IEOR 222 Final Project - Q Learning for Optimal Trade Execution

Santhosh Subramanian

***Abstract*— Optimal Order Execution has been a problem in many decades in financial institutions. In class, we learned about couple of ways to tackle this issue. Mainly, we focused on the model provided by Bertsimas and Lo. The goal of the project is to use Q Learning to get the optimal order execution strategy under various price impact models.**

I. INTRODUCTION

Optimal Order Execution has been a problem for nearly decades. Say we need to buy or sell a certain amount of shares by a certain time period. If we buy too many shares at once, the stock might rise, and we are negatively impacted by the increasing buying price. If we buy too little shares, then we have to liquidate majority of our shares towards the end of the interval, or we won't finish the job The goal of the project is to use reinforcement learning, specifically Q learning, in order to learn the optimal execution strategy. After learning about Bertsimas & Lo, we know that given the linear permanent price impact (1), the optimal execution strategy is to buy equal amount of shares in each time interval from the first to the last interval.

$$(1)\ P_{t+1} = P_t + \epsilon - S_t\theta$$

In the linear permanent price impact model, P_t is the price of the security at time t, ϵ is a zero-mean independently and identically distributed (IID) white noise, S_t is the number of shares sold at time t, and θ is an impact premium.
The second model I used was the quadratic temporary model which has a formula of:

$$(2)\ P_{t+1} = P_t + \epsilon - (a(S_t)^2 + bS_t)$$

In the quadratic temporary impact model, P_t is the price of the security at time t, ϵ is zero-mean noise, and $(a(S_t)^2+bS_t)$ is the impact premium in quadratic form. The last model I used was the linear transient model. Which I will talk more about in depth in the later parts of the paper.

II. PROCEDURE

A. Defining Helper Functions

The first thing I did was develop helper functions. These functions were used for synthetic price generation, generating actions from a certain state, and reward functions for each of the three models I chose to work with.

B. Reward Functions

The reward function was straight forward. In order to generate the rewards for taking an action from a certain space state, I would generate the new price from taking that action and multiply that by the number of shares that the trader is going to by the state.

$$R_t = S_t \times P_{t+1}$$

Here R_t is the reward generated at time t, S_t is the shares that are bought at time t, and P_{t+1} is the price that would be generated by the impact of those shares sold. I also wrote helper functions that implemented each of the three impact models talked about in the equation above.

III. ALGORITHM

Algorithm 1: Q Learning Model

```
Initialize Q Table
 begin
   $\alpha = 0.1$;
   $\epsilon = 1$;
   for $i$ in episodes do
     $P_0 = 100$;
     $S_t$ = total shares in inventory;
     for $t$ in time interval do
       current state = $(t, P_t, S_t)$;
       if current state not in Q Table then
         Add current state to Q Table
       end if
       if $t$ is not the last time interval then
         if random value $< \epsilon$ then
           shares buying = action $\leq S_t$;
         end if
         else
           shares buying = max(current state
            in Q Table);
         end if
       end if
       else
         shares buying = $S_t$
       end if
       $R_t$ = Reward from buying $S_t$;
       $P_{t+1}$ = Price change due to $S_t$;
       $F_Q = max$(Q val of future state generated
        by current action);
       $Q_{val} = (1-\alpha)$(Current State Q Value) -
        $\alpha(R_t + F_Q)$;
       Update Q Table;
       Update Total Inventory
     end for
   end for
 end
```

A. *Algorithm Notes*

Above I wrote pseudocode of how my algorithm works for the three models. Even though the actual algorithm isn't the same for each model as it shouldn't be due to different price impacts and if the model is temporary or permanent, this is the skeleton of the code. For different models, I would change some code and add some code. For instance, for the linear transient, I added an array that tracked all the shares bought in every time period, and an array that contained all the noises. The algorithm was also an epsilon greedy approach where epsilon would decrease as the iterations increased so that the algorithm would move from exploration to exploitation. My ϵ started at 1 and after every episode, I would reduce epsilon by 0.00001. I did this so to help my algorithm converge faster. Secondly, my γ when generating q values was 1. Lastly, my learning rate, α was the industry standard at 0.1.

B. *Algorithm Explanation*

- **Initial Steps:** I initialized a Q table which was blank data frame that I would add actions and state spaces as the algorithm runs. I also initialized the values for ϵ, α, and γ.
- **Generating Actions:** First we would iterate through the time intervals. At each time interval we first recorded the current state. If it was in the Q table, we would move on. Otherwise, we would add the current state into the Q table and initialize all the actions for the current state to have a Q value of 0. Next, we would generate a random float between 0 and 1, if it was lower than epsilon, we would pick a random action that is valid (the action must be lower than the total shares inventory). For instance, if the action says to sell 1000 shares, but we only have 500 shares to sell, this action would be invalid, and a new action would be selected. If the random float between 0 and 1 is greater than epsilon, we would pick the maximum Q value for the current state that is a valid action. If we are in the last time interval, the amount of shares we sell is the amount of shares left in the inventory.
- **Generating Rewards and Q Values:** Now, we are generating rewards. Based on the current price and the number of shares we are buying, we generate rewards by using the equation in the procedure section. Then after getting the rewards for taking that particular action, we can find the updated Q value for that current state. First, we log the present Q value for the current state. We then we need to find the max Q value for the future state that is generated by the action we are taking in the current state. To do this, we generate the new state space and then find the highest Q value in the q table for that new state space. We then can find the updated Q value by using the equation provided in the pseudo-algorithm.
- **Updating and Final Steps:** After getting the updated Q value, we then updated our Q table and total inventory. Once this step is finished, we then move on to the next time step. Once, we finish all the steps, we move on to the next episode/iteration.

C. *Linear Transient Model*

In this section, we will go more in depth about the linear transient model that I built. At first I was using the equation that was given in office hours, this equation was:

$$P_t = P_0 + \sum_i^t \epsilon_i + \sum_i^t [e^{-\alpha(t-i)} \theta S_i]$$

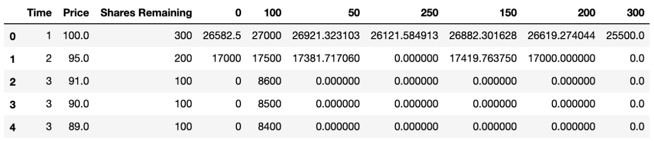
In the equation above, the P_0 is the initial price that I generated, the second part is the sum of all the noises till time t that have been generated, and then the last part is a sum of all the shares till time $t$multiplied by a decay kernel. In this case $\rightarrow e^{\alpha(t-i)}$. For the equation above, I used an initial price of 100, an α of 0.25, and a θ of 0.05.

I also used another transient price impact model with another decay kernel. Instead of $e^{-\alpha(t-i)}$ as my decay kernel, I used $\frac{1}{1+(t-i)^2}$. This was found in Gatheral's paper: *Dynamical models of market impact and algorithms for order execution.*

IV. Results

A. *Linear Permanent Model*

For the linear permanent model, we see an optimal strategy of buying evenly down the middle. I tested this by selling 600 shares in 6 intervals. The θ I chose was 0.05. My noise was generated by a random choice function in numpy that had 0 mean. The highest reward for time 1 was selling 100 shares, time 2 selling 100 shares, time 3 selling 100 shares, and so on and so forth. The optimal strategy is $\frac{\overline{S}}{T}$ where $\overline{S}$ is the total number of shares to be liquidated, and T is in how many intervals you need to do so. This matches Bertsimas & Lo's strategy outlined in their paper. Further for this model, we had 10,000 episode.



	Time	Price	Shares Remaining	0	100	50	250	150	200	300
0	1	100.0	300	26582.5	27000	26921.323103	26121.584913	26882.301628	26619.274044	25500.0
1	2	95.0	200	17000	17500	17381.717060	0.000000	17419.763750	17000.000000	0.0
2	3	91.0	100	0	8600	0.000000	0.000000	0.000000	0.000000	0.0
3	3	90.0	100	0	8500	0.000000	0.000000	0.000000	0.000000	0.0
4	3	89.0	100	0	8400	0.000000	0.000000	0.000000	0.000000	0.0

Fig. 1. Filtered State Space

If you take a look at the state space here, for every state space, we see that the action with the highest reward is 100. This model also worked at $n = 6$ and $S = 600$.

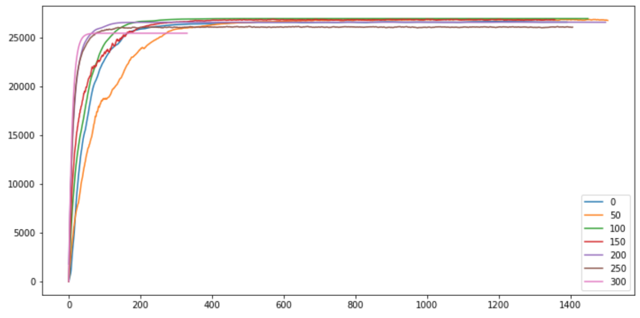


Fig. 2. Convergence of Expected Rewards

Though it is hard to see this graph shows the convergence of the expected reward for state 1. We see that as the iterations go by that at state 1 with 1000 shares remaining, the state converges to taking action where we buy 100 shares. This is shown as the action 100 at state 1 has the highest expected reward.

B. *Quadratic Temporary Model*

For the quadratic temporary model, we also see an optimal strategy of buying evenly down the middle. The optimal strategy is $\frac{\overline{S}}{T}$ where $\overline{S}$ is the total number of shares to be liquidated, and T is in how many intervals you need to do so.

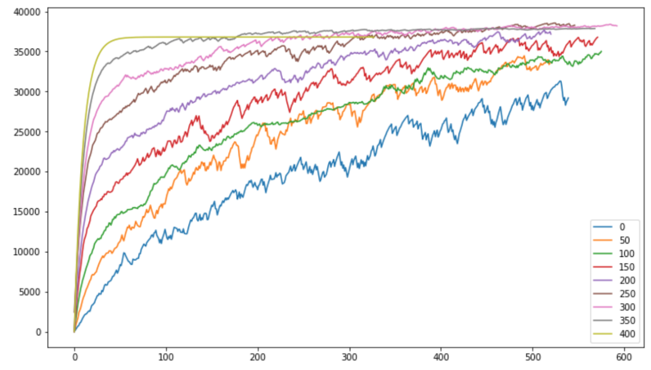
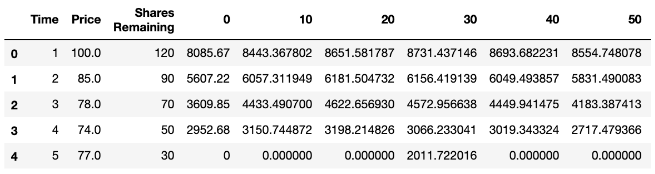


Fig. 3. Q Learning Graph in the process

Here is a graph different from the one above. This graph (from the quadratic temporary) I chose as we can see how the network learns to pick up rewards over time. We see that as the episodes increase that the Q learning network keeps finding the path the higher and higher reward. This graph has not converged yet, but as the network runs more episodes the graph will start to look like the one above.

C. *Linear Transient Models*

For the linear transient model, I chose θ to be 0.05 and α as 0.25. In my algorithm, I chose to sell 120 shares in 5 time intervals. The different episodes I used were 10000, 20000, 30000, 50000, and 75000. The first four number of episodes I chose did not give me the right optimal strategy; however, after increasing the number of episodes. I finally reached the optimal strategy of 30, 20, 20, 20, 30. This is equivalent to Gatheral's conclusions in his paper. The optimal trading strategy here is a convex strategy where we buy a block at time 0 and time t, and then buy smaller equivalent shares in the middle.



	Time	Price	Shares Remaining	0	10	20	30	40	50
0	1	100.0	120	8085.67	8443.367802	8651.581787	8731.437146	8693.682231	8554.748078
1	2	85.0	90	5607.22	6057.311949	6181.504732	6156.419139	6049.493857	5831.490083
2	3	78.0	70	3609.85	4433.490700	4622.656930	4572.956638	4449.941475	4183.387413
3	4	74.0	50	2952.68	3150.744872	3198.214826	3066.233041	3019.343324	2717.479366
4	5	77.0	30	0	0.000000	0.000000	2011.722016	0.000000	0.000000

Fig. 4. Condensed State Space

Here we see the most optimal path is selling 30, selling 20, selling 20 again, selling 20, and finally liquidating the last 30 shares. We can see that as for every state, the expected rewards for these actions is the highest.

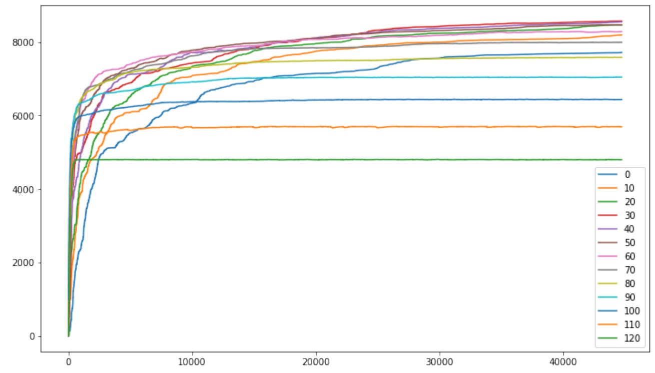


Fig. 5. State 1 Convergence

Here we see the convergence graph of the first state. It takes around 40,0000 steps to finally converge and give me the right optimal strategy. We see that at state 1, it converges to action 3 with an expected reward of just above 8000. Before, we were getting suboptimal strategies as the states were not converging by the steps given (10000, 20000, 30000, 50000). This optimal strategy found in this solution is almost exactly what Gatheral's solution was.

V. CONCLUSIONS

For this project, we see how the optimal trading strategy changes depending on the given model. The linear permanent model's optimal strategy is to buy the same amount of shares until the end of the liquidation period. This strategy produced by my reinforcement algorithm makes complete sense as we end up with the same strategy that Bertsimas and Lo come up with. The quadratic model's optimal strategy was to buy even amount of shares. Though I don't know if this is correct, I believe this makes complete sense. The linear price impact model is $P_{t+1} = P_t + \epsilon - S_t\theta$ and the quadratic impact model is $P_{t+1} = P_t + \epsilon - (a(S_t)^2 + bS_t)$. I believe the quadratic impact model could have an optimal strategy that is equivalent to the linear price impact model as there is an α, β, and θ where $a(S_t)^2 + bS_t = \theta S_t$. Lastly, the transient model gave us nice results. Using the approach Gatheral took and the equation Nan gave us, we were able to produce the results that Gatheral produced. We were able to produce the optimal strategy which was convex in nature.