LIQUIDITY MATH IN UNISWAP V3

TECHNICAL NOTE

Atis Elsts

August 13, 2021

ABSTRACT

Uniswap is the largest decentralized exchange (DEX) and one of cornerstones of Decentralized Finance (DeFi). Uniswap uses liquidity pools to provide Automated Market Making (AMM) functionality. Uniswap v3 is the most recent version of the protocol that introduces a number of new features, notably the *concentrated liquidity* feature, which allows the liquidity providers to concentrate their liquidity in a specific price range, leading to an increased capital efficiency. However, the mathematical relationship between the liquidity of a position, the amount of assets in that position, and its price range becomes somewhat complex. This technical note shows how derive some of the results from the Uniswap v3 whitepaper, as well as presents several other equations not discussed in the whitepaper, and shows how to apply these equations.

1 Introduction

The core concepts of liquidity math are defined in the Uniswap v3 whitepaper [1]. However, the paper is quite terse and many aspects are not fully worked out. There is a gap between the information that the whitepaper provides, and the questions Uniswap users and developers have, such as:

- Given the total liquidity and price range of a position, how much of asset X and asset Y does this position have at a specific price P?
- Given a price range, current price P, and the amount x of asset X, how much of asset Y should one supply to cover that price range?
- Given x amount of asset X and y amount of asset Y that are to be deposited in a liquidity pool, as well as the current price P and lower bound of the expected price range, what value should be used as the upper bound?

This technical note aims to bridge this gap and to provide basic intuition about the Uniswap v3 liquidity math. For the accompanying example code, see: `https://github.com/atiselsts/uniswap-v3-liquidity-math/blob/master/uniswap-v3-liquidity-math.py`. Note that the code is meant to illustrate the math: it's not directly suitable for use in real projects!

Table 1: Symbols used

Symbol name	Whitepaper	Uniswap code	Notes
Price	P	`sqrtRatioX96`	Code tracks $\sqrt{P}$ for efficiency reasons
Lower bound of a price range	p_a	`sqrtRatioAX96`	Code tracks $\sqrt{p_a}$
Upper bound of a price range	p_b	`sqrtRatioBX96`	Code tracks $\sqrt{p_b}$
The first asset	X	`token0`	
The second asset	Y	`token1`	
Amount of the first asset	x	`amount0`	
Amount of the second asset	y	`amount1`	
Virtual liquidity	L	`liquidity`	

2 Calculations

2.1 Calculating liquidity and the amounts of assets

The Equations 6.5 and 6.6 in the whitepaper [1] appear to give an easy way to calculate L, x, and y (from Table 1):

$$L = x_{virtual}\sqrt{P} = \frac{y_{virtual}}{\sqrt{P}}$$

However, x and y here are the *virtual* amounts of tokens, not the real amounts! The math to calculate the real amounts is of x and y is given at the very end of the whitepaper, in Eqs. 6.29 and 6.30. The implementation of this math can be found in the file `LiquidityAmounts.sol`[1].

These equations can be derived from the key Equation 2.2 in the whitepaper:

$$(x_{real} + \frac{L}{\sqrt{p_b}})(y_{real} + L\sqrt{p_a}) = L^2$$

Trying to solve Eq. 2.2 for L directly gives a very messy result. Instead, we can notice that outside the price range the liquidity is fully provided by a single asset, either x or y depending on which side of the price range the current price is. We have three options:

1. Assuming $P \leq p_a$, the position is fully in X, so $y = 0$:

$$(x + \frac{L}{\sqrt{p_b}})L\sqrt{p_a} = L^2 \tag{1}$$

$$x\sqrt{p_a} + L\frac{\sqrt{p_a}}{\sqrt{p_b}} = L \tag{2}$$

$$x = \frac{L}{\sqrt{p_a}} - \frac{L}{\sqrt{p_b}} \tag{3}$$

$$x = L\frac{\sqrt{p_b} - \sqrt{p_a}}{\sqrt{p_a} \cdot \sqrt{p_b}} \tag{4}$$

The liquidity of the position is:

$$L = x\frac{\sqrt{p_a} \cdot \sqrt{p_b}}{\sqrt{p_b} - \sqrt{p_a}} \tag{5}$$

$$\tag{6}$$

2. Assuming $P \geq p_b$, the position is fully in Y, so $x = 0$:

$$\frac{L}{\sqrt{p_b}}(y + L\sqrt{p_a}) = L^2 \tag{7}$$

$$\frac{y}{\sqrt{p_b}} + L\frac{\sqrt{p_a}}{\sqrt{p_b}} = L \tag{8}$$

$$y = L(\sqrt{p_b} - \sqrt{p_a}) \tag{9}$$

The liquidity of the position is:

$$L = \frac{y}{\sqrt{p_b} - \sqrt{p_a}} \tag{10}$$

3. The current price is in the range: $p_a < P < p_b$. I believe the way to think about it is to consider that in an optimal position both assets are going to contribute to the liquidity equally. That is, the liquidity L_x provided by asset x in one side of the range (P, p_b) must be equal to the liquidity L_y provided by the asset y in the other side of the range (p_a, P)[2].

[1] `https://github.com/Uniswap/uniswap-v3-periphery/blob/main/contracts/libraries/LiquidityAmounts.sol#L120`

[2] The Uniswap source code calculates both L_x and L_y and returns the minimum of them: $L = min(L_x, L_y)$. Essentially, this forces the position to be balanced, if it was unbalanced for whatever reasons – either because the user provided an incorrent amount of x or y, or because of rounding errors that are likely if the current price is close to one of the boundaries of the range.

From Eqs. 5 and 10 we know how to calculate the liquidity of a single-asset range. When P is in the range (p_a, p_b), we can think of (P, p_b) as the sub-range where X provides liquidity and (p_a, P) as the sub-range where Y provides liquidity. Plugging this in the Eqs. 5 and 10 and asking that $L_x(P, p_b) = L_y(p_a, P)$ we get:

$$x\frac{\sqrt{P}\cdot\sqrt{p_b}}{\sqrt{p_b}-\sqrt{P}} = \frac{y}{\sqrt{P}-\sqrt{p_a}} \tag{11}$$

The equation Eq. 11 is important because can be solved for either of the five terms x, y, P, p_a, p_b without any reference to liquidity. However, for x and y this is not necessary; a simple modification of Eqs. 4 and 9 is sufficient:

$$x = L\frac{\sqrt{p_b}-\sqrt{P}}{\sqrt{P}\cdot\sqrt{p_b}} \tag{12}$$

$$y = L(\sqrt{P}-\sqrt{p_a}) \tag{13}$$

To sum up:

- If $P \le p_a$, $y = 0$ and x can be calculated by Eq. 4.
- If $P \ge p_b$, $x = 0$ and y can be calculated by Eq. 9.
- Otherwise $p_a < P < p_b$ and x and y can be calculated by Eqs. 12 and 13 respectively.

Conceptually, this result is just a restatement of the Eqs. 6.29 and 6.30 from the whitepaper in a slightly different form. However, use of the Δ in those equations can be confusing, although it is easy to see that if the liquidity is increased from zero, then $\Delta L = L$, so the Δ can be applied for completely new positions as well.

2.2 Calculating the price range from the amounts of assets

2.2.1 Price range boundaries

It is not possible to infer *both* boundaries p_a and p_b at once, given the amount of assets and the current price P within these boundaries. There is just a single equation (Eq. 11), so the result is under-determined when there are two unknown variables. Infinitely many ranges correspond to any given proportion of assets.

Instead, it is possible solve these problems:

- P, x, y and p_a are known; what is the value of p_b?
- P, x, y and p_b are known; what is the value of p_a?
- P, x and y are known. What is the ratio $\frac{p_b}{P}$ given $\frac{p_a}{P}$ and vice versa?

One way to compute these answers is to start by calculating the liquidity on one side of the price. Here and further we assume that the price is within a nonempty range and not equal to the endpoints of the range; formally, $p_a < P < p_b$.

Once L is known, p_a and p_b can be calculated from Eqs. 13 and 12 respectively. It's more convenient to work with square roots, because it simplifies the solutions a lot:

$$\sqrt{p_a} = \sqrt{P} - \frac{y}{L} \tag{14}$$

$$\sqrt{p_b} = L\frac{\sqrt{P}}{L-\sqrt{P}\cdot x} \tag{15}$$

Alternatively, it is possible to skip the liquidity calculation and solve Eq. 11 for the square roots of p_a ans p_a directly:

$$\sqrt{p_a} = \frac{y}{\sqrt{p_b}\cdot x} + \sqrt{P} - \frac{y}{\sqrt{P}\cdot x} \tag{16}$$

$$\sqrt{p_b} = \frac{\sqrt{P}y}{\sqrt{p_a}\sqrt{P}x - Px + y} \tag{17}$$

2.2.2 Price range proportions

The user may also think about price range in terms of increase or decrease in the current price. Let us introduce new symbols c and d for that, such that c^2P is equal to the upper bound of the range and d^2P is equal to the lower bound:

$$c \stackrel{\text{def}}{=} \sqrt{\frac{p_b}{P}} \tag{18}$$

$$d \stackrel{\text{def}}{=} \sqrt{\frac{p_a}{P}} \tag{19}$$

Now its possible to use Eq. 11 to express them in terms of each another:

$$c = \frac{y}{(d-1) \cdot P \cdot x + y} \tag{20}$$

$$d = 1 + \frac{(1-c) \cdot y}{c \cdot P \cdot x} \tag{21}$$

Now if it is known that for example p_a corresponds to 70 % of the current price (c^2 is equal 70 % = 0.7), the % of the current price corresponding to p_b can be calculated by computing d^2 using Eq. 21.

3 Applications

3.1 Implementation details

When applying this math in source code, be aware of the following aspects [1]:

- *Ticks*. Price ranges in Uniswap have discrete boundaries called ticks. The price of the i-th tick defined to be $p(i) = 1.0001^i$. Only specific price-points, corresponding to initialized ticks, can serve as price range boundaries.
- *Tick spacing*. Not all ticks can be initialized. The exact tick indexes that can be indexed depends of the fee levels of the pools. 1 % pools have the widest tick spacing, 0.05 % pools the smallest.
- *Fixed point math*. Uniswap v3 implements the math in fixed point way in order to minimize rounding errors[3] The math adds new challenges, e.g. need to multiply or divide numbers with the fix-point base to keep their range correct.
- *Decimals*. ERC20 tokens define a field called "decimals". Internally in the Ethereum ecosystem, the amounts of cryptocurrencies operated on are expressed as integers. To convert the amount of an ERC20 token from this internal representation to a human readable value, division by $10^{decimals}$ is necessary. Different ERC20 contracts define a different number of decimals for their token; for example, DAI has 18 decimals while USDC has 6. The human-readable price of 1 USDC is around 1 DAI; but from the perspective of liquidity calculations, the price of 1 USDC in terms of DAI is approximately $10^{18-6} = 10^{12}$.

3.2 Examples

In this example section, we ignore the implementation details mentioned above in order to increase the clarity of the explanations. Note that this may lead to small numerical errors compared with the results obtained from the Uniswap code and UI.

3.2.1 Example 1: Amount of assets from a range

Problem: A user has 2 ETH and wants to set up a liquidity position in an ETH/USDC pool. The current price of ETH is 2000 USDC and target price range is from 1500 to 2500 USDC. How much USDC do they need?

Solution: First, calculate the liquidity of the top half of the range by using Eq. 4 (replace p_a with P to get the top half):

$$L = \frac{\sqrt{p_b} - \sqrt{P}}{\sqrt{P}\sqrt{p_b}}$$

[3] You should do the same! For this, JavaScript needs to use a library such as JSBI. Python has support for large integers and (since version 3.8) integer square roots out of the box.

Then, use the value of L to calculate y using Eq. 9:

$$y = L\sqrt{P}\sqrt{p_a}$$

The answer is $y = 5076.10$ USDC.

3.2.2 Example 2: Range from amounts of assets

Problem: A user has 2 ETH and 4000 USDC, and wants to use 3000 USDC per ETH as the top of the price range. What is the bottom of the range that ensures the opened position uses the full amount of their funds?

Solution: It is possible to calculate the liquidity of the position using the same half-range trick as in the Example 1 above, and then calculate the lower bound of the range of that. However, p_a can also be calculated directly by Eq. 16:

$$p_a = (\frac{y}{\sqrt{p_b} \cdot x} + \sqrt{P} - \frac{y}{\sqrt{P} \cdot x})^2$$

The answer is $p_a = 1333.33$ USDC. The lower price is two thirds of the current price, and the current price is two thirds of the upper price: this happens because the initial values of USDC and ETH are equal.

3.2.3 Example 3: Assets after a price change

Problem: Using the liquidity position created in Example 2, what are asset balances when the price changes to 2500 USDC per ETH?

Solution: First, calculate the liquidity of the position, using 1333.33 as the value of p_a:

$$L_x = x\frac{\sqrt{p_b} - \sqrt{P}}{\sqrt{P}\sqrt{p_b}}$$
$$L_y = \frac{y}{\sqrt{p_b} - \sqrt{p_a}}$$

Using 64-bit floating point math, the values $L_x = 219.08820141977066$ and $L_y = 219.08847528053587$ respectively. L is the minimum of these two: $L = min(L_x, L_y)$, corresponding to L_x in this case.

Now, setting $P' = 2500$ we can find x' and y' using Eqs. 4 and 9:

$$x' = L\frac{\sqrt{p_b} - \sqrt{p_a}}{\sqrt{p_a}\sqrt{p_b}}$$
$$y' = L\sqrt{P'}\sqrt{p_a}$$

The answer is $x = 0.85$ ETH and $y = 6572.89$ USDC. Impermanent loss can be computed easily from this.

We can alternatively compute this answer using the delta math provided in the whitepaper, Eqs. 6.14 and 6.16:

$$\Delta\sqrt{P} = \sqrt{P'} - \sqrt{P}$$
$$\Delta\frac{1}{\sqrt{P}} = \frac{1}{\sqrt{P'}} - \frac{1}{\sqrt{P}}$$
$$\Delta x = \Delta\frac{1}{\sqrt{P}} \cdot L$$
$$\Delta y = \Delta\sqrt{P} \cdot L$$
$$x' = x + \Delta x$$
$$y' = y + \Delta y$$

The values of the deltas are: $\Delta x = -1.15$ ETH and $\Delta y = +2572.89$ USDC.

Acknowledgments

Various discussions on the Uniswap Discord have both helped and motivated the author to work on this note.

Disclaimer

The information provided in this article does not constitute financial advice. The author is not responsible for the use of this information, and does not guarantee the accuracy of this information. This article was written in the author's free time, reflects the personal opinions of the author, and is not related to his professional activity or to his employer.

References

[1] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 core," Tech. rep., Uniswap, Tech. Rep., 2021.