LIQUIDITY MATH IN UNISWAP V3

TECHNICAL NOTE

Atis Elsts
atis.elsts@edi.lv

September 30, 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], which has already attracted significant attention from the research community [2–4]. However, the whitepaper 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[1]

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:

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

[1]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`, `amount`	

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`[2].

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:

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

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

$$\frac{L}{\sqrt{p_b}}(y + L\sqrt{p_a}) = L^2 \tag{6}$$

$$\frac{y}{\sqrt{p_b}} + L\frac{\sqrt{p_a}}{\sqrt{p_b}} = L \tag{7}$$

$$y = L(\sqrt{p_b} - \sqrt{p_a}) \tag{8}$$

The liquidity of the position is:

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

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

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)[3].

From Eqs. 5 and 9 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 9 and asking that $L_x(P, p_b) = L_y(p_a, P)$ we get:

$$x\frac{\sqrt{P}\cdot\sqrt{p_b}}{\sqrt{p_b}-\sqrt{P}} = \frac{y}{\sqrt{P}-\sqrt{p_a}} \tag{10}$$

The equation Eq. 10 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 8 is sufficient:

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

$$y = L(\sqrt{P}-\sqrt{p_a}) \tag{12}$$

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. 8.
- Otherwise $p_a < P < p_b$ and x and y can be calculated by Eqs. 11 and 12 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 the whitepaper can be confusing to new users – a delta from what, exactly? What if it is a completely new position? The equations 4–12 above avoid any mention of deltas for this reason, aiming for simplicity. Of course, on a deeper look it is clear that the whitepaper Eqs. 6.29 and 6.30 can be still applied even for a new position: in that case we simply need to take $\Delta L = L$.

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. 10), 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. The price would reach p_b if if increased z % from the current price. What % drop in the price does p_a correspond to? In other words, what is the ratio $\frac{p_a}{P}$ given the ratio $\frac{p_b}{P}$?

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. 12 and 11 respectively. It's more convenient to work with square roots, because it simplifies the solutions a lot:

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

$$\sqrt{p_b} = L\frac{\sqrt{P}}{L-\sqrt{P}\cdot x} \tag{14}$$

[3]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 incorrect 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.

Alternatively, it is possible to skip the liquidity calculation and solve Eq. 10 for the square roots of p_a ans p_a directly:

$$\sqrt{p_a} = \frac{y}{\sqrt{p_b} \cdot x} + \sqrt{P} - \frac{y}{\sqrt{P} \cdot x} \tag{15}$$

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

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:

$$c \stackrel{\text{def}}{=} \sqrt{\frac{p_b}{P}} \tag{17}$$

$$d \stackrel{\text{def}}{=} \sqrt{\frac{p_a}{P}} \tag{18}$$

Clearly the following is true:

$$\sqrt{P} = \frac{\sqrt{p_b}}{c} \tag{19}$$

$$\sqrt{P} = \frac{\sqrt{p_a}}{d} \tag{20}$$

$$\sqrt{p_b} = c \cdot \sqrt{P} \tag{21}$$

$$\sqrt{p_a} = d \cdot \sqrt{P} \tag{22}$$

Now its possible to use Eq. 10 to express them in terms of each another:

$$c = \frac{y}{(d-1) \cdot P \cdot x + y} \tag{23}$$

$$d = 1 + \frac{(1-c) \cdot y}{c \cdot P \cdot x} \tag{24}$$

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. 24.

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 of 200 ticks, 0.3 % pools intermediate of 60 ticks, and 0.05 % pools the smallest one: 10 ticks.
- *Fixed point math*. Uniswap v3 uses fixed point math. This is because Solidity has no support for floating point numbers, and because fixed point math helps to minimize rounding errors[4] 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 $10^{18-6} = 10^{12}$.

[4] In your code that interfaces with Uniswap, you should also use fixed point math – even if your language does support floating point! In JavaScript, use a library such as JSBI. Python has support for large integers and (since version 3.8) integer square roots out of the box.

3.2 Examples without implementation details

In this example subsection, 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 $x = 2$ ETH and wants to set up a liquidity position in an ETH/USDC pool. The current price of ETH is P =2000 USDC and target price range is from $p_a = 1500$ to $p_b = 2500$ USDC. How much USDC (y) do they need?

Solution: First, calculate the liquidity of the top half of the range by using Eq. 5 (replace p_a with P to get the top half):

$$L_x = x\frac{\sqrt{P} \cdot \sqrt{p_b}}{\sqrt{p_b} - \sqrt{P}}$$

Then, use the value of L to calculate y using Eq. 8:

$$y = L_x(\sqrt{P} - \sqrt{p_a})$$

The answer is $y = 5076.10$ USDC.

3.2.2 Example 2: Range from amounts of assets

Problem: A user has $x = 2$ ETH and $y = 4000$ USDC, and wants to use $p_b = 3000$ USDC per ETH as the top of the price range. What is the bottom of the range (p_a) 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. 15:

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

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 $P = 2500$ USDC per ETH?

Solution: First, calculate the liquidity of the position, using the previously calculated $p_a = 1333.33$:

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

Using 64-bit floating point math, the values are $L_x = 487.41718030204123$ and $L_y = 487.4144693682443$ 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 8:

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

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

3.3 Examples that include tick math and price conversions

3.3.1 Uniswap v3 tick math

Tick math is required to interpret both the values provided by the Uniswap v3 API and the data indexed in the Uniswap v3 subgraph[5]. First, let briefly recap the tick math from the whitepaper [1]. Uniswap v3 maps the continuous space of all possible prices to a discrete subset indexed by $ticks$. A tick has unique relation with price, defined by the tick base parameter, which is equal to 1.0001 in Uniswap. The price corresponding to the i-th tick is:

$$p(i) = 1.0001^i$$

This implies:

$$i = \log_{1.0001} p(i)$$

When working with square roots of prices:

$$\sqrt{p(i)} = 1.0001^{i/2}$$
$$i = 2 \cdot \log_{1.0001} \sqrt{p(i)}$$

3.3.2 Converting prices to a human-readable form

Internally, the price is just the relation between y and x. However, in order to show it in a UI, it must be scaled taking into account the number of decimals of the assets X and Y.

Let's analyze an Uniswap v3 NFT to demonstrate this, and let's take the NFT with ID 60000[6] as an example. As shown in Fig. 1, this NFT has "Min Tick" equal to 200240 and "Max Tick" equal to 200700. Converting these tick numbers to prices we get this price range:

$$p_a = 496452748.01$$
$$p_b = 519821773.17$$

The problem is that USDC (the first assert, X) has just $decimals_x = 6$ decimals while wrapped ETH (the second asset, Y) as $decimals_y = 18$ decimals. Since price is $\frac{y}{x}$, the price without the decimals is equal to:

$$p_{adjusted} = \frac{y}{10^{decimals_y}} / \frac{x}{10^{decimals_x}} = \frac{y}{x} \cdot \frac{10^{decimals_x}}{10^{decimals_y}} = \frac{y}{x} \cdot 10^{x-y} = p \cdot 10^{x-y}$$

In the example pool:

$$p_{adjusted_a} = 496452748.01 \cdot 10^{6-18} = \frac{496452748.01}{10^{12}} = 0.00049645274801$$
$$p_{adjusted_b} = 519821773.17 \cdot 10^{6-18} = \frac{519821773.17}{10^{12}} = 0.00051982177317$$

[5] `https://thegraph.com/legacy-explorer/subgraph/uniswap/uniswap-v3`
[6] `https://app.uniswap.org/#/pool/60000`

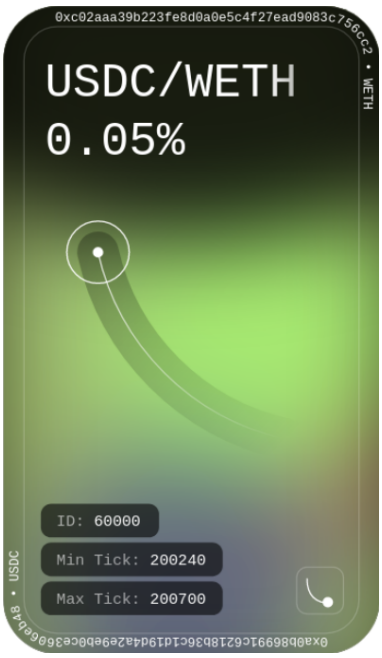

Figure 1: A sample Uniswap position NFT.

These express the price of USDC in terms of ETH. To get ETH in terms of USDC, invert the prices:

$$\frac{1}{p_{adjusted_a}} = 2014.29$$
$$\frac{1}{p_{adjusted_b}} = 1923.74$$

Note that the last range is inverted, with inverted price at the top tick being lower than the inverted price at the bottom tick.

3.3.3 The amount of assets in the current tick range

In order to learn the amount of the assets in a price range, first we must obtain the liquidity of the pool. A pool variable tracks the liquidity of the current tick range in particular. Unlike in Uniswap v2, where the liquidity of a pool remains constant as long as assets are not added or removed to the pool, the pool liquidity in Uniswap v3 may change when the price crosses a tick range boundary. In particular, it changes when crossing the boundary of a tick with non-zero `liquidityNet` value.

The liquidity of a pool can be queried either from The Graph, or directly from a blockchain node. Let's assume that The Graph is used, and look at the USDC/ETH pool with 0.3 % fee rate with the following query[7]:

```
query pools {
  pools (where: {id: "0x8ad599c3a0ff1de082011efddc58f1908eb6e6d8"}){
    tick
    liquidity
  }
}
```

At the time of writing this, the query returns $L = 22402462192838616433$ and $tick = 195574$, corresponding to price of 3211.84 USDC/ETH.

[7]The full code of this example can be found at `https://github.com/atiselsts/uniswap-v3-liquidity-math/blob/master/subgraph-liquidity-query-example.py`

The tick spacing of a pool is determined by its fee rate. For 0.3 % pools, the tick spacing is equal to 60. This means that only ticks divisible by 60 can be initialized. The ticks nearest to the returned value that have this property are:

$$tick_{bottom} = 195540$$
$$tick_{top} = 195600$$

As a result, the price range is:

$$p_a = 1.0001^{tick_{bottom}} = 1.0001^{195540}$$
$$p_b = 1.0001^{tick_{top}} = 1.0001^{195600}$$

Let us use Eq. 11 and 12 to calculate the maximum amount of USDC or ETH that can be obtained from the liquidity in the current tick range:

$$x = L\frac{\sqrt{p_b} - \sqrt{P}}{\sqrt{P} \cdot \sqrt{p_b}} = 1649346952148$$
$$y = L(\sqrt{P} - \sqrt{p_a}) = 671393300975203516416$$

When adjusted for the decimals of the respective tokens:

$$x_{adjusted} = \frac{x}{10^6} \approx 1\,649\,347$$
$$y_{adjusted} = \frac{y}{10^{18}} \approx 671.39$$

Approximately 1.65 million USDC and 671 ETH are locked in the current tick range in this pool. This means that someone buying 1.65 million USDC would move the current price to one end of the current price range. If some was buying 671 ETH instead, that would move the price to the other end of the range.

Acknowledgments

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

Disclaimer

This article is for general information purposes only and does not constitute investment advice. The author does not guarantee the accuracy of the information provided in this article. This article was written in the author's free time and is not related to his professional activity. Furthermore, it reflects the current opinions of the author, and does not necessarily reflect the opinions of his employer.

References

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

[2] M. Neuder, R. Rao, D. J. Moroz, and D. C. Parkes, "Strategic Liquidity Provision in Uniswap v3," *arXiv preprint arXiv:2106.12033*, 2021.

[3] J. Clark, "The Replicating Portfolio of a Constant Product Market with Bounded Liquidity," *Available at SSRN*, 2021.

[4] J. Yin and M. Ren, "On Liquidity Mining for Uniswap v3," *arXiv preprint arXiv:2108.05800*, 2021.