Inf368 -p3

By Shania Muganga and Sara Ahmadi

Part 1: Problems:

Exercise 1 - Monte Carlo Methods

1. Consider a square with a side length of 1. Define analytically the ratio X between the area of the circle inscribed in the square and the square itself.

The ratio X could be analytically defined as such:

$$X = \frac{Area_{circle}}{Area_{square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

So, the ratio of the area of a circle and the area of a square is always pi over 4 regardless of side length.

2. Equate the analytical ratio X that you have computed to an empirical ratio ^X you will compute via Monte Carlo, and isolate π.

$$X = \hat{X}$$

$$\frac{\pi}{4} = \hat{X}$$

$$\pi = 4\hat{X}$$

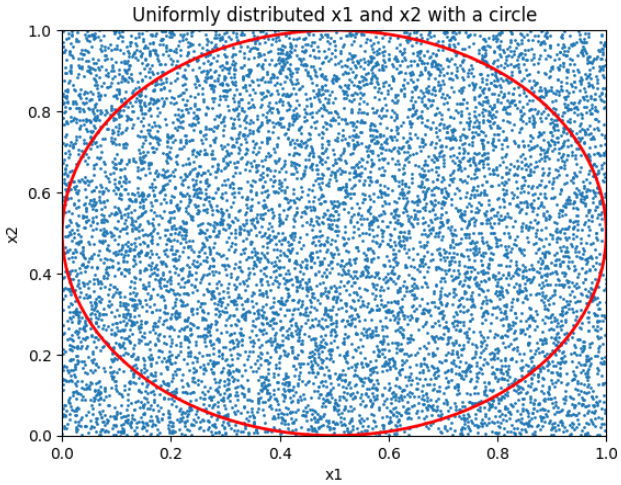
So given an empirically gathered ratio $\hat{X}$, we can estimate the value of π by multiplying it with 4.

3. Sample 10000 pairs of numbers (x1, x2) generated as follows:

- x1 ~ Unif[0, 1] are i.i.d. samples from a uniform distribution over [0, 1];
- x2 ~ Unif[0, 1] are i.i.d. samples from a uniform distribution over [0, 1].

To sample 10 000 pairs, we used python:

```
# Generate 10,000 samples of x1 and x2 pairs
x1 = np.random.uniform(0, 1, 10000)
x2 = np.random.uniform(0, 1, 10000)
```



4. Estimate X^ from the samples, by taking the ratio of points falling within and without the circle.

Using the points vectors x1 and x from the previous task, we calculate the distance of each point from the center of the circle as such, with d being a vector containing each of the distances from the center to the points:

$$d = \sqrt{(\mathrm{x1} - 0.5)^2 + (\mathrm{x2} - 0.5)^2}$$

Next, we compare the number of points inside the circle with the total number of points to find the ratio $\hat{X}$:

$$\hat{X} = \frac{number\ of\ points\ inside\ circle}{total\ number\ of\ point} = \frac{|d <= 0.5|}{|d|} = \frac{7882}{10000} = 0.7882$$

5. Estimate π.

Using $\pi = 4\hat{X}$ from task 2, we that π can be estimated to be:

$$\pi_1 = 4 * 0.7882$$

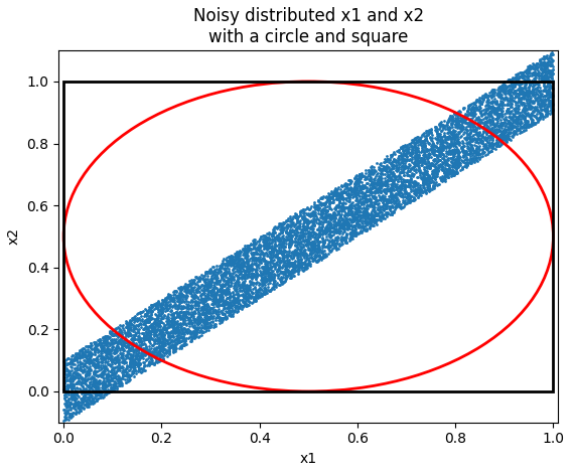
$$\pi_1 = 3.1528$$

6. Generate a new dataset of 10000 pairs of numbers (x1, x2) generated as follows:

- x1 ~ Unif[0, 1] are i.i.d. samples from a uniform distribution over [0, 1];
- x2 = x1 +Unif[−0.1, 0.1], that is, x2 is equal to x1 plus a i.i.d. random noise sampled from a uniform distribution over [−0.1, 0.1].

We again used python to generate the samples:

```
# Generate 10,000 samples of x1 and x2 pairs
x1 = np.random.uniform(0, 1, 10000)
x2 = x1+np.random.uniform(-0.1,0.1, 10000)
```



7. Estimate the new X^ and π

Same process as above.

$$\hat{X} = \frac{number\ of\ points\ inside\ circle}{total\ number\ of\ point} = \frac{|d\ <=\ 0.5|}{|d|} = \frac{7028}{10000} = 0.7028$$

$$\pi_2 = 4 * 0.7028$$

$$\pi_2 = 2.8112$$

8. (*) How do the two estimates change and why?

The noisy dataset had a lower $\hat{X}$ and π estimate, this is due to how we defined the π estimate. The π was based on the ratio: the area of a circle and the area of the square it is inscribed in. When plotting the noisy data set, it can be clearly seen that the ratio of the points is not representative of the area of the circle or the area of the square. The ratio and the π estimate decreased, this suggests there's less points in the circle and/or more points outside of the circle, which if we look at the figure, it can be clearly seen more points are outside of the circle as well as outside of the square. Which all corresponds to how the x2 vector was created.

However, the initial dataset, from the figure seems to have uniformly distributed the points over the area, resulting in an estimate that closely fits the $\pi = 3.14$. with our estimate being $\pi_1 = 3.1528$.

Exercise 2 - MC and MABs

1. What sort of problem are we solving in a MAB: prediction or control?

In a Multi-Armed Bandit (MAB) problem, we are primarily solving a control problem. The goal in a control problem is to find a policy that maximizes the expected return (rewards) over time. In the context of MABs, this involves deciding which arm to pull (action) to maximize the cumulative reward. Unlike prediction problems, which focus on estimating the value of states or actions under a given policy, control problems, including MABs, seek to identify the optimal policy itself.

2. In a MAB you compute expected rewards of arms. Would this correspond to a state-value function or an action-value function?

In the context of MABs, computing the expected rewards of arms corresponds to an action-value function. The action-value function, often denoted as $Q(s,a)$, estimates the expected return of taking an action a in a state s and following a certain policy thereafter. Since MABs can be conceptualized as having a single state with multiple actions (each

arm being an action), the expected reward of an arm is essentially the value of taking that action from the single state, fitting the definition of an action-value function.

3. What would correspond to a trajectory in a MAB? What would be a collection of trajectories?

In a MAB, a trajectory corresponds to a sequence of actions (arm pulls) and the observed rewards from those actions. Since MABs are typically framed as having a single state, the notion of transitioning between states does not apply as it would in more complex environments. Therefore, a trajectory in a MAB context is simply the history of actions taken and the rewards received. A collection of trajectories would be a set of such sequences, representing multiple instances of interacting with the bandit, where each instance starts at the beginning of the interaction and concludes after a predetermined number of actions or an ending condition is met.

4. To what MC family of algorithms would you relate the MAB algorithms we studied: MC prediction or MC control?

The ϵ-greedy algorithm for MABs can be seen as an instance of an MC control algorithm. MC control algorithms are concerned with finding an optimal policy that maximizes the expected return. The ϵ-greedy strategy in MABs, where with probability ϵ a random arm is selected and with probability $1-\epsilon$ the arm with the highest estimated value is chosen, is a method for balancing exploration and exploitation to ultimately converge on an optimal policy of arm selection.

5. What would correspond to MC prediction in a MAB?

In a MAB, MC prediction would correspond to the process of estimating the expected reward (action-value) of each arm based on sampled returns, without necessarily seeking to improve the policy. This involves using the outcomes of arm pulls (rewards) to update our estimates of each arm's value, with the goal of accurately predicting the expected return of each action over time.

6. What is the purpose of ϵ in MABs and in MC control?

The purpose of ϵ in MABs and in MC control algorithms is to introduce exploration. In ϵ-greedy strategies, ϵ represents the probability of selecting an action at random, as opposed to choosing the best-known action. This allows the algorithm to explore less-frequented actions that might yield higher rewards than the current best estimate, ensuring that the algorithm does not prematurely converge to a suboptimal policy due to insufficient exploration.

7. Could we reduce a standard RL problem solved by MC to a MAB problem? If so, what would we lose?

A standard RL problem involves decisions across multiple states and is typically more complex than a MAB problem, which can be thought of as an RL problem with only one state. While it is possible to simplify a standard RL problem to resemble a MAB problem by focusing on a single decision point or action selection mechanism, this reduction would result in the loss of state-dependent decision-making. In other words, the dynamics of transitioning between different states and the strategy of selecting actions based on those states' values are lost, simplifying the problem to decision-making based on immediate rewards alone, without considering the long-term consequences of actions in a stateful environment.

Exercise 3 - Importance Sampling

1. As ground truth to use for reference, compute the exact expected values E[R] and E[S]

$$E[X] = \sum reward_i \times prob_i$$

$$E[R] = 0 \times 0.35 + 5 \times 0.3 + 10 \times 0.25 + 20 \times 0.1$$

$$E[R] = 6$$

$$E[S] = 0 \times 0.3 + 5 \times 0.35 + 10 \times 0.3 + 20 \times 0.05$$

$$E[S] = 5.75$$

2. Implement the two slot machines as functions from which you can get samples.

The implementation in python:

```python
def slot_machine_R(n):
    '''
    Simulate n plays of slot machine R
    '''
    rewards = [0, 5, 10, 20]
    probabilities = [0.35, 0.3, 0.25, 0.1]
    return np.random.choice(rewards, p=probabilities,size=n)

def slot_machine_S(n):
    '''
    Simulate n plays of slot machine S
    '''
    rewards = [0, 5, 10, 20]
    probabilities = [0.3, 0.35, 0.3, 0.05]
    return np.random.choice(rewards, p=probabilities,size=n)
```

3. Play with slot machine S. Sample 10000 rewards and compute the empirical expectation ˆE[S].

Using the function implemented above I got 10000 samples from machine S, I then took the mean of these rewards to find the empirical expected value $\hat{E}[S]$:

$$\hat{E}[S] = 5.721$$

4. (*) Compute the ratio ρ between the distribution probability of R and S.

$\rho = P_R(\text{rewards}) / P_S(\text{Rewards})$

For 0$: 0.35/0.3 = 1.1667

For 5 $: 0.3/0.35 = 0.8571

For 10 $: 0.25/0.3 = 0.8333

For 20 $: 0.1 / 0.05 = 2

5. (*) Suppose now that you only have the samples from S that you have already collected. Suppose, also, that someone let you know the value of ρ. Use importance sampling to compute the empirical expectation of ˆE[R] from the samples of S and ρ.

For each of the ratios above and the estimated expected value of S we can estimate $\hat{E}[R]$ using importance sampling:

$$\hat{E}[R] = \frac{\sum_{s \in Samples_S} ratio_s \times reward_s}{\sum_{s \in Samples_S} ratio_s}$$

This can also be calculated using the total counts of each reward that was observed as such:

$$\hat{E}[R] = \frac{\sum_{r \in Rewards} ratio_r \times reward_r \times counts_r}{\sum_{r \in Rewards} ratio_r \times count_r}$$

So, using the counts attained from the sample above:

$$\hat{E}[R] = \frac{1.1667 \times 0 \times 2973 + 0.8571 \times 5 \times 3528 + 0.8333 \times 10 \times 3041 + 2 \times 20 \times 485}{1.1667 \times 2973 + 0.8571 \times 3528 + 0.8333 \times 3041 + 2 \times 485}$$

$$\hat{E}[R] = \frac{58781.6667}{9942.6667}$$

$$\hat{E}[R] = 5.9121$$

6. Suppose now that you play with a machine S' with a reward distribution of [.25, .25, .25, .25]. Sample again 10000 rewards, evaluate ˆE[S'], compute the ratio ρ between the distribution probability of R and S', and estimate ˆE[R] via importance sampling from the samples of S' and ρ.

So new values:

rewards: [0, 5, 10, 20]

Counts: [2533, 2439, 2564, 2464]

$$\hat{E}[S'] = 8.7115$$

Ratios: [1.4, 1.2, 1.0, 0.4]

Using the same formula as above we now get:

$$\hat{E}[R] = \frac{59986.0}{10022.6} = 5.9850$$

7. Suppose now that you play with a machine S'' with a reward distribution of [.001, .499, .499, .001]. Sample again 10000 rewards, evaluate ˆE[S''], compute theratio ρ between the distribution probability of R and S'', and estimate ˆE[R] via importance sampling from the samples of S'' and ρ.

So new values:

rewards: [0, 5, 10, 20]

Counts: [13, 5027, 4949, 11]

$$\hat{E}[S'] = 7.4845$$

Ratios: [349.9994, 0.6012, 0.5010, 100.0]

Using the same formula as above we now get:

$$\hat{E}[R] = \frac{61905.8116}{11151.7034} = 5.5512$$

8. (*) How do the results you computed differ? What explains the differences?

The results differ for each of the samples used but only slightly. As we used Jupyter notebook with each run the numbers varied quite a bit with each run if we didn't run it all. In general, we expect $\hat{E}$[R] calculated from S to be the most similar to its true value E[R] due to the probability distributions being more closely aligned. However, with the seed we had both S and S' were close to each other. Both yielding slightly higher estimates whilst S'' only yielding a slightly lower value than the true expected reward.

The reason for the differences is the importance sampling for different samples S, S', and S'' can be attributed to the characteristics of each sample distribution and their respective ratios with the distribution of R.

Sample Distribution S:

- Sample distribution S has probabilities [0.3, 0.35, 0.3, 0.05], which are relatively close to the distribution of R.
- The ratio between the distributions R and S varies but is not extremely skewed.
- The estimated expected value of R obtained using importance sampling is 5.9121, which is close to the true expected value of R (5.75).
- The small differences can be attributed to the variation in the samples and the approximation introduced by importance sampling.

Sample Distribution S':

- Sample distribution S' has equal probabilities [0.25, 0.25, 0.25, 0.25] for each reward, which differs significantly from the distribution of R.
- The ratio between the distributions R and S' varies but remains relatively high for rewards 0 and 20 and lower for rewards 5 and 10.
- The estimated expected value of R obtained using importance sampling is 5.9850, which is slightly higher than the true expected value of R.
- The difference can be explained by the higher probability assigned to rewards 0 and 20 in S' compared to the true distribution of R.

Sample Distribution S'':

- Sample distribution S'' has probabilities [0.001, 0.499, 0.499, 0.001], which are skewed towards rewards 5 and 10, with very low probabilities for rewards 0 and 20.
- The ratio between the distributions R and S'' varies significantly, with a very high value for reward 0 and a relatively low value for reward 20.
- The estimated expected value of R obtained using importance sampling is 5.5512, which is lower than the true expected value of R.
- The significant skewness in the sample distribution S'' towards rewards 5 and 10, combined with the low probabilities for rewards 0 and 20, leads to a biased estimation of the expected value of R.

In summary, the differences in the estimated expected values of R arise from the discrepancies between the sample distributions S, S', and S'' and the true distribution of R, as well as the varying ratios between these distributions. The effectiveness of importance sampling depends on how closely the sample distribution matches the target distribution, with larger discrepancies leading to less accurate estimations.

Exercise 4 - Q-learning

In class we introduced Q-learning as an off-policy algorithm.

1. (*) Given that we do off-policy control, why do we not need to rely on importance ratio in Q-learning?

In Q-learning, we update the Q-values directly using the Bellman equation without explicitly considering importance sampling ratios. This is because Q-learning is an off-policy algorithm that utilizes a technique called "bootstrapping," which means it updates its Q-values based on the maximum Q-value of the next state, regardless of the policy being followed.

In off-policy learning, such as Q-learning, the agent learns from data generated by following a different policy (often an exploration policy like epsilon-greedy) than the target policy (the policy it's trying to learn). This means that the agent can update its Q-values using experiences collected under a different policy without the need for importance sampling ratios.

Importance sampling is typically used in situations where we need to adjust for differences in the probability distributions between the behaviour policy (the policy generating the data) and the target policy (the policy being learned). However, in Q-learning, the updates are based on the maximum Q-value of the next state, which inherently handles this discrepancy without requiring explicit importance sampling ratios.

Therefore, in Q-learning, we do not need to rely on importance ratios because the updates are based on the maximum Q-value of the next state, allowing the algorithm to learn effectively from off-policy data.