Denaria Perpetual DEX

v 0.1

calmQuant_

May 9, 2025

Abstract

Denaria Perpetual Exchange introduces a novel decentralized protocol for trading perpetual futures. The core innovation lies in its dynamic virtual Automated Market Maker (dynAMM), built around a custom implementation that combines oracle-based price feeds with an internally recalibrated pricing curve. This design ensures that trades are executed near the real market price, with protection mechanisms managed directly by the dynAMM. The protocol supports a flexible multi-stablecoin collateral system, allowing users to deposit different stablecoins in arbitrary proportions and settle PnL accordingly. Liquidity providers benefit from the ability to be matched with traders at the market price, avoiding unfavorable market inefficiencies, and from the flexibility to provide imbalanced or one-sided liquidity, with an efficient algorithm to track share ownership and provide a fair fee distribution and passive strategic exposure. To maintain equilibrium, Denaria employs a funding payment mechanism not based on mark-index price deviation, but on the net imbalance between long and short positions—thus incentivizing traders to reduce LP risk. Denaria offers a new approach to perpetual trading, focused on minimizing LP risk, optimizing execution, and ensuring robustness in decentralized on-chain derivatives trading.

Contents

1 Protocol overview

Perpetual futures are derivative contracts that allow users to speculate on the future price of an underlying asset without an expiry date. In our protocol, every perpetual contract revolves around two virtual assets. The first, vAsset, mirrors the underlying asset whose price the trader intends to predict, while the second, vStable, is a dollar-pegged stablecoin employed solely for accounting and quotation. Users never purchase or sell the real underlying asset; their positions exist only in these virtual tokens, which they exchange through our dynamic virtual automated market maker (dynAMM).

The protocol's architecture intertwines several modules. A **multi-collateral vault** secures user deposits; the **dynAMM** provides price computation and trade execution; an in-house **share-tracking algorithm** keeps precise records of each liquidity provider's contributions and fee earnings; a dedicated **funding fee mechanism** limits market exposure; a carefully designed schedule of trader and LP **fees** stabilizes the pool; and, finally, a **liquidation process** safeguards solvency whenever positions become under-collateralised. Together these components create a seamless environment where traders can take leveraged positions and liquidity providers (LPs) can earn fees while managing risk.

Interaction with the system begins with depositing collateral. The vault accepts several approved assets and does not impose rigid, hard-coded ratios between them, so users may deposit any blend of the allowed tokens, subject only to mild limits that prevent extreme changes in the vault's composition. When a position is opened, the collateral is locked and any profit-and-loss (PnL) payments are, under ordinary circumstances, returned in the same proportions as the original deposit. If the vault's overall mix differs sharply from a new user's deposit -imagine the vault holds mostly Asset A, but the user provides Asset B- the protocol dynamically adjusts withdrawals so that payouts reflect both the vault's global ratios and the user's preferred mix. The first version of our protocol restricts collateral to USD-pegged stablecoins, valued initially at one dollar each.

Each underlying asset has its own dedicated vault. After funding their account, users may borrow vAsset or vStable up to the leverage cap. vAsset is valued through an external oracle, while vStable is fixed at one dollar. These vTokens remain internal bookkeeping entries; they never leave the protocol. To open a long position the trader borrows vStable and swaps it for vAsset in the dynAMM; to close, the trader performs the inverse swap, and the difference between vStable repaid and vStable received—adjusted for funding payments—constitutes the final profit or loss.

The dynAMM itself continuously recentres its pricing curve on the oracle price, following mathematics akin to Curve's StableSwap and CryptoSwap [1], [2] formulas but re-implemented from scratch for gas efficiency. As a result, trades are executed at the prevailing market price with minimal slippage, and the funding rate is no longer tied to AMM slippage. Liquidity providers may supply balanced, imbalanced, or even single-sided liquidity, enabling precise control over exposure—depositing only vAsset, for example, is equivalent to taking a passive short. Because every trade shifts the pool's composition, we maintain each LP's claim with a single global accumulator plus periodic per-LP snapshots, an approach that keeps gas costs low without sacrificing accuracy.

Trading fees are charged on every swap and collected in vStable, while additional composition fees steer LP deposits toward a healthy pool balance. Funding payments counteract prolonged market trends: traders on the popular side pay, and traders on the less-popular side receive, a fee proportional to position size and a novel funding rate computed from LP exposure rather than a mark-index price gap. The same accumulator-and-snapshot technique that tracks LP shares also measures each account's funding liability or entitlement.

Positions must maintain a minimum margin ratio, defined as collateral value after PnL were applied divided by the notional position value. If this ratio falls below the liquidation threshold, anyone may liquidate the position by purchasing it at a discount. The discount is chosen so that slippage cannot erode the liquidator's expected profit, ensuring strong incentives to keep the system solvent.

2 Multi stable-asset collateral vault

In the initial release of the protocol, collateral is restricted to USD-pegged stablecoins. Users may deposit any combination of the approved coins in whatever proportions suit them, and -under ordinary conditions- their PnL is settled in those same proportions. This symmetry cannot always be satisfied, though. Suppose a vault contains funds from only two users, one who contributed exclusively stablecoin A and another who supplied only stablecoin B. The vault clearly cannot satisfy each payout in the exact coin mix originally deposited. To handle such situations the protocol employs an adaptive settlement algorithm, outlined in this next section, that redistributes assets fairly whenever the vault's composition diverges from a user's preferred mix.

The user is permitted to deposit collateral in any of n stablecoins, denoted by Stable_1 to Stable_n . Let $y_1, ..., y_n$ the amount of each coin he deposits respectively as collateral. Some y_i can be 0. We define:

$$r_i = \frac{y_i}{y_1 + y_2 ... + y_n}, i = 1, 2, ...n$$

All the stable coins are assumed to hold their peg. The sum in the denominator (let's denote it by y) is the total value in USD deposited as collateral by the user.

If these ratios were the same for each user we could force the users to pay for their PnL in these same ratios and then the system would have the desired properties. But this would be too restrictive, so we will describe a more flexible approach.

We denote by $r_{i,tot}$ the same ratios for the whole system/vault before the new collateral deposit i.e if $y_{1,tot}, ...y_{n,tot}$ the total amount of Stable_1 to Stable_n tokens in the vault before this new collateral deposit:

$$r_{i,tot} = \frac{y_{i,tot}}{y_{1,tot} + ... + y_{n,tot}} \tag{1}$$

and by $r'_{i,tot}$ the total ratios after the new deposit i.e.:

$$r'_{i,tot} = \frac{y_{i,tot} + y_i}{(y_{1,tot} + y_1) + ... + (y_{n,tot} + y_n)} \tag{2}$$

- The user is free to deposit collateral in any ratio provided that:

$$\left| \frac{r_{i,tot} - r'_{i,tot}}{r_{i,tot}} \right| < th_i \tag{3}$$

  The thresholds th_i will be determined before deployment.

- When the user wants to close his position: Let's say that his total PnL are y_{PnL} USD (negative PnL means that the user should be paid and positive that he should pay). The total USD value of the collateral he can withdraw is $y - y_{PnL}$. We compute $r'_i, i = 1, 2, ...n$, as described below, and give him back $r'_1 \cdot (y - y_{PnL})$ Stable_1 , ... $r'_n \cdot (y - y_{PnL})$ Stable_n .

Computation of the ratios:If everyone was depositing collateral in the exact same ratios r_i, then r'_i should also be equal to r_i, meaning the user pays their PnL proportionally to their initial deposit. However, since not everyone deposits collateral at the same ratio, r'_i will depend on the user's initial ratio (r_i), as long as the total ratios of the pool immediately before ($r_{i,tot}$) their collateral withdrawal.

We first compute $dy_i = (y - y_{PnL}) \cdot r_i$ and then the total ratios after the withdrawal:

$$r'_{i,tot} = \frac{y_{i,tot} - dy_i}{(y_{1,tot} - dy_1)) + ... + (y_{n,tot} + dy_n)} \tag{4}$$

Observe that the denominator equals $y' = y_{1,tot} + ... + y_{n,tot} - (y - y_{PnL})$ and therefore does not depend on the splitting.

If

$$\left| \frac{r_{i,tot} - r'_{i,tot}}{r_{i,tot}} \right| < th'_i \tag{5}$$

holds for every i, we set $r'_i = r_i$ for every i. The withdrawal threshold th'_i could be the same or smaller than th_i.

If the above condition is not satisfied, or if some of the dy_i as copmputed above exceed the available $y_{i,tot}$ in the vault, we set $r'_i = r_{i,tot}$ for every i.

Note: We are aware that the user can bypass the restrictions (3) and (4) by splitting his deposit or withdrawal to smaller ones that will gradually shift the total ratios of the vault. A mitigation against it is to use instead of the actual ratios $r_{i,tot}$ time averages.

3 Dynamic AMM

Every perpetual protocol needs a pricing mechanism -a way to decide how much vAsset a trader receives for their vStable when opening a long position and vice verse for short- while keeping that quoted price aligned with the underlying market price.

Traditionally, protocols either use oracle pricing, which takes the live market price of the underlying from an oracle, or virtual AMM (vAMM) pricing, which lets a constant-product AMM determine the price and relies on arbitrage to pull it back toward the market. Our design blends the strengths of both. We start with an AMM that has very low slippage and then continuously re-center its curve around the oracle price instead of waiting for external arbitrage. We call this mechanism a dynamic virtual AMM, or dynAMM.

The dynAMM fairly compensates LPs because trades clear close to the true market value. Capital efficiency is preserved: minimal slippage protects the pool from being drained yet still lets trading pressure show up in prices. For traders, execution remains near the market price without needing outside market makers.

Dynamic AMMs have been discussed in prior academic works [3], but not in the context of perpetual futures and only for simpler AMM models. Our dynAMM extends the concept to a more sophisticated perpetual-trading environment.

In this section we denote by (x_0, y_0) and (x, y) the amounts of the two assets in the pool (vAsset and vStable respectively for our dynAMM case) right before thd right after a trade, respectively, and p is the market price (provided by an external oracle) of the first asset with respect to the second one. We will introduce an invariant $I_{p,x_0,y_0}(x, y)$. From this equation one variable can be expressed in terms of the other -solving for x yields y and vice versa. Plotting $I_{p,x_0,y_0}(x, y) = 0$ in the (x, y)-plane produces a continuous curve. We choose the invariant so that this curve passes through the point (x_0, y_0) and its spot price at that point, i.e. $-dy/dx$ equals p. As a result, trades clear close to the prevailing market price (small orders incur only minimal slippage; significant slippage appears only when the trade size becomes large). We construct the invariant by starting with the graph of a constant function, then applying suitable shifts and scalings so that it meets both conditions.

The graphs for all the formulas discussed in this section can be found here: `https://www.desmos.com/calculator/ptnnvsv1eo`.

3.1 The formula of the dynAMM

Our invariant is a dynamic extension of the Curve Finance invariant. For the sake of completeness -and because the resulting formulas may be useful beyond our protocol- we present a detailed derivation below.

We begin by introducing two simpler invariants, whose combination will produce the final invariant.

The first one is a dynamic version of the constant sum formula:

$$S_{p,x_0,y_0}(x,y) := \frac{p(x-x_0)+y-y_0}{px_0+y_0} \tag{6}$$

Using this invariant, all trades are executed exactly at the market price p, with zero slippage. Note also that this and all subsequent invariants are normalized and dimensionless.

The second is a dynamic version of the constant product formula [4]. For this invariant there are two possible choices, as the shift can be applied either in the x or the y variable:

$$P^x_{p,x_0,y_0}(x,y) := 1 - \frac{y_0^2}{[p(x-x_0)+y_0]y} \tag{7}$$

$$P^y_{p,x_0,y_0}(x,y) := 1 - \frac{px_0^2}{[y-y_0+px_0]x} \tag{8}$$

Although both shifted versions of the constant product formula exhibit qualitatively similar behavior, they define two distinct AMMs -meaning that, for the same input, they produce entirely different outputs. We use the first version for short positions and the second for long positions. Here's why:

Consider the first formula. A key distinction arises between the cases $p \cdot x_0 < y_0$ and $p \cdot x_0 > y_0$ (the special case $p \cdot x_0 = y_0$ corresponds to the standard -non dynamic- constant product formula). When $p \cdot x_0 < y_0$, the graph intersects the y-axis, making it suitable only for short positions, which only increase x. However, for long positions (which increase y and thus decrease x), this formula can yield negative values of x, which are invalid.On the other hand, when $p \cdot x_0 > y_0$, the graph can be used for both longs and shorts. But for long positions in particular, the graph becomes asymptotic to the line $x = x_0 - y_0/p$. This implies that the amount of the x-asset cannot decrease below this threshold, which limits the efficiency of the AMM -it prevents full utilization of the x-asset in the pool. This inefficiency becomes especially pronounced when $p \cdot x_0 >> y_0$.

For the second formula, this behavior is reversed: it is more suitable for long positions and less effective for shorts.

We would prefer a dynAMM that combines the best properties of both the constant sum and constant product formulas. Specifically, we would like its graph to approximate the constant sum curve for values of (x, y) close to (x_0, y_0). This approach minimizes slippage for swaps that do not significantly move the pool, ensuring such trades are executed near the market price p. However, for larger swaps that shift the pool far from (x_0, y_0), significant slippage should apply, as with the constant product rule, to prevent the pool from being depleted.

This behavior can be achieved by defining a new invariant that is a linear combination of the constant sum and constant product invariants:

$$A_{p,x_0,y_0}(x,y) \cdot S_{p,x_0,y_0}(x,y) + P^x_{p,x_0,y_0}(x,y) = 0 \tag{9}$$

for short positions and

$$A_{p,x_0,y_0}(x,y) \cdot S_{p,x_0,y_0}(x,y) + P^y_{p,x_0,y_0}(x,y) = 0 \tag{10}$$

for long. $A_{p,x_0,y_0}(x,y)$ should be a positive function. Observe that for $A = 0$ we get the (shifted) constant product formula and for $A \to \infty$ the constant sum. The graph of this new AMM will lie between the graphs of the constant sum and that of the constant product.

We will use a function $A_{p,x_0,y_0}(x,y)$ inspired from Curve:

$$A_{p,x_0,y_0}(x,y) = \frac{Ay_0^4}{\left(By_0^2 + y_0^2 - [p(x-x_0)+y_0]y\right)^2} \tag{11}$$

We can then can write the invariant as a polynomial equation of degree 3:

$$ay^3 + by^2 + cy + d = 0 \tag{12}$$

where:

$$\begin{aligned}
a &= \lambda^3 \\
b &= \frac{Ay_0^4}{px_0 + y_0}\lambda - 2\lambda^2 By_0^2 - 3y_0^2\lambda^2 \\
c &= \lambda\left(\frac{Ay_0^4}{px_0+y_0}(p(x-x_0)-y_0) + (B+1)^2y_0^4 + 2(B+1)y_0^4\right) \\
d &= -y_0^6(B+1)^2 \\
\lambda &= p(x-x_0) + y_0
\end{aligned}$$

For longs we can use:

$$A_{p,x_0,y_0}(x,y) = \frac{Ap^2x_0^4}{\left(Bpx_0^2 + px_0^2 - [y - y_0 + px_0]x\right)^2} \tag{13}$$

and the corresponding polynomial equation is:

$$ax^3 + bx^2 + cx + d = 0 \tag{14}$$

where:

$$\begin{aligned}
a &= \lambda^3 \\
b &= \frac{Ap^2x_0^4}{px_0 + y_0}\lambda p - 2\lambda^2(B+1)px_0^2 - px_0^2\lambda^2 \\
c &= \lambda\left(\frac{Ap^2x_0^4}{px_0+y_0}(y-y_0-px_0) + (B+1)^2p^2x_0^4 + 2p^2x_0^4(B+1)\right) \\
d &= -p^3x_0^6(B+1)^2 \\
\lambda &= y - y_0 + px_0
\end{aligned}$$

We will also need the "inverse" formulas of the above. To compute the amount of vAsset required to receive a specified amount of vStable, the following equation should be used:

$$ax^3 + bx^2 + cx + d = 0 \tag{15}$$

where:

$A' = \dfrac{Ay_0^4}{px_0 + y_0}$

$\lambda = y - y_0 - px_0$

$k = y_0 - px_0$

$a = p^3y^3$

$b = A'yp^2 + 2p^2y^3k - 2p^2(B+1)y_0^2y^2 + (ky - y_0^2)p^2y^2$

$c = A'py(k+\lambda) + (B+1)^2py_0^4y + pk^2y^3 - 2(B+1)py_0^2y^2k + 2py^2k(ky - y_0^2) - 2(B+1)y_0^2yp(ky - y_0^2)$

$d = A'yk\lambda + (ky - y_0^2)(B+1)^2y_0^4 + (ky - y_0^2)k^2y^2 - 2(B+1)y_0^2yk(ky - y_0^2)$

If the state of the pool before the trade was (x_0, y_0) and someone wants to get exactly dy vStable ($y = y_0 - dy$), he should add to the pool dx vAsset ($x = x_0 + dx$), and this dx can be found by solving equation (15). Also, we can use this equation to compute the amount of vAsset a user should add to the pool to get the amount of vStabe he needs to close his long position.

On the contrary, to calculate the amount of vStable required to obtain a specific amount of vAsset, we use the following equation:

$$ay^3 + by^2 + cy + d = 0 \tag{16}$$

where:

$A' = \dfrac{Ap^2x_0^4}{px_0 + y_0}$

$\lambda = p(x - x_0) - y_0$

$k = px_0 - y_0$

$a = x^3$

$b = A'x + 2x^3k - 2p(B+1)x_0^2x^2 + (kx - px_0^2)x^2$

$c = A'x(k+\lambda) + (B+1)^2p^2x_0^4x + k^2x^3 - 2(B+1)px_0^2x^2k + 2(kx - px_0^2)kx^2 - 2(B+1)(kx - px_0^2)px_0^2x$

$d = A'xk\lambda + (kx - px_0^2)(B+1)^2p^2x_0^4 + (kx - px_0^2)k^2x^2 - 2(B+1)px_0^2xk(kx - px_0^2)$

If the state of the pool before the trade was (x_0, y_0) and someone wants to get exactly dx vAsset ($x = x_0 - dx$), he should add to the pool dy vStable ($y = y_0 + dy$), and this dy can be found by solving equation (16). Also, we can use this equation to compute the amount of vStable a user should add to the pool to get the amount of vAsset he needs to close his short position.

Note: In the smart contract implementation, we solve these equations using numerical methods, specifically Newton's method. We chose to work with the polynomial form of the invariant because, once the polynomial coefficients are computed, each iteration of the numerical method becomes simpler—and, importantly, more gas-efficient.

3.2 Using our dynamic vAMM to execute trades

Let's now discuss how these formulas are employed to execute trades. Recall that in the previous sections we used distinct formulas/graphs for short and long positions (they coincide only when $x_0/y_0 = p$). This was done to ensure that the resulting equations yield positive solutions and that the deposited liquidity is used optimally for trades.

For clarity -purely for exposition purposes without affecting the implementation- we combine these formulas and graphs into a single unified curve. This curve uses the formula for shorts when $x > x_0$ and the formula for longs when $x < x_0$, resulting in a graph that is both continuous and smooth, with a continuously differentiable first derivative. It can be described by an invariant $f(p, x_0, y_0, x, y)$, where the equation follows equation (12) or (15) for $x > x_0$, and equation (14) or (16) for $x < x_0$.

How should a trade be executed? Since we would like to execute trades as close to the market price as possible -a key motivation behind our vAMM design-, we could do the following. We start by retrieving the current pool deposits (x_0 and y_0) along with the current market price p from the oracle, and then solve the equation $f(x_0, y_0, p, x_0 + dx, y_0 + dy) = 0$, where dx, dy are the exchanged amounts.

In simpler terms, we continuously readjust the graph so that the initial state is always the current (x_0, y_0), ensuring that the price asymptotically aligns with the market price and slippage remains minimal. However, this dynamic mechanism introduces a potential exploit: a user could split a larger position into several smaller trades, thereby reducing overall slippage. In the extreme, by splitting a position into infinitely small trades, the user could achieve an execution price arbitrarily close to the market price (albeit still marginally higher).

To counter this exploit, we are imposing a minimum position size. Even without such a restriction, our dynamic vAMM ensures that LPs always receive at least the market price, thus protecting them from impermanent loss.

We are also implementing another measure to prevent this attack. Rather than constantly readjusting the graph with every trade around the current state, we adjust it so that two consecutive trades in the same direction (for instance, opening two long or two short positions) executed in quick succession have the same impact -actually slightly worse for the user- as a single trade whose size is the sum of the two trades. This approach renders the attack ineffective.

Let's discuss the specifics of how this works. For this method, the contracts will store three additional variables, denoted dx_0, dy_0, and t_0 (or $lastCurveUpdate$ in the contracts) and also the type of the last trade (long or short). Consider the case of long trades.A user executes a long trade (let's suppose that a long enough time has passed since the previous trade) by depositing dy vStable and getting back dx vAsset from the vAMM.We set t_0 to the current timestamp, $dx_0 := +dx, dy_0 := +dy$.We denote by x_0, y_0 the total amount of vAsset and vStable respectively in the vAMM after this trade. Let's assume the next trade is also long and its timestamp differs from t_0 by less than a specific threshold set in the contracts.

If the user exchanges dy' vStable, we first solve the equation $f(p, x_0 + dx_0, y_0 - dy_0, x_2, (y_0 - dy_0) + dy_0 + dy')$ for x_2 i.e. we align the graph around a state ($(x_0 + dx_0, y_0 - dy_0)$)as if the previous long trade has not happened and we just deposited the total vStable amount of all the last long trades $(dy_0 + dy')$ at once. The difference $dx_{tot} = (x_0 + dx_0) - x_2$ is the total amount of vAsset someone would get if he executed both trades at once (actually a little bit less, because of the slippage inrease after the first trade).The amount of vAsset the second trader gets is computed as $dx_{tot} - dx_0$. We finally increase dx_0, dy_0 by dx' and dy' respectively and set x_0, y_0 equal to the total vAsset and vStable liquidity in the pool after this second trade.

If the second trade was a short trade, or time more than the specified threshold had passed, we would reset dx_0 and dy_0 to 0 and would store this new timestamp as the new t_0.

4 Algorithm for efficiently tracking the LP shares

We aim to allow imbalanced deposits, meaning LPs can contribute liquidity to the pool in proportions where the ratio of the two deposited assets does not necessarily match either the pool's current ratio or the market price. This flexibility allows LPs to implement more general strategies and select risk profiles that best suit their needs. However, this added flexibility introduces extra complexity into the system.

The LP's share of each asset is not the same and, in fact, not even constant. Consider an edge case where an LP deposits only one asset. In this scenario, the LP's share for that particular asset is non-zero, but their share of the other asset is zero. After a swap that utilizes some portion of the deposited asset, the LP will also own a non-zero amount of the second asset. This change must be reflected in the dynamic adjustment of their shares.

To make this approach feasible, we need an efficient method to compute these constantly changing shares in a way that is scalable and independent of the number of swaps or LPs. In these notes, we outline an algorithm to achieve this.

By defining the LP shares dynamically, we can accurately track the liquidity that each LP can withdraw, depending on what they have deposited and also fairly distribute the fees earned through swaps, proportional to each LP's dynamically updated shares.

4.1 Description of the algorithm

Let's start by introducing some notation:

- t: discritized time, updated by 1 after each action (either swap or liquidity deposit/withdrawal).
- $x(t), y(t)$: the amount of the X and Y assets in the pool respectively, at time t.

- $S_x(t), S_y(t)$: The total shares for the X and Y asset respectively, at time t.
- $s_{x,i}(t), s_{y,i}(t)$: The shares of the i LP for the X and Y asset respectively, at time t. The ratio of the LPs shares to the corresponding total shares, equals the percentage of the assset in the pool the LP owns and therefore can withdraw.

We will prove in the next section that the shares of each LP are linear combinations of his initial shares (we assume for convenience that his deposit happened at time t=0):

$$s_{x,i}(t) = M_{x,x}(t)s_{x,i}(0) + M_{x,y}(t)s_{y,i}(0) \tag{17}$$
$$s_{y,i}(t) = M_{y,x}(t)s_{x,i}(0) + M_{y,y}(t)s_{y,i}(0) \tag{18}$$

The coefficients are the same for all LPs, therefore if we can compute them efficiently we will be able to compute every LP's shares at every moment. We will also prove that these coefficients should be updated as follows, depending on the action (short trade, long trade or liquidity deposit/withdrawal).

For long swaps (i.e. when a user gets an amount of X asset from the pool and deposits an amount of Y asset):

$$M_{x,x}(t+1) = M_{x,x}t) \tag{19}$$
$$M_{x,y}(t+1) = M_{x,y}(t) \tag{20}$$
$$M_{y,y}(t+1) = M_{y,y}(t) + M_{x,y}(t) \cdot A_y(t+1) \tag{21}$$
$$M_{y,x}(t+1) = M_{y,x}(t) + M_{x,x}(t) \cdot A_y(t+1) \tag{22}$$

For short swaps (i.e. i.e. when a user gets an amount of Y asset from the pool and deposits an amount of X asset)

$$M_{x,x}(t+1) = M_{x,x}(t) + M_{y,x}(t) \cdot A_x(t+1) \tag{23}$$
$$M_{x,y}(t+1) = M_{x,y}(t) + M_{y,y}(t) \cdot A_x(t+1) \tag{24}$$
$$M_{y,y}(t+1) = M_{y,y}t) \tag{25}$$
$$M_{y,x}(t+1) = M_{y,x}(t) \tag{26}$$
$$\tag{27}$$

For liquidity deposits and withdrawals only the total shares are updated and not the individual shares (with the exception of the LP that deposits or withdraws, of course), therefore in this case the coefficients should not be updated. It will be very helpful to express these equations using 2x2 matrices. We define:

$$M(t) = \begin{bmatrix} M_{x,x} & M_{x,y} \\ M_{y,x} & M_{y,y} \end{bmatrix}(t) \tag{28}$$

Using this notation, the equations describing the updates of the coefficints can be written as:

$$M(t+1) = A(t+1) \cdot M(t) \tag{29}$$

$$M(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{30}$$

where:

$$A(t+1) = \begin{bmatrix} 1 & 0 \\ A_y(t+1) & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & A_x(t+1) \\ 0 & 1 \end{bmatrix} \tag{31}$$

if the swap is long or short respectively. The formulas for A_x, A_y are given in the next section. The important thing is that they depend only on global variables (total shares, total deposits) and on the exchanged amounts and therefore are the same for all LPs. Applying recursively this formula we get theformula for $M(t)$:

$$M(t) = A(t) \cdot A(t-1) \dots A(1) \tag{32}$$

This formula can be used for the computation of the shares of an LP that has deposits since time $t = 0$. If an LP deposits at time t_0 the matrix coeeficient for him should at time $t \geq t_0$ be:

$$M_{t_0}(t) = A(t) \cdot A(t-1) \dots A(t_0+1) = M(t) \cdot M^{-1}(t_0) \tag{33}$$

Putting all these together, the steps for computing the shares for the LPs are:

- For the first LP we are minting shares equal to the deposited amounts (or any multiple of them).
- At every action we update the matix M using (32).
- When a new LP deposits some liquidity we are computing his initial deposits $(s_{x,i}(0), s_{y,i}(0))$ and take a snapshot of the matrix M. Whenever we need to compute the shares of this LP, we should first compute the coefficient matrix using (33) and then use (17) replacing M by M_{t_0}.

4.2 Proof that the algorithm correctly computes the shares

We will prove that (17) hold for every t by induction. For $t = 0$ this is trivial, since:

$$s_{x,i}(0) = 1 \cdot s_{x,i}(0) + 0 \cdot s_{y,i}(0) \tag{34}$$
$$s_{y,i}(0) = 0 \cdot s_{x,i}(0) + 1 \cdot s_{y,i}(0) \tag{35}$$

We assume that the formulas are valid for t and we will prove that they are also valid for $t+1$. The shares should be updated only after an action. We will study each action separately.

Long swaps: A user buys a dx amount of X-asset from the pool by depositing a dy amount of Y-asset. Of course dx and dy are related through the formula of the AMM, but a nice feature of our approach is that it does not depend on the details of the AMM. The X-shares of the LPs do not change. What changes is the amount of X-asset corresponding to these shares. Each LP loses $dx\frac{s_{x,i}(t)}{S_x(t)}$ and therefore he should earn the same percentage in Y-assets. $y(t)$ is increased by dy. We increase the total Y-shares by:

$$\delta S_y = S_y(t)\frac{dy}{y(t)} \Rightarrow \tag{36}$$

$$S_y(t+1) = S_y(t)\cdot\left(1+\frac{dy}{y(t)}\right) \tag{37}$$

The i LP owns $\delta s_{y,i}$ of these extra δS_y shares:

$$\delta s_{y,i} = \delta S_y \frac{s_{x,i}}{S_x(t)} \tag{38}$$

$$= s_{x,i}(t)\frac{S_y(t)}{S_x(t)}\frac{dy}{y(t)} \tag{39}$$

$$= s_{x,i}(t)A_y(t+1) \tag{40}$$

where

$$A_y(t+1) = \frac{S_y(t)}{S_x(t)}\frac{dy}{y(t)} \tag{41}$$

Therefore:

$$\begin{aligned} s_{y,i}(t+1) &= s_{y,i}(t) + \delta s_{y,i} \\ &= M_{y,x}(t)s_{x,i}(0) + M_{y,y}(t)s_{y,i}(0) + (M_{x,x}(t)s_{x,i}(0) + M_{x,y}(t)s_{y,i}(0))\cdot A_y(t+1) \quad \text{(t inductive step)} \\ &= (M_{y,x}(t) + M_{x,x}(t)\cdot A_y(t+1))\, s_{x,i}(0) + (M_{y,y}(t) + M_{x,y}(t)\cdot A_y(t+1))\, s_{y_i}(0) \end{aligned}$$

Short swaps: The idea is identical, we only have to replace x by y and vice versa.

Liquidity deposits:When a new LP deposits dx and dyamounts in the pool, the shares of the old LPs do not change. Only the total shares change. For this new LP, the protocol should mint initial shares (assuming that the time of the deposit is $t+1$):

$$s_{x,i}(0) = S_x(t)\frac{dx}{x(t)} \tag{42}$$

$$s_{y,i}(0) = S_y(t)\frac{dy}{y(t)} \tag{43}$$

and the total shares increase, as follows:

$$S_x(t+1) = S_x(t)\left(1+\frac{dx}{x(t)}\right) \tag{44}$$

$$S_y(t+1) = S_y(t)\left(1+\frac{dy}{y(t)}\right) \tag{45}$$

5 Fees

5.1 Fees for opening/closing a position

Whenever a user opens, closes, or modifies a position, a small fee is applied. This fee is calculated as a percentage of the traded amount and is always denominated in the quote asset -vStable, in our protocol. By using a stable quote asset, fee revenues remain unaffected by market fluctuations.

The fees collected are divided among: the protocol, the insurance fund, and the LPs. The portion allocated to the protocol supports ongoing development and maintenance. The insurance fund receives a share to cover potential losses in cases of bad debt when positions are closed. The remaining portion goes to LPs and is added directly to the liquidity pool.

In the case of short positions, where vStable is taken from the pool, the LP fee distribution is based on each LP's vStable shares. For long positions, the distribution is instead based on their vAsset shares. This mechanism ensures LPs are fairly compensated according to the type of position being taken and the assets being utilized.

5.2 Fees for liquidity deposits that imbalance the pool

The protocol we are designing allows for liquidity deposits in any ratio $r = \frac{Dy}{Dx}$, where Dx represents the amount of vAsset and Dy the amount of vStable deosited by an LP in the pool. This includes one-sided liquidity deposits ($r = 0$ or ∞).This flexibility is made possible by two key features of the protocol:

- *The dynAMM*: This enables exchanges to occur near the market price, even when the ratio of total deposits in the pool differs from the market price.
- *Two-Shares-Per-LP Mechanism*: This approach efficiently tracks the continuously changing shares of LPs for imbalanced deposits.

This flexibility is an advantage of our protocol, as it allows LPs to implement strategies fitting their risk profiles and market expectations. For example, by depositing one-sided liquidity (e.g., only vAsset), an LP primarily gains short exposure, since traders can only buy their vAsset (although, over time, as trades occur, the LP's shares will naturally shift, and they will gradually own a mix of both vAsset and vStable).

While this flexibility benefits LPs and allows for diverse trading strategies, it can also lead to imbalanced pools—where the ratio of total vStable to total vAsset in the pool deviates significantly from the market price. Although our protocol remains robust under such conditions, continuing to offer prices close to the market price for most trades, heavily imbalanced pools can result in higher slippage.

We are implementing a fee mechanism for liquidity deposits that incentivizes LPs to help maintain a balanced pool. The pool is considered balanced when the ratio of vStable to vAsset in the dynAMM matches the current market price. To encourage this equilibrium, higher fees will be applied to deposits that increase the imbalance, while minimal or even zero fees will be charged for deposits that move the pool closer to balance.

6 Funding payments

Funding payments are a core component of any perpetual protocol. While different protocols use various formulas to calculate funding rates (see [5] for a summary), they generally share two key features:

- *Direction of Payments*: If the funding rate is positive, long positions pay short positions; if negative, short positions pay long positions. This is just a convention.
- *Balancing the System*: Funding payments are designed to help balance the system.Although, what it means "balancing of the system" varies for each protocol.

Most commonly, funding payments help keep the perpetual price (mark price) aligned with the index price (market price of the underlying asset). For instance, if the perpetual price is too high, funding becomes positive, encouraging shorts and discouraging further longs -helping prices converge.

In our protocol, this alignment is already achieved through the dynAMM, which keeps the perpetual price close to the market price without relying on funding payments. This allows us to offer lower funding costs, giving us a competitive edge.

However, funding payments serve broader purposes:

- They incentivize traders to take the less popular side.
- In balanced markets, the funding payments lead to positive returns for LPs; in directional markets, they help reduce LPs' losses.
- They also reduce system exposure by discouraging the most popular positions and encouraging balanced participation, limiting the risk LPs must absorb.

6.1 Definitions

In its most general form the fundyng payment a position holder -either a trader or an LP- should make (or receive) is given by the formula:

$$\textit{Funding fees for the position} = \textit{position value} \cdot \textit{Funding rate} \tag{46}$$

where

- positive funding fees means that the position should pay these fees and negative that the position should receive the fees.
- position value: equals *(position size · price)*, and is positive for long and negative for short positions
- Funding rate: it is a global quantity (the same for every position), changes with time and trades and liquidity changes, is positive if longs should pay shorts and negative if shorts should pay longs

6.1.1 The time component of the funding rate

First, we need to determine how frequently the funding payments should be calculated and accounted for. To clarify, this does not necessarily mean that actual payments must occur immediately. The funding rate can be expressed as:

$$\textit{Funding rate} = \frac{\textit{Funding coefficient} \cdot \Delta t}{\textit{Funding Interval}} \tag{47}$$

where

- *Funding coefficient*:T his is not a standard term and will be formally defined in a later section. It represents a global variable that updates with every user action (e.g. trades or liquidity deposits).
- *Funding interval*: When time equal to the Funding interval has passed, the total fees (that should be paid) equal ((Funding coefficient) · (total value of the most popular position type)). The standard choice for the funding interval is 1 day (computed in seconds), but we could decrease it to increase the rate.
- Δt: Represents the elapsed time (in seconds) since the previous user interaction (e.g. trade or liquidity update). Our system uses a continuous funding model rather than discrete time intervals. This approach minimizes the risk of manipulation -such as timing position changes around scheduled funding events- and ensures smoother, more consistent fee accrual. On each user action, the funding rate is computed using the current funding coefficient and Δt, after which the funding coefficient is updated to reflect the new state.

6.1.2 Position sizes

If FF_j are the funding fees for a position j (this could be either a trader or an LP position), then the sum of the funding fees should be zero (total sum of positive fees should equal minus the total sum of negative i.e. payments of funding fees should cancel out and there should be no need for extra funds). This should be true for every funding payment (after every action) and not only for total payments over a period. This is true iff:

$$\begin{aligned}\sum_j FF_j &= \sum_j \textit{Funding rate} \cdot \textit{Value of position } j \\ &= \textit{Funding rate} \cdot \sum_j \textit{Value of position } j\end{aligned}$$

i.e. iff the sum of the position values (positive for long and negative for shorts) is zero.

The position value can be computed by multiplying the position size (that we have to define) by the price (average price over the time period between two consecutive actions -check the algorithms in the next section-):

$$\begin{aligned}\sum_j FF_j &= \textit{Funding rate} \cdot \sum_j \textit{Value of position } j \\ &= \textit{Funding rate} \cdot \textit{price} \cdot \sum_j \textit{Size of position } j\end{aligned}$$

Therefore, the sum of the funding payments will be zero if and only if the sum of the position sizes is zero. This holds true because positions are opened by exchanging amounts in a vAMM. However, let's explore this further to clarify the details. A long trader will borrow vStable and exchange them in the vAMM for vAssets. Therefore at the end his vAsset debt will be zero (he didn't borrow any vAssets) and his vAssets balance will be the the output of the vAMM. In the contrary a short trader will borrow vAssets and exchange them for vStable. In both cases the difference:

$$\textit{Position size} = \textit{vAsset balance} - \textit{vAsset debt} \tag{48}$$

captures the size of the position, that will be positive for longs and negative for shorts.

		Debt	Balance
long trader	vAssets	0	dx
	vStable	dy	0
short trader	vAssets	dx	0
	vStable	0	dy

The debt of a position changes only if the owner of the position interacts with the protocol to close or update the position. This is true for both traders and LPs. The balance of a trader can also only change by a direct interaction of the trader with the protocol, therefore a trader's position has constant size (between two interactions of the user with the protocol) and this simplifies the computation of the funding payments as we will see in the next section. The balance, and therefore the position size, of an LP constntly changes, as traders open positions transforming the liquidity of the pool from vStable to vAssets and vice versa. But we can still effectively compute these balances, using the shares:

$$\textit{j LP's balance at time } t = \frac{s_{x,j}(t) \cdot x(t)}{S_x(t)}$$

When an LP deposits liquidity in the pool, this can be splitted into two actions: borrowing vAssets and vStable from the protocol (these amounts are his debts) and depositing these amounts to the pool (these are his balances). Therefore immediately after the deposit (until the first trade) the position of the LP is zero. Therefore initially all the positions, and therefore their sum, is zero. Since each trade is executed through an exchange in the dynAMM, it is clear that opening a new position simultaneously increases the LPs' total position by an equal amount with the opposite sign. As a result, all trades preserve the zero-sum of the funding payments.

6.1.3 A new approach to funding rates using the total exposure

Since the primary purpose of funding payments in our system is to reduce total exposure, we define the funding coefficient to be proportional to total exposure as follows:

$$\textit{Funding coefficient} = \frac{\textit{Total exposure}}{c \cdot \textit{Total liquidity}} \tag{49}$$

where

- c: a constant between 0 and 1 (percentage of the pool liquidity)
- *Total liquidity*: total liquidity in the pool. The exact definition is not really crucial. We just use it to be able to define when an certain exposure should be considered small or large. It can simply be computed as $p \cdot x + y$.Where x and y are the amounts in the pool and p the market price.
- *Total exposure*: Sum of long minus sum of short positions **only for traders**. Since, as we have shown, the sum of all the positions is zero, this is also equal to minus the total position of the LPs. Setting the funding rate proportional to this, the traders are incentivized to decrease the total exposure of the LPs.

6.2 An algorithm for efficiently computing the funding payments of traders and LPs

We denote by $F(t)$ the funding rate at time t (actually it will be the funding rate times the market price i.e. funding fees = position size · price · funding rate and we store to F the product price times funding rate). The initial value of the funding rate should be zero, since there are no open positions by traders at the protocol initially. The funding rate should change after each trade and liquidity deposit/withdrawal, the LP's positions change after each swap or if the owner of the LP position deposits/withraws and position of a trader changes only when he updates it and not when other users interact with the protocol. After each trade we also compute the total exposure $Exp(t) = Exp_s(t) \cdot p$ (where Exp_s is the size in terms of vAssets of the total exposure i.e. balance of vAssets held by long traders minus debt of vAssets of short traders) and total liquidity $L(t)$. We denote by T the funding interval.

- Funding paymements for traders:
  - The user opens his position at t_0 and the previous trade happened Δt seconds ago. The size of the position is Dx vAssets (positive for long and negative for short).
  - We update the funding rate:

$$\begin{aligned} F(t_0) &= F(t_0 - \Delta t) + \Delta F(t_0 - \Delta t) & (50) \\ &= F(t_0 - \Delta t) + \bar{p}\frac{Exp_s(t_0 - \Delta t) \cdot \bar{p} \cdot \Delta t}{c \cdot L(t_0 - \Delta t) \cdot T} & (51) \end{aligned}$$

    and take a snapshot and store it at the user's state info. $\bar{p}$ is the average market price for the Δt time interval. We can get it from the price oracle. $L(t_0 - \Delta t) = \bar{p} \cdot x(t_0) + y(t_0)$.
  - We update the total exposure size:

$$Exp_s(t_0) = Expt_s(t_0 - \Delta t) + Dx \qquad (52)$$
$$(53)$$

- When, at some time t, the trader decides to close or update his position, we first compute the current funding rate $F(t)$ and his total funding fees (positive means he has to pay, negatve means he should receive funding payments) as:

$$\textit{Total trader's funding fees} = Dy \cdot (F(t) - F(t_0))$$

  These fees should be added to his pnl from price changes and then settled, by pairing the trader with another user with opossite side unsettled pnl (check the notes with the example about funding payments). We then make the changes in the pool and update the funding rate as described in the previous step.

- Funding payments for LPs:
  - The funding rate is computed and updated as in the case of traders'. When an LP deposits or withdraw liquidity, the total exposure does not change (it is just distributed to the remaining LPs), but the funding rate changes because of the change of the total liquidity. The tricky part is that the LP position change not only when an LP interacts directly with the protocol, but on every trade.

– If two consecutive actions happen at times t_i and t_{i-1}, then for this time interval the total position hold by the LPs equals $-Exp(t_{i-1})$ and the funding payment corresponding to all the LPs are:

$$\begin{aligned} FP_{LP}(t_i) &= -Exp(t_{i-1}) \cdot \frac{Exp(t_{i-1}) \cdot (t_i - t_{i-1})}{c \cdot L(t_{i-1}) \cdot T} \\ &= -\frac{Exp^2(t_{i-1}) \cdot (t_i - t_{i-1})}{c \cdot L(t_{i-1}) \cdot T} \end{aligned}$$

Observe that it is always negative (i.e. LPs in total will always receive and not pay funding fees-although some individual LPs may happen to pay fees-) as expected.

– The extra difficulty in the computation of the funding payments of LPs, compared to the funding payments for traders, is that an LP position chaanges not only when the LP actively interacts with the protocol, but also on every trade (since when a trader opens a position, the LPs take passively the opossite position). We denote by $FP_j(t_i)$ the total funding payments of the j LP till time t_i (positive if he has to pay, negative if he should receive a payment).

$$\begin{aligned} FP_j(t_i) &= FP_j(t_i - 1) + (j \textit{ LP's position size}) \cdot \Delta F(t_{i-1}) \\ &= FP_j(t_i - 1) + (x_j(t_{i-1}) - d_j) \cdot \Delta F(t_{i-1}) \\ &= FP_j(t_i - 1) + x_j(t_{i-1}) \cdot \Delta F(t_{i-1}) - d_j \cdot \Delta F(t_{i-1}) \end{aligned}$$

where $x_j(t_{i-1})$ is the vAsset balance of the j LP at time t_{i-1} and d_j is his vAsset debt (the balance constanlty change after each trade, but the debt changes only if the LP interacts with the protocol). Recall that we can compute effectively the liquidity of each LP at every time using the shares:

$$\begin{aligned} x_j(t_{i-1}) &= sx, j(t_{i-1}) \frac{x(t_{i-1})}{S_x(t_{i-1})} \\ &= (M_{x,x}(t_{i-1}) \cdot s_{x,j}(0) + M_{x,y}(t_{i-1} \cdot s_{y,j}(0))) \cdot \frac{x(t_{i-1})}{S_x(t_{i-1})} \end{aligned}$$

Therefore:

$$\begin{aligned} FP_j(t_i) = FP_j(t_i - 1) + \frac{M_{x,x}(t_{i-1}) \cdot x(t_{i-1})}{S_x(t_{i-1})} \Delta F(t_{i-1}) \cdot s_{x,j}(0) + \\ + \frac{M_{x,y}(t_{i-1}) \cdot x(t_{i-1})}{S_x(t_{i-1})} \Delta F(t_{i-1}) \cdot s_{y,j}(0) - d_j \cdot \Delta F(t_{i-1}) \end{aligned}$$

Using matrices, this can be written as:

$$FP_j(t_i) = FP_j(t_{i-1}) + [1 \quad 0] B(t_i) M(t_{i-1}) \cdot \begin{bmatrix} s_{x,j}(0) \\ s_{y,j}(0) \end{bmatrix} - d_j \cdot \Delta F(t_{i-1}) \tag{54}$$

where

$$B(t_i) = \begin{bmatrix} \frac{x(t_{i-1})\cdot\Delta F(t_{i-1})}{S_x(t_{i-1})} & 0 \\ 0 & 0 \end{bmatrix} \tag{55}$$

– Therefore we should store a global matrix sum (actually we need only his first row -two sums-):

$$G(t_i) = G(t_{i-1}) + B(t_i)M(t_{i-1}) \tag{56}$$

– Take two snapshots: $G(t_0)$, $F(t_0)$ whenever an LP deposits liquidity (two extra snapshots along with the snapshot of the matrix M, used for the computation of the shares). Whenever we want to compute the total collected funding payments for the LP we use the formula:

$$FP_j = [1 \quad 0](G(t) - G(t_0))M^{-1}(t_0)\begin{bmatrix} s_{x,j}(0) \\ s_{y,j}(0) \end{bmatrix} - d_j \cdot (F(t) - F(t_0)) \tag{57}$$

7 Liquidations

To maintain system solvency, positions must be liquidated when the ratio of collateral to position value falls below a specified threshold. Liquidation is the process by which a user's position is sold to another participant at a discount. This discount acts as a fee imposed on the liquidated user, incentivizing traders to maintain sufficient collateral and avoid liquidation.

7.1 Liquidation criteria

A position becomes eligible for liquidation based on its margin ratio (MR), calculated as:

$$MR = \frac{account\ value}{position\ value}$$

where:

- *Account value*: The USD value of the deposited collateral minus the total PnL for the position and any funding payments.
- *Position Value*: The size of the position multiplied by the index price of the underlying asset. Here, the index price is a time-weighted average of the market price over a few minutes (around 10 minutes). We should avoid using the actual (spot) market price to avoid liquidations due to temporary price fluctuations.

We define several margin thresholds:

$$MM_1 > MM_2 > ... > MM_n$$

and corresponding percentages:

$$p_1 < p_2 < ... < p_n$$

If a position's margin ratio satisfies $MM_{i+1} \leq MR < MM_i$, then up to p_i percent of the position size can be liquidated.

For large positions, full liquidation by a single user may not be feasible. Therefore, partial liquidations are allowed, enabling multiple users to collectively liquidate portions of the position up to the total allowable size.

7.2 Liquidation steps

The liquidation process is symmetric for both long and short positions. We describe the liquidation of a long position below; the short case follows analogously.

User A holds a long position of size dx_A with margin ratio $MM_{i+1} \leq MR < MM_i$ and therefore it is liquidatable up to size $dx_{max} = p_i \cdot dx_A$. User B decides to buy $dx \leq dx_{max}$ of it. Let's call d the liquidation discount. This can be either fixed or could depent on the margin ratio of A i.e.

$$d = \frac{d_i}{2}\left(1 + \frac{MM_i - MR}{MM_i - MM_{i+1}}\right)$$

where d_{i+1} should be higher than d_i and , if we prefer continuous increase in the liquidation fee as the MR increases, $d_{i+1} = 2 \cdot d_i$.

The steps for the liquidation are as follows:

1. We compute (using the formulas we have described in a previous report)user's A funding fees and add them to the total funding fees he should pay. We could also force him pay a fee to the insurance fund (equal to a percentage of the USD value of dx).

2. To incentivize liquidators, the purchase price of the position should be set so that, if they choose to close it immediately after acquisition, they secure a profit based on the discount d (excluding any trading fees associated with closing the position).

   We compute (using the vAMM formulas of a previous report) the amount of vStable dy' that we would get if we exchanged dx vAsset, without applying trading fees (i.e. the ampunt of vStable that would get someone holding a position of size dx). This is just for accounting and we do not add or withdraw anything from the vAMM pool. The amount user B should pay to user A to get his position is $dy = (1 - d) \cdot dy'$. (The idea is that if user B immediately closes this new position he will get -minus trading fees- dy'. Therefore he earned $dy' - dy = d \cdot dy'$).

3. If user A is an LP, his vAssets are deposited as liquidity in the pool, so we cannot immediately transfer them to user B. We should first force user A to withraw (part of) his vAsset liquidity from the pool (this will increase

the price of vAsset, therefore the above estimation is an underestimation for user B i.e. if he closes the newly acquired position immediately he will get get even more -excluding trading fees-).If dx is the whole position of user A, we force him to withdraw all his liquidity. If it is just a percentage of his total position, we force him to withdraw the same percentage of his vAsset liquidity i.e. an amount dx' s.t.

$$\frac{dx'}{total\ vAsset\ liquidity\ of\ user\ A} = \frac{dx}{user\text{'}s\ A\ total\ position}$$

4. We implement the transfer/sale of the position ie. we change the vStable and vAsset balances of A and B respectively by $(dy, -dx)$ and $(-dy, dx)$.If user A was an LP, we redeposit his remaining $dx' - dx$ in the pool as liquidity taking new snapshots.

5. We take all the necessary snapshots for both users.

6. If the margin ratio of the liquidator is below a threshold we revert.

7. If user's A collateral cannot cover his losses (bad debt), the insurance fund should cover them.

References

[1] M. Egorov, *StableSwap- efficient mechanism for stablecoin liquidity*, https://docs.curve.fi/assets/pdf/whitepaper_stableswap.pdf

[2] M. Egorov, *Automatic market-making with dynamic peg*, https://docs.curve.fi/assets/pdf/whitepaper_cryptoswap.pdf

[3] B. Krishnamachari, Q. Feng and E. Grippo, *Dynamic Automated Market Makers for Decentralized Cryptocurrency Exchange*, 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Sydney, Australia, 2021, pp. 1-2, https://anrg.usc.edu/www/papers/dynamicautomation.pdf

[4] H. Adams, *Uniswap whitepaper*, https://hackmd.io/C-DvwDSfSxuh-Gd4WKE_ig#Introduction

[5] *Funding rates:under the hood*, https://medium.com/derivadex/funding-rates-under-the-hood-352e6be83ab

Disclaimer

Educational and Scientific Purpose: This whitepaper and all contents herein are provided strictly for scientific, educational, and informational purposes only. The technical designs, financial models, and mechanisms described in this paper represent theoretical constructs and should not be interpreted as financial, investment, legal, or tax advice.
Intellectual Property Notice: The implementation of Denaria's design into smart contracts is protected under a Business Source License. To learn more about the Business Source License visit mariadb(dot)com.
No Guarantees or Warranties: The information provided in this paper is presented "as is" without any representations or warranties, express or implied. Denaria makes no guarantees regarding the accuracy, completeness, timeliness, suitability, or validity of the information contained herein. All forward-looking statements, projections, or future plans mentioned in this document are speculative in nature.
Risk Disclosure: Decentralized finance (DeFi) protocols, particularly those involving perpetual derivatives, carry significant risks including but not limited to: smart contract vulnerabilities, market volatility and liquidation risks, regulatory uncertainty, oracle failures or manipulations, economic design flaws, and governance attacks. Users and developers who choose to interact with any implementation of the designs described in this paper do so entirely at their own risk.
Regulatory Compliance: This paper does not constitute an offer to sell, a solicitation of an offer to buy, or a recommendation of any cryptocurrency, digital asset, or any other financial instrument. Users must ensure their compliance with all applicable laws and regulations in their respective jurisdictions before interacting with any implementation of the designs described.
Independence of Action: No part of this document should be interpreted as encouraging or soliciting any person to engage in any action or transaction. All decisions related to the use of any protocol or product derived from this paper should be made independently after conducting appropriate due diligence.
Living Document: This whitepaper is subject to updates, modifications, and amendments without prior notice. Readers are responsible for ensuring they are reviewing the most current version.