I am currently developing an automated x*y=k

market maker for ETH and ERC20 tokens. For those unfamiliar with x*y=k

market makers, this post explains the basic idea:

Improving front running resistance of x*y=k market makers[

Decentralized exchanges

](/c/decentralized-exchanges)

Two years ago I made a post where I suggested that we could run on-chain decentralized exchanges in a similar way to what Augur and Gnosis were proposing for on-chain market makers (eg. LMSR), and Martin Köppelmann from Gnosis suggested a simple approach for doing so, that I call the "x*y=k market maker". The idea is that you have a contract that holds x coins of token A and y coins of token B, and always maintains the invariant that x*y=k for some constant k. Anyone can buy or sell coins by ess…

One major issues that exist for this type of exchange is price slippage. Basically, the more tokens you buy, the higher the rate per token. This is not a big problem for small trades, but becomes problematic for users trying to make single trades that are large relative to the total liquidity of the market. Recently, I have been playing with an on-chain order book of limit orders as a solution to this problem. Consider the two scenarios:

Scenario 1

- Starting state (tokenA, tokenB): (10, 10)
- Buyer sends 1 tokenA, gets back 0.909091 tokenB
- Updated state: (11, 9.090909)
- Buyer receives tokens at a rate of 0.909091 tokenB/tokenA

Scenario 2

- Starting state (tokenA, tokenB): (10, 10)
- Buyer sends 5 tokenA, gets back 3.33333 tokenB
- Updated state: (15, 6.66667)
- Buyer receives tokens at a rate of 0.666667 tokenB/tokenA

In scenario 2, the buyer is attempting to purchase 33% of the tokenB liquidity and is getting a significantly worse rate than scenario 1, where the buyer attempted to purchase just under 10% of tokenB liquidity. It can be seen that this type of exchange is only good for smaller purchases, and cannot compete with order books for high value trades. Lets modify scenario 2 with an on-chain limit order.

Scenario 2 - with limit order

- Starting state (tokenA, tokenB): (10, 10)
- Buyer1 posts a limit order saying they are selling up to 20 tokenB for tokenA at a rate of 1 tokenB/tokenA. This is above the current market rate for tokenB, and cannot be executed.
- If the price is shifted to the point where this trade can be made, anyone can execute this order on their behalf. If the executor gets a better deal than 1 tokenB/tokenA they can keep it as a bounty.
- Within a single smart contract call
- Buyer2 sends 5 tokenA, gets back 3.33333 tokenB
- Updated state: (15, 6.66667)
- Buyer2 sells 3 tokenB of Buyer1's limit order for 4.65517 tokenA
- Updated state: (10.34482, 9.66667)
- Buyer2 forwards 3 tokenA to Buyer1, at a rate of 1 tokenB/tokenA
- Buyer2 claims the remaining 1.65517 tokenA as a bounty
- Buyer2 sends 5 tokenA, gets back 3.33333 tokenB

- Updated state: (15, 6.66667)
- Buyer2 sells 3 tokenB of Buyer1's limit order for 4.65517 tokenA
- Updated state: (10.34482, 9.66667)
- Buyer2 forwards 3 tokenA to Buyer1, at a rate of 1 tokenB/tokenA
- Buyer2 claims the remaining 1.65517 tokenA as a bounty
- Buyer2 has now received 3.33333 tokenB for 5 - 1.65517 = 3.34483 tokenA
- Buyer receives tokens at a rate of 0.99656 tokenB/tokenA

This shows that using on-chain limit orders allows for the creation of a single exchange that combines the simple price determination mechanism of an x*y=k market maker with the high volume trades of an order book. Discrepancies between a desired token purchase and limit orders available are settled automatically using liquidity stored in the market maker. This feature should reduce price slippage, and by extension could reduce the profitability of front running.

Potential problems: Limit orders cannot be pulled instantly like on off chain exchanges. If the market is rapidly shifting, miners will likely deny transactions pulling limit orders, and instead fill the order themselves for a profit. I am very interested in feedback on this mechanism, and the idea of combining automated market makers with order books. I have not begun implementing this feature, but am planning to in the near future.

Please reach out if you are interested in contributing or want to learn more. My implementation is open source and can be found here:

GitHub

Uniswap/contracts-vyper

An interface for seamless ERC20 exchange. Contribute to Uniswap/contracts-vyper development by creating an account on GitHub.