Cubic Concentrated Liquidity Pool (3-CLP): Technical Overview

Ariah Klages Mundt Steffen Schuldenzucker

Aug 2022

Gyroscope's 3-CLPs are AMMs that concentrate liquidity of three assets to a pricing range. 3-CLPs are implemented using a three-dimensional constant product (or geometric mean) curve with virtual reserves. In this implementation, virtual reserves include both the real reserves and 'reserve offsets' that allow the pool to achieve its pricing region.

Note that the price bounds of the three assets cannot be independent: choosing one price bound restricts the price bounds that are possible in the pool for other assets. Intuitively, whereas the 2-CLP is effectively taking a segment of the two-dimensional constant product curve where the endpoints can be taken independently, the 3-CLP is effectively taking the intersection of the three-dimensional surface with a plane for the boundary. Additionally, because of this, the pricing region of a 3-CLP is not a multi-dimensional interval, but is instead a subset of this. For instance, the lowest (or highest) prices that can be represented in the pool are only achieved when two of three reserve balances are zero.

The first 3-CLPs are designed for *symmetric* price bounds $[\alpha, 1/\alpha]$ on the three asset pairs in the pool. The parameter in Table 1 specifies a symmetric 3-CLP.

Parameter	Description
$0 < \sqrt[3]{\alpha} < 1$	Cube root of the lower bound of the price range

Table 1: Parameters for a symmetric 3-CLP

Calculating the Invariant L. Let x, y, z denote the reserves in the pool. The invariant is the three-dimensional constant product

$$L^3 = (x + a)(y + a)(z + a)$$

where the virtual reserve offsets are all a in the symmetric case. The price bounds are achieved when

$$a = L\sqrt[3]{\alpha}$$

This makes the invariant equation into the following cubic equation

$$(1 - \alpha)\, L^3 - (x + y + z)\alpha^{2/3} L^2 - (xy + yz + xz)\sqrt[3]{\alpha} L - xyz = 0.$$

The cubic equation is solved by applying Newton's method. The polynomial has one local minimum L_+ and a unique root in $[0, \infty)$. The root is greater than $1.5 \cdot L_+$ and converges to $1.5 \cdot L_+$ as $\alpha \to 1$. L_+ is computed by a quadratic equation, and $1.5 \cdot L_+$ is used as the starting point for Newton's method.[1] Newton's method typically converges within five iterations to find the invariant.

Note that special numerical care is needed to ensure that the invariant calculation doesn't overflow as L^3 and xyz are large numbers when balances are large and α is close to 1.

Swap Execution. Given the invariant, and so also a, and the current reserves x and y, we can compute a swap of Δx to Δy by preserving the constant product where the final reserves are $x + \Delta x$ and $y - \Delta y \geq 0$. The swap calculations simplify to

$$\Delta y = \frac{(y + a)\Delta x}{x + a + \Delta x}$$

$$\Delta x = \frac{(x + a)\Delta y}{y + a - \Delta y}$$

Note that we need $\Delta y \leq y$. When the swap is calculated based on Δy, this is obvious; when the swap is calculated based on Δx, the resulting Δy needs to satisfy $\Delta y \leq y$.

The equations are analogous when swapping Δy to Δx or when swapping in

[1] For small values of α (specifically, $\alpha < 0.5$), $2 \cdot L_+$ is used instead as the starting point, as this yields a better approximation of the root for these values.

the third asset after substituting z and Δz.

Implementation. Table 2 lists the most important functions that implement the above calculations. They are all defined in the file `GyroThreeMath.sol`.

Function	Purpose
`calculateInvariant`	Computes the invariant L from current reserves (x, y, z).
`calculateCubicStartingPoint`	Computes the starting point for Newton's method.
`calcNewtonDelta`	Calculates a step in Newton's method with special care not to overflow.
`runNewtonIteration`	Executes Newton's method, including an appropriate stopping criterion.
`calcOutGivenIn`	Computes the amount that leaves the pool when a certain amount enters it, after fees.
`calcInGivenOut`	Computes the amount that needs to enter the pool when a certain amount should leave it, after fees.
`liquidityInvariantUpdate`	New invariant when liquidity is added/removed in a "balanced" fashion (without affecting the price). This avoids fully re-calculating the invariant.

Table 2: Most important functions