Eliminating impermanent loss by leveraged liquidity

Michael Egorov
(Dated: 2 March 2025)

Ever since the Automatic Market Makers (AMMs) were introduced in Decentralized Finance (DeFi), they have suffered from "impermanent loss" (IL, or sometimes called LVR - loss vs rebalance), meaning AMMs often perform as a worse store of value than simply holding the components of liquidity idle. In this work, I propose a method to eliminate IL and make the position priced similarly to an individual component of liquidity while earning exchange fees. The simulations show, for example, that BTC/USD liquidity can make around 20% APR (average over 6 years) fundamentally while being priced similarly to BTC. Of course, the same method is applicable to other cryptocurrencies.

GENERAL IDEA

In Curve Cryptoswap AMM, price of liquidity (excluding earned trading fees) is approximately calculated similarly to that in classic $xy = k$ invariant $p_{LP} = \sqrt{p}$, where p is price of the token y (for example, BTC) in terms of token x (for example, USD). This is where impermanent loss comes from: for example $\sqrt{p} < 1/2 + p/2$ for all $p \neq 1$ means that holding an asset with initial price of 1 and equal amount of USD would outperform always-rebalanced liquidity if trading fee is set to zero.

Now, let's consider *compounding* leverage L. Here we define compounding leverage as the number which defines the fraction of debt relative to collateral size. If one borrows against any token with a price p' to buy even more of that token so that the value of the loan d is *always* kept to be equal to $d = V_c\,(1 - 1/L)$, where V_c is value of collateral, the position will be leveraged with the compounding leverage L at all times, and price of the whole position p_* will be proportional to $(p')^L$.

Let's prove this formula. Imagine that value of collateral V_c changes by a small amount dV_c, and we split compounding leverage in two steps: growth of collateral price and changing the debt to keep with the d/V_c ratio. In equilibrium, if collateral amount is y and its price is p':

$$V_* = V_c - d = \frac{V_c}{L} = \frac{p'y}{L}. \tag{1}$$

For the first step, value of leverage position V_* would change:

$$V_* + \delta V_* = (V_c + \delta V_c) - V_c\,(1 - 1/L) = V_* + \delta V_c = V_* + y\delta p', \tag{2}$$

$$\frac{\delta V_*}{V_*} = \frac{y\delta p'}{p'y/L} = L\frac{\delta p'}{p'}. \tag{3}$$

When we adjust the debt to buy (or sell) some infinitely small amount of collateral, value V_* doesn't change, so this relation remains. The ratio $\delta V_*/V_*$ can also be replaced with relative LP token price $\delta p_*/p_*$.

Small change in the value p_* of the position with price p' leveraged with the compounding leverage L satisfied the relationship:

$$\frac{dp_*}{p_*} = L\frac{dp'}{p'}. \tag{4}$$

Integrating that gives:

$$\log p_* = L \log p' + \mathbf{const}. \tag{5}$$

Exponentiation of both sides gives:

$$p_* \propto (p')^L. \tag{6}$$

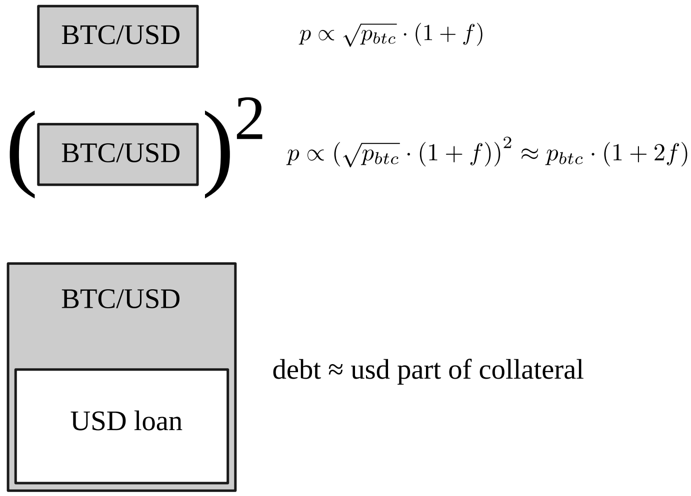


Figure 1: Schematic of leveraging liquidity to have no impermanent loss

Therefore, if $L = 2$ and $p_{LP} = \sqrt{p}$, leveraging liquidity would give $p_* \propto \left(\sqrt{p}\right)^2 = p$, simply price of the token y, while the position makes exchange fees in addition (Fig. 1). While the idea appears simple, it does not work with the $xy = k$ invariant: the losses from rebalancing to maintain constant leverage (i.e. *releverage losses*) are not lower than the pool's earnings. The situation is significantly improved with Curve Cryptoswap, as demonstrated in the simulations.

Another curious property is that for $L = 2$, size of the loan is equal to half of the size of liquidity leveraged, on average. But USD portion of that liquidity is also equal to half of its total value. Therefore, if leverage is kept constant and equal to 2, liquidity will on average have enough USD to close the position, which is a very convenient property.

APR of the resulting position can be expressed as:

$$APR = 2r_{pool} - \left(r_{borrow} + r_{loss}\right). \tag{7}$$

So this method only works when the rate pool (multiplied by leverage) is significantly higher than the total of borrow rate and losses introduced by releverage of the position. It's important to point out that this expression is only approximate, and a more precise APR is given by a combined simulation of cryptopool and releverage.

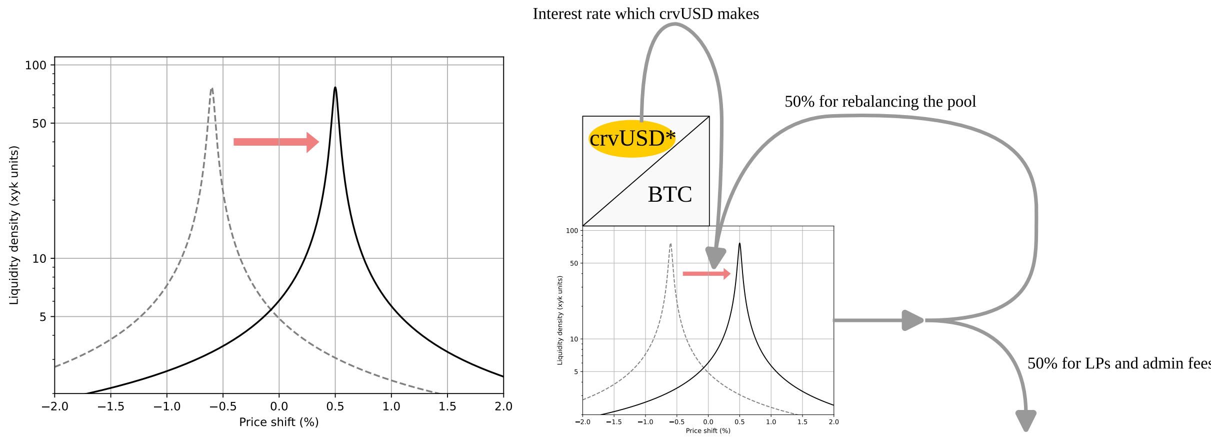
USING CDP INTEREST RATE FOR REBALANCING

In Curve, all liquidity pools have concentrated liquidity. However, when price of the asset is not constant, concentrated liquidity is moved towards current prices automatically (Fig. 2a).

SIMULATIONS AND POOL OPTIMIZATION

All simulations done in this paper are performed by simulating arbitrage between the AMM and external price feed (usually taken from Binance). In that, a model of pool AMM is being arbitraged with an infinitely deep "heatsink" which has a price changed between highs and lows of the candles of the feed. All the AMM volume is assumed to be only arbitrage between the current state of the AMM and the external prices. If, profit which arbitrage trader is making is smaller than some threshold value (typically 0.03%) - the trade is not happening. Also volumes stop flowing as soon as the arbitrage volume hits half of the exchange volume in the pair: that, however, plays only a small role in the simulations.

First, we simulate $xy = k$ bonding curve (e.g. like Uniswap2) for sanity check of the method (Fig. 3). The only adjustable parameter in this AMM is its fee. There is a clear maximum in dependency of APR over the AMM fee, but it is fair to say that over the simulated range (1 Jan 2023 - 1 Nov 2024) the APR of $xy = k$ AMM appears around 3%. This APR is on top of the value $xy = k$ bonding curve would have had if the AMM fee was 0 (e.g. on top of unremoved impermanent loss).



(a) Movement of concentrated liquidity towards current prices

(b) Subsidy for moving concentrated liquidity

Figure 2: Automatic management of concentrated liquidity

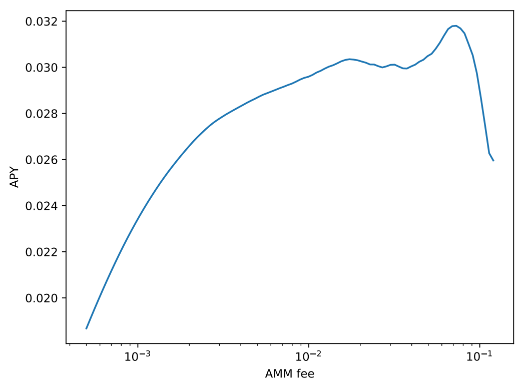


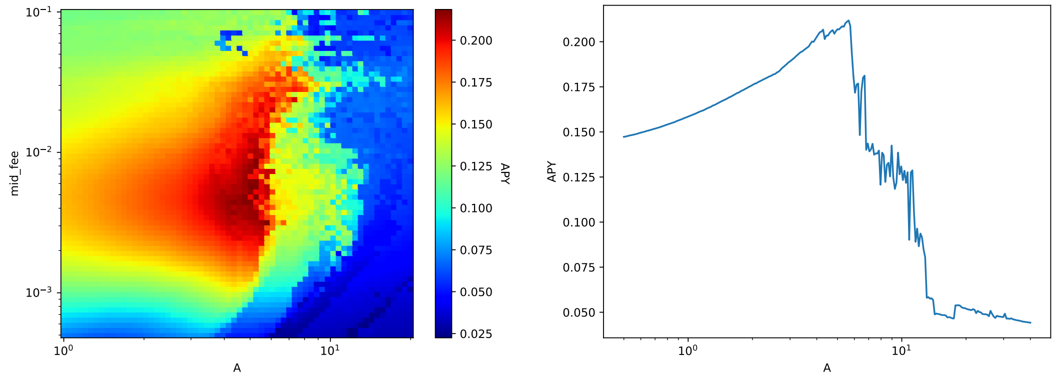
Figure 3: Reference study: optimizing $xy = k$ pool fee. Data over 2 years (Jan 2023 - Oct 2024) used

Simulation for cryptoswap is more complicated. I have found that replacing cryptoswap invariant with stableswap invariant while keeping the algorithm of the cryptoswap the same gives better results if subsidy with crvUSD interest rate is used while having less adjustable parameters. In this simulation we set borrow rate to 10% which is -5% APR spent on subsidizing liquidity rebalancing and continuously taken from LPs (while they earn fees). If we use stableswap invariant, the parameter optimization should be seeking for maximum APR in the parameter space which includes A (concentration parameter), f_{mid} (minimum AMM fee), f_{out} (maximum AMM fee) and γ_{fee} (fee steepness parameter). Ideally the borrow rate also should be optimized since we are in the full control of it, but this was not done in these simulations (will be done in the near future).

Initially, we assume the fee to be always constant to approximately find the good range for A and f_{mid}. In this simulation, we set f_{out} to be equal to f_{mid}, and that turns the dynamic fee off. The results of such a scan are shown on Fig. 4. One can notice that as concentrated liquidity becomes denser (e.g. A increases), the APR grows. However, around a certain threshold (around $A = 4$), the APR drops in a step-wise manner. At that point, volatility becomes too high for the rebalance algorithm to follow. When there's another place of high volatility on the price graph - we observe the second step forming. We should be choosing the highest value of A before the first discontinuity. It is also interesting to note that the optimal f_{mid} appears to be 0.3% - similar to typical Uniswap 2 fee.

In the next step, we optimize dynamic fee (Fig. 5). It appears that the f_{mid} found previously with non-dynamic fees ($f_{mid} = 0.3\%$) holds for dynamic fees as well, and A could be slightly higher than previously found. So we first set A and f_{mid} at the parameters found in the non-dynamic scan, and then scan (f_{out}, γ_{fee}). When we find optimal values - we can slightly vary A and f_{mid} to get the final adjustments and then rescan. Figure 5 shows the result. I also provide results of a similar optimization with no borrow rate donation feature for comparison.

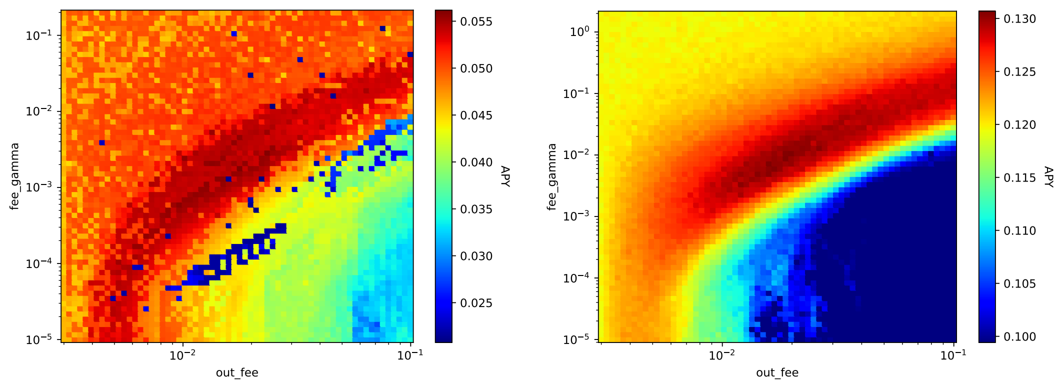
While releverage algorithm will be described further, I reference the full simulation of releverage here (Fig. 6). The



(a) Optimizing concentration parameter A and fee f_{mid} simultaneously

(b) Optimizing concentration parameter A at $f_{mid} = 0.003$ - slice of the 2D scan

Figure 4: Initial optimization of cryptopool: no dynamic fee applied. Data over 6 years (Jan 2019 - Oct 2024) used



(a) Optimization of unboosted cryptoswap pool: $\left(f_{out}, \gamma_{fee}\right)$optimization

(b) Optimization of boosted cryptoswap pool: $\left(f_{out}, \gamma_{fee}\right)$optimization

Figure 5: Final optimization step for standard "unboosted" and boosted cryptoswap pools. Data over 2 years (Jan 2023 - Oct 2024) used

releverage is made via a special AMM which is also simulated as arbitrage.

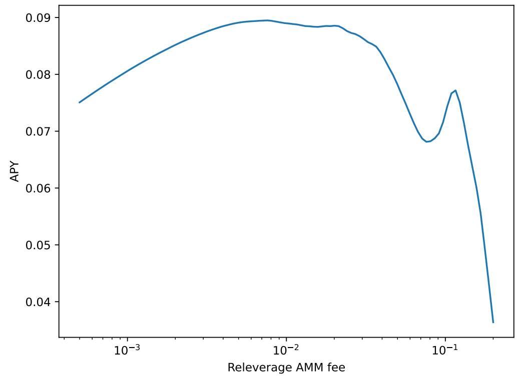
Noteworthy, simulation of releverage is separable from simulation of the base AMM (Fig. 7). It turns out that losses on releverage r_{loss} from Eq. 7 appears to be extremely close to twice the optimized returns of $xy = k$ pools (Fig. 3) which makes the APR to be not higher than 0 for $xy = k$ bonding curve. This result in itself is worth to study later. However, APR in Curve cryptopools is strictly higher than $xy = k$ returns when the parameters are optimal, which means that releverage should give a positive APR. We confirm that with direct simulations of the releverage AMM.

Simulation results from Fig. 6 show promising numbers for BTC/USD returns. The graphs show value of the resultant LP position in BTC over time. Notable features of the simulation results:

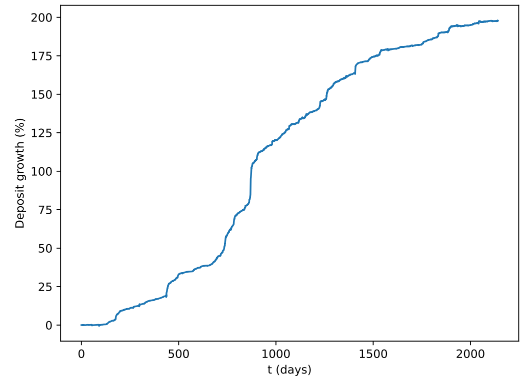
- Dependency on the releverage AMM fee is very flat (Fig. 6a);

- Average APR over 6 years is around 20% (peaking over year 2021 to 60%) (Fig. 6b), however being just under 10% in more calm (and bearish) market conditions (Fig. 6c);

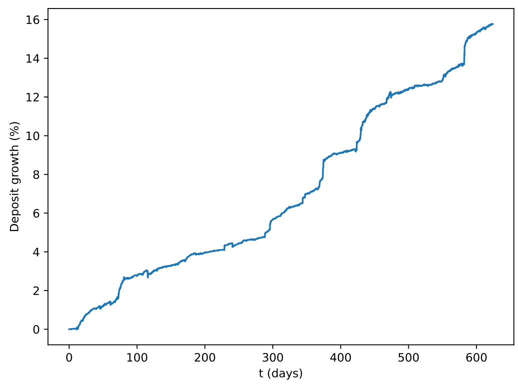
- If concentration A is set slightly higher than necessary - releverage algorithm experiences a loss at the point of the rebalance breaking down, however growth rate in calmer market conditions is higher, so the average APR over 6 years is the same (Fig. 6d). This example shows how to estimate volatility risks in the product.



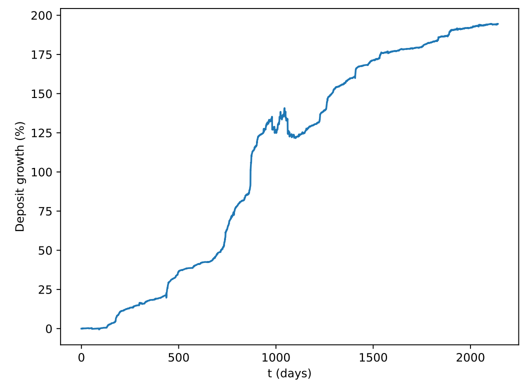
(a) Optimizing releverage AMM fee (optimal value: $f = 0.007$)



(b) Simulation from Jan 2019 till Oct 2024: average APR = 20%



(c) Simulation from Jan 2023 till Oct 2024 (tail of Fig. b): average APR = 9%



(d) Example of too aggressive parameters ($A = 4.8$ instead of $A = 4.13$). Partial loss is visible (in year 2021), however average APR is approximately the same

Figure 6: Simulation with releverage AMM removing impermanent loss. Data is obtained for BTC/USD price feed

RELEVERAGE ALGORITHM

Keeping leverage constant manually is not efficient, although can be done. Problem with manual approach is that the threshold price change at which it should happen is relatively high (10%) which makes variations in the returns very high (e.g. returns can go negative very often) (Fig. 7).

Instead, a special AMM is used for releveraging. The AMM uses an external oracle with price p_o for the asset which is being re-leveraged. The AMM keeps reserves of collateral y. Instead of keeping reserves of stablecoins, it borrows those having a debt d. When market price p is equal to p_o, in order to keep the leverage L, ideal debt should be equal to:

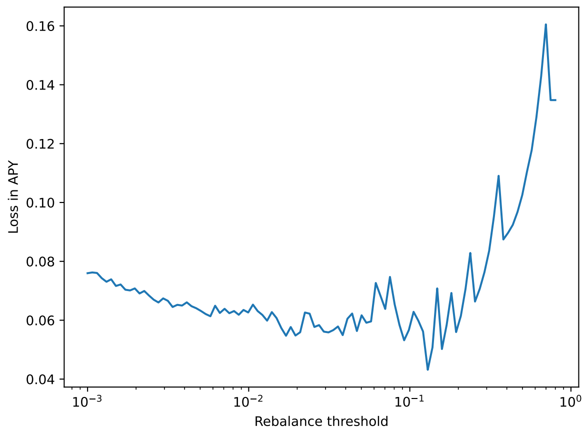
$$\widetilde{d} = \frac{L-1}{L} p_o \widetilde{y}, \tag{8}$$

where values with $\widetilde{}$ mean that they are taken at the time when market price is equal to the oracle price. For example, one can see that for $L = 2$ (our case) $\widetilde{d} = p_o y/2$, which matches the intuition of keeping constant leverage.

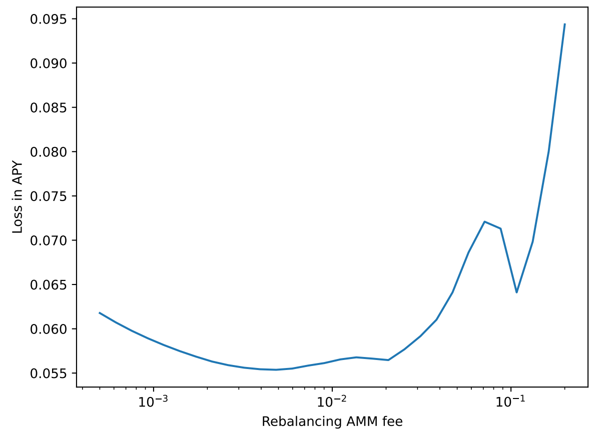
We keep leverage constant via a variant of $xy = k$ AMM with x being represented as a function of oracle price and debt:

$$\left(x_0(p_o) - d\right) y = I\left(p_o\right), \tag{9}$$

where invariant I is constant at the same p_o, $x \equiv x_0\left(p_o\right) - d$.



(a) Losses from manual releverage depending on a price change threshold for triggering it



(b) Losses from releverage via an AMM depending on the fee of the AMM

Figure 7: Comparison of manual releverage and releverage via a special AMM

In order to find $x_o\left(p_o\right)$ function, let's use the ideal values for $p = p_o$and property of $xy = k$ invariant: $p = x/y$. When we apply this to $p = p_o$:

$$\frac{x_0\left(p_o\right) - \widetilde{d}}{\widetilde{y}} = p_o, \tag{10}$$

and therefore, substituting Eq. 8:

$$x_0\left(p_o\right) = \frac{2L-1}{L} p_o \widetilde{y}. \tag{11}$$

As an example (which we will use later to choose the right solution), at $\widetilde{y} = 2$, $p_o = 1$, $L = 2$, $\widetilde{d} = 1$, we find $x_0 = 3$, and indeed, that satisfies $x_0 - \widetilde{d} = p_o \widetilde{y}$.

Now, let's find the function $x_0\left(p_o\right)$ for *any* current values of y and d (since y,d and p_o should fully define the state of the AMM). First, if we did know x_0 - we would be able to express "ideal" $\widetilde{y}$:

$$\widetilde{y} = \frac{L}{2L-1} \frac{x_0}{p_o}. \tag{12}$$

We also know that at constant p_o the value of invariant I is conserved and the same as at "ideal" parameters (e.g. $\widetilde{y}$, $\widetilde{d}$):

$$y\left(x_0 - d\right) = \widetilde{y}\left(x_0 - \frac{L-1}{L} p_o \widetilde{y}\right). \tag{13}$$

Here we take $x_0 \equiv x_0\left(p_o\right)$ for simplicity.

Now, when we substitute $\widetilde{y}$ expressed from x_0 in Eq. 12 into Eq. 13, we obtain a quadratic equation for x_0:

$$x_0^2 \left(\frac{L}{2L-1}\right)^2 - p_o y x_0 + p_o y d = 0 \tag{14}$$

Given the "simple" obvious solution mentioned previously in Eq. 11, we choose the larger root of the quadratic equation as the solution for x_0:

$$x_0\left(p_o\right)=\frac{p_o y+\sqrt{p_o^2 y^2-4 p_o y d\left(\frac{L}{2L-1}\right)^2}}{2\left(\frac{L}{2L-1}\right)^2}. \tag{15}$$

This expression defines everything necessary for the state of the AMM. Before any exchange, one should calculate x_0 for the current state, and it stays the same while we are on the same bonding curve and p_o is unchanged.

Now let's calculate value in the AMM. In order to reduce noise, it makes sense to base it on $p = p_o$ setting (in this case, value obtained on chain would not be susceptible to sandwich attacks, for example):

$$V=\widetilde{y} p_o-d=\frac{1}{L} \widetilde{y} p_o=\frac{x_0}{2L-1}, \tag{16}$$

value of invariant in such conditions is:

$$I=\left(x_0-\widetilde{d}\right) \widetilde{y}=\frac{x_0^2}{p_o}\left(\frac{L}{2L-1}\right)^2, \tag{17}$$

so another way to express value in the pool V is:

$$V=2 \sqrt{I p_o}-x_0. \tag{18}$$

We can use V when calculating shares when doing deposits and withdrawals.

From Eq. 17, we can clearly see that x_0 is proportional to $\sqrt{I}$ at a given p_o which appears useful when we work around deposits and withdrawals further.

DEPOSITS AND WITHDRAWALS

We are removing impermanent loss in curve Cryptoswap pool, and keeping $L = 2$ leverage of its liquidity (LP tokens) for that.

Deposits

When deposit is happening, we deposit first in Cryptoswap pool, and take a loan with d stablecoins against LP token created by this deposit. User brings just cryptocurrency (like BTC) with the amount c_{in}. So deposit works in the following sequence:

- User brings c_{in} of cryptocurrency and specifies the debt Δd (which has value equal to the value of cryptocurrency ideally, or can be calculated proportionally to balances in Cryptoswap);
- System takes a loan of Δd stablecoins and deposits $(\Delta d, c_{in})$ in Cryptoswap, yielding l LP tokens;
- Calculate value in Yield Basis AMM using Eq. 16 and x_0 calculated using Eq. 15 for the state before deposit (d, y) and after the deposit $(d + \Delta d, y + l)$;
- Calculate amount of ybBTC tokens to mint for the depositor proportionally to the value increase from their deposit.

Withdrawals

When a user withdraws a fraction of liquidity, in order to not have any unfair disadvantage for liquidity providers, one cannot specify an arbitrary value of the debt. Instead, both collateral and debt are reduced by the same fraction equal to fraction of overall LP tokens being withdrawn. So withdrawal sequence inside the smart contract looks as the following:

- User brings t LP tokens to withdraw. If supply of LP tokens is s, liquidity gets reduced by s/t;
- Collateral gets reduced by $\delta c = cs/t$, and debt gets reduced by $\delta d = ds/t$;
- Since collateral is LP tokens of cryptopool, we withdraw δc LP tokens but in such a way that the amount of stablecoin withdrawn is exactly δd, and amount of cryptocurrency is kept varied as a function of δc and δd, given by `withdraw_fixed_out` method of the smart contract;
- Part of debt δd gets repaid from the withdrawn stablecoins, and cryptocurrency withdrawn from cryptoswap gets passed to the user.

SPLITTING REVENUES BETWEEN STAKED AND UNSTAKED LIQUIDITY

In Yield Basis, real yields are only going to those users who did not opt in to earn YB tokens (e.g. stake). If they did not stake - they earn real yield from fees. If they did - they don't earn any yield from fees, but they earn governance tokens from emissions. The system also earns admin fee which is taken from the fees earned and given to YB token stakers (e.g. veYB).

The fee portion going to veYB dynamically depends on how much is staked. Let's denote:

- s - amount of LP tokens staked to earn YB tokens;
- T - total amount of LP tokens;
- $f_{\min}$- minimal value of admin fee;
- r - natural return rate (e.g. how much LPs would have been earning if no token system / staking / unstaking existed).

With those, admin fee will be:

$$f_a = 1 - (1 - f_{\min}) \sqrt{1 - \frac{s}{T}}. \tag{19}$$

This formula has the following properties:

- if nothing is staked ($s/T = 0$), system admin fee is equal to $f_{\min}$ (for example, 10%), and the rest (90%) goes to LPs;
- if everything is staked ($s/T = 100\%$), system admin fee is equal to 100% because LPs are all earning YB tokens and nothing else.

Let's consider returns for the unstaked r_{us} part in different limits of how much is staked. If $s \to 0$:

$$f_a \to f_{\min,} \tag{20}$$

$$r_{us} \to (1 - f_{\min})\mathrm{r}. \tag{21}$$

And if $s \to T$:

$$f_a \to 100\%, \tag{22}$$

$$r_{us} \to \infty. \tag{23}$$

The infinite r_{us}while admin fee is going to 100% is deliberate and the reason why the admin fee formula is constructed this way.

From Eq. 16, total value of liquidity expressed in crypto is:

$$v = \frac{x_0}{(2L - 1)\, p_o}. \tag{24}$$

We define:

- v_{-1} - previous total value of liquidity;
- v_{+1} - new total value of liquidity before admin fee is taken;
- v_s - value assigned to staked liquidity (which earns no fees);
- v_{us} - value assigned to unstaked liquidity (the one which earns fees);
- v_{s*} - "ideal" value of staked liquidity (the it'd have if we have no losses);
- a - running value of collected admin fees.

Let's look at the situation when the value v is just changed due to trades (not deposits or withdrawals). Then running value of admin fees increases by:

$$\Delta a = (v_1 - v_{-1})\, f_a, \tag{25}$$

and value to be split between v_s and v_{us} is the rest:

$$\Delta v_{use} = (v_1 - v_{-1})\,(1 - f_a)\,. \tag{26}$$

After admin fee is taken, new total value of liquidity is:

$$v_1' = v_{-1} + \Delta v_{use}. \tag{27}$$

If we did not have any losses, staked liquidity value is equal to its upper limit: $v_s = v_{s*}$. However, if we experience a loss, e.g. $\Delta v_{use} < 0$, we share this loss between staked and unstaked liquidity:

$$\Delta v_s = \Delta v_{use} \frac{s}{T}, \qquad \Delta v_{use} \leq 0. \tag{28}$$

If we had profit but previously have loss, then $v_s < v_{s*}$, and we need to add value to v_s until it reaches v_{s*}:

$$\Delta v_s = \min\left(\Delta v_{use} \frac{s}{T}, \max\left(v_{s*} - v_s, 0\right)\right), \qquad \Delta v_{use} > 0. \tag{29}$$

The amount of unstaked LP tokens does not change if no deposits/withdrawals are happening, so one LP token represents a value-accruing cryptocurrency. Without deposits or withdrawals then $T - s = \text{const}$. The value of "staked" tokens, however, experiences negative rebases. Let's call reduction of number of staked LP tokens as δs. Then total and staked tokens after such a rebase accordingly change:

$$s' = s - \delta s, \tag{30}$$

$$T' = T - \delta s. \tag{31}$$

The ratio between these values should be the same as ratio between new staked and total values:

$$\frac{s - \delta s}{T - \delta s} = \frac{v_s'}{v_1'}. \tag{32}$$

From this:

$$\delta s = \frac{s v_1' - T v_s'}{v_1' - v_s'}. \tag{33}$$

ARBITRAGING LEVERAGE AMM AND VIRTUAL POOL

When we leverage liquidity, the AMM exchanges between stablecoins and LP token of cryptopool. Arbitrage trader, however, wants to exchange between real cryptocurrency and stablecoin, not LP token. Therefore, we create a “virtual pool” smart contract which combines leverage AMM, cryptopool and stablecoin (crvUSD) flash loans for that task.

Exchange from cryptoasset to stablecoin in virtual pool

The exchange consists of the following steps:

- Use a flash loan to borrow exact amount of crvUSD to add cryptoasset and those crvUSD as balanced liquidity in cryptopool. This action does not change the spot price in cryptopool;
- Exchange LP token to crvUSD in leverage AMM;
- Repay the flash loan from the crvUSD we obtained in leverage AMM;
- Return the rest of the crvUSD to the user.

Exchange from stablecoin to cryptoasset in virtual pool

Exchange in this direction is much more elaborate:

- Flash-borrow some additional amount of crvUSD (how much - we will calculate further);
- Exchange all the crvUSD we’ve got (provided by user + flash loan) to cryptopool LP in the leverage AMM;
- Remove liquidity from the cryptopool symmetrically (e.g. without changing crypto-to-stablecoin ratio and price);
- Now we should get exactly correct amount for stablecoins to return the flash loan. Thus, it is important to precisely calculate this amount before the swap.

If leverage AMM fee is f, debt is d, collateral amount in AMM is y, x_0 is known, and also cryptopool has x_c stablecoins, $y_c$amount of crypto and S_c total supply of LP tokens, we can find the flash amount φ from the input amount of stablecoins x by solving a quadratic equation:

$$\varphi^2 + \varphi \left(x_0 - d + x - \frac{x_c}{S_c} (1 - f)\, y \right) - yx \frac{x_c}{S_c} (1 - f) = 0. \tag{34}$$

If we denote $r = \frac{x_c}{S_c} (1 - f)$, we obtain:

$$D = b^2 + 4yxr, \qquad b = x_0 - d + x - ry, \tag{35}$$

$$\varphi = \frac{1}{2} \left(\sqrt{D} - b \right). \tag{36}$$

CONCLUSION

It was shown how to eliminate impermanent loss and provide automatically managed concentrated liquidity in a special automatic market-maker which is a combination of Curve cryptoswap AMM and a special releverage AMM. Simulations show promising results over 6 years timespan (20% average APR). The method described here is being implemented in Yield Basis.

Future directions of work which are currently ongoing include optimizations of borrow rate, making this borrow rate and other AMM parameters dynamic depending on asset volatility, as well as excluding any effect of quality of price data on the simulated profits and optimal parameters.