The Costs of Automated Market Makers

The Motivation for Automated Market Making

In the early days of the decentralized finance (DeFi) ecosystem, protocols like EtherDelta adopted the classic order book design that users with traditional finance (TradFi) or centralized finance (CeFi) backgrounds were already familiar with. In an order book, market makers provide liquidity by placing limit orders on both sides of the market (bid and ask), and market orders are filled either by the lowest ask price or the highest bid price. While some exchanges like dYdX adopted the order book model with partial decentralization (e.g., handling operations such as order matching on their servers), some exchanges operate fully on-chain (see Demex). Users of such exchanges benefit from complete transparency but may suffer from slow execution of the underlying blockchain or high transaction fees due to complex on-chain operations.

Independent from its implementation (fully vs. partially on-chain), the order book design works efficiently when there is sufficient liquidity of assets traded. As liquid markets enable low bid-ask spread and fast order filling, they are preferred by traders. However, not all markets are liquid. The emergence of fungible tokens like ERC-20 led to new markets in the DeFi space. Although popular tokens always attract attention and have sufficient market makers, some are rarely traded. Such tokens have illiquid markets that can end up having large bid-ask spreads. In order book exchanges, traders of these assets suffer from matching time of their orders due to the lack of liquidity provision by the market makers.

The *Automated Market Maker* (AMM) design emerged to enable efficient trading in illiquid markets. As the name suggests, AMMs enable intermediary-free, algorithmic market-making where order books are replaced with liquidity pools (asset baskets) maintained through smart contracts. Unlike order books, AMMs do not require market makers to update their quotes regularly. Instead, these protocols only need an initial supply by liquidity providers (LP) to create asset pools, and the protocol's smart contracts automatically handle the rest (e.g., market-making and price discovery).

The smart contracts of AMMs are programmed to follow market-making functions such as *constant product*. Traders can directly trade against these protocols without waiting for their order to get matched as in order book exchanges. While this automation provides efficiency for the traders, it also reduces the number of on-chain operations required. As a result, AMMs are considered more suitable for fully on-chain implementation and adoption compared to order books.

Today, AMMs such as Uniswap take the dominant market share of the decentralized exchange space. According to the famous "DEX metrics" dashboard on Dune, in the last seven days, close to $15 B in trade volume accumulated on decentralized exchanges. Around 71% of this volume took place on Uniswap. Although AMMs are popular among users due to the benefits they offer to traders (e.g., low-cost on-chain operations, always available liquidity) and the LPs (e.g., no price maintenance), they are not pure profit machines. In this article, we present the cost of using AMMs for traders and the LPs and discuss the fine details that users should consider beforehand.

A Brief Refresher on Constant Product Markets

AMMs can instantly offer prices based on the liquidity available in the asset pools and the followed *pricing function*. While various pricing functions exist, arguably the simplest and the most popular one is constant product which Uniswap v1 and v2 adopt. In two-token, constant product markets, say a market for *X* and *Y*, the product of the assets' reserves is kept unchanged ($x * y = k$) while their relative ratio (y/x) determines the AMM price p_{amm}.

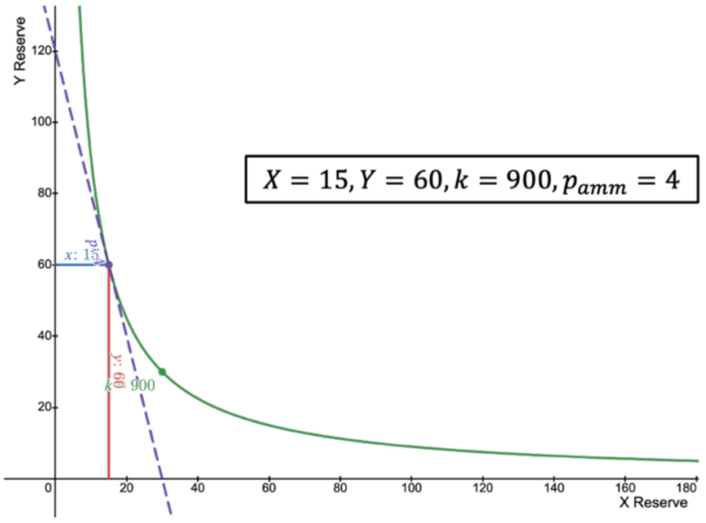


Figure 1: The initial curve of the AMM with 15 X and 60 Y assets.

Each new trade on AMMs changes the asset pool reserves ($\Delta x, \Delta y$) while keeping k constant ($(x - \Delta x) * (y - \Delta y) = k$). Depending on the size of the trade, the AMM price p_{amm} also changes. For an infinitesimally small trade, p_{amm} would be kept the same and be equal to the execution price $p_{execution}$. With the increasing size of the trade, the difference between p_{amm} and $p_{execution}$ also grows. This difference is also known as expected slippage. In the following section, we will further discuss the impact of slippages on users' trades.

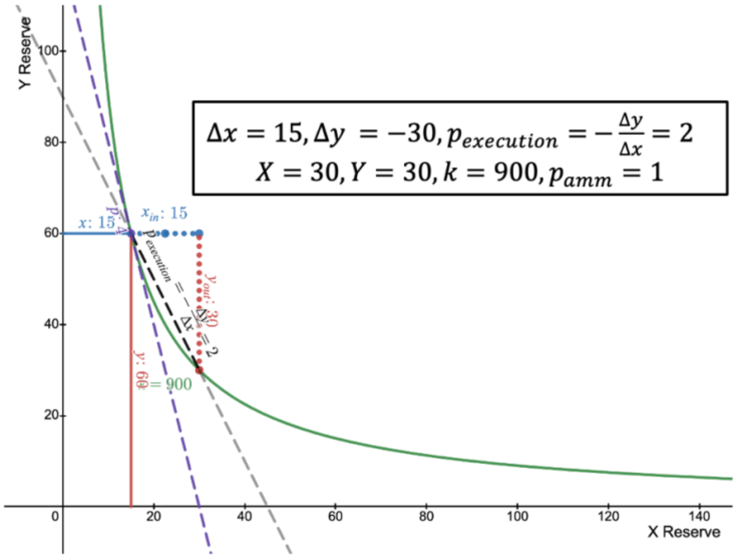


Figure 2: The updated price of the AMM after a trade of 15 X for 30 Y.

Unlike trades on an AMM, liquidity addition or withdrawal (L_x, L_y) keeps the AMM price constant while updating k. To keep p_{amm} unchanged, LPs are required to supply an equal value of assets to both liquidity pools ($y/x = \frac{y+L_y}{x+L_x}$). Depending the on the size of their supply, LPs receive tokens from the underlying protocol which determines their share in the pool.

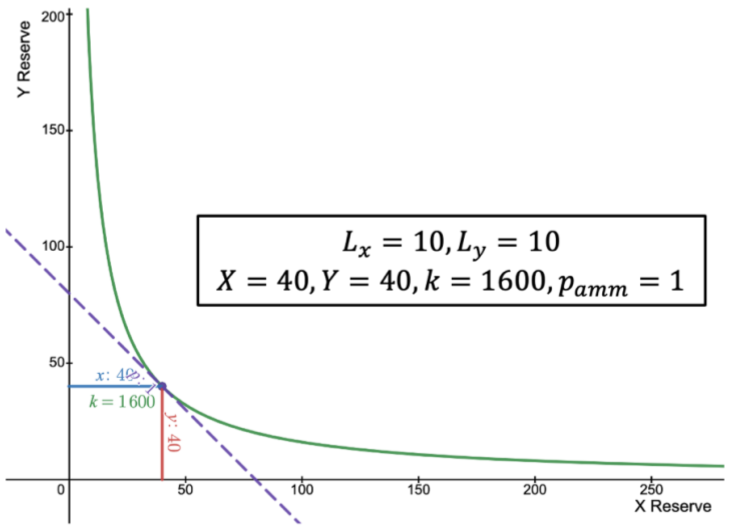


Figure 3: The updated curve of the AMM after a liquidity provision of 10 X and 10 Y.

AMM Traders

The 24-hour trading volume on Uniswap ($1.2 B) indicates how popular constant product AMMs are among traders. Although there are valid reasons for this, such as instant liquidity for any trade size, availability to trade more exotic tokens, and simple, user-friendly interfaces, traders must be cautious with the slippage rates.

Slippage on order book exchanges refers to the change in the price of an asset from the placement of the order until its execution. On AMMs, this type of slippage is known as *unexpected slippage*. However, AMMs also have an *expected slippage* which indicates the change in the price *during* the execution of a trade. It is called expected slippage, as the amount can be pre-calculated with respect to the pricing formula of the AMM. The expected slippage rate grows with the relative size of the trade to the liquidity in the pools. The deeper the pools are or, the smaller the trade is, the less expected slippage occurs.

Like on order book exchanges, unexpected slippage also occurs on AMMs when a trade is executed from a price different than when the order was placed. This deviation is due to the change in the AMM state (i.e., liquidity in the pools) during the waiting time between the trade's request and its execution. As users submit their trades in the form of blockchain transactions, it takes some time until they are included in a block. Meanwhile, another trade interacting with the same AMM can get executed before theirs. In such cases, the pre-calculated execution price can change, and the trader would leave with a different amount of assets than the initially expected amount. To minimize this effect, AMMs require traders to set an unexpected slippage tolerance for their trades such that if the price shifts more than their tolerated slippage rate, the transaction reverts, and the trade never takes place.

For instance, if a trader intends to swap ETH for USDC, he would benefit from unexpected slippage if a trade in the opposite direction occurs before his trade (drives up the ETH price even more, and the trader ends up with more USDC than expected). However, it is more probable that unexpected slippage will not favor the trader. This is due to the adversaries monitoring the awaiting transactions pool to find any opportunities to execute frontrunning and backrunning attacks (or, the combination, a *sandwich attack*).

For example, let us think about an ETH/USDC market consisting of 10 ETH and 10,000 USDC tokens in its pools. Initially, the AMM price p_{amm} is 1000 USDC and k is 100,000. Alice, a benign trader, submits a transaction to the Ethereum blockchain T_{Alice} where she interacts with the AMM, calling the swap function. Alice initially inputs 10,000 USDC and expects to receive 5 ETH. Hence, her expected slippage is 5 ETH or 50% (remember that if there were no slippage, Alice would have received 10 ETH for 10,000 USDC). Alice also configures her trade's unexpected slippage to 90%, although she is unsure exactly what it does.

Awaiting Transactions Pool

Position	ID	Content	Fee
1	T_{Alice}	AMM swap (input: 10,000 USDC, unexpected slippage: 90%)	0.02 ETH

Bob, an adversary trader, monitors the awaiting transactions pool and spots Alice's transaction. Upon further examination, Bob realizes that Alice set her unexpected slippage really high, which means he can sandwich her transaction with a frontrunning and a backrunning transaction to make profits.

First, Bob calculates Alice's worst-case expected ETH amount by applying her unexpected slippage rate to her expected output amount ($5 - 5 * \frac{90}{100} = 0.5\ ETH$). To make Alice receive 0.5 ETH for 10,000 USDC, Bob knows that his frontrunning transaction T_{Bob_1} has to leave the AMM state in 2.5 ETH and 40,000 USDC. Hence, he configures T_{Bob_1} to swap 30,000 USDC for 7.5 ETH, with a transaction fee a little higher than T_{Alice} to ensure that T_{Bob_1} gets executed first.

Awaiting Transactions Pool

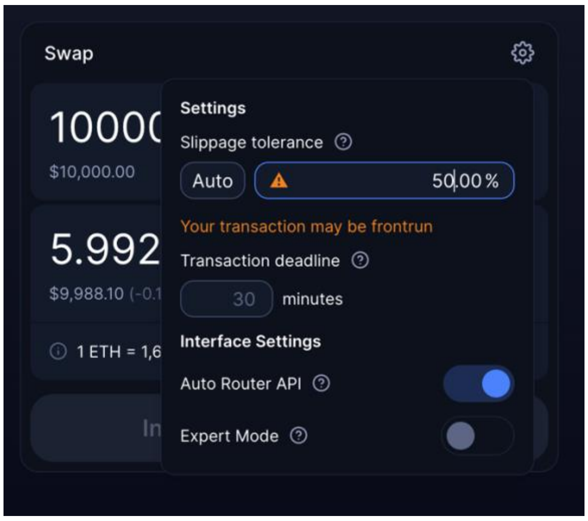
Position	ID	Content	Fee
1	T_{Bob_1}	AMM swap (input: 30,000 USDC, unexpected slippage: 1%)	0.03 ETH
2	T_{Alice}	AMM swap (input: 10,000 USDC, unexpected slippage: 90%)	0.02 ETH

To complete his attack, Bob must backrun Alice and make the opposite trade of his frontrunning transaction T_{Bob_1}. Thus, Bob issues T_{Bob_2} where he swaps back the 7.5 ETH he received from T_{Bob_1}. As T_{Bob_2} must be executed right after T_{Alice}, Bob sets a transaction fee that is slightly less than Alice's. Finally, applying the constant product formula, Bob calculates his expected output to be $50{,}000 - \frac{100{,}000}{2+7.5} = 39{,}473.68$ USDC from T_{Bob_2}.

Awaiting Transactions Pool

Position	ID	Content	Fee
1	T_{Bob_1}	AMM swap (input: 30,000 USDC, unexpected slippage: 1%)	0.03 ETH
2	T_{Alice}	AMM swap (input: 10,000 USDC, unexpected slippage: 90%)	0.02 ETH
3	T_{Bob_1}	AMM swap (input: 7,5 ETH, unexpected slippage: 1%)	0.01 ETH

After all transactions in the awaiting transactions pool get executed in their respective position, Bob ends up making $39{,}473.68 - 30{,}000 = 9{,}473.68\ USDC$ profit (excluding transaction fees), while Alice only receives 0.5 ETH for 10,000 USDC, as opposed to her expected output of 5 ETH. Alice suffered from Bob's sandwich attack because of the high unexpected slippage tolerance she set for T_{Alice}. If she had set it to 1% (as recommended by Uniswap), then Bob would not be able to conduct his attack, or otherwise, Alice's transaction would have reverted.


Settings

Slippage tolerance: Auto | 50.00 %

Your transaction may be frontrun

Transaction deadline: 30 minutes

Interface Settings

Auto Router API

Expert Mode

Figure 4: A screenshot taken from Uniswap UI showing a warning since the slippage tolerance is set too high.

AMM Liquidity Providers

AMMs incentivize users to add liquidity into the pools by offering some fee (usually 0.3%) from every executed trade. These fees are distributed to the LPs based on their share in the pools. Over time, these fees can grow into large amounts, especially in markets with high trading volume, such as ETH/USDC on Uniswap v2 ($46 M in the last seven days).

However, liquidity provision is not always profitable. There is also a cost associated with it resulting from the adverse selection that LPs face as they accept to execute trades on both sides of the market by providing assets into an AMM. This becomes an issue when a trader with more information trades on the AMM. The profits that the trader makes are losses of the LPs. Ironically, AMMs depend on these informed traders, such as arbitrageurs, to maintain the price balance with external markets.

To analyze whether it makes sense to provide liquidity into a pool, one can compare it to a strategy like going long (i.e., holding the assets). For example, let us assume Alice has provided $2000 worth of tokens to the ETH/USDC pool on an AMM (1 ETH and 1000 USDC). If the ETH/USDC price on an external market jumps to $4000, an arbitrageur can buy 0.5 ETH for 1000 USDC on the AMM and sell it for 2000 USDC on the external market, pocketing $1000 profits while bringing the AMM price to $4000 ($2000/0.5$).

After the arbitrage balances the prices, if Alice withdraws her liquidity, she receives 0.5 ETH and 2000 USDC, which have a total value of $4000. Compared to the initial worth of her portfolio, Alice now has $2000 more, but what if Alice just held her portfolio and never provided liquidity into the AMM? In that case, her total portfolio would have been worth \$5000 as she initially had 1 ETH and 1000 USDC.

The $\$5000 - \$4000 = \$1000$ that Alice lost by providing liquidity into the AMM is known as "Loss-vs-Holding" or "Impermanent Loss" (IL). IL refers to the value difference between the liquidity provider's deposited assets under current prices and their value in the AMM. It is caused by the volatility in the token prices, which creates arbitrage opportunities. However, as the name suggests, this loss is not permanent.

In our example scenario, if Alice does not withdraw her liquidity after the arbitrage and the external market price goes back to \$1000, then someone will do the arbitrage on the AMM to take its price back to \$1000. If Alice withdraws her liquidity now, she will receive 1 ETH and 1000 USDC, which has the same value as her initial portfolio. In this case, IL would be 0, as Alice ends up with the same portfolio value as she would have if she just held her assets. Hence, IL only gets realized in the scenario where LPs withdraw their assets before the price returns to where it was when they deposited their assets. However, can we conclude that Alice has no loss in this case?

As mentioned earlier, LPs in AMMs do not update their prices as order book market makers do. In AMMs, prices are balanced by arbitrageurs' trades. The profits that arbitrageurs make are paid by LPs, as providing liquidity into an AMM pool means accepting to be on either side of a trade (in the case of arbitrages, being on the *wrong* side). However, as we have seen, if the AMM price eventually returns to the point where an LP joins the pool, IL indicates no loss. Hence, there is a contradiction.

In "Automated Market Making and Loss-Versus-Rebalancing," Milionis et al. (2022) argue that IL does not isolate the adverse selection costs faced by LPs. They show that LPs suffer from trades of informed traders, like arbitrageurs, by being on the wrong side of the trade. Therefore, the cost of providing liquidity into a pool increases with every informed trade (unlike IL). To calculate this cost, Milionis et al. introduce "Loss-vs-Rebalancing" (LVR). Although there exist different definitions, the most straightforward way to think about LVR is to consider it as the extra value an LP can gain by dynamically rebalancing his portfolio based on the changing prices rather than passively locking his assets into a liquidity pool.

The rebalancing strategy is simple, copy the AMM trade but execute it on an external market. In our example, Alice initially deposits 1 ETH and 1000 USDC into the AMM. After the external market price jumps to \$4000, an arbitrageur balances the AMM price by swapping 1000 USDC for 0.5 ETH. Now, let us consider a scenario where Alice does not provide liquidity to the AMM but duplicates its trades. In this scenario, Alice has to sell 0.5 ETH as the AMM does, but she conducts this trade on the external market. Hence, she gets 2000 USDC for 0.5 ETH, and her portfolio now has a total value of \$5000 (0.5 ETH and 3000 USDC). Instead of rebalancing, if Alice had supplied her assets to the liquidity pool, her total portfolio value would have been \$4000 (0.5 ETH and 2000 USDC). Hence, LVR is $\$5000 - \$4000 = \$1000$.

Continuing with our example, if the price on the external market drops back to \$1000, the arbitrageur again balances the AMM price, this time by swapping 0.5 ETH for 1000 USDC. Following the rebalancing strategy, Alice executes the AMM trade (buy 0.5 ETH) on the external market and pays 500 USDC for it. After the trade, Alice has 1 ETH and 2500 USDC in her portfolio. The total value of her assets ($3500) is again greater than what her assets would have been worth ($1\,ETH + 1000\,USDC = \$2000$) if she had provided liquidity into the AMM. The total profit of the rebalancing strategy over liquidity provision is now $\$3500 - \$2000 = \$1500$, and LVR of this last trade is $500.

LVR accumulates with every trade, and LPs suffer more from it with increasing volatility in the price. Before committing to a liquidity pool, LPs should consider their loss to LVR and how much trading volume is needed so that their income from trading fees exceeds LVR losses. LPs can also analyze how much they have already lost to LVR by looking at the transaction history and adding up the LVR for every trade on the AMM they hold shares. If this value is greater than their income from the trading fees, then becoming an LP was not a good idea in the first place.

Conclusion

In this article, we have discussed the emergence of AMMs in the DeFi space as an alternative to order books and analyzed the potential costs they incur on traders and liquidity providers. We have shown through examples how traders can suffer from setting high slippage rates. We also examined liquidity providers' costs and discussed how they could lose money by passively committing liquidity into an AMM instead of adopting a dynamic portfolio rebalancing strategy. To minimize costs, users should consider these finer points when trading on or providing liquidity to AMMs.

Note: Figures 1,2, and 3 are created with the “Uniswap math (with LVR)” Desmos graph.