AUTOMATED MARKET MAKING ON A LIMIT ORDER BOOK

Chris Slaughter
LVL
Los Angeles, CA
chris@lvl.co

December 20, 2020

ABSTRACT

In this white paper we discuss the background and design decisions of our new AMM v2 "Yield Farming" algorithm. We provide a quick review of constant function pricing models (CFPMs) with a focus on constant product pricing. We then propose a new automted market making algorithm. Our algorithm extends constant product pricing to offer price protection and operate on limit order books. We analyze the algorithm and share the results of simulations.

***K*eywords** Exchange · Liquidity · Automated Market Making

1 Background

LVL is a US-based financial institution and mobile app that offers free cryptocurrency trading and other consumer financial services. LVL operates its own centralized exchange and order book. LVL does not route order flow to institutional market makers or allow them to trade on LVL. To ensure the availability of liquidity on LVL's spot market, we developed and deployed automated market making on LVL in 2019 [1]. Automated market making is available to all LVL Premium members as part of the Autopilot feature of the mobile app. LVL does not participate in automated market making on its own spot market. The automated market making algorithm, which we refer to in this paper as AMM v1, implements the market making strategy of Avellaneda and Stoikov [2], the optimal strategy for market making on limit order books under the standard brownian motion model of market price. AMM v1 modifies this algorithm in several ways to simplify user configuration and manage asset inventory. Since March 2019, Autopilot accounts running AMM v1 have provided liquidity for millions of dollars of market order flow while earning a potential income for users. While AMM v1 has demonstrated efficient pricing to market, it has experienced numerous instances where inventory has been either sold or bought out on a pair. When inventory is sold or bought out, buy and sell market orders can no longer be executed. We refer to this as the inventory management problem, and it is the central motivation for AMM v2.

Since we introduced automated market making on LVL, there has been an explosion of interest in automated market making in decentralized finance, or DeFi. DeFi is a system of censorship-resistant, decentralized financial primitives including lending [3], stable assets [4], and exchanges [5]. DeFi exchanges provide a mechanism of exchange of digital assets between interested parties similar to exchanges found in traditional finance. These digital assets can include online ad units, prediction market bets, or cryptocurrencies. Because digital assets are an emerging class of assets, DeFi exchanges have historically suffered from low liquidity, which has led to the introduction of automated market makers to "bootsrap" liquidity in nascent digital asset markets [6]. Automated market makers allow market participants to passively allocate their digital assets to the provision of liquidity on an exchange. In return, these participants earn a potential yield on their assets through either trading commissions or a spread on the marginal exchange price [7]. Automated market making algorithms use a *scoring rule* to map liquidity pool reserves and/or a reference market price provided by an oracle, to a marginal asset price. A popular scoring rule for prediction markets is Hanson's *Logarithmic Market Scoring Rule* (LMSR) [8]. Our AMM v1 market making algorithm uses a scoring rule based on stochastic optimal control to price limit orders on a spot market. Bancor introduced a scoring rule based on bond curves [9]. The most well known scoring rule for DeFi exchanges is the *constant product pricing model*, first introduced in the Uniswap Protocol. Constant product pricing is part of a broader family of *constant function pricing models* (CFPMs) which

require any trade to modify reserves in such a way that some function of the reserves is always equal to a constant k. In CPPM exchanges, this function is $xy = k$. CPPMs appear to work exceptionally well at solving the inventory management problem despite their simplicity [10]. CPPMs have been demonstrated to closely track the reference price under common conditions.

In this white paper we discuss the background and design decisions of our new AMM v2 "Yield Farming" algorithm. We provide a quick review of constant function pricing models (CFPMs) with a focus on constant product pricing. We then propose a new automted market making algorithm. Our algorithm extends constant product pricing to offer price protection and operate on limit order books. We analyze the algorithm and share the results of simulations.

2 Constant Function Pricing Models

This section explains the motivation and mathematical formulation of several well known CFPMs. Recall that CFPMs require that the asset reserves, represented in their notional value of a single currency (typically USD), $R_1, R_2, \ldots, R_n \in \mathbb{R}^+$ satisfy at all times the relation $\varphi(R_1, R_2, \ldots, R_n) = k$. Here, k is the constant and φ is the *constant function* (not to be confused with the constant map $f(x) = a$). The set of all valid reserve alloctions $(R_1, R_2, \ldots, R_n)$, the preimage of k under φ.

2.1 Constant Product Pricing Model

The constant product pricing model, introduced by Uniswap [5], uses the constant function $xy = k$. We denote the constant function as $\varphi_\times(R_1, R_2)$, and valid reserve allocations satisfy:

$$\varphi_\times(R_1, R_2) = R_1 R_2 = k \tag{1}$$

An easy way to visualize the set of valid asset allocations (R_1, R_2) is to plot them as a curve on a graph where the x and y axes represent the value of reserves R_1 and R_2 respectively. In this case, the curve is $R_2(R_1) = k/R_1$, the familiar inverse linear function $y = a/x$. Because any trade must result in another point on this curve, it is useful to charactize which trades of a given asset allocation (R_1, R_2) result in a new asset allocation (R_1', R_2') that satisfies the constant function $\varphi_\times(R_1', R_2') = \varphi_\times(R_1, R_2) = k$. We can characterize a trade as a pair of fills (Δ_1, Δ_2). W.l.o.g., we can assume that the market order being executed is trading asset Δ_1 for asset Δ_2. Execution of the trade (Δ_1, Δ_2) results in a new asset allocation $(R_1, R_2) \to (R_1 + \Delta_1, R_2 - \Delta_2)$. For this asset allocation to be a valid, it must satisfy $\varphi_\times(R_1 + \Delta_1, R_2 - \Delta_2) = k$. Through manipulation, we can see that Δ_1 can be expressed as a function of (Δ_2, R_1, R_2):

$$\Delta_1 = \frac{R_1 \Delta_2}{R_2 - \Delta_2} \tag{2}$$

Δ_1 is the amount of a first asset that must be *tendered* to purchase Δ_2 units of another asset. We can see that as Δ_2 approaches R_2, the total amount of Δ_1 that must be tendered tends towards infinity. As a result, it is impossible for either R_1 or R_2 to sell out, provided that $R_1, R_2 > 0$ at the outset. We can also consider the marginal price for a market order to purchase asset Δ_2, and for the liquidity pool to *sell*, Δ_2, $P_{\text{sell}}(\Delta_2) = \Delta_1/\Delta_2$, which we can see from the above equation is:

$$P_{\text{sell}}(\Delta_2) = \frac{R_1}{R_2 - \Delta_2} \tag{3}$$

As a buyer attempts to purchase more Δ_2 from the liquidity pool, both the price and the amount of Δ_1 that must be tendered become unbounded. We can also see that the marginal cost is R_1/R_2. It will be helpful later to derive $P_{\text{buy}}(\Delta_1)$, addressing the case in which the liquidity pool is *buying*. By solving $\varphi_\times(R_1 - \Delta_1, R_2 + \Delta_2) = k$, we see that:

$$P_{\text{buy}}(\Delta_1) = \frac{R_1 - \Delta_1}{R_2} \tag{4}$$

2.2 Constant Sum Pricing Model

A more elementary constant function is the addition function, which we denote $\varphi_+(R_1, R_2)$. This model uses the sum $x + y = k$, and valid reserve alloctions satisfy:

$$\varphi_+(R_1, R_2) = R_1 + R_2 \tag{5}$$

Solving the invariant $\varphi_+(R_1, R_2) = \varphi_+(R_1 + \Delta_1, R_2 + \Delta_2)$ leads to the simple solution that $\Delta_1 = \Delta_2$ under the constant pricing model. As a result, the marginal cost Δ_1/Δ_2 is always equal to one. The constant sum model can also run out of liquidity in two ways, $(R_1, R_2) \to (R_1 + R_2, 0)$ and its opposite.

3 Automated Market Making

In this section, we describe a new market making algorithm for decentralized liquidity providers on LVL: AMM v2. This algorithm succeeds AMM v1, which produced high returns for users but was prone to liquidity outages. The primary objective of AMM v2 is to mitigate these liquidity outages by implementing ideas from constant function pricing models. The design objectives for AMM v2 are:

- Operate on a digital asset-fiat limit order book.
- Provide liquidity priced to market according to a pricing oracle.
- Incentivize a 50/50 allocation between base and quote assets at all times.
- Implement a minimum spread, or marginal profit margin, for liquidity providing.
- Guarantee price protection for takers withing a fixed percentage of the oracle price.
- Provision a liquidity pool acroos multiple, semi-autonomous liquidity providers.

These objectives reflect a number of challenges inherent to adapting constant function from decentralized liquidity pools to a centralized exchange. While DeFi marketplaces exchange between two tokens, pricing the reserves of each token according to a reference asset such as the US dollar, our platform exchanges digital assets and dollars, pricing both in dollars. Moreover, permissionless DeFi liquidity pools operate on a "taker" model in which the liquidity pool prices assets according to the market order size. Our marketplace operates a limit order book, which requires liquidity providers to post prospective prices as limit orders rather than price to taker demand instantaneously. Finally, US regulations present challenges to pooling user assets together, requiring a "pool of pools" wherein each liquidity provider participating in the liquidity pool manages inventory for their own account. We explore solutions to these problems in this section, culminating in the formulation of AMM v2.

3.1 Constant Function Pricing on Fiat Pairs

On a fiat pair, such as a BTC-USD pair, orders are sized in the base currency and quoted in the quote fiat currency. Reserves of both base and quote are priced in fiat. In this case, $R_1 = D$, the amount of dollars held in reserve. And $R_2 = \tilde{P}b$, where $\tilde{P}$ is the oracle price, b is the amount of base currency in reserve, and $\tilde{P}b$ is the value of base asset priced in dollars. For the remainder of this paper, we consider reserve allocations of (D, b) to differentiate fiat pairs from pairs of arbitrary tokens. Trades are characterized as (Δ_D, δ_b), the fill sizes of quote (dollars) and base (Bitcoin), respectively. The price, Δ_D/δ_B represents the cost in quote D per unit δ_B. Note that the amount of base δ_B is *not priced to notional*.

In the previous section, we explored the prices determined by scoring rules that, w.l.o.g., fix the constant function $\varphi(R_1, R_2) = \varphi(R_1 + \Delta_1, R_2 - \Delta_2) = k$. By the change of variables, this rule becomes:

$$\varphi(D, \tilde{P}b) = \varphi(D + \Delta_D, \tilde{P}(b - \delta_b)) = k \tag{6}$$

Recall that solving these equations for the constant sum pricing model yielded the constraint $\Delta_1 = \Delta_2$, requiring the *unit price of exchange* to be $\Delta_1/\Delta_2 = 1$ regrdless of the reserve allocations. Solving the constant function equation for the fiat case yields the constraint $\Delta_D = \tilde{P}\delta_b$ and the resulting *price of exchange* $\Delta_D/\delta_b = \tilde{P}$. It makes intuitive sense that to fix the combined notional value of dollars D and Bitcoin b, Bitcoin and dollars must be exchanged at exactly the notional price.

Under the constant product pricing model, the constant function $\varphi_\times(D, \tilde{P}b)$ applied to the trade (Δ_D, δ_B) is the equation:

$$D\tilde{P}b = (D + \Delta_D)\tilde{P}(b - \delta_b) \tag{7}$$

Solving these equations yields the following familiar pricing functions:

$$P_{\text{buy}}(\Delta_D) = \frac{D - \Delta_D}{b},\ P_{\text{sell}}(\delta_b) = \frac{D}{b - \delta_b} \tag{8}$$

We can see that the marginal price Δ_D/δ_b as $\Delta_D, \delta_b \to 0$ is D/b. Interestingly, the marginal price does not rely on $\tilde{P}$ in any way despite the fact that the constant product is between the *notional values* of D and b. However, the notional value of these reserves are exactly equal when $b = D/\tilde{P}$, and as a result $\tilde{P}$ is the marginal price when the reserves are at equilibrium. As in DeFi liquidity pools, deviation in the marginal price from $\tilde{P}$ creates an incentive for arbitrageurs to balance the notional reserves of D and b.

3.2 Pricing on a Limit Order Book

DeFi liquidity pools are often referred to as decentralized exchanges, but in reality they function more like token swaps. The liquidity pool holds reserves (R_1, R_2). Takers offer Δ_1 in return for Δ_2. As the liquidity pool reserve R_2 is depleted, the pool tenders smaller and smaller amounts of Δ_2 for offers of Δ_1. This results in a marginal exchange rate Δ_1/Δ_2 that approaches infinity as the reserve is depleted. The liquidity pool behaves exactly the same way in the converse case, charging higher marginal rates to exchange Δ_2 for Δ_1 as R_1 depletes. Furthermore, DeFi pools operate on a taker model. The amount of one asset to tender in exchange for a taker's offer of another asset is determined *at the time the taker makes an offer*. While this yields a marginal price of exchange, that price does not need to be determined until the taker offer is placed.

This taker model presents a challenge for adapting constant function pricing to limit order books. Limit order books operate on a *maker-taker model* where market makers must price and post offers that are then matched to taker orders in price-time priority. If we assume that any taker order matches to the entirety of an offer posted to the order book or not at all, and that only one maker offer is posted at a time, it is clear that the pricing rules P_{buy} and P_{sell} result in trades that fix the constant function $\varphi_\times$. We extend this to the multi-agent setting through greedy pricing. That is, a market maker which is aware of (D, b) for the total pool, and which has Δ_D available to post a buy order, will price according to $P_{\text{buy}}(\delta_b; D, b)$ with no consideration paid to the behavior of other market makers. We also incorporate a minimum spread α which ensures a fee is paid to the market maker.

As a regulated financial institution, we believe it is unreasonable to allow market makers to post prices which deviate excessively from the prevailing price $\tilde{P}$, regardless of the depletion of reserves. Reasonable deviations of the marginal price from the prevailing price still incentivize arbitrageurs to balance market makers' reserves. Furthermore, those deviations in marginal price drive an increase in total reserves which ultimately improves marketplace health by preventing depletions. Accordingly, we implement price protection beyond a deviation γ as follows:

$$\begin{aligned} P_{\text{buy}}(\Delta_D; D, b) &= \max\left\{\frac{D - \Delta_D}{b} - \alpha\tilde{P},\ (1-\gamma)\tilde{P}\right\} \\ P_{\text{sell}}(\delta_b; D, b) &= \min\left\{\frac{D}{b - \delta_b} + \alpha\tilde{P},\ (1+\gamma)\tilde{P}\right\} \end{aligned} \tag{9}$$

Note that $\delta_b = b \Rightarrow P_{\text{sell}} = (1+\gamma)\tilde{P}$, intuitively and since $b - \delta_b$ approaches 0 from the right.

References

[1] Chris Slaughter and Brandon Eng. Automated Market Making. *White Paper*, Samsa Technologies Inc. dba LVL, 2019. `https://github.com/chriscslaughter/amm/blob/master/paper/AMM.pdf`

[2] Marco Avellaneda and Sasha Stoikov. High-frequency trading in a limit order book. In *Quantitative Finance, Vol. 8, No. 3*, pages 217–224. Routledge, 2008.

[3] R. Leshner and G. Hayes. Compound: The money market protocol *Technical Report*, Compound Labs, 2019. `https://compound.finance/documents/Compound.Whitepaper.pdf`

[4] The Maker Protocol: MakerDAO's Multi-Collateral Dai (MCD) System *White Paper*, MakerDAO, 2015-2020. https://makerdao.com/en/whitepaper

[5] Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 Core *White Paper*, Uniswap, 2020. https://uniswap.org/whitepaper.pdf

[6] A. Othman, D. M. Pennock, D. M. Reeves, and T. Sandholm. A practical liquidity- sensitive automated market maker. In *ACM Transactions on Economics and Computation (TEAC), vol. 1, no. 3*, pp. 1–25, 2013.

[7] Nikolai Kuznetsov. DeFi yield farming, explained. *Website Article*, CoinDesk, September 2020, accessed December 2020. https://cointelegraph.com/explained/defi-yield-farming-explained

[8] R. Hanson. Combinatorial information market design. In *Information Systems Frontiers, vol. 5, no. 1*, pp. 107–119, 2003.

[9] Eyal Hertzog, Guy Benartzi, Galia Benartzi. Continuous Liquidity for Cryptographic Tokens through their Smart Contracts. *White Paper*, Bancor Protocol, 2018. https://storage.googleapis.com/website-bancor/2018/04/01ba8253\protect\discretionary{\char\hyphenchar\font}{}{}bancor_protocol_whitepaper_en.pdf

[10] Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. An analysis of Uniswap markets. *ePrint*, arXiv:1911.03380. https://arxiv.org/abs/1911.03380