Question 1

Let $\phi : XxA \to \mathbb{R}^d$
The task is to compute $\nabla_\theta log(\pi_\theta(a|x))$ for the Boltzmann policy:

$$\pi_\theta(a|x) = \frac{e^{\theta^\top \phi(x,a)}}{\sum_b e^{\theta^\top \phi(x,b)}} \tag{1}$$

To answer this, we take logs and the derivative:

$$\nabla_\theta \log \pi_\theta(a|x) = \nabla_\theta \log \frac{e^{\theta^\top \phi(x,a)}}{\sum_b e^{\theta^\top \phi(x,b)}} \tag{2}$$

$$= \nabla_\theta \log e^{\theta^\top \phi(x,a)} - \nabla_\theta \log \sum_b e^{\theta^\top \phi(x,b)} \tag{3}$$

$$= \phi(x,a) - \frac{1}{\sum_b e^{\theta^\top \phi(x,b)}} \sum_{b'} e^{\theta^\top \phi(x,b')} \phi(x,b') \tag{4}$$

$$= \phi(x,a) - \sum_{b'} \frac{e^{\theta^\top \phi(x,b')}}{\sum_b e^{\theta^\top \phi(x,b)}} \phi(x,b') \tag{5}$$

Finally obtaining:

$$\nabla_\theta log(\pi_\theta(a|x)) = \phi(x,a) - \sum_b \pi_\theta(b|x)\phi(x,b) \tag{6}$$

Which can be rewritten as:

$$\nabla_\theta \log \pi_\theta(a|x) = \phi(x,a) - \mathbb{E}_{\pi_\theta}[\phi(x,\cdot)] \tag{7}$$

Question 2

Task: Run a policy gradient algorithm with a Boltzmann policy using $\phi_i(x,a) = 1_{(a=1)}$

The MDP we study has one state x (that is, $X = \{x\}$) and two actions $\mathcal{A} = \{a_1, a_2\}$. The rewards are defined as follows:

$$R_k = \begin{cases} 1, & \text{with probability } p_{I_k} \\ 0, & \text{otherwise} \end{cases}$$

The rewards are received with probability $p_1 = 1/2$ and $p_2 = 1/2 + \Delta$.
Given the environment set-up, we define a learning algorithm in order to find the optimal policy π_{θ_k}.
As instructed, we use a temporal difference actor critic algorithm to learn the optimal weight parameters $\theta_{k+1} = \theta_k + \alpha_k g_k$.
The arguments our algorithm takes are:

1. Enviroment: Takes delta as argument and react to actions paying 1 or 0 with probability $p_1 = 1/2$ and $p_2 = 1/2 + \Delta$.
2. Gamma: Discount factor for future rewards.

3. Policy: In this case we used Boltzmann policy as defined previously
4. Seed: We choose a random seed to make sure it is possible to reproduce the simulations.
5. Step type: Here we defined a fixed learning rate and also different rate decay in function of the number of steps.
6. Delta: The environment parameter that gives how much the reward from action 2 is better than action 1.
7. Weights θ: Weights of our policy
8. Beta: Learning rate for the value function

Our simulation function takes the number of steps and draws, at each training step, a pair of action and the according reward, updating the policy based on the policy gradient.

Question 2.1

Influence of γ on algorithm:

We find that γ does not play a role in this policy gradient. We get this result because we use the particularly simple estimation of the expectation in the policy gradient function suggested in the hint. When we set $h(x) = \gamma V^{\pi_{\theta_k}}(x)$, take a single sample to approximate the expectation, and take $\widehat{Q}^{\pi_{\theta_k}}(x, A_k) = R_k + \gamma V^{\pi_{\theta_k}}(x)$, the expression for the policy gradient simplifies as follows:

$$\begin{aligned}
\nabla_\theta \rho(\theta) &= \sum_{(x,a)\in\mathcal{X}\times\mathcal{A}} \mu_\theta(x)\pi_\theta(a|x)\nabla_\theta \log \pi_\theta(a|x)\left(Q^{\pi_\theta}(x,a) - h(x)\right) && (8)\\
&= \mathbb{E}_{X\sim\mu_\theta, A\sim\pi_\theta(\cdot|X)}\left[\nabla_\theta \log \pi_\theta(A|X)\left(Q^{\pi_\theta}(X,A) - h(X)\right)\right] && (9)\\
&\approx \nabla_\theta \log \pi_\theta(a|x)\left(\widehat{Q^{\pi_\theta}(x,a)} - h(x)\right) && (10)\\
&= \nabla_\theta \log \pi_\theta(a|x)\left(R_k + \gamma V^{\pi_{\theta_k}}(x) - \gamma V^{\pi_{\theta_k}}(x)\right) && (11)\\
&= \nabla_\theta \log \pi_\theta(a|x) R_k && (12)\\
&= \left(\phi(x,a) - \sum_b \pi_\theta(b|x)\phi(x,b)\right) R_k && (13)
\end{aligned}$$

The discount factor γ cancels out. Further, since the state in this MDP does not change, we do not have to take into account the consequences of our actions now for our future as we adjust our policy, therefore the choice of γ should intuitively not play a role in our policy gradient.

Question 2.2

Influence of step

Question 2.3

Best step size and policy as function of Δ

Question 3

Performance of some basic algorithms on the same bandit problem

Question 3.1

First we define the standard UCB algorithm for our bandit problem. As described in the question, at each iteration of the algorithm, we start by choosing an "arm" or action in this case by finding

$$I_t = \arg\max_{i \in \{1,2\}} \left(\widehat{\mu}_{t,i} + \sqrt{\frac{2 \ln t}{N_{t,i}}} \right)$$

we initialize the algorithm by forcing two trials of each action. After taking an action we store the resulting reward and update our mean parameter $\mu t, i$.
The following plot shows the total reward for $T = 10,000$ rounds as a function of Δ of the environment:

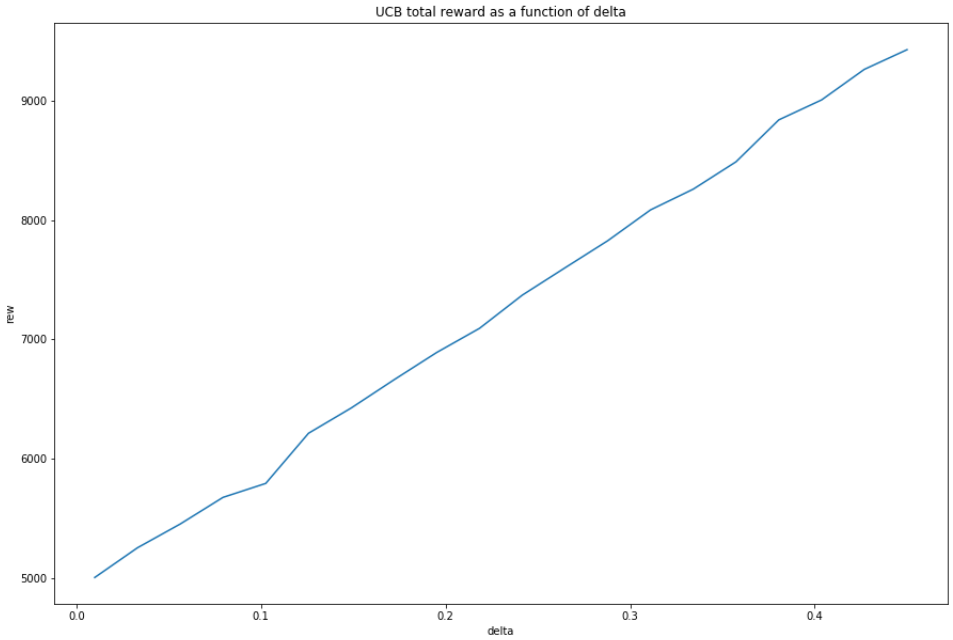

Figure 1

One can easily see that larger values of Δ lead to greater total rewards. The intuition behind this is that large Δ lead to a clearer seperation between the expected rewards of each action. For very small Δ, the expected rewards of each action are almost $\frac{1}{2}$. Unsurprisingly, the total reward for a Δ of 0 is 5000. Conversely, for a Δ of 0.5 the total reward is almost 10,000, but not quite. This is due to the fact that we still have to run a tiny amount of trials to get convergence of the estimated mean to the true value. After that we always pick the right action, which almost certainly results in a reward.

Question 3.2

The ϵ-greedy algorithm works in the following way:
For a given probability ϵ, either choose a uniformly random action, or choose the action which has the maximum expected reward based on the previously recorded rewards. The value of ϵ drops in each step, resulting in a smaller and smaller chance of taking random actions. Consequently, the drop in ϵ can be seen as a switch from exploring the environment to exploiting it based on the learned features. ϵ in this case is set as $\epsilon = c/t$ where c is a constant.
The following plot shows the total reward for a range of values of c and Δ.

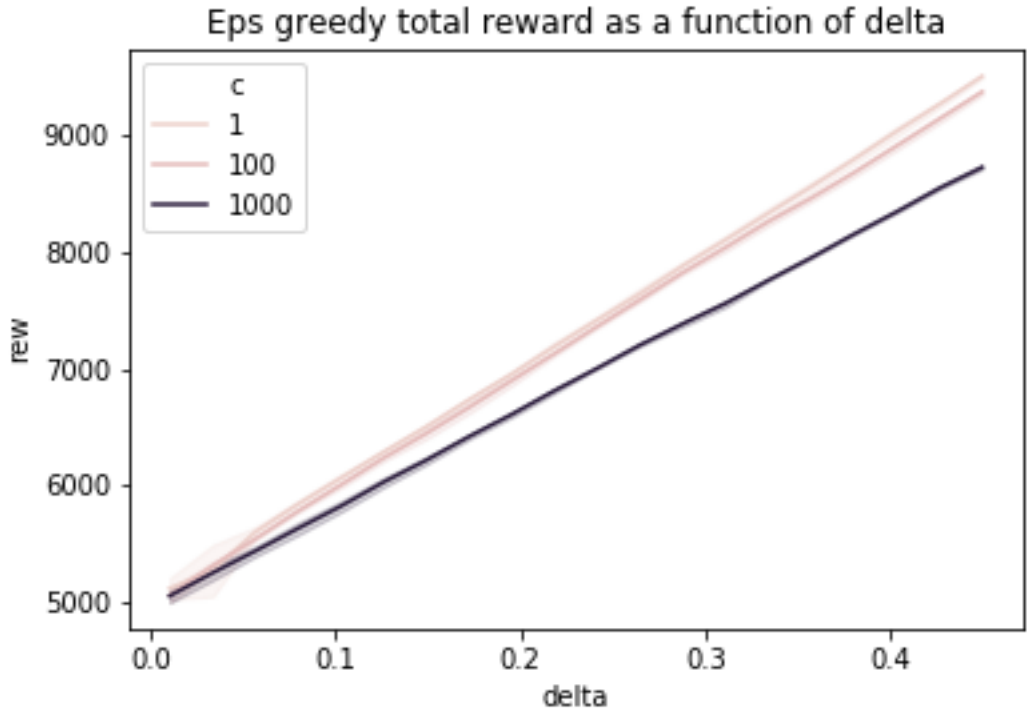

Figure 2

First we note that similarly to the UCB algorithm, larger values of Δ lead to higher total rewards. Additionally, for small c and Δ all total rewards are similar. For greater Δ, with smaller parameter c, the algorithm converges quicker to the optimal policy. This can be seen in the higher overall rewards. Again, the intuition is that for high Δ we don't need to explore the environment as in depth as before, so smaller c will lead to a smaller amount of trials in which we randomly explore the environment.

Question 3.3

The last plot shows the combined results of all three algorithms for our small bandit problem.

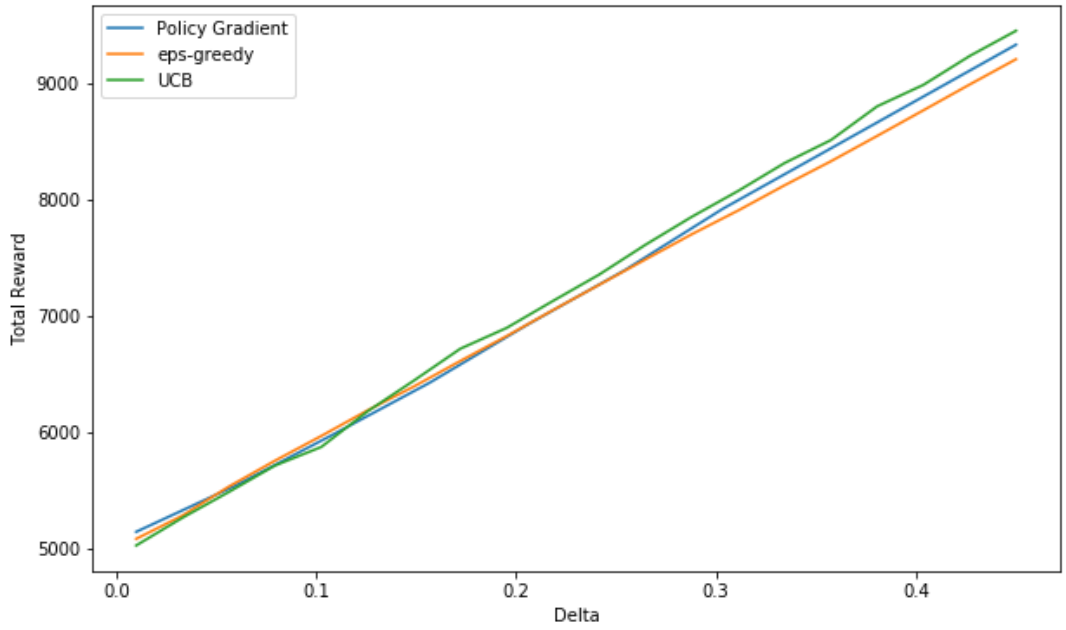

Figure 3