Liquidity Options - a scalable solution for liquidity pooling

Nathan Lou

April 3, 2022

Abstract

CryptoSwap introduces the concepts of **balancing liquidity** and **time-locked liquidity** which provide an efficient and scalable solution to decentralized exchanges. When bounded liquidity and time-locked liquidity are paired, they're able to replicate the functionality of stock options, which we'll call **liquidity options**. This paper gives a brief overview of some of the design decisions behind the CryptoSwap protocol, go over how balancing and time-locked liquidity work, and finally, liquidity options.

1 Introduction

Building a decentralized exchange is a lot like creating an ecosystem. A flourishing ecosystem is one that is **sustainable** and **scalable** - that is, it has the ability to maintain its structure (**security** and **efficiency**) and improve (**expansion**) over time in the face of external stress (**reliability**).

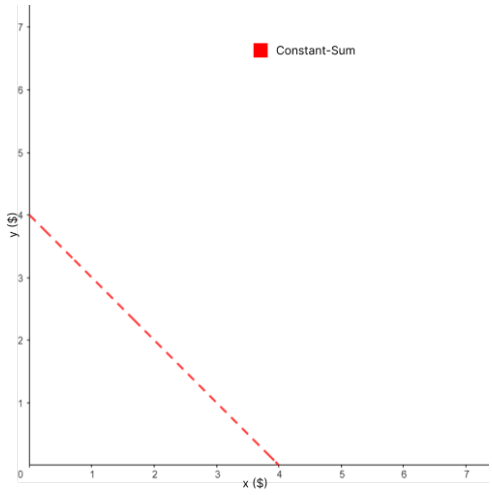
I firmly believe that Decentralized Finance (DeFi) will be the future of global finance. However, the systems that we currently have in place are not scalable nor do they build upon each other. Systems of the present were built on systems of the past, but in the DeFi industry, systems of the past (previous versions) have to be completely be scrapped in order to bring about a new version. Not only is this method unfit for scalability, but it also discourages innovation as it sends the idea that innovation is building an entirely new system rather than improving upon a previously existing system. What the CryptoSwap protocol will do is establish a method of innovation that builds upon the next, allowing for innovative ideas to build upon one another.

CryptoSwap has a few defining characteristics that set it apart from other decentralized exchanges:

1. Balancing Liquidity
2. Time-Locked Liquidity
3. Liquidity Options

2 How it works

Liquidity pools are initialized by depositing a liquidity pair, which is a price-determined equally-weighted pairing of two assets X and Y. Suppose that a liquidity provider has a constant price $k \in \mathbb{R}$, thus for two tokens X and Y, the price graph would look something like this:



So swapping dx coins of X for Y would result in the user receiving $-dy = dx$ of Y. The price is determined by $-dy/dx$ which in this case is always exactly 1. This function in particular is known as the constant-sum function $x + y = k$ and can be generalized for any number of coins X_i resulting in a function:

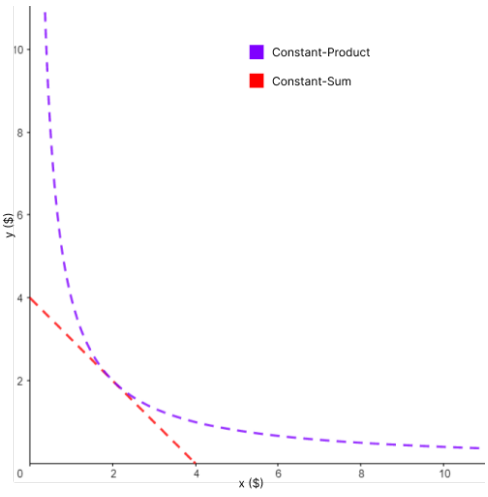
$$\sum x_i = k$$

However, in the reality, price ratios between two assets $-dx_i/dx_j$ may not always be 1, we can probably only expect this of wrapped assets or stablecoins. But even for those assets we cannot always expect them to always retain a 1-to-1 ratio. CryptoSwap wants to prioritize retaining efficient functionality in the **worst case scenario** so we certainly can do better.

Most decentralized exchanges, notably Uniswap [1] use variants of the constant-product function:

$$xy = k$$

The standard constant-product function is a great base to start from because it is zero-assumption in nature. Unlike further optimizations that will be shown later on, the constant product function on relies on the fact that liquidity pools are balanced in the beginning, which is a given since liquidity pairs are essentially forced to balance the liquidity pool. From then on it will "rebalance" itself. The constant-product function is a great base to start from because it is the most efficient constant function market maker under no assumptions. The graph would look something like this:



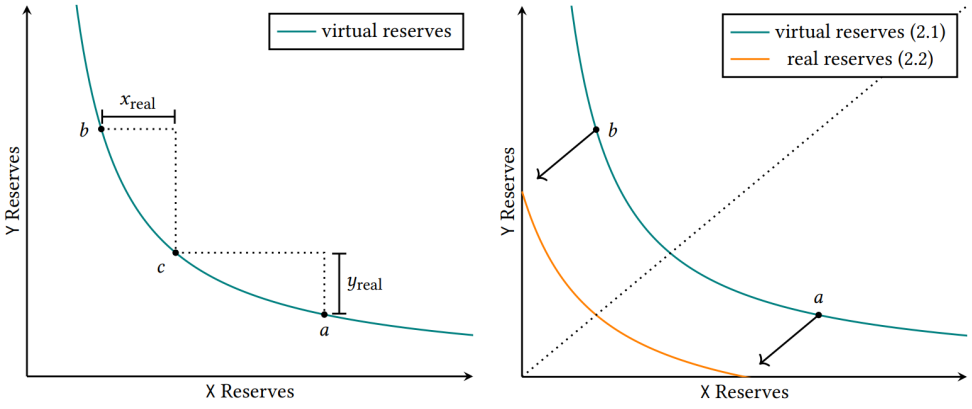
Although there is nothing bad about arbitrageurs, in fact they are necessary for any market to be efficient, we can't rely on them during volatile times and arbitrageur profits can take away from liquidity provider profits. Thus it is crucial for the protocol's design to minimize arbitrageur profits.

Although it's predecessors utilized the standard constant-product formula, Uniswap v3 performs some notable optimizations. By introducing the idea of *concentrated* liquidity, i.e liquidity bounded within some price range $[p_a, p_b]$. Uniswap v3 is further able to replicate dynamic market making by using liquidity providers as the "dynamic" variable. This is great because it allows liquidity providers to be able to compete against arbitrageurs. After all, to get rid of arbitrage, we must arbitrage first.

Under Uniswap v3's *concentrated liquidity* there are *virtual reserves* and *real reserves*. Virtual reserves is liquidity concentrated to a smaller price range while real reserves is general liquidity in the standard constant product. Let $[p_a, p_b]$ be the bounds for the virtual reserves then for $L = \sqrt{k}$, the real reserve positions can be described by the curve:

$$(x + \frac{L}{\sqrt{p_b}})(x + L\sqrt{p_a}) = L^2$$

The graphs below is a side by side comparison of the relationship between real and virtual reserves (source: Uniswap v3 whitepaper):



2.1 Problems With Current Models

Although protocols Uniswap v3 have made quite a few optimizations to the constant product function that make pricing more efficient, a major problem is that these models can sometimes break. The problem is not that these models can break when their assumptions fall apart, this is the real world after all. Instead, the problem is that these smart contracts may be further optimized or changed in the future to prevent them from falling apart. Why is this a problem?

Most decentralized exchanges have numerous version updates that force liquidity providers to migrate from their liquidity pool. Not only does this defeat the purpose of "passive investing" but is also extremely inefficient at scale. Hypothetically, let's say there is a global adoption of decentralized exchanges. Imagine how long it would take to migrate a liquidity pool of 1 trillion dollars, not to mention the gas fees of that process.

For decentralized exchanges to see global adoption and efficiency at scale. It is imperative that we find a solution that is flexible in assumptions, infinitely optimizable, and does not force a change in the main smart contract.

3 Balancing Liquidity

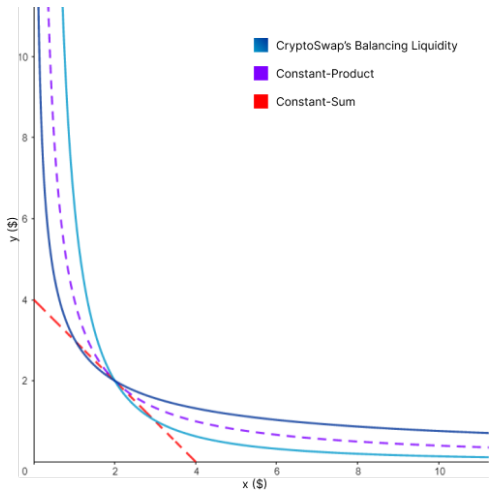
The core concept that the CryptoSwap protocol introduces is the *balancing liquidity function* - a function that "balances" the core liquidity pool. For dollar denominated value of assets x, y with we define the *balancing liquidity function* as

$$x^{w_1} y^{w_2} = k$$

for any $\{w_1, w_2 \in \mathbb{R}_{>0} \mid w_1 + w_2 = 2\}$. Similarly, we define a *symmetric balancing liquidity function as*

$$x^{w_2} y^{w_1} = k$$

We use the balancing liquidity function and it's symmetrical balancing liquidity function to effectively "bound" the main liquidity pool $xy = k$.



3.1 Perfectly Balanced Pools Are Constant Product Proof

The graph above is an example of a set of *perfectly balanced symmetric liquidity pools*. We want to show that under the constant product function assumptions (perfectly balanced), symmetric balancing liquidity pools replicate constant product functions. That is, we assume that $x_1 = x2$, $y_1 = y_2$, and $k_1 = k_2$.

Suppose that for symmetric liquidity pools ${x_1}^{w_1}{y_1}^{w_2} = k_1$ and ${x_2}^{w_2}{y_2}^{w_1} = k_2$. $x_1 = x_2$, $y_1 = y_2$, and $k_1 = k_2$. Then $x^{w_1}y^{w_2} = k$ and $x^{w_2}y^{w_1} = k$. Now, if we take the geometric average of these two pools, we get:

$$(x^{w_1}y^{w_2})(x^{w_2}y^{w_1}) = k^2$$

$$x^{w_1+w_2}y^{w_1+w_2} = k^2$$

$$x^2y^2 = k^2$$

$$\sqrt{(xy)^2} = k$$

$$xy = k$$

This means that under a perfect market conditions, i.e a scenario where the spot price matches the true price of a given set of assets, symmetric liquidity pools will be no different from the constant function market maker, and the constant function market maker will reflect the true price.

3.2 Equivalence At Equilibrium Proof

By introducing *balancing liquidity* we have effectively come up with a method to replicate constant product market makers while also allowing for the ability to arbitrage on both sides of an AMM trade. However, what if a scenario pops up where $x_{\mathrm{o}}y_{\mathrm{o}} = k_{\mathrm{o}}$, ${x_{\mathrm{i}}}^{w_1}{y_{\mathrm{i}}}^{w_2} = k_{\mathrm{i}}$, $x_{\mathrm{i}} \neq x_{\mathrm{o}}$, $y_{\mathrm{i}} \neq y_{\mathrm{i}}$, and $k_{\mathrm{i}} \neq k_{\mathrm{o}}$? Arbitraguers on the liquidity provider's side will always be able turn $x_{\mathrm{i}} = x_{\mathrm{o}}$ and $y_{\mathrm{i}} = y_{\mathrm{o}}$ by depositing or removing liquidity to the balancing liquidity pool. This implies that under the state when $x = y$, $k_{\mathrm{i}} = k_{\mathrm{o}}$.

3.3 Asset Pricing Model Part 1

In this section we will go over:

1. How the constant function market maker determines theoretical price.
2. How the balancing liquidity function affects this price.

The theoretical price p at time t can be computed by dividing the reserves of asset a and reserves of asset b.

$$p_t = \frac{r_t^a}{r_t^b}$$

Under this model, there is price slippage, especially for large movements of assets. To visualize this slippage mathematically, calculate the realized price. Suppose we sell Δx for Δy at the price ratio above including a $1-\phi$ trading fee for liquidity providers.

$$(x-\Delta x)(y+\phi\Delta y)=k$$
$$\phi\Delta y=\frac{xy}{x-\Delta x}-y$$
$$\phi\Delta y=\frac{xy-y(x-\Delta x)}{x-\Delta x}$$
$$\phi\Delta y=\frac{y\Delta x}{x-\Delta x}$$
$$\Delta y=\frac{1}{\phi}\cdot\frac{y\Delta x}{x-\Delta x}$$

Since price is the ratio of Δy received and Δx paid. Divide both sides by Δx getting:

$$\frac{\Delta y}{\Delta x}=\frac{1}{\phi}\cdot\frac{y}{x-\Delta x}$$

Suppose that there was no trading fee, so $\phi=1$. We see that as amounts traded get smaller, i.e $\Delta x\to 0$, the realized price tends to the theoretical price.

$$p_t=\lim_{\Delta x\to 0}(\frac{\Delta y}{\Delta x})=\lim_{\Delta x\to 0}(\frac{y}{x-\Delta x})=\frac{r_a}{r_b}$$

So let's go on to how the balancing liquidity function can not only reduce price slippage, but also help obtain a price that more closely matches the market (true) price. Let's first define a *balancing price function* as the curved obtained by the geometric mean between symmetric balancing price pools a and b

$$\sqrt{(x_a^{w_1}y_a^{w2})(x_b^{w_2}y_b^{w1})}=\sqrt{k_ak_b}$$

Let $\{P_0,P_1P_2,...,P_{\mathrm{i}},...,P_{\mathrm{n}}\}$ be the total reserves of the liquidity pools where P_i represents the i^{th} pair of balancing price liquidity pools and P_0 is the base liquidity pool with the function $xy=k$. Moreover, let $\{w,w_1,w_2,...,w_{\mathrm{i}},...,w_{\mathrm{n}}\}$ represent the "weight" of each respective liquidity pool. So for each for each balancing liquidity pool P_j where $0\leq i,j\leq n$, w_j is the arithmetic weight.

$$w_j=\frac{P_j}{\sum_{i=1}^{n}P_i}$$

We will use these weights to find an alternative method to price assets but we first need to prove that under equilibrium conditions, i.e x=y, the balancing price function will always be equivalent to the constant function market maker to ensure that we have a point in which this system wont fail and will tend to revert to.

3.4 Trading at Equilibrium State

Let $x_a^{w_1} y_a^{w_2} = k_\text{a}$ and $x_b^{w_2} y_b^{w_1} = k_\text{b}$ be symmetric liquidity pools and $xy = k$ be the main liquidity pool.

For balancing price function $\sqrt{(x_a^{w_1} y_a^{2_w})(x_b^{w_2} y_b^{w_1})} = \sqrt{k_a k_b}$, we will show (note that I'm too dumb to prove)* that at point $x = y$, $\sqrt{(x_a^{w_1} y_a^{w_2})(x_b^{w_2} y_b^{w_1})} = xy$ [Statement 1].

Observe that for $x_a = x + \Delta_a$ and $y_a = y + \Delta_a$ note that -Δ_a cannot exceed x or y since $x_a, y_a > 0$

So going into any graphing app like Desmos, for xy = k, k is a constant so we can type something like $xy = 4$. Additionally, for arbitrary constants $\Delta_a, \Delta_b > -2$ compute $k_a = (2 + \Delta_a)^2$ and $k_b = (2 + \Delta_b)^2$ since we assumed $x = y$. Now, graph $\sqrt{(x + \Delta_a)^{w_1}(y + \Delta_a)^{w_2}(x + \Delta_b)^{w_2}(y + \Delta_b)^{w_1}} = \sqrt{k_a k_b}$ this is the *balancing price function* and it's clear that at every point within the constraint, [Statement 1] is satisfied. Feel free to try the same example with a different k, just remember that $k_a = (\sqrt{k} + \Delta_a)^2$ and same with k_b.

Some ways that may be able to prove this are:

1. Showing that a linear transformation that turns the *balancing price function* to a constant product function exists.
2. Some sort of real analysis at x = y
3. Brute force algebraically

Thus, we have shown that that *every* balancing price function within the constraint will have an equilibrium state equal to that of the constant product market maker. And since the function will always tend towards this equilibrium state by design (similar to that of the assumptions of a CFMM), the *balancing price function* will always *tend* to trade towards being perfectly balanced.

3.5 Asset Pricing Model Part 2

For each *balancing price pool*, i, and asset reserves at each pool r_i^a and r_i^b, define the theoretical price as

$$p_i = \frac{r_i^a}{r_i^b}$$

Define the *balancing price* p_b as

$$p_b = \prod p_i^{w_j}$$

3.6 Price Effect

In the real world, the balancing price may more closely reflect the true price. However, in the worst case scenario, this may not be true. So we won't let this price affect the CFMM quoted price. In the next section, we will see how **liquidity options** will use the balancing price to "regulate" the main liquidity pool.

4 Liquidity Options

Balancing liquidity can be used to manipulate prices in the short term if not time locked (flash liquidity). Therefore, we must time-lock balancing liquidity to allow time for arbitrageurs on the liquidity providers' side to correct attempted manipulations. To put it simply, liquidity options are balancing liquidity but with a time-lock and a few other rules.

4.1 Time-Locked Liquidity

4.2 Time Value of Liquidity

4.3 Liquidity Options

5 Liquidity Options Architecture and Implementation

References

[1] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer and Dan Robinson. 2020. *Uniswap v3 Core*. Retrieved Feb 02, 2022 from https://uniswap.org/whitepaper-v3.pdf.