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

SHIVAM NAGORI 60009210083 D12

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.

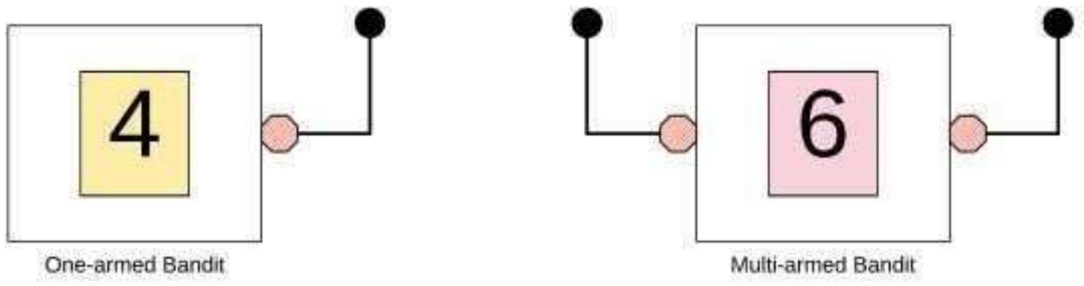
- Exploitation: Make the best decision given current information. The best long-term strategy may involve short-term sacrifices
- Exploration Gather more information. Gather enough information to make the best overall decisions

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).



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 to maximize returns.

Epsilon-greedy

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

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.

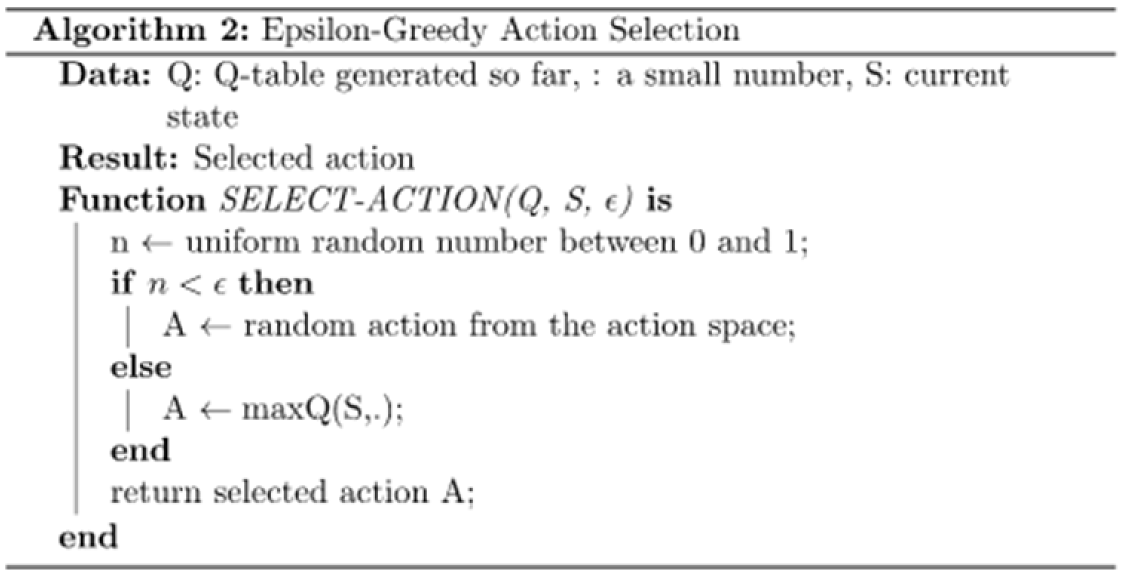
Usually, epsilon is set to be around 10%. Epsilon-Greedy can be represented as follows:

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

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, $\epsilon$) 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.

SOLUTION :

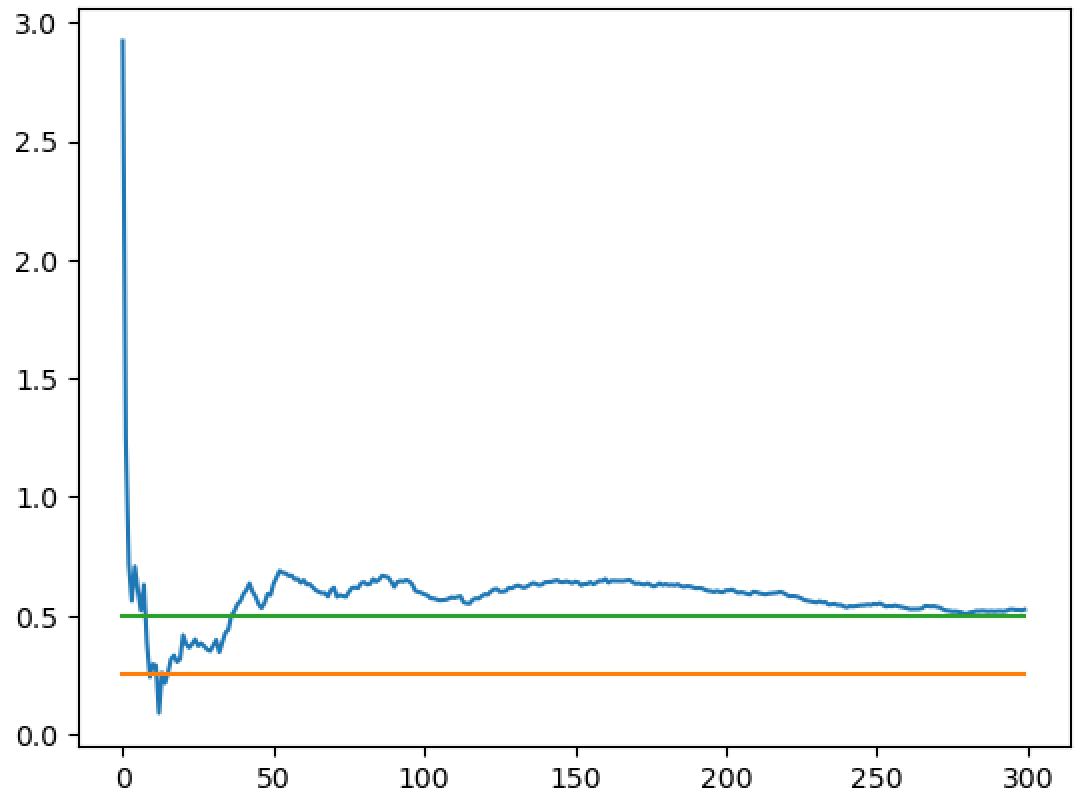
```
import numpy as np
import matplotlib.pyplot as plt
```

```
class Action:              #This class represents an action that can be taken in the bandit
problem.
  def __init__(self, m):
    self.m = m             #Initializes an action with a given mean m, which represents the
true mean reward of the action.
    self.mean = 0          #Represents the current estimate of the mean reward for the
action.
    self.N = 0             #Represents the number of times this action has been chosen.

  def select(self):        #Chooses the action and returns a reward.
    return np.random.normal(0,1) + self.m   #It randomly samples from a normal distribution
with mean self.m and standard deviation 1. This simulates the reward received when
selecting the action.

# Update the action-value estimate
  def update(self, x, alpha=0.1):   #Updates the action's estimate of the mean reward based
on a new observation x
    self.N += 1
    self.mean = self.mean + alpha*(x-self.mean) #simple incremental update rule that
updates the mean reward estimate using a step size alpha which is 0.1.
```

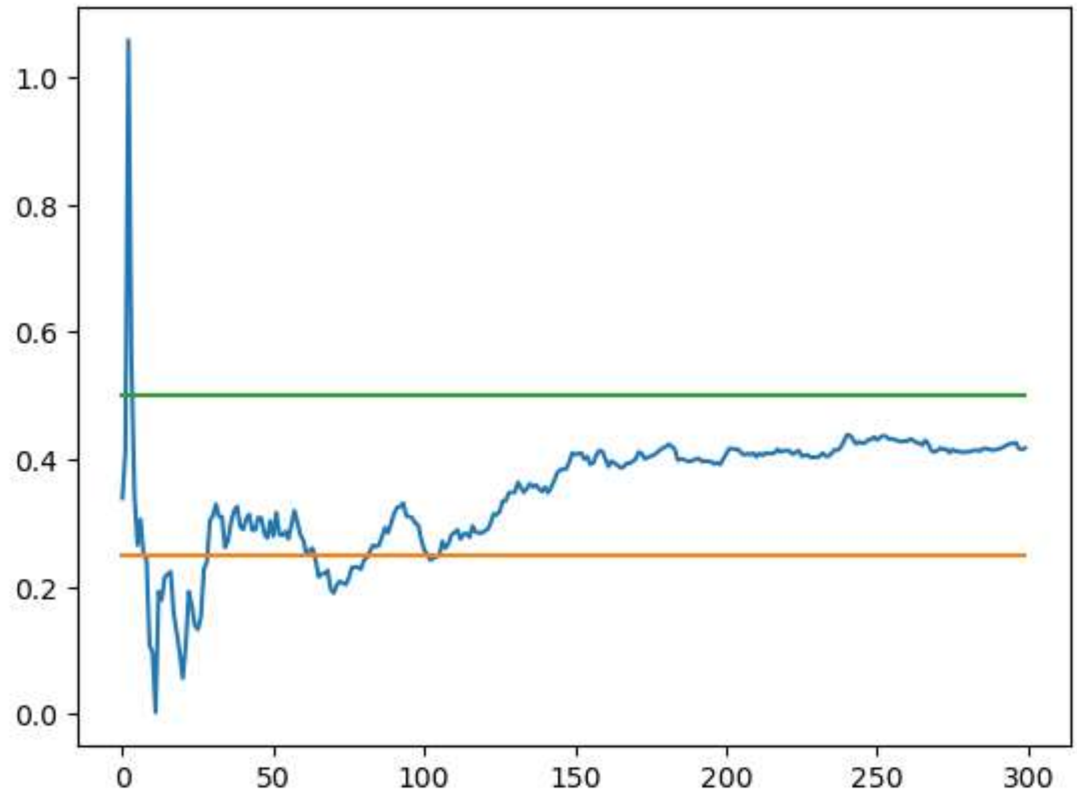
```
'''This function takes four parameters:
m1: The mean reward of the first action (arm).
m2: The mean reward of the second action (arm).
eps: The probability of choosing a random action (exploration rate).
```

```
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.
```

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

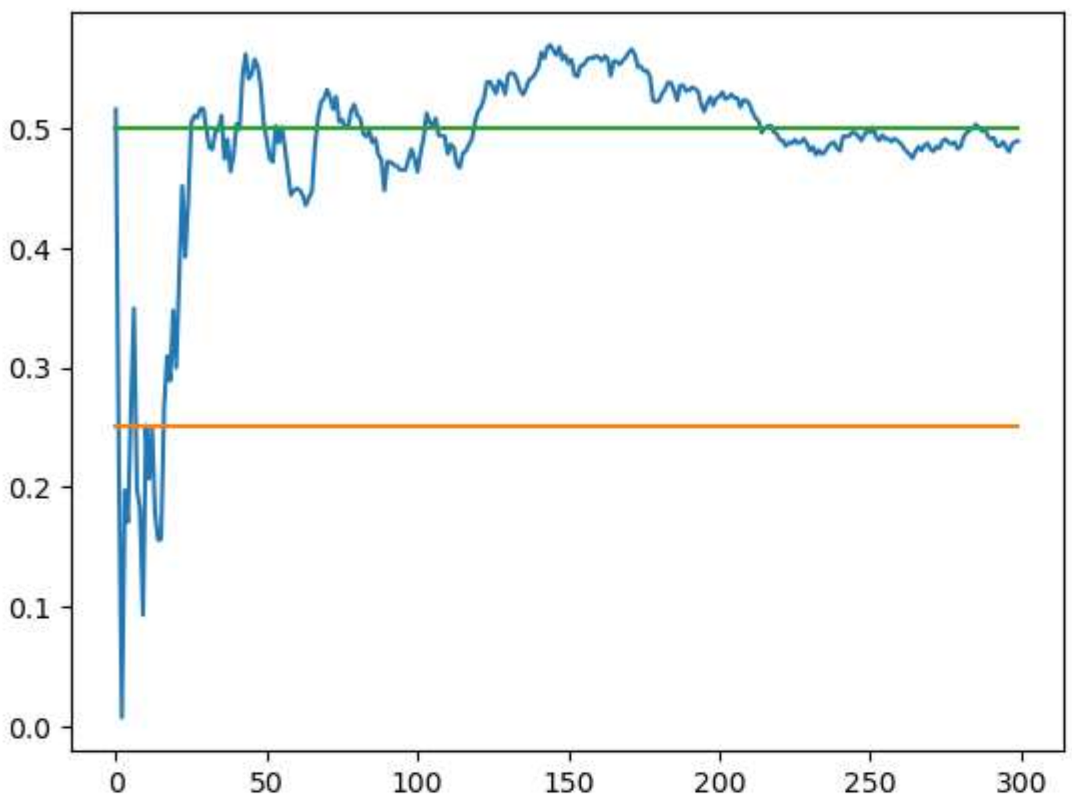


```
c2 = two_arm_bandit_epsilon_greedy(0.25, 0.5, 0.05, 300)
```



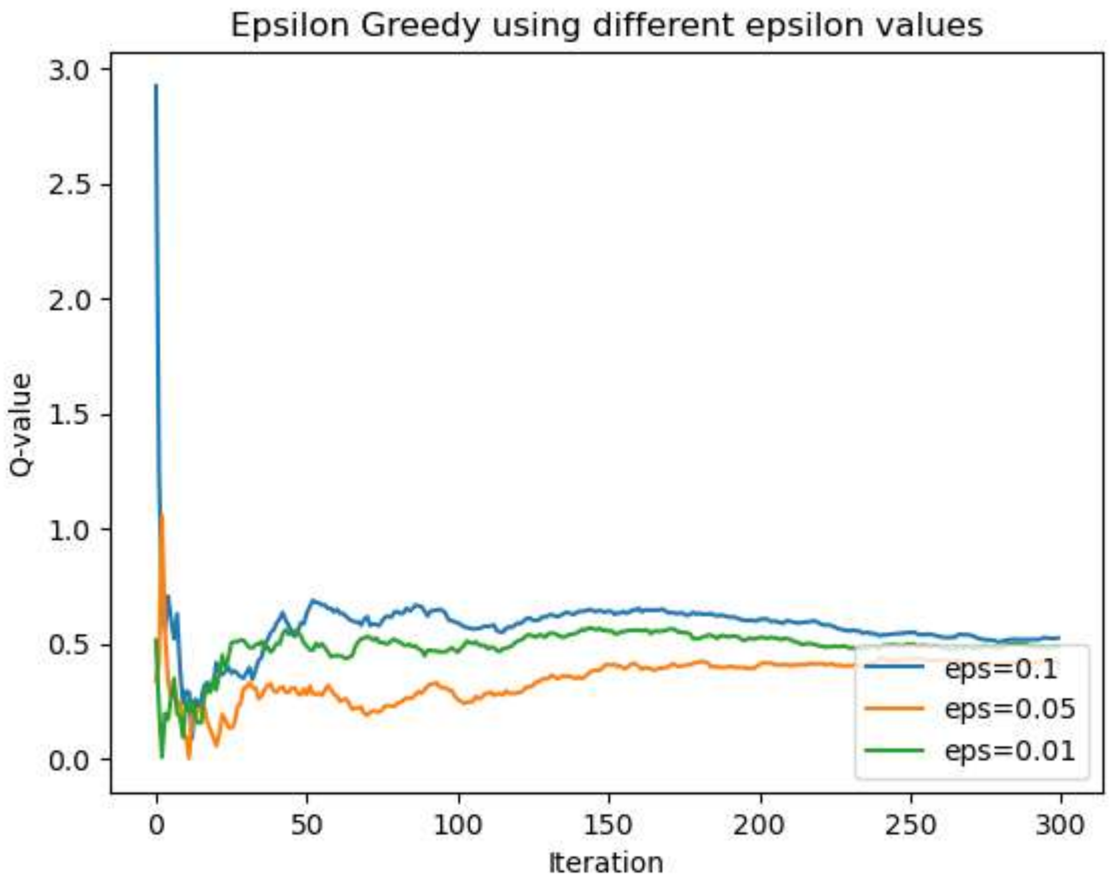
```
No of times explored:  8
No of times exploited:  292
Mean of rewards from arm 1: -0.1449035755721813
               Mean of rewards from arm 2: 0.39124503570478636
```

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



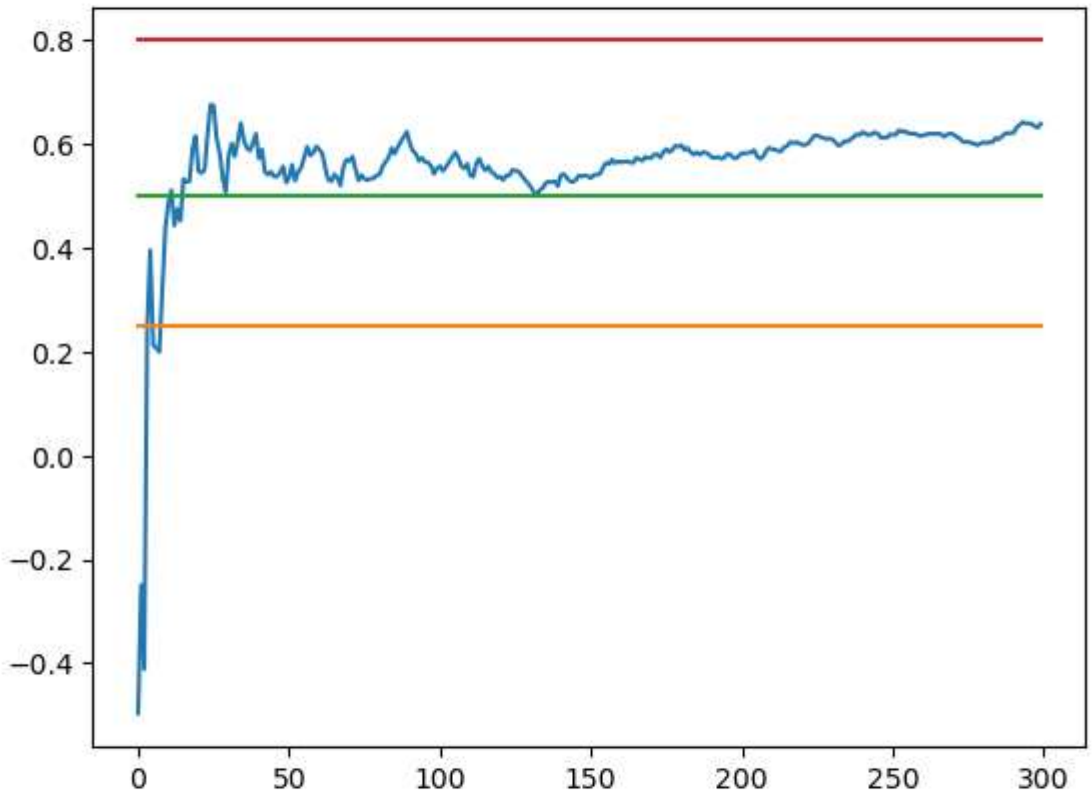
```
No of times explored:  5
No of times exploited:  295
Mean of rewards from arm 1: -0.14699517906610954
Mean of rewards from arm 2: 0.4900145589345406
```

```
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')
```



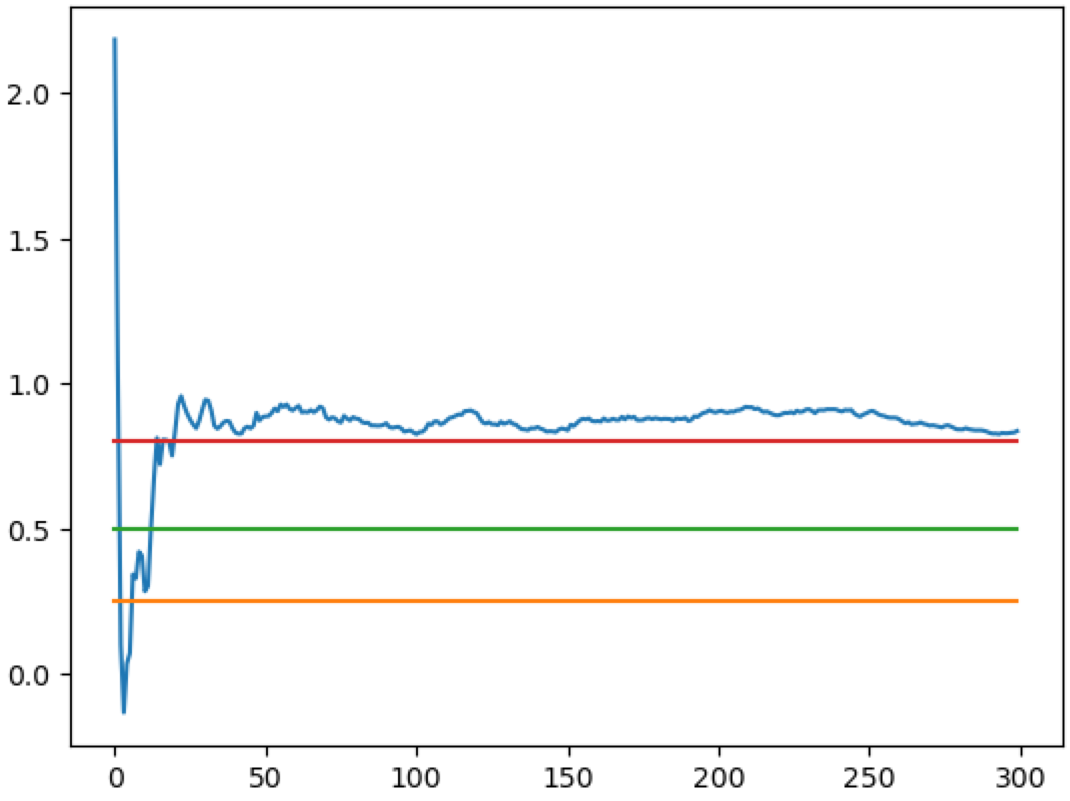
```
def three_arm_bandit_epsilon_greedy(m1, m2, m3, eps, N):
  actions = [Action(m1), Action(m2), Action(m3)]
  data = np.empty(N)
  explore,exploit=0,0
  for i in range(N):
    p = np.random.random()
    if p < eps:
      j = np.random.choice(3)
      explore+=1
    else:
      j = np.argmax([a.mean for a in actions])
      exploit+=1
    x = actions[j].select()
    actions[j].update(x,0.1)
    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()
  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
```

```
c1 = three_arm_bandit_epsilon_greedy(0.25, 0.5, 0.8, 0.1, 300)
```



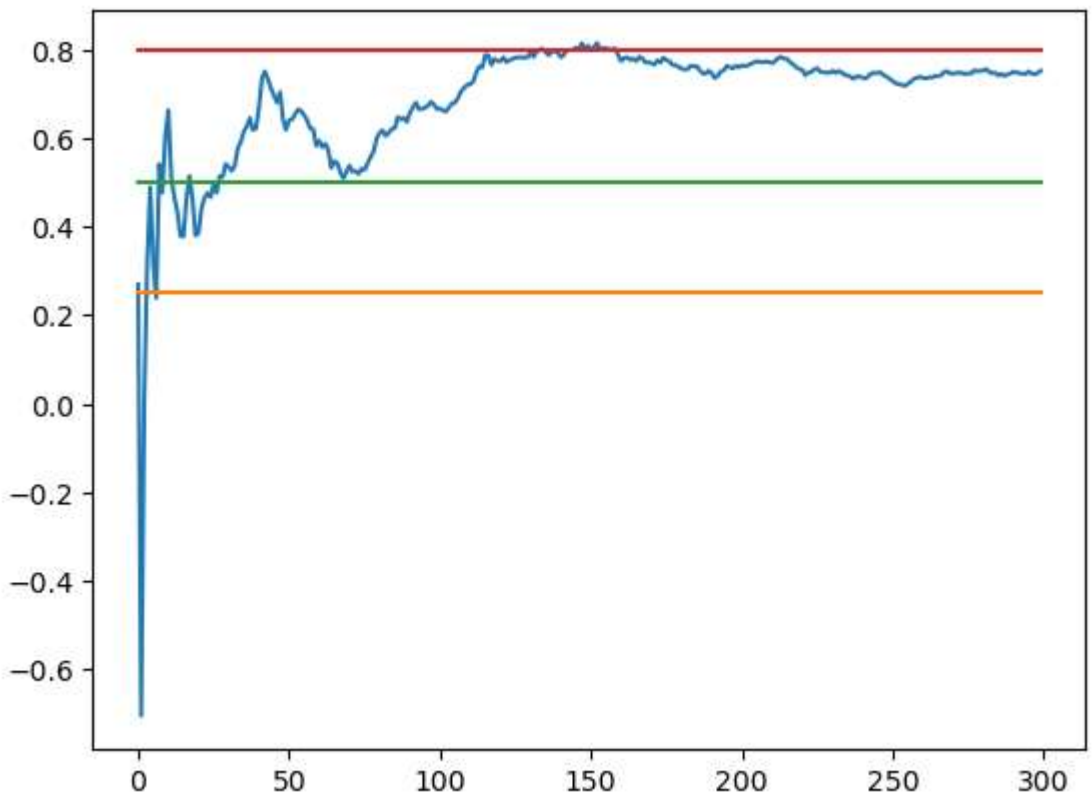
```
No of times explored:  19
No of times exploited:  281
Mean of rewards from arm 1: 0.28008428954686815
Mean of rewards from arm 2: 0.3278283252761306
Mean of rewards from arm 3: 0.9933814886908834
```

```
c2 = three_arm_bandit_epsilon_greedy(0.25, 0.5, 0.8, 0.05, 300)
```



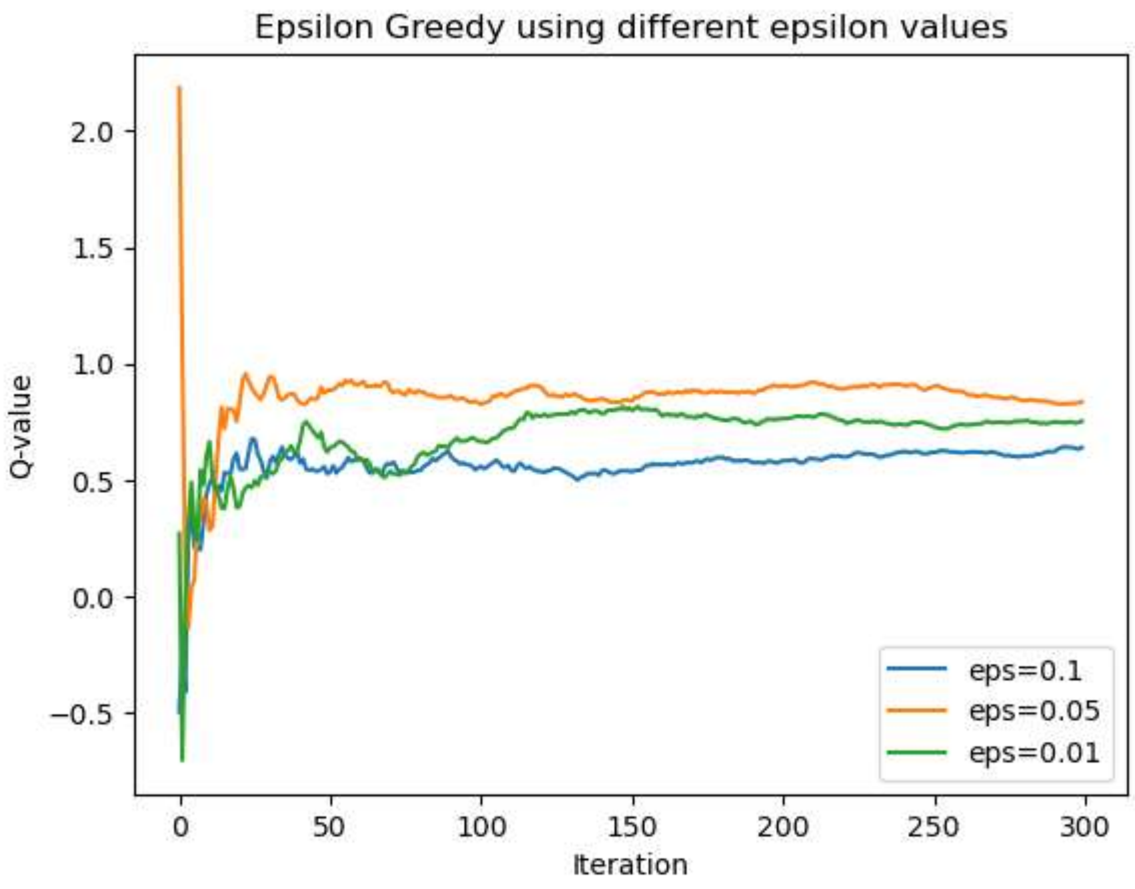
```
No of times explored:  19
No of times exploited:  281
Mean of rewards from arm 1: -0.05129024915452795
Mean of rewards from arm 2: 0.19489130087127066
Mean of rewards from arm 3: 0.9200113861667234
```

```
c3 = three_arm_bandit_epsilon_greedy(0.25, 0.5, 0.8, 0.01, 300)
```



```
No of times explored:  4
No of times exploited:  296
Mean of rewards from arm 1: -0.14360211926481306
Mean of rewards from arm 2: 0.03836324104485397
Mean of rewards from arm 3: 0.9151671066872358
```

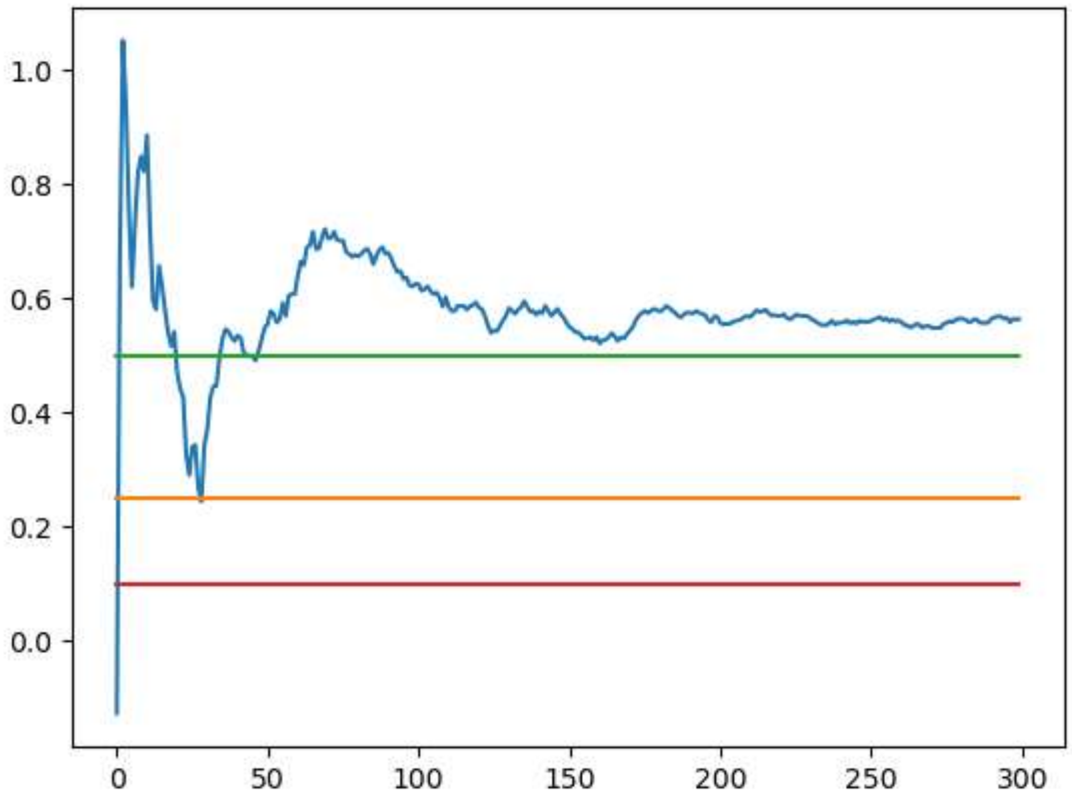
```
plt.plot(c1,label='eps=0.1')
plt.plot(c2,label='eps=0.05')
plt.plot(c3,label='eps=0.01')
plt.xlabel('Iteration')
plt.ylabel('Q-value')
plt.title('Epsilon Greedy using different epsilon values')
plt.legend(loc='lower right')
```



```
'''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 multi_arm_bandit_epsilon_greedy(m, eps, N):
    # m is the number of arms
  actions= []
  avgs = []
  for i in range(m):
    avg_value = float(input(f"Enter the average for the action {i+1} : "))
    avgs.append(avg_value)
    actions.append(Action(avg_value))


  # 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.
```

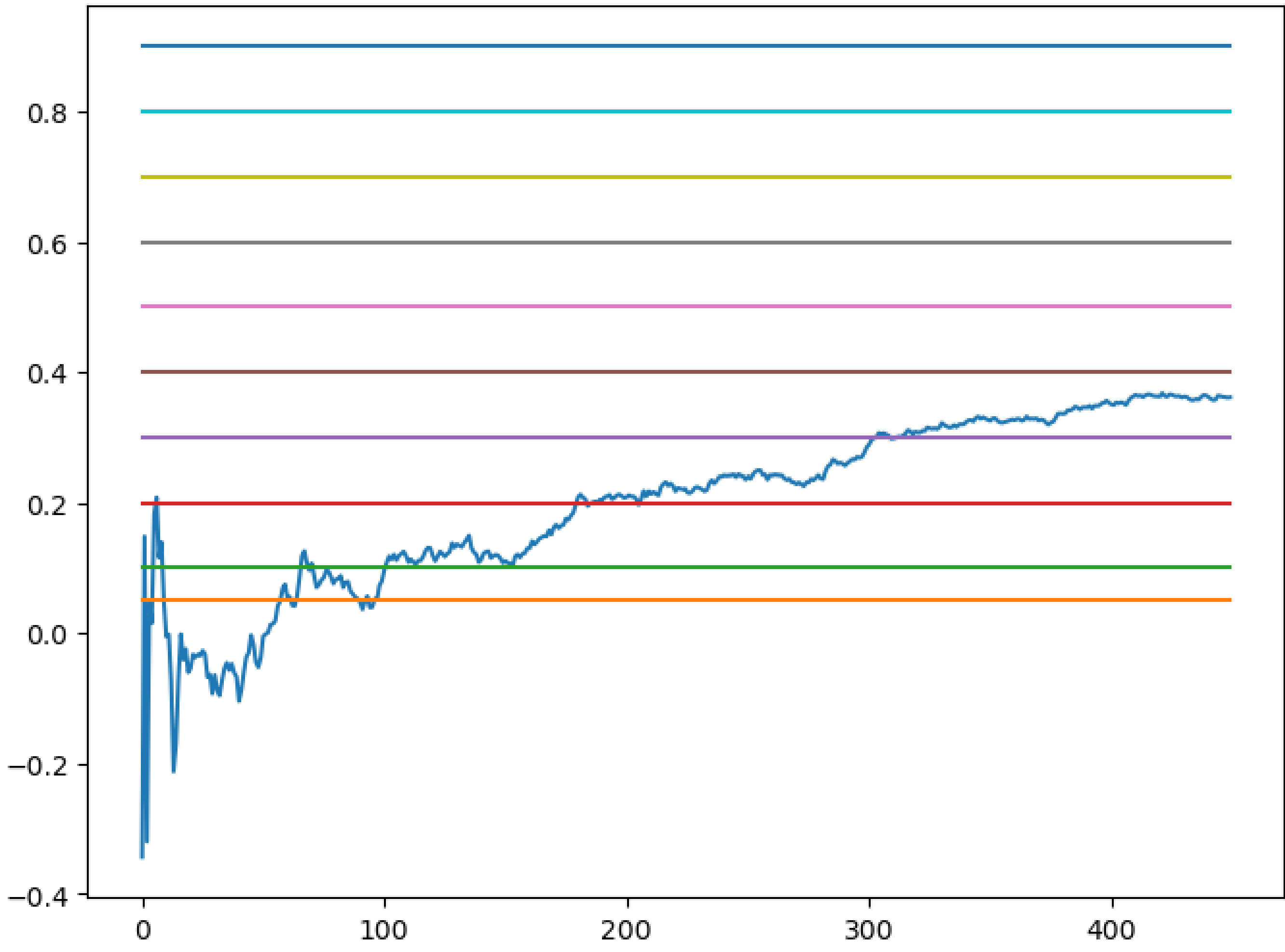
```
    if p < eps:                                               #If p is less than eps, the
algorithm chooses to explore by randomly selecting one of the actions.
      j = np.random.choice(2)                                 #'''If p is greater than or equal
to eps, the algorithm chooses to exploit 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)
  for i in range(len(actions)):
    plt.plot(np.ones(N)*avgs[i])
  # plt.plot(np.ones(N)*m2)
  plt.show()
  print("No of times explored: ",explore)
  print("No of times exploited: ",exploit)
  print("The value of eps : ", eps)
  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.
```

```
multi_arm_bandit_epsilon_greedy(3, 0.1, 300)
```



```
No of times explored:  35
No of times exploited:  265
The value of eps :  0.1
Mean of rewards from arm 1: 0.38428891992780423
Mean of rewards from arm 2: 0.4529211581880015
Mean of rewards from arm 3: -0.055288080895908254
```

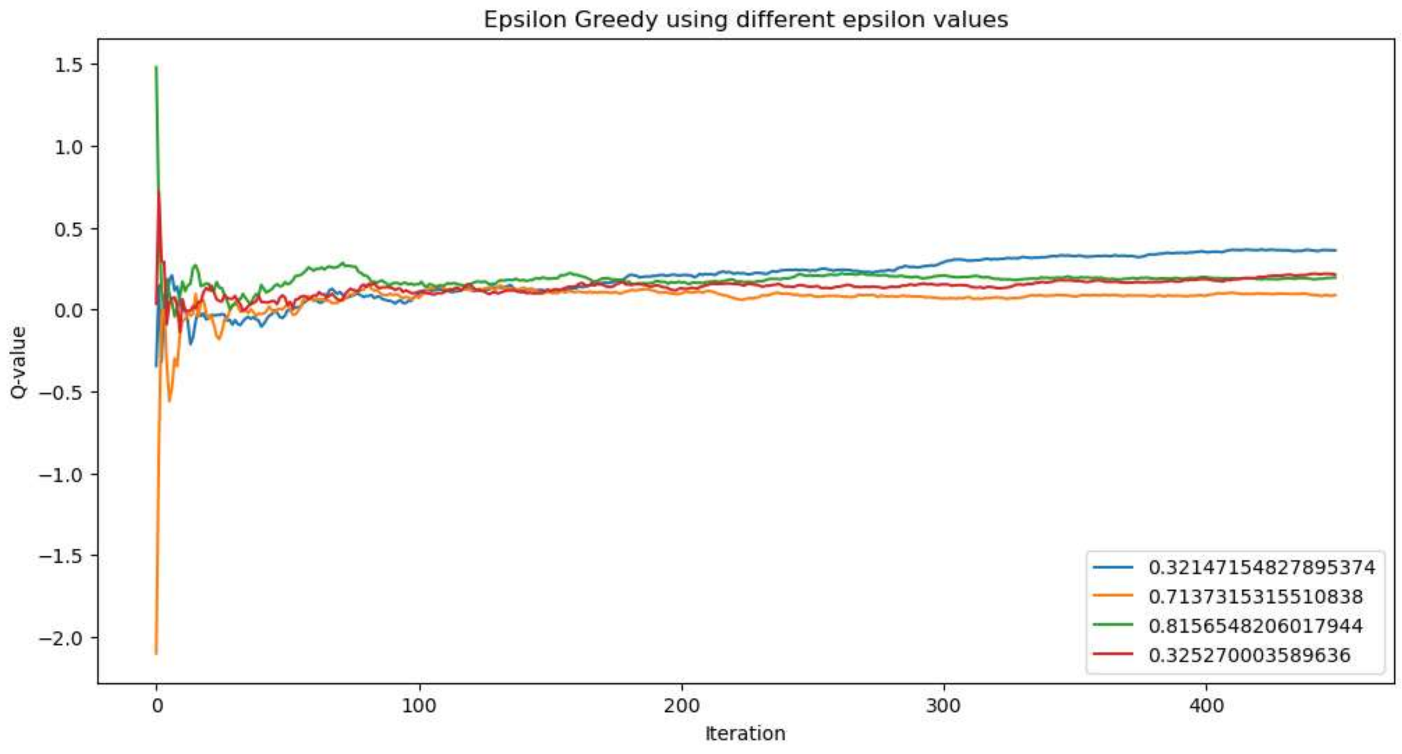
```python
epsilons = []
c = []

for i in range(4):
    fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(8, 6))

    # Generate a random epsilon value for each iteration
    eps = random.random()
    epsilons.append(eps)

    # Call your multi-arm bandit function
    c_av_value = multi_arm_bandit_epsilon_greedy(10, eps, 450)
    c.append(c_av_value)

    # Plot the cumulative average for the current subplot
    axes.plot(c[i], label="EPS : {}".format(eps))
    axes.set_title('Subplot {}'.format(i+1))
    axes.legend()

plt.show()
```



```
            No of times explored:  139
No of times exploited:  311
The value of eps :  0.32147154827895374
Mean of rewards from arm 1: 0.151579898589846
Mean of rewards from arm 2: -0.19881319169040007
Mean of rewards from arm 3: -0.04883479591076319
Mean of rewards from arm 4: -0.014223432536913938
Mean of rewards from arm 5: -0.04161819417500309
Mean of rewards from arm 6: -0.01597629845596365
Mean of rewards from arm 7: -0.09515774688533733
Mean of rewards from arm 8: -0.11957667684958412
Mean of rewards from arm 9: 0.6613022353570643
Mean of rewards from arm 10: 0
```

```python
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(12, 6))

# Your existing code for plotting
plt.plot(c[0], label='{}'.format(epsilons[0]))
plt.plot(c[1], label='{}'.format(epsilons[1]))
plt.plot(c[2], label='{}'.format(epsilons[2]))
plt.plot(c[3], label='{}'.format(epsilons[3]))

plt.xlabel('Iteration')
plt.ylabel('Q-value')
plt.title('Epsilon Greedy using different epsilon values')
plt.legend(loc='lower right')
```

```
# Show the plot
plt.show()
```


Epsilon Greedy using different epsilon values
Q-value
Iteration