Curve Stableswap: From Whitepaper To Vyper

The stableswap invariant was derived by Michael Egorov and promulgated in the whitepaper, "StableSwap - efficient mechanism for Stablecoin liquidity". The whitepaper clearly explained the invariant and its implications for DeFi; however, there are differences with how it is implemented in practice, currently across dozens of live contracts across Ethereum and other layer 2s and chains.

In particular, implementation of fees, both for exchanges and adding/removing liquidity, is not explained in the whitepaper. Also, the actual solution procedure for the invariant, particularly in integer arithmetic, is not given.

The practitioner seeking to understand the live functionality of the stableswap pools must look toward the vyper code for help, which while very readable, has minimal comments and explanation (indeed some comments are even wrong!). To understand the vyper code, the reader must have a solid grasp of the whitepaper in order to translate to the appropriate variables and understand various tweaks needed for implementation.

This note seeks to close the gap between the whitepaper and the vyper contracts. It seeks to give a consistent derivation of the necessary mathematics, using the notation and language of the contracts.

Preliminaries (notation and conventions)

Stableswap equation

This is the original stableswap equation:

$$A \cdot n^n \sum_i x_i + D = A \cdot n^n \cdot D + \frac{D^{n+1}}{n^n \prod_i x_i}$$

In the vyper code, the amplification constant A actually refers to $A \cdot n^{n-1}$, so the equation becomes:

$$A \cdot n \sum_i x_i + D = A \cdot n \cdot D + \frac{D^{n+1}}{n^n \prod_i x_i}$$

This is the form we use for all our derivations.

Coin balances

We denote the coin balances (as in the contracts) with x_i, x_j etc. In the context of a swap, i is the "in" index and j is the "out" index.

The quantities though are in special units. They are *not* native token units. For example, if x_i represents the USDC amount, one whole token amount would not be 1000000 as might be assumed from the 6 decimals for USDC. Instead

x_i would be 1000000000000000000 (18 zeroes). All the x balances should be assumed to be in the same units as D. For lack of a better term, sometimes we will call these *virtual* units (as the amount of D per LP token is often referred to as "virtual price") and we will call the x balances *virtual balances*.

While putting balances into virtual units often involves only a change of decimals, this is not the correct way of thinking about the process. The stableswap equation assumes a 1:1 peg between coins. This means the balances being used must reflect the proper value in the common units of D being used. For the example of USDC, this means adjusting simply the decimals. For a rebasing token however, it may not be. Indeed, for metapools, when exchange the basepool's tokens for the main stable, the basepool token conversion into D units must take into account the accrued yield. This is done by multiplying the amount by the basepool virtual price.

Solving for D

Since the arithmetic mean is greater than the geometric mean (unless the balances x_i are equal, in which case the means are identical), the form of the equation suggests that there ought to be a D in-between the means that satisfies the equation.

To see this rigorously, we use the auxiliary function:

$$f(D) = A \cdot n \cdot (D - A \cdot n \sum_i x_i) + D \cdot (\frac{D^n}{n^n \prod_i x_i} - 1)$$

Let $P = n \ (\prod_i x_i)^{\frac{1}{n}}$ and $S = \sum_i x_i$. This is a continuous function (away from zero balances) with $f(P) < 0$ and $f(S) > 0$. So there is a D such that $P < D < S$ and $f(D) = 0$. In particular, note

$$f'(D) = A \cdot n + (n+1)\frac{D^n}{n^n \prod_i x_i} - 1$$

the derivative of f, is positive (assuming $A >= 1$), so f is strictly increasing and there is a unique D that solves $f(D) = 0$.

Newton's method

The stableswap contracts utilize Newton's method to solve for D. It is easy to check $f'' > 0$, i.e. f is convex. An elementary argument shows that this guarantees convergence of Newton's method starting with initial point S to the solution.

The vyper code (from 3Pool) is:

```
@pure
@internal
def get_D(xp: uint256[N_COINS], amp: uint256) -> uint256:
    S: uint256 = 0
    for _x in xp:
        S += _x
    if S == 0:
        return 0

    Dprev: uint256 = 0
    D: uint256 = S
    Ann: uint256 = amp * N_COINS
    for _i in range(255):
        D_P: uint256 = D
        for _x in xp:
            D_P = D_P * D / (_x * N_COINS)  # If division by 0, this will be borked: only wi
        Dprev = D
        D = (Ann * S + D_P * N_COINS) * D / ((Ann - 1) * D + (N_COINS + 1) * D_P)
        # Equality with the precision of 1
        if D > Dprev:
            if D - Dprev <= 1:
                break
        else:
            if Dprev - D <= 1:
                break
    return D
```

This code is used with minimal difference between all the stableswap contracts. Later versions choose to revert if the 255 iterations are exhausted before converging.

The iterative formula is easily derived:

$$
\begin{aligned}
d_{k+1} &= d_k - \frac{f(d_k)}{f'(d_k)} \\
&= d_k - \frac{An(d_k - A\sum_i x_i) + d_k(\frac{d_k^n}{n^n \prod_i x_i} - 1)}{\frac{(n+1)d_k^n}{n^n \prod_i x_i} + An - 1} \\
&= \frac{An\sum_i x_i + \frac{nd_k^{n+1}}{n^n \prod_i x_i}}{\frac{(n+1)d_k^n}{n^n \prod_i x_i} + An - 1} \\
&= \frac{AnS + nD_p(d_k)}{\frac{D_p(d_k)}{d_k} + An - 1} \\
&= \frac{(AnS + nD_p(d_k))d_k}{D_p(d_k) + (An-1)d_k}
\end{aligned}
$$

where $S = \sum_i x_i$ and $D_p(d_k) = \frac{d_k^{n+1}}{n^n \prod_i x_i}$

Quadratic convergence Convergence is easily argued based on convexity of f. However we need much better than that, we need at least linear convergence, otherwise 255 iterations is not sufficient. Also, in practice, exceeding more than half a dozen iterations is not sufficiently gas efficient enough to be competitive.

Integer arithmetic

The swap equation

The stableswap equation allows you to solve for any coin balance given the other balances and the value of D. This is a fundamental property needed for enabling swap functionality. Since this is not derived in the whitepaper, we go through it here.

The stableswap equation can be re-written in the form:

$$An\left(x_j + \sum_{k\neq j} x_k\right) + D = AnD + \frac{D^{n+1}}{n^n x_j \prod_{k\neq j} x_k}$$

where j is the out-token index.

Let's denote $\sum_{k\neq j} x_k$ by S' and $\prod_{k\neq j} x_k$ by P'.

Then we have, after some re-arranging

$$x_j + S' + \frac{D}{An} = D + \frac{D^{n+1}}{An^{n+1} x_j P'}$$

This becomes

$$x_j^2 + \left(S' + \frac{D}{An} - D\right) x_j = \frac{D^{n+1}}{An^{n+1} P'}$$

or

$$x_j^2 + bx_j = c$$

where $b = S' + \frac{D}{An} - D$ and $c = \frac{D^{n+1}}{An^{n+1} P'}$.

This quadratic equation can be solved by Newton's method:

$$
\begin{aligned}
x_j &:= x_j - \frac{x_j^2 + bx_j - c}{2x_j + b} \\
&:= \frac{x_j^2 + c}{2x_j + b}
\end{aligned}
$$

Note the actual vyper code cleverly defines b as our b without the $-D$ term. This allows b to be defined as a `uint256` since otherwise it could be negative (although of course $2x_j + b$ is always positive).

Fees

Exchange

Adding and removing liquidity

Balanced deposits and withdrawals

Removing one coin

Useful formulas

Price

Slippage

Depth