Optimal Liquidity Provision leveraging Atomic AutoSwap Adjustments in CLAMM Protocols

Invarinat Labs
contact@invariant.app

March 2025

Abstract

In concentrated liquidity automated market makers (CLAMMs), liquidity provisioning requires supplying tokens in an exact ratio defined by the current market price and a user-specified price interval. While this design improves capital efficiency, it imposes significant operational constraints: users must calculate and supply a precise token composition, and any deviation from the required ratio results in partial capital utilization or transaction failure.

This paper formulates the problem of constraint-aware, atomic liquidity provisioning from arbitrary initial token balances. We define a formal optimization model in which, given a target tick range, available asset holdings, and user-defined constraints on capital utilization and price impact, the objective is to determine the minimal token swap necessary to enable valid and efficient liquidity provision. We propose an iterative binary search algorithm that solves this problem under monotonic pricing functions and discrete liquidity granularity, and we prove convergence and correctness under reasonable assumptions.

The presented mechanism, AutoSwap, was initially developed during the *Sonic Hackathon*, motivated by the challenges of liquidity provision in early-stage DeFi ecosystems. AutoSwap is currently deployed and tested exclusively on the Sonic network.

Contents

1 Introduction and Problem Definition

1.1 Problem Definition

Automated Market Makers (AMMs) play a foundational role in decentralized finance (DeFi), enabling permissionless token exchange through liquidity pools. In traditional constant-product AMMs (CPAMMs), liquidity is distributed uniformly across all possible prices, which results in substantial capital inefficiency: most of the liquidity remains idle far from the current price.

Concentrated Liquidity AMMs (CLAMMs), such as Uniswap v3 and Invariant, address this limitation by allowing users to allocate liquidity within a specified price interval $[p_l, p_u]$. This design increases fee generation and capital efficiency, but introduces stricter constraints: to add liquidity, users must supply tokens in a precise ratio dictated by the current price and selected tick range.

Formally, for a position size L at price $p \in [p_l, p_u]$, the required token amounts (x, y) are:

$$x = \frac{L(\sqrt{p_u} - \sqrt{p})}{\sqrt{p} \cdot \sqrt{p_u}}, \quad y = L(\sqrt{p} - \sqrt{p_l}).$$

The required ratio $\frac{y}{x}$ depends on both the current price and the range, and is not constant across pools or positions. In practice, users hold arbitrary token balances (x_0, y_0) that often do not match this ratio. This makes direct liquidity provisioning infeasible without a prior swap.

We define the core problem as follows: Given arbitrary token balances (x_0, y_0) and a target tick range $[p_l, p_u]$, determine whether it is possible to:

- perform a swap of part of x_0 or y_0,
- obtain post-swap balances (x', y') that are valid for the position,
- maximize capital utilization $\frac{x'+y'}{x_0+y_0}$,
- ensure execution constraints are respected (e.g., price impact, slippage),
- and execute the entire operation atomically (in one transaction).

This must be done under realistic AMM pricing models, where swap functions and liquidity functions are non-linear and tick-based.

A naive solution involves manual swap execution followed by a separate LP transaction. However, this approach:

- is not atomic,
- exposes users to front-running and reverts due to price shifts,
- often leads to underutilized capital due to inaccurate swap sizing.

To address this, we propose **AutoSwap** — an on-chain mechanism that accepts any initial balance (x_0, y_0), and computes the minimum swap needed to open a valid liquidity position with maximum possible utilization, subject to user-defined constraints. The mechanism is atomic by design and requires no off-chain computation.

AutoSwap was initially developed during the *Sonic Hackathon*, motivated by the challenges of liquidity provisioning in early-stage DeFi environments. It is currently deployed and being tested on the Sonic network, where such constraints and inefficiencies are especially prominent.

1.2 Basis of Liquidity Ratios

Liquidity provisioning is constrained by the underlying mathematical framework of AMMs. A traditional constant-product AMM adheres to the fundamental equation:

$$x \cdot y = k, \tag{1}$$

where:

- x and y represent the reserves of the two assets,

- k is an invariant constant.

The instantaneous exchange rate p of asset X in terms of asset Y is:

$$p = \frac{y}{x}. \tag{2}$$

A liquidity provider intending to contribute new reserves (x', y') must do so in a manner that maintains the existing liquidity ratio:

$$\frac{y'}{x'} = \frac{y}{x}. \tag{3}$$

If this condition is violated, a fraction of the deposited liquidity remains idle, reducing capital efficiency. This issue becomes particularly significant in concentrated liquidity AMMs, where liquidity is only effective within a predefined price interval.

1.3 Concentrated Liquidity in AMMs

Unlike traditional AMMs, where liquidity spans the entire price domain, concentrated liquidity enables providers to allocate funds within a specific range $[p_l, p_u]$, where p_l and p_u denote the lower and upper price bounds, respectively. This methodology presents several advantages:

- **Enhanced capital efficiency**: Liquidity is positioned in proximity to the prevailing price, minimizing unused liquidity.
- **Optimized fee accrual**: Trades occur with greater frequency within active liquidity zones, increasing fee generation.
- **Controlled risk exposure**: Liquidity providers can configure strategic price ranges to mitigate exposure to adverse price movements.

However, this model imposes an additional constraint: liquidity positions are valid only if the deposited asset ratios conform to the required proportion dictated by the selected price range. This necessitates a preliminary swap operation to adjust token balances before liquidity is added.

2 Mathematical Model of Liquidity Provision

2.1 Constant-Product AMM Model: $x \cdot y = k$

Automated Market Makers (AMMs) adhering to the **constant-product invariant** are defined by the equation:

$$x \cdot y = k, \tag{4}$$

where:

- x and y denote the reserves of the two assets within the liquidity pool,
- k is an invariant quantity that remains constant under idealized trading conditions.

Any swap event modifies the reserves according to:

$$(x + \Delta x)(y - \Delta y) = k. \tag{5}$$

Solving for Δy, we obtain:

$$\Delta y = y - \frac{k}{x + \Delta x}. \tag{6}$$

This formulation governs how token Y is removed from the pool when an amount Δx of token X is introduced. The instantaneous price function follows as:

$$p = \frac{dy}{dx} = \frac{y}{x}, \tag{7}$$

a direct consequence of differentiating the constant-product equation.

2.2 Conditions for Balanced Liquidity Provisioning

Liquidity providers seeking to contribute assets to the pool must satisfy the equilibrium constraint:

$$\frac{y'}{x'} = \frac{y}{x}. \tag{8}$$

A liquidity provider whose deposits satisfy this ratio ensures that all contributed funds are actively engaged in trading, thereby maximizing fee accrual. Within a bounded price range $[p_l, p_u]$, the liquidity depth L is computed as:

$$L = \min\left(\frac{x}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}, \frac{y}{\sqrt{p_u} - \sqrt{p_l}}\right). \tag{9}$$

Deposits failing to adhere to this constraint result in inefficient capital allocation, with a fraction of the liquidity remaining inert.

2.3 Deviations from Idealized Models in Practical Implementations

While the constant-product formulation provides an elegant theoretical foundation, real-world implementations introduce several deviations:

- **Discrete Tick-Based Pricing**: Modern AMMs such as Invariant and Uniswap v3 rely on quantized price increments, rather than a continuous price function.
- **Finite Liquidity and Slippage**: The idealized assumption of infinite liquidity does not hold in practice, leading to price slippage during large swaps.
- **Arbitrage-Driven Price Corrections**: AMM prices often diverge from external market prices, necessitating arbitrageurs who restore equilibrium.

A particularly significant departure from theoretical models arises in **concentrated liquidity AMMs**, wherein liquidity is confined to specific price intervals.

2.4 Liquidity Constraints in Discrete Tick-Based Models

Tick-based AMMs, such as Invariant, define price levels at discrete intervals indexed by ticks:

$$p_i = p_0 \cdot (1.0001)^i, \tag{10}$$

where p_i represents the price at tick index i, and p_0 is a reference price. Unlike traditional AMMs, price movements occur in discrete jumps rather than continuously.

Liquidity providers must structure their deposits to conform to tick-based constraints:

$$\frac{y'}{x'} = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{11}$$

Any deviation from this ratio results in an imbalance, leaving part of the capital underutilized.

2.5 Price Discontinuities and Their Impact on Liquidity

In a continuous model, price changes occur smoothly in response to supply and demand. However, in a tick-based AMM, price transitions occur in quantized steps when liquidity crosses a tick boundary. The price change is non-infinitesimal, introducing constraints on liquidity movements:

$$p_{i+1} = p_i \cdot 1.0001. \tag{12}$$

As a result, liquidity providers must account for the possibility that their position may become inactive if price movements cause their range to be exceeded.

2.6 Pre-Swap Adjustments for Optimal Liquidity Provisioning

In tick-based AMMs, liquidity providers must ensure that their deposits adhere to tick-imposed constraints. This often necessitates a preliminary swap operation to balance token proportions before liquidity is added. However, performing this swap in the same pool where liquidity is subsequently provisioned introduces further complications:

- **Price Displacement**: The swap itself impacts the price level, altering the optimal liquidity ratio.
- **Iterative Nature of Adjustment**: A single swap may not perfectly balance liquidity; iterative refinement may be required.
- **Computational Complexity**: The optimal pre-swap adjustment cannot always be determined analytically and may necessitate numerical approximation.

The presence of ticks necessitates a structured computational approach to determining the optimal pre-swap amount, avoiding unnecessary slippage and ensuring that all deposited liquidity remains active.

2.7 Computational Strategies for Pre-Swap Optimization

The discrete nature of tick-based AMMs precludes an exact analytical solution to the pre-swap problem. Instead, liquidity providers must employ computational techniques such as:

- **Binary Search Methods**: Iteratively refining the swap amount to converge on the optimal token ratio.
- **Newton's Method**: A root-finding approach to efficiently approximate the correct balance adjustment.
- **Empirical Backtesting**: Evaluating the performance of different swap strategies on historical data to derive heuristics for optimal execution.

2.8 Mathematical Implications of Discrete Liquidity Models

Given the constraints imposed by discrete ticks, the traditional formulation of liquidity depth:

$$L = \frac{\Delta y}{\sqrt{p_u} - \sqrt{p_l}}, \tag{13}$$

must be modified to incorporate tick granularity. The revised liquidity equation under a tick-based model becomes:

$$L = \frac{\Delta y}{\sqrt{\tau(u)} - \sqrt{\tau(l)}}, \tag{14}$$

where $\tau(i) = p_0 \cdot (1.0001)^i$ is the tick function. This modification ensures that liquidity depth is defined in terms of tick-aligned price levels.

2.9 Conclusion

The classical AMM invariant $x \cdot y = k$ is insufficient for describing liquidity behavior in tick-based models. Instead, a more rigorous treatment must consider:

3 Challenges in Tick-Based Liquidity Models

3.1 Tick-Based Liquidity Constraints in Concentrated AMMs

Unlike traditional AMMs, where liquidity is distributed continuously over the entire price spectrum, concentrated liquidity AMMs partition liquidity into discrete price intervals known as **ticks**. These constraints fundamentally alter the mechanics of liquidity provisioning and trade execution.

A tick-based AMM quantizes price levels according to a base price p_0 and a predefined tick spacing Δi, yielding a discrete price sequence:

$$p_i = p_0 \cdot (1.0001)^i, \tag{15}$$

where p_i denotes the price corresponding to tick index i. Given a selected liquidity range $[p_l, p_u]$, a liquidity provider must ensure that liquidity is allocated within the tick boundaries:

$$p_l = p_0 \cdot (1.0001)^{i_l}, \quad p_u = p_0 \cdot (1.0001)^{i_u}. \tag{16}$$

Since swaps in tick-based AMMs progress through discrete price levels rather than adjusting continuously, liquidity providers must align their asset ratios with the tick-determined constraints:

$$\frac{\Delta y}{\Delta x} = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{17}$$

A mismatch in this ratio results in inefficient liquidity placement, as part of the deposited capital remains inactive until price movements bring it within the active range.

3.2 Theoretical Continuous Swaps vs. Discrete Tick-Based Execution

In classical AMM theory, swaps occur along a continuously differentiable price curve, with infinitesimal price adjustments dictated by the liquidity invariant. However, in tick-based AMMs, price movements are inherently discrete:

$$x_{i+1} \cdot y_{i+1} = k, \quad \text{where} \quad p_{i+1} = p_i \cdot (1.0001)^{\Delta i}. \tag{18}$$

This fundamental distinction introduces several notable consequences:

- **Liquidity non-uniformity**: Certain ticks may accumulate substantial liquidity, whereas others remain devoid of liquidity.
- **Price discontinuities**: A swap that moves the price from one tick to another may experience liquidity fragmentation, leading to irregular price behavior.
- **Execution inefficiencies**: If a tick lacks sufficient liquidity, traders may experience excessive slippage beyond what is predicted by theoretical models.

3.3 Impact of Tick Granularity on Swap Efficiency

The spacing between ticks, often referred to as **tick granularity**, directly influences the execution properties of swaps. A finer tick spacing provides increased liquidity resolution, resulting in:

- More precise liquidity placement, reducing gaps in the liquidity profile.
- Decreased slippage due to a higher density of price levels.
- Greater control over price execution, mitigating adverse price impact.

Conversely, a coarser tick spacing leads to:

- Fewer executable price levels, restricting trade flexibility.
- More pronounced price jumps when crossing tick thresholds.

- Increased inefficiency for large swaps, as they must traverse multiple ticks.

Given a tick spacing Δi, the set of accessible price levels is given by:

$$p_n = p_0 \cdot (1.0001)^{n \cdot \Delta i}. \tag{19}$$

As swaps traverse multiple tick boundaries, execution must account for the liquidity available at each tick, necessitating an iterative computational approach.

3.4 Absence of a Closed-Form Solution in Discrete AMMs

In a continuous AMM model, the output of a swap can be derived by integrating the price function over the swap interval:

$$\int_{x_0}^{x_1} p(x)\, dx. \tag{20}$$

In a tick-based AMM, this integral must be replaced by a discrete summation over tick intervals:

$$\sum_{i=i_0}^{i_1} \Delta x_i \cdot p_i. \tag{21}$$

This fundamental shift introduces several mathematical challenges:

1. **Discontinuity**: The discrete nature of ticks prevents direct application of continuous calculus-based swap formulas.
2. **Liquidity fragmentation**: The amount of liquidity at each tick determines whether further price movement is feasible.
3. **Tick-dependent constraints**: Execution mechanics depend not only on price but also on the distribution of liquidity at discrete tick intervals.

These factors preclude a closed-form solution for swap execution, requiring an alternative approach based on iterative computation.

3.5 Pre-Swap Optimization for Liquidity Provisioning

To ensure efficient liquidity placement, a liquidity provider must first adjust their token holdings to match the required ratio for their selected price range. If an initial holding (x_1, y_1) deviates from the optimal ratio, a preliminary **pre-swap** is required to align the balances before liquidity provisioning.

The optimal swap amount Δx must satisfy:

$$\frac{y_1 + \Delta y}{x_1 - \Delta x} = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{22}$$

Rearranging for Δx:

$$\Delta x = x_1 - \frac{y_1 + \Delta y}{\frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}}. \tag{23}$$

Since Δy is itself a function of liquidity constraints and swap execution paths, this equation lacks a direct analytical solution. Instead, an iterative numerical approach must be employed.

3.6 Computational Approach to Pre-Swap Optimization

Given the absence of a closed-form solution, practical implementation requires iterative numerical methods such as:

- **Binary search algorithms**: Efficiently refining the swap amount to converge on the optimal token ratio.
- **Newton's method**: Approximating the root of the liquidity constraint function for rapid convergence.
- **Backtesting and simulation**: Empirically validating swap strategies against historical trade data to optimize performance.

3.7 Mathematical Reformulation in Tick-Based AMMs

To incorporate tick constraints into liquidity depth calculations, the conventional liquidity equation:

$$L = \frac{\Delta y}{\sqrt{p_u} - \sqrt{p_l}}, \tag{24}$$

must be adjusted for discrete tick values, yielding:

$$L = \frac{\Delta y}{\sqrt{\tau(u)} - \sqrt{\tau(l)}}, \tag{25}$$

where $\tau(i) = p_0 \cdot (1.0001)^i$ represents the tick function.

4 Iterative Approach to Pre-Swap Optimization

4.1 Need for Token Balancing Before Adding Liquidity

Efficient liquidity provisioning in a concentrated liquidity AMM requires that the deposited asset ratio aligns precisely with the liquidity function for the chosen price range. Given an initial asset allocation (x_1, y_1), a liquidity provider must ensure that their token holdings conform to the required proportion dictated by the price range $[p_l, p_u]$.

A liquidity position is constrained by two fundamental conditions:

- **Liquidity Ratio Constraint**: The provided token amounts must satisfy the fundamental liquidity equation:
$$\frac{y'}{x'} = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{26}$$
- **Tick-Based Constraint**: Liquidity is active only at discrete price ticks, leading to the possibility that an improperly balanced deposit leaves part of the capital unused.

If the initial asset ratio $\frac{y_1}{x_1}$ deviates from Equation (26), the excess amount remains idle. To prevent capital inefficiency, a pre-swap must be performed to adjust the token holdings to match the required liquidity ratio before liquidity provisioning.

4.2 Derivation of the Swap Amount for Optimal Liquidity Provisioning

The objective of the pre-swap is to determine the precise swap amount Δx such that the final token allocation satisfies the liquidity condition:

$$\frac{y_1 + \Delta y}{x_1 - \Delta x} = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{27}$$

As the swap follows a constant-product AMM mechanism:

$$(x + \Delta x)(y - \Delta y) = k, \tag{28}$$

we express Δy as:

$$\Delta y = y - \frac{k}{x + \Delta x}. \tag{29}$$

Substituting into the liquidity condition:

$$\frac{y_1 + \left(y - \frac{k}{x+\Delta x}\right)}{x_1 - \Delta x} = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{30}$$

4.3 Mathematical Formulation of Pre-Swap Calculations

Rearranging Equation (30) results in a quadratic equation for Δx:

$$A(\Delta x)^2 + B\Delta x + C = 0, \tag{31}$$

where:

$$A = 1, \tag{32}$$

$$B = -\left(x_1 + \frac{y_1}{m}\right), \tag{33}$$

$$C = \frac{k}{m} - x_1 y_1. \tag{34}$$

Here, m is the required liquidity ratio:

$$m = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{35}$$

Solving for Δx:

$$\Delta x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}. \tag{36}$$

However, in a tick-based AMM, this theoretical solution is not directly applicable due to:

- Quantized price movement, as swaps execute at discrete tick levels rather than continuously.
- Variability in liquidity availability across ticks, affecting execution path constraints.
- Price movement induced by the swap itself, dynamically influencing the required ratio.

4.4 Effectiveness of Iterative Approximations

Since a closed-form analytical solution does not adequately capture tick-based constraints, a numerical iterative method is required. The optimization follows these steps:

1. Compute the ideal liquidity ratio for the selected tick range.
2. Simulate candidate swap amounts to approximate the optimal swap quantity.
3. Adjust search bounds dynamically, using binary search or Newton's method for rapid convergence.
4. Validate the final swap amount by ensuring maximized liquidity utilization.

4.4.1 Iterative Search Algorithm

The search algorithm begins by defining an upper and lower bound for the swap amount:

$$\text{Lower Bound:} \quad \Delta x_{\min} = 0, \tag{37}$$

$$\text{Upper Bound:} \quad \Delta x_{\max} = x_1. \tag{38}$$

A midpoint candidate swap amount is tested:

$$\Delta x_{\text{mid}} = \frac{\Delta x_{\min} + \Delta x_{\max}}{2}. \tag{39}$$

Using this candidate, the resulting liquidity balance is computed, and the ratio is evaluated:

$$\text{if } \frac{y_1 + \Delta y}{x_1 - \Delta x} > \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}, \quad \text{then decrease } \Delta x. \tag{40}$$

Otherwise, increase Δx, repeating until convergence.

4.5 Practical Implementation in the Invariant Protocol

The iterative approach is implemented programmatically as follows:

```
let low = new BN(0);
let high = amountX;
let precision = amountX.mul(minPrecision).div(DENOMINATOR);

while (low.add(precision).lt(high)) {
    let mid = low.add(high).addn(1).divn(2);
    const sim = simulateSwap({
        xToY,
        byAmountIn,
        swapAmount: mid,
        ...swap
    });

    if (sim.status === SimulationStatus.Ok) {
        high = mid;
    } else {
        low = mid;
    }
}
```

This efficiently converges on the optimal swap amount while respecting tick-based liquidity constraints.

5 Limitations of Theoretical Models and the Strength of Iterative Methods

5.1 Why the Theoretical Model Fails Under Tick-Based Liquidity Constraints

The classical AMM model assumes smooth price transitions governed by the constant-product invariant:

$$x \cdot y = k. \tag{41}$$

Under this assumption, price adjustments occur continuously as liquidity fluctuates. However, in tick-based AMMs, liquidity is discretized into predefined price levels:

$$p_i = p_0 \cdot (1.0001)^i. \tag{42}$$

This discretization imposes structural constraints that invalidate direct theoretical formulations. The primary limitations of closed-form solutions in tick-based AMMs include:

- **Discontinuous Price Adjustments:** Price changes occur in discrete jumps rather than continuous shifts.
- **Liquidity Fragmentation:** Some ticks accumulate liquidity, while others remain empty, disrupting smooth execution.
- **Execution Constraints:** Theoretical models assume continuous paths for price evolution, whereas real execution follows a tick-by-tick progression.

If a closed-form expression for the optimal swap amount Δx is derived from continuous liquidity assumptions:

$$\Delta x = f(x_1, y_1, p_l, p_u), \tag{43}$$

it fails to account for:

- The impact of crossing multiple ticks with varying liquidity.
- The execution trajectory, which depends on real market conditions.
- Slippage and price impact due to liquidity constraints.

For these reasons, an iterative numerical approach is required to determine the **actual** swap amount that aligns liquidity optimally within a tick-based framework.

5.2 Proof That an Iterative Approach Converges to the Optimal Swap Amount

An iterative method such as binary search guarantees convergence to an optimal solution, provided that:

- The function defining swap outcomes is **monotonic**.
- The target liquidity ratio condition is **well-posed**.
- The search space for the swap amount Δx is **bounded**.

To establish convergence, define:

$$g(\Delta x) = \frac{y_1 + \Delta y}{x_1 - \Delta x} - \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{44}$$

Since $g(\Delta x)$ is monotonic, applying binary search ensures convergence to the unique solution:

$$\lim_{n \to \infty} \Delta x_n = \Delta x^*, \tag{45}$$

where Δx^* represents the optimal swap amount that satisfies the liquidity ratio constraint.

5.3 Mathematical Convergence of the Binary Search for Swap Amounts

The binary search algorithm operates as follows:

$$\text{Initialize bounds:} \quad \Delta x_{\min} = 0, \quad \Delta x_{\max} = x_1. \tag{46}$$

$$\text{Compute midpoint:} \quad \Delta x_{\text{mid}} = \frac{\Delta x_{\min} + \Delta x_{\max}}{2}. \tag{47}$$

$$\text{Evaluate function:} \quad g(\Delta x_{\text{mid}}). \tag{48}$$

If $g(\Delta x_{\text{mid}}) > 0$, decrease $\Delta x_{\max}$; otherwise, increase $\Delta x_{\min}$.
Each step **halves the search interval**, ensuring convergence in:

$$O(\log_2 N) \tag{49}$$

iterations, where N represents the resolution of Δx.

5.4 Error Bound for the Binary Search Algorithm

The final error ε in the estimated swap amount Δx^* is:

$$\varepsilon_n = \frac{\Delta x_{\max} - \Delta x_{\min}}{2^n}. \tag{50}$$

After n iterations, the error satisfies:

$$\varepsilon_n \leq \frac{\text{Initial Interval Width}}{2^n}. \tag{51}$$

Thus, to achieve a precision of ε, the required number of iterations is:

$$n = \log_2 \left(\frac{\text{Initial Interval Width}}{\varepsilon} \right). \tag{52}$$

5.5 Computational Complexity of Iterative Methods

Compared to closed-form solutions, the iterative approach demonstrates superior adaptability to real-world constraints:

- Lower sensitivity to liquidity fragmentation.
- Logarithmic time complexity in binary search ($O(\log N)$).
- Robustness against execution path variations in tick-based AMMs.

5.6 Simulation-Based Validation of the Iterative Method

To ensure practical efficacy, the iterative method is validated through swap simulations:

```
while (low.add(precision).lt(high)) {
    let mid = low.add(high).addn(1).divn(2);
    const sim = simulateSwap({
        xToY,
        byAmountIn,
        swapAmount: mid,
        ...swap
    });

    if (sim.status === SimulationStatus.Ok) {
        high = mid;
    } else {
        low = mid;
    }
}
```

This simulation-driven strategy guarantees:

- **Correct execution** within tick-based liquidity constraints.
- **Minimal slippage** by dynamically optimizing swap execution.
- **Maximized capital efficiency** for liquidity providers.

5.7 Final Remarks on Theoretical vs. Iterative Models

The iterative approach not only replicates theoretical efficiency in a continuous setting but also outperforms closed-form expressions in practical AMM implementations. The algorithm achieves:

- Rapid convergence with logarithmic complexity.
- Exact liquidity ratio matching under discrete tick constraints.
- Superior performance in real swap execution compared to theoretical approximations.
- Robust handling of market dynamics, ensuring optimal capital utilization.

The subsequent section will extend this analysis by incorporating full-scale simulations, comparing theoretical models, iterative approximations, and real-execution outcomes in an empirical framework.

6 Refining Swap Calculations Using the Iterative Method

6.1 How the Iterative Method Refines the Swap Calculation

The iterative method refines swap calculations by dynamically adjusting the swap amount Δx to satisfy the required liquidity ratio under tick-based constraints. Unlike direct analytical formulas, which assume smooth price transitions, the iterative approach:

- **Accounts for discrete price jumps**, inherent to tick-based AMMs.
- **Minimizes price impact dynamically**, ensuring the swap does not unnecessarily shift the price beyond the intended range.
- **Optimizes capital efficiency**, ensuring both assets are fully utilized within the selected liquidity range.
- **Handles non-uniform liquidity distributions**, where liquidity depth varies across ticks.

A direct formula fails to predict swap execution precisely, as the AMM invariant holds at each tick level, but the swap execution path introduces complexity. The iterative approach resolves this by refining the swap amount step-by-step until the optimal token balance is achieved.

6.2 Formal Description of the Iterative Procedure

The problem of determining the swap amount can be framed as an optimization problem:

$$\text{Find } \Delta x \text{ such that } \frac{y_1 + \Delta y}{x_1 - \Delta x} = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{53}$$

Since the AMM invariant holds:

$$(x + \Delta x)(y - \Delta y) = k, \tag{54}$$

we derive:

$$\Delta y = y - \frac{k}{x + \Delta x}. \tag{55}$$

6.2.1 Algorithm: Iterative Refinement of Swap Amount

The iterative procedure follows these steps:

1. **Initialize Bounds:** Define an initial search range for Δx.
2. **Compute Midpoint Swap:** Evaluate a midpoint swap amount Δx_{mid}.
3. **Simulate Resulting Liquidity Balance:** Compute Δy and check whether the liquidity ratio condition is satisfied.
4. **Refine Search:**
   - If $\frac{y_1+\Delta y}{x_1-\Delta x}$ exceeds the required ratio, decrease Δx.
   - If it is below the required ratio, increase Δx.
5. **Repeat Until Convergence:** Stop when the error:

$$\left| \frac{y_1 + \Delta y}{x_1 - \Delta x} - \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}} \right| \tag{56}$$

   falls below a predefined threshold.

6.3 Convergence Guarantees for Optimal Swap Amounts

The iterative approach employs a binary search framework, ensuring logarithmic convergence. Convergence is guaranteed if:

- The function mapping Δx to the liquidity ratio is **monotonic**.
- The search space is **bounded**, specifically $\Delta x \in [0, x_1]$.
- The liquidity ratio function is **continuous** within each tick range.

At iteration n, the error bound satisfies:

$$\varepsilon_n = \frac{\Delta x_{\max} - \Delta x_{\min}}{2^n}. \tag{57}$$

Since each iteration halves the error, the number of iterations required for accuracy ε is:

$$n = \log_2 \left(\frac{\text{Initial Interval Width}}{\varepsilon} \right). \tag{58}$$

Thus, the iterative approach converges in logarithmic time:

$$O(\log N). \tag{59}$$

6.4 Necessity of Numerical Search Over Explicit Formulas

An explicit formula for Δx is infeasible because:

- Tick-based constraints cause price discontinuities.
- Liquidity fragmentation alters the execution trajectory.
- Dynamic slippage effects modify the actual swap path.

The iterative method provides a practical alternative by:

- **Adapting dynamically** to liquidity depth variations.
- **Minimizing price displacement** by selecting the smallest valid swap amount.
- **Ensuring full capital efficiency** by precisely satisfying the liquidity constraints.

6.5 Implementation in Invariant Protocol

The iterative algorithm is implemented using a binary search procedure to refine the swap amount within the predefined tick range. The core implementation follows:

```
let low = new BN(0);
let high = amountX;
let precision = amountX.mul(minPrecision).div(DENOMINATOR);

while (low.add(precision).lt(high)) {
    let mid = low.add(high).addn(1).divn(2);
    const sim = simulateSwap({
        xToY,
        byAmountIn,
        swapAmount: mid,
        ...swap
    });

    if (sim.status === SimulationStatus.Ok) {
        high = mid;
    } else {
        low = mid;
    }
}
```

This procedure guarantees:

- Adaptive refinement, adjusting swap size based on liquidity availability.
- Convergence within a finite number of iterations, ensuring computational efficiency.
- Minimal price impact, preserving favorable liquidity conditions.

6.6 Atomic Transaction Execution in Invariant Protocol

A defining feature of the Invariant Protocol is its ability to execute both the swap operation and the liquidity position creation within a single atomic transaction. This ensures that liquidity provisioning is always performed with the correct asset ratio, eliminating inefficiencies that arise from separate transactions.

6.6.1 Atomicity and Full Capital Utilization

Unlike traditional AMMs, where liquidity providers must first execute a separate swap transaction before adding liquidity, Invariant's atomic execution guarantees:

- Full capital utilization within the same pool, meaning that both the swap and liquidity provision occur within the same liquidity structure without requiring multiple transactions.
- No idle assets—the protocol ensures that all available liquidity contributes to the pool, avoiding scenarios where assets remain unutilized due to incorrect ratios.
- Automatic balance adjustments—if the initial token holdings do not satisfy the required ratio, the protocol calculates and executes the precise swap amount needed to match the liquidity conditions.

The atomicity condition enforces:

$$\mathcal{T}_{\text{atomic}}(x_0, y_0) = \begin{cases} \text{execute swap} & \text{if swap is needed,} \\ \text{add liquidity} & \text{if post-swap balance is correct,} \\ \text{revert transaction} & \text{if conditions are not met.} \end{cases} \tag{60}$$

6.6.2 Optimized Swap Amounts and Risk Management

One of the critical risks in liquidity provisioning is price fluctuations between the swap and liquidity position creation. In traditional AMMs, this risk arises due to separate transactions for the swap and deposit, exposing liquidity providers to adverse price movements. Invariant mitigates this risk through:

- Optimized swap execution, where the swap is dynamically calculated to minimize price impact and slippage.
- Risk-managed position opening, ensuring that all necessary swap adjustments are completed before the liquidity deposit is finalized.
- Reversion safeguards, meaning that if any part of the transaction cannot execute within the predefined conditions, the entire transaction is automatically reverted.

This approach guarantees that liquidity positions are never exposed to unnecessary price shifts, preserving capital efficiency and execution predictability.

6.6.3 High Level of Customization for Market Adaptation

Unlike conventional AMMs, which provide rigid liquidity provisioning mechanisms, Invariant's atomic transactions offer advanced customization features, including:

- User-defined swap parameters—liquidity providers can specify the maximum allowable swap deviation, ensuring that execution remains within pre-set risk thresholds.
- Minimum liquidity utilization enforcement, allowing users to define a required minimum percentage of capital that must be deployed in the liquidity position.
- Adaptability to market volatility—by dynamically adjusting the swap amount and position size, the protocol optimizes capital deployment based on real-time price fluctuations.

6.6.4 Modularity and Interoperability

A fundamental advantage of the Invariant Protocol is the modular structure of its swap execution mechanism. Unlike rigid AMM implementations, Invariant's swap algorithm operates independently of the underlying trading venue, allowing seamless integration with multiple liquidity sources.

Mathematical Representation of Modular Swap Execution: Let $\mathcal{S}$ represent the swap execution function, which determines the optimal amount Δx to be swapped before liquidity provisioning. In the general case:

$$\mathcal{S}(\Delta x, \mathcal{M}) \to (\Delta y, p) \tag{61}$$

where:

- Δx is the input token amount,
- $\mathcal{M}$ is the selected market mechanism (e.g., AMM, order book, or aggregator),
- Δy is the output token amount after execution,
- p is the resulting market price.

The modularity of the system ensures that $\mathcal{M}$ can be dynamically replaced based on the most optimal execution path:

$$\mathcal{M} \in \{\mathcal{A}_{\text{CP}}, \mathcal{A}_{\text{CL}}, \mathcal{A}_{\text{CLOB}}, \mathcal{A}_{\text{AGG}}\} \tag{62}$$

where:

- $\mathcal{A}_{\text{CP}}$ represents a constant-product AMM ($x \cdot y = k$).

- $\mathcal{A}_{\text{CL}}$ represents a concentrated liquidity AMM.
- $\mathcal{A}_{\text{CLOB}}$ represents an order book model where swaps are executed at the best available limit orders.
- $\mathcal{A}_{\text{AGG}}$ represents a DEX aggregator, dynamically selecting the most liquid execution route.

By allowing $\mathcal{M}$ to be interchangeable, the protocol dynamically adapts to market conditions, ensuring the most efficient execution strategy.

Modular Execution Flow:

1. The swap function $\mathcal{S}$ is called with the desired input amount Δx.
2. The optimal execution mechanism $\mathcal{M}$ is selected based on liquidity depth, slippage, and fee considerations.
3. The swap executes through the chosen mechanism ($\mathcal{A}_{\text{CP}}, \mathcal{A}_{\text{CL}}, \mathcal{A}_{\text{CLOB}}, \mathcal{A}_{\text{AGG}}$), producing output Δy.
4. The liquidity provisioning step ensures that the post-swap token balance satisfies:

$$\frac{y_0 + \Delta y}{x_0 - \Delta x} = \frac{\sqrt{p_u} - \sqrt{p_l}}{\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}}. \tag{63}$$

Benefits of a Modular Swap Execution System:

- Optimized Execution Routes – The protocol dynamically selects between AMMs, order books, and aggregators, ensuring the most capital-efficient swap.
- Cross-Liquidity Adaptability – Swaps can be executed across multiple trading venues, mitigating liquidity fragmentation.
- Market-Agnostic Design – Future integrations can introduce new trading models without modifying the core algorithm.

6.6.5 Competitive Advantage Over Traditional AMMs

Most AMMs, including Balancer, PancakeSwap, SushiSwap, Orca, and QuickSwap, are designed as static, single-model liquidity pools, limiting their efficiency in volatile or fragmented markets.

- Traditional AMMs require users to interact with a single liquidity model (e.g., constant-product pools), leading to higher slippage in thinly liquid pools.
- Order books are not natively integrated in most AMM models, meaning that traders cannot access deeper liquidity through limit orders.
- DEX aggregators operate separately from AMMs, requiring external routing rather than seamless in-protocol execution.

In contrast, Invariant's modular architecture enables:

- Real-time liquidity adaptation, ensuring that swaps always execute through the optimal mechanism.
- Seamless interoperability between AMMs and order books, allowing hybrid execution strategies.
- Dynamic cross-pool capital allocation, ensuring that liquidity is always deployed in the most efficient venue.

7 Contribution and Future Perspectives

7.1 Contributions to LPs Usability and Capital Efficiency

AutoSwap directly addresses a long-standing challenge in concentrated liquidity AMMs: the friction of providing liquidity when a user's token balances do not match the required ratio. In current systems, this requires manual swaps, ratio calculations, and execution across multiple transactions. This process introduces risk, complexity, and inefficiency.

AutoSwap resolves this by wrapping the entire swap-and-provide flow into a single atomic operation. The mechanism automatically determines the optimal amount to swap based on the user's current balances, price range, and execution constraints. This allows users to provide liquidity with minimal effort, regardless of their initial holdings.

From a user experience standpoint, AutoSwap drastically lowers the barrier to entry for liquidity provision. It is especially helpful for new users, mobile-first interfaces, and emerging ecosystems where infrastructure is still evolving. For advanced users and automated strategies, AutoSwap enables capital-efficient liquidity provisioning with clear guarantees on utilization and price impact — making it suitable for bots, vaults, and on-chain strategies.

Overall, AutoSwap bridges the gap between the theoretical benefits of concentrated liquidity and its real-world usability — reducing friction, improving execution reliability, and unlocking broader participation.

7.2 Innovation Beyond Existing Solutions

While previous solutions have attempted to address token rebalancing in CLAMMs, none have delivered a complete, atomic, and constraint-aware experience. The most advanced implementation before AutoSwap was Orca's AutoSwap, which introduced token pre-swapping before liquidity provisioning. However, Orca's version has several limitations:

- It does not guarantee atomic execution — swaps and LP creation happen in separate transactions.
- It supports only a fixed swap path tied to a CLAMM pool.
- It lacks full customization of constraints such as minimum utilization or maximum price impact.

AutoSwap introduces several key innovations:

- **Atomic execution:** Swap and LP creation occur within a single transaction, removing the risk of mid-process price shifts.
- **Binary search optimization:** Efficiently determines the minimal swap amount needed to reach a valid position.
- **Constraint validation:** Supports user-defined thresholds for utilization, slippage, and price impact.
- **Source-agnostic design:** Liquidity can be sourced from any system — including CLAMMs, CPAMMs, order books, or DEX aggregators.

Together, these design choices unlock the full potential of token rebalancing in CLAMMs. AutoSwap provides a generalized, modular mechanism that is not only safer and more efficient than previous solutions, but also ready to adapt to evolving DeFi architectures.

7.3 Ecosystem Impact and Future Integration

While AutoSwap is fully functional and optimized today, its architecture is designed to scale with the DeFi ecosystem. It is modular and agnostic to the swap source, meaning it can be combined with different pricing models and integrated into third-party protocols or frontends.

In mature environments, we expect AutoSwap to be combined with external `quote()` functions powered by DEX aggregators. These can provide optimal swap routes — including multihop paths —

before executing the final position in one atomic transaction. Since AutoSwap does not assume any specific source of liquidity, such extensions require no changes to the core optimization logic.

By abstracting away the technical details of token rebalancing, AutoSwap makes capital-efficient liquidity provision accessible to a much broader range of users and protocols. It serves as both a backend primitive for protocol developers and a usability layer for end users.

As DeFi continues to expand across new networks and applications, tools like AutoSwap will be essential for lowering operational friction and improving user confidence. In that sense, AutoSwap is not just an algorithmic innovation — it is an enabler of more efficient, inclusive, and composable DeFi infrastructure.