CS765 Assignment 3 Report

Indian Institute of Technology, Bombay

Ananya Kulashreshtha : 22B0906
Nitin Singh Patel : 22B0974
Tanish Agarwal : 22B0978

April 13, 2025

Key Metrics Plots over time

This report summarizes the behavior of the decentralized exchange (DEX) implemented using a constant product AMM. The following plots illustrate various metrics collected over time during the simulation.

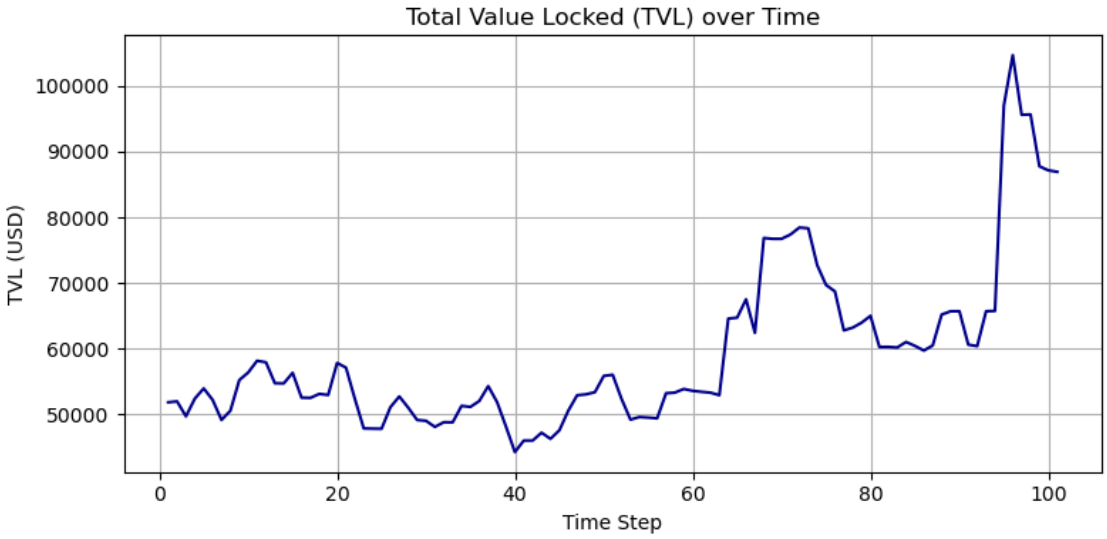

Figure 1: Total Value Locked (TVL) over time

As swap ins and swap outs are done, total value locked goes up and down. Also, LPs remove and add liquidity, that also affects the TVL.

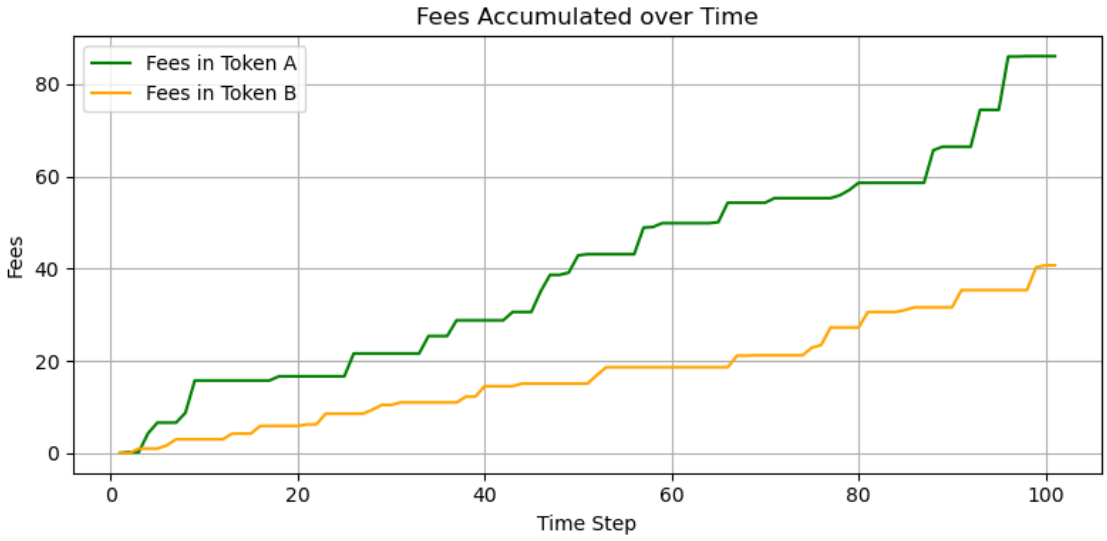

Figure 2: Fees collected in Token A and Token B over time

We observe that cumulative fees steadily increase with trading activity. The staircase pattern indicates that fees are collected in discrete steps corresponding to individual transactions.

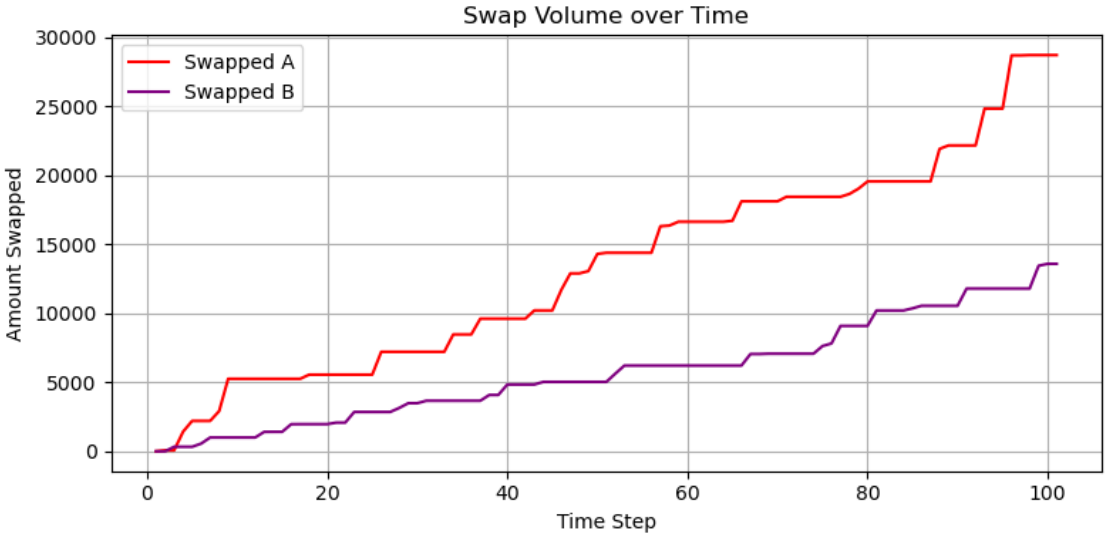

Figure 3: Swap volume of Token A and Token B over time

Cumulative swap values for both tokens and hence, always increasing.

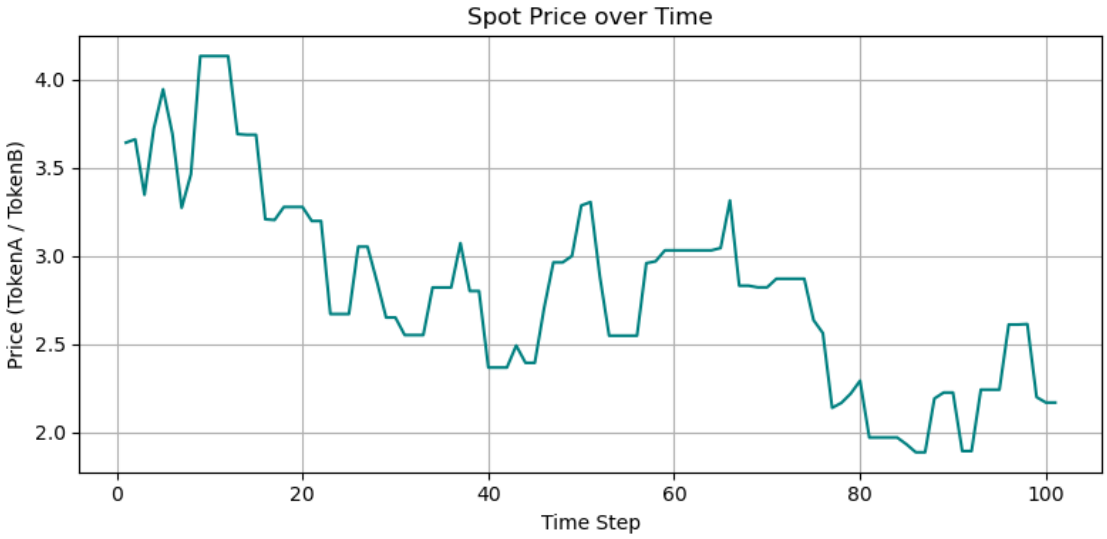

Figure 4: Spot price (Reserve_A / Reserve_B) over time

As reserve A and reserve B change in DEX due to swaps, their ratio also changes and hence, the spot price goes up and down.

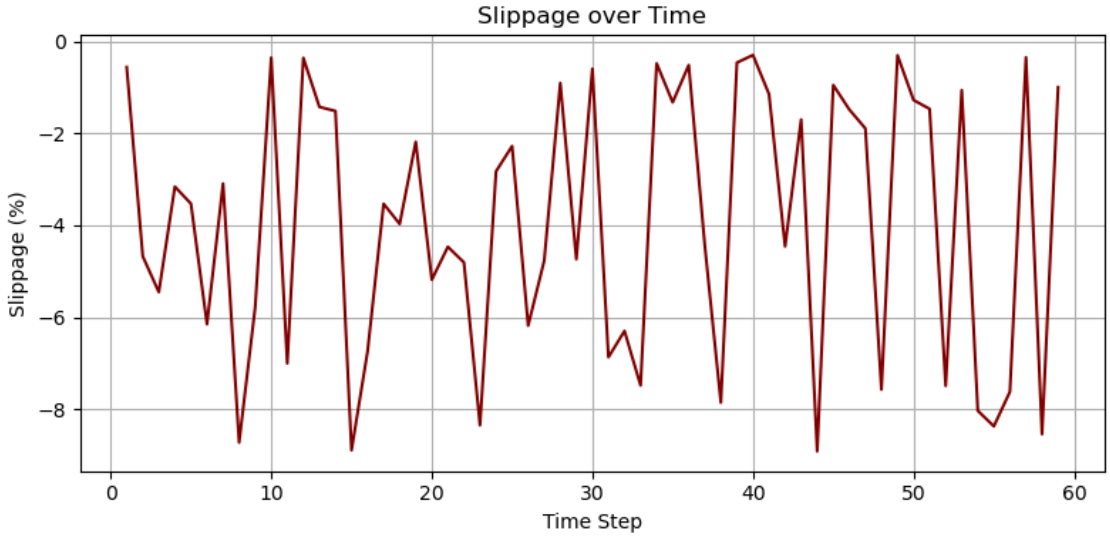

Figure 5: Slippage (in %) over time

Slippage is defined for a particular swap, so depending on the trade lot fraction in that swap, the value of slippage squiggles over time.

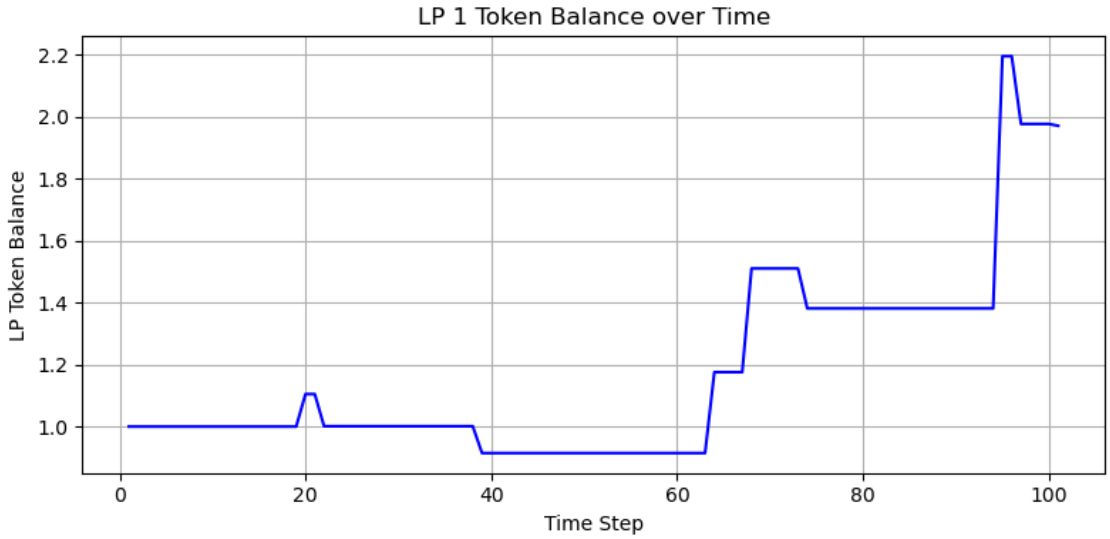

Figure 6: LP token balance of LP 1 (assumed owner)

LP Tokens balance for the DEX owner over time which changes when he adds / removes liquidity. Observe that the owner's share of LPTokens is higher than other LP providers since he is given some LP Tokens for free (5e12) at the start.

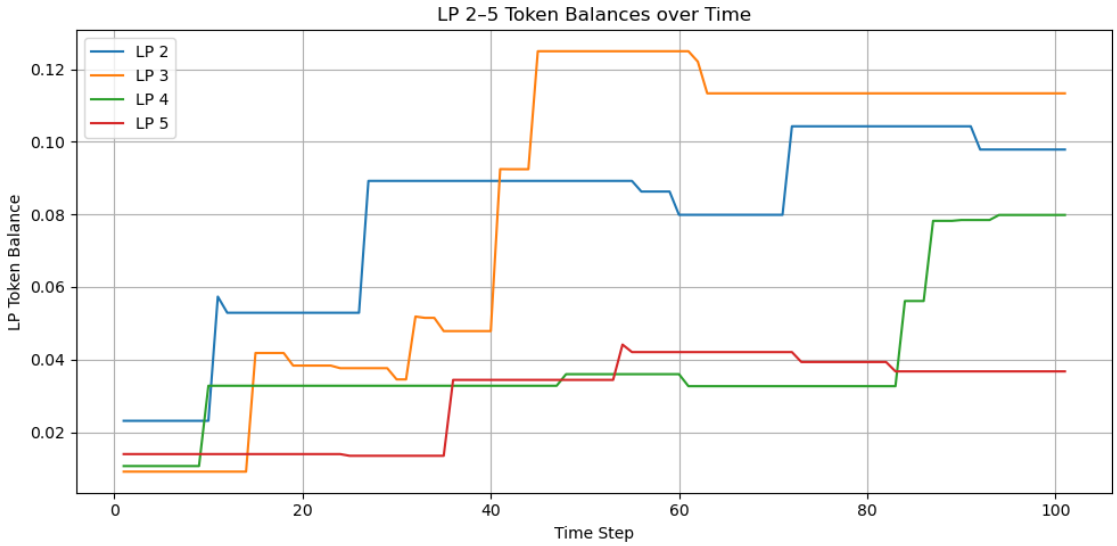

Figure 7: LP token balances of LPs 2–5

LP Tokens balance for the other liquidity providers over time which changes as and when they add / remove liquidity.

Sanity checks implemented

- First of all we always check if the requested amount of tokenA or tokenB or LPToken provided by the user should not exceed his balance in any function.

```
require(tokenA.balanceOf(msg.sender) >= amountA, "Insufficient token A balance");
require(tokenB.balanceOf(msg.sender) >= amountB, "Insufficient token B balance");
require(lpToken.balanceOf(msg.sender) >= lpAmount, "Insufficient LP balance");
```

- We also check that non-zero amounts are provided.

```
require(lpAmount > 0, "Zero amount in remove liquidity");
require(amountA > 0 && amountB > 0, "Zero amounts in add liquidity");
require(amt > 0, "Enter non-zero amount in swap_A_to_B");
```

- We also assume that all amounts are written in terms of mini-tokens (1 token = 10^{18} mini-tokens) so as to handle decimal arithmetic.

```
const tokenA = await tokenAFactory.deploy({
    data: tokenMetadata.data.bytecode.object,
    arguments: [(BigInt(400000) * SCALE).toString()]
}).send({ from: owner, gas: 50000000 });
```

We mint in terms of mini-tokens only and then elsewhere in our script we are taking percentages of the balances, so automatically scaled amounts are passed.

- We are returning the scaled reserve ratio (multiplying it by 10^{18}) to handle floating point values.

```
return (reserveB * amt_A)/reserveA;
return (reserveA * amt_B)/reserveB;
```

- We are checking that only the DEX can mint or burn LPTokens.

```
require(msg.sender == dexadd, "ONLY DEX CAN MINT LP TOKENS");
require(msg.sender == dexadd, "ONLY DEX CAN BURN LP TOKENS");
```

- We are also checking that the transferFrom functions work (maybe the user forgot to approve the transaction)

```
require(tokenA.transferFrom(msg.sender,address(this),amountA),
"Approve the liquidity of A");
require(tokenB.transferFrom(msg.sender, address(this),amountB),
"Approve the liquidity of B");
```

- We are also checking that only the owner can call the perform_arbitrage function.

```
require(msg.sender == owner, "Only owner can call");
```

- We are allowing some error in deposit amount ratio and reserve ratio in addLiquidity function (to account for precision errors).

```
require(
    a > b ? (a-b <= 100) : (b-a <= 100),
    // Allowing 16th decimal place to be different
    "Invalid deposit ratio"
);
```

Theory Questions and Answers

1. Which address(es) should be allowed to mint/burn the LP tokens?

Only the DEX contract address should be allowed to mint and burn LP tokens. This ensures that LP tokens are issued only when liquidity is added or removed through valid contract functions, preventing unauthorized manipulation.
In the LPToken.sol file, we have written these which ensures only DEX can burn or mint the LPTokens.

```
require(msg.sender == dexadd, "ONLY DEX CAN MINT LP TOKENS");
require(msg.sender == dexadd, "ONLY DEX CAN BURN LP TOKENS");
```

2. How do DEXs level the playing field between powerful and lower-resource traders?

In traditional stock markets (CEXs), High-Frequency Traders (HFTs) gain a massive edge by being able to execute trades themselves using ultra-low-latency infrastructure like:

- Co-location near exchange servers
- Direct access to order books
- Proprietary high-speed networks

Speed is Power in CEXs because the fastest trader wins. On the other hand, in DEXs (like Uniswap):

- Traders cannot directly execute trades.
- All trades must go through miners/validators who include them in blocks.
- Speed advantage shifts from the trader to the block producer.

This causes

- HFT-style latency advantages to disappear.
- Everyone – retail or institutional – to play by the same rule: *"Fastest block wins, not fastest trader."*
- The playing field to shift from speed to transparency, optimal strategy, and willingness to pay gas fees.

Hence, DEXs remove the unfair *execution-speed advantage* that HFTs enjoy in traditional markets by outsourcing trade execution to miners, ensuring all users are bound by blockchain confirmation times.

3. How can miners exploit the mempool, and how to make DEXs robust?

Problem of Exploitation: In a DEX, all pending transactions first enter the miner's *mempool* before getting included in a block. Miners (or validators) can observe these transactions and exploit them using strategies like:

- **Front-loading:** Seeing a large buy order and placing their own buy order just before it to profit from the price increase.
- **Back-loading:** Placing a trade immediately after a large transaction to benefit from price reversions.
- **Sandwich Attack:** Placing a buy order before and a sell order after a victim's trade to capture profit from price movement.

Making DEXs Robust Against Mempool Exploits

- **Commit-Reveal Schemes:** Traders first commit a hash of their trade, and reveal details in a later transaction, preventing immediate front-loading.
- **Batch Auctions:** Collecting trades over a time window and executing them together at a uniform clearing price, eliminating ordering advantages. Basically, users trading on same pair of tokens are matched peer-to-peer in a "**Coincidence of Wants**" trade. This provides better prices and bypasses MEV, as orders don't have to tap on-chain liquidity.
- **MEV-resistant DEX Designs:** Mechanisms like TWAMM (Time-Weighted AMM) or frequent batch auctions reduce maximal extractable value (MEV). The TWAMM breaks the long-term orders submitted by trader into infinitely small virtual sub-orders, which trade against the embedded AMM(constant product AMM) at an even rate over time.

4. How do gas fees affect the DEX and arbitrage viability?

- **DEX Viability:** High gas costs can deter small trades. When transaction fees exceed the potential gains, especially for low-value trades, retail investors are less likely to participate—reducing overall liquidity and adoption.
- **Arbitrage:** Arbitrage relies on price differences between exchanges. However, if the gas fees required to execute a trade exceed the profit from the price difference, the opportunity becomes unprofitable and is left unexploited.

5. Can gas fees cause unfair advantages? How?

Yes, higher gas fees can prioritize transactions in **congested networks**, giving an **advantage to users willing to pay more** (e.g., institutional traders). Retail investors with limited budgets may experience delays or failed transactions during high network activity. Layer 2 scaling solutions and batching transactions can help reduce this disparity.

6. How to minimize slippage in swaps?

Slippage occurs when the price of a token changes between the time a trade is submitted and when it is confirmed on the blockchain. It typically results in receiving fewer tokens than expected, especially during periods of high volatility or low liquidity.

Minimizing Slippage

To reduce the chances and impact of slippage:

- **Trade in high-liquidity pools:** Pools with large reserves are less affected by individual trades, reducing price impact.
- **Break large trades into smaller batches:** This helps avoid major shifts in reserve balances that can affect pricing.

- **Use limit orders instead of market orders:** Market orders are meant to be executed as quickly as possible at the current market price. Limit orders set the maximum or minimum price you'll pay or receive when buying or selling stock.
- **Set slippage tolerance:** Most DEX platforms allow users to set a maximum slippage tolerance, automatically cancelling trades if the price moves beyond the acceptable range.
- **Trade during stable market conditions:** Lower volatility generally means more predictable prices and less slippage risk.

7. Plot of slippage vs trade lot fraction

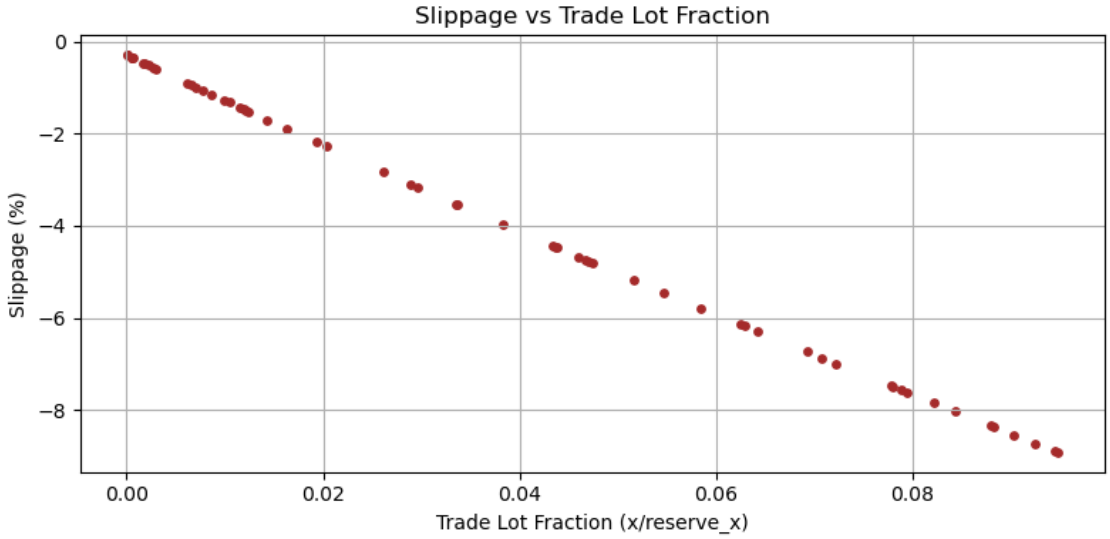

Figure 8: Slippage vs trade lot fraction

Slippage S is defined as:

$$S = \left(\frac{\frac{\text{token Y received in Swap}}{\text{token X deposited in Swap}} - \frac{\text{token Y reserves before Swap}}{\text{token X reserves before Swap}}}{\frac{\text{token Y reserves before Swap}}{\text{token X reserves before Swap}}} \right) \times 100\%$$

By the constant product formula, suppose Δx token X is deposited and Δy token Y is received, then we have

$$(x + \Delta x)(y - \Delta y) = xy \tag{1}$$

$$\frac{\Delta y}{\Delta x} = \frac{y}{x + \Delta x} \tag{2}$$

Substituting the expressions:

$$S = \left(\frac{\frac{y}{x+\Delta x}}{\frac{y}{x}} - 1 \right) \times 100\% = \left(\frac{x}{x + \Delta x} - 1 \right) \times 100\% = \left(\frac{-\Delta x}{x + \Delta x} \right) \times 100\%$$

Let f denote the **trade lot fraction**, i.e., the ratio of tokens being traded to the pool's reserves:

$$f = \frac{\Delta x}{x}$$

Then,

$$x + \Delta x = x(1 + f) \quad \Rightarrow \quad \frac{\Delta x}{x + \Delta x} = \frac{f}{1 + f}$$

Thus, the slippage becomes:

$$\boxed{S(f) = \frac{-f}{1 + f} \cdot 100\%}$$

Interpretation

When we find the slope of this curve, we get:

$$\frac{dS}{df} = \frac{-100}{(1 + f)^2} \tag{3}$$

This explains the nonlinear, downward-sloping slippage curve observed in AMMs (plotted above). We see that as f increases, the magnitude of the slope decreases(in ideal cases it kind of looks like a rectangular hyperbola). We are only seeing the upper portion of it because we capped the swap amounts at 10% of reserves, if we hadn't then we would observe the complete graph.

Arbitrage profit threshold

To account for the gas fees(although gas fees were not considered in this assignment, but we tried to estimate according to real life), slippage and the adjustment factor(depending on how conservative we are), we decided the threshold at 3%.

References

- https://www.ulam.io/blog/blockchain-front-running-risks-and-protective-measures#
- https://cow.fi/learn/understanding-batch-auctions
- https://www.paradigm.xyz/2021/07/twamm
- https://medium.com/coinmonks/commit-reveal-scheme-in-solidity-c06eba4091bb