# Simulation of the Uniswap Exchange

Christopher Murray, Michael Einhorn, Varun Lakshmanan, Sri Ghantasala

## Abstract

Financial markets are incredibly complex systems that can be difficult to predict that can easily transition from stability to instability. There has been an abundance of research into modeling and analyzing financial markets in an attempt to better understand these systems. Our project is focused on performing such an analysis on a simulated decentralized cryptocurrency exchange. Our goal is to develop a better understanding of how the behaviors of many different agents within the exchange influence the system as a whole. One form of instability is known as slippage which is where the price of the exchange changes during the transaction for trades that are large compared to the liquidity of the exchange. High fees can attract more liquidity providers but also make the pool more expensive to use creating a tradeoff that can be beneficial or not for different agents.

## Description

There are a large number of different cryptocurrencies people can trade with on a variety of traditional and decentralized exchanges. In addition to trading, providing liquidity and arbitraging for profit, there are people that need to trade for a specific currency to use features on the network, for example paying gas fees for transactions with Eth.

Uniswap uses an automated market marker to run a decentralized exchange. Instead of trading with other people, users trade with or provide liquidity to a contract using a formula. The unique feature of Uniswap is concentrated liquidity. Virtual reserves follow the constant product rule, but only need enough real reserves to support trading in its price range. When making a transaction with the pool, users pay a flat gas fee to the network, and a % fee to the liquidity pool which is higher for more volatile pairs. Earned fees are not reinvested and stored separately until collected. Price movements in either direction can cause a loss in the value of the position relative to just holding the assets, and this is called Impertinent loss. This effect is larger for smaller liquidity ranges, and when outside the range all of the position is in the less valuable asset [6].

The system is discrete time as changes are only made in blocks about 4 times per minute [7]. A transaction can be sent at any time, but it is only processed at specific times when incorporated into a block. Due to the decentralization of the blockchain, users can’t be confident the transaction was processed until multiple blocks have been added after. For Ethereum, which is the chain Uniswap is currently on, this delay is 7 blocks at a minimum which takes about 2 minutes, and 250 blocks on the safest side which takes about 1 hour [7].

## Literature Review

### I. Agents in a Traditional Market: Fundamentalists vs Chartists

#### 1.1 Summary

A popular approach to modeling the behavior of agents in a traditional market is to split them into two groups: fundamentalists, who inform their behavior by their belief in the intrinsic value of the asset they are trading, and chartists, who inform their behavior based on forecasting trends from past data. In [5] and [3], this modeling approach is evaluated from the standpoint of validating a simulation based on real data and analyzing the stability of configurations that may arise from a simulation using this model, respectively. The former demonstrated the validity of this approach using the Bank of America stock. The latter discovered that there were equilibria where either type was able to drive the other type out of the market and equilibria where they coexisted within the market, however the only equilibria that the agents could adaptively learn were ones in which the fundamentalists outperformed the chartists.

### 1.2 Relevance

These papers provide us with a foundational understanding of how a market can be divided into agents of different types whose local behavior creates emerging properties within a simulated market system that is representative of real world observations. This knowledge will be useful for us going forward; however, due to the fact these papers are analyzing traditional stock trading, we will need to look towards other resources to gain deeper insights into how we can develop models for crypto exchanges.

## II. An analysis of Uniswap markets (Section 4.2)

### 2.1 Summary

This paper models a Uniswap market, and its interaction with a reference market. We will specifically focus on section 4.2 of this paper here. Section 4.2 of this paper focuses on the different agents involved in this simulation. There are three types of agents: arbitrageurs, liquidity providers, and traders. Arbitrageurs are those who trade between the Uniswap market and the regular market, and are trying to increase their profits, by following a quadratic cost of risk model (in this case). Another type of agent is a liquidity provider, who can further be classified into two categories: initial and rational. Initial liquidity providers provide a certain amount (in the form of coins and money) to the reserve and charge a certain amount on each transaction made. Rational liquidity providers use optimization techniques to decide what and how much to trade, so that they can balance their portfolio (which can also be modeled using a few equations - they are omitted here to increase readability, but are outlined in the paper). Lastly, traders are the agents who trade a certain amount of money for a certain amount of coins, or the other way around [2].

### 2.2 Relevance

This paper gives us a good idea of the different people and factors involved in the cryptocurrency market, which is what we plan on modeling in our project. The Uniswap market is a market that deals with a decentralized way of trading cryptocurrency. And this paper outlines three types of agents that are involved in transactions between the Uniswap market and reference market. So, we will definitely incorporate some (if not all) of these agents in our model, especially because our research shows that the outlined agents are the standard across various other similar models and can be followed. We might, however, adapt different equations and methods to model the behavior of each agent based on further research [2].

## III. Uniswap v3 Whitepaper

### 3.1 Summary

Uniswap uses an automated market marker to run a decentralized exchange. Instead of trading with other people, users trade with or provide liquidity to a contract using a formula. The unique feature of uniswap is concentrated liquidity. Virtual reserves follow the constant product rule, but only need enough real reserves to support trading in its price range. When making a transaction with the pool, users pay a flat gas fee to the network, and a % fee to the liquidity pool which is higher for more volatile pairs. Earned fees are not reinvested and stored separately until collected. The price range is between 2 ticks, so that liquidity only changes when price crosses a tick. More ticks allow for finer control of ranges but use more gas for a transaction. Each tick only needs to store a single number ΔL which is how much net liquidity changes when crossing [1]. Price movements in either direction can cause a loss in the value of the position relative to just holding the assets, and this is called Impertinent loss. This effect is larger for smaller liquidity ranges, and when outside the range all of the position is in the less valuable asset [6].

### 3.2 Relevance

This paper describes technical details of how uniswap functions, and suggests a few metrics to consider, which will be useful for making and interpreting the simulation. The flat gas fee discourages high frequency trading, which is a major difference from the stock market. Impertinent loss may be a potential source of instability as if the price is too volatile for providing liquidity to be profitable, then there will be less liquidity in the pool leading to increased volatility.

## IV. AgentPy: A package for agent-based modeling in Python

### 4.1 Summary

The paper describes the open-source Python library AgentPy, which allows users to develop agent-based models and the ecosystems surrounding them. Compared to proprietary software or previous open-source software in the agent modeling space, the novelty of AgentPy is that it allows its users to perform all of the necessary tasks of modeling using a single library. This includes the construction of specific agents and environments (and their corresponding behaviors and roles), the ability to run interactive and parallelized simulations, and the ability to use statistical methods for data analysis of these simulations. Because of AgentPy is targeted towards the scientific computing audience, it integrates well with other libraries used in scientific computing. The paper also provides a short text and code overview of how agents (and functions defining their behavior) can be constructed. Agents are the lowest level structures in AgentPy, and they can be placed in one or multiple environments to contextualize the various potential situations an agent may find itself in. These environments themselves have attributes that can be customized to the user's wishes. Environments containing agents are themselves encapsulated by models, which are used to run simulations of environments and collect the resulting data. Finally, experiments encapsulate models as the highest level structure, and are used to tune the hyperparameters of the models [4].

### 4.2 Relevance

AgentPy will most likely be used in our project. Since there are three major types of agents in a Uniswap market (arbitrageurs, liquidity providers, and traders) [2], specific agent behaviors could be developed for each type without difficulty. In addition, malicious agents could be developed. A variety of Uniswap markets could be simulated by building different environment topologies to test the models in. Its abilities to connect with other statistical, mathematical, and data analysis Python libraries like pandas, NumPy, and seaborn also add to the appeal of using AgentPy. The library is well-documented and has an active user community in resolving issues, in case we encounter any when working on the project. The data visualization capabilities of AgentPy through interactive simulations (which allow users to modify parameters of models and see how agents respond in real-time) could be used to demo our project to a larger audience [4].

## Conceptual Model

The system we are going to model in our project is the Cryptocurrency market, where time is discrete. There are multiple agents involved in this system, trading with the uniswap protocol. The major ones are liquidity providers, traders, and arbitrageurs. For the time being, we plan on assuming that there is no noise in the system. Liquidity providers can be initial or rational. The former charge a set amount for a certain amount that they give to the reserves. The latter on the other hand strive to balance out their portfolio by using optimization techniques while making decisions about what to buy and sell. There are other agents who can be classified as traders if they make transactions of one cryptocurrency for a different cryptocurrency. When there is a reference market, traders engage in the Uniswap market only if the Uniswap market profit is similar to an equivalent transaction in the reference market. Lastly, arbitrageurs trade between discrepancies in different exchanges. Our simulation will show the interactions between these types of agents in the Uniswap market given that time is a discrete entity (just like the real system). While the system is technically discrete space using fixed point numbers, it will be conceptually simpler to think of as continuous state and using floating point numbers.

## Simulation

Our team developed a simulation of the Uniswap exchange using the AgentPy package, as described in section IV of the literature review. The simulation is agent-based; this methodology involves defining the different types of agents within the system and determining what each of their behaviors should be at every time step. There are three main components that work together to produce the simulation: the agents, the model, and the exchange. Our simulation uses four different types of agents, one model, and two exchanges. Each of these components will be detailed below.

### Agent 1: Arbitrage Agent

The arbitrage agent trades between the 2 pools whenever there is a discrepancy large enough to make a profit in both currencies. This process keeps the prices of the 2 pools similar to each other. It decides on a midpoint price weighted by each pools liquidity, since larger liquidity means a slower change in price, and then swaps with both pools towards this price making sure not to put more into one swap than it gets out from the other swap. In the real world this process is implemented with flash loans which are large loans returned in the same transaction they are made in. This lets an arbitrager borrow the large initial quantity needed while keeping most of the profits and unless there is a profit at the end none of the transaction occurs [7]. In our simulation we gave the arbitrager a higher starting amount of currency than other agents.

### Agents 2: Initial Liquidity Agent

We implemented two types of liquidity agents in our model: initial and rational. The initial liquidity provider is responsible for starting the simulation by putting a certain number of coins of both types into the reserve. The initial liquidity provider invests uniformly across both pools and price ranges and collects their fees once every 20 timesteps. The investments are broken up into uniform ranges for measuring the fees earned at different prices and creating a histogram animation from this data.

### Agents 3: Rational Liquidity Agent

There are 5 rational liquidity agents while only 1 initial liquidity agent so most of the liquidity is managed by rational agents. The rational liquidity provider makes 10 positions around the current price. Initially it invests favoring the investment goes to the high fee pool. Later it allocates between the pools based on the ratio of fees that it earns from each pool to use the more beneficial pool. The ranges are randomly assigned around the current price of the chosen pool, and every 5 steps the worst performing position is replaced with a new random position. This should result in good investments sticking around and bad ones eventually being replaced with good ones.

## Agent 4: Trader Agent

The trader agent randomly decides which currency it wants to swap for and what portion of the currency it has to swap. It then checks the average price of the swap on the low fee pool and the high fee pool and executes the swap on the better pool. While the fees do make the high fee pool more expensive to use there are other things that impact this average price such as a difference in the current price and slippage which is the price changing during a swap. Slippage becomes a larger effect for larger trades and lower liquidity, so it is expected that large trades may be better on the high fee pool. In the Uniswap simulation implementation from our literature review, there was a reference market, which has been replaced by the two Unisim pools used in this project's simulation.

## Model: UniModel

The UniModel is the component that facilitates the flow of the simulation. The model consists of four different methods: setup, step, update, and end. Before the simulation begins, the user inputs a set of parameters that will be passed to the model. The model is then instantiated with this set of constant parameters that can be accessed by both the model itself as well as all of the agents in the simulation. After the model is instantiated, the setup method is called; this method is responsible for instantiating all of the agents and initializing variables. The parameters of the model are used to determine how many agents should be created and how they should be set up. After the setup method finishes execution, the update method is called. This method is intended to be performed in between each time step in order to ensure that the dynamic state of the simulation is updated and recorded. For our purposes, the update method tracks all of the information that we use to generate an animation after the simulation ends. The crux of the simulation lies in the step method. This method is responsible for advancing the state of the simulation at each time step by making calls to every agent in the simulation to perform their actions. After the step method runs, execution returns to the update method before the simulation checks if it has reached the final timestep. Once the final timestep is reached, the simulation will call the end method after it finishes updating. This method completes the run of the simulation by making a "last call" to the liquidity agents and creating reports about the final state of the system.

## Exchanges: UnisimHigh and UnisimLow

The simulation uses two different Uniswap exchanges: one with a high fee 0.3% (UnisimHigh) and one with a low fee 0.05% (UnisimLow). These numbers were chosen to match the fees for the real exchange with the Eth - USDC pair where there is a high fee pool with low volume and high liquidity and a low fee pool with high volume and low liquidity.

Agents can interact with this exchange using the methods swap and modify position. Since the main difference between creating, collecting from and burning a position is the handling of currency on the blockchain and not the internal function, all of these are wrapped into modify position which is analogous to an internal method in the real system.

These methods include a read only mode which returns the expected outcome without changing the state of the system for agents to use in planning. One major difference between this simulation and the real one is that there is a delay in confirming the state of the blockchain and transactions are prioritized by gas price so the expected outcome may not correspond to the actual outcome causing the transaction to either be incomplete or fail. We did not implement a flat gas price fee and scheduling actions prioritized by it.

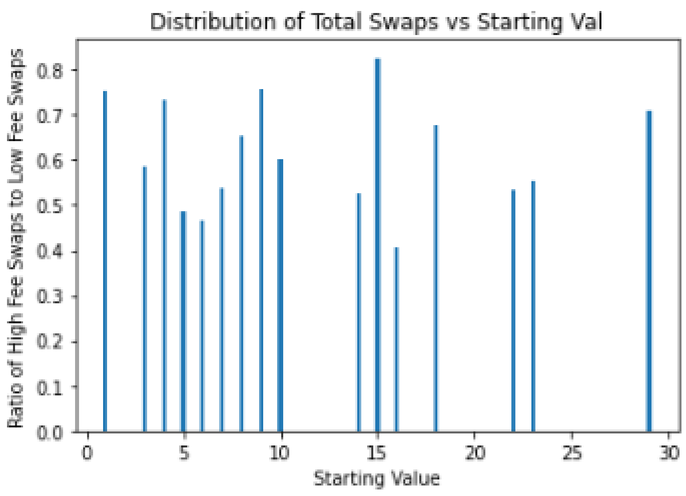
## Experimental Results and Validation

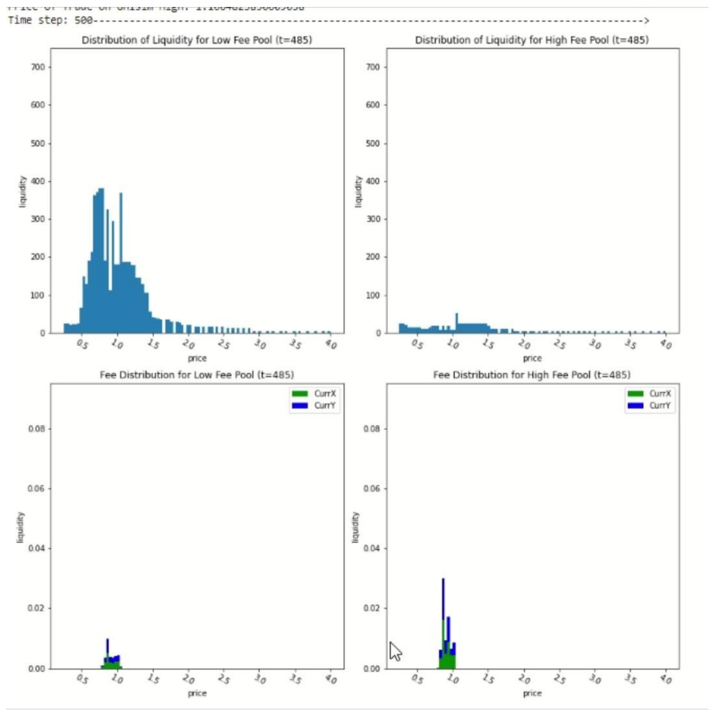
The experiment we ran was 1 Arbitrager with 20 of each currency, 1 Initial Liquidity Provider with 10 of each currency, 5 Rational Liquidity Providers with 10 of each currency, and 10 traders with a random amount of each currency from 1 to 10. The High Fee pool has a fee of 6 times larger than the Low Fee Pool, so it is expected that the rational liquidity providers will favor the High Fee Pool. Since slippage is a larger impact for large trades we expect the traders with more currency to favor the high fee pool and traders with less currency to favor the low fee pool.

Since the fee structure is the same as the real Eth-USDC pool, we can compare our results for the reserves and volume between our simulated pools and the real pools. The high fee Eth-USDC pool has a total reserves of 391 million, and a 7D volume of 343 million. The low fee Eth USDC pool has a total reserves of 271 million and a 7D volume of 3.7 billion. The reserves in the low fee pool are 31% less, while the volume is over 10 times larger.

In our simulation the low fee pool dominated and had much more liquidity and volume. Volume over 500 steps. High Fee: 4174. Low Fee: 14893
Total Reserves Value at end of simulation. High Fee: 10.36. Low Fee: 86.585

This graph was created by combining data from multiple simulation runs. It appears there is no difference based on the value the agents start with. Regardless of size agents make about 0.6 swaps on the high fee pool for every 1 swap on the low fee pool.




```
Time step: 500---------------------------------------------------------------------------------->
```

Left side is Low Fee Pool Right Side is High Fee Pool.
Top graphs are distributions of liquidity, lower graphs are the fees collected per unit of liquidity.

## Discussion/Conclusions

Whether one pool or the other dominates or we see both pools being used as in the real world example likely depends on the ratio of assets the traders and liquidity providers have, and the addition of the flat gas fee may also play a role. We did not have time to test a wide range of these values, but it would have been interesting to find a point where this behavior switches.

In our simulation the low fee pool tended to dominate after a short time, so most of the liquidity and swaps were in this pool.

There was a bit over a 3x difference in volume, but since fees are a 6x difference in rates, the high fee pool still collected more fees. The difference in reserve value is almost 9x.

## References

1. Adams, H., Zinsmeister, N., Salem, M., Keefer, R., & Robinson, D. (2021, March). *Uniswap v3 Core*. Uniswap. Retrieved March 12, 2022, from https://uniswap.org/whitepaper-v3.pdf
2. Angeris, G., Kao, H. T., Chiang, R., Noyes, C., & Chitra, T. (2020). An Analysis of Uniswap markets. *Cryptoeconomic Systems*. https://doi.org/10.21428/58320208.c9738e64
3. Berardi, M. (2011). Fundamentalists vs. chartists: Learning and predictor choice dynamics. *Journal of Economic Dynamics and Control*, *35*(5), 776–792. https://doi.org/10.1016/j.jedc.2011.01.010
4. Foramitti, J. (2021). AgentPy: A package for agent-based modeling in Python. *Journal of Open Source Software*, *6*(62), 3065. https://doi.org/10.21105/joss.03065
5. Kamyab Hessary, Y., & Hadzikadic, M. (2016). An Agent-Based Simulation of Stock Market to Analyze the Influence of Trader Characteristics on Financial Market Phenomena. *Computational Social Science*. http://computationalsocialscience.org/wp-content/uploads/2016/11/CSSSA_2016_paper_25.pdf
6. Rumiano, D. (2022, January 6). *Impermanent Loss in Uniswap V3 - Damien Rumiano*. Medium. Retrieved March 12, 2022, from https://crypto-defiworld.medium.com/impermanent-loss-in-uniswap-v3-222d01668d6e
7. I. (2021, September 2). *Build a Flash Loan Arbitrage Bot on Infura, Part I*. Infura Blog | Tutorials, Case Studies, News, Feature Announcements. Retrieved April 29, 2022, from https://blog.infura.io/build-a-flash-loan-arbitrage-bot-on-infura-part-i/