Assignment 7 by Haobin Tang

1.Prove the Epsilon-Greedy Policy Improvement Theorem (we sketched the proof in Class)

Prove Monotonic ε-greedy Policy Improvement. Let π_i be an ϵ-greedy policy. Then, the ϵ-greedy policy with respect to Q^{π_i}, denoted π_{i+1}, is a monotonic improvement on policy π. In other words, $V^{\pi_{i+1}} \geq V^{\pi_i}$.

$$\begin{aligned}
V^{\pi_{i+1}}(s) &= Q^{\pi_i}(s, \pi_{i+1}(s)) \\
&= \sum_{a \in A} \pi_{i+1}(a|s) Q^{\pi_i}(s, a) \\
&= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s, a) + (1-\epsilon) max_{a'} Q^{\pi_i}(s, a') \\
&= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s, a) + (1-\epsilon) max_{a'} Q^{\pi_i}(s, a') \frac{1-\epsilon}{1-\epsilon} \\
&= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s, a) + (1-\epsilon) max_{a'} Q^{\pi_i}(s, a') \sum_{a \in A} \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1-\epsilon} \\
&= \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s, a) + (1-\epsilon) \sum_{a \in A} \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1-\epsilon} max_{a'} Q^{\pi_i}(s, a') \\
&\geq \frac{\epsilon}{|A|} \sum_{a \in A} Q^{\pi_i}(s, a) + (1-\epsilon) \sum_{a \in A} \frac{\pi_i(a|s) - \frac{\epsilon}{|A|}}{1-\epsilon} Q^{\pi_i}(s, a) \\
&= \sum_{a \in A} \pi_i(a|s) Q^{\pi_i}(s, a) \\
&= V^{\pi_i}(s)
\end{aligned}$$

2. Provide (with clear mathematical notation) the defintion of GLIE (Greedy in the Limit with Infinite Exploration)

1.A policy π is greedy in the limit of in nite exploration (GLIE) if it satis es the following two properties:

$$lim_{i \to \infty} N_i(s, a) \to \infty$$

where $N_i(s, a)$ is the number of times action a is taken at state s up to and including episode i.

2.The behavior policy converges to the policy that is greedy with respect to the learned Q-function. i.e. for all $s \in S, a \in A$,

$$lim_{i \to \infty} \pi_i(s, a) = argmax_a q(s, a)$$

with probability 1

3.Implement the tabular Q-Learning algorithm

In [ ]:

```python
# I implement Q-learning using the project example.

class Q_learninig():
    def __init__(self):
        self.iteration=2000000

        self.t_size=6
        self.Price_size=100
        self.N_size=50
        self.start_price=80
        self.gama=1
        self.stepsize=0.1
        self.epsilon=0.5


    def env(self):
        Q={}
        for i in range(self.t_size):
            Q[i]={}
            for j in range(self.Price_size):
                Q[i][j]={}
                for k in range(self.N_size):
                    Q[i][j][k]={}
                    for l in range(self.N_size):
                        Q[i][j][k][l]=0

        action={}
        for i in range(self.t_size):
            action[i]={}
            for j in range(self.Price_size):
                action[i][j]={}
                for k in range(self.N_size):
                    action[i][j][k]=random.randint(0,k)


        return Q, action


    def simulator(self,Rt,Pt,Nt):
        alpha=1
        #error=np.random.randn()
        #error=int(random.uniform(-2, 2))
        #error=0
        error=np.random.binomial(10, 0.5, size=None)-5
        Pt=int(Pt-alpha*Nt-error)
        if Pt<0:
            Pt=0
        Rt=Rt-Nt
        reward=Nt*(Pt)

        return Rt,Pt, reward



    def Q_learning_or_Sarsa(self, method="q_learning"):
        #initialize
        Q, action = self.env()

        iteration=[]
        Q_plot=[]
        for i in range(self.iteration):
             # Using Sigmoid function as epsilon decrease
            epsilon=1-(1/(1+math.exp(5-10*i/self.iteration)))

            #Set s0 as the starting state
            t=0
            Pt=self.start_price
            Rt=self.N_size-1

            best_action=max(Q[t][Pt][Rt], key=Q[t][Pt][Rt].get)
            if random.uniform(0, 1)<epsilon:
                action[t][Pt][Rt]= random.randint(0,Rt)
            else:
                action[t][Pt][Rt]= best_action

            while t<self.t_size-1 and Rt>0:  # loop until episode terminates
                Nt=action[t][Pt][Rt] # Sample action at from policy π(st)

                #print (t,Pt,Rt,Nt)
                #Take action at and observe reward rt and next state st+1
                Rt_next,Pt_next, reward= self.simulator(Rt,Pt,Nt)
```

```
                t_next=t+1
                # update Q
                    # find next state Qmax

                best_action=max(Q[t_next][Pt_next][Rt_next], key=Q[t_next][Pt_next][Rt_next].get)

                #print (t_next,Pt_next,Rt_next,Nt,best_action)
                if method=="q_learning":
                    Qmax=Q[t_next][Pt_next][Rt_next][best_action]
                    Qnow= Q[t][Pt][Rt][Nt]
                    Q[t][Pt][Rt][Nt]=Qnow+self.stepsize*(reward+self.gama*Qmax-Qnow)

                if method=="sarsa":
                    Nt_next=action[t_next][Pt_next][Rt_next]
                    Qnext= Q[t_next][Pt_next][Rt_next][Nt_next]
                    Q[t][Pt][Rt][Nt]=Qnow+self.stepsize*(reward+self.gama*Qnext-Qnow)

                #update S
                t=t_next
                Pt=Pt_next
                Rt=Rt_next

                if random.uniform(0, 1)<epsilon:
                    action[t][Pt][Rt]= random.randint(0,Rt)
                else:
                    action[t][Pt][Rt]= best_action

            #for plotting
            if i%(self.iteration/10)==0:
                first_day_sell=max(Q[0][self.start_price][self.N_size-1], key=Q[0][self.start_price][self.N_size-1].get)
                print (i//(self.iteration/10),"/10", "first_day_sell=",first_day_sell, "Q_value=", Q[0][self.start_price][self.N_size-1][first_day_sell])

                shares_to_sell_everyday=[first_day_sell]
                sell=first_day_sell
                remain=self.N_size-1-sell
                price=self.start_price-sell
                for j in range(1,self.t_size):
                    Q_value=0
                    sell=max(Q[j][price][remain], key=Q[j][price][remain].get)
                    shares_to_sell_everyday.append(sell)
                    remain=remain-sell
                    price=price-sell
                    #print(remain)
                    '''
                    for k in range(self.Price_size):
                        temp=max(Q[j][k][remain], key=Q[j][k][remain].get)
                        if Q[j][k][remain][temp]>Q_value:
                            sell=temp
                            Q_value=Q[j][k][remain][temp]
                    if sell>remain:
                        sell=0
                    shares_to_sell_everyday.append(sell)
                    #print(sell)
                    remain=remain-sell
                '''
                print(shares_to_sell_everyday)

            if i%(self.iteration/1000)==0:
                iteration.append(i)
                first_day_sell=max(Q[0][self.start_price][self.N_size-1], key=Q[0][self.start_price][self.N_size-1].get)
                Q_plot.append(Q[0][self.start_price][self.N_size-1][first_day_sell])


        plt.plot(iteration,Q_plot)
        #print(iteration)
        return Q, action
```