YieldSpace: An Automated Liquidity Provider for yTokens

Allan Niemerg allan@yield.is

Dan Robinson dan@paradigm.xyz

Lev Livnev lev@dapp.org

August 2020
WORKING DRAFT

Abstract

The Yield Protocol introduced the concept of “yTokens,” a fungible token similar to a zero-coupon bond that allows fixed-rate borrowing and lending and interest rate discovery on the Ethereum blockchain. While yTokens can be traded anywhere, including automated liquidity providers like Uniswap, the formulas used by those systems are not optimized for yTokens, resulting in higher price impact and fees for traders of close-to-maturity yTokens, and predictable losses due to arbitrage for liquidity providers. This paper introduces a new formula for automated liquidity provision, the “constant power sum invariant,” which incorporates time to maturity as an input and ensures that the liquidity provider offers a constant interest rate—rather than price—for a given ratio of its reserves. The paper also introduces a generally useful methodology that can be used to derive invariants for pools with desired properties.

1 Introduction

In the Yield Protocol paper [1], the authors introduced yTokens, a synthetic token that is redeemable for a target asset after a fixed maturity date. The price of a yToken floats freely before maturity, and that price implies a particular interest rate for borrowing or lending that asset until the yToken’s maturity. yTokens can be traded anywhere that supports ERC20 trading, including existing protocols for pooled liquidity provision like Uniswap. For yTokens with a target asset that is itself an ERC20, the most obvious pair for it to be traded against would be that target asset.

For this paper, we will focus on yDai, the first token that is being implemented using the Yield Protocol [2] (although the design described will likely work for other yTokens). yDai are yTokens with Dai as the target asset. yDai may be redeemed one-for-one for Dai at maturity. When far from maturity, yDai could trade at varying prices. When close to maturity, yDai is likely to trade close to 1 Dai, regardless of the interest rate, because of its impending redeemability for Dai.

Most popular protocols for swapping tokens on Ethereum use pooled liquidity provision and set their prices automatically based on a formula. These formulas are typically expressed as *invariants* specifying some fixed relation between the reserves of the two assets. For example, Uniswap uses the constant product invariant $x \cdot y = k$, where x and y are the pool's reserves of the two assets. These invariants can be chosen based on the properties that they guarantee the pool's reserves will always obey.

The invariants used by existing automated liquidity providers are not optimized for yTokens. The constant product formula used by Uniswap [3] and Balancer [4] would likely be capital-inefficient for yTokens that are close to maturity. The constant sum formula used by mStable [5] gives up on price discovery entirely, and thus would likely discourage trading (since the price of yDai will vary over time and will almost always be below 1 Dai before maturity). Hybrid formulas like Curve's compromise between these two models, but do not tell us how to adjust their parameters based on time to maturity.

Indeed, these formulas are typically pure functions of reserves (and other constant parameters), and none of them incorporate time. For given reserve balances, the price offered remains constant as time passes. These pools would therefore suffer from predictable losses to arbitrage as maturity approaches and yDai price moves toward 1, even if interest rates do not change.

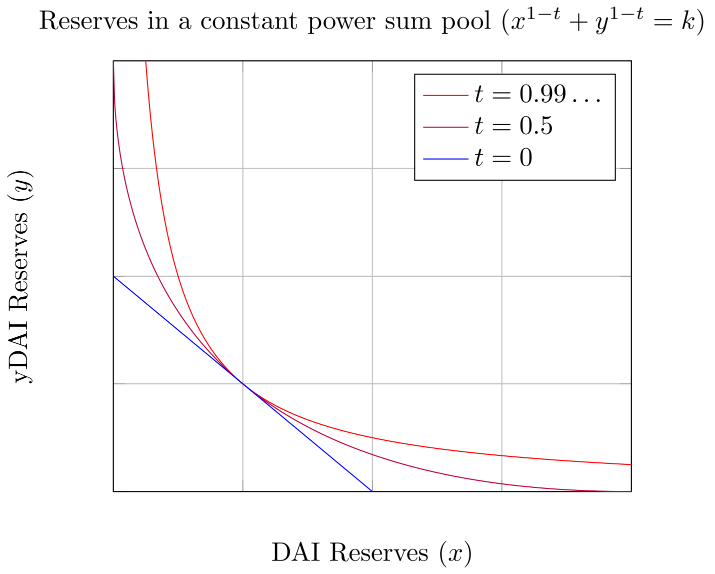
This paper introduces a new invariant that is optimized for market-making between yTokens and their target tokens, the "constant power sum formula":

$$x^{1-t} + y^{1-t} = k \tag{1}$$

y represents the reserves of the yToken (such as yDai), and x represents the reserves of the target token (such as Dai).

t represents time to maturity. We will assume that the units for t are normalized so that $0 \leq t < 1$. In the initial implementation of yDai, the units are normalized so that $t = 1$ at four years from maturity.

This curve resembles a hybrid between constant sum and constant product. When $t = 0$, the formula reduces to the constant sum formula $x + y = k$, which provides liquidity at a constant price of 1. When you take the limit as t approaches 1, the formula approximates the constant product formula $x \cdot y = k$ (as proven in Appendix D).



This formula can be treated as a pricing formula in "yield space," rather than "price space." As explained below, this formula has the unique property that the marginal interest rate of yDai that is offered by the pool at any time is equal to the ratio of the yDai reserves to the Dai reserves (minus 1):

$$r = \frac{y}{x} - 1 \tag{2}$$

For example, if the yDai reserves are 110 and the Dai reserves are 100, the marginal price offered by the pool would imply an interest rate of 10%.

The formula also ensures that the marginal interest rate offered by the pool does not change over time if the reserves do not change. Put another way, the target yDai allocation of the pool increases over time at exactly the same interest rate implied by the current price of yDai. While this does not eliminate losses due to arbitrage, it does mean that arbitrage opportunities should only arise when interest rates change.

The constant power sum invariant has been implemented as part of an automated pooled liquidity provider for yDai: YieldSpace. While YieldSpace was implemented by the Yield team, it is decoupled from the core yDai contracts, and can be upgraded independently.

Section 2 describes some prior work on invariant-based liquidity provision formulas. Section 3 describes a generally applicable methodology for deriving new invariants with desired properties, by expressing those properties as differential equations and solving them. Section 4 shows how the constant power sum formula can be derived. Section 5 describes how to incorporate fees into this formula in a way that takes into account interest rates and time to maturity. Finally, Section 6 describes how liquidity tokens work, and introduces an optimization that improves the capital efficiency of the pool.

2 Invariant-based liquidity provision formulas

Most of the popular protocols for swapping tokens on Ethereum, including Uniswap, Curve, Balancer, and Bancor, make use of some invariant—an algorithm for automatic liquidity provision that can be expressed (ignoring fees) in terms of the relation between the pool's reserves of one asset, y, and its reserves of another asset, x.

2.1 Constant sum formulas

Perhaps the simplest possible liquidity provider, the constant sum formula, always trades assets at the same price:

$$x + y = k \tag{3}$$

mStable [5] is a protocol that builds on this basic formula to provide liquidity between stablecoins.

This formula could be generalized to support prices other than 1. The following is the formula for a liquidity provider that always buys and sells asset x at price p:

$$p \cdot x + y = k \tag{4}$$

The constant sum formula is optimized for tokens that always trade at exactly the same relative price. If the relative price of the tokens diverges from the expected price, a pool using a constant sum formula would be expected to end up holding only the less valuable token, and doing relatively little volume (since it would have none of the more valuable token to sell, and would only offer the less valuable token above its market price). Since yDai can trade at varying prices over their lifetime, the constant sum formula would be a poor fit unless the yDai is already mature (and thus should trade at parity with Dai).

2.2 Constant product formulas

The constant product formula was first proposed in its most recognizable form[1] by Alan Lu of Gnosis [8]:

$$x \cdot y = k \tag{5}$$

This formula was implemented on Ethereum as part of the Uniswap protocol [3]. This formula maintains the property that at the current price offered by the protocol, the pool's reserves of each token are equal in value.

[1] Around the same time, Bancor [6] [7] proposed an equivalent formula, expressed in a different way. Bancor's formula—which, changing some variable names, can be written as $\Delta s = s_0 \cdot ((1 + \frac{x_{in}}{x_0})^{w_x} - 1)$—relates a minting or burning of "smart tokens" to deposits or withdrawals of just one of the reserve assets. It can be shown that this formula is equivalent to the invariant $x^{w_x} \cdot y^{w_y} \cdot z^{w_z} \ldots = s$, where s is the current supply of the smart tokens.

Lu also proposed a version of the constant product formula that supports more than two assets (x, y, $z\dots$), as well as custom weightings (w_x, w_y, $w_z\dots$) for the pool's holdings:

$$x^{w_x} \cdot y^{w_y} \cdot z^{w_z} \dots = k \tag{6}$$

This formula was implemented on Ethereum as part of the Balancer protocol [4].

Constant product formulas could be suitable for yTokens whose maturities are very distant. However, they are a poor fit for yTokens that are closer to maturity (such as those maturing within a year).

First, as a yToken approaches maturity, its price tends towards parity with its target token. The constant product formula reserves liquidity for prices along the full price spectrum, leaving a relatively small amount of liquidity around any particular price. This means that larger trades will likely have a significant impact on the interest rate earned or paid.

Second, the trading fees charged by those formulas are proportional to the amount traded, regardless of the time to maturity or the current interest rate. When the yToken is very close to maturity, even a small fee could result in a huge interest rate spread. For example, if a yToken is 1 month from maturity and annualized interest rates are 5%, a 0.3% fee would mean that the marginal borrow rate would be $\frac{1}{0.95 \cdot 0.997^{12}} - 1 = 9.1\%$ and the marginal lending rate would be $\frac{0.997^{12}}{0.95} - 1 = 1.01\%$—more than an 8% spread.

Finally, because these formulas do not take into account time to maturity, liquidity providers are subject to predictable "impermanent loss" as yTokens approach maturity and traders execute arbitrage trades that push the price toward parity. While the constant power sum formula cannot eliminate impermanent loss, it may reduce it, by exposing the pool to arbitrage only when the *interest rate* changes, rather than whenever the price changes.

2.3 Other invariants

Curve implements a custom formula for assets whose prices are expected to trade around (but not necessarily exactly at) a constant price. The formula shown in Curve's StableSwap paper [9] can be rewritten for the two-token case as the following (where χ is an amplification coefficient):

$$\chi \cdot k \cdot (x + y) + x \cdot y = \chi \cdot k^2 + \frac{k^2}{4} \tag{7}$$

Note that if $\chi = 0$, this curve reduces to the constant product formula. As χ approaches infinity, the curve approaches the constant sum formula.

While Curve's formula is more flexible than the constant sum or constant product formulas, it does not give us a way to compute this amplification factor χ as a function of time to maturity. As a result, it does not ensure that the interest rate offered by the pool remains constant, or that the pool's liquidity remains balanced across a range of interest rates, as time passes.

3 How to build your own invariant

Suppose you want to come up with a liquidity provision formula that satisfies some desired property. For example, suppose we are looking for the formula that maintains a 50-50 portfolio of two assets. Given this desired property, how can we come up with a liquidity provision formula that satisfies it, expressed in terms of an invariant relationship between the reserves x and y?

First, let's observe an important fact about all of these curves. If Alice sells some quantity of the x asset to the pool, the pool's x reserves increase by that amount—call it dx—and the pool's y reserves decrease by some amount—call it $-dy$ (since y is going down, the change in y, dy, is negative, so $-dy$ is the amount of the y asset that is sent out). The ratio $\frac{-dy}{dx}$ represents the price at which Alice sold the x asset. This ratio for an infinitesimally small dx— $\lim\limits_{dx \to 0} \dfrac{-dy}{dx}$—the negation of the derivative at that point—gives you the *marginal price* offered by the contract at that point.

$$p_x = -\frac{dy}{dx} \tag{8}$$

This means that if we want a formula that satisfies some relation between price, reserves, and any other quantity (like time), we can try to find an invariant that satisfies it by rewriting that property as a differential equation. (By specifying x_{start} and y_{start}, we can turn it into an initial value problem.)

For example, suppose we want a formula that always offers to trade at the fixed price of 1. This can be expressed as the trivial differential equation:

$$1 = -\frac{dy}{dx} \tag{9}$$

As shown in Appendix A, the solution to this is the constant sum formula.

As another example, suppose we wanted a formula that gives us the property that, at the current price offered by the contract, the reserves of the pool's two assets are equal in value. Since p_x is the price of the x token in terms of the y token, this desired relationship can be expressed as $p_x \cdot x = y$. Since $p_x = -\frac{dy}{dx}$, we can rewrite this property as:

$$-\frac{dy}{dx} = \frac{y}{x} \tag{10}$$

As shown in Appendix B, the solution to this differential equation is the constant product formula $x \cdot y = k$.

The same process can be used to find the version of the constant sum formula that offers tokens at prices other than 1, as well as the versions of the constant product formula that support more than two tokens, or weights other than 50/50.

Not every desired relation will give us a differential equation that can be solved, but this works often enough to be a powerful tool to discover new formulas with desired properties.

4 The constant power sum formula

We want to build a liquidity provision formula that works in "yield space" rather than "price" space. Specifically, we want the interest rate—not the price—to be a pure function of reserves.

There are many possible choices for what this function could be. We chose to find the formula where the interest rate[2] is the ratio of the yDai reserves (y) and Dai reserves (x), minus one.

$$r = \frac{y}{x} - 1 \tag{11}$$

We can motivate this choice by analogy to the constant product formula, which, as described above, uses the ratio between the reserves as the *price*. We also note that it observes the desirable property that increases in yDai reserves increases the interest rate, while increases in the Dai reserves decrease the interest rate. Finally, this property happens to give us a solveable differential equation, allowing us to find a closed-form invariant that can be efficiently implemented in the contract.

As described in the Yield Protocol paper [1], the interest rate of yDai can be computed as a function of the current price of Dai in terms of yDai, p, and the time to maturity, t:

$$r = p^{\frac{1}{t}} - 1 \tag{12}$$

For example, if 1 yDAI that expires six months from today is currently worth 0.5 DAI (meaning the Dai price in terms of yDai, p, is 2), that implies that the annualized interest rate is $2^{\frac{1}{0.5}} - 1 = 300\%$.

Simplifying the two equations above gives us this equation:

$$p = \left(\frac{y}{x}\right)^t \tag{13}$$

In any invariant-based liquidity provision formula, the price at any point along the curve is equal to the negation of the derivative at that point.

$$p = -\frac{dy}{dx} \tag{14}$$

So, finding the curve where the interest rate is always equal to the ratio of the reserves minus one reduces to solving the following differential equation:

$$-\frac{dy}{dx} = \left(\frac{y}{x}\right)^t \tag{15}$$

As shown in Appendix C, the solution to this differential equation is the following equation:

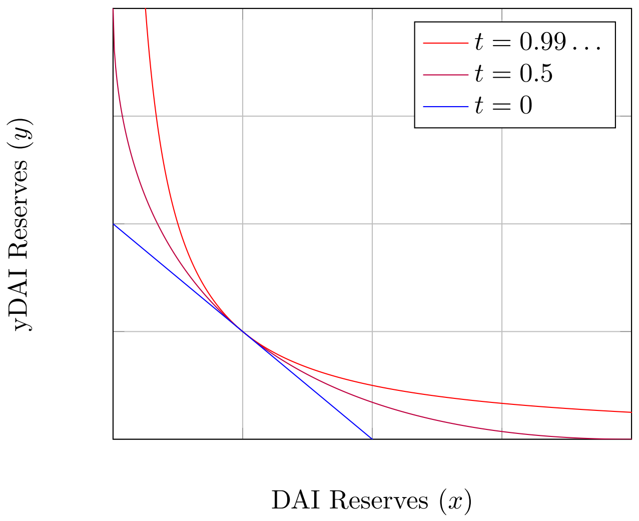
$$x^{1-t} + y^{1-t} = k \tag{16}$$

[2]For purposes of this derivation, we can imagine that this is the annualized interest rate. As discussed below, we could actually normalize this to any time period, simply by changing the units for t.

4.1 Properties

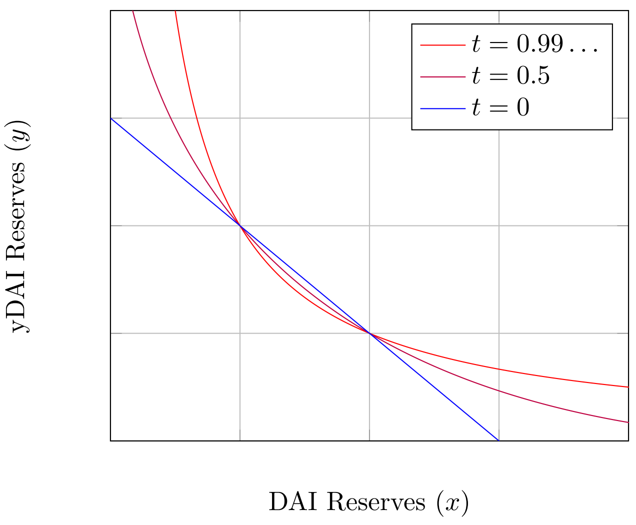
As mentioned in Section 1 and shown in Appendix D, the constant power sum formula gradually evolves from the constant product formula at $\lim_{t\to 1}$ to the constant sum formula at $t = 0$:

Evolution of reserve formula in a constant power sum pool (0% rates)



The invariant $x_{start}^{1-t} + y_{start}^{1-t}$ is recomputed for each trade based on the current value of t, and the current reserves for x and y. This means the curve "pivots" around the point representing the current reserves. The above chart shows how the curve evolves when rates are 0%. If rates are 100% (meaning the pool has twice as much yDai as Dai), the evolution of the curve would look like:

Evolution of reserve formula in a constant power sum pool (100% rates)

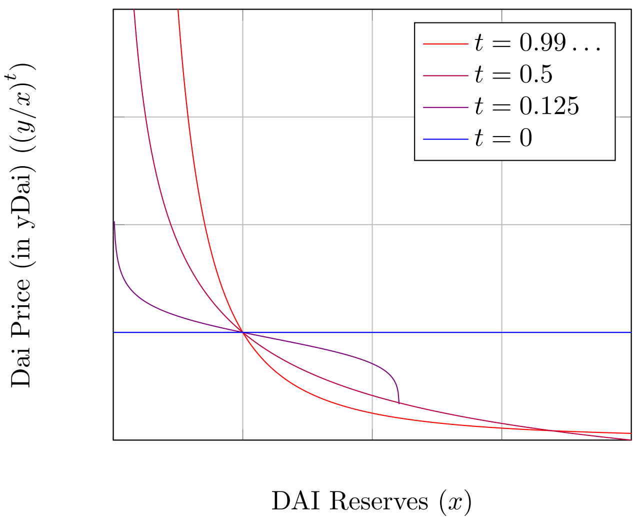


The marginal price (of Dai in terms of yDai) for a given x_{start}, y_{start}, and t is given by the formula:

$$\left(\frac{y}{x}\right)^t = \left(\frac{\left(x_{start}^{1-t} + y_{start}^{1-t} - x^{1-t}\right)^{\frac{1}{1-t}}}{x}\right)^t \tag{17}$$

The price curve gradually flattens as maturity approaches:

Dai price (in yDai) in a constant power sum pool

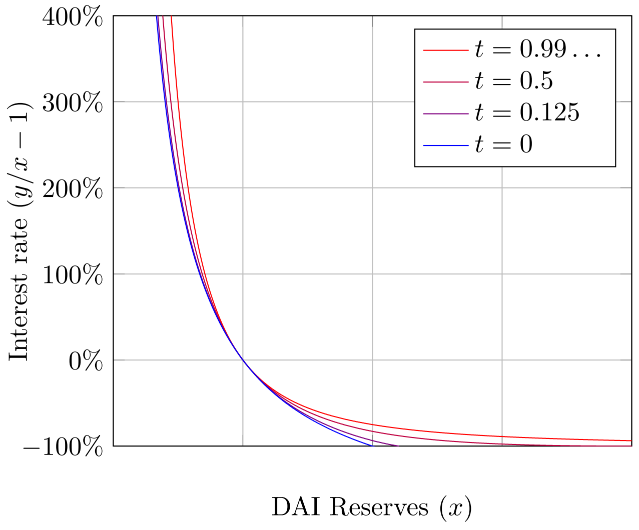


However what if we look at *interest rates* instead of price? Recall the formula for the marginal interest rate:

$$\frac{y}{x} - 1 = \frac{\left(x_{start}^{1-t} + y_{start}^{1-t} - x^{1-t}\right)^{\frac{1}{1-t}}}{x} - 1 \tag{18}$$

The graph of interest rates changes only minimally as maturity approaches:

Interest rates in a constant power sum pool



In other words, this formula provides liquidity across a relatively consistent range of interest rates over the lifetime of the contract.

Notice that, when interest rates are 0%, at least half of the Dai liquidity in the contract is reserved for providing liquidity at interest rates greater than 100%. If this represented annualized interest rates, that would be a lot of liquidity reserved for a relatively unlikely case. To maximize the useful liquidity in the pool, the contract normalizes the units for t so that $t = 1$ at four years, which will mean that the interest rate given by the formula $\frac{y}{x}$ is the *four-year* interest rate, rather than the annualized interest rate.

Even more significantly, when interest rates are 0%, *all* of the pool's yDai is reserved for providing liquidity at negative interest rates, which means it is wasted (since yDai should never trade at a price higher than 1 Dai). Section 6.3 discusses this problem and a mitigation for it.

5 Fees

To incentivize liquidity providers to deposit tokens into the pool, it is customary to charge a fee on every trade.

Because the constant power sum formula buys yDai (and earns interest on it) when interest rates rise and sells it when interest rates fall, it is possible for the pool to outperform the returns on both yDai and Dai, *even before fees.* (By contrast, a *static* invariant that does not incorporate time will always underperform at least one of its constituent assets, before fees.) Still, to further reward liquidity, we can build a fee into the protocol.

In other protocols, fees are typically charged in proportion to the amount being sold. For example, in Uniswap v2, a 0.3% fee is charged on the tokens sent into the contract before the amount sent out is computed. However, when a yToken is close to maturity, even a modest fee can translate into a wide spread in interest rate terms.

We want our fees, like our prices, to be computed in "yield space" rather than "price space"—meaning that they should impose some proportional spread on interest rates, rather than on prices. For example, in a fee-adjusted formula, a buyer of yDai should receive a lower interest rate (equivalent to paying a higher price) than with the no-fee formula derived in section 4. We believe the simplest way to adjust the interest rate downward is to raise it (before subtracting 1) to some constant power g that is less than 1:

$$r = \left(\frac{y}{x}\right)^g - 1 \tag{19}$$

For example, if the pool's yDai balance is 110, its Dai balance is 100, and g is 0.95, then the unadjusted formula would give us a marginal interest rate of $\frac{110}{100} - 1 = 10\%$. The adjusted formula will give us a marginal interest rate of $\frac{110}{100}^{0.95} - 1 \approx 9.47\%$.

Notice that the fee charged—0.5%—was approximately 5% of the current interest rate. When interest rates are relatively small and $g < 1$, the binomial approximation tells us that $(1 + r)^g \approx 1 + gr$.

The actual invariant used depends on the direction of the trade. When the trader is *buying* yDai from the contract, the above invariant is used with a g less than 1 to reduce the interest rate (and thus increase the price paid for the yDai). If the user is *selling* yDai to the contract, the interest rate is raised to the power of $\frac{1}{g}$ instead, causing the user to pay a higher interest rate (in other words, receive a lower price for the yDai they are selling).

In the above example, the marginal interest rate for a borrower—someone *selling* yDai to the pool—would be $\frac{110}{100}^{\frac{1}{0.95}} - 1 \approx 10.55\%$.

By transforming the fee equation described above into a differential equation and solving it using essentially the same steps used in section 4 and Appendix C, we derive the following invariant (replacing gt with $\frac{t}{g}$ when selling yDai[3]):

$$x^{1-gt} + y^{1-gt} = k \tag{20}$$

One other convenient property of this fee formula is that splitting up a trade into multiple pieces does not result in paying higher or lower fees (modulo

[3]Using $\frac{t}{g}$ in this way means we need to change our assumption that $t < 1$ to an assumption that $\frac{t}{g} < 1$, to avoid any divisions by 0 in our calculations.

precision errors). When fees are charged as a percentage of the tokens sent in (as they are in Uniswap and most others), this property does *not* hold—a user who splits a trade into multiple pieces pays higher fees than if they had combined them. This helps make fees more amenable to analysis.

6 Liquidity tokens

As in Uniswap, anyone can add liquidity to a pool by providing Dai and yDai in proportion to the reserves that are already in the pool. When they provide liquidity, they are minted a proportional share of *liquidity tokens*.

6.1 Initialization

Pools must be initialized with equal reserves of Dai and yDai, implying a price of 1 and an interest rate of 0%. To initialize a pool with a different interest rate, you can initialize it at 0% and then atomically trade with it to push it to the desired interest rate.

The number of pool tokens initially minted is equal to the quantity of Dai supplied.

When liquidity pool tokens are initialized in this way, the following invariant is guaranteed to not decrease with any state changes of the contract (as shown in Appendix E):[4]

$$\frac{\left(\frac{x_{start}^{1-\frac{t}{g}}+y_{start}^{1-\frac{t}{g}}}{2}\right)^{\frac{1}{1-\frac{t}{g}}}}{s} \tag{21}$$

As the pool accumulates fees from trading and interest from its yDai holdings, the value of this invariant rises.

6.2 Minting and burning

Minting and burning liquidity tokens works similarly as in Uniswap and other pooled liquidity provision protocols. When a new liquidity provider enters the pool, they deposit Dai and yDai in proportion to the Dai and yDai already in the contract's reserves, and receive newly minted liquidity tokens in the same proportion to the total supply in the pool:

$$\frac{\Delta s}{s_{start}} = \frac{\Delta x}{x_{start}} = \frac{\Delta y}{y_{start}} \tag{22}$$

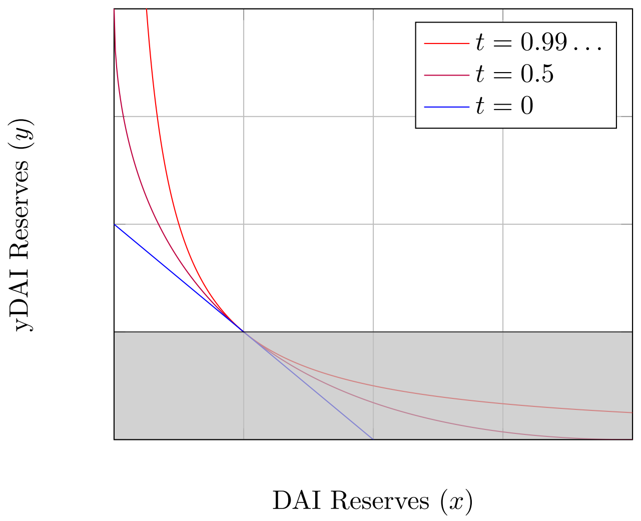
[4]As discussed in the Appendix, the version of the invariant with $\frac{t}{g}$ is used to properly handle an edge case in which Dai is donated to the contract.

6.3 Capital efficiency

Under normal circumstances, 1 yDAI will never naturally trade at a higher price than 1 DAI (which would correspond to a negative interest rate). The core yDai protocol always allows you to mint 1 yDAI by depositing 1 DAI collateral (with no risk of liquidation), so there is no reason to buy yDai from the pool at a price greater than 1 DAI. Additionally, the above formulas assume that the price of yDAI is never greater than 1.[5] As a result, the pool checks at the end of each trade that the yDAI price is not above 1 DAI—or, equivalently, that the yDai reserves are greater than the Dai reserves.

As a result of this requirement, there is some portion of the yDai reserves that the pool will never touch:

Inaccessible *yDai* reserves in a constant power sum pool



For example, when the pool is first initialized, the yDai reserves and Dai reserves are equal, so none of the yDai remaining in the pool can be sold. The quantity of inaccessible yDai at any given time is given by computing the amount of yDai that would be in the contract if there was an immediate trade to a 1-to-1 ratio. The solution to this[6] happens to be this formula:

$$\left(\frac{x_{start}^{1-gt} + y_{start}^{1-gt}}{2}\right)^{\frac{1}{1-gt}} \tag{23}$$

[5]For example, the fee formula is constructed under the assumption that the interest rate is positive.

[6]Note that this formula assumes that the trade needed to push the price to parity is a yDai buy rather than a sell. The latter case can happen after someone donates Dai to the pool, but in that case, no yDai is accessible at all, since buys are prohibited, so we do not need to consider that case.

We can improve the capital efficiency of the pool by making some of these excess reserves "virtual" and not requiring liquidity providers to contribute them.

Notice that the above formula is similar to the numerator from the invariant in equation 21. As shown in Appendix F, the amount of inaccessible yDai will always be greater than or equal to the total supply of shares.

As a result, we can use the total supply of liquidity tokens s as the virtual yDai reserves. Whenever a trade occurs, the virtual yDai reserves are added to the actual yDai reserves to determine the reserves used to calculate the outcome of the trade. But whenever liquidity tokens are minted, the minter only has to provide yDai proportional to the *actual* yDai in the pool's reserves. The virtual yDai reserves will automatically go up in proportion (since total supply s increases proportionally, thanks to the newly minted liquidity tokens).

For example, suppose Alice initializes a pool with 100 Dai. The new total supply of liquidity tokens s is 100, and Alice receives 100 liquidity tokens.

Suppose Alice then sells 100 yDai to the pool at time $t = 0.5$. Since this is a sell, the pool uses the invariant $x_{end}^{1-\frac{t}{g}} + y_{end}^{1-\frac{t}{g}} = x_{start}^{1-\frac{t}{g}} + y_{start}^{1-\frac{t}{g}}$. For x_{start}, the pool uses the actual Dai reserves, 100. For y_{start}, the pool adds the actual yDai reserves (0) to the virtual yDai reserves, which is equal to the total supply s, 100. Finally, for y_{end}, the pool adds the yDai in (100) to y_{start} (including the virtual reserves), to get 200. Solving for x_{end} (with $g = 0.95$) gives us the result $x = 34.31$, meaning $100 - 34.31 = 65.69$ Dai is sent out of the contract.

Now suppose Bob mints 10 new liquidity tokens (10% of the total supply) by adding liquidity. The pool currently has 34.31 Dai in its reserves, so Bob deposits 3.43 Dai. The pool has only 100 *actual* yDai, so Bob only has to deposit 10 yDai. Finally, when the 10 liquidity tokens are minted to Bob, the total supply (and thus the virtual yDai reserves) increases to 110. In the next trade, the value for y_{start} will be $100 + 110 = 210$.

With this optimization, the contract has zero inaccessible yDai upon initialization. However, as soon as there is a trade, the fees increase the value of liquidity shares. As the pool accumulates trading fees and the value of its yDai increases due to interest, the invariant above increases, meaning that an increasing share of yDai becomes inaccessible.

7 Acknowledgments

This paper is heavily indebted to discussions with Hayden Adams, Guillermo Angeris, Dmitriy Berenzon, Tarun Chitra, Zubin Koticha, Hart Lambur, Teo Leibowitz, Mikhail Vladimirov, Cyrus Younessi, and Noah Zinsmeister. The authors are particularly grateful to Paradigm for supporting this research.

Appendices

A Deriving the constant sum formula

We can show that the solution to the differential equation below is the constant sum formula, by integrating both sides with respect to x and simplifying.

$$1 = -\frac{dy}{dx} \tag{24}$$

$$\int dx = \int -dy \tag{25}$$

$$x + c_1 = -y + c_2 \tag{26}$$

$$x + y = c_1 + c_2 = k \tag{27}$$

B Deriving the constant product formula

We can show that the solution to the differential equation below is the constant product formula.

$$-\frac{dy}{dx} = \frac{y}{x} \tag{28}$$

First, we rewrite the equation:

$$-\frac{1}{y} \cdot \frac{dy}{dx} = \frac{1}{x} \tag{29}$$

We then integrate both sides with respect to x:

$$\int -\frac{1}{y} dy = \int \frac{1}{x} dx \tag{30}$$

$$-\ln y + c_1 = \ln x + c_2 \tag{31}$$

$$\ln x + \ln y = c_1 - c_2 \tag{32}$$

By raising e to the power of each side, and defining the constant $e^{c_1 - c_2}$ as k, we can rewrite this as the constant product formula:

$$x \cdot y = k \tag{33}$$

C Deriving the constant power sum formula

We can show that the solution to the differential equation below is the constant power sum formula.

$$-\frac{dy}{dx} = \left(\frac{y}{x}\right)^t \tag{34}$$

First, we rewrite the equation:

$$-\frac{dy}{dx} = x^{-t} \cdot y^t \tag{35}$$

$$-y^{-t} \cdot \frac{dy}{dx} = x^{-t} \tag{36}$$

We then integrate both sides with respect to x:

$$\int -y^{-t} dy = \int x^{-t} dx \tag{37}$$

$$-\frac{y^{1-t}}{1-t} + c_1 = \frac{x^{1-t}}{1-t} + c_2 \tag{38}$$

By multiplying both sides by $1-t$ and defining the constant $(1-t) \cdot (c_1 - c_2)$ as k, we can rewrite this as the constant power sum formula:

$$x^{1-t} + y^{1-t} = k \tag{39}$$

D Deriving the limit of the constant power sum formula as t approaches 1

We can show that as t approaches 1, the constant power sum formula ($x^{1-t} + y^{1-t} = x_{start}^{1-t} + y_{start}^{1-t}$) approaches the constant product formula ($x \cdot y = x_{start} \cdot y_{start}$).

The constant power sum formula can be rewritten to make y a function of x (with x_{start} and y_{start} as constants):

$$y = (x_{start}^{1-t} + y_{start}^{1-t} - x^{1-t})^{\frac{1}{1-t}} \tag{40}$$

This function can be rewritten in log form:

$$\ln(y) = \frac{\ln(x_{start}^{1-t} + y_{start}^{1-t} - x^{1-t})}{1-t} \tag{41}$$

When $t = 1$, the right side of the equation is the indeterminate form $\frac{0}{0}$, but applying L'Hôpital's rule gives us the limit:

$$\ln y = \lim_{t \to 1} \frac{\frac{\partial}{\partial t} \ln(x_{start}^{1-t} + y_{start}^{1-t} - x^{1-t})}{\frac{\partial}{\partial t}(1-t)} \tag{42}$$

$$\ln y = \frac{-\ln(x_{start}) - \ln(y_{start}) + \ln(x)}{-1} \tag{43}$$

$$\ln y = \ln(x_{start}) + \ln(y_{start}) - \ln(x) \tag{44}$$

$$\ln x + \ln y = \ln(x_{start}) + \ln(y_{start}) \tag{45}$$

Raising e to the power of both sides gives us the constant product function $x \cdot y = x_{start} \cdot y_{start}$.

E Proving the invariant for total supply

We will show that the invariant below begins at 1 when the pool is initialized, and does not decrease during any of the contract's state changes (minting/burning, trading, and time passing).

$$\frac{\left(\frac{x^{1-\frac{t}{g}} + y^{1-\frac{t}{g}}}{2}\right)^{\frac{1}{1-\frac{t}{g}}}}{s} \tag{46}$$

The reason we use this invariant rather than the one with $1-t$ is to handle an edge case where someone directly donates enough Dai to the contract to cause the yDai price to go above 1 Dai (and implied interest rate to go negative). Buys will be prohibited in this state, but sells of yDai will not (and are needed to return to positive interest rates). When interest rates are negative, the fee benefits the *trader* rather than the pool, meaning that it is possible for the invariant that uses $1 - t$ to decrease when someone sells yDai to the pool at a price above 1. Using the above invariant (which is directly enforced during sells) avoids this.

E.1 Initialization

When the pool is initialized, x_{start}, y_{start}, and s are all equal. Substituting s for the other variables in the above formula shows that the initial value for the invariant is 1:

$$\frac{\left(\frac{s^{1-\frac{t}{g}} + s^{1-\frac{t}{g}}}{2}\right)^{\frac{1}{1-\frac{t}{g}}}}{s} \tag{47}$$

$$\frac{\left(s^{1-\frac{t}{g}}\right)^{\frac{1}{1-\frac{t}{g}}}}{s} \tag{48}$$

$$1 \tag{49}$$

E.2 Minting and burning

We can show that minting or burning shares (which causes x, y, and s to be multiplied by the same proportion, m) does not change the invariant:

$$\frac{\left(\frac{(m\cdot x)^{1-\frac{t}{g}}+(m\cdot y)^{1-\frac{t}{g}}}{2}\right)^{\frac{1}{1-\frac{t}{g}}}}{m\cdot s} \tag{50}$$

$$\frac{\left(\frac{m^{1-\frac{g}{t}}\cdot(x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}})}{2}\right)^{\frac{1}{1-\frac{t}{g}}}}{m\cdot s} \tag{51}$$

$$\frac{m\cdot\left(\frac{(x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}})}{2}\right)^{\frac{1}{1-\frac{t}{g}}}}{m\cdot s} \tag{52}$$

$$\frac{\left(\frac{(x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}})}{2}\right)^{\frac{1}{1-\frac{t}{g}}}}{s} \tag{53}$$

E.3 Trading

Next, we will show that the invariant does not decrease as a result of trading. Since s and $\frac{1}{1-\frac{t}{g}}$ do not change during trading, we can simplify our analysis and look only at the core expression of this invariant:

$$x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}} \tag{54}$$

Note that this is exactly the same invariant that is enforced by the formula for selling yDai described in section 5. As a result, we only need to show that this invariant only increases during *buys* of yDai.

The above invariant was constructed so that the derivative $\frac{dy}{dx}$ would be:

$$\frac{dy}{dx}=-\left(\left(\frac{y}{x}\right)^{t}\right)^{\frac{1}{g}} \tag{55}$$

Buys of yDai use a different adjusted version of the constant power sum formula:

$$x^{1-gt}+y^{1-gt}=k \tag{56}$$

Recall that this invariant was constructed so that the derivative $\frac{dy}{dx}$ would be:

$$\frac{dy}{dx}=-\left(\left(\frac{y}{x}\right)^{t}\right)^{g} \tag{57}$$

Buys involve an increase in x and a decrease in y. For a given increase in x, we need to show that the decrease in y using the invariant for buys is less than or equal to than the decrease in y using the other invariant:

$$\left(\left(\frac{y}{x}\right)^t\right)^g \leq \left(\left(\frac{y}{x}\right)^t\right)^{\frac{1}{g}} \tag{58}$$

Raising both sides to the power of $\frac{1}{t}$, taking the logarithm of both sides, and multiplying both sides by g gives us:

$$g^2 \cdot \ln \frac{y}{x} \leq \ln \frac{y}{x} \tag{59}$$

Since $g \leq 1$, this inequality will be true if we can divide both sides by $\ln \frac{y}{x}$ without flipping the inequality, which will only be possible if $\ln \frac{y}{x} > 0$. This will be true when $y > x$. Since buys are disallowed whenever $y \leq x$, this is guaranteed to be true for all buys, so the invariant will never decrease during a buy.

E.4 Time

Finally, we can show that as time passes (causing t, time to maturity, to decrease), the numerator of the invariant always increases (or stays the same). (Since the denominator of the invariant does not change as time passes, we can ignore it. We can also use t rather than gt, since this coefficient will not change whether the derivative with respect to t will ever be positive.) To do this, we will take the partial derivative with respect to time, and show that it is always less than or equal to 0:

$$\frac{\partial}{\partial t} \frac{\left(\frac{x^{1-t}+y^{1-t}}{2}\right)^{\frac{1}{1-t}}}{s} \leq 0 \tag{60}$$

$$\left(\frac{x^{1-t}+y^{1-t}}{2}\right)^{\frac{1}{1-t}} \cdot \left(\frac{\ln\left(x^{1-t}+y^{1-t}\right)-\ln 2}{(1-t)^2} - \frac{x^{1-t}\cdot \ln x + y^{1-t}\cdot \ln y}{(1-t)\cdot\left(x^{1-t}+y^{1-t}\right)}\right) \leq 0 \tag{61}$$

We can divide both sides by the numerator from our invariant, multiply by $(1-t)^2 \cdot (x^{1-t}+y^{1-t})$, and further rewrite some terms to get:[7]

$$\left(\ln \frac{x^{1-t}+y^{1-t}}{2}\right) \cdot \left(x^{1-t}+y^{1-t}\right) - \left(x^{1-t}\cdot \ln x^{1-t} + y^{1-t}\cdot \ln y^{1-t}\right) \leq 0 \tag{62}$$

We can show with substitution that if $x^{1-t} = y^{1-t}$, then the left side of the expression is equal to 0.

[7]The values used in these multiplications and divisions are guaranteed to always be greater than 0.

$$\ln x^{1-t} \cdot \left(2 \cdot x^{1-t}\right) - \left(2 \cdot x^{1-t} \cdot \ln x^{1-t}\right) \tag{63}$$

We can then show that this is a local and global maximum—the left side is only equal to 0 if $x^{1-t} = y^{1-t}$, and is less than 0 if they are not equal.

First, we define $a = x^{1-t}$ and $b = y^{1-t}$, which allows us to rewrite the expression as:

$$\ln \frac{a+b}{2} \cdot (a+b) - (a \cdot \ln a + b \cdot \ln b) \tag{64}$$

We can take the derivative of this expression with respect to a, and show that it is only equal to 0 if $a = b$:

$$\frac{\partial}{\partial a} \ln \frac{a+b}{2} \cdot (a+b) - (a \cdot \ln a + b \cdot \ln b) = 0 \tag{65}$$

$$\ln \frac{a+b}{2} - \ln a = 0 \tag{66}$$

$$\ln \frac{a+b}{2} = \ln a \tag{67}$$

$$\frac{a+b}{2} = a \tag{68}$$

$$b = a \tag{69}$$

Finally, we can determine that this is a maximum rather than a minimum or inflection point by taking the second derivative and showing that it is less than 0 when $a = b$ (and indeed for any positive a and b):

$$\frac{\partial}{\partial a} \ln \frac{a+b}{2} - \ln a \tag{70}$$

$$-\frac{b}{ab + a^2} \tag{71}$$

This proves that the derivative of the invariant with respect to t is less than or equal to 0 at all points, meaning that as time to maturity decreases, the invariant will either increase or stay the same.

F Proving that number of shares will always be less than or equal to inaccessible yDai

We can show that the total supply of shares will always be greater than or equal to the amount of inaccessible yDai (meaning yDai that can safely be "virtual" rather than actual):

$$y_\varnothing \leq s \tag{72}$$

As shown in Appendix E, the following condition is guaranteed to hold throughout the lifetime of the contract:

$$\frac{\left(\frac{x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}}}{2}\right)^{\frac{1}{1-\frac{t}{g}}}}{s} \geq 1 \tag{73}$$

This can be rewritten as:

$$\left(\frac{x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}}}{2}\right)^{\frac{1}{1-\frac{t}{g}}} \geq s \tag{74}$$

The amount of inaccessible yDai at any time (when it is possible to buy yDai at all) is:

$$y_\varnothing = \left(\frac{x^{1-gt}+y^{1-gt}}{2}\right)^{\frac{1}{1-gt}} \tag{75}$$

Therefore, to prove that $y_\varnothing \geq s$, it is sufficient to prove that the following equation is true when $0 < x$, $0 < y$, $0 < g \leq 1$ and $0 \leq t < g$:

$$\left(\frac{x^{1-gt}+y^{1-gt}}{2}\right)^{\frac{1}{1-gt}} \geq \left(\frac{x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}}}{2}\right)^{\frac{1}{1-\frac{t}{g}}} \tag{76}$$

Taking the logarithm of both sides gives us:

$$\frac{\ln\left(x^{1-gt}+y^{1-gt}\right)-\ln 2}{1-gt} \geq \frac{\ln\left(x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}}\right)-\ln 2}{1-\frac{t}{g}} \tag{77}$$

This can be rewritten as:

$$\frac{\ln\left(x^{1-gt}+y^{1-gt}\right)-\ln 2}{1-gt} - \frac{\ln\left(x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}}\right)-\ln 2}{1-\frac{t}{g}} \geq 0 \tag{78}$$

Note that when $x = y$, the left side is equal to 0. Therefore, all we need to show is that this is a global minimum of that expression. To do that, we take the first derivative of the expression with respect to x:

$$\frac{d}{dx}\left(\frac{\ln\left(x^{1-gt}+y^{1-gt}\right)-\ln 2}{1-gt} - \frac{\ln\left(x^{1-\frac{t}{g}}+y^{1-\frac{t}{g}}\right)-\ln 2}{1-\frac{t}{g}}\right) \tag{79}$$

$$\frac{y^{gt}}{x \cdot y^{gt} + y \cdot x^{gt}} - \frac{y^{\frac{t}{g}}}{x \cdot y^{\frac{t}{g}} + y \cdot x^{\frac{t}{g}}} \tag{80}$$

$$x \cdot y^{gt+\frac{t}{g}} + y^{1+gt} \cdot x^{\frac{t}{g}} - \left(x \cdot y^{gt+\frac{t}{g}} + y^{1+\frac{t}{g}} \cdot x^{gt}\right) \tag{81}$$

$$y^{1+gt} \cdot x^{\frac{t}{g}} - y^{1+\frac{t}{g}} \cdot x^{gt} \tag{82}$$

By solving for when this first derivative is 0, we can find the function's only stationary point:

$$y^{gt-\frac{t}{g}} = x^{gt-\frac{t}{g}} \tag{83}$$

$$y = x \tag{84}$$

Therefore, this derivative is only equal to 0 when $x = y$ (given that $gt - \frac{t}{g} \neq 0$).

To prove that this is a minimum, we can take the second derivative with respect to x:

$$\frac{\partial}{\partial x}\left(y^{1+gt} \cdot x^{\frac{t}{g}} - y^{1+\frac{t}{g}} \cdot x^{gt}\right) \tag{85}$$

$$\frac{t}{g} \cdot y^{1+gt} \cdot x^{\frac{t}{g}-1} - gt \cdot y^{1+\frac{t}{g}} \cdot x^{gt-1} \tag{86}$$

When $x = y$, $x > 0$, and $0 < t < g < 1$ this second derivative is positive:

$$\frac{t}{g} \cdot x^{gt+\frac{t}{g}} - gt \cdot x^{gt+\frac{t}{g}} \tag{87}$$

$$\left(\frac{t}{g} - gt\right) \cdot x^{gt+\frac{t}{g}} \tag{88}$$

This means this point is a minimum. This tells us that the expression above always greater than or equal to 0, and thus that $y_{\varnothing} \leq s$.

References

[1] Dan Robinson and Allan Niemerg. *The Yield Protocol: On-Chain Lending With Interest Rate Discovery.* URL: https://yield.is/Yield.pdf.

[2] Allan Niemerg. *Introducing yDai.* URL: https://medium.com/yield-protocol/introducing-ydai-43a727b96fc7.

[3] Hayden Adams. *Uniswap.* URL: https://uniswap.org/docs/.

[4] Fernando Martinelli and Nikolai Mushegian. *A non-custodial portfolio manager, liquidity provider, and price sensor.* URL: https://balancer.finance/whitepaper/.

[5] mStable. *mASSETS.* URL: https://docs.mstable.org/mstable-assets/massets.

[6] Guy Benartzi Eyal Hertzog and Galia Benartzi. *Bancor Protocol: Continuous Liquidity for Cryptographic Tokens through their Smart Contracts.* URL: `https://whitepaper.io/document/52/bancor-whitepaper`.

[7] Meni Rosenfeld. *Formulas for Bancor system.* Dec. 2016. URL: `https://drive.google.com/file/d/0B3HPNP-GDn7aRkVaV3dkVl9NS2M/view`.

[8] Alan Lu. Mar. 2017. URL: `https://blog.gnosis.pm/building-a-decentralized-exchange-in-ethereum-eea4e7452d6e`.

[9] Michael Egorov. *StableSwap - efficient mechanism for Stablecoin liquidity.* URL: `https://medium.com/yield-protocol/introducing-ydai-43a727b96fc7`.

8 Disclaimer

This paper is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. It should not be relied upon for accounting, legal or tax advice or investment recommendations. This paper reflects current opinions of the authors and is not made on behalf of Paradigm or its affiliates and does not necessarily reflect the opinions of Paradigm, its affiliates or individuals associated with Paradigm. The opinions reflected herein are subject to change without being updated.