Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)
Department of Computer Science and Engineering (Data Science)

Subject: Reinforcement Learning

AY: 2023 – 24

Bhuvi Ghosh
60009210191

Experiment 1
Exploration Exploitation Dilemma

AIM:

a) To solve the exploration exploitation dilemma using epsilon greedy strategy
b) To understand the effect of epsilon by comparing the different values of epsilon

THEORY:

With partial knowledge about future states and future rewards, our reinforcement learning agent will be in a dilemma on whether to exploit the partial knowledge to receive some rewards or it should explore unknown actions which could result in much larger rewards.

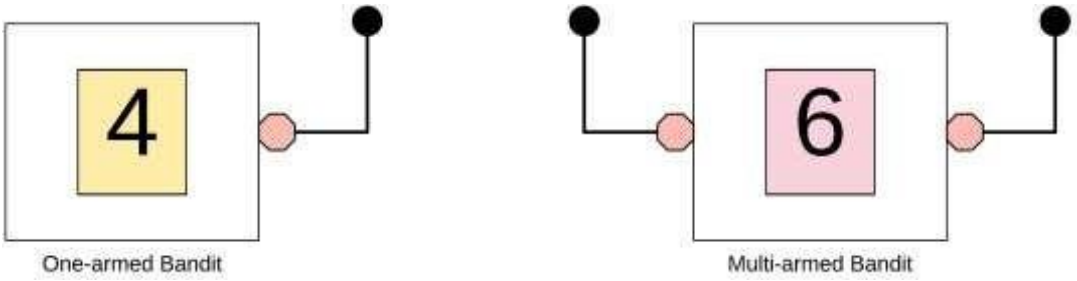
- Exploitation: Make the best decision given current information. The best long-term strategymay involve short-term sacrifices
- Exploration Gather more information. Gather enough information to make the best overalldecisions

However, we cannot choose to explore and exploit simultaneously. In order to overcome the Exploration-Exploitation Dilemma, we use the Epsilon Greedy Policy.

MULTI ARMED BANDIT PROBLEM (MAB)

In a multi-armed bandit problem (MAB) (or n-armed bandits), an Agent makes a choice from a set of actions. This choice results in a numeric reward from the Environment based on the selected action. In this specific case, the nature of the Environment is a stationary probability distribution. By stationary, we mean that the probability distribution is constant (or independent) across all states of the Environment. In other words, the probability distribution is unchanged as the state of the Environment changes. The goal of the Agent in a MAB problem is to maximize the rewards received from the Environment over a specified period.

The MAB problem is an extension of the "one-armed bandit" problem, which is represented as a slot machine in a casino. In the MAB setting, instead of a slot machine with one-lever, we have multi-levers. Each lever corresponds to an action the Agent can play. The goal of the Agent is to make plays that maximize its winnings (i.e., rewards) from the machine. The Agent will have to figure out the best levers (exploration) and then concentrate on the levers (exploitation) that will maximize its returns (i.e., the sum of the rewards).



One-armed Bandit Multi-armed Bandit

Left: One-armed bandit. The slot machine has one lever that returns a numerical reward when played.

Right: Multi-armed bandits. The slot machine has multiple (n) arms, each returning a numerical reward when played. In a MAB problem, the reinforcement agent must balance exploration and exploitation tomaximize returns.

Epsilon-greedy

The agent does random exploration occasionally with probability ἐ and takes the optimal action most of the time with probability 1− ἰ.

Epsilon greedy method. At each step, a random number is generated by the model. If the number was lower than epsilon in that step (exploration area) the model chooses a random action and if it was higher than epsilon in that step (exploitation area) the model chooses an action based on what it learned.

Usually, epsilon is set to be around 10%. Epsilon-Greedy can be represented as follows:

$$A_t \leftarrow \begin{cases} argmax\ Q_t(a) & with\ probability\ 1-\epsilon \\ a \sim Uniform(\{a_1 ... a_k\}) & with\ probability\ \epsilon \end{cases}$$

The Action that the agent selects at time step t, will be a greedy action (exploit) with probability(1-epsilon) or may be a random action (explore) with probability of epsilon.

ALGORITHM:

Algorithm 2: Epsilon-Greedy Action Selection

Data: Q: Q-table generated so far, : a small number, S: current state

Result: Selected action

Function *SELECT-ACTION(Q, S, ϵ)* **is**

    n ← uniform random number between 0 and 1;
    if $n < \epsilon$ **then**
        A ← random action from the action space;
    else
        A ← maxQ(S,.);
    end
    return selected action A;

end

LAB ASSIGNMENT TO DO:

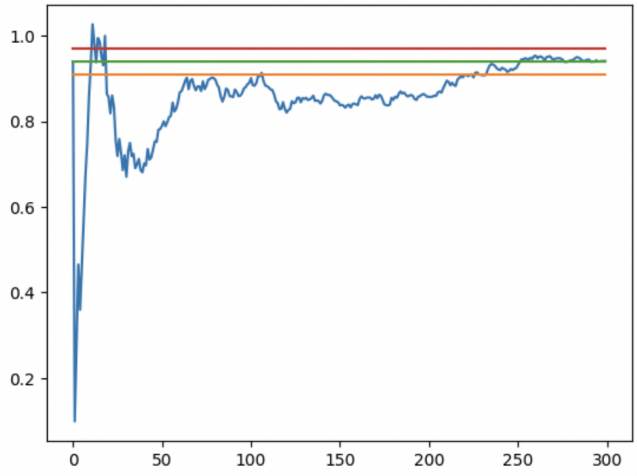
1. Create a multi armed bandit agent which would estimate the win rate using the epsilon-greedy strategy.
2. Understand the effect of the value of epsilon on the win rate by comparing the win rates corresponding to different values of epsilon. Hence draw conclusions.

```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

```
class Action:
  def __init__(self, m):
    self.m = m
    self.mean = 0
    self.N = 0
  def select(self):
    return np.random.normal(0,1) + self.m
  def update(self, x):
    self.N += 1
    self.mean = self.mean + (1/self.N)*(x-self.mean)
```

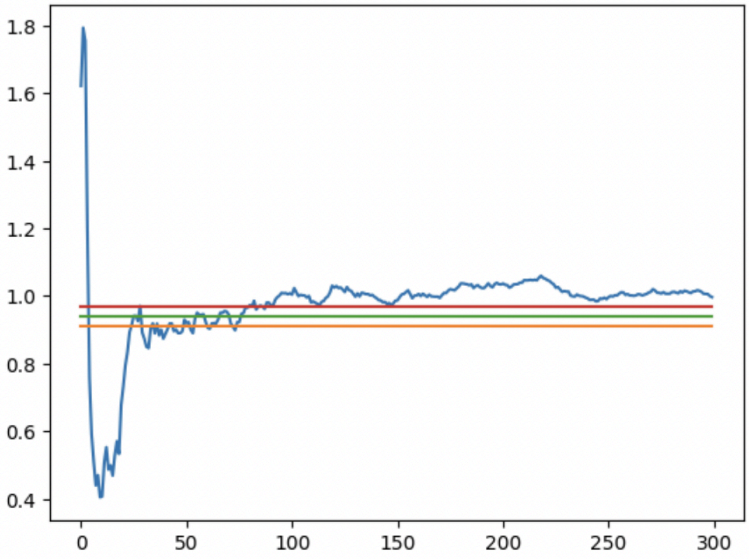
```
def three_arm_bandit_ucb(m1, m2, m3, c, N):                #parameter c for the confidence level
  actions = [Action(m1), Action(m2), Action(m3)]
  data = np.empty(N)
  for i in range(N):
    j = np.argmax([a.mean + c*np.sqrt(np.log(i)/a.N) for a in actions])
                                                           #selects the action with the highest upper confidence bound
                                                           #a.mean is the mean reward of the action,
                                                           #c is a parameter controlling the exploration-exploitation trade-off, and
                                                           #a.N is the number of times the action has been selected.
                                                           #reward is sampled from the action's reward distribution

    x = actions[j].select()
    actions[j].update(x)
    data[i] = x
  cumulative_average = np.cumsum(data) / (np.arange(N) + 1)
  plt.plot(cumulative_average)
  plt.plot(np.ones(N)*m1)
  plt.plot(np.ones(N)*m2)
  plt.plot(np.ones(N)*m3)
  plt.show()
  for i in range(len(actions)):
    print(f"Number of times arm {i+1} selected: {actions[i].N}")
    print(f"Mean of rewards from arm {i+1}: {actions[i].mean}")
```

```
three_arm_bandit_ucb(0.91, 0.94, 0.97, 1, 300)
```



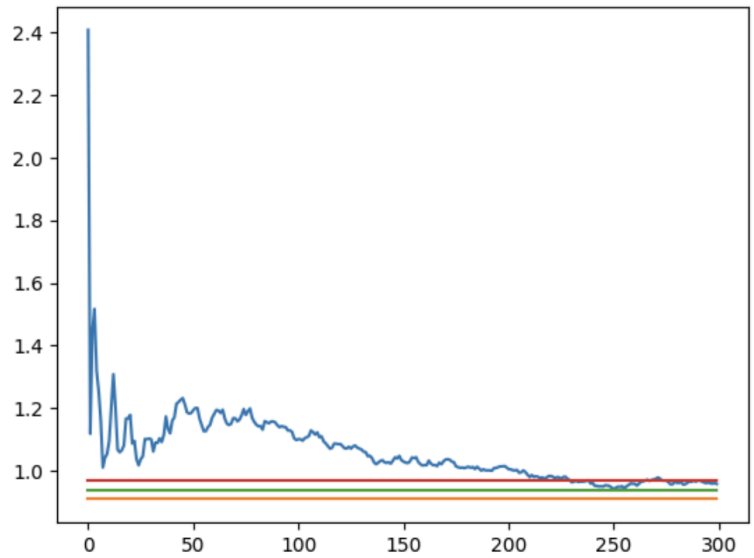

```
Number of times arm 1 selected: 161
Mean of rewards from arm 1: 1.0098246198497558
Number of times arm 2 selected: 106
Mean of rewards from arm 2: 0.9139035969260397
Number of times arm 3 selected: 33
Mean of rewards from arm 3: 0.69145915296707
```

```
[ ] three_arm_bandit_ucb(0.91, 0.94, 0.97, 2, 300)
```



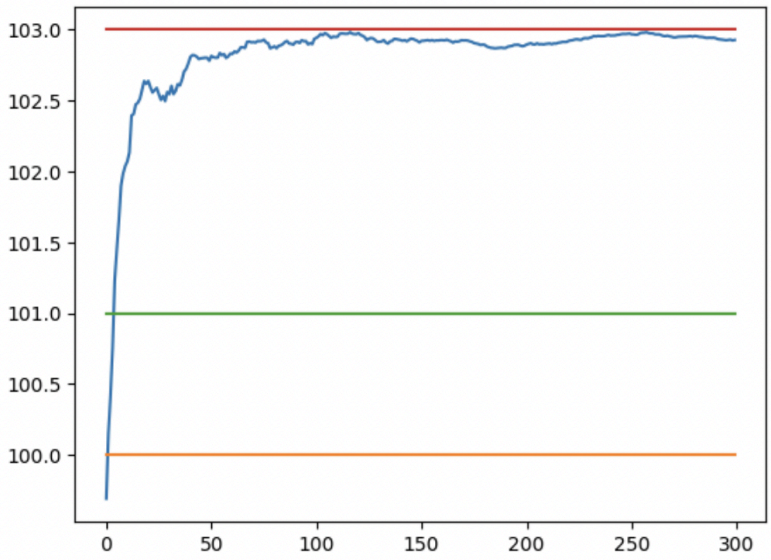

```
Number of times arm 1 selected: 100
Mean of rewards from arm 1: 0.9975271719041642
Number of times arm 2 selected: 115
Mean of rewards from arm 2: 1.026687160605454
Number of times arm 3 selected: 85
Mean of rewards from arm 3: 0.9560442388768976
```

```
[ ] three_arm_bandit_ucb(0.91, 0.94, 0.97, 3, 300)
```



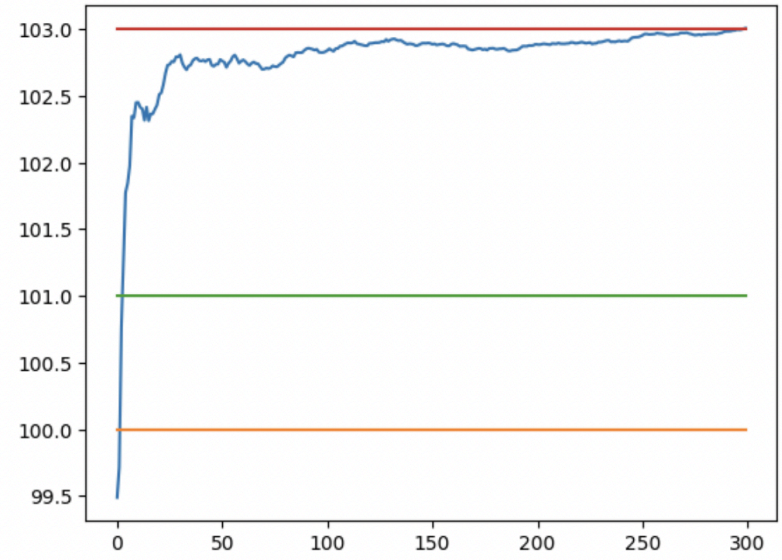

```
Number of times arm 1 selected: 99
Mean of rewards from arm 1: 0.9509291832717517
Number of times arm 2 selected: 72
Mean of rewards from arm 2: 0.828514347001897
Number of times arm 3 selected: 129
Mean of rewards from arm 3: 1.0382217996755896
```

```
[ ]  three_arm_bandit_ucb(100, 101, 103, 1, 300)
```



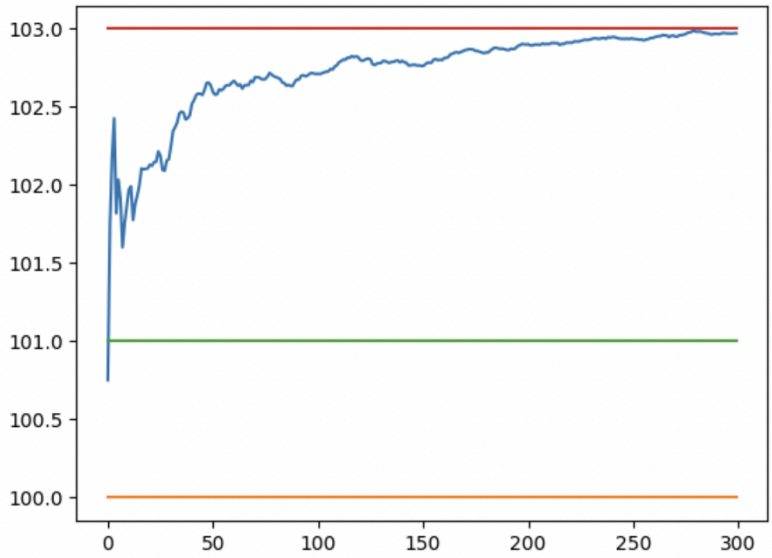

```
Number of times arm 1 selected: 1
Mean of rewards from arm 1: 99.69084792596031
Number of times arm 2 selected: 1
Mean of rewards from arm 2: 100.6083321829762
Number of times arm 3 selected: 298
Mean of rewards from arm 3: 102.94710991591367
```

```
[ ]  three_arm_bandit_ucb(100, 101, 103, 2, 300)
```



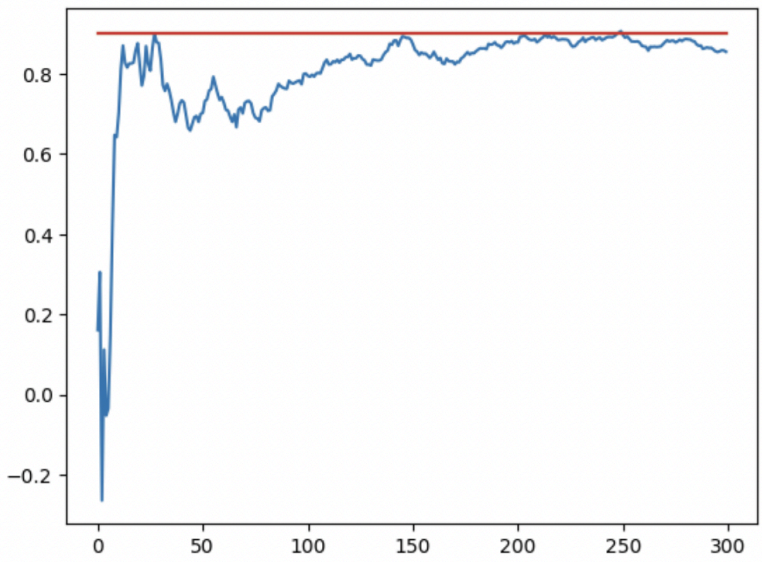

```
Number of times arm 1 selected: 3
Mean of rewards from arm 1: 99.92358872526458
Number of times arm 2 selected: 6
Mean of rewards from arm 2: 101.2291845831146
Number of times arm 3 selected: 291
Mean of rewards from arm 3: 103.07587175793402
```

```
three_arm_bandit_ucb(100, 101, 103, 3, 300)
```



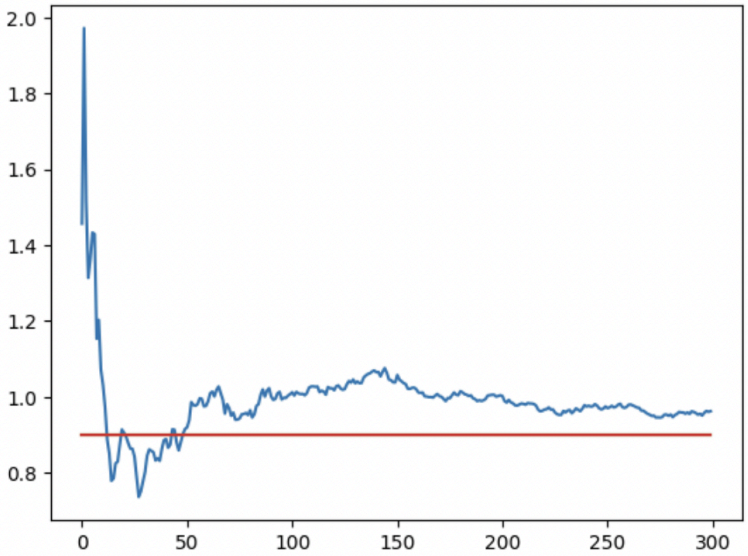

```
Number of times arm 1 selected: 5
Mean of rewards from arm 1: 100.01901427056241
Number of times arm 2 selected: 7
Mean of rewards from arm 2: 100.78605663937978
Number of times arm 3 selected: 288
Mean of rewards from arm 3: 103.07645186361088
```

```
three_arm_bandit_ucb(0.9, 0.9, 0.9, 1, 300)
```



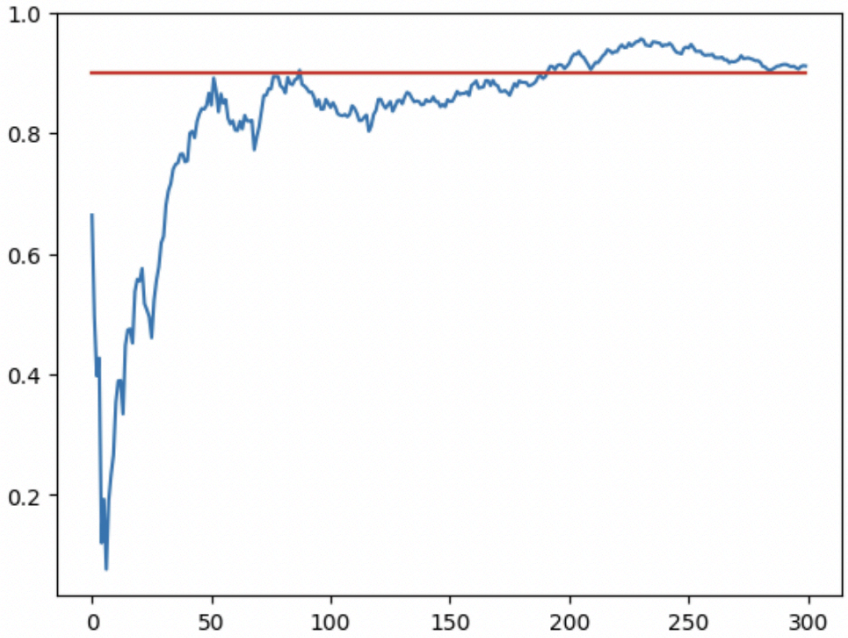

```
Number of times arm 1 selected: 14
Mean of rewards from arm 1: 0.3508889979822385
Number of times arm 2 selected: 285
Mean of rewards from arm 2: 0.8881705024281648
Number of times arm 3 selected: 1
Mean of rewards from arm 3: -1.405068404832262
```

```
three_arm_bandit_ucb(0.9, 0.9, 0.9, 2, 300)
```




```
Number of times arm 1 selected: 71
Mean of rewards from arm 1: 0.8983541292373464
Number of times arm 2 selected: 141
Mean of rewards from arm 2: 1.0228961041027391
Number of times arm 3 selected: 88
Mean of rewards from arm 3: 0.9180044267019893
```

```
three_arm_bandit_ucb(0.9, 0.9, 0.9, 3, 300)
```




```
Number of times arm 1 selected: 95
Mean of rewards from arm 1: 0.8808138762396474
Number of times arm 2 selected: 133
Mean of rewards from arm 2: 0.9983080045177701
Number of times arm 3 selected: 72
Mean of rewards from arm 3: 0.790557698245236
```

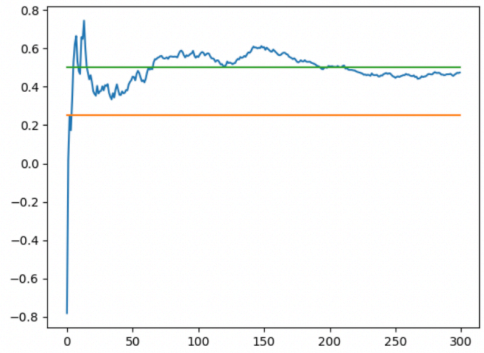
Two Arm Bandit

```
'''This function takes four parameters:
m1: The mean reward of the first action (arm).
m2: The mean reward of the second action (arm).
eps: The probability of choosing a random action (exploration rate).
N: The total number of iterations (time steps) to run the bandit problem.'''

def two_arm_bandit_epsilon_greedy(m1, m2, eps, N):
  actions = [Action(m1), Action(m2)]  # initializes two actions with means m1 and m2.
  data = np.empty(N)  # initializes an empty array data to store the rewards obtained in each iteration.
  explore, exploit = 0, 0  # counters explore and exploit to keep track of how many times the algorithm explores
# (chooses a random action) and exploits (chooses the action with the highest estimated reward), respectively.
  for i in range(N):
    p = np.random.random()                                  #generates a random number p between 0 and 1.
    if p < eps:                                             #If p is less than eps, the algorithm chooses to explore by randomly selecting one of the actions.
      j = np.random.choice(2)                               #If p is greater than or equal to eps, the algorithm chooses to exploit by selecting the action with'''
                                                            #the highest estimated reward.'''
      explore+=1
    else:
      j = np.argmax([a.mean for a in actions])              #It selects the chosen action (j), selects a reward (x), and updates the action's estimate based'''
                                                            #on the observed reward using a fixed step size of 0.1.'''
      exploit+=1
    x = actions[j].select()
    actions[j].update(x,0.1)
    data[i] = x                                              #It stores the observed reward x in the data array.
  cumulative_average = np.cumsum(data) / (np.arange(N) + 1)     #calculates the cumulative average reward obtained at each time step and plots it.
  plt.plot(cumulative_average)
  plt.plot(np.ones(N)*m1)
  plt.plot(np.ones(N)*m2)
  plt.show()
  print("No of times explored: ",explore)
  print("No of times exploited: ",exploit)
  for i in range(len(actions)):
    print(f"Mean of rewards from arm {i+1}: {actions[i].mean}")     #prints the final estimated mean rewards for each action.
    return cumulative_average                                       #returns the cumulative average rewards over time.
```

Two Armed Bandit Problem

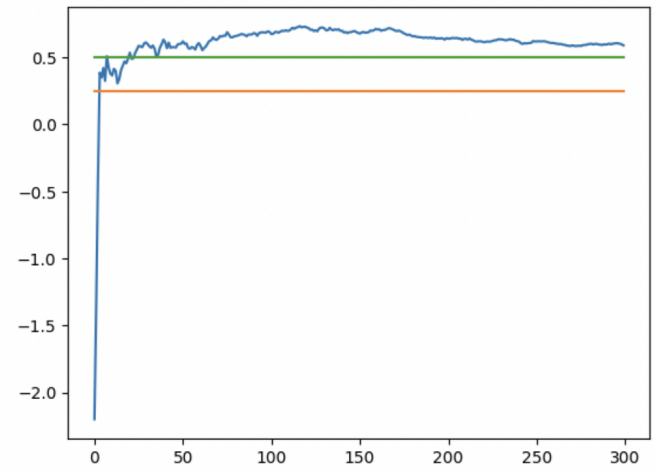
- The algorithm balances exploration and exploitation using the epsilon-greedy strategy. With probability eps, it explores by choosing a random action, and with probability 1-eps, it exploits by choosing the action with the highest estimated reward.
- It maintains estimates of the mean rewards for each action and updates them incrementally based on observed rewards. The algorithm's performance is evaluated by plotting the cumulative average reward over time and printing relevant statistics such as exploration and exploitation counts and final estimated mean rewards.

```
c1 = two_arm_bandit_epsilon_greedy(0.25, 0.5, 0.1, 300)
```



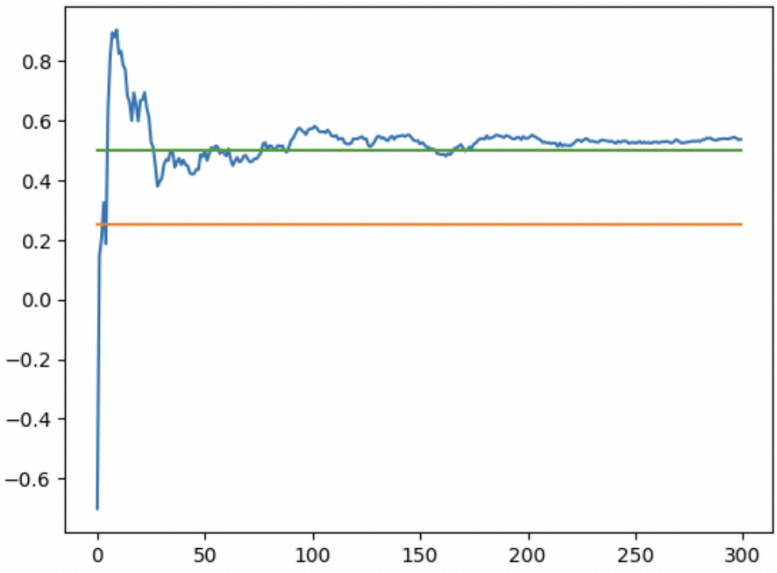

```
No of times explored:  30
No of times exploited:  270
Mean of rewards from arm 1: 0.719710397412403
```

```
c2 = two_arm_bandit_epsilon_greedy(0.25, 0.5, 0.05, 300)
```




```
No of times explored:  22
No of times exploited:  278
Mean of rewards from arm 1: 0.19725477255606397
```

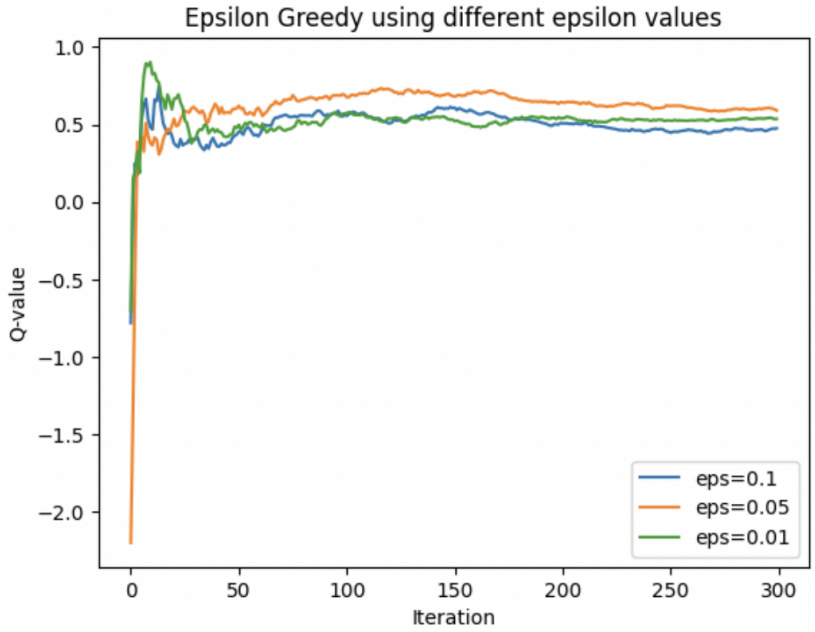
```
[ ]  c3 = two_arm_bandit_epsilon_greedy(0.25, 0.5, 0.01, 300)
```




```
No of times explored:  4
No of times exploited:  296
Mean of rewards from arm 1: 0.06670290395877577
```

```
plt.plot(c1,label='eps=0.1')
plt.plot(c2,label='eps=0.05')
plt.plot(c3,label='eps=0.01')
plt.xlabel('Iteration')
plt.ylabel('Q-value')
plt.title('Epsilon Greedy using different epsilon values')
plt.legend(loc='lower right')
```

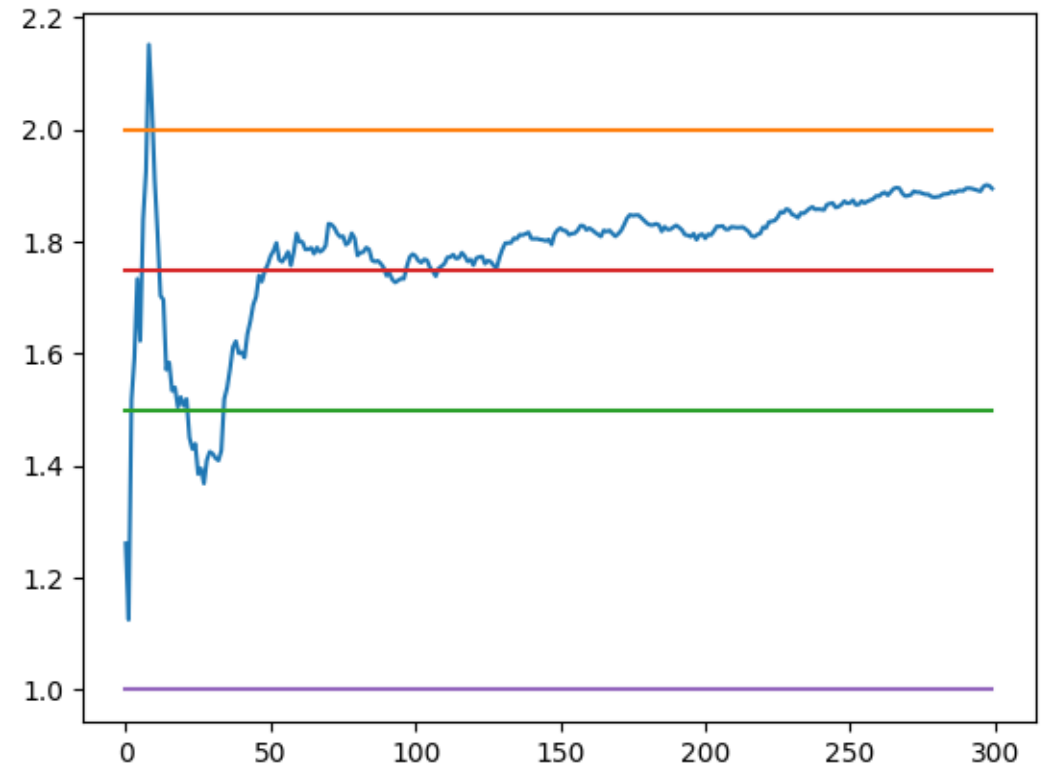
```
<matplotlib.legend.Legend at 0x7837603dfdc0>
```



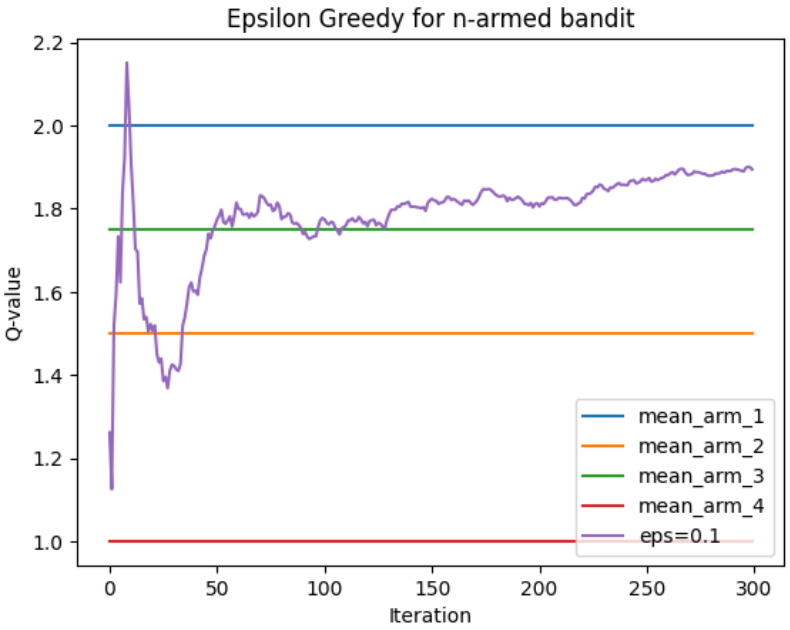
```python
import numpy as np
import matplotlib.pyplot as plt

class Action:
    def __init__(self, m):
        self.m = m
        self.mean = 0
        self.N = 0

    def select(self):
        return np.random.normal(0, 1) + self.m

    def update(self, x, alpha=0.1):
        self.N += 1
        self.mean = self.mean + alpha * (x - self.mean)

def epsilon_greedy(actions, eps):
    p = np.random.random()
    if p < eps:
        return np.random.choice(len(actions))
    else:
        return np.argmax([a.mean for a in actions])

def multi_arm_bandit_epsilon_greedy(means, eps, N):
    actions = [Action(m) for m in means]
    data = np.empty(N)
    explore, exploit = 0, 0

    for i in range(N):
        j = epsilon_greedy(actions, eps)
        x = actions[j].select()
        actions[j].update(x, 0.1)
        data[i] = x

        if j < len(actions) - 1:
            explore += 1
        else:
            exploit += 1

    cumulative_average = np.cumsum(data) / (np.arange(N) + 1)
    plt.plot(cumulative_average)
    for m in means:
        plt.plot(np.ones(N) * m)
    plt.show()

    print("No of times explored: ", explore)
    print("No of times exploited: ", exploit)
    for i in range(len(actions)):
        print(f"Mean of rewards from arm {i + 1}: {actions[i].mean}")

    return cumulative_average

# Input from the user
n = int(input("Enter the number of arms: "))
means = [float(input(f"Enter the mean for arm {i + 1}: ")) for i in range(n)]
```

```
epsilon = float(input("Enter the epsilon value: "))
iterations = int(input("Enter the number of iterations: "))

# n-armed bandit example
cumulative_results = multi_arm_bandit_epsilon_greedy(means, epsilon, iterations)

# Plotting the results
for i in range(n):
    plt.plot(np.ones(iterations) * means[i], label=f'mean_arm_{i + 1}')

plt.plot(cumulative_results, label=f'eps={epsilon}')
plt.xlabel('Iteration')
plt.ylabel('Q-value')
plt.title('Epsilon Greedy for n-armed bandit')
plt.legend(loc='lower right')
plt.show()
```

```
Enter the number of arms: 4
Enter the mean for arm 1: 2
Enter the mean for arm 2: 1.5
Enter the mean for arm 3: 1.75
Enter the mean for arm 4: 1
Enter the epsilon value: 0.1
Enter the number of iterations: 300
```




```
No of times explored:  298
No of times exploited:  2
Mean of rewards from arm 1: 1.963458697963026
Mean of rewards from arm 2: 0.4024628507321192
Mean of rewards from arm 3: 0.989716317030424
Mean of rewards from arm 4: 0.08871626606074454
```



2. Understand the effect of the value of epsilon on the win rate by comparing the win rates corresponding to different values of epsilon. Hence draw conclusions.

```
plt.plot(c1,label='eps=0.1')
plt.plot(c2,label='eps=0.05')
plt.plot(c3,label='eps=0.01')
plt.xlabel('Iteration')
plt.ylabel('Q-value')
plt.title('Epsilon Greedy using different epsilon values')
plt.legend(loc='lower right')
```