Proposals for Parameter Optimization in dAMMs (WIP)

mystericalfox

December 2021

Abstract

We provide some approaches for optimally parameterizing a vAMM (or an entirely configurable vAMM, also known as a dAMM). Our first approach treats the vAMM as a reinforcement learning problem, from which we can determine a temporal-based optimal policy for parameterizing the vAMM based on trading activity. Our second approach uses machine and deep learning techniques on historical data collected from the vAMM to obtain an alternative policy for setting the peg multiplier/k.

1 dAMM Background

The aim of the dAMM is to automate the virtual market-making process through the constant product formula $x \cdot y = k$, where x can be considered as the "base asset amount", y can be considered as the "quote asset amount", and k is an arbitrary constant considered to be the "liquidity depth" we wish to maintain.

Assets in a base-quote pair are priced as follows:

$$\text{price}_x = \frac{y}{x} \cdot \text{peg multiplier}$$

For simplicity, we first consider a cross margin scenario (i.e. a protocol with only one quote asset, USDC). An example of a cross margin protocol implementing the dAMM is Drift Protocol.

We will also primarily consider the problem of finding an optimal "peg multiplier" and "k" for the vAMM; other parameters, such as traunch limits for the fee pool, may exist within the dAMM, however these seem to be more arbitrary in nature/independent of trading volume.

2 Q-Learning/Reinforcement Learning Approach

2.1 Previous Applications of Q-Learning in Finance

The idea of using deep learning techniques in traditional finance isn't necessarily a new one; for instance, Mani et al. [2019] introduced a framework for automated market-making using risk-sensitive reinforcement learning. In particular, they established an agent (market maker)-environment (trading activity) model by defining it as a finite Markov decision process in discrete time with a linear reward function, and solving it using temporal difference learning algorithms. Moreover, Zhang et al. [2019] have proposed using deep reinforcement learning to design trading strategies for continuous futures contracts, considering both discrete and continuous action spaces.

2.2 Proposed Model

2.2.1 Typical Setup

In a typical portfolio-maximization problem, we might expect an agent to choose the sequence of actions that will maximize the expected utility of wealth at the end of finite time horizon T:

$$E[U(W_T)] = E[U(W_0 + \sum_{t=1}^{T} \Delta W_t)]$$

We aim to take a similar approach in our model by defining the concept of expected "wealth" (i.e. immediate reward) in the vAMM setting.

2.2.2 Notation

We aim to define the vAMM as a finite stationary Markov Decision Process (MDP), described by the tuple $M = \langle S, A, R, P \rangle$, where:

- S is a finite collection of states; in this case, the long-short imbalance at time t.

$$S_t = IMB_t = \text{long open interest}_t - \text{short open interest}_t$$

- A is a finite collection of actions; in this case, the agents' (collective) choice to long or short at time t.

$$A_t = \begin{cases} +1 & \text{for a long position} \\ -1 & \text{for a short position} \end{cases}$$

- R is the set of possible values of immediate rewards; in this case we consider "reward" to mean the fee pool (in Drift Protocol, this is 0.1% of trading volume) capped by the traunch limit (in Drift Protocol, this is 50%) at time t.

$$R_t = \text{trading volume}_t \cdot 0.001 \cdot 0.5$$

  Since Drift's trading volume (measured in USDC) can be treated as a function of price and liquidity depth, we can initially run a linear regression model to obtain a numerical approximation for trading volume in the form:

$$\text{trading volume}_t = \beta_0 + \beta_1 \cdot \text{price}_t + \beta_2 \cdot k_t$$

  Depending on the relative magnitudes of typical prices and k's, it might make sense to scale either variable by some further transformation, such as square-rooting k or the price.

  Nonetheless, the "reward" can be written in terms of both k and the peg multiplier by:

$$\begin{aligned} R_t &= (\beta_0 + \beta_1 \cdot \text{price}_t + \beta_2 \cdot k_t) \cdot 0.001 \cdot 0.5 \\ &= (\beta_0 + \beta_1 \cdot \frac{y_t}{x_t} \cdot \text{peg multiplier}_t + \beta_2 \cdot k_t) \cdot 0.001 \cdot 0.5 \\ &= (\beta_0 + \beta_1 \cdot \frac{k_t}{x_t^2} \cdot \text{peg multiplier}_t + \beta_2 \cdot k_t) \cdot 0.001 \cdot 0.5 \end{aligned}$$

- P describes the transition probabilities i.e. $P(s', r \mid s, a) = P(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$ for $s, s' \in S, a \in A, r \in R$. We expect to approximate this numerically from Drift Protocol's trading data.

Furthermore, we assume that:

- Informed traders enter the market with probability λ_i, take a long position if $mark < oracle$ and a short position if $mark > oracle$.
- Uninformed traders enter the market with probability λ_u and have an even chance of taking a long or short position.

We assume that these are mutually exclusive events such that $\lambda_i + \lambda_u = 1$.

The agent-environment dynamics is as follows: at time t, the aggregate market maker observes a state S_t and takes an action A_t according to a Markovian policy, from which a numerical reward R_{t+1} is provided at the next time step, leading to the next state S_{t+1}.

At any time t, the goal is to maximize the expected return, denoted as:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1}(\lambda_i R_k + \lambda_u R_k)$$

where $\gamma \in (0, 1)$ is a discount factor representing the agent's uncertainty about the time horizon remaining on the contract. We weight the rewards accordingly by the probability of observing a group of informed or uninformed traders, as we expect the immediate rewards to differ greatly based on strategic trading activity.

Assuming that the utility of final wealth is linear over time, we can see that optimizing $E[G]$ is also equivalent to optimizing our expected wealth.

2.2.3 Optimal Policy

Define the state-action value function $Q_\pi : S \times A \to \mathbb{R}$ for some fixed policy π as follows:

$$\begin{aligned} Q_\pi(s, a) &= E_\pi[\sum_{k=t+1}^{T} \gamma^{k-t-1}(\lambda_i R_k + \lambda_u R_k) \mid S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma \sum_{a'} Q_\pi(S_{t+1}, a')\pi(a' \mid S_{t+1}) \mid S_t = s, A_t = a] \end{aligned}$$

Thus the optimal policy is determined by:

$$\begin{aligned} Q_*(s, a) &= \max_\pi Q_\pi(s, a) \\ &= E[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \\ \implies \pi_* &= \text{argmax}_a Q_*(s, a) \end{aligned}$$

3 Supervised Machine/Deep Learning Approach

Another way to approach the optimization problem could be to use a supervised machine learning approach. In some sense, applying supervised machine learning algorithms could be useful here as it would allow us to establish relationships between different data variables, from which we could infer the optimal direction or magnitude for a new peg multiplier or k, given similar market dynamics observed historically.

3.1 Classification of Repeg vs k Adjustment

We could first use Drift's historical data to classify previous instances of "success" or "failure" during previous repegs or k adjustments. Using the Drift Flat Data as an example, we could arbitrarily classify instances as "success" or "failure" post parameter changes based on some thresholds defined by the following attributes:

- open_interest; Does changing the price or liquidity incentivize more trading volume?
- adjustment_cost as a proportion of total_fee_minus_distributions; Is the relative cost of performing the change fairly low in comparison to the fees collected post change?

Examples of appropriate classification techniques could include nearest neighbor, Naive Bayes, decision trees or logistic regression.

3.2 Choice of Repeg/k Adjustment

Based on whether we wish to to repeg or adjust k, we could then construct a recurrent neural network/convolutional neural network to predict the appropriate change for these values.

3.3 Caveats

Drift Protocol does not currently have enough historical data to ensure a robust model, especially since the team has not repegged or changed k frequently. However it'd still be interesting to train (and continuously retrain) a machine learning model in conjunction with the reinforcement learning approach in an attempt to not only verify the optimal policy of the reinforcement learning problem, but also potentially take an "ensemble" approach to determining the dAMM's overall optimal policy.

4 References

1. Drift Protocol dAMM deep dive: `https://www.notion.so/Drift-dAMM-deep-dive-ff154003aedb4efa83d6e7f4440c`
2. Drift Flat Data: `https://github.com/0xbigz/drift-flat-data/blob/main/data/curve_history.csv`
3. Perpetual Protocol vAMM guide: `https://medium.com/perpetual-protocol/a-deep-dive-into-our-virtual-amm`
4. Paradigm Intro to AMMs: `https://www.paradigm.xyz/2021/04/understanding-automated-market-makers-part-`
5. Paradigm Report on Uniswap v3: `https://www.paradigm.xyz/2021/06/uniswap-v3-the-universal-amm/`
6. Applications of Reinforcement Learning in Automated Market Making: `http://www.agent-games-2019.preflib.org/wp-content/uploads/2019/05/GAIW2019_paper_5.pdf`
7. Deep Reinforcement Learning for Trading: `https://arxiv.org/pdf/1911.10107.pdf`