A Framework for Evaluating Price Slippage

Charles Acevedo

Spring Quarter 2022

June 12, 2022

Abstract. Blockchain technology has accelerated innovation across industries and has been implemented in various use cases. This technology provides a secure, trustless, and transparent platform for several activities, including transactions, storing information, and establishing digital ownership. One of the largest implementations is through decentralized financial applications (DeFi). Within DeFi users can engage in a variety of financial activities like taking loans and investing in prediction markets. Embracing blockchain's ethos of eliminating intermediary parties, an Automated Market Maker (AMM) is a mechanism that enables trustless, peer-to-peer (P2P) exchanges to take place. Little research has been done regarding exchange dynamics and price slippage within liquidity pools. This paper presents a conceptual model to evaluate the relationship between exchange rates, asset price, and price slippage; the Exchange Rate, Pricing, and Price Slippage Model (ERPPS Model). I further explore the various pricing algorithms used in AMM implementations and their effects on price slippage and impermanent loss. The paper concludes by evaluating the dynamic tensions present within the ERPPS model.

1 Introduction

Distributed ledger technology enables a network architecture that records information of the same system state across multiple nodes or client computers. Distributed record keeping is one of blockchain's core principles enabling its network security. This schema is a form of DLT but is a very inefficient method of record keeping. A blockchain network utilizes valuable computer resources to record and verify these states across the nodes in its network. This allocation of resources is vital for the proper functioning of a blockchain network. Since the introduction of the current understanding of Blockchain in the 2008 Bitcoin white paper (Nakamoto, 2008), there has been an influx of innovation in fields ranging from supply chain management to healthcare. Decentralized Finance (DeFi) in particular, has seen a concentration of activity in investment and pilots of upcoming financial applications on blockchain networks. Innovation was needed to propose an approach to enable interactions free of financial intermediaries. Automated Market Makers (AMMs) are now a widely implemented solution that enables peer-to-peer (P2P) transactions across DeFi applications.

Within the current literature, there has been much focus on developing economic models that capture the behavior of agents engaging with liquidity pools and AMMs. These studies make valuable contributions by creating frameworks to build intuition enabling understanding of these dynamic systems. Outside of the development of these models, research has been lacking concerning the empirical testing exploring the various dynamics present within Automated Market Makers. In this study, the focus will be on the empirical comparison of the newly proposed invariant, Nested Automated Market Makers, and the Constant Product Market Maker – this will be elaborated upon further in the following discussion.

An automated market maker uses blockchain technology to execute trading orders based on an algorithmic function, often referred to as the invariant, autonomously. My study is structured as a simulation based on commonly used AMM invariants for Decentralized Finance applications (DeFi). The Constant Product Market Maker (CPMM) is the most widely implemented invariant (utilized by decentralized exchanges such as Uniswap V2, Curve, etc.), and will serve as the basis of my comparison. A dynamic, theoretical model is developed by exploring the mechanics of a newly proposed algorithmic function, the Nested Automated Market Maker (NAMM). By using these functions as a framework, the analysis harnesses a simulation that generates data associated with each, respectively. We can evaluate the variations in behavior among the different invariants by generating measures of interest, such as price slippage levels. These measures provide a clearer understanding of this neglected field. The research questions guiding this paper are as follows: What can be done to improve price slippage within Automated Market Makers (AMMs), and how do assets, prices, and price slippage interact within trading pools? The content in this paper will begin by evaluating the current state of literature regarding Automated Market Makers, followed by presentation of methods used in the empirical approach. After this, the significance of the results will be discussed within their context within the current literature and followed by any concluding remarks.

2 Institutions

2.1 Blockchain Fundamentals

Blockchain is a technology that has far-reaching implications and uses cases, ranging from supply chain management, peer-to-peer collaboration, and decentralized finance. Its first modern implementation is attributed to Satoshi Nakamoto, a pseudo anonymous entity that is credited with the creation of Bitcoin. Prior to the Bitcoin whitepaper, distributed database technologies existed but lacked the consensus infrastructure necessary to create a trustless value-transfer system. In the proposal of his peer-to-peer electronic cash system, Nakamoto proposed a way to align actor behavior to encourage honest actions that are conducive to the proper functioning of the network, all without dependence on a trusted third-party (Nakamoto, 2008).

Blockchain provides alternative monetary policy for individuals who reside in countries with untrustworthy institutions (Cantalini and Gans, 2016), and serves as a ledger tracking transactions between agents of the economy without the need for a central institution to coordinate exchanges (Zhang et al., 2020). This peer-to-peer (P2P) network relies on trust through past transactions that are validated by consensus mechanisms.

A consensus mechanism is a method in which a system verifies the validity of a transaction and preserves the integrity of the network, commonly implemented as Proof-of-Work structure. Once this was proposed, it finally made the concept of digital currency feasible and has resulted in innovation seen in the blockchain ecosystem over the past 14 years. The distributed ledger technology (DLT) has many benefits, all stemming from its decentralized characteristics. This includes the elimination of a single point of failure that is typical in a centralized system. The absence of intermediaries solves two issues; namely, it enables a trustless system, where users must trust/rely on the underlying mathematical techniques that secure the system. The elimination of third parties increases user security by giving them control over their personal information (Nofer et al., 2017).

During development, blockchains face a tradeoff between three tensions. To establish functional trust among users of the system, they aim to balance decentralization, consensus, and scalability. Each of these elements present unique challenges; decentralization struggles to establish consensus mechanisms that maintain decentralized architecture of system, consensus is difficult to establish across distributed system and balances sacrificing processing efficiency and decentralization and adjusting block size impacts scalability but is very difficult to do so at the base protocol—innovations at layer two offer a solution (Chen et al., 2021). Research done in this space has focused on developing robust models that adequately explain market equilibria and asset valuation. There are other smaller research streams that connect economic theory with the "economic and technical fundamentals of blockchain technology" (Ante, 2020).

Blockchain technology aims to optimize interactions between network users by expediting processes and eliminating third-party transaction fees. A blockchain's ethos can be summarized as removing third parties and building a trustless network for peer-to-peer (P2P) interactions (Satoshi, 2008). Several blockchain networks exist in the ecosystem, with a wide range of functions; from the broadest to the finest applications. Ethereum has the largest ecosystem in terms of both number of layer two (L2) networks and variety of use cases. Since Ethereum is based on a Turing complete programming language, it can theoretically solve any problem. As a result of this space's rapid innovation, the Decentralized Finance sector (DeFi) has developed into one of the largest sectors in the blockchain industry. DeFi is a collection of decentralized applications that provide an alternative to traditional financial services. These platforms offer services like loans, crypto asset exchanges, and predictive markets (Jensen, et al., 2021). In the absence of financial institutions, Decentralized Finance faced the unique challenge of facilitating traditional financial activities, which require liquidity on hand. A price discovery solution was required to match counterparties who were interested in buying or selling assets at a given price.

3 Literature Review

3.1 Automated Market Makers

Through the innovation of Decentralized Finance, a mechanism was developed for price discovery in the absence of financial intermediaries. Price discovery is crucial for the smooth operation of financial markets and must meet two requirements. The first requirement is that there be deep liquidity, and the second requirement is that traders be matched with counterparties who are willing to buy or sell at a price that is mutually agreed upon. To facilitate decentralized trades, automated market makers introduce an additional actor into the equation - a liquidity provider. By pooling liquidity, trades can be executed in a timely and efficient manner (Pourpouneh, et al., 2020). In AMMs, market participants trade against a liquidity pool. Liquidity is crucial to the proper operation of Automated Market Makers; however, large trades can cause asset volatility if there is not an adequate amount of assets available in the pool. When executed in a volatile environment, swaps may cause significant price fluctuations while an order is pending execution. This phenomenon is known as price slippage and is one of the two main issues that commonly arise within AMMs, along with impermanent loss (Jensen, et al., 2021). In a liquidity pool, prices and exchange rates are determined by the algorithmic function of the AMM. The dynamics of supply and demand determine the exchange rates and prices of assets within the pool. As a result, AMMs operate independently of centralized crypto exchanges. Due to the isolation of markets within AMMs from the real value of assets on central exchanges, price discrepancies may arise. When this gap becomes a significant one, negative externalities arise, namely impermanent losses for liquidity providers.

3.1.1 Impermanent Loss

AMM users can be negatively affected by price slippage and impermanent loss, as mentioned above. Due to the current landscape of invariants used, both are systematic occurrences. Regarding the latter, an impermanent loss occurs when an asset's relative value within the liquidity pool shifts away from its trading value on other exchanges. If asset *a* has an influx of supply, its price, in terms of asset b, within the liquidity pool will decrease (due to the algorithmic function maintaining a constant ratio, *k*). This price reduction will signal informed traders of an arbitrage opportunity (Aoyagi, 2021). They see that asset *a* is trading at a discounted price that is below its market value within the liquidity pool. Trades are executed by swapping asset *b* for the discounted asset *a,* which can then be sold for an arbitrage profit on the open market. In a liquidity pool, arbitrageurs stabilize prices by reigning in asset prices to reflect their true market value. While this benefits those profiting, liquidity providers (LPs) will suffer from the opportunity cost of providing liquidity. If they had not entered the pool and continued to independently hold their assets LPs would have been free to take advantage of the arbitrage opportunity as the market value fluctuated, versus taking a loss on the asset whose value decreased relative to its market trading price. Theoretically, the prices of the assets within the pool could return to their initial values as the prices are shifted by the arbitrage trades; hence, the term "impermanent loss." This is a serious concern when considering when to enter and exit a pool as an LP, as the loss's permanence is dependent on this timing (Bartoletti, et al., 2021). Furthermore, this can cause an unexperienced "noise" trader to experience substantial, real losses and can serve as a deterrent for future public participation in the ecosystem. While impermanent loss is often a product of external market behavior, price slippage is much more dependent on the invariant chosen for the liquidity pool. Further discussion will be focused on comparing the behavior of price slippage between CPMMs and NAMMs.

3.2 Literature Results

The remainder of this paper will be aimed at addressing gaps seen in current literature. As discussed above, much focus is given on the conceptual framing of AMMs without empirically testing the underlying dynamics. This study frames quantifying AMMs' measures of interest as the main interest and then presents a model capturing the empirical behavior exhibited in simulations. Much research has been done regarding the creation of standardized trade commands for liquidity pools, as seen in papers by Aoyagi and Bartoletti. In these studies, the development of theoretical models that adequately explain and capture behavior within liquidity pools have been the focus. These are firmly rooted in economic theory, such as constant elasticity of substitution curves (Perroni and Rutherford, 1995), and attempt to apply these frameworks to explain a narrow scope – seen in Aoyagi's development of an economic model for liquidity providers.

In contrast to the existing literature, this study will be solely focused on empirically exploring behavior seen in data. To address the unexplored area of the interaction between assets, asset price, and price slippage in liquidity pools, this paper examines the data associated with these measures. After investigating this data, relationships become more apparent and serve as a valuable tool for understanding trading within liquidity pools.

4 Empirical Framework

4.1 Constant Function Market Makers and Price Slippage

As mentioned in the previous discussion, Automated Market makers utilize an algorithmic function that enables actors to execute trades at a given price while simultaneously maintaining a constant ratio of assets within the pool. Constant Product Market Makers accomplish this by holding the geometric mean of the pool's assets (or the Liquidity Pool's value) constant when executing trades. Consider a liquidity pool with two token assets, *a* and *b,* their balances, given by B_a and B_b., and the total value, as *k*. The total value of the liquidity pool is given by:

$$(1)\ k = B_a * B_b$$

Now consider a trader participating in the pool as wanted to exchange Δ_a of token *a* for Δ_b of token *b* (buying token *b* with token *a*). According to this pool's invariant function, it will return an amount of token *b* to maintain the balance of equation 1, or *k* as constant. The dynamics of this trade in the CPMM would be as follows:

$$(2)\ k = (B_a + \Delta_a) * (B_b - \Delta_b)$$

Say a trader would like to execute trades for low quantities of assets within the pool. Price slippage is not a significant factor, regardless of the pool's invariant. However, if one executes a large quantity of one asset for another, the asset's price, or exchange rate, at which the trade was expected to be executed, may fluctuate significantly. A detailed example will follow while discussing the CPMM's price slippage. This common invariant functions as intended, but the interoperabity, or ease of substitution, of the pool's assets is not considered in the design. Our discussion now turns toward the Nested Automated Market Maker.

4.1.1 NAMMs and Elasticity of Substitution

The design of the nested AMM endogenizes the substitutability of assets through the inclusion of elasticity of substitution parameters. This smooths price slippage and allows for trades between disjoint assets. However, executing trades between such an asset pair under a CPMM specification would result in extreme price slippage and impermanent loss. Consider a liquidity pool with three token assets: token *a*, token *b*, and token *y*. Put into an equation:

$$(3)\ k = \left((a^{1-\sigma} + b^{1-\sigma})^{(1/1-\sigma)}\right)^{1-\eta} + y^{1-\eta}\text{ ; where } \sigma < \eta$$

In this AMM specification, assets *a* and *b* are highly substitutable assets and have highly correlated price paths on the open market (Zhang, 2021). Asset *y* does not serve as a reliable substitute of the nested assets, *a* and *b.* Like other Constant Product Market Maker invariants, *k* represents the constant ratio the algorithmic function maintains through process of adding and returning assets during the execution of trades. The liquidity pool's trading assets are *a, b,* and *y*. Nested asset trades result in relatively low price-slippage, while the pool allows higher price slippage for swaps with assets outside of the local nest.

Unchecked price slippage is controlled by the inclusion of the parameters σ and η. These represent the substitution elasticity parameters. Conceptually, this is how easily one asset can be traded with another. These parameters are characterized as a proportion between 0 and 1, where a value closer to 0 denotes higher substitutability and a value closer to 1 indicates lower substitutability. The calculation of these parameters is beyond the scope of this paper but can be visualized as the ratio of paired assets over the ratio which one asset can be substituted for the other, holding *k* constant (Perroni and Rutherford, 1995). These parameters simultaneously reduce price slippage for trades between uncorrelated assets – trades that would otherwise result in relatively high price slippage – and drastically reduce and control price slippage for trades executed within the nest. The NAMM invariant is an alternative that proposes grouping assets based on their substitutability to minimize price slippage and impermanent loss in liquidity pools. Implementing NAMMs could serve to create liquidity pools in which highly correlated assets, such as those with low price volatility, to be traded within a nest and include less correlated assets as an unnested trading pair. To facilitate an empirical comparison, we now move the discussion towards the procedure used to evaluate the levels of price slippage for trades of the CPMM and NAMM.

4.2 Simulation Methodology

To compare the dynamics of trades between the NAMM and CPMM, various measures pertinent to the discussion of AMMs were simulated with Python. The commands used were based on concepts presented in literature regarding AMMs and Constant Elasticity of Substitution curves. Using Python, the constant return functions of the CPMM and NAMM were coded into the model along with its dependent functions. As an integral part of the NAMM, σ and η are taken as exogenous parameters in the simulation. They take on the values of 0.2 and 0.8, respectively. This indicates assets within the nest have highly correlated behavior, and that assets within and outside of the nest are not as interchangeable. Other measures are calculated within the model and are assigned corresponding functions. These include the quantity returned during swaps, the price/exchange rate a given trade would execute at, and the price slippage of the specified trade quantity.

Price slippage is defined as the proportion of the executed trade price over the price you were expecting. To illustrate, say a user is placing an order for ten units of asset *b*. The current price of asset *b*, in terms of asset *a*, is two dollars. One would expect to swap twenty units of asset *a* and receive ten units of asset *b*. However, due to the supply and demand dynamic of a liquidity pool, when the trade executes the price of asset *b* has increased by three dollars. Instead of receiving ten units of asset *b* for twenty units of asset *a*, the liquidity pool now returns 6.66 units of asset *b*. Price slippage for this trade is 50%. In the simulation, this information is contained within the function *slippage*, seen in the appendix.

Another measure needed for understanding liquidity pool dynamics is the quantity of an asset returned during swaps. For Constant Function Market makers, this means returning an amount of the desired asset to maintain a ratio, *k*. In the simulation, this is accomplished by defining a separate function, *quantity*. This function contains the updated ratio with the influx of asset (which is higher than the previous level of *k*) along with the ratio associated with the previous level of assets. A root function is used to optimize this function and to find where the difference between the new level of assets and the old level of assets equals zero. This is done while making the return asset the objective to solve. This accomplishes the goal of returning the correct asset amount to maintain the constant ratio, *k*.

Lastly, the simulation generates the relative price of the asset being traded for, seen in the function *price*. To calculate, this takes the return result from the function *quantity* and then divides the amount of asset being swapped during the trade. For example, if one is trading *x* amount of asset A for asset Y, the return value from *quantity* is passed to *price*. *Quantity's* return value is the amount of asset Y the user will obtain for a trade of *x* asset A. The *price* function then takes this amount and divides input amount of asset A by the return amount of asset Y – thus obtaining the price of asset Y in terms of asset A. The graphical representations of these analyses are obtained through simple usage of plot commands in Python, which can be reviewed in the appendix. While formal hypothesis testing did not take place, the graphical representations clearly communicate the differing levels of price slippage across trades and functional forms. This visual presentation of data is the study's main objective. The associated code script and data is in the attached GitHub repository, included under references (Acevedo, 2022).

5 Results

5.1 Swap Dynamics

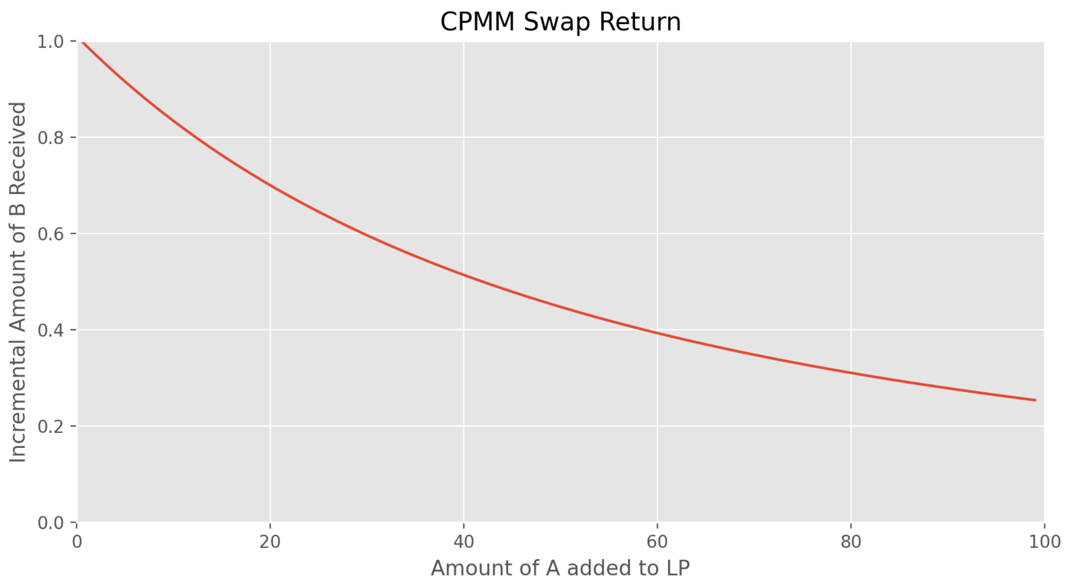
As previously discussed, Automated Market Makers that utilize a Constant Product invariant do not consider price slippage and impermanent loss that result from disparate assets. Utilizing the framework established above, comparing trading across the CPMM and NAMM gives insight on price slippage and exchange rate behavior. Recall that price slippage is dependent on the actual amount of asset returned versus the expected amount returned during a trade. This dependency can be seen in the connectedness of the *quantity, price,* and *slippage* functions in the appendix. Tracing this logic, we begin our discussion comparing the returns that the NAMM and CPMM yield. Following this, price slippage and its contributing factors across invariants will be analyzed. Finally, building on the insight of this analysis, a conceptual model for evaluating the dynamics associated with liquidity pool trades will be constructed. This model is the Exchange Rate, Price, and Price Slippage Model. Our analysis begins with the most common invariant, the Constant Product Market Maker.

Recall that the CPMM utilizes a geometric mean to maintain its constant ratio, *k*. Presenting its functional form again, we see in Equation 1 that it is a simple product.

$$(1)\ k = B_a * B_b$$

Turning towards our example of trading asset *a* for asset *b*, the return is calculated by evaluating the quantity of asset *b* necessary to maintain the constant ratio *k*. Recall that this is accomplished in software by evaluating where the difference between the updated state in period 1 and the past state in period 0 is equal to zero.

Figure 1



Notes: Incremental return of asset B for AB swaps under the CPMM invariant. The geometric mean utilized to maintain the constant ratio results in the return path for trades to exponentially decay. Data are generated within Python simulation, utilizing the *quantity* function.

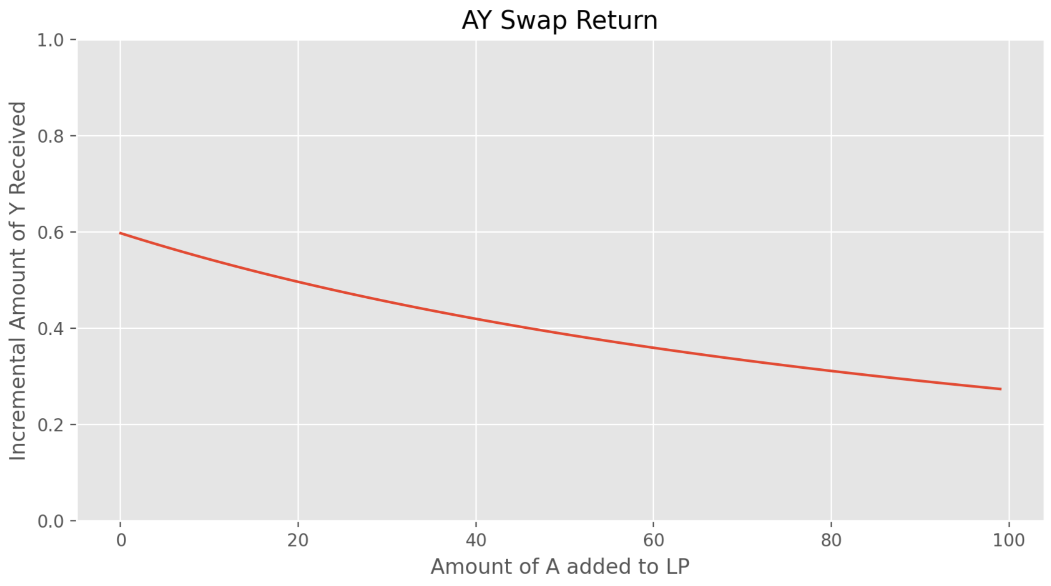
Seen in Figure 1, the incremental return of asset *b* exponentially decays as the level of asset *a* increases in the liquidity pool. Framing this through the dynamics of simple supply and demand, one can see that as asset *b* becomes scarcer compared to asset *a*, all else equal, the amount of asset *b* the same amount of asset *a* can obtain declines. Another way to understand the evolution of behavior is through the relative weights of assets within the pool. As the level of asset *b* declines while the level of asset *a* increases, the proportion of asset *a* and its influence on the constant ratio, *k*, increases. As a result, the amount of asset *b* required to be returned to maintain the constant product declines, and the relative exchange rate of asset *a* for asset *b* depreciates along the path displayed. Since there is no inclusion of elasticity of substitution parameters, the price of *b* in terms of *a* quickly rises as trades progressively become larger. This rapid price increase should also correspond in the highest rates of price slippage – when compared with the other functions of interest – and will be evaluated following the discussion regarding the remaining invariants' exchange rates.

We now consider trades within the Nested AMM. For the following analysis, AY and BY swaps are considered equivalent to simplify discussion. Recalling the NAMM's functional form (Equation 3), one can see that these assets are passed through equivalent paths during the execution of trades outside of the nest.

$$(3)\ k = \left(\left(a^{1-\sigma} + b^{1-\sigma}\right)^{(1/1-\sigma)}\right)^{1-\eta} + y^{1-\eta}\ ;\ \text{where } \sigma < \eta$$

This invariant treats the nested assets, *a* and *b*, as a composite asset when traded with asset *y,* with their elasticity curvature governed by the parameter h. For this analysis, AY and BY trades are considered as one due to the assets exhibiting identical behavior. Swaps executed for asset *y* have two dependencies; the levels of asset *a* relative to asset *b* (via the nested $\left(\left(a^{1-s} + b^{1-s}\right)^{(1/1-s)}\right)$ and the levels of the nested assets relative to asset *y* (indicated through exponent 1 - h). Seen below in Figure 2, the trading path of asset *a* for asset *y* exhibits lower diminishing returns when compared with the CPMM.

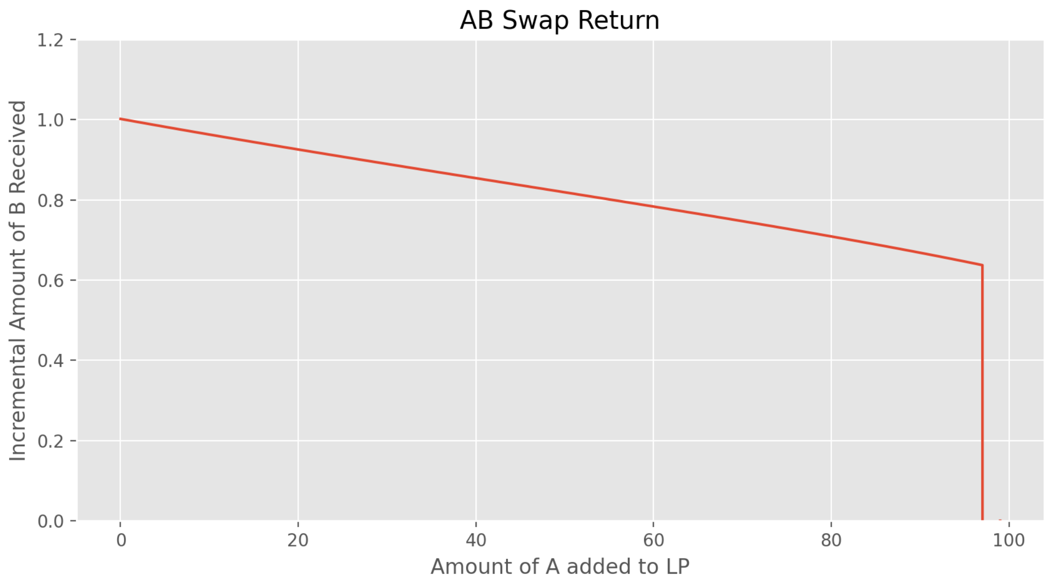
Figure 2



Notes: Incremental return of asset Y for AY trades as the quantity of A increases, under the NAMM invariant. Note the exponential decay of Y returned, largely due to the substitution parameter, h. This is the first panel (NW) in the ERPPS model, pictured below in Figure 8.

This change in behavior is due to the dynamics discussed above and is the result of the inclusion of the asset's substitution parameters – also falling in line with the hypothesis of the NAMM resulting in higher predictability for trades and price slippage. Turning towards trades between asset *a* and asset *b*, the nested assets should exhibit the greatest, sustained exchange rate. Compared with the parameter h, the nested asset's substitution parameter denotes the user's high willingness to execute trades between asset *a* and *b* and will result in the lowest levels of price slippage – discussed in following sections. In Figure 3, the AB swap path exhibits the lowest diminishing returns as the exchange rate between asset *a* and asset *b* remains relatively strong.

Figure 3



Notes: Incremental return of asset B for AB swaps. This depicts the return path for trades that take place within the nest of the NAMM. The amount of asset B being returned for A remains constant and predictably declines – due to the parameter s. The graph declines towards zero at approximately x = 96 due to the liquidity pool's reserves of asset B being depleted and is not considered in the main analysis.

This is due to the liquidity pool's path for trades executed within the nest. Seen in $(a^{1-s} + b^{1-s})^{(1/1-s)}$, the return is dependent only on the relative levels of assets within the nest and s. The substitution parameter's value as taken at 0.2, meaning these assets have highly correlated behavior and trade with low friction and slippage.

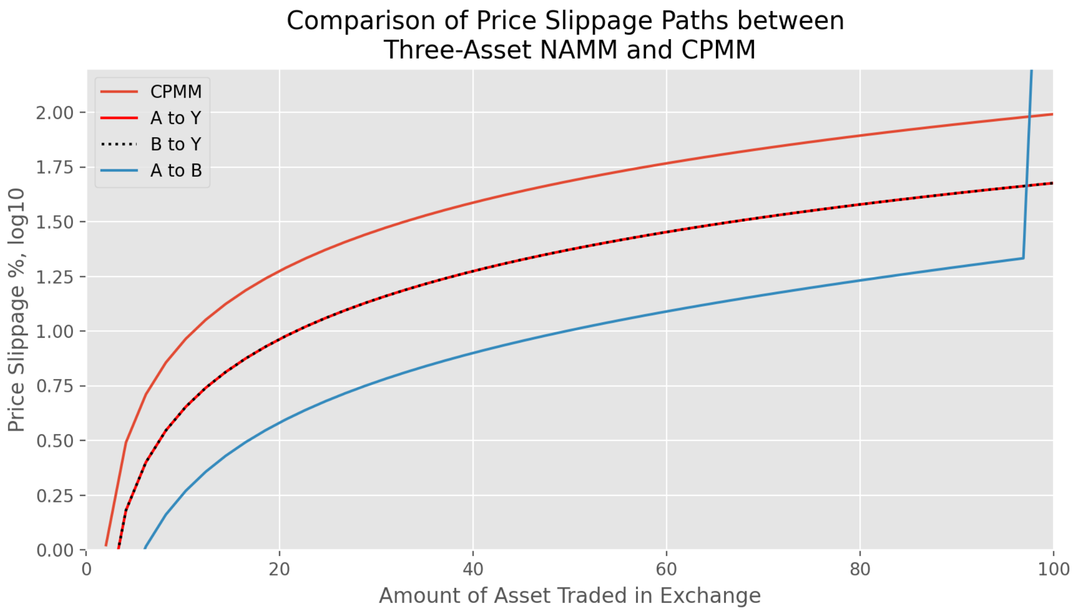
Taken together, the AMMs' three trading paths vary across the two implementations: one that does not consider the substitutability of assets and the other that controls exchange rate depreciation through elasticity parameters. The CPMM invariant is consistent with the former and the exchange rate path communicates this. Consider the initial exchange rate of 0.99 units of asset *b* per 1 unit of asset *a*. This eventually depreciates to 0.25 units of asset *b* per 1 unit of asset *a*, implying a 74% decline in value. Taking the value of h as exogenously determined, the set value of 0.8 implies that these trades are executed with relatively high price slippage. Comparing this against AY and AB swaps within the NAMM, these trading scenarios result in an exchange rate depreciation of 54% and 36%, respectively. One can see that the NAMM preserves trading performance across different asset combinations, including trades that take place between two disjoint tokens (AY swaps). Turning towards the benefits of this, recall the risk of impermanent loss carried by LP's. Preserving a more constant exchange rate results in lower fluctuations in asset prices relative to their market prices, and results in lower negative externalities imposed upon liquidity providers. As discussed previously, within liquidity pools an asset's trading price is dependent on its exchange rate against the other pooled assets. This relative price is what a user of a liquidity pool will pay once the trade executes and is a contributing factor of price slippage. Turning towards price slippage, recall the interconnectedness between these trading dynamics during the following analysis.

5.2 Price Slippage Performance

As presented above, the Constant Product Market Maker does not offer a mechanism for controlling price slippage within the liquidity pool. This becomes apparent when one considers towards the dynamics of a liquidity pool. Assets within the pool are set to maintain the constant ratio, *k*, which in turn impacts how the assets are priced against each other within the pool. Turning towards a simple supply and demand framework, it intuitively makes sense that if one asset's quantity dramatically increases during the execution of the trade, the other asset in the pool is relatively scarce. This causes the injected asset's price to decrease relative to the target asset. Once the trade executes, the diminished value of the asset causes a depreciated exchange rate, resulting in the other asset now being much more expensive than anticipated. Put succinctly, this example highlights price slippage; the trader received much less of the target asset they had hoped to acquire, while having to pay a higher price. Following a similar approach to the previous swap analysis, we will now evaluate the price slippage levels across the CPMM, AY NAMM swaps, and AB NAMM swaps.

There is an emerging pattern highlighted in the performance benchmark of liquidity pool's exchange rates. Recall the interconnectedness between the token's exchange dynamics, execution price, and price slippage within liquidity pools. Taking the CPMM's swap rate performance, as depreciation accelerates for large quantity trades, the relative price of the target asset (asset *b*) continues to increase. Therefore, increases in depreciation correspond with increases in price slippage. Under the Nested AMM's preservation of the exchange rates of its assets, price slippage performance should be better – benchmarked against the CPMM. Generating price slippage data via Python simulation yielded the following, seen below.

Figure 4



Notes: Price Slippage Paths across the invariants in the analysis, testing the paper's hypothesis. Price Slippage under the CPMM function yields the highest price slippage which is in line with predictions. Trades executed between an asset within the nest (A or B) and with assets outside (Y) produces the second highest levels of Price Slippage. The slippage path for A to Y and A to B overlap, due to the same elasticity parameter, h, between these trades. A to B trades yield the lowest price slippage levels. This results from their position within the nest and high levels of substitutability, denoted by their elasticity parameter, s. The jump seen at x = 96 of the AB path is due to asset *b*'s depletion in the pool and will not be considered in the analysis. A log transformation was implemented to better visualize the scale and behavior of price slippage.

The invariants' performance across the preservation of exchange rates directly corresponds to their relative performance in price slippage levels. Controlling price slippage results in higher liquidity pool performance for users and encourages further trades and future use. Adopting a liquidity pool invariant that accomplishes this will increase ease of use for users and ultimately lead to increased public adoption. However, this will not build a user's understanding of liquidity pool dynamics – something that will ultimately benefit an individual much more throughout their utilization of DeFi. While performing these analyses, their connectivity to one another continued to stand out, and a unifying pattern became apparent. In the following analysis, the dynamics of exchange rates, prices, and price slippage will be constructed to provide a cohesive framework to intuitively understand the effects asset fluctuations have in liquidity pools.

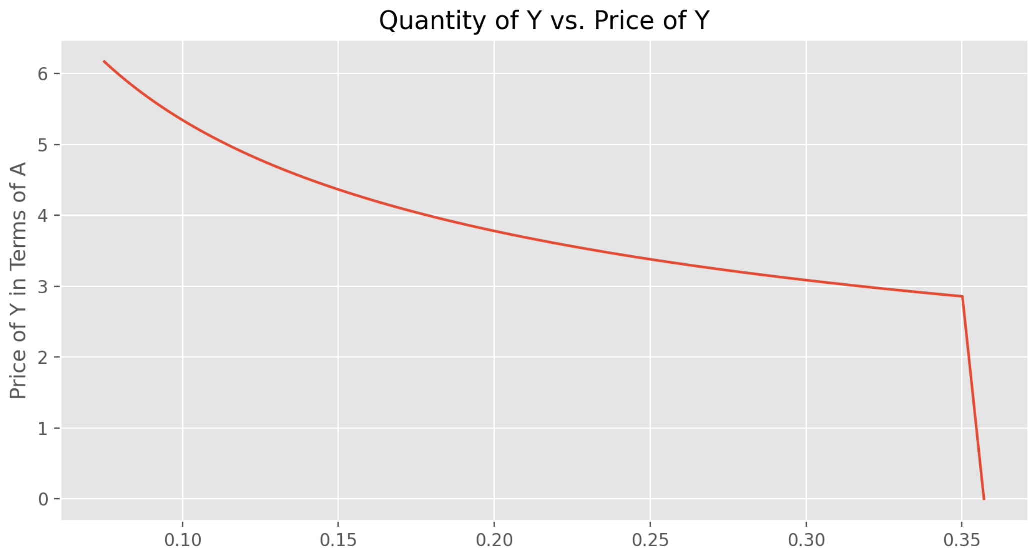
5.3 Exchange Rate, Price, and Price Slippage Model

To begin discussion, we will expand upon the exchange rate dynamics highlighted above. In the following analysis, the AY swap will be used as a case study. In Figure 2, the exchange rate path serves to act as a proxy for the supply and demand mechanics present within the liquidity pool. As the level of asset a in the liquidity pool increases via swaps for asset y, the liquidity pool's level of asset y declines as trades are executed to balance the constant ratio, k. This is illustrated below in Equation 4.

$$(1)\ k = \left(\left((a + \Delta a)^{1-\sigma} + b^{1-\sigma}\right)^{(1/1-\sigma)}\right)^{1-\eta} + (y - \Delta_y)^{1-\eta}$$

Connecting this with the behavior of asset y's price, an inverse relationship between the price of asset y and the quantity of asset y becomes apparent. Seen below in Figure 5, the negative relationship serves as a connection point for Figure 2 and Figure 5.

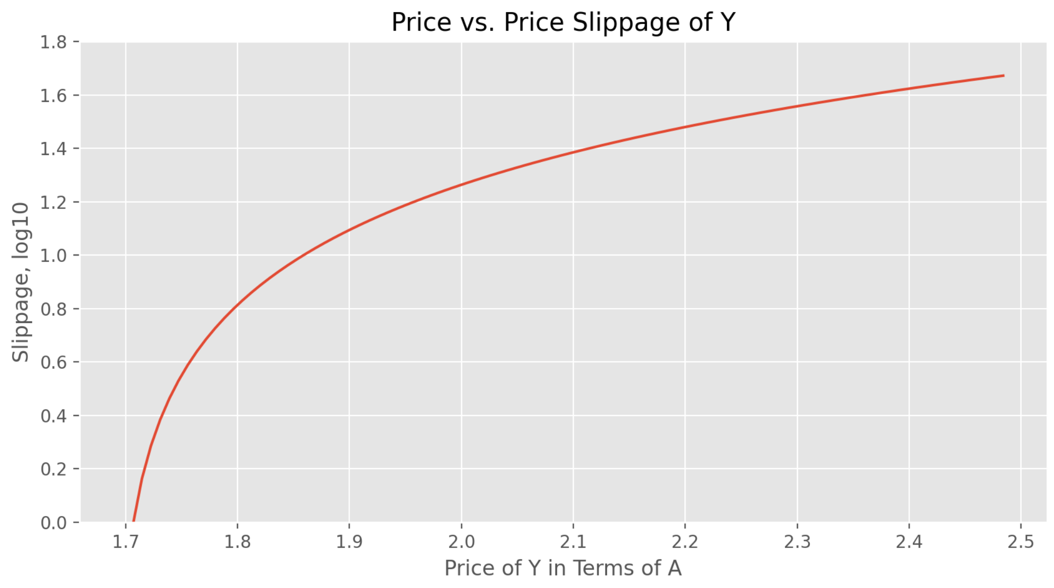
Figure 5



Notes: This path depicts the relationship between the quantity of Y being returned and the price of Y, for AY swaps in the NAMM. The exponential decay makes intuitive sense; as the quantity of Y decreases in the pool (viewing the x axis from right to left), the price of asset Y increases. A squared transformation was used to visualize the scale of the relationship. This graph is seen in the NE panel of the ERPPS model.

Between these two panels, one can understand the intuition underlying the dynamics of a liquidity pool's level of assets and a target asset's price. To build the third linkage, recall the connection established investigating the relationship between an asset's price path and price slippage. This evolves from the relationship of higher quantity trades causing an influx of one asset relative to another. This results in the desired asset's price increasing relative to the asset being injected into the pool and resulting in increased price slippage for the executed trade. This relationship is pictured below in Figure 6.

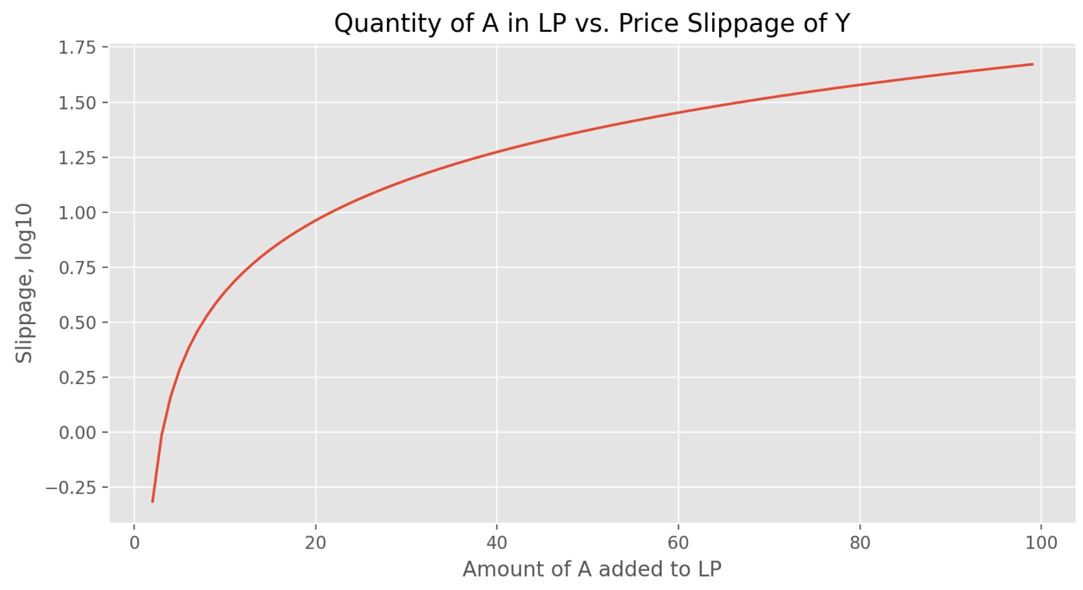
Figure 6



Notes: The graph above is the relationship of asset Y's price and price slippage. This makes up the SE panel in the ERPPS model. As done before, a log transformation was implemented on the dependent variable. This represents an important linkage illustrating as asset Y's price increases, so will the slippage in an executed trade.

The price slippage path of asset y in relation to its price shows the effect fluctuating asset levels have. Lastly, this relationship can be represented in a similar manner, but from an alternate perspective. Like the inverse relationship between the quantity of asset y and its price slippage, the level of asset a and the price slippage of asset y have a positive relationship. In Figure 7, this relationship is visualized through data.

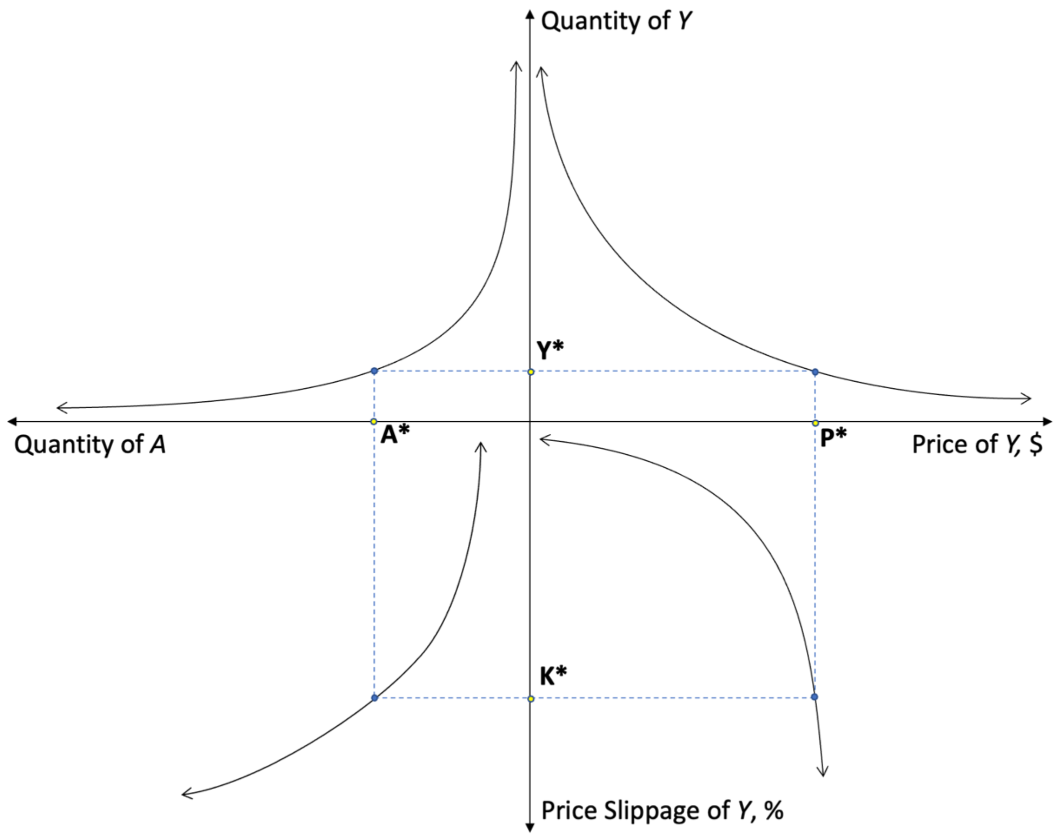
Figure 7



Notes: The graph above represents a similar relationship presented in Figure 9. Instead, it represents the positive exponential relationship between asset A and corresponding price slippage of asset Y. A log transformation is used for the dependent variable. This is the SW panel of the ERPPS model.

While exploring the behavior of data during the simulation, the relationship between the visualizations became apparent. The effect of changing the level of one asset would result in changes of the other measures of interest. An asset's exchange rate, price, and price slippage are the main metrics of interest within a liquidity pool. However, little work has been done to establish a conceptual framework for evaluating their effect on another. Doing so would offer a valuable framework for intuitively understanding effects trades would have on liquidity pools. Through generating a four-quadrant model, the dynamics are graphically captured below.

Figure 8



Notes: The above panel presents the framework of the Exchange Rate (NW), Price (NE), and Price Slippage (SE/SW) Model. The relationship between two assets in a liquidity pool is presented along with the main asset of interest's price and levels of price slippage. The ERRPS model offers an intuitive way to frame a mental model when evaluating the dynamic effect of changing an asset's level in a liquidity pool. Seen above, there are theoretical equilibrium points for the quantities of assets A and Y, the price of asset Y, and level of price slippage for asset Y. Calculating these measures are beyond the scope of this paper but can be done via the use of software.

The ERPPS Model opens the door to discover equilibria points for the levels of assets *a* and *y* and the price and price slippage of asset *y*. Viewing the model beginning in the NW quadrant, one can trace the effect changing the asset levels would have via following a clockwise path through the remaining three quadrants. Seen above, this is demonstrated with the dotted blue line establishing the equilibrium points. While establishing these levels are beyond the scope of this text, the model is a valuable tool for evaluating trades within a liquidity pool. Note the relationships present both within and between the model's quadrants.

For further illustration, we discuss movements associated with a prior example within the established framework. Recall the above example of "buying" asset y with asset a, keeping the relative asset levels within the liquidity pool constant, this would result in leftward movement along the NW quadrant's curve. This shows the levels of asset a increasing and asset y levels decreasing. This corresponds with movement along the NE quadrant's curve and a higher price for asset y. Following the path to the SE quadrant, one can see higher levels of price also correspond with higher levels of price slippage. Lastly, the SW quadrant summarizes the relationship between quantity of asset a and the price slippage for asset y. Once generalized, the ERPPS model provides a framework for the understanding of different trades within liquidity pools.

The main implications of the ERPPS model are two-fold. Firstly, by establishing a four-quadrant model that summarizes a liquidity pool's reaction to trades, a relatively inexperienced trader can quickly grasp the dynamics present and use the model to develop their own intuition. As previously discussed, negative externalities, such as volatile price slippage and impermanent loss, associated with liquidity pool pose a large risk for inexperienced traders. These occurrences often result in financial losses for these traders. By increasing the baseline understanding of participants within this DeFi application, the welfare loss can be minimized and can serve to encourage further public adoption. Secondly, the model shows that there are equilibrium levels for these measures. While finding these values is beyond the scope of this paper, the framework developed can serve as a foundational study for discovering these levels. Following the development of this methodology, the model can be utilized to find the equilibrium levels of liquidity pools. Evaluating the levels of a liquidity pool against their equilibrium levels may allow a sophisticated trader to make improved trading decisions, leading to an increase in consumer welfare.

6 Conclusion

The following are the study's main findings. To begin, the NAMM's performance, as measured by price slippage, outperforms that of the CPMM. In this paper, I simplify the NAMM's functional form for analysis. As compared to the CPMM, the NAMM endogenizes parameters that represent the elasticity of substitution for assets in the trading pool. The purpose of this is to exponentially smooth out price slippage that might otherwise be present during the execution of trades within the liquidity pool. A conceptual contribution is made by developing a model to evaluate the exchange rate of fungible crypto tokens, their prices, and price slippage; the Exchange Rate, Pricing, and Price Slippage (ERPPS) Model. Simulations revealed the dynamics of liquidity pools, which can be summarized and presented succinctly in the ERPPS model. A formal framework for evaluating these dynamics is discussed in the results section of the paper. This framework opens the door to further advancements in this field and provides a mental model for building intuition.

Blockchain technology has the potential to reshape everyday activities ranging from inventory management to gaming. Removing the private ownership of networks and data reduces the economic incentive for companies to prey on sensitive consumer information and eliminates the need for centralized governance, such as company servers. Further embracing the foundational principles of blockchain technology, Decentralized Finance aims to democratize access to secure and effective financial instruments. The automated market maker is a powerful tool that allows peer-to-peer transactions on decentralized financial applications to be carried out. The objective of this study is to empirically test the effect of fluctuating asset levels within liquidity pools on measures such as asset price and price slippage. Analyses included simulating asset trades and comparing these measures across various trading scenarios. Results supported the initial hypothesis regarding the improvement of price slippage levels under a NAMM invariant. Discussion led to the development of the ERRPS model, an effective framework that captures a liquidity pool's behavior.

This study's conclusions contribute to the existing literature by offering an analytical approach to test the dynamics of AMM trades. In addition, they offer a conceptual framework to intuitively understand the dynamics present between asset levels, prices, and price slippage. Existing literature discusses some models and functional notation for describing trades within AMMs but fall short of empirical analysis. While beyond the scope of this paper, the ERPPS model provides a framework for establishing equilibria levels. It can be further generalized to include trades of more than two assets.

Further research establishing these levels and an approach to formally test the significance of price slippage remains to be done. Negative effects of price slippage can serve as a deterrent for further public participation and adoption of this technology. It is vital to present an effective solution for minimizing the volatility of trades' execution price. The aim of this study is to provide an alternative to a commonly used invariant. This is to encourage further contemplation during the parameterization of key dynamics present during the execution of trades.

7 References

Acevedo, Charles. “AMM.” *GitHub* (2022). https://github.com/cmaceve/AMM

Aoyagi, Jun. "Liquidity provision by automated market makers." Available at SSRN 3674178 (2020).

Ante, Lennart. "A place next to Satoshi: foundations of blockchain and cryptocurrency research in business and economics." *Scientometrics* 124.2 (2020): 1305-1333.

Bartoletti, Massimo, et al. "A theory of automated market makers in defi." *International Conference on Coordination Languages and Models*. Springer, Cham, 2021.

Catalini, Christian and Joshua S Gans. “Some Simple Economics of the Blockchain” (2016).

Chen, Long, Lin William Cong, and Yizhou Xiao. "A brief introduction to blockchain economics." *Information for Efficient Decision Making: Big Data, Blockchain and Relevance (2021): 1-40.*

Jensen, Johannes Rude, et al. "An introduction to decentralized finance (defi)." *Complex Systems Informatics and Modeling Quarterly 26 (2021): 46-54.*

Liu, Ziyao, et al. “A Survey on Blockchain: A Game Theoretical Perspective.” IEEE access 7 (2019): 47615–47643.

Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." *Decentralized Business Review (2008): 21260*.

Nofer, Michael, et al. "Blockchain." *Business & Information Systems Engineering 59.3 (2017): 183-187*.

Perroni, Carlo, and Thomas F. Rutherford. "Regular flexibility of nested CES functions." *European Economic Review 39.2 (1995): 335-343.*

Pourpouneh, Mohsen, Kurt Nielsen, and Omri Ross. “Automated market makers.” No. 2020/08. IFRO Working Paper, 2020.

Zhang, Anthony. “Nested AMMs.” 2021. https://github.com/anthonyleezhang/nestedamm /blob/main/nestedamm.pdf

Zhang, Zixuan, Michael Zargham, and Victor M Preciado. “On Modeling Blockchain-Enabled Economic Networks as Stochastic Dynamical Systems.” *Applied network science 5, no. 1 (2020): 1–24.*

8 Code Appendix

```
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt
import pyplot_themes as themes
import pandas as pd
themes.theme_ggplot2(figsize [10, 5])


class data_ay():
    def __init__(self, sigma, eta, a0, b0, y0):
        self.sigma   sigma
        self.eta   eta
        self.a   a0
        self.b   b0
        self.y   y0

    def Ufun(self, a, b, y):
        x   (a**(1-self.sigma) + b**(1-self.sigma))**(1/(1-self.sigma))
        U   x**(1-self.eta) + y**(1-self.eta)
        return(U)

    def quantity(self, delA):
        UStart   self.Ufun(self.a, self.b, self.y)
        delY   opt.root(lambda delY: self.Ufun(self.a + delA, self.b, self.y - delY) -
UStart, 3.41)

        return delY

    def price(self, delA):
        delY   self.quantity(delA).x[0]
        ratio   delA / delY

        return ratio

    def slippage(self, delA):
        return (((self.price(delA) / self.price(1)) - 1) * 100)

d   data_ay(.2, .8, 100, 100, 100)
UStart_ay   d.Ufun(d.a, d.b, d.y)
UStart_ay



class data_by():
    def __init__(self, sigma, eta, a0, b0, y0):
        self.sigma   sigma
        self.eta   eta
        self.a   a0
        self.b   b0
        self.y   y0

    def Ufun(self, a, b, y):
        x   (a**(1-self.sigma) + b**(1-self.sigma))**(1/(1-self.sigma))
        U   x**(1-self.eta) + y**(1-self.eta)
        return(U)

    def quantity(self, delB):
        UStart   self.Ufun(self.a, self.b, self.y)
        delY   opt.root(lambda delY: self.Ufun(self.a, self.b + delB, self.y - delY) -
UStart, 3.41)

        return delY

    def price(self, delB):
        delY   self.quantity(delB).x[0]
        ratio   delB / delY

        return ratio

    def slippage(self, delB):
        return (((self.price(delB) / self.price(1)) - 1) * 100)

c   data_by(.2, .8, 100, 100, 100)
UStart_by   c.Ufun(c.a, c.b, c.y)



class data_ab():
    def __init__(self, sigma, eta, a0, b0, y0):
        self.sigma   sigma
        self.eta   eta
        self.a   a0
        self.b   b0
        self.y   y0

    def Ufun(self, a, b, y):
        x   (a**(1-self.sigma) + b**(1-self.sigma))**(1/(1-self.sigma))
        U   x**(1-self.eta) + y**(1-self.eta)
        return(U)

    def quantity(self, delA):
        UStart   self.Ufun(self.a, self.b, self.y)
        delB   opt.root(lambda delB: self.Ufun(self.a + delA, self.b - delB, self.y) -
UStart, 3.44)
```

```
        return delB

    def price(self, delA):
        delB   self.quantity(delA).x[0]
        ratio   delA / delB
        return ratio

    def slippage(self, delA):
        return ((self.price(delA) / self.price(1)) - 1) * 100

b   data_ab(.2, .8, 100, 100, 100)
UStart_ab   b.Ufun(b.a, b.b, b.y)
UStart_ab


class data_CPMM():
    def __init__(self, a0, b0):
        self.a   a0
        self.b   b0

    def Ufun(self, a, b):
        U   self.a * self.b
        return(U)

    def quantity(self, delA):
        UStart   self.Ufun(self.a, self.b)
        delB   opt.root(lambda delB: ((self.a + delA) * (self.b - delB)) - UStart, 1)
        return delB

    def price(self, delA):
        delB   self.quantity(delA).x[0]
        ratio   delA / delB
        return ratio

    def slippage(self, delA):
        return ((self.price(delA) / self.price(1)) - 1) * 100

cp   data_CPMM(100, 100)
UStart_cp   cp.Ufun(cp.a, cp.b)
UStart_cp



'''
Figure Section
'''
```

```
#Graph Displaying price slippage paths for NAMM swaps and CPMM swaps
outer   np.linspace(0, 101, 50)
inner   np.linspace(0, 101, 50)

discrete_out   [i for i in range(0, 101)]
discrete_in   [i for i in range(0, 101)]

ay_slippage   [d.slippage(i) for i in discrete_out]
by_slippage   [c.slippage(i) for i in discrete_out]
ab_slippage   [b.slippage(i) for i in discrete_in]
cp_slippage   [cp.slippage(i) for i in discrete_in]
ay_slippage   [d.slippage(i) for i in outer]
by_slippage   [c.slippage(i) for i in outer]
ab_slippage   [b.slippage(i) for i in inner]
cp_slippage   [cp.slippage(i) for i in inner]
plt.plot(inner, np.log10(cp_slippage), label   'CPMM')
plt.plot(outer, np.log10(ay_slippage), 'r', label   'A to Y')
plt.plot(outer, np.log10(by_slippage), 'k:', label   'B to Y')
plt.plot(inner, np.log10(ab_slippage), label   'A to B')
plt.xlim(0, 100)
plt.ylim(0, 2.2)
plt.legend()
plt.title('Comparison of Price Slippage Paths between \nThree-Asset NAMM and CPMM')
plt.xlabel('Amount of Asset Traded in Exchange')
plt.ylabel('Price Slippage %, log10')
plt.show()
plt.savefig('SlippagePaths.png')



'''
AY Swap graphs
'''
#Plot displaying incremental amount of y received for AY swaps
delA   [i for i in range(100)]
difference   [(d.quantity(i).x[0]-d.quantity(i-1).x[0]) for i in range(100)]

plt.plot(delA, difference)
plt.xlabel('Amount of A added to LP')
plt.ylabel('Incremental Amount of Y Received')
plt.ylim(0,1)
plt.title('AY Swap Return')
plt.themes
plt.show()



#plot displaying AY swaps price slippage as a function of price
```

```
slippage  np.array([d.slippage(i) for i in range(100)])
price   np.array([d.price(i) for i in range(100)])
price   np.log10(price)
slippage   np.log10(slippage)

plt.plot(delA, slippage)
plt.xlabel('Price of Y in terms of A, log10')
plt.ylabel('Slippage, log10')
plt.ylim(0, 1.75)
plt.title('AY Price Slippage vs. Quantity of A')
plt.show()



'''
AB swap graphs
'''
#AB swaps incremental amount of B received per 1 unit increase in A
difference   [(b.quantity(i).x[0]-b.quantity(i-1).x[0]) for i in range(100)]
delA   [i for i in range(100)]

plt.plot(delA, difference)
plt.ylim([0,1.2])
plt.xlabel('Amount of A added to LP')
plt.ylabel('Incremental Amount of B Received')
plt.title('AB Swap Return')
plt.show()



#plot displaying AB swaps price slippage as a function of price
slippage  np.array([b.slippage(i) for i in range(100)])
price   np.array([b.price(i) for i in range(100)])
price   np.log10(price)
slippage   np.log10(slippage)

plt.plot(delA, slippage)
plt.xlabel('Price of B in terms of A, log10')
plt.ylim(0, 2)
plt.ylabel('Slippage, log10')
plt.title('AB Price Slippage vs. Quantity of A')
plt.show()



'''
CPMM swap graphs
'''
```

```
#AB swaps incremental amount of B received per 1 unit increase in A
difference   [(cp.quantity(i).x[0]-cp.quantity(i-1).x[0]) for i in range(100)]
delA   [i for i in range(100)]

plt.plot(delA, difference)
# plt.ylim([0, 2])
plt.xlim([0,100])
plt.ylim(0,1)
plt.xlabel('Amount of A added to LP')
plt.ylabel('Incremental Amount of B Received')
plt.title('CPMM Swap Return')
plt.show()



#plot displaying AB swaps price slippage as a function of price
slippage  np.array([cp.slippage(i) for i in range(100)])
price   np.array([cp.price(i) for i in range(100)])
price   np.log10(price)
slippage   np.log10(slippage)

plt.plot(delA, slippage)
plt.xlabel('Price of B in terms of A, log10')
plt.ylabel('Slippage, log10')
plt.title('CPMM Price Slippage vs. Quantity of A')
plt.show()



'''
Conceptual Model graphs
'''



#A Swaps for Y
delA   [i for i in range(100)]
difference   [(d.quantity(i).x[0]-d.quantity(i-1).x[0]) for i in range(100)]

plt.plot(delA, difference)
plt.xlabel('Amount of A added to LP')
plt.ylabel('Incremental Amount of Y Received')
plt.title('AY Swap Return')
plt.themes
plt.show()
```

```
#Quantity of Y vs Price of Y
difference   np.array([(d.quantity(i).x[0]-d.quantity(i-1).x[0]) for i in range(100)])
price   np.array([d.price(i) for i in range(100)])
price   (price)

plt.plot(price, difference)
plt.xlabel('Incremental Amount of Y Returned in LP')
plt.ylabel('Price of Y in Terms of A')
# plt.ylim(0, 1.75)
plt.title('Quantity of Y vs. Price of Y')
plt.xticks(0,100)
plt.show()



#Quantity of A vs. price slippage
slippage  np.array([d.slippage(i) for i in range(100)])
delA   [i for i in range(100)]
slippage   np.log10(slippage)

plt.plot(delA, slippage)
plt.xlabel('Amount of A added to LP')
plt.ylabel('Slippage, log10')
plt.title('Quantity of A in LP vs. Price Slippage of Y')
plt.themes
plt.show()



#Price of y versus price slippage
price   np.array([d.price(i) for i in range(100)])
slippage  np.array([d.slippage(i) for i in range(100)])
slippage   np.log10(slippage)

plt.plot(price, slippage)
plt.xlabel('Price of Y in Terms of A')
plt.ylim(0, 1.8)
plt.ylabel('Slippage, log10')
plt.title('Price vs. Price Slippage of Y')
plt.themes
plt.show()



'''
Exporting data
'''
```

```
dict1   {
    'delA' : [i for i in range(100)], 'Quantity_Returned_Y' : [d.quantity(i).x[0] for i
in range(100)], 'Incremental_Y' : [(d.quantity(i).x[0]-d.quantity(i-1).x[0]) for i in
range(100)], 'Price_Y' : [d.price(i) for i in range(100)], 'Slippage_Y' : [d.slippage(i)
for i in range(100)], 'Quantity_Returned_B' : [b.quantity(i).x[0] for i in range(100)],
'Incremental_B'  :  [(b.quantity(i).x[0]-b.quantity(i-1).x[0])  for  i  in  range(100)],
'Price_B' : [b.price(i) for i in range(100)], 'Slippage_B' : [b.slippage(i) for i in
range(100)],  'Quantity__Returned_CPMM'  :  [cp.quantity(i).x[0]  for  i  in  range(100)],
'Incremental_CPMM' : [(cp.quantity(i).x[0]-cp.quantity(i-1).x[0]) for i in range(100)],
'Price_CPMM' : [cp.price(i) for i in range(100)], 'Slippage_CPMM' : [cp.slippage(i) for
i in range(100)]
}
print(dict1)
df   pd.DataFrame.from_dict(dict1, orient 'index')
df   df.transpose()
df.to_csv("SimulationData.csv")
```