Question A

In chapter 4, we found that Eval_Trunc could be used in place of Eval. Briefly and clearly explain the following:

a. Why is this important for Dynamic Programming?

   Ans: Since Dynamic Programming includes policy evaluation in each iteration, but only the first couple iterations have large impact, therefore it's more efficient to truncate policy evaluation whenever it converges well enough.

b. Why is this important for Reinforcement Learning? Explain the significance of this result.

   This is important for RL because the idea can be applied to other algorithms, it can significantly save computation cost and its result won't lose convergence guarantees of policy iteration.

Question B:

(10 points) Clearly explain the importance of the Bellman equation in the context of formulating the Dynamic Programming algorithm for prediction? Use equations where needed for clarity.

The bellman equation is important because it allows us to formulate the value of a state and thus we can use dynamic programming to update the evaluation and find policy based on the state value. According to Bellman Equation

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_*(s') \Big], \end{aligned}$$

we can see we can formulate the value of a state by the sum of next states' values and reward multiply by the possibility of reaching next state, or we can also use Bellman's state action equation

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a \Big] \\ &= \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma \max_{a'} q_*(s', a') \Big], \end{aligned}$$

to evaluate the value of taking a specific action at certain state.

Question C:

Clearly answer the below.

a. (10 points) Explain how exploration concretely manifests itself in Reinforcement Learning as presented for Dynamic Programming and Monte Carlo methods.

   In Dynamic Programming since the environment is fully known the exploration is simply iterate over all states and evaluate them and update policy accordingly. But in Monte Carlo method we do not have complete knowledge over environment and thus we need a proactive exploration step, that is, e-greedy or e-soft, to take an exploration step and explore actions we have not tested yet, then we can improve our policy so that we can find optimal action.

b. (5 points) Why is exploration needed?

   Because if we always follow the policy we have we will not be able to try new actions, and some of the actions may lead to better result than current action and has potential to be the optimal action. Without the exploration we will only be updating existing states' or actions' values repeatedly.

Question D:

(10 points) In Figure 4.1 of the text, observe the k=3 value function. Clearly show the calculations that produced row 1 of this value function.

Denote row 1 at k = 3 as s0, s1, s2, s3

S0 = 0 because it's the final state

S1 = ¼ * (0 + -1) + ¼ * (-1.7 + -1) + ¼ * (-2 + -1) + ¼ * (-2 + -1)

= -1/4 + ¼ * -2.7 + ¼ * -3 + ¼ * -3

= - 2.425 = -2.4

S2 = ¼ * (-1.7 + -1) + ¼ * (-2 + -1) + ¼ * (-2 + -1) + ¼ * (-2 + -1)

= -2.925 = -2.9

S3 = ¼ * (-1.7 + -1) + ¼ * (-2 + -1) + ¼ * (-2 + -1) + ¼ * (-2 + -1)

= -2.925 = -2.9

Question E:

I have a policy with 10 possible actions {a0, a1, ..., a9 } in state s.

q(a0|s)=q(a1|s) > q(ai | s) forall ai \in {a2, ... a9}

a. (5 points) Show the optimization action selection policy for s.

   Pi(s) = argmax_{ai}(q(ai|s)) forall ai in {a0, … a9} = a0 or a1

b. (5 points) Show a fair epsilon-greedy action selection policy for s.

   Let e = 0.5

   P(q(ai | s)) = e/10 forall ai \in {a2, ... a9} = 5%

   P(q(a0|s)) = P(q(a2|s)) = ½ * (1 – e + e/10 + e/10) = 30%

c. (5 points) Show a epsilon-soft action selection policy for s that is distinct from b

   P(q(a0|s)) = P(q(a2|s)) = … = P(q(a9|s)) = 10%

d. (5 points) What differentiates e-soft and e-greedy action selection policies?

   e-greedy action selection policy is an e-soft policy, but a soft max is not required, and e-greedy is also the closest to greedy policy in e-soft policies.

Question F:

In MC, the off-policy methods generate an episode with b which is distinct from pi.

a. (5 points) How are we able to estimate Q_pi using b as the episode-generating policy?

   We need to make sure b is an e-soft policy, that is all actions is possible to be explored at least occasionally, then we can say that every action taken under pi is also taken at least occasionally under b, which is the assumption of converge.

   We can calculate the importance-sampling ratio,

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k,A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

   then we can estimate Q_pi by the ratios and average the result with number of times visited.

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}.$$

   or we can use weighted average

$$V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}},$$

b. (5 points) In computing the returns, why does the iteration process episodes in reverse-time order?

Ans: Since with importance sampling we need next states information, in p110's function we will need next state's weight, value and cumulate sum to compute the return. Forward-time order will require recurrent function which is inefficient.

c. (10 points) Suppose we wrote the iteration in forward-time order. Show the recurrent (sample update) expression for the computation of the return. That is, how do the first two lines of the inner loop on p 110 change.

Loop For Each $t=0, \cdots, T-1$, while $w \neq 0$

$G = \sum_{t'} r^{t'} R_{t'+1}$ for $t' = t+1, \cdots, T-1$

$C(S_t, a_t) += \pi \frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t'}|S_{t'})}$ for $t' = t+1, \cdots, T-1$

d. (5 points) Suppose we wrote the iteration in forward-time order. Show how the inner loop of p 111 would have to be modified to process the episodes.

Loop For each $t=0, \cdots, T-1$ which $w \neq 0$

$G' = \sum r^{t'} R_{t'+1}$ for $t' = t+1, \cdots, T-1$

$w' = \pi \frac{\pi(A_{t'}|S_{t'})}{b(A_{t'}|S_{t'})}$ for $t' = t+1 \cdots, T-1$

$G = \gamma G' + R_{t+1}$

$C(S_t, A_t) = C(S_t, A_t) + w'$

$Q(S_t, A_t) = Q(S_t, A_t) + \frac{w}{C(S_t, A_t)} [G - Q(S_t, A_t)]$

$\pi(S_t) \leftarrow \text{argmax}_a (Q_{S_t}, a)$

If $A_t \neq \pi(S_t)$ exit inner loop

Question G:

Suppose we are using on-policy MC on a simulated environment.

a. (5 points) Explain the two approaches for GPI with on-policy MC.

   On-policy MC uses use the same policy to generate episode and update policy, whereas off-policy MC uses a behavior policy to generate episodes and aim to estimate the target policy.

b. (5 points) Which of the two are most appropriate for our problem and why? Why would you choose this over the other?

   In my case I would choose off policy, since I think off policy don’t have to consider exploration step and thus it’s more likely to explore all the states and return a better result

c. (10 points) Suppose you used the less appropriate version, while your colleague used the more appropriate one. You both have waited a very long time for your algorithms to converge. After converging, you find that your policy is performing worse than your colleagues. Explain the reasons why this has likely occurred, and how would you adjust the algorithm you have chosen (note: you can't use your friend's algorithm) to improve performance.

   If I’m using on-policy approach and it performs worse then it’s probably because the e-greedy method is not good enough, which means the approaches rarely explores and thus it did not find the optimal policy. I will adjust epsilon value so that the approach explores more often.