Hey guys, I had some ideas on reducing AMM slippage by incorporating limit orders. Note that@haydenadams proposed similar ideas in his earlier post. But here let's describe the mechanism slightly more formally.

The basic idea is straightforward. The DEX consists of a Uniswap-like liquidity pool, and an order-book which records the limit

buy/sell orders. Whenever there is a token swap request, the limit orders will be taken first. Then, tokens in the liquidity pool get consumed, which moves the price, until it hits the next limit order. This process repeats until the swap is done. To describe the swap mechanism more formally, let us look at a trading pair A/B:

Definitions

Before we dive into the token swap algorithm, we define the following terminologies/symbols:

x_o

: the total number of A tokens in the limit-buy order book

x_l

: the total number of A tokens in the liquidity pool

y_o

: the total number of B tokens in the limit-sell order book

y_l

: the total number of B tokens in the liquidity pool

The state of the DEX can thus be described with this tuple: (x_o, y_o, x_l, y_l)

. In addition, we define the following quantities:

p = x_l / y_l

: the "price" of token B in terms of A, i.e. the exchange rate between B and A.

x_o(p)

: the total number of A tokens in the buy orders at price point p

. Note that x_o(p)

can be viewed as a function of p

.

y_o(p)

: the total number of B tokens in the sells orders at price point p

. Similarly, note that y_o(p)

can also be viewed as a function of p

.

Token Swap Mechanism

Let us look at the scenario where a user wants to swap \Delta x

number of A tokens into B tokens. Suppose that before the swap, the DEX state is (x_o, y_o, x_l, y_l)

. After the swap, the DEX state becomes (x^{\prime}_o , y^{\prime}_o, x^{\prime}_l, y^{\prime}_l)

. Given \Delta x

, the swap()

function of the DEX needs to properly determine the following four differences:

\Delta x_o = x^{\prime}_o - x_o

, $\Delta x_l = x^{\prime}_l - x_l$

, $\Delta y_o = y^{\prime}_o - y_o$

, and $\Delta y_l = y^{\prime}_l - y_l$

With these values, the amount of B tokens returned to the user can thus be calculated by $\Delta y = -(\Delta y_o + \Delta y_l)$

.

Roughly speaking, the swap()

function works like this: The limit sell orders at the current price point $p = x_l / y_l$

will be taken first. If there are insufficient limit sell orders (i.e. $p \cdot y_o(p) < \Delta x$

) at price point p

, tokens from the liquidity pool will be taken. This in turn results in price increase and might trigger the next limit sell orders. This process continues until all Δx

A tokens are swapped.

Now let us see how to mathematically determine the four differences Δx_o

, Δx_l

, Δy_o

, and Δy_l

given the amount of input token Δx

.

First, since the user is swapping A token for B token, none of the x_o

buy orders are consumed. Thus, $\Delta x_o = 0$

.

To determine the remaining three quantities, we note that the liquidity pool needs to maintain the following invariant:

$(x_l + \Delta x_l) \cdot (y_l + \Delta y_l) = x_l \cdot y_l = k$

Thus, before the swap, the starting price point can be written as $p_s = p = \frac{x_l}{y_l} = \frac{k}{y^2_l}$

. After the swap is done, the ending price point would be $p_e = \frac{x_l + \Delta x_l}{y_l + \Delta y_l} = \frac{k}{(y_l + \Delta y_l)^2}$

. Based on the swap()

algorithm described above, we can write down the following equation:

$\Delta x = \int^{p_e}_{p_s} p \cdot y_o(p) dp + \Delta x_l$

, where $\Delta x_l \geq 0$

.

The equation above should be pretty intuitive, since the $y_o(p)$

B tokens that are available as limit-sell orders at price point p

can be swapped with $p \cdot y_o(p)$

A tokens. Thus, the integral gives the total number of A tokens needed for swapping with the limit-sell orders of B between p_s

and p_e

. One small caveat is that the right-hand-side of the above equation may “overshoot” since the limit-sell orders at price point p_e

has more more tokens than enough for the swap. Just consider the simple case where the current price is p = 1

, and there is a limit-sell order of 100 B tokens, while \Delta x = 20

A tokens. Obviously, the swap will only consumes 20 out of the 100 B tokens. To handle such scenarios, we introduce a variable r

to represent the “excess” of the limit-sell orders at price point p_e

, we can instead rewrite the equations as a minimization problem:

Minimize \Delta x_l

, subject to

(x_l + \Delta x_l) \cdot (y_l + \Delta y_l) = k

\Delta x = \int^{p_e}_{p_s} p \cdot y_o(p) dp - p_e \cdot r + \Delta x_l

, where p_s = \frac{k}{y^2_l}

, p_e = \frac{k}{(y_l + \Delta y_l)^2}

0 \leq r \leq y_o(p_e)

\Delta x_l \geq 0

Notice that the upper bound of p_e

can be derived by p_e = \frac{k}{(y_l + \Delta y_l)^2} = \frac{(x_l + \Delta x_l)^2}{k} \leq \frac{(x_l + \Delta x)^2}{k}

, since \Delta x_l \leq \Delta x

. As a result, \Delta y_l = \sqrt{k/p_e} - y_l

is also lower-bounded by \frac{k}{x_l + \Delta x} - y_l

. On the other hand, we know that \Delta y_l \leq 0

since token B in the liquidity pool can only be consumed for this swap. Thus, the above minimization problem can be solved efficiently by performing a binary search of \Delta y_l

between [\frac{k}{x_l + \Delta x} - y_l, 0]

.

Solving the minimization problem gets up \Delta x_l

and \Delta y_l

. As a byproduct, we can also calculate \Delta y_o = -\int^{p_e}_{p_s} y_o(p) dp

, just need to keep in mind that the limit-sell orders at price point p_e

might not be fully taken.

Finally, the swap()

function can return the user \Delta y = -(\Delta y_l + \Delta y_o)

amount of token B.

Please let me know if you guys have any thoughts/comments. Any feedback is appreciated!