Machine Learning for Finance
Multi-Agent Deep Reinforcement Learning for Liquidation Strategy Analysis

Students :
Benoît Ane
Antoine Lefebvre
Johan Macq
Jérémy Marck

Supervisor :
Romuald Elie

May 17, 2020

Contents

1 Introduction

1.1 Context

Statistical learning tools are more and more present in finance due to a strong growth in computing power and data. In particular, machine learning and reinforcement learning techniques are beginning to show promising results in areas such as derivatives pricing and optimal execution. The use of such algorithms is starting to spread and the understanding of these techniques is becoming a major issue. Through this project, we wish to propose a simple review of reinforcement learning principles and its concrete application in the case of financial asset liquidation strategy. Such strategies have been studied for many years. They represent a major stake for financial institutions because of liquidation cost.

1.2 Considered article

Our project is based on Bao and Liu (2019). This article proposes a reinforcement learning approach to develop a liquidation strategy with several agents. Liquidation is the process of selling a large number of shares of a security sequentially within a given timeframe, taking into account the costs arising from market impact and agents' risk aversions. The main challenge of optimization is to find a suitable modeling system integrating the complexities of the stock market and generating strategic executions. In this paper, the authors use a multi-agent reinforcement learning model, which allows a better understanding of the complexities of the trading environment and allows agents to learn how to make better sales decisions. First, the authors theoretically analyze the Almgren and Chriss model and propose an extension of the basic model so that it can be used as the business environment for the multi-agent model. Second, they analyze cooperative and competitive behavior between agents by adjusting the reward functions. Finally, they derive an optimal liquidation strategy with practical constraints, thus showing the capacity of reinforcement learning when applied to liquidation problems.

1.3 Objectives

Our objective here is multiple. Indeed, we wish to propose a fine understanding of the algorithms of reinforcement learning applied to the financial case of liquidation. Moreover, we wish to extend the initial problem by varying exogenous and endogenous parameters as well as defining new reward functions.

2 Optimal liquidation in an Algrem & Chriss -based multi-agent environment

In this section, we present the single-agent liquidation model of Almgren and Chriss (2001). We then explain how this model is adapted by Bao and Liu (2019) to the case of a multi-agent environment in which optimal liquidation strategy is obtained by reinforcement learning.

2.1 Almgren & Chriss model

2.1.1 Optimal liquidation problem

The basic question we want to answer is, given a quantity $q_0 > 0$ of a stock, what is the optimal way to liquidate it over a time horizon T ? We will consider a discrete time setup. At each date, the objective is to liquidate the optimal quantity dealing with the following trade-off:

1. Liquidating too fast is costly because of market impact and execution costs.
2. Liquidating too slowly exposes to price changes in the market, it may go down while we are liquidating and it would have been better executing faster.

Hence, an optimal strategy is defined as being a good compromise between these two points.

2.1.2 Solving the problem

The optimal liquidation problem has been largely studied and the framework proposed by Almgren and Chriss (2001) has been chosen as the market model. Below a reminder of the optimal solution of this model. We will use the following discretization grid, consisting of discretizing the time horizon T in N steps associated with the time step $\tau = \frac{T}{N}$. Hence the discretization grid: $t_0 = 0 < ... < t_k = k\tau < ...t_N = N\tau = T$. We wish to sell a quantity X of stock and we note x_k the quantity remaining at time t_k. Thus, we have $x_0 = X$ and $x_T = 0$.

We will consider the evolution of the shares x_k and of the stock price S_k, for $k \in \{0, ..., N\}$. Regarding shares, the evolution writes down:

$$x_k = x_{k-1} - v_k\tau$$

Where $v_k = \frac{n_k}{\tau}$ is the trading rate, consisting in selling n_k shares during the time interval $[t_{k-1}, t_k] = \tau$.

Regarding the evolution of the stock price, it is characterized by:

$$S_k = S_{k-1} + \sigma\sqrt{\tau}\underbrace{\xi_k}_{\mathcal{N}(0,1)} - \tau g(v_k)$$

with $g(v_k)$ being the permanent impact of selling on the stock price, here chosen linear: $g(v_k) = \gamma v_k$, and ξ_k are iid.

We introduce now the notion of temporary impact. When selling stocks, we are liquidity taker because we "eat" the order book. We assume that this consumed liquidity comes back after a certain time and that this

effect is temporary. When we sell a quantity n_k, we get an average price of $\hat{S_k} = S_{k-1} - h(v_k)$. Here we have a temporary impact also linear: $h(v_k) = \epsilon + \eta v_k$, where ϵ represents the bid-ask spread.

The cash made out of the selling process is given by:

$$\sum_{k=1}^{N} n_k \hat{S_k} = XS_0 + \sum_{k=1}^{N} (\sigma\sqrt{\tau}\xi_k - \tau g(v_k))x_k - \sum_{k=1}^{N} n_k h(v_k)$$

We define the trading cost C as the cost induced by the selling action: $C = XS_0 - \sum_{k=1}^{N} n_k \hat{S_k}$

Objective: minimizing the trading cost C. In the general Almgren Chriss model, it consists of finding the liquidation trajectory x minimizing $U(C) = \mathbb{E}[C] + \lambda V[C]$, where λ can be seen as the risk aversion of the trader.

Given the impact functions g and h, we can rewrite the trading cost as follow:

$$C = \sum_{k=1}^{N} (\tau g(v_k) - \sigma\sqrt{\tau}\xi_k)x_k + \sum_{k=1}^{N} n_k h(v_k) = \sum_{k=1}^{N} (\gamma n_k - \sigma\sqrt{\tau}\xi_k)x_k + \sum_{k=1}^{N} n_k(\epsilon + \eta\frac{n_k}{\tau})$$

Hence the following moments:

$$\mathbb{E}(C) = \frac{1}{2}\gamma X^2 + \epsilon X + \frac{\hat{\eta}}{\tau}\sum_{k=1}^{N} (x_{k-1} - x_k)^2$$

$$\mathbb{V}(C) = \sigma^2\tau\sum_{k=1}^{N} x_k^2$$

where $\hat{\eta} = \eta - \frac{1}{2}\gamma\tau$

The optimization problem rewrites:

$$min_x U(C) = min_x \, [\frac{1}{2}\gamma X^2 + \epsilon X + \frac{\hat{\eta}}{\tau}\sum_{k=1}^{N} (x_{k-1} - x_k)^2 + \lambda\sigma^2\tau\sum_{k=1}^{N} x_k^2]$$

The first order conditions are:

$$\text{for j=1,...,N-1} \;:\; \frac{\partial U}{\partial x_j} = 2\tau(\lambda\sigma^2 x_j - \hat{\eta}\frac{x_{j-1} - 2x_j + x_{j+1}}{\tau^2})$$

From the border conditions and the first order conditions we have:

$$x_0 = X, \; x_T = 0$$

$$\text{for j=1,...,N-1} \quad \frac{x_{j-1} - 2x_j + x_{j+1}}{\tau^2} = \hat{K}x_{j+1}$$

with $\hat{K} = \lambda\sigma^2/\hat{\eta}$

Thus we have the following optimal solution for the trading strategy:

$$x_0 = X$$

$$\text{for j=1,...,N} \ : \ x_j = \frac{sinh(K(T - t_j))}{sinh(KT)} X$$

where K satisfies: $cosh(K\tau) = \tau^2 \hat{K}/2 + 1$

2.2 Reinforcement Learning

2.2.1 General principle of Reinforcement Learning

Reinforcement Learning is when an agent learns from an environment to take the right action at the right time. During the learning process, the agent is either penalized or rewarded depending on the action taken and its outcome. The agent is immersed in an environment, and makes decisions based on the current observed state. These decisions impact the environment and, in return, the agent receives positive or negative feedback on the action he undertook. The agent seeks, through iterative experiences, an optimal strategy. It is in this sense that he maximizes the sum of rewards over time. The agent's final behaviour is therefore the result of an environment-agent interaction.

2.2.2 Reinforcement Learning in the multi-agent liquidation framework

We consider a trader who aims to sell X shares of one stock within a time frame T. On the last day of the time frame, the liquidation process ends and the number of shares should be 0. Since the trading volume is tremendous, the market price P will drop during selling, temporarily and permanently, potentially resulting in enormous trading costs. The trader seeks to find an optimal selling strategy, minimizing the expected trading cost E(X), or called implementation shortfall, subject to certain optimization criterion. The trader has access to all the visible environment information but he does not know about the other agents' decisions. The problem of an optimal liquidation strategy is investigated using the Almgren-Chriss market model on the background that the agents liquidate assets completely in a given time frame. The model serves as the trading environment such that when agents make selling decisions, the environment would return price information. Given the stochastic and interactive nature of the trading market, the authors model the stock trading process as a Markov decision process, which is specified as follows:

- State $s = [r; m; l]$: a set that includes the information of the log-return, where D is the number of days of log-return, the remaining number of trades m normalized by the total number of trades, the remaining number of shares l, normalized by the total number of shares.
- Action a: interpret the action a_k as a selling fraction. In this case, the actions will take continuous values in between 0 and 1.
- Reward $R(s; a)$: to define the reward function, they use the difference between two consecutive utility

functions. Here are the formulas :

$$U(\boldsymbol{x}) = E(\boldsymbol{x}) + \lambda V(\boldsymbol{x})$$
$$E(\boldsymbol{x}) = \sum_{k=1}^{N} \tau x_k g\left(\frac{n_k}{\tau}\right) + \sum_{k=1}^{N} n_k h\left(\frac{n_k}{\tau}\right)$$
$$V(\boldsymbol{x}) = \sigma^2 \sum_{k=1}^{N} \tau x_k^2$$

where λ is the risk aversion level, and x is the trading trajectory or the vector of shares remaining at each time step $k, 0 \leq t_k \leq T$. After each time step, they compute the utility using the equations for $E(\boldsymbol{x})$ and $V(\boldsymbol{x})$ from the Almgren and Chriss model for the remaining time and inventory while holding parameter λ constant. Denoting the optimal remaining trading trajectory computed at time t by $\boldsymbol{x}_t^*$, they define the reward as:

$$R_t = U_t\left(\boldsymbol{x}_t^*\right) - U_{t+1}\left(\boldsymbol{x}_{t+1}^*\right)$$

- Policy $\pi(s)$: The liquidation strategy of stocks at state s. It is essentially the distribution of selling percentage a at state s.
- Action-value function $Q_\pi(s; a)$: the expected reward achieved by action a at state s, following policy π.

They specify the Multi-agent Reinforcement Learning setting as follows :

- states $s = [r, m, l]$: in a multi-agent environment, the state vector should contain information about the remaining stocks of each agent. Therefore, in a J agents environment, the state vector at time t_k would be:
$$[r_{k-D}, \ldots, r_{k-1}, r_k, m_k, l_{1,k}, \ldots, l_{J,k}]$$
where
  - $r_k = \log\left(\frac{P_k}{P_{k-1}}\right)$ is the log-return at time t_k
  - $m_k = \frac{N_k}{N}$ is the number of trades remaining at time t_k normalized by the total number of trades.
  - $l_{j,k} = \frac{x_{j,k}}{X_j}$ is the remaining number of shares for agent j at time t_k normalized by the total number of shares.
- Action a : using the interpretation in Section 3.1, we can determine the number of shares to sell for each at each time step using:
$$n_{j,k} = a_{j,k} \times x_{j,k}$$
where $x_{j,k}$ is the number of remaining shares at time t_k for agent j
- Reward $R(s, a)$: denotes the optimal trading trajectory computed at time t for agent j by $x_{j,t}^*$, we define the reward as:
$$R_{j,t} = U_{j,t}\left(\boldsymbol{x}_{j,t}^*\right) - U_{j,t+1}\left(\boldsymbol{x}_{j,t+1}^*\right)$$
- Observation O : Each agent only observes limited state information. In other words, in addition to the environment information, each agent only knows its own remaining shares, but not other agents' remaining shares. The observation vector at time t_k for agent j is:
$$O_{j,k} = [r_{k-D}, \ldots, r_{k-1}, r_k, m_k, l_{j,k}]$$

Finally, we can present the whole Reinforcement Algorithm implemented in the article and in this project.

Deep Deterministic Policy Gradient-Based Multi-agent Training

Input: number of episodes M, time frame T, minibatch size N, learning rate λ, and number of agents J
for $j = 1, J\%$ initialize each agent separately **do**

- Randomly initialize critic network $Q_j\left(O_j, a|\theta_j^Q\right)$ and actor network $\mu_j\left(O_j|\theta_j^\mu\right)$ with random weight θ_j^Q and θ_j^μ for agent j
- Initialize target network Q'_j and μ'_j with weights $\theta_j^{Q'} \leftarrow \theta_j^Q, \theta_j^{\mu'} \leftarrow \theta_j^\mu$ for each agent j
- Initialize replay buffer B_j for each agent j

end for
for episode $= 1, M$ **do**

- Initialize a random process $\mathcal{N}$ for action exploration;
  Receive initial observation state s_0
- **for** $t = 1, T$ **do**
  - **for** $j = 1, J\%$ train each agent separately **do**
    Select action $a_{j,t} = \mu_j\left(O_{j,t}|\theta_j^\mu\right) + \mathcal{N}_t$ according to the current policy and exploration noise;

  end for
- Each agent executes action $a_{j,t}$
- Market state changes to s_{t+1}
- Each agent observes reward $r_{j,t}$ and observation $O_{j,t+1}$
- **for** $j = 1, J$ **do**
  - Store transition $(O_{j,t}, a_{j,t}, r_{j,t}, O_{j,t+1})$ in B_j
  - Sample a random minibatch of N transitions $(O_{j,i}, a_{j,i}, r_{j,i}, O_{j,i+1})$ from B_j
  - Set $y_{j,i} = r_{j,i} + \gamma Q'_j\left(s_{t+1}, \mu'_j\left(O_{j,i+1}\left|\theta_j^{\mu'}\right|\theta_j^{Q'}\right)\right)$ for $i = 1, \ldots, N$
  - Update the critic by minimizing the loss: $L = \frac{1}{N}\sum_i\left(y_{j,i} - Q_j\left(O_{j,i}, a_{j,i}|\theta_j^Q\right)\right)^2$
  - Update the actor policy by using the sampled policy gradient:

$$\nabla_{\theta^\mu}\pi \approx \frac{1}{N}\sum_i \nabla_a Q_j\left(O, a|\theta_j^Q\right)\Big|_{O=O_{j,i}, a=\mu_j(O_{j,i})} \times \nabla_{\theta^\mu}\mu_j\left(O_j|\theta^\mu\right)\big|_{s_i}$$

  - Update the target networks: $\theta_j^{Q'} \leftarrow \tau\theta_j^Q + (1-\tau)\theta_j^{Q'}$ $\theta_j^{\mu'} \leftarrow \tau\theta_j^\mu + (1-\tau)\theta_j^{\mu'}$
    end for

  end for

end for

3 Theoretical foundations of the Deep Deterministic Policy Gradient Algorithm

The reinforcement learning algorithm used by Bao and Liu is the Deep Deterministic Policy Gradient (DDPG) algorithm. It was introduced in Lillicrap et al. (2016). In this section, we present some theoretical results that help understand where the DDPG algorithm comes from. These theoretical results are mainly taken from Silver et al. (2014) and from Sutton et al. (1999).

3.1 Stochastic Policy Gradient

We consider a standard reinforcement learning setup consisting of an agent interacting with an environment E in discrete timesteps. At each timestep t the agent receives an observation O_t, takes an action a_t and receives a scalar reward r_t.

Notations and definitions:
- $p_1(s_1)$: initial state distribution;
- $p(s_{t+1}|s_t, a_t)$: stationary transition dynamics distribution satisfying Markov property;
- $\pi_\theta : S \to \mathcal{P}(\mathcal{A})$: stochastic policy with vector of parameters θ that maps state $s \in S$ to the probability distribution $a \in \mathcal{A} \mapsto \pi_\theta(a|s)$;
- $r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k)$: total discounted reward as seen from timestep t, with $0 \leq \gamma \leq 1$ the discounting factor;
- $Q^\pi(s, a) = \mathbb{E}[r_1^\gamma | S_1 = s, A_1 = a; \pi]$: action-value function;
- $J(\pi) = \mathbb{E}[r_1^\gamma | \pi]$: the objective function the agent wants to maximize;
- $p(s \to s', t, \pi)$: probability distribution of joining state s' in t steps starting from state s and following policy π;
- $\rho^\pi(s') = \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \to s', t, \pi) ds$: (improper) discounted state distribution.

Using the definitions of the reward and the discounted state distribution, the objective function can be rewritten as $J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(s, a) r(s, a) da ds$, which leads to:

$$J(\pi_\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[r(s, a)].$$

Sutton et al. (1999) derived a simple formula for the gradient of the objective function.

Theorem 1 (Stochastic Policy Gradient Theorem):

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)].$$

This numerical estimate of $\nabla_\theta J(\pi_\theta)$ enables **policy gradient ascent**, which is way more appealing than straightforward Q-learning when dealing with continuous actions. We notice that, even though $\rho^\pi(s)$ depends on θ, the policy gradient does not depend on $\nabla_\theta \rho^\pi(s)$. As a result, we can simply sample simulation paths, and at each time step, we calculate $\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)$. But how do we estimate $Q^\pi(s, a)$?

In practice, we use an estimator function called **actor** to estimate the policy π_θ, and an estimator function $Q^w(s, a)$ called **critic** to estimate the action-value function $Q^\pi(s, a)$. But in general, using w instead of θ induces a bias in the estimation of the action-value function. Fortunately, Sutton at al. (1999) provide

conditions under which there is no bias in the estimation.

Theorem 2 (Compatible Function Approximation Theorem):
If the two following conditions are satisfied:
1. $\nabla_w Q^w(s,a) = \nabla_\theta \log(\pi_\theta(a|s))$
2. w minimizes the error: $\mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[(Q^w(s,a) - Q^\pi(s,a))^2]$
Then $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log(\pi_\theta(a|s))Q^w(s,a)]$ i.e. there is no bias in the policy gradient that uses the critic $Q^w(s,a)$.

A simple way to enable compatible approximation is to set $Q^w(s,a)$ to be linear in its features. In practice, the second condition is relaxed in favour of policy evaluation algorithms that estimate the action-value function more efficiently. Such algorithms often rely on the recursive Bellman equation:

$$Q^\pi(s_t,a_t) = \mathbb{E}_{r_t,s_{t+1}\sim E}[r(s_t,a_t) + \gamma \mathbb{E}_{a_{t+1}\sim\pi}[Q^\pi(s_{t+1},a_{t+1})]].$$

To estimate policy gradient according to theorem 1, we need to generate paths that depend on the current policy. But how can we learn about the optimal policy if we behave according to the current policy? One idea is to use two policies: one that is learned and that becomes the optimal policy, and another that is more exploratory and that is used to generate behavior. We call these policies target policy and behavior policy. Since the learning is from paths "off" the target policy, the learning process is called **off-policy learning**.

In an off-policy setting, the objective function is typically modified:

$$J_\beta(\pi_\theta) = \int_{\mathcal{S}} \rho^\beta(s) \int_{\mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) da ds,$$

and the gradient becomes:

$$\nabla_\theta J_\beta(\pi_\theta) \approx \mathbb{E}_{s\sim\rho^\beta, a\sim\beta}[\frac{\pi_\theta(a|s)}{\beta_\theta(a|s)} \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)].$$

In the last approximation, we omitted the term with the gradient of the action-value function. According to Degris et al. (2012), this approximation preserves the sets of local minima to which the gradient converges. The expectation also exhibits an importance-sampling ratio that adjusts for the fact that we generate paths according to β rather than π.

3.2 Deterministic Policy Gradient

Silver et al. (2014) extend the previous results to deterministic policies. Instead of a stochastic policy with distributions $\pi_\theta(a|s)$, we consider a deterministic policy i.e. a function $\mu_\theta : S \to \mathcal{A}$ that maps a given state to a single action.

In the deterministic case, the objective function simply writes:

$$J(\mu_\theta) = \int_{\mathcal{S}} \rho^\mu(s) r(s, \mu_\theta(s) ds = \mathbb{E}_{s\sim\rho^\mu}[r(s, \mu_\theta(s)].$$

And the following theorem allows for deterministic policy gradient ascent:

Theorem 3(Deterministic Policy Gradient Theorem):

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}].$$

In a deterministic off-policy setting, the objective function writes:

$$J_\beta(\mu_\theta) = \int_{\mathcal{S}} \rho^\beta(s) Q^\mu(s, \mu_\theta(s)) ds,$$

and its gradient is approximated by:

$$\nabla_\theta J_\beta(\mu_\theta) \approx \mathbb{E}_{s\sim\rho^\beta}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}].$$

As in the stochastic case, we have in the deterministic case a theorem that enables the use of a critic to estimate the action-value function. This theorem is valid on and off-policy, which makes it very appealing. Below, $\mathbb{E}[.]$ denotes either $\mathbb{E}_{s\sim\rho^\mu}[.]$ or $\mathbb{E}_{s\sim\rho^\beta}[.]$.

Theorem 4 (Approximation Theorem for Deterministic Policies):
If the two following conditions are satisfied:
1. $\nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} = \nabla_\theta \mu_\theta(s).w$
2. w minimizes the error: $\mathbb{E}[\epsilon(s;\theta,w).\epsilon(s;\theta,w)]$ where $\epsilon(s;\theta,w) = \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} - \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}$.
Then there is no bias in the deterministic policy gradient that uses the critic $Q^w(s,a)$.

3.3 Deep Deterministic Policy Gradient

Now that we have in mind some important theoretical results on policy gradients, we are able to understand the Deep Deterministic Policy Gradient (DDPG) algorithm derived by Lillicrap et al. (2016) and used by Bao and Liu in their multi-agent liquidation framework.

The DDPG algorithm is nothing else than an off-policy deterministic actor-critic algorithm in which neural networks are used as function approximators of the actor and the critic, and in which the off-policy trajectories used for learning are randomly drawn from past experiences of the agent.

As the policy is deterministic, the recursive Bellman equation writes as follows:

$$Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}\sim E}[r(s_t,a_t) + \gamma Q^\mu(s_{t+1}, a_{t+1})].$$

Hence the parameters w^Q of the critic network are updated at each timestep by minimizing the loss:

$$L(w^Q) = \mathbb{E}_{s_t\sim\rho^\beta, a_t\sim\beta, r_t}[Q(s_t,a_t|w^Q) - y_t],$$

where $y_t = r(s_t,a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|w^Q)$.

However, with a neural network as critic approximator, condition 1 of theorem 4 is no longer satisfied and there may be a bias in the estimation of the deterministic policy gradient. The Q-update may thus be prone to instability and divergence. As a solution to this problem, Lillicrap et al. (2016) use "soft" target updates. They create a copy of the actor and the critic networks before learning. These target networks are slowly updated keeping track of the learned networks: $w^i_{target} \leftarrow \tau w^i_{target} + (1-\tau) w^i_{learned}$ with $\tau << 1$ and $i \in \{actor, critic\}$. This trick may slow the learning process, but it greatly improves its stability.

At each step of the DDPG algorithm, an off-policy batch (i.e. a finite number of off-policy transition states) is randomly drawn from the replay buffer. The replay buffer is a set of transitions states that accumulate during the learning process until maximal capacity of the replay buffer is reached. When the replay buffer is full, the oldest states are removed. The replay buffer can be large compared to the batch size, which guarantees learning from uncorrelated samples.

The use of two actor networks in the DDPG makes it possible to incorporate exploration noises in the learning process without taking too much risk. This noise must be chosen according to the environment. Bao and Liu chose an Ornstein-Uhlenbeck process to add noise to the actions generated by the learned network.

4 Algorithm results

Now that we have laid the theoretical foundations of this article, we present and analyse in this section the results of the authors' algorithm.

4.1 Environment

The algorithm is executed in the following market environment. The total number of shares to sell is 1Mio and the initial price is \$50. The stock price has an annual volatility of 12% and the bid-ask spread is fixed to \$1/8 = \$0.125. The daily volume traded on this stock is 5Mio shares/day. The liquidation has to be done in a time $T = 60$ days with a number of trade $N = 60$, which gives $\tau = \frac{T}{N} = 1$ meaning one trade per day. The fixed cost of selling is set to half the bid-ask spread, meaning when the trading rate v is very close to zero we have the temporary impact $h(v) = 1/2 * 1/8 = 1/16$ so $\epsilon = 1/16$. We set η such that each percent of the daily volume traded has a temporary impact equal to one bid-ask spread, so $\eta = \frac{1/8}{0.01*5e6} = 2.5e-6$. The permanent impact of selling 10% of the daily volume is also set to one bid-ask spread so $\gamma = \frac{1/8}{0.1*5e6} = 2.5e-7$.

In order to normalize the reward, the following formula is used:

$$R_{j,t} = \frac{U_{j,t}\left(\boldsymbol{x}^*_{j,t}\right) - U_{j,t+1}\left(\boldsymbol{x}^*_{j,t+1}\right)}{U_{j,t}\left(\boldsymbol{x}^*_{j,t}\right)}$$

4.2 Results

As we expected, in a multi-agent environment the equilibrium is different from the single-agent framework provided by Almgren and Chriss.

The first result illustrates the theorem 4.1 on page 5 of the article: to liquidate 1Mio shares in a single agent model leads to a greater expected shortfall than liquidate 0.3Mio + 0.7Mio shares split between two agents in a multi-agent environment. We can see this results on the figure below, where all the agents have the same risk aversion $\lambda = 1e-6$.

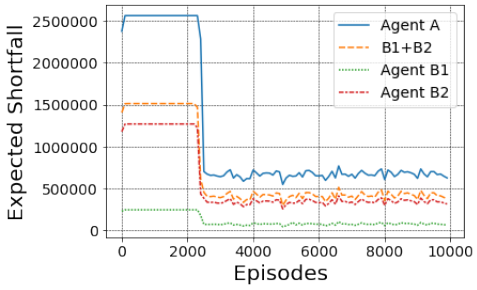

Figure 1: The expected shortfall of agent A is higher than the sum of expected shortfalls of agents B1 and B2.

As it moves the equilibrium, it also moves the trading strategies. One agent's decisions influence the other

agent's. On the figure below, we can see that trading strategies are closer to each other when training agents in a multi-environment framework. This means that, as the more aggressive agent is trading, the market moves in such a way that the other agent (more risk averse) has to adopt a strategy close to the first agent's one in order not to be left behind.

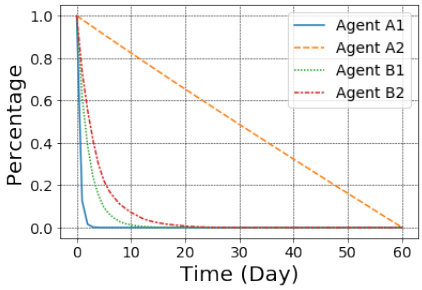

Figure 2: Trading strategies of two agents trained in a single environment: A1 and A2 with $\lambda_{A1} = 1e-4$ and $\lambda_{A2} = 1e-9$. Trading strategies of their counterparts trained together in a multi-agent environment: B1 and B2 with $\lambda_{B1} = 1e-4$ and $\lambda_{B2} = 1e-9$.

We can see that agent B2 has to be more aggressive than agent A2, with the same risk aversion, to accomplish its optimal strategy. The same way, agent B1 has to be less aggressive that agent A1 because there is agent B2 moving the market as well.

Multi-agent environment can be used to analyse the effect of coordinated behavior on the market. We analyse two cases of coordination between our two agents: cooperation and competitive behavior. The agents have to sell 0.5Mio shares and have the same risk aversion $\lambda = 1e-6$.

To give a cooperative behavior to the agents, we define the reward as below:

$$\hat{R_{1,t}} = \hat{R_{2,t}} = \frac{R_{1,t} + R_{2,t}}{2}$$

The agent will be fully cooperative as their reward depends as much on their result as on the other agent's.

To give a competitive behavior to the agents, we define the reward as below:
if $R_{1,t} > R_{2,t}$:
$\hat{R_{1,t}} = R_{1,t}$
$\hat{R_{2,t}} = R_{2,t} - R_{1,t}$
else:
$\hat{R_{2,t}} = R_{2,t}$
$\hat{R_{1,t}} = R_{1,t} - R_{2,t}$

Here the objective is to be always above the other agent's result in order not to be penalized. Our reward depends only on our position compared to the other agent's, regardless of how good we perform.

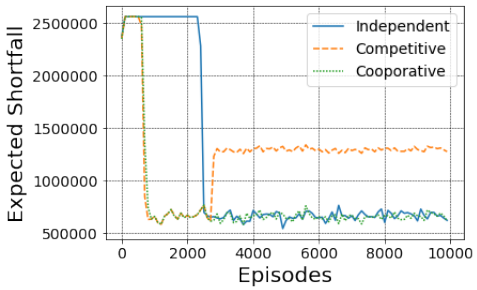

Figure 3: The expected shortfall of three agents regarding their behavior.

On the figure above, we compare the sum of expected shortfalls of two independent agents each selling 0.5 Mio shares and trained independently in a single-agent framework, with the sum of the expected shortfalls of two agents with cooperative (or competitive) behavior, each selling 0.5 Mio shares and trained in a multi-agent framework. We can see that the interaction between agents has a good impact on their speed of learning: the expected shortfall falls to its lower values within around 700 episodes compared to 3000 episodes for independent agents. We see that these lower values are the same, regardless of their interactions or the absence of interaction. This means that we cannot beat the optimal shortfall of Almgren and Chriss. However, when the agents are in a blind competition, the expected shortfalls increase a lot after a few episodes because the competition make them more aggressive in their strategy and it results in greater trading costs. They are less penalized with greater trading costs than to have a negative reward, being below the other agent.

5 Experimentations

In this section we detail changes we made in the framework proposed by the authors and analyze the consequences on the results of the algorithm.

5.1 Not blind competition

We have seen that the main default of the competitive behavior of agents was the blind competition or competition at all prices. Thus, we create a new behavior such that agents are cooperative up to a certain point. If one of them is doing much better than the other, then the latter becomes "jealous" and enters in competition behavior for this round. The best agent remains in cooperative behavior. Reward is then defined as follow:

if $R_{1,t} > 1.5 * R_{2,t}$:
$\hat{R_{1,t}} = (R_{1,t} + R_{2,t})/2$
$\hat{R_{2,t}} = 0.5 * (R_{2,t} - R_{1,t})$
elif $R_{2,t} > 1.5 * R_{1,t}$:
$\hat{R_{2,t}} = (R_{1,t} + R_{2,t})/2$
$\hat{R_{1,t}} = 0.5 * (R_{1,t} - R_{2,t})$
else:
$\hat{R_{1,t}} = (R_{1,t} + R_{2,t})/2$
$\hat{R_{2,t}} = \hat{R_{1,t}}$

We chose a discriminant factor of 1.5 but further study could analyze the effect of this factor on agents' behaviors. A factor equal to one would lead to a blind competition (up to the case $R_1 = R_2$). A factor going to infinity would lead to a full cooperation.

On the figures below we represent the expected shortfall of our new agents compare to the ones of the article. We used the same market framework as before, we trained two "not blind-competitive" agents in the multi-agent framework, each selling 0.5Mio shares and with a risk aversion of 1e-6. We plot the sum of the expected shortfall of our two agents. Due to the computation resources needed to train the algorithm, we did our analysis on 3000 episodes.

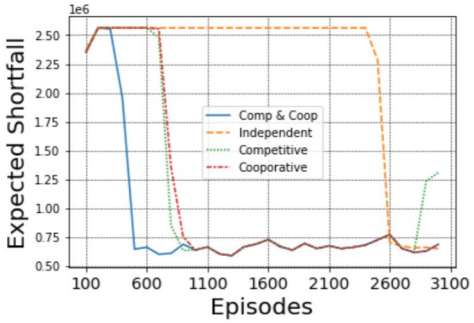

Figure 4: The expected shortfall of four agents regarding their behavior.

We can see that our new agents are learning much faster than the blind-competitive and fully cooperative ones. It takes around 500 episodes to reach the lower values of expected shortfall. Unlike blind competition behavior, it seems that our agents do not increase their expected shortfall after a certain time. This assumption certainly depends on the discriminant factor chosen. It would need to be checked with a greater number of episodes because competition, even not blind, might lead to worse results after an amount of time positively correlated with the discriminant factor.

The figure below represents the average trajectory of our two agents, trained with the different behaviors mentioned above. The independent agent is trained in a single-agent framework. We see here an interesting fact: the slight competition introduced allows our new agents to be slightly more aggressive in their strategy, leading to a quicker liquidation compared other behaviors. The competitive behavior leads to the less efficient trajectory. However, cooperation allows agents to liquidate quicker by coordinating their actions. Introducing a taste of competition allows agents to be aggressive on very specific cases, when the competitor is well above, leading to quicker liquidation without ruining the mutual gain.

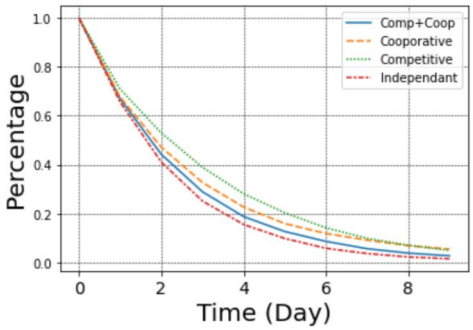

Figure 5: The trading trajectories of four agents regarding their behavior.

5.2 Non-linear market impact

The authors chose the Almgren and Chriss market framework which implies a linear price impact on the market when executing trades. Several papers on market microstructure have shown that the volume of the order book must be linear with the price when limit order prices are getting closer to the spot price. This results in a market impact proportional to the square root of the order volume. Then, we changed the market impact and follow the framework proposed by Toth and al (2018). Here our agent has to execute a metaorder of 0.5Mio shares. The metaorder is split into smaller orders and executed trough time. From Toth and al (2018), the market impact of a meta order of volume Q, with a daily volume V, a daily volatility σ, P_0 the spot price and Y a constant of order 1 is:

$$\Delta Q = Y \sigma \sqrt{\frac{Q}{V}} P_0$$

More generally, the square root can be replaced by a power law with a factor between 0.4 and 0.7. In this paper they also provide the market impact of metaorders executed on the future market (log-scale):

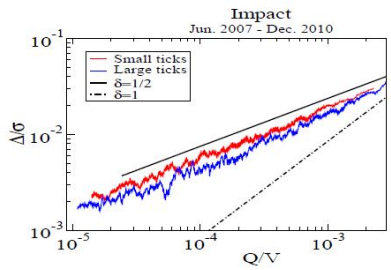

Figure 6: The impact of metaorders for CFM proprietary trades on futures markets, in the period June 2007 to December 2010.

In this study containing nearly 500,000 trades, we can see that the square root law is verified empirically.

To the light of this result, we changed the market impacts as follow:

$$h(v_k) = \epsilon + \eta\sqrt{v_k}$$

$$g(q_k) = \Delta Q \frac{\sqrt{q_k} - \sqrt{q_{k-1}}}{\sqrt{Q}}$$

As the book order is linear, the temporary impact must evolve with the square root of the volume instantaneously traded. However, the permanent impact at the end of the order, according to Toth and al (2018), should be ΔQ. As we do not know when a metaorder will be finished, it is logic to have the evolution of its permanent impact of the market as $\Delta Q * \sqrt{\frac{q_k}{Q}}$, where q_k is the total volume executed at time k since T_0. In our case, the function g is the increment of the impact, thus the subtraction of the previous impact. As $q_T = Q$ and $q_0 = 0$, the total permanent impact defined as $\sum_{k=0}^{T} g(q_k)$ is equal to ΔQ as expected.

We trained the multi-agent model on this market framework for all the behavior defined above. However, the results of the independent agent is still with the linear market impacts.

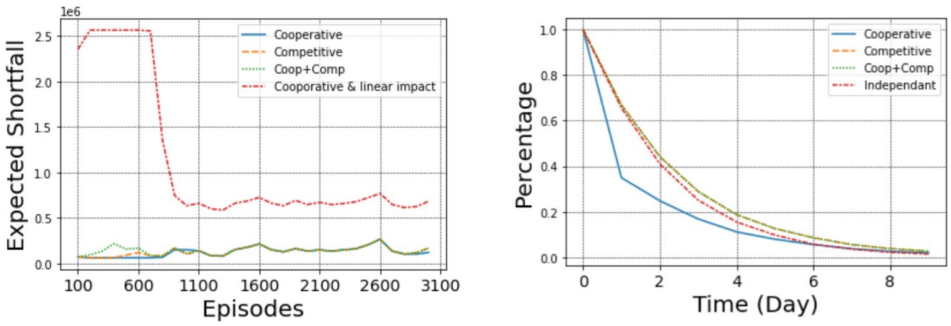

Figure 7: Expected shortfalls and trading trajectories of four agents with non-linear market impacts.

Regarding the expected shortfalls, we can see on the figure above that the average value is well below its counterpart with linear price impact. This was expected: as the number of shares sold is greater than one, the square root impact is always smaller than the linear impact. However, an interesting fact is that the

agents seem to start with small values and to reach very quickly their optimal gain. We also see that the blind-competitive agents do not increase their expected shortfall in the range of 3000 episodes as before. As in the previous framework, the optimal expected shortfall is the same regardless of the agent behavior. The similarities between the curves of square root and linear impacts may come from the fact that we changed only the price impacts. We did not change the utility function that drives the agent's learning and that is derived from Almgren and Chriss model. The closed form formula of the utility with non-linear impacts being hard to obtain, we chose not to include it here. However, this is an interesting fact to look at for further studies.

The cooperative agents seem to over perform the other behaviors regarding trading trajectories. The competitive agents we created keep the same trajectory. The blind-competitive agents have now the same trajectory as the competitive agents we created. We can explain this by the fact that, as we said above, the square root impact is smaller than the linear one, so the competitive agents can liquidate much faster. For the same reason, we see that our agents liquidate much more faster than in the linear case.

6 Some comparative statistics

In this section we show our results of the analysis of the expected shortfall sensibility to some exogenous parameters.

Initial set of parameters: liquidation time = 60, number of trades = 60, episodes = 500, risk aversion = 1e-6.

Varying risk aversion.

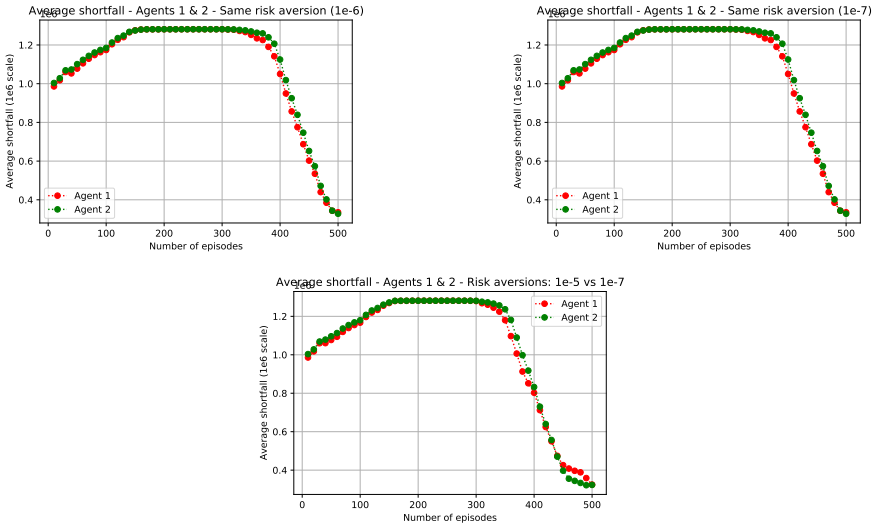

Figure 8: Various risk aversion.

Liquidation time.

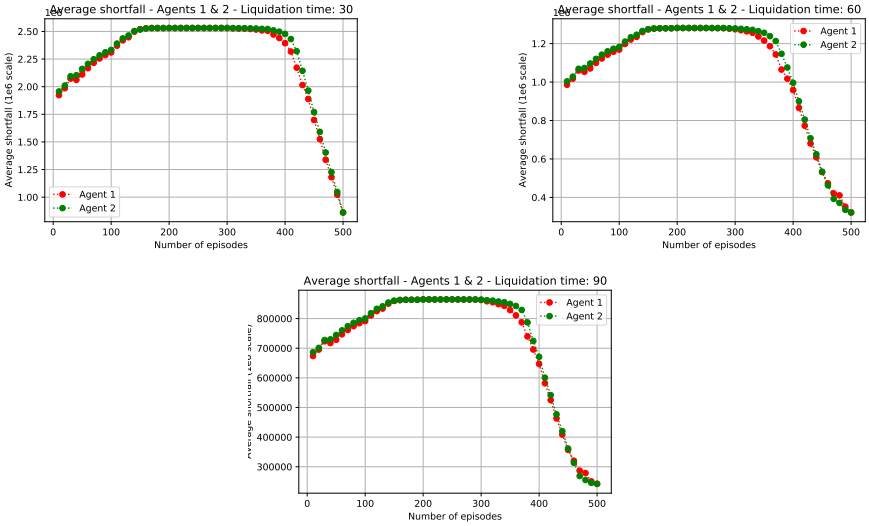

Figure 9: Liquidation time.

Number of trades.

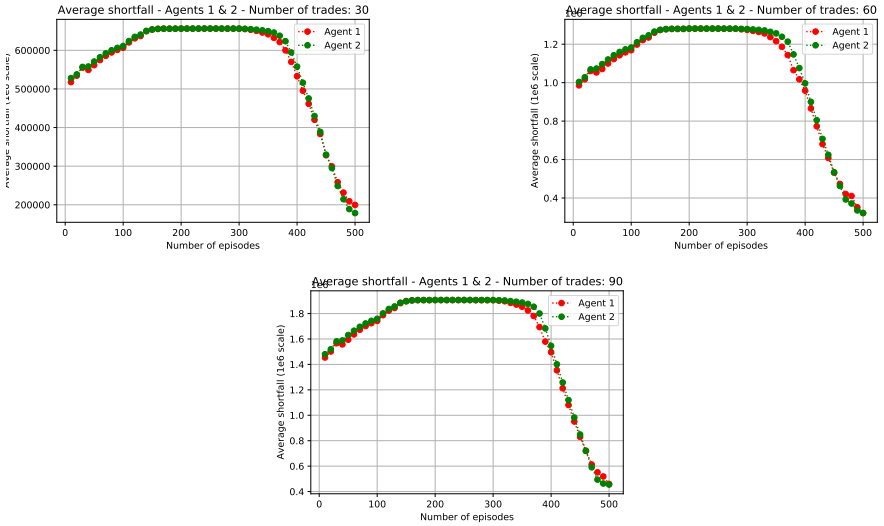

Figure 10: Number of trades.

As we already observed on figure 2 on page 11, the trading trajectories are much less sensitive to risk aversion in a multi-agent model than in a single-agent model. This result is also observable on expected shortfall: figure 8 shows that there is not much sensibility to risk aversion. The shape of the curves is very similar and the upper and lower bounds are very close.
Liquidation time has a big impact on expected shortfall as we can see on figure 9. Although the curves have the same shape, their bounds are very different. The more time we give to our agents to liquidate their position, the lower their expected shortfall will be. Keeping the number of trades at 60, a liquidation time of 30 days implies 2 trades per day, resulting in much more market impact and an expected shortfall up to \$2.5Mio. On the contrary, a liquidation time of 90 days, thus one trade every day and a half, gives an expected shortfall up to \$0.8Mio. The lower bound is not observable on the first graph, but for a liquidation time of 60 days we can see that it is between \$0.3Mio and \$0.4Mio. For 90 days, it is between \$0.2Mio and \$0.3Mio. From these observations, we can say that the upper bound of the expected shortfall is decreasing with the liquidation time. Furthermore, we can strongly assume that the lower bound of the expected shortfall is also decreasing with the liquidation time, though we must verify this assumption with a greater number of episodes. These results are quite intuitive, because when giving more time to the agent and keeping the number of trades constant, we give him more freedom for its strategy.
Regarding the sensibility of the expected shortfall to the number of trades, we observe on figure 10 the opposite of what was seen for the liquidation time. The lower and upper bounds are increasing with the number of trades the agents must do, keeping liquidation time constant. The parameter of interest here is the trading rate, when increasing (by either increasing the number of trades or decreasing the liquidation time) we increase the expected shortfall. The fixed cost of selling is an important parameter to consider when setting the number of trades. Also, the recovery time from the temporary impact is another parameter to carefully consider.

7 Discussion

There are a few points assumed in this paper that should not be taken for granted and can be widely discussed.

First, the algorithm was trained within a 2-agent framework, with each selling the same amount of shares. In order to fit better with reality, we must train the model with a much greater number of agents, with their total number of shares to sell being a significant proportion of the daily volume. Indeed, we saw that the sum of two agents is different from a single agent selling the same total amount of shares. Thus, we cannot train our model and assume that the second agent represents all the remaining agents of the markets, even selling a much greater number of shares than the first agent. The agents have to be individually defined. Considering the computational resources of the algorithm, the model might be very difficult to train with a significant number of agents. However, considering the extent of application of Almgren and Chriss solution by market participants, it could be interesting to create a market environment composed of many A&C agents and a few RL agents.

In our study we tried to develop the market price impacts and redefined them to non-linear functions. Inspiring from studies on market micro-structure (see Toth and al (2018)), we chose a square root temporary and permanent impact. This point must be deepened by also redefining the objective function of the agents. The expected shortfall and the variance must be changed to fit with such kind of market impacts.
As we mentioned in the last part, the recovery time from the temporary market impact is an important parameter to consider. Here, by the construction of the algorithm, the authors assumed that this time is one period i.e. one day. In reality, this parameter is much bigger and should be defined with a proper order book model. This time, named τ_{life} in Toth and al (2018), is the characteristic time to reach stationary state after a perturbation of the book order (i.e. the execution of an order). We know that when $T >> \tau_{life}$, we have a linear market impact, so the first model were not completely wrong because we had $T = 60$ and $\tau_{life} = 1$, but the time unit should not be days in that case. Construct an order book model that is fitting to reality of market impact is complex and was not the purpose of this study. Consider a RL model for multi-agent market making could be a very interesting point to combine with the RL execution model considered here. Such a work could be inspired from Ganesh and al (2019).

8 Conclusion

Bao and Liu (2019) proposed an interesting model to develop the framework of Almgren and Chriss (2001) by introducing a multi-agent market environment. As we have seen, their model produced interesting results regarding the interactions of agents and how well RL agents can change their behavior depending on the other's actions, when adapting their reward function. We also saw that this model has limits and does not reflect market reality, thus it should not be considered as a "magical solution" to execution strategies. Even though these strategies have been studied for many years, there are still difficulties to model market impacts and more generally, order books. An important point to mention is the computational resources needed to train such RL models. Although it would be a better fit to reality to change some parameters, as the number of RL agents dealing with one another, this would require a great amount of time to train such models.

References

Almgren, R. and Chriss, N. (2001). Optimal execution of portfolio transactions. *Journal of Risk, 3:5–40*.

Bao, W. and Liu, X.-y. (2019). Multi-agent deep reinforcement learning for liquidation strategy analysis. *arXiv preprint arXiv:1906.11046*.

Degris, White, and Sutton (2012). Linear off-policy actor-critic. *29th International Conference on Machine Learning*.

Ganesh and al (2019). Reinforcement learning for market making in a multi-agent dealer market. *arXiv:1911.05892v1*.

Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, and Wierstra (2016). Continuous control with deep reinforcement learning. *ICLR*.

Silver, Lever, Heess, Degris, Wierstra, and Riedmiller (2014). Deterministic policy gradient algorithms. *ICML*.

Sutton, McAllester, Singh, and Mansour (1999). Policy gradient methods for reinforcement learning with function approximation. *Neural Information Processing Systems*.

Toth and al (2018). Anomalous price impact and the critical nature of liquidity in financial markets. *arXiv:1105.1694v3*.