`magma-vaults`

November 21, 2024

Abstract

`magma-vaults` are a fork of CharmFi vaults for EVM [3] adapted to the Osmosis chain with minor enchancements. `magma-vaults` allow you to automatically manage 3 liquidity positions for Osmosis Supercharged pools [4]. One full range one, one concentrated, and a third one using out-of-proportion balances. Users can configure different vaults with different parameters, make them permisionless or permisioned, and decide the exact liquidity that will go into each position.

1 Introduction

UniswapV3 introduces the concept of ticks to allow for concentrated liqudity positions over custom ranges. Thus, we distinguish 2 spaces we want to work in: A tick space, where all ticks live, and a price space, where all prices live. As prices are logarithmic, we construct a linear tick space by taking logarithms. Its in this way that UniswapV3 price function $\rho(t) = 1.0001^t$ is constructed (equation (6.1) [2]), with the nice property that given the current tick t of a liquidity pool, any concentrated position equidistant to t will be balanced. That its, for any $\varepsilon > 0$, positions in tick range $[t-\varepsilon, t+\varepsilon]$ with reserves (x, y) will have $\rho(t) = y/x$.

One of the emergent features in UniswapV3 is that you can use concentrated liquidity positions as limit orders, with the problem that, due to their logarithmic nature, as prices grow, tick precision decreases. To give an example, if the current price of bitcoin was 80000USDC, the difference between the 2 closest ticks is of almost 8USDC. For this reaon, Osmosis introduces geometric tick spacing with additive ranges [5], that allows us to define an order book on top of the AMM with proper tick precision. Its easy to verify that its price function is:

$$\rho(t) = 10^{\lfloor t/9e6 \rfloor - 6} \left(t + 10^6 \left(1 - 9e6 \left\lfloor \frac{t}{9e6} \right\rfloor \right) \right)$$

Naturally, mapping the tick space and price space in this way breaks our nice equidistant property. For this reason, `magma-vaults` does operations over the price space instead of the tick space, and then takes the price function generalized inverse to decide the ticks for the balanced base position and the limit one. It is then trivial to prove that, given the current tick t and the current price $p := \rho(t)$, any position in the price range $[p/k, pk]$ will be balanced (we keep $k > 1$

for simplicity). We refer to those factors k as `PriceFactors` in the code, and they will be, with the the `Weight` type, our fundamental building block to configure custom strategies.

2 Architecture overview

Magma vaults manage up to 3 concentrated liqudity positions. A full range position, a base balanced position, and a limit position. Both, the full range position and the base position will be balanced, but as prices move, the base position could become unbalanced. Thus we expose a `rebalancing` functionality, that will burn all positions and create them again centered around the new price. Moreover, its easy to see that the total vault reserves wont always be in proportion, so any out-of-proportion reserves will be used for the limit position. Actually, vault total reserves almost never will be in proportion, but its still theoretically possible a vault could get rebalanced to have no limit position. Thus, the core business logic of the vault contract, i.e, the rebalancing procedure, will:

1. Burn all active liquidity positions to withdraw total reserves (X, Y).

2. Calculate all the balanced reserves (x, y) in the vault. Its trivial to verify that:

$$(x, y) = \begin{cases} (X, Xp) & , \ Y/p > X \\ (Y/p, Y) & , \ \text{otherwise} \end{cases}$$

3. Calculate the reserves (x_0, y_0) to put into the full range position. For this, the vault admin (user) will decide a liquidity `Weight` $w \in (0, 1)$. For simplicity, we dont allow for extreme value of w, but the user still could make the difference between w and extremes negligible. The reserves will be calculated in such way that, if L_0 is the liquidity of the full range position, and L_1 is the liquidity of the base one, $L_0/(L_0 + L_1) = w$ will be invariant. We will elaborate on this computation in following sections.

4. Calculate trivially the reserves (x_1, y_1) to put into the base range position as $(x_1, y_1) = (x - x_0, y - y_0)$. Then, calculate the concentrated tick range as $[\rho^{-1}(p/k_1), \rho^{-1}(pk_1)]$, for current price p and `PriceFactor` k_1 decided by the user. The inverse ρ^{-1} will be discussed in following sections.

5. Calculate trivially the reserves (x_2, y_2) to put into the limit position as $(x_2, y_2) = (X - x, Y - y)$. Naturally, $x_2 = 0 \vee y_2 = 0$ is invariant, and trivial from (2). We then simply

compute the tick range for the limit position as:

$$[t_a, t_b] = \begin{cases} [\rho^{-1}(p/k_2), \rho^{-1}(p)] & , \; x_2 = 0 \\ [\rho^{-1}(p), \rho^{-1}(pk_2)] & , \; y_2 = 0 \end{cases}$$

Where p is the current price, and k_2 a `PriceFactor` decided by the user.

Thus, the only remaining question is when to rebalance. For this, any vault can use up to 3 rebalancing strategies:

- `admin`: Only the vault admin can rebalance the vault whenever they feel like. Thus, its up to the admin to decide the rebalancing strategy off-chain.
- `delegate`: Any delegate address, decided by the admin, can rebalance the vault whenever they feel like. This allows for the system extensionality with, for example, oracle integration to rebalance automatically.
- `anyone`: Anyone can rebalance the vault as long as the price has moved outside the range $(p/k, pk)$ (where p is the last snapshoted price during rebalance, and k a `PriceFactor` decided by the admin) and as long as enough time has passed (threshold also decided by the admin). This allows for fully permisionless vaults if the vault admin decides to burn its ownership.

3 Full range reserves computation

Charmfi Alpha Vaults allocate liquidity to the full range position with the formula $L = w\sqrt{XY}$ [1]. But this introduces an imprecision: As the base range size is variable, its liquidity will also be. Thus, `magma-vaults`, to make easier to reason about fees, ensures the liquidity of the full range position remains constant through the invariant:

$$w = \frac{L_0}{L_0 + L_1}$$

Thus, for example, if we set $w = 0.5$ we will be sure that, during any period of time, both positions will earn the same amount of fees (assuming of course that the base position stays in range during that time).

Clearly, $L_0 = x_0\sqrt{p}$. We then express L_1 in terms of UniswapV3 trading curve (equation (2.1) of the paper [2]) by taking liqudity on the right, and k as the base range `PriceFactor`:

$$L_1 = x_1 \frac{\sqrt{p}\sqrt{pk}}{\sqrt{pk} - \sqrt{p}} = x_1 \frac{\sqrt{pk}}{\sqrt{k} - 1}$$

Substituting then into our w invariant:

$$\begin{aligned}
w &= \frac{L_0}{L_0 + L_1} \\
&= \frac{x_0\sqrt{p}}{x_0\sqrt{p} + \frac{x_1\sqrt{pk}}{\sqrt{k}-1}} \\
&= x_0 \left(x_0 + \frac{x_1\sqrt{k}}{\sqrt{k} - 1} \right)^{-1} \\
&= x_0 \left(\frac{x_0(\sqrt{k} - 1) + x_1\sqrt{k}}{\sqrt{k} - 1} \right)^{-1} \\
&= \frac{x_0(\sqrt{k} - 1)}{x_0(\sqrt{k} - 1) + (x - x_0)\sqrt{k}} \\
&= \frac{x_0\sqrt{k} - x_0}{x_0\sqrt{k} - x_0 + x\sqrt{k} - x_0\sqrt{k}} \\
&= \frac{x_0\sqrt{k} - x_0}{x\sqrt{k} - x_0} = w \overset{(A)}{\Longrightarrow} x_0 = \frac{\sqrt{k}wx}{\sqrt{k} + w - 1}
\end{aligned}$$

Where at step (A) we simply solve for x_0. And once we know x_0, we of course can simply get y_0 as $y_0 = x_0 p$, which ensures the reserves to be in proportion. We can ensure this derivation is sound in multiple ways. For example, by proving $x_0 \leq x \leq X$ (left as an exercise). Intuition also tells us the final expression for calculating x_0 makes sense. Why would we care about Y, y or p to compute the allocation of tokens x_0, if y_0 is already its dual, as the position is balanced?

3.1 Computation security proof

We now prove that the way x_0 is computed in the `magma-vaults` codebase is secure. Thus considered the implementation:

```
pub fn calc_x0(k: &PriceFactor, w: &Weight, x: Decimal) -> Decimal {
    if w.is_zero() { return Decimal::zero() }
    do_me! {
        let sqrt_k = k.0.sqrt();

        let numerator = w.mul_dec(&sqrt_k);

        let numerator = Decimal256::from(numerator)
            .checked_mul(x.into())?;                        // (1)

        let denominator = sqrt_k
            .checked_sub(Decimal::one())?                   // (2)
            .checked_add(w.0)?;                             // (3)

        let x0 = numerator.checked_div(denominator.into())?; // (4)
        Decimal::try_from(x0)?                              // (5)
    }.unwrap()
}
```

The `do_me!` macro only creates a `anyhow::Error` closure and runs it, so we only need to ensure the commented lines wont produce any errors:

(1). Wont overflow as $(2^{128} - 1)^2 < 2^{256} - 1$.

(2). Wont underflow as $k \in [1, \infty)$, thus $\sqrt{k} - 1 = 0$ in the worst case.

(3). Wont overflow as $w \in [0, 1]$, and we just subtracted 1 from $\sqrt{k}$.

(4). Could only produce a division by zero if $\sqrt{k} - 1 + w = 0$. Assume $k = 1$, then $\sqrt{k} - 1 + w = 0 \iff w = 0$, in which case x_0 computation is trivial (see first line of the implementation). Note that also $k = 1 \wedge w = 0$ produces a vault with generally idle capital and only limit positions (which the first version does not support). On the other hand, the division wont overflow because $x_0 \leq x$, as stated earlier.

(5). Finally, the downgrade to 128 bits wont fail because, again, $x_0 \leq x$, but x fits in 128 bits by definition.

4 Price function inverse computation

Its:

$$t(p) = 10^{6-\lfloor \log p \rfloor} \Big(p + 10^{\lfloor \log p \rfloor} \left(9 \lfloor \log p \rfloor - 1 \right) \Big)$$

But still work in progress; The derivation is not important, whats important is to prove its a generalized inverse of osmosis price function.

4.1 Computation security proof

Work in progress. I already got the proof in a obsidian note, just need to review and formalize it.

References

[1] Charmfi liquidity computation. https://github.com/charmfinance/alpha-vaults-v2-contracts/blob/main/contracts/AlphaProVault.sol#L420, 2021. Accessed: 2024-11-20.

[2] Hayden Adams et al. Uniswap v3 core. https://app.uniswap.org/whitepaper-v3.pdf, March 2021. Accessed: 2024-11-20.

[3] CharmFi. Alpha Vaults whitepaper. https://learn.charm.fi/charm/products-overview/alpha-vaults/whitepaper, 2021. Accessed: 2024-11-20.

[4] Osmosis. Concentrated Liquidity module documentation. https://docs.osmosis.zone/osmosis-core/modules/concentrated-liquidity. Accessed: 2024-11-20.

[5] Osmosis. Geometric Tick Spacing with Additive Ranges documentation. https://docs.osmosis.zone/osmosis-core/modules/concentrated-liquidity/#geometric-tick-spacing-with-additive-ranges. Accessed: 2024-11-20.