EulerSwap White Paper

Euler Labs

May 2025

Abstract

Liquidity in decentralised exchanges (DEXs) is often underutilised, with capital sitting idle even when it could be earning yield. EulerSwap addresses this inefficiency by integrating an automated market maker (AMM) directly with Euler's lending vaults. Liquidity providers (LPs) can earn swap fees, accrue lending yield, and borrow natively against their positions—all within a single modular system. Reserves are held directly in Euler vaults, making them dual-purpose by default: productive both during and outside of trading. Each EulerSwap instance is owned and managed by a single market maker. This design avoids pooled liquidity, giving each LP full customisation and control. It also enables low-cost, automated dynamic hedging strategies to mitigate impermanent loss in uncorrelated asset pairs. Custom AMM curves allow LPs to define asymmetric, single-sided, or concentrated liquidity profiles and to update parameters using operator contracts. The protocol also supports just-in-time (JIT) liquidity provision, allowing the AMM to borrow the output token using the input token and vault collateral as backing. This mechanism can simulate up to 50x greater depth than conventional AMMs, particularly valuable for bootstrapping liquidity in stable asset markets. Euler's lending vaults further enhance capital efficiency by acting as liquidity hubs across many trading pairs. EulerSwap is fully compatible with Uniswap v4's hook architecture, enabling integration with its routing and solver networks. By combining lending, borrowing, and swaps in one composable system, EulerSwap delivers deeper markets, lower capital costs, and greater strategic control to LPs and token issuers alike.

1 Introduction

Liquidity on decentralised exchanges (DEXs) is frequently underutilised, with capital often sitting idle instead of earning yield elsewhere. This inefficiency raises costs for liquidity providers (LPs), limits market depth, and reduces the competitiveness of DEXs relative to centralised exchanges. EulerSwap addresses these issues by integrating an automated market maker (AMM) directly with Euler's lending vaults, enabling capital to serve multiple purposes—facilitating swaps, earning lending yield, and backing borrows—within a single unified system.

In EulerSwap, DEX reserves are held natively in Euler vaults and thus become dual-purpose by default. Idle liquidity automatically earns lending yield, even outside of active trading, similar to Balancer's boosted pools [1] or Bunni's rehypothecation vaults [2]. Moreover, since LP capital remains in the vault, it can also be used as collateral for borrowing. This eliminates the need to wrap LP tokens for use elsewhere, allowing market participants to hedge or leverage LP positions directly from the same account that provides swap liquidity.

Unlike traditional AMMs that aggregate liquidity from multiple LPs into a shared pool, each EulerSwap instance is controlled by a single Euler account holder who supplies both the swap liquidity and the associated collateral. Paired with EulerSwap's customisable AMM curve, this design gives market makers the flexibility to dynamically rebalance their positions. As a result, EulerSwap supports sophisticated automated hedging strategies [3], akin to those sometimes used on Uniswap v3 [4]. While dynamic hedging through rebalancing is a proven method for mitigating impermanent loss in uncorrelated pairs, such strategies often incur significant costs. EulerSwap enables both on-chain and off-chain rebalancing at substantially lower cost. When combined with lending yield, this allows LPs to maintain neutral portfolios while supplying deep liquidity for volatile pairs like ETH/USDC.

For correlated assets, EulerSwap can further amplify liquidity by dynamically borrowing the output token against the input token and LP collateral. This just-in-time (JIT) borrowing mechanism simulates deep liquidity using relatively small amounts of initial capital—reminiscent of the model described in

Fluid [5], where AMMs tap into the borrowing power of lending protocols. Under optimal conditions, $1 million of deposited capital can back a market with effective depth equivalent to a $50 million pool on a conventional AMM. This structure is particularly valuable for new stable asset issuers, who face high upfront costs in bootstrapping liquidity. Instead of seeding large pools, they can deploy JIT-powered markets that are both capital-efficient and yield-generating, while simultaneously launching a lending market for their token.

EulerSwap also improves capital efficiency by allowing a single vault to act as a liquidity hub for multiple trading pairs. Similar in spirit to Curve's 3pool [6], this design extends to any number of assets. For instance, the USDC vault inside the Euler Yield market could support JIT swaps with other correlated assets such as USDT, USDe, DAI, USDS, and more—concurrently. Alternatively, Euler Prime's USDC vault could be used as a liquidity hub to support trading for hundreds of pairs of long-tail uncorrelated assets. This would reduce the cost of LP capital for new projects, by bringing in yield on the USDC side of the AMM pool, and simultaneously massively increase the liquidity inside Euler Prime.

The protocol introduces a flexible AMM curve (see Figure 1) that supports deep liquidity for short-term trading while helping to keep LPs' positions neutral over longer time horizons. LPs can customise pricing logic, apply asymmetric or single-sided liquidity strategies, and update their parameters by swapping out their EulerSwap operator contract (see below)—giving them far more control than pooled AMM models allow. This flexibility is particularly useful for professional market makers and token issuers who require precision in managing inventory and risk.

Thanks to the modular nature of both protocols, EulerSwap is also fully compatible with the Uniswap v4's hook architecture [7], enabling it to be deployed either as a standalone AMM or integrated within the Uniswap ecosystem. This unlocks access to Uniswap's routing infrastructure and solver network, increasing volume and visibility for LPs without compromising the modular design of EulerSwap.

From the end-user perspective, EulerSwap delivers a seamless, Uniswap-style interface. Behind the scenes, it leverages innovations like JIT liquidity, liquidity hubs, and customisable AMM mechanics—delivering deeper markets, lower costs, and greater control for liquidity providers. For DAOs, token projects, or market makers—EulerSwap offers a powerful tool for unlocking capital efficiency by uniting swap and lending markets in one modular, composable, and customisable system.

2 Just-in-time (JIT) liquidity

Deeper liquidity for swaps can be obtained in EulerSwap using just-in-time (JIT) liquidity. Under the hood, JIT liquidity works by leveraging the unique `operator` functionality of the Ethereum Vault Connector (EVC), a core immutable primitive at the heart of Euler. EVC operators allow Euler users to delegate authority over their account balances to a smart contract. An EulerSwap operator is a specialised contract that manages an LP's account, rebalancing collateral and debt to support swaps and earn swap fees.

Consider the following example.

Suppose that Euler supports borrowing USDT with USDC as collateral at a loan-to-value (LTV) ratio of 0.95, and vice versa. This means that for every $1 of USDC or USDT collateral, a user can borrow up to $0.95 of the other asset.

Now suppose an LP has an Euler account with $1 million of USDC deposited as initial margin liquidity. Using maximum leverage, the account could hypothetically support deposits of $20 million in USDC and debts of $19 million in USDT. Conversely, if the LP swaps the initial $1 million USDC into USDT, it could support $20 million in USDT deposits and $19 million in USDC debt. EulerSwap converts this $40 million swing in notional liquidity between two leveraged positions into usable, virtual liquidity available for swaps.

To facilitate swaps and earn yield, the LP installs an EulerSwap operator on their account. Suppose another user wishes to swap $10 million USDC for USDT. The process proceeds as follows:

1. The swapper sends $10 million USDC to the LP's EulerSwap operator.
2. The operator deposits the received USDC as collateral into Euler.

3. Against this collateral—which now includes the LP's initial margin and the swap input—the operator borrows approximately \$10 million USDT.
4. The borrowed USDT is sent to the user as the swap output.

This is not a 1:1 swap for two reasons: a) the EulerSwap operator charges a fee for facilitating the trade, and b) the swap output is priced using a custom AMM curve, defined by the LP, which accounts for price impact that increases with larger trade sizes.

After the trade, the LP's Euler account now holds \$11 million in USDC deposits and \$10 million in USDT debt. When a reverse-direction swap occurs, the incoming input token is used to repay the outstanding loan, and excess collateral deposits are then returned to the swapper as output. This is essentially the reverse process of the original trade.

Note that if the LP account already holds enough of the output token in Euler, the swap can be fulfilled by withdrawing these deposits rather than initiating a borrow. In such cases, the swap is fulfilled using the LP's existing deposits, avoiding the need to borrow and thus eliminating associated interest costs. By configuring the AMM curve appropriately, an LP can even ensure that swaps remain within the bounds of available deposits, enabling a pool that never borrows. These non-borrowing pools still facilitate efficient swaps and can coexist with unrelated borrow positions in the same account. When the input and output vaults are both used as collateral for a broader leveraged position, swaps between them effectively shift collateral without altering overall exposure—an arrangement sometimes referred to as a "collateral swap."

The AMM curve inside the operator helps ensure that imbalances in the LPs' collateral and debt positions incentivise trades that restore a neutral balance. This dynamic encourages temporary positions that self-resolve through market activity, reducing long-term exposure to interest costs. Although the account incurs some borrowing costs over time, these are offset by swap fees earned through the utilisation of idle capital in Euler's lending markets.

2.1 Liquidation risks

Using JIT liquidity or borrowing against LP positions introduces risks alongside capital efficiency. JIT and borrowing positions incur interest on borrowed assets, which can reduce profitability if not offset by swap fees, interest earned on collateral, or external incentives. Moreover, leveraged LPs are vulnerable to liquidation if collateral values drop or debt ratios rise. This can happen for reasons completely independent of any swap activity. For example, collateral can sometimes incur bad debt. EulerSwap account owners are therefore responsible for monitoring the health of their vaults and should take proactive steps if collateral values drop or debt values rise in order to avoid liquidation.

3 Dynamic hedging and risk-neutral LP strategies

A popular delta-neutral hedging strategy for market making involves borrowing one-side of the pool in order to provide liquidity. For example, a market maker might use USDC collateral to borrow ETH and then supply liquidity to an ETH/USDC pair. This strategy helps a market maker remain relatively neutral with respect to their exposure to ETH. However, such a strategy still exposes the market maker to impermanent loss.

A more sophisticated strategy is therefore to dynamically hedge, by periodically rebalancing the borrowing position. If the ETH price rises, a market maker will swap some of their collateral into ETH and repay a portion of the loan. If the ETH price falls, they will borrow more ETH and swap it into their collateral. While rebalancing is a proven method for mitigating impermanent loss in uncorrelated pairs [4], such strategies often incur significant costs which eat into any benefit of hedging other types of losses. A market maker typically needs to borrow from one protocol to supply liquidity to another protocol, and then pay the costs to strategically rebalance positions, which can be costly.

EulerSwap makes dynamic hedging more efficient. To rebalance, a market maker can simply uninstall and reinstall their EulerSwap operator using off-chain scripts or an on-chain smart contract designed to automate the process. Whilst most dynamic hedges require the market maker to perform periodic swaps, this process can be avoided on EulerSwap when reinstalling the curve in a way designed to encourage arbitrage, allowing the wider market to cover the swap fees.

Finally, another interesting use-case for single-sided JIT is a slightly different hedging strategy that takes advantage of funding rates on perpetual futures markets. A practical strategy could involve deploying an ETH or BTC long-only EulerSwap operator while hedging with a short-only position on a perpetuals platform. Since short strategies tend to benefit from funding rates on perpetual futures exchanges, this strategy can be used to remain approximately delta-neutral whilst earning yield on both Euler + EulerSwap and the external exchange.

4 Capital-efficient liquidity for new tokens

A substantial amount of liquidity in DEX markets is supplied by the projects themselves as protocol-owned liquidity or incentivised by the projects themselves through the use of liquidity mining campaigns. The cost of capital for new projects is extremely high. Whilst their own base ABC tokens might be abundant, the quote asset in the pair, like USDC or WETH, often has a high cost of capital.

A simple use-case for EulerSwap is therefore to create an ABC/USDC or ABC/WETH instance paired with Euler Prime USDC or Euler Prime WETH. This ensures that at least one side of the pool is able to earn lending yield whilst trading is infrequent, lowering the cost of capital for liquidity provision in new assets.

A more interesting use-case emerges when ABC has a robust oracle. In this case, it is possible to create an ABC/USDC pair in which ABC accepts USDC as collateral. This allows single-sided JIT liquidity, where a market maker can borrow ABC as needed to service swaps, but not the other way around. The more ABC price rises, the more borrowing the market maker will be forced to do, bringing organic lending yield to ABC as a result of trading volumes. Whilst a profit-seeking market maker might not want to run such a strategy, note that it can be operated by the project itself, even at a loss, as an alternative to more expensive liquidity mining campaigns. This mechanism can be run at a loss by the project itself, effectively treating the interest paid on borrowed assets as a capital-efficient alternative to traditional liquidity mining.

5 Curve mechanics and virtual reserves

While the operator logic manages execution and capital flow, the behaviour of swaps is ultimately governed by a customisable AMM curve. EulerSwap features a unique curve that allows different amounts of liquidity and different concentrations of liquidity on different sides of the pool based on the reserves available. The space of possible reserves is determined by how much real liquidity an LP has and how much debt their operator is allowed to hold.

Since EulerSwap AMMs do not always hold the assets used to service swaps at all times, they perform calculations based on *virtual* reserves and debt limits, rather than on strictly *real* reserves. Each EulerSwap LP can independently configure virtual reserve levels. These reserves define the maximum debt exposure an AMM will take on. Note that the effective LTV must always remain below the borrowing LTV of the lending vault to prevent liquidation. EulerSwap account owners are also responsible for monitoring the health of their vaults and should take proactive steps if their collateral accrues bad debt or drops in value—since this can happen independently of swap activity, opening up the potential for liquidation if someone performs a large swap on their curve. Additionally, different AMM curves influence whether the maximum virtual reserves are achievable.

The EulerSwap curve passes through a reserve equilibrium point (x_0, y_0), at which the marginal price is defined by:

$$\left.\frac{dy}{dx}\right|_{(x_0,y_0)} = -\frac{p_x}{p_y}. \tag{1}$$

Unlike most AMM curves, which are usually defined by a single convex function, EulerSwap uses a piecewise-defined curve, with different functions guiding trading behaviour either side of the equilibrium point. In the domain $0 < x \leq x_0$, the curve is defined by

$$y = y_0 + \frac{p_x}{p_y}(x_0 - x)\left(c_x + (1 - c_x)\left(\frac{x_0}{x}\right)\right), \tag{2}$$

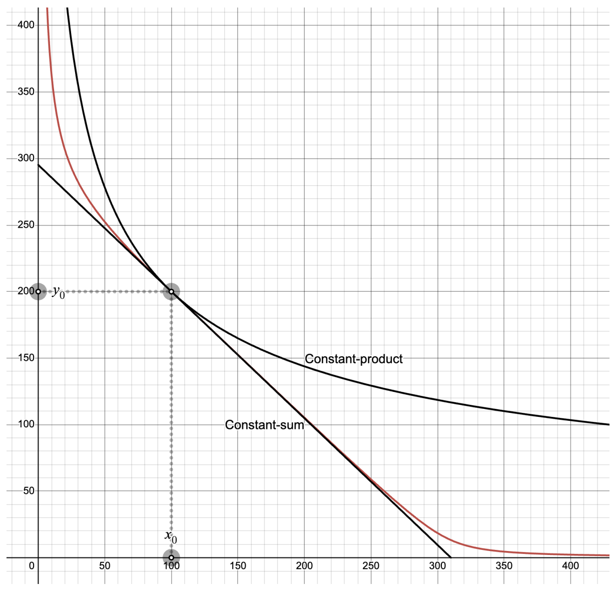

Figure 1: **EulerSwap AMM curve.** The EulerSwap curve (red line) consists of two sides with separate reserve values x_0, y_0 and liquidity concentration parameters c_x, c_y, allowing liquidity to be distributed asymmetrically. This means liquidity can be more or less dense or concentrated on one side of the AMM relative to the other. The exchange rate at equilibrium is determined by the pricing parameters p_x, p_y and is fully flexible. The curve is also available as an interactive model on Desmos to compare its behaviour with traditional constant-sum and constant-product curves (black lines).

with y depending on x. In the region $x_0 < x$, we let x become the dependent variable, so that the domain is $0 < y \leq y_0$, and the curve is defined by

$$x = x_0 + \frac{p_y}{p_x}(y_0 - y)\left(c_y + (1 - c_y)\left(\frac{y_0}{y}\right)\right), \tag{3}$$

with x depending on y. Although defining an AMM using this technique is unusual, both curves have well-defined inverses, allowing trading behaviour to be fully defined for any swap input or output amount in any direction.

The c_x, c_y parameters in the equations are liquidity concentration parameters that control how liquidity is distributed along the curve, with values closer to 1 concentrating liquidity around the equilibrium point, similar to a Curve StableSwap pool [8], and values closer to 0 distributing it across a wider price range, similar to a classic Bancor [9] or Uniswap v2 [10] pool. When the c parameters are both 1, the AMM becomes a constant-sum AMM, and when they are both 0, the AMM becomes a constant-product AMM.

A full description of the mechanics is given in the Appendix.

5.1 Novel use-cases

This flexibility enables EulerSwap to be used for entirely new use cases or to simulate the behaviour of atypical AMM protocols, such as Maker's peg stability module [11]. By configuring asymmetric liquidity curves, EulerSwap can be used as a launchpad for new tokens. For example, concentrating liquidity on the quote asset side (e.g., USDC) while distributing it broadly on the base asset side (e.g., a new token) allows the project to establish a price floor while supporting price discovery. Proceeds from sales can immediately begin earning yield, and the AMM's debt mechanics can enable community-backed liquidity bootstrapping without requiring large treasury allocations upfront.

6 Conclusion

EulerSwap enhances AMMs by leveraging just-in-time liquidity to expand the depth of liquidity available to swappers. By integrating Euler's lending vaults, it enables LPs to earn swap fees on top of their ordinary deposits while offering a customisable AMM curve that supports concentrated, distributed, and asymmetric liquidity structures. These features make EulerSwap adaptable to a wide range of trading strategies. By catering to professional market makers, DAOs, and algorithmic trading systems, EulerSwap positions itself as a foundational infrastructure for efficient, programmable on-chain liquidity.

Acknowledgments

During a security review of EulerSwap, Chris Michel noted similarities between some of its underlying concepts and BlackHoleSwap, a project prototype he had reviewed years earlier. While EulerSwap was designed independently and without influence from that project, the resemblance is significant enough to warrant its recognition here as prior art.

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 Euler Labs or its affiliates and does not necessarily reflect the opinions of Euler Labs, its affiliates or individuals associated with Euler Labs. The opinions reflected herein are subject to change without being updated.

References

[1] Balancer Protocol. *Introducing Boosted Pools*. 2021. URL: https://medium.com/balancer-protocol/100-boosted-pools-powered-by-aave-v3-53fe5b920833.
[2] Bunni Labs. *Bunni v2 Whitepaper*. https://github.com/Bunniapp/whitepaper/blob/main/bunni-v2.pdf. 2025.
[3] Atis E. *Liquidity Provider Strategies for Uniswap v3: Dynamic Hedging*. May 2023. URL: https://atise.medium.com/liquidity-provider-strategies-for-uniswap-v3-dynamic-hedging-9e6858bea8fa.
[4] Hayden Adams et al. *Uniswap v3 Core*. https://uniswap.org/whitepaper-v3.pdf. 2021.
[5] Fluid Protocol. *Introducing Fluid*. https://blog.instadapp.io/fluid/. 2023.
[6] Curve Finance. *3pool: The most used stablecoin pool on Curve*. 2020. URL: https://curve.fi/#/ethereum/pools/3pool.
[7] Uniswap Labs. *Uniswap v4 Core: Hooks and Custom Liquidity*. https://uniswap.org/whitepaper-v4.pdf. 2023.
[8] Michael Egorov. *StableSwap: Efficient Mechanism for Stablecoin Liquidity*. https://curve.fi/files/stableswap-paper.pdf. 2019.
[9] Eyal Hertzog, Guy Benartzi, and Galia Benartzi. *Bancor Protocol: Continuous Liquidity for Cryptographic Tokens through their Smart Contracts*. https://bravenewcoin.com/assets/Whitepapers/Bancor-Protocol-Whitepaper-en.pdf. 2017.
[10] Hayden Adams et al. *Uniswap v2 Core*. https://uniswap.org/whitepaper-v2.pdf. 2020.
[11] MakerDAO. *MIP29: Peg Stability Module*. 2020. URL: https://mips.makerdao.com/mips/details/MIP29.

7 Appendix

7.1 Curve description

Here we describe how the EulerSwap curve generalises the behaviours of both the constant-sum (CSMM) and constant-product market maker (CPMM) curves using liquidity concentration parameters. We begin with an automated market maker (AMM) holding initial liquidity reserves of two assets, X and Y, denoted as x_0 and y_0, respectively. In the absence of trading, the AMM remains at equilibrium at the point (x_0, y_0).

Our goal is to find a curve for a constant-function trading market maker (CFMM) that supports swaps between the two assets with the following properties:

- Passes through the equilibrium point (x_0, y_0).
- Maintains an exchange rate, given by the slope of the AMM curve, of $-p_x/p_y$ at (x_0, y_0).
- Allows liquidity concentration to be adjusted via parameters c_x and c_y, which control the liquidity available for swaps to the left and right of the equilibrium point.

To develop such a function, we first introduce two fundamental AMM curve models.

7.1.1 Constant-sum and constant-product curves

The canonical CSMM and CPMM curves are given by:

$$x + y = x_0 + y_0, \tag{4}$$
$$xy = x_0 y_0. \tag{5}$$

The CSMM is simply a line, whilst the CPMM is a hyperbola. These curves can be thought of as two extremes when it comes to the distribution of liquidity. The CSMM concentrates liquidity at a single exchange rate, whilst the CPMM distributes liquidity across a wide range of different exchange rates. By default, these curves intersect at the equilibrium point (x_0, y_0), where their slopes are:

$$\frac{dy}{dx} = -1, \tag{6}$$
$$\frac{dy}{dx} = -\frac{y}{x}. \tag{7}$$

Since real-world markets often operate at variable exchange rates at equilibrium, we introduce custom pricing parameters p_x and p_y to allow flexibility in defining the slope at the equilibrium point:

$$p_x x + p_y y = p_x x_0 + p_y y_0, \tag{8}$$
$$x^{p_y y_0} y^{p_x x_0} = x_0^{p_y y_0} y_0^{p_x x_0}. \tag{9}$$

The CSMM is now a line with a slope parameterised by the ratio of the pricing parameters, whilst the CPMM is now a *weighted* hyperbola. Taking the derivatives of these equations with respect to x, we obtain:

$$\frac{dy}{dx} = -\frac{p_x}{p_y}, \tag{10}$$
$$\frac{dy}{dx} = -\frac{p_x}{p_y}\frac{x_0 y}{y_0 x}. \tag{11}$$

These results confirm that at equilibrium (x_0, y_0), the slope of both functions is:

$$\frac{dy}{dx}\bigg|_{(x_0,y_0)} = -\frac{p_x}{p_y}.$$

While this weighted CPMM generalises the constant-product formula to support a customisable price at equilibrium, it introduces significant computational complexity. In particular, the exponential form involves power functions of reserves, which are expensive to compute on-chain in the EVM. As a result, this formulation is not directly implementable in practice. To address this, we construct an alternative using artificial reserves—a design trick that preserves the slope and equilibrium properties while greatly simplifying the functional form.

7.1.2 Introducing artificial reserves to create a new, simpler curve

Note that in the interval $0 < x \leq x_0$ swaps should only increase liquidity beyond y_0 and deplete x_0 liquidity. That is, our trading function in this interval need not depend on the initial amount of y_0 liquidity. This suggests that we can split the domain of the AMM curves into two, and replace the real reserve y_0 in the interval $0 < x \leq x_0$ with a carefully chosen artificial reserve y_v designed to eliminate the exponential form in the weighted hyperbola.

Re-arranging equations (8) and (9) into explicit functions of y, we obtain

$$y = y_0 + \frac{p_x}{p_y}(x_0 - x), \tag{12}$$

$$y = y_0 \left(\frac{x_0}{x}\right)^{\frac{p_y y_0}{p_x x_0}}. \tag{13}$$

In this view, it is easy to see that a substitution of $y_0 \to y_v$, given by

$$y_v = x_0 \frac{p_x}{p_y}$$

will eliminate the exponential form in equation (13). We then have equations

$$y = \frac{p_x}{p_y}(2x_0 - x), \tag{14}$$

$$y = \frac{p_x}{p_y}\frac{x_0^2}{x}. \tag{15}$$

Note that simply shifting the curve up or down does not impact the slope or shapes of the curves and therefore has no impact on trading behaviour. Since these curves no longer pass through (x_0, y_0), we therefore correct them by adding back the difference $y_0 - p_x/p_y x_0$. This leads to:

$$y = y_0 + \frac{p_x}{p_y}(x_0 - x), \tag{16}$$

$$y = y_0 + \frac{p_x}{p_y}(x_0 - x)\left(\frac{x_0}{x}\right). \tag{17}$$

Taking the derivatives of these equations with respect to x, we obtain:

$$\frac{dy}{dx} = -\frac{p_x}{p_y}, \tag{18}$$

$$\frac{dy}{dx} = -\frac{p_x}{p_y}\left(\frac{x_0}{x} + \frac{x_0(x_0 - x)}{x^2}\right) = -\frac{p_x}{p_y}\left(\frac{x_0}{x}\right)^2. \tag{19}$$

These results confirm that at equilibrium (x_0, y_0), the slope of both functions is:

$$\frac{dy}{dx}\bigg|_{(x_0,y_0)} = -\frac{p_x}{p_y}.$$

7.1.3 Unifying into a single curve in the region $0 < x \leq x_0$

To create a single unified curve, we introduce a liquidity concentration parameter $c_x \in [0, 1]$ that determines the curve's convexity:

- When $c_x = 1$, the AMM functions as a constant-sum AMM.
- When $c_x = 0$, the AMM behaves as a constant-product-like AMM.
- Intermediate values of c_x create a hybrid trading function, with liquidity that is more or less concentrated around the equilibrium point.

This parameterisation leads to the following equation:

$$y = y_0 + \frac{p_x}{p_y}(x_0 - x)\left(c_x + (1 - c_x)\left(\frac{x_0}{x}\right)\right). \tag{20}$$

This equation is equivalent to equation (2) in the main text. It is implemented as the function 'f()' in Solidity within the 'CurveLib.sol' contract. It serves two main purposes:

- Swap quoting: Providing quotes for y given x in the domain $0 < x \leq x_0$.
- System invariant: Acting as a key invariant within the EulerSwap system (detailed below).

The marginal price anywhere along the curve is given by the derivative, which is given by:

$$\begin{aligned}\frac{dy}{dx} &= \frac{p_x}{p_y}\left[-\left(c_x + (1 - c_x)\left(\frac{x_0}{x}\right)\right) - (x_0 - x)(1 - c_x)\left(\frac{x_0}{x}\right)^2\right] \\ &= -\frac{p_x}{p_y}\left[c_x + (1 - c_x)\left(\frac{x_0}{x}\right)^2\right].\end{aligned} \tag{21}$$

This equation is useful for solvers who wish to know where on the curve a particular price is reached. The equation can also be re-arranged to solve for the x-coordinate corresponding to a particular price, $p = dy/dx$:

$$x = \frac{x_0}{\sqrt{\frac{-\frac{p_y}{p_x}p - c_x}{1 - c_x}}} \tag{22}$$

This equation can be particularly useful for solvers who want to calculate the effective liquidity available between two prices.

To perform as a complete trading function, equation (20) can also be inverted to compute x given y in the range $0 < x \leq x_0$. By rearranging the formula into a quadratic in x, we derive:

$$c_x x^2 + \left(\frac{p_y}{p_x}(y - y_0) - (2c_x - 1)\, x_0\right) x - (1 - c_x)\, x_0^2 = 0. \tag{23}$$

This is a quadratic equation in x that we can solve for using the classic quadratic formula. The components of a classic solution would be given by:

$$\begin{aligned} A &= c_x, \\ B &= \frac{p_y}{p_x}(y - y_0) - (2c_x - 1)\, x_0, \\ C &= -(1 - c_x)\, x_0^2. \end{aligned} \tag{24}$$

With this, our solution for the positive real root is given by:

$$x = \frac{-B + \sqrt{B^2 - 4AC}}{2A}. \tag{25}$$

In some circumstances, particularly when $B \geq 0$, this form of the quadratic equation is numerically unstable. In Solidity, we therefore sometimes use an alternative form called "citardauq" formula, which is a rearrangement of equation (25) to give:

$$x = \frac{2C}{-B - \sqrt{B^2 - 4AC}}. \tag{26}$$

By re-defining $C = (1 - c_x)\, x_0^2$, and noting that $B = -B$ when $B < 0$, we can further simplify these equations to give:

$$x = \begin{cases} \dfrac{B + \sqrt{B^2 + 4AC}}{2A}, & \text{if } B \leq 0 \\ \\ \dfrac{2C}{B + \sqrt{B^2 + 4AC}}, & \text{if } B > 0. \end{cases} \tag{27}$$

This solution is implemented as the function 'fInverse()' in Solidity within the 'CurveLib.sol' contract. Its main purpose is to provide quotes to swappers.

7.1.4 Extending the curve to the $x \geq x_0$ region

To support swaps where the input token lies in the region , we need a symmetric extension of the trading function to the right-hand side of the curve. Rather than defining a new function from scratch, we reflect the existing function by interchanging the roles of x and y. However, this reflection alone is not sufficient: we must also reparameterise the function to use the correct pricing p_x, p_y and concentration c_x, c_y parameters that govern the behaviour of the AMM on the y-side of the pool.

This leads us to the following reparameterised version of equation (20):

$$x = x_0 + \frac{p_y}{p_x}(y_0 - y)\left(c_y + (1 - c_y)\left(\frac{y_0}{y}\right)\right). \tag{28}$$

This equation is equivalent to equation (3) in the main text. It provides a pricing function for swaps in which x is given and y is unknown, and where we are in the domain $0 < y \leq y_0$ (which corresponds to $x \geq x_0$).

Importantly, this is not a new curve, but a reparameterisation of the original left-hand side. It shares the same structural form and equilibrium properties but is applied to the opposite side of the pool using flipped arguments and parameters. In practice, this function is implemented using the same Solidity routine ('f()') as equation (20), with the arguments appropriately swapped to match the trading direction. The inverse of this function can be derived analogously by reparameterising equation (27).

7.2 Invariant

In traditional AMM protocols, the curve is typically defined as an implicit function of y. For example, the classic Uniswap AMM follows a constant-product equation:

$$xy = x_0 y_0 \tag{29}$$

where x_0 and y_0 are the reserves of the equilibrium point. This equation defines an invariant condition, ensuring that any valid swap must satisfy:

$$xy \geq x_0 y_0. \tag{30}$$

This condition guarantees that after any trade, the product of the new reserves remains at least as large as the initial product, ensuring that swaps cannot drain liquidity from the pool.

Rearranging this condition, we can express the invariant as a lower bound for y:

$$y \geq \frac{x_0 y_0}{x}. \tag{31}$$

This means that for any valid trade, the resulting reserve state must lie on or above the AMM curve.

7.2.1 Extending the invariant to EulerSwap

For EulerSwap, we apply a similar principle. Given any reserve state (x, y), we check whether it satisfies an equivalent invariant condition.

From equation (20), for values where $0 < x \leq x_0$, the AMM curve constraint requires a lower bound for y:

$$y \geq y_0 + \frac{p_x}{p_y}(x_0 - x)\left(c_x + (1 - c_x)\left(\frac{x_0}{x}\right)\right). \tag{32}$$

For values where $0 < y \leq y_0$, which is equivalent to $x > x_0$, we simply use a lower bound for x instead:

$$x \geq x_0 + \frac{p_y}{p_x}(y_0 - y)\left(c_y + (1 - c_y)\left(\frac{y_0}{y}\right)\right). \tag{33}$$

These conditions together define the valid liquidity states in EulerSwap, ensuring that the AMM remains balanced while allowing for greater flexibility in liquidity provisioning.