Please refer to the *readme.text* for running the code.

Answer for Q1:

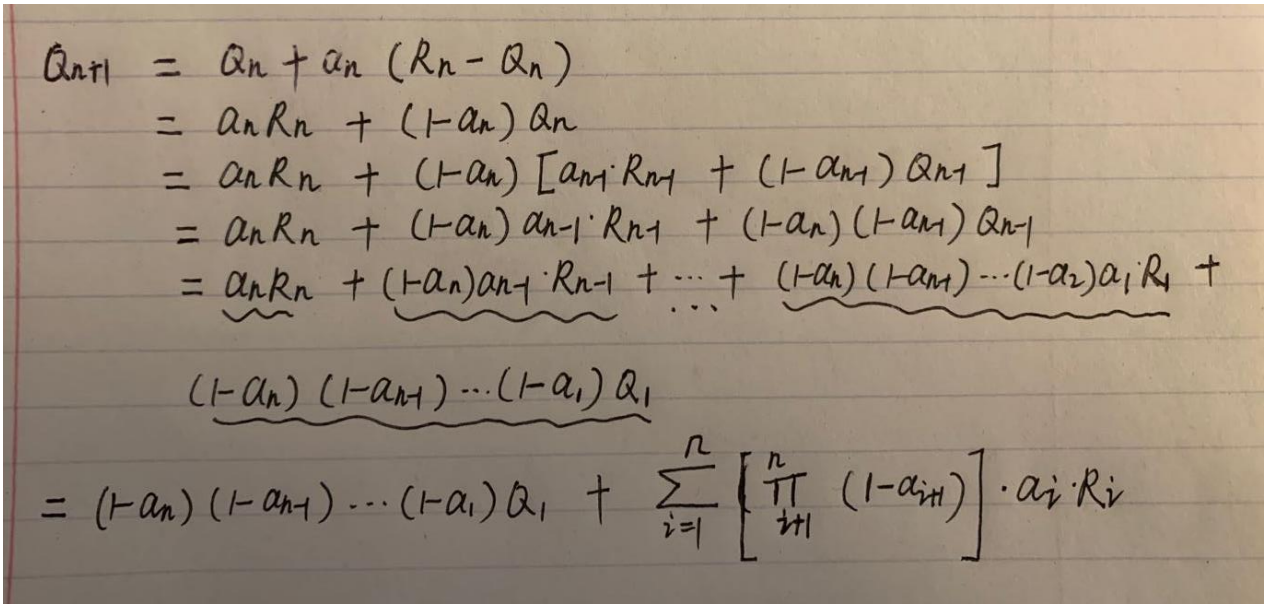
Step number	Q1	Q2	Q 3	Q4	action taken	reward	episolon case(random select) or definite?
1	0	0	0	0	1	-1	one of the max values but possibly happen
2	-1	0	0	0	2	1	one of the max values but possibly happen
3	-1	1	0	0	2	-2	max value and exploitation
4	-1	-0.5	0	0	2	2	epsilon case
5	-1	0.333	0	0	3	0	epsilon case

Details about the answer:

At time step 1, the agent selected the action 1 and it could be either epsilon case (select the action randomly) or it might be the exploitation case (take the max-reward action). Step 2 is the same as step 1. At step 3, it takes the max-value action and it is the exploitation case. At step 4 and 5, it is the epsilon case (select the action randomly).

Answer for Q2:

The <u>underline</u> in the picture is used to emphasize some critical terms.



Answer for Q3:

(a) Sample-average estimate is unbiased.

$\because Q_n \doteq \frac{R_1 + \cdots + R_{n-1}}{n-1}$, where $R_1 \cdots R_{n-1}$ are IID.

$$E(Q_n) = \frac{1}{n-1} E(R_1 + \cdots + R_{n-1})$$

$$= \frac{1}{n-1} \left(E(R_1) + \cdots + E(R_{n-1}) \right) \quad (1)$$

$\because E(R_1) = E(R_2) = \cdots = E(R_{n-1}) = q_*$ (2)

$\therefore$ from equation (1) and (2), $E(Q_n) = \frac{1}{n-1} \cdot (n-1) q_* = q_*$

(b) Exponential recency-weighted average estimate:

$$Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i$$

When $Q_1 = 0$, $E(Q_{n+1}) = E\left(\sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i \right)$

$$= \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} \cdot E(R)$$

$$= \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} \cdot q_*$$

if $\sum_{i=1}^{n} \alpha (1-\alpha)^{n-i}$ equals to 1, Q_n will be unbiased.

if $\sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} \neq 1$, Q_n would be biased.

(c) Following part (b), $E(Q_{n+1}) = (1-\alpha)^n E(Q_1) + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} \cdot q_*$

We want find the solution for $E(Q_{n+1}) = q_*$

$$\begin{cases} (1-\alpha)^n E(Q_1) = 0 \\ \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1 \end{cases} \longrightarrow \begin{cases} Q_1 = 0 \text{ or } E(Q_1) = 0 \\ \alpha = 1 \end{cases} \longrightarrow \text{following} \quad ①$$

$$\sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = \alpha \cdot \frac{1-(1-\alpha)^n}{1-(1-\alpha)} = \alpha \cdot \frac{1-(1-\alpha)^n}{\alpha} = 1-(1-\alpha)^n \quad ②$$

if we want $1-(1-\alpha)^n = 1 \longrightarrow \alpha = 1$ ③

From ①, ②, ③, we could get that the unbiased condition is $\alpha = 1$

(d) $E(Q_{n+1}) = (1-\alpha)^n E(Q_1) + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} \cdot q_*$ ①

$\because 0 < \alpha < 1$

$\lim_{n\to\infty} (1-\alpha)^n E(Q_1) = 0$ ②

$\lim_{n\to\infty} \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} \cdot q_* = \lim_{n\to\infty} [1-(1-\alpha)^n] \cdot q_* = q_*$ ③

From ①, ②, ③, we could find Q_n is asymptotically unbiased.

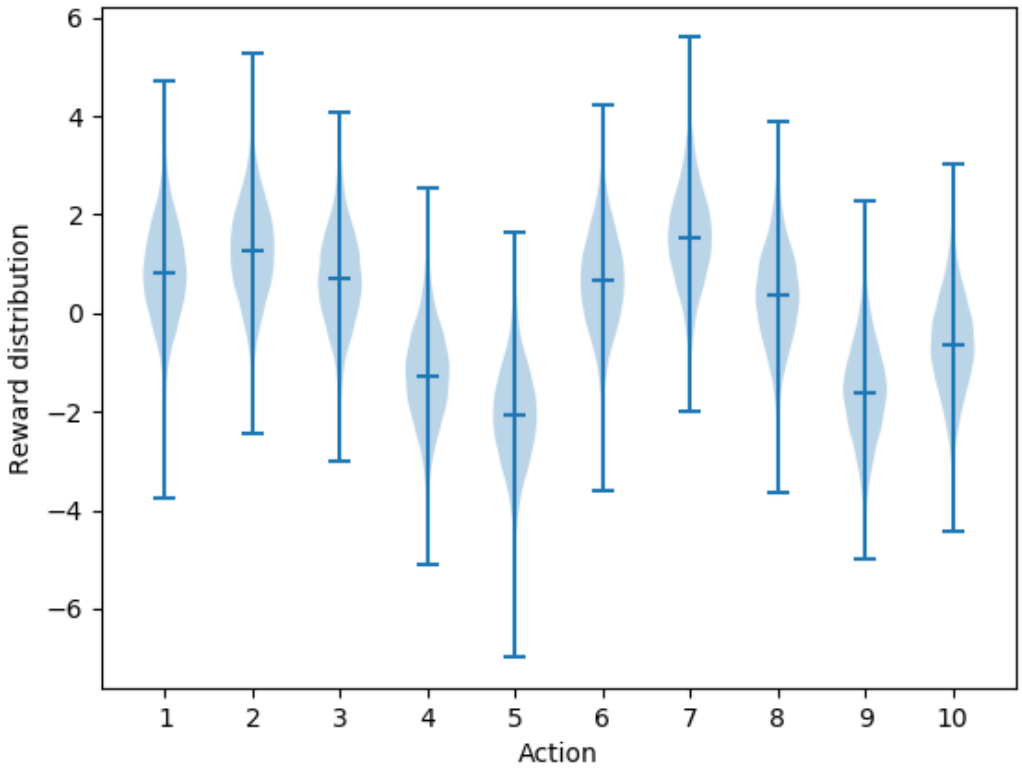
(e) In general, we couldn't set $\alpha = 1$. It's due to that when $\alpha = 1$, the stepsize is too big and the agent forgets everything before the current step, which means it doesn't learn

Another reason is that we couldn't use infinite n in practice.

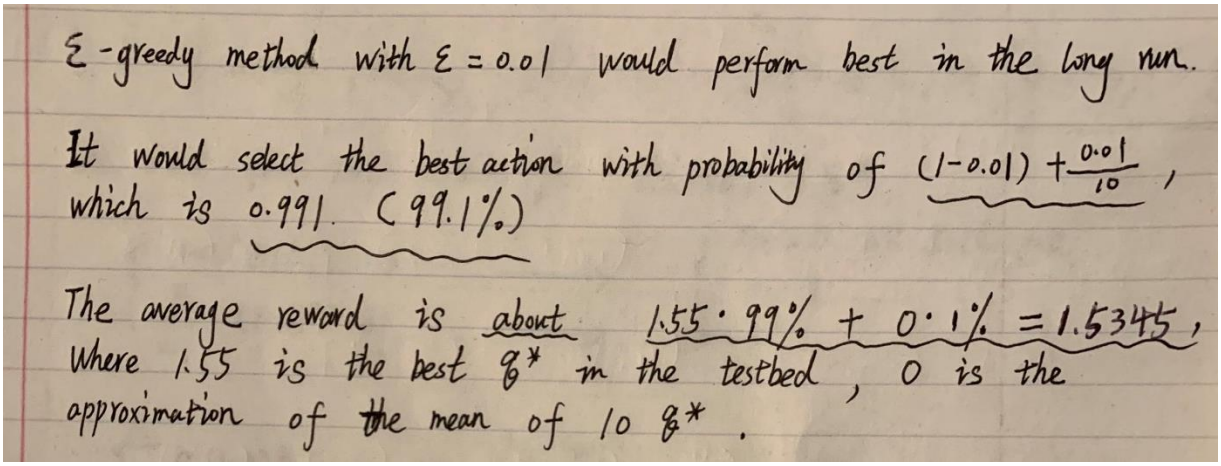
Answer for q4:

The code *testbed.py* is used to generate the data. It would pull each arm 10000 times and store the data with size of 10 by 10000 (each row is corresponding to one arm).

The code *plot_violin.py* would load the data and plot the figure as below:



Answer for q5:



While epsilon=0.1, the agent would select the best action with (1-10%) + 10%/10 =91% probability

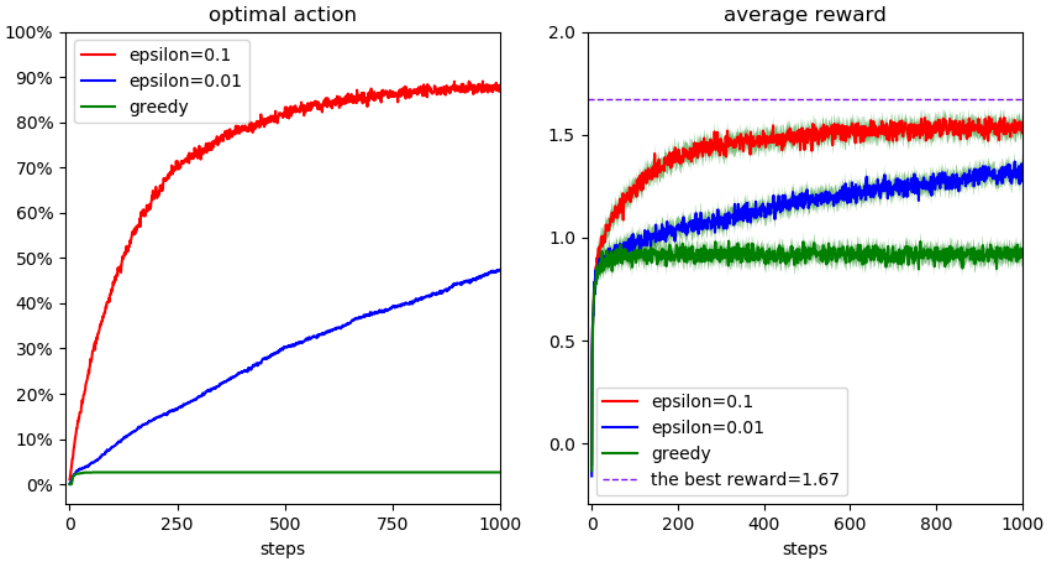
I used "medium setting" for most of the tasks except q6, q7 and there is a specific description in each answer.

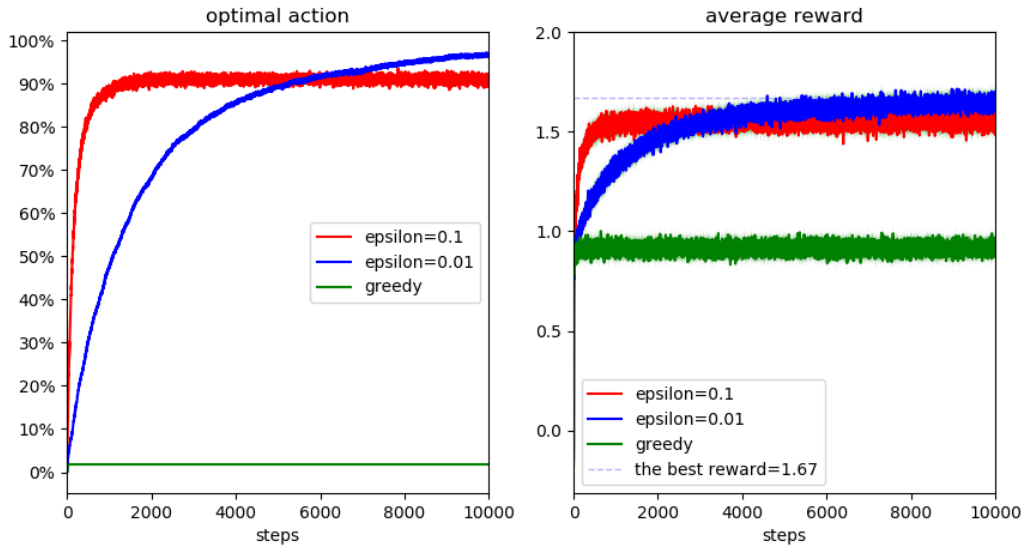
Answer for q6:

The code *epsilon.py* is used to generate the data. It would run 2000 trials with each trial has 1000 steps and store the data into 2000-by-1000 array. I also tried 2000 runs of 10000 steps to find the final performance of epsilon=0.01.

The code *epsilon_plot.py* is used to plot the data as follows:

(the 95% confidence bound is in shallow green in the average reward graph, not easy to see in the second graph)



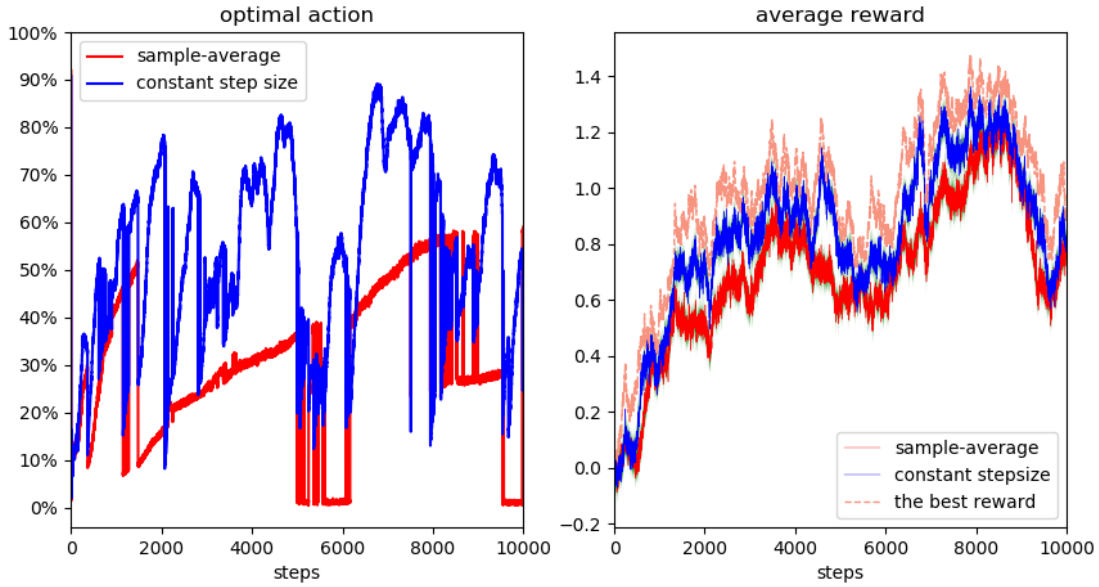


Answer for q7:

The code *nonstationary.py* is used to generate the data. It would run 2000 trials with each trial has 10000 steps and store the data into 2000-by-10000 array.

The code *nonstationary_plot.py* is used to plot the data as follows:

(the 95% confidence bound is in shallow green in the average reward graph)



The sample-average method was drawn in red line, the constant step size was drawn in blue line, and the confident bound was drawn in shallow green in the average reward graph.

Details about the q7:

Firstly, I used a 10000 by 10 matrix to design the increment of q* value at each step. Each row is corresponding with one step's increment. And each column has 10000 points sampled from a normal distribution with mean of 0 and std of 0.001 dependently.

Secondly, each of the 2000 runs used the same increment matrix mentioned above.

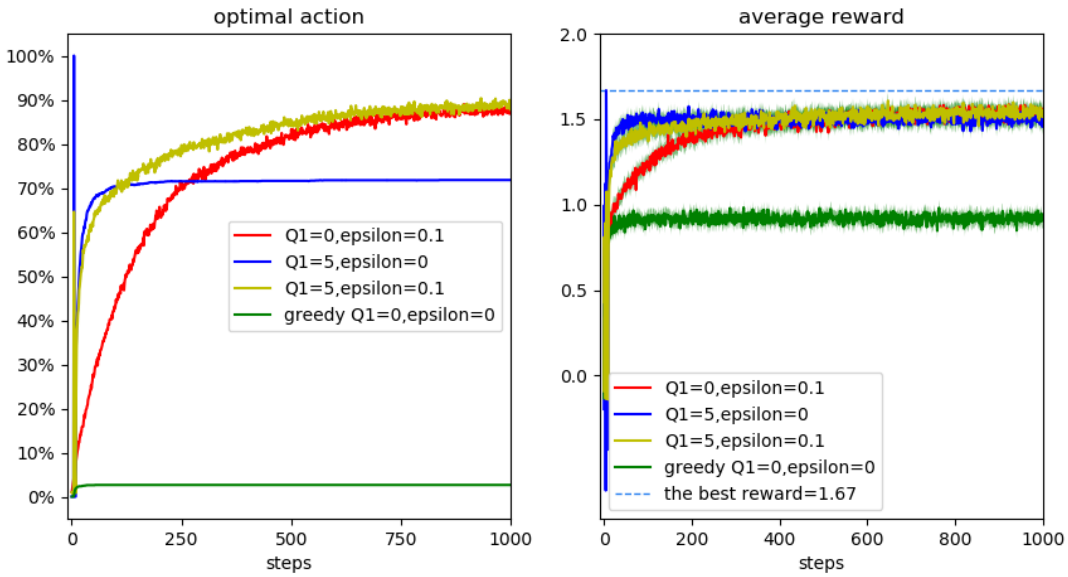
Thirdly, the blue curve performs better than the red curve, which indicated that the constant step size is better than the average sample method for the nonstationary problem.

Answer for q8: The optimistic initial values part

The code *oi.py* is used to generate the data. It would run 2000 trials with each trial has 1000 steps and store the data into 2000-by-1000 array.

The code *oi_plot.py* is used to plot the data as follows:

(the 95% confidence bound is in shallow green in the average reward graph)

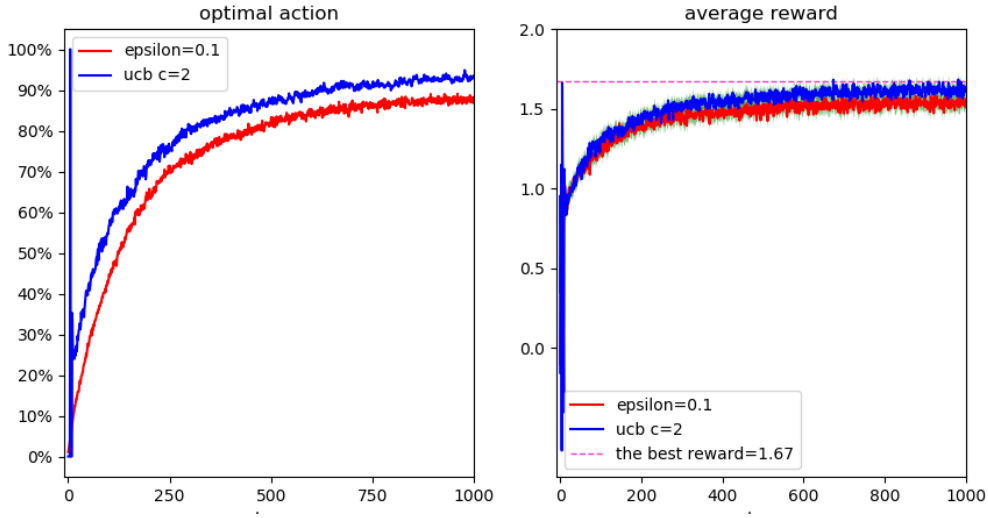


The UCB part:

The code *ucb.py* is used to generate the data. It would run 2000 trials with each trial has 1000 steps and store the data into 2000-by-1000 array.

The code *ucb_plot.py* is used to plot the data as follows:

(the 95% confidence bound is in shallow green in the average reward graph)



The interpretation of the spike:

- The first spike happened at step 6 with 100% optimal action selection. It is because the action that has not been experienced by the agent has larger value and forces the agent to explore, **np.argmax()** would always return the first max values in the list and q*(6) has the maximal values. After taking the action 6, the estimated value of this action would drop either due to the high initial value or the less uncertainty. In this case, both the optimistic initial value and UCB would force the agent to try new action in the beginning in the order of 1,2,3,4,5,6…..,10.
- The second step happened in step 11. The optimistic initial value method has 34.5% optimal action selection rate and UCB has 35.2%. (It's clear in the average reward graph)
- The reason is that the agent would explore actions that hasn't been tried and after one round of experiencing all the action one time, it would prefer the action with best reward and actions that were experienced with less times have big reward. After taking one action, the estimated value of this action would drop either due to the high initial value or the less uncertainty. This why the spike appears. (sharp increase and sharp decrease)

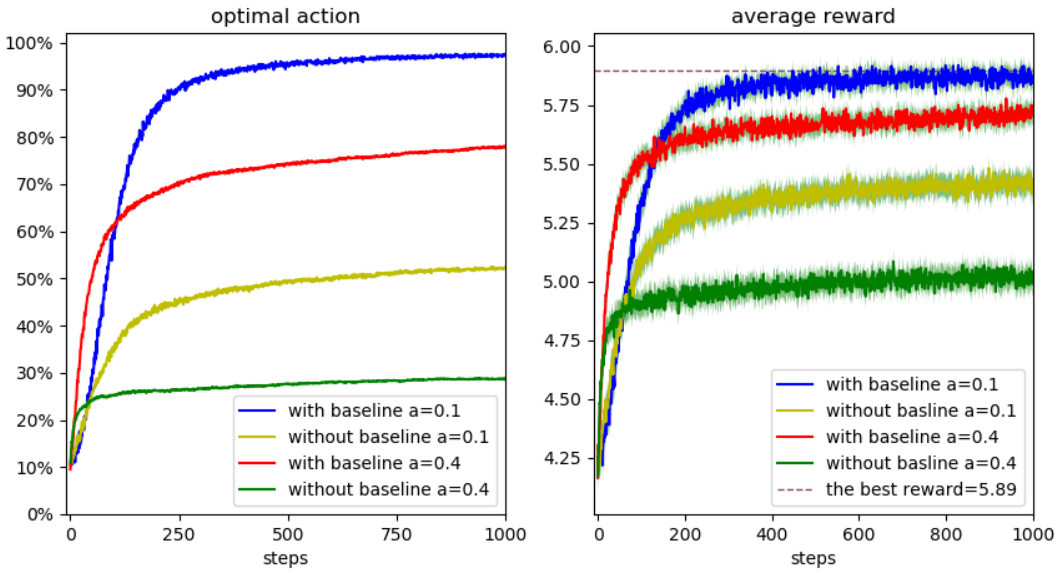
Answer for q9:

Experiment with gradient bandit algorithm (with baseline and without baseline):

The code *investigation.py* is used to generate the data for the bandit gradient algorithm. It would run 2000 trials with each trial has 1000 steps and store the data into 2000-by-1000 array.

The code *investigation_plot.py* is used to plot the data as follows:

(the 95% confidence bound is in shallow green in the average reward graph)



- Gradient methods (step size=0.1) perform better than that with step size of 0.4 in the testbed with the mean of 4
- Gradient bandit with baseline performs better than gradient bandit without baseline.