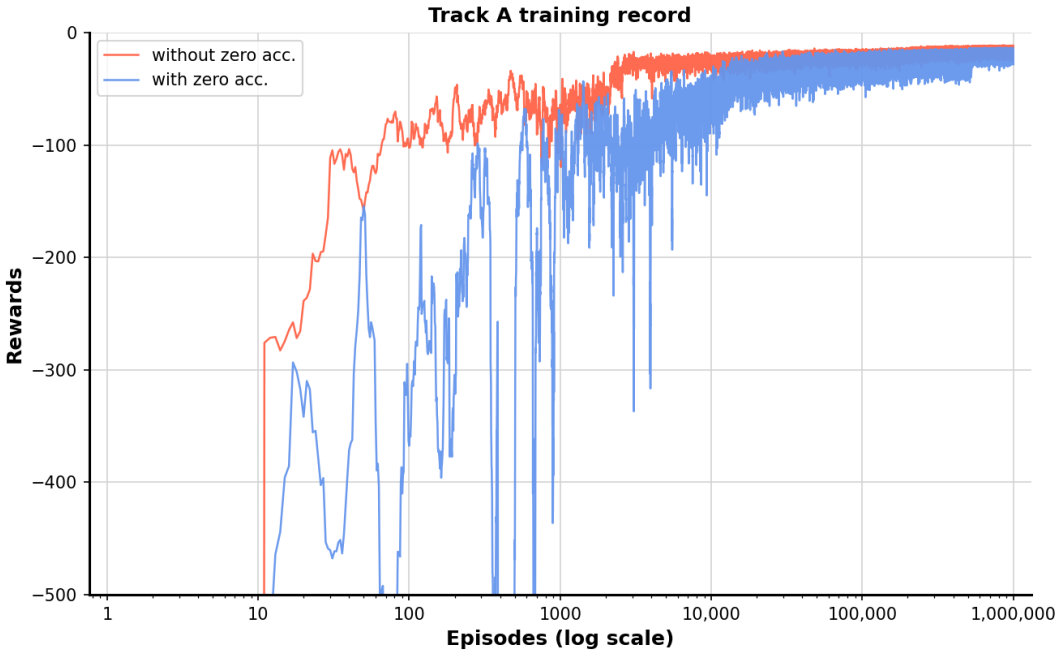
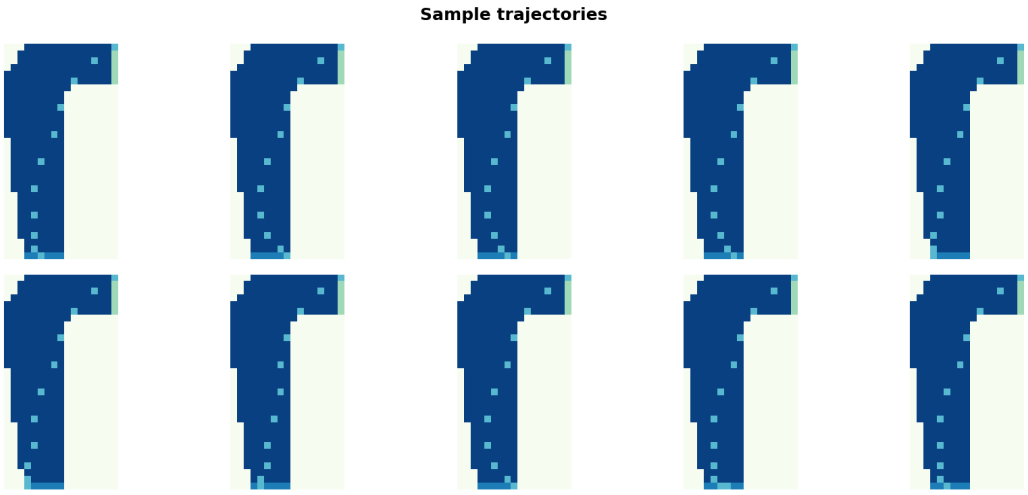



**Figure 5.5:** A couple of right turns for the racetrack task.

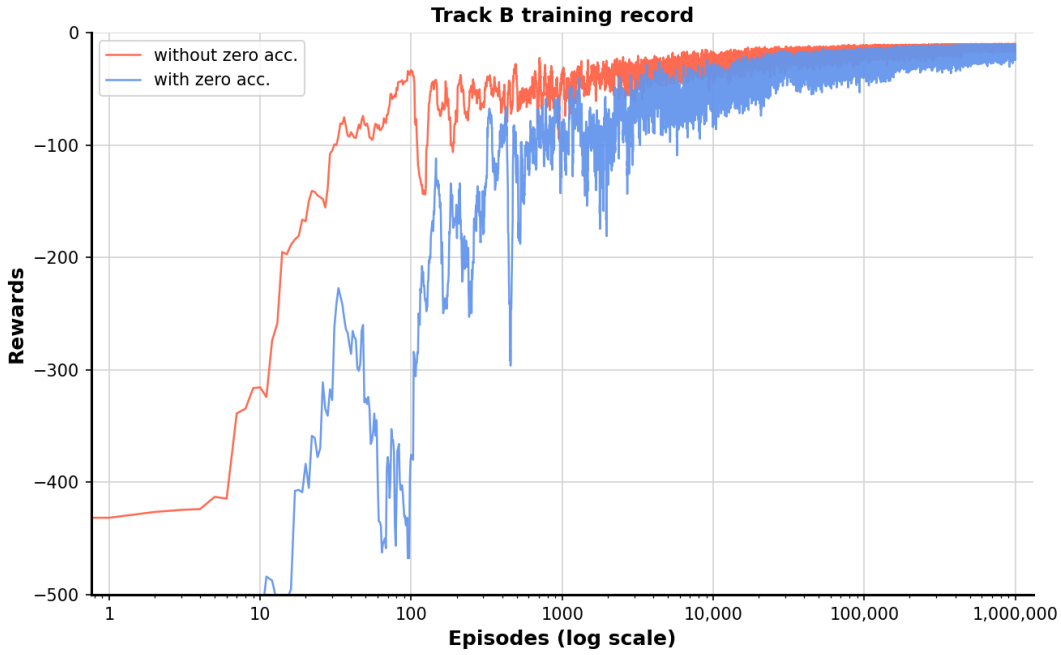
*Implementation:*

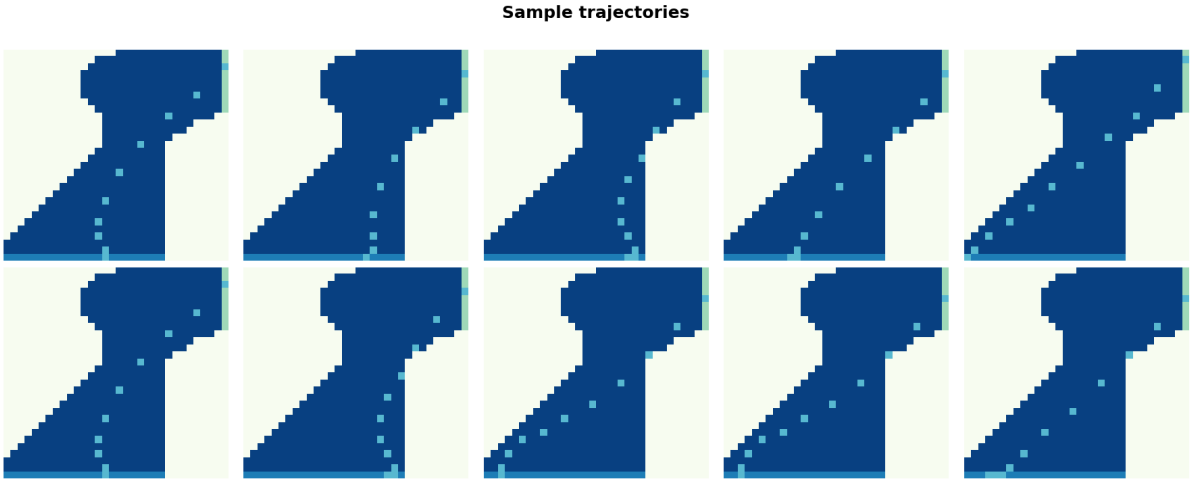
Track a:


Track A training record
without zero acc.
with zero acc.
Rewards
0
−100
−200
−300
−400
−500
1
10
100
1000
10,000
100,000
1,000,000
Episodes (log scale)


Sample trajectories

Track b:




https://github.com/terrence-ou/Reinforcement-Learning-2nd-Edition-Notes-Codes/blob/main/chapter_05_monte_carlo_methods/exercise_5_12_racetrack.py