In [1]:
```python
import matplotlib.pyplot as plt
```

In [2]:
```python
"""
Environment for the multi-armed bandit problem
"""

import numpy as np

class bandit_env():
    """
    Initialize the multi-arm bandit environment.
    :params:
    r_mean: takes a list of reward mean
    r_stddev: takes a list of reward standard deviation
    """
    def __init__(self, r_mean, r_stddev):
        if len(r_mean) != len(r_stddev):
            raise ValueError("Reward distribution parameters (mean and variance) mus

        if any(r <= 0 for r in r_stddev):
            raise ValueError("Standard deviation in rewards must all be greater than

        self.n = len(r_mean)
        self.r_mean = r_mean
        self.r_stddev = r_stddev

    def pull(self, index_arm):
        """
        Performs the action of pulling the arm/lever of the selected bandit
        :inputs:
        index_arm: the index of the arm/level to be pulled
        :outputs:
        reward: the reward obtained by pulling tht arm (sampled from their correspon
        """
        reward = np.random.normal(self.r_mean[index_arm], self.r_stddev[index_arm])
        return reward
```

Q1.

In [3]:
```python
true_reward = [2.5, -3.5, 1.0, 5.0, -2.5]
std_deviation = [0.33, 1.0, 0.66, 1.98, 1.65]
```

In [4]:
```python
env = bandit_env(true_reward,std_deviation)
```

Epsilon Greedy Algorithm

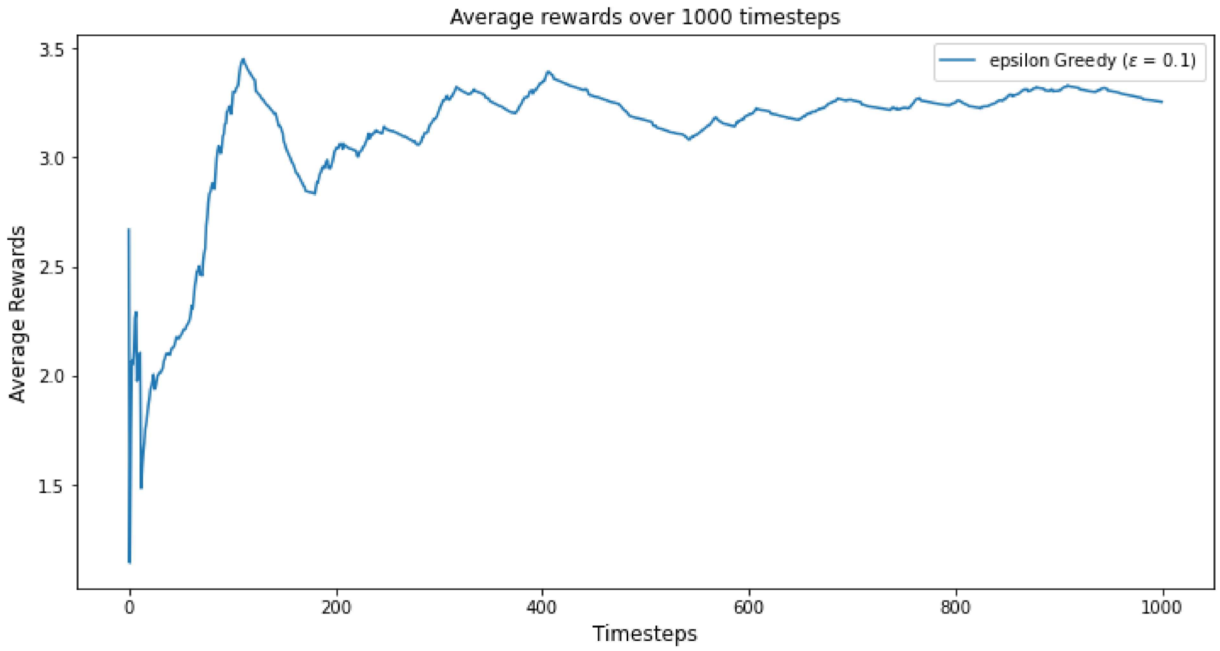
In [5]:
```python
class Actions_epsilon:
    def __init__(self,a):
        self.a = a
        self.Qt_a = 0
        self.N_a = 0

    def update(self,R_a):
        self.N_a += 1
        self.Qt_a = (self.Qt_a + R_a)/self.N_a
```

In [6]:

```
def epsilon_greedy_experiment(eps,N):

    actions = [Actions_epsilon(1),Actions_epsilon(2),Actions_epsilon(3),Actions_epsi
    returns = []

    for i in range(1,N+1):
        # epsilon greedy
        p = np.random.random()
        if p < eps:
            j = np.random.choice(5)
        else:
            j = np.argmax([act.Qt_a for act in actions])

        x = actions[j]
        R_a = env.pull(j)
        x.update(R_a)

        returns.append(R_a)

    cumulative_average = np.cumsum(returns)/(np.arange(N) + 1)

    # plot
    plt.figure(figsize=[12, 6])
    plt.plot(cumulative_average, label = r"epsilon Greedy ($\epsilon$ = " + f"{eps})
    plt.title(f"Average rewards over {N} timesteps", fontsize=12)
    plt.xlabel("Timesteps", fontsize=12)
    plt.ylabel("Average Rewards", fontsize=12)
    plt.legend()
```

In [7]:

```
c_1 = epsilon_greedy_experiment(0.1,1000)
```



UCB algorithm

In [8]:

```
class Actions_UCB:

    def __init__(self,a):
        self.a = a
        self.Qt_a = 0
        self.N_a = 0
        self.ucb_Qt = 0
```

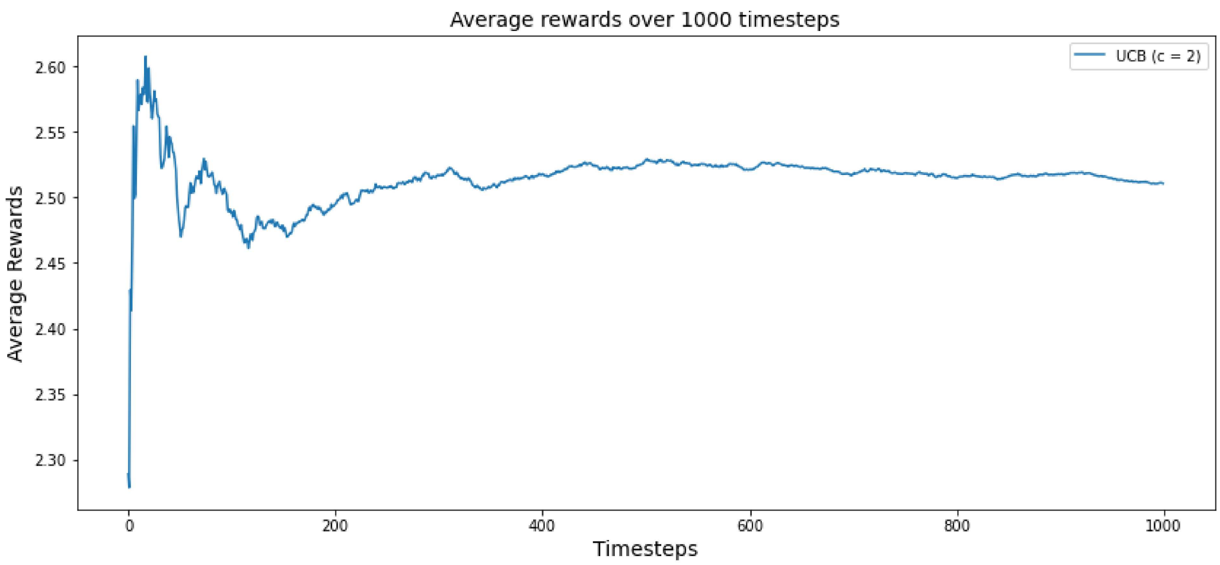
```python
    def update_estimate(self,R_a):
        self.N_a += 1
        self.Qt_a = (self.Qt_a + R_a)/self.N_a

    def update_ucb(self,c,t):
        if self.N_a > 0:
            self.ucb_Qt = self.Qt_a + c * np.sqrt(np.log(t)/self.N_a)
        else:
            self.ucb_Qt = 1e400
```

In [9]:

```python
def UCB_experiment(N,c):
    actions_ucb = [Actions_UCB(1),Actions_UCB(2),Actions_UCB(3),Actions_UCB(4),Actio

    returns = []

    # ucb algorithm
    for t in range(1, N + 1):
        j = np.argmax([act.ucb_Qt for act in actions_ucb])
        x = actions_ucb[j]
        R_a = env.pull(j)
        x.update_estimate(R_a)
        x.update_ucb(c,t)
        returns.append(R_a)


    cumulative_average = np.cumsum(returns)/(np.arange(N) + 1)

    plt.figure(figsize=[14, 6])
    plt.plot(cumulative_average, label=f"UCB (c = {c})")
    plt.title(f"Average rewards over {N} timesteps", fontsize=14)
    plt.xlabel("Timesteps", fontsize=14)
    plt.ylabel("Average Rewards", fontsize=14)
    plt.legend()
```

In [10]:

```python
ucb = UCB_experiment(1000,2)
```



Greedy with the optimistic initial value method

In [11]:

```python
class Actions_optimistic:
    def __init__(self,a,initial_action_value):
        self.a = a
```

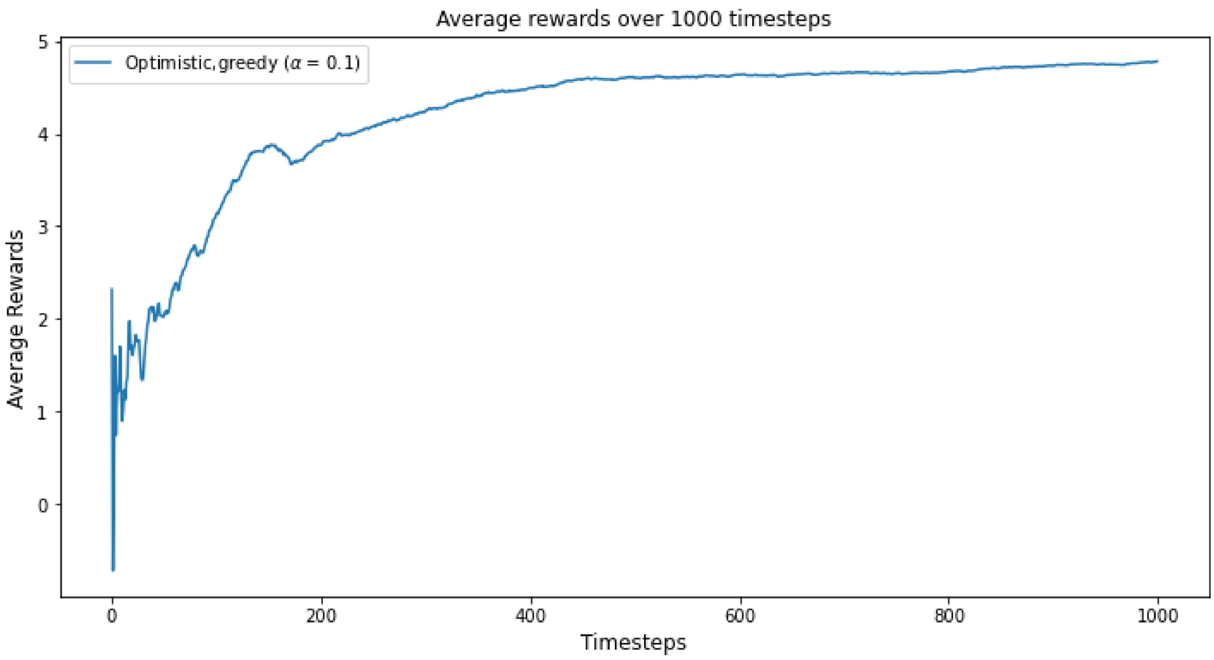
```python
        self.Qt_a = initial_action_value
        self.N_a = 1

    def update_oiv(self,R_a,alpha):
        self.N_a += 1
        self.Qt_a = self.Qt_a + alpha*(R_a - self.Qt_a)
```

In [12]:

```python
def optimistic_intial_experiment(N,initial_action_value,alpha):
    actions_oiv = [Actions_optimistic(1,initial_action_value),Actions_optimistic(2,i
    returns = []

    for i in range(1,N+1):
        # optimistic initial values
        j = np.argmax([act.Qt_a for act in actions_oiv])
        x = actions_oiv[j]
        R_a = env.pull(j)
        x.update_oiv(R_a,alpha)
        returns.append(R_a)


    cumulative_average = np.cumsum(returns)/(np.arange(N) + 1)

    # plot
    plt.figure(figsize=[12, 6])
    plt.plot(cumulative_average, label = r"Optimistic,greedy ($\alpha$ = " + f"{alph
    plt.title(f"Average rewards over {N} timesteps", fontsize=12)
    plt.xlabel("Timesteps", fontsize=12)
    plt.ylabel("Average Rewards", fontsize=12)
    plt.legend()
```

In [13]:

```python
oiv = optimistic_intial_experiment(1000,10,0.1)
```



Gradient bandit algorithm

In [14]:

```python
def Probability(H):
    Pr_a = np.exp(H)/np.sum(np.exp(H))
    return Pr_a
```

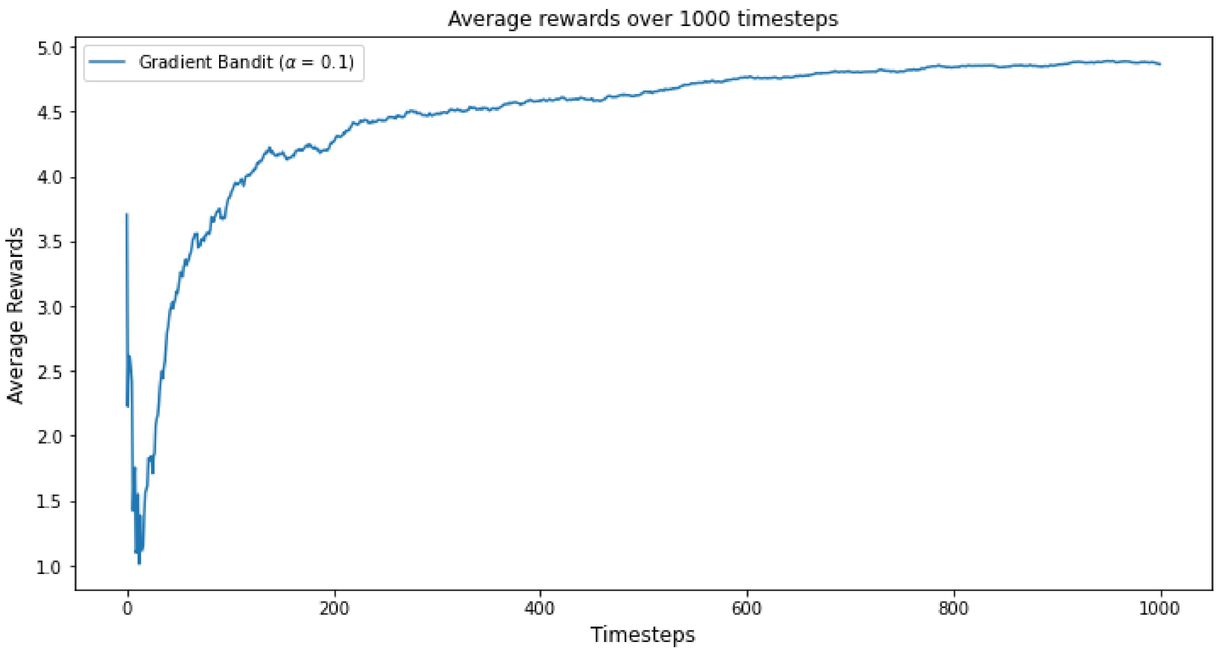
```python
def update_prefrence(R_t, R_tbar, a, alpha, H, prob):
    actions = np.arange(5)
    for act in actions:
        if act == a:
            H[act] = H[act] + alpha*(R_t - R_tbar)*(1 - prob[act])
        else:
            H[act] = H[act] - alpha * (R_t - R_tbar) * prob[act]
    return H
```

In [15]:

```python
def gradient_bandit(N,alpha):
    H = np.zeros(5)
    returns = []
    R_tbar = 0
    for i in range(1, N+1):
        prob = Probability(H)
        a = np.random.choice(5, p=prob)
        R_t = env.pull(a)
        H = update_prefrence(R_t, R_tbar, a, alpha, H, prob)
        returns.append(R_t)
        R_tbar = np.average(returns)

    cumulative_average = np.cumsum(returns)/(np.arange(N) + 1)

    plt.figure(figsize=[12, 6])
    plt.plot(cumulative_average, label=r"Gradient Bandit ($\alpha$ = " + f"{alpha})"
    plt.title(f"Average rewards over {N} timesteps", fontsize=12)
    plt.xlabel("Timesteps", fontsize=12)
    plt.ylabel("Average Rewards", fontsize=12)
    plt.legend()
```

In [16]:

```python
gb = gradient_bandit(1000,0.1)
```



plot of average over 1000 steps of all algorithms

In [17]:

```python
# eps
def epsilon_greedy_experiment(eps,N = 1000):

    actions = [Actions_epsilon(1),Actions_epsilon(2),Actions_epsilon(3),Actions_epsi
    returns = []
```

```
    avg_reward = 0
    for i in range(1,N+1):
        # epsilon greedy
        p = np.random.random()
        if p < eps:
            j = np.random.choice(5)
        else:
            j = np.argmax([act.Qt_a for act in actions])

        x = actions[j]
        R_a = env.pull(j)
        x.update(R_a)

        returns.append(R_a)

    cumulative_average = np.cumsum(returns)/(np.arange(N) + 1)

    avg_reward = np.average(returns)
    return avg_reward

# ucb
def UCB_experiment(c,N = 1000):
    actions_ucb = [Actions_UCB(1),Actions_UCB(2),Actions_UCB(3),Actions_UCB(4),Actio

    returns = []
    avg_reward = 0
    # ucb algorithm
    for t in range(1, N + 1):
        j = np.argmax([act.ucb_Qt for act in actions_ucb])
        x = actions_ucb[j]
        R_a = env.pull(j)
        x.update_estimate(R_a)
        x.update_ucb(c,t)
        returns.append(R_a)


    cumulative_average = np.cumsum(returns)/(np.arange(N) + 1)
    avg_reward = np.average(returns)
    return avg_reward


# opiv
def optimistic_intial_experiment(initial_action_value,alpha = 0.1,N = 1000):
    actions_oiv = [Actions_optimistic(1,initial_action_value),Actions_optimistic(2,i

    returns = []
    avg_reward = 0
    for i in range(1,N+1):
        # optimistic initial values
        j = np.argmax([act.Qt_a for act in actions_oiv])
        x = actions_oiv[j]
        R_a = env.pull(j)
        x.update_oiv(R_a,alpha)
        returns.append(R_a)


    cumulative_average = np.cumsum(returns)/(np.arange(N) + 1)

    avg_reward = np.average(returns)
    return avg_reward


# gradiant
def gradient_bandit(alpha,N = 1000):
```

```
    H = np.zeros(5)
    returns = []
    R_tbar = 0
    for i in range(1, N+1):
        prob = Probability(H)
        a = np.random.choice(5, p=prob)
        R_t = env.pull(a)
        H = update_prefrence(R_t, R_tbar, a, alpha, H, prob)
        returns.append(R_t)
        R_tbar = np.average(returns)

    cumulative_average = np.cumsum(returns)/(np.arange(N) + 1)

    return R_tbar
```

In [18]:

```
x_axis = [1/128,1/64,1/32,1/16,1/8,1/4,1/2,1,2,4]

eps_greedy = []
ucb_ = []
oiv_ = []
gb = []


for i in x_axis :
    eps_greedy.append(epsilon_greedy_experiment(i))
    ucb_.append(UCB_experiment(i))
    oiv_.append(optimistic_intial_experiment(i))
    gb.append(gradient_bandit(i))

# plotting
plt.figure(figsize = (12,6)) # to change the size of graph

plt.plot(eps_greedy,x_axis,'b',label="Epsilon Greedy")
plt.plot(oiv_,x_axis,'r',label="Optimistic Greedy")
plt.plot(ucb_,x_axis,'y',label="UCB")
plt.plot(gb,x_axis,'g',label="Gradient Bandit")

plt.ylabel("Average Reward over first 1000 steps")
plt.xlabel("e,alpha,c,Qo")
plt.legend()
plt.show()
```