Dglobal Notes (2024), , 1–8

ARTICLE

Constant product AMM (Uniswap V1) easily explained.

Tiz

1. Constant Product Market Maker

The constant $k = x * y$ is used in constant product market makers to determine prices in an automated and trustless manner, anybody can deposit their own funds into a liquidity pool sometimes referred to as exchange in order for other users to trade or swap between them and the liquidity providers are able to accrue 0.3% in fees for every swap, without fees the constant product would remain constant for the lifetime of the exchange, but the swapping fee ensures that the pool grows with every trade.

To establish some baseline requirements, UniSwap V1 allows any given pool to be created with native ETH and an ERC-20 compliant token which will be referred to as T token.

1.1 No fees

For simplicity removing fees from the protocol (this is not economically feasible as the opportunity cost of the funds would prevent users from depositing and maintaining a large pool balance but it can be done as a starting point to grasp how the system works.)

Constant product is defined as follows:

$$k = x * y \tag{1}$$

Where k is the constant product the pool maintains, x is the amount of ETH in the pool referred to as ETH reserves and y is the amount of token T in the pool, referred to as tokens reserves, *UNI* is the token issued to liquidity providers of a given pool.
A user creates a pool with $x = 1\ ETH$ and $y = 100\ T$, believing this to be the current fair market value (assuming this is correct for simplicity).

Note: this ratio is not checked or enforced at the protocol level at the creation of the pool meaning that if he were to deposit $x = 2$ ETH instead this would imply that this pool is selling every ETH for $\frac{100}{2} = 50\ T$ meaning that anyone scanning the chain would be have the incentive to buy as much as possible in order to achieve fair market price of $1\ ETH$ for $100\ T$, where the initial liquidity provider would suffer the impermanent loss, same with the inverse case where a pool were to be created with $1\ ETH$ and $200\ T$ where the pool would effectively be selling $1\ ETH$ for $200\ T$ or double the current value and users would have the incentive to sell as much ETH as possible until reaching fair valuation.

Now we have established the pool with $1\ ETH$ and $100\ T$ while receiving himself 100 UNI (liquidity tokens) that anyone can swap with, note that slippage will be naturally introduced and it will be higher than usual due to the small size of the pool.

There are now 6 actions any user can take around this pool which are:

1. Adding Liquidity
2. Removing Liquidity
3. Selling ETH (from input)
4. Selling ETH (from output)
5. Buying ETH (from input)
6. Buying ETH (from output)

Going through the calculations for each of these actions to get a complete picture of the inner workings of the protocol.

1.1.1 Adding Liquidity

When adding liquidity, unlike the the user who creates the liquidity pool the user must obey the current ratio of the pool, adding T and ETH following the ratio of total T to ETH in the pool as follows:

$$\frac{T_{add}}{ETH_{add}} = \frac{T_{tot}}{ETH_{tot}} \tag{2}$$

The UniSwap V1 contract calculates the amount of tokens needed in order to keep the correct pool ratio by solving for T_{add} like so:

$$T_{add} = ETH_{add} * \frac{T_{tot}}{ETH_{tot}} \tag{3}$$

Ensuring that the amount of tokens sent is AT LEAST the amount of tokens required to keep the correct ratio of ETH to T. For a concrete example, another user wants to add a further 0.1 ETH into the existing pool, thus in this case $T_{tot} = 100$, $ETH_{tot} = 1$, $ETH_{add} = 0.1$, plugging these values into the equation:

$$T_{add} = 0.1ETH * \frac{100T}{1ETH} = 10T \tag{4}$$

That means that the user must add a further 10 T along with 0.1 ETH into the pool, the amount of UNI tokens they receive is based on the ratio of ETH deposited and the total liquidity present in the pool:

$$\frac{UNI_{mint}}{UNI_{tot}} = \frac{ETH_{mint}}{ETH_{tot}} \tag{5}$$

solving for UNI_{mint}

$$UNI_{mint} = UNI_{tot} * \frac{ETH_{mint}}{ETH_{tot}} \tag{6}$$

Using the example:

$$UNI_{mint} = 100 * \frac{0.1}{1} = 10 \tag{7}$$

Meaning the user would now hold 110 UNI tokens representing $\frac{10}{110} * 100\% \approx 9.1\%$ of the liquidity tokens for that pool, which also means that every time a user swaps on the pool they are entitled to that percentage of the protocol's 0.3% swap fee.

1.1.2 Removing Liquidity

Let's say that now the user wants to remove that same liquidity from the pool, now the current ratio of tokens to liquidity tokens to total liquidity tokens must be respected, calculating the amount of T and ETH separately with the ratio of liquidity to burn and total liquidity, for the token T:

$$\frac{T_{rem}}{ETH_{tot}} = \frac{UNI_{rem}}{UNI_{tot}} \tag{8}$$

$$T_{rem} = ETH_{tot} * \frac{UNI_{rem}}{UNI_{tot}} \tag{9}$$

and for ETH:

$$\frac{ETH_{rem}}{T_{tot}} = \frac{UNI_{rem}}{UNI_{tot}} \tag{10}$$

$$ETH_{rem} = T_{tot} * \frac{UNI_{rem}}{UNI_{tot}} \tag{11}$$

Meaning that for the amount of liquidity UNI_{rem} T_{rem} amount of token T needs to be removed and ETH_{rem} amount of eth needs to be removed. Using the simple example the user would be looking to burn 10 UNI tokens, there are 110 UNI tokens in the pool and 1.1 total ETH, plugging in these values:

$$ETH_{rem} = 1.1 * \frac{10}{110} = 0.1ETH \tag{12}$$

$$T_{rem} = 110 * \frac{10}{110} = 10T \tag{13}$$

Getting back his original position of 1 ETH and 100 T.
As was mentioned beforehand, a constant product AMM works by maintaining the product of the reserves a constant when buying or selling an asset off a pool as long as fees are not taken into account, in our example case the constant product may be computed as:

$$k = x\gamma = (1)(100) = 100 \tag{14}$$

Meaning that the product to maintain is 100.

Establishing x as the ETH reserve, γ as the T reserve and x_{in}, x_{out}, γ_{in}, γ_{out} as ETH or T input or output respectively.

1.1.3 Selling ETH (from input)

When selling ETH we can calculate the expected output from the input of ETH given by the form:

$$(x + x_{in})(\gamma - \gamma_{out}) = x\gamma \tag{15}$$

Where our unknown variable is γ_{out} since the protocol knows the reserves of each token in the contract and the amount of ETH sent by the seller, solving for γ_{out}:

$$\gamma_{out} = \frac{-x\gamma}{(x + x_{in})} + \gamma = \frac{-x\gamma + x\gamma + x_{in}\gamma}{(x + x_{in})} = \frac{x_{in}\gamma}{(x + x_{in})} \tag{16}$$

For a quick concrete example if a user wanted to sell 0.1 ETH the amount of tokens he would receive can be computed as follows (overline means recurring decimal):

$$\gamma_{out} = \frac{(0.1)(100)}{(1 + 0.1)} = 9.09090909 = 9.\overline{09}T \tag{17}$$

NOTE: the output is 10% lower than the expected 10 T tokens implied by the ratio of the pool. This is known as slippage and it's the mechanism by which the pool can maintain it's constant product even through the reserves are changing with every swap, by adjusting the reserves and giving more or less output depending on how much is bought or sold relative to the size of the pool, this is in part why tokens with low liquidity tend to be more volatile, in our case such a small pool is extremely prone to slippage, a bigger pool would mitigate this (try with different pool reserve numbers).

1.1.4 Selling ETH (from output)

If the user indicates that they want to get a certain amount of T tokens for the respective amount of ETH, the input required can also be calculated from output, starting from the same form:

$$(x + x_{in})(y - y_{out}) = xy \tag{18}$$

Now solving for x_{in}

$$x_{in} = \frac{xy}{(y - y_{out})} - x = \frac{xy - xy + xy_{out}}{(y - y_{out})} = \frac{xy_{out}}{(y - y_{out})} \tag{19}$$

For another simple example now the user wants to see how many ETH tokens they would need to sell in order to get 10 T tokens, condition which would need to be met for the swap to go through.

$$x_{in} = \frac{(1)(10)}{(100 - 10)} = 0.11111111 = 0.\overline{1}ETH \tag{20}$$

Thus they would need to send $0.\overline{1}$ ETH in order to get 10 T tokens

1.1.5 Buying ETH (from input)

The inverse case can be calculated from the inverse form where x_{out} ETH is output and y_{in} T is input to the protocol:

$$(x - x_{out})(y + y_{in}) = xy \tag{21}$$

Solving for x_{out}

$$x_{out} = \frac{-xy}{(y + y_{in})} + x = \frac{-xy + xy + xy_{in}}{(y + y_{in})} = \frac{xy_{in}}{(y + y_{in})} \tag{22}$$

Same concrete example as before:

$$x_{out} = \frac{(100)(0.1)}{1 + 0.1} = 9.\overline{09} \tag{23}$$

1.1.6 Buying ETH (from output)

From the form:

$$(x - x_{out})(y + y_{in}) = xy \tag{24}$$

Solving for y_{in}

$$y_{in} = \frac{xy}{(x - x_{out})} - y = \frac{xy - xy + yx_{out}}{(x - x_{out})} = \frac{yx_{out}}{(x - x_{out})} \tag{25}$$

Following the same example:

$$y_{in} = \frac{100(0.1)}{1 - 0.1} = 11.\overline{1} \tag{26}$$

1.2 With fees

All previous calculations were done under the assumption that the protocol took no fees for themselves or the liquidity providers, it is unreasonable to assume that such a protocol could work as there are no incentives for participants to provide liquidity, subjecting themselves to impermanent loss or simply losing opportunity cost on their funds by locking them up out of the goodness of their hearts, which is why UniSwap V1 implemented a 0.3% fee out of every trade that liquidity providers are able to claim in accordance to the percentage of the *UNI* tokens they hold.
Since there are no fees on adding or removing liquidity, fees only affect users doing a swap, both by selling ETH for token T or buying ETH with token T, the fee is added to the liquidity pool in order to grow the positions of liquidity providers.
Starting from the same reasoning for the constant product market maker

$$xy = k \tag{27}$$

When getting the price from the input, the pool must take out the fees from the user's input funds

$$(x + x_{in})(y - y_{out}) = k \tag{28}$$

Which in principle is similar to what the protocol with no fees presents only for a few differences, the main one being that now the "constant" product does not remain constant forever, since after every single swap the pool is taking a small difference, either taking more from the input or giving less on the output, which is the fee that users need to pay in order to maintain the protocol profitable.

1.2.1 Selling ETH from input

With the addition of fees the protocol must receive more input than the protocol with no fees, since Solidity only supports integer operations in order to minimize loss of precision in the EVM, it is not possible to simply multiply any number by 0.003 in order to get 0.3% of that number, we must instead use ONLY integers, floating points or rational number are not allowed, thus we must multiply by $\frac{3}{1000}$ in order to get the required percentage of a number, going through this operation.

$$(x + x_{in})(y - y_{out}) = k \tag{29}$$

Adding 0.3% to the ETH input in x_{in}

$$(x + x_{in} - x_{in}\frac{3}{1000})(y - y_{out}) = (x + x_{in}(1 - \frac{3}{1000}))(y - y_{out}) = (x + x_{in}(\frac{997}{1000}))(y - y_{out}) = k \tag{30}$$

$$(x + x_{in}(\frac{997}{1000}))(y - y_{out}) = (\frac{1000x + 997x_{in}}{1000})(y - y_{out}) = xy \tag{31}$$

Solving for y_{out}

$$y_{out} = \frac{-xy}{(\frac{1000x+997x_{in}}{1000})} + y = \frac{-1000xy}{(1000x + 997x_{in})} + y = \frac{-1000xy + 1000xy + 997x_{in}y}{1000x + 997x_{in}} \tag{32}$$

Which simplifies to the form:

$$y_{out} = \frac{997x_{in}y}{1000x + 997x_{in}} \tag{33}$$

Calculating a simple transaction where a users wants to sell 0.1 ETH

$$y_{out} = \frac{997(0.1)(100)}{1000(1) + 997(0.1)} = 9.06611 \tag{34}$$

1.2.2 Selling ETH from output

In the case of selling ETH and from output of T tokens the pool needs to do the inverse calculation, now from an output of y_{out} tokens the protocol needs to give an input amount that is 0.3% greater, give in the following form:

$$(x + (x_{in} - x_{in}\frac{3}{1000}))(y - y_{out}) = (x + x_{in}\frac{997}{1000})(y - y_{out}) = xy \tag{35}$$

Solving for x_{in}

$$x_{in} = \frac{1000}{997}\frac{xy}{(y - y_{out})} - x = \frac{1000}{997}\frac{xy - xy + y_{out}x}{(y - y_{out})} = \frac{1000}{997}\frac{y_{out}x}{(y - y_{out})} \tag{36}$$

Where once again 0.3% is taken from the input in order to be added to the liquidity pool and reward liquidity providers.
Doing a quick calculation of a user looking to buy 10 T tokens.

$$x_{in} = \frac{1000}{997}\frac{y_{out}x}{(y - y_{out})} = \frac{1000}{997}\frac{10(1)}{(100 - 10)} = 0.111445 \tag{37}$$

1.2.3 Buying Eth from input

Using the familiar form:

$$(x - x_{out})(y + y_{in}) = xy \tag{38}$$

Now with fees the protocol needs to subtract 0.3% from for the LP:

$$(x - x_{out})(y + y_{in}(\frac{997}{1000})) = xy \tag{39}$$

Solving for x_{out}

$$x_{out} = \frac{-1000xy}{(1000y + 997y_{in})} + x = \frac{-1000xy + 1000xy + 997y_{in}x}{(1000y + 997y_{in})} = \frac{997y_{in}x}{(1000y + 997y_{in})} \tag{40}$$

Calculating a user looking to buy ETH with 10 T tokens

$$x_{out} = \frac{(997)(10)(1)}{(1000(100) + 997(10))} = \frac{997}{10997} \approx 0.09066 \tag{41}$$

1.2.4 Buying Eth from output

Finally to buy ETH calculating from output:

$$(x - x_{out})(y + y_{in}) = xy \tag{42}$$

Same as before subtracting fees:

$$(x - x_{out})(y + y_{in}(\frac{997}{1000})) = xy \tag{43}$$

Solving for y_{in}

$$y_{in}(\frac{997}{1000}) = \frac{xy}{(x - x_{out})} - y \tag{44}$$

$$y_{in} = \left(\frac{1000}{997}\right)\frac{xy - xy + yx_{out}}{(x - x_{out})} = \left(\frac{1000}{997}\right)\frac{yx_{out}}{(x - x_{out})} \tag{45}$$

Now calculating for a user looking to get 0.1 ETH for their T tokens

$$y_{in} = \left(\frac{1000}{997}\right)\frac{(100)(0.1)}{(1 - 0.1)} \approx 11.1445 \tag{46}$$

Comparing these values the differences can be appreciated in both outputs and inputs, those differences is what makes the protocol grow and why it makes sense to add liquidity into UniSwap.

2. References:

1.- UniSwap, The Uniswap V1 Smart Contracts, https://docs.uniswap.org/contracts/v1/guides/connect-to-uniswap, visited 25th Febuary 2025.

2.- Adams Hayden, Uniswap Whitepaper, https://hackmd.io/@HaydenAdams/HJ9jLsfTz?type=view, visited 25th Febuary 2025.